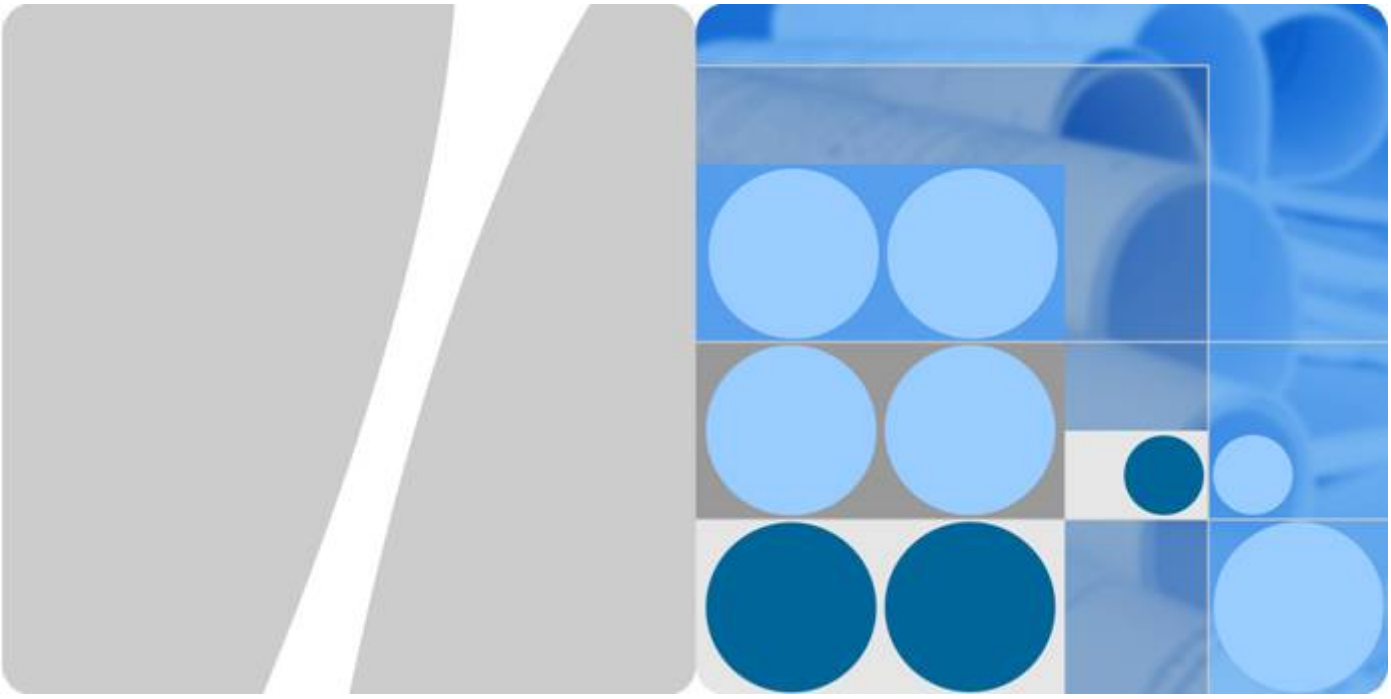


分布式数据库中间件

最佳实践

文档版本　02

发布日期　2021-03-08

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 合理制定分片策略

DDM创建逻辑表时，若逻辑表类型选择为拆分表，则需要制定分片策略，并选择拆分键。

当数据表之间存在E-R关系时，可以制定相同的分片规则，各数据表分别选择有关联关系的字段作为拆分键，这样各表中有关联关系的数据将会存储在一个分片上，避免数据跨分片JOIN操作。

如客户表、订单表与订单明细表，在创建拆分表时，建议都选取客户ID作为拆分键。

2 广播表和单表的使用场景

单表指数据只存储在其中一个默认分片上的表；广播表指在所有分片上都存储全量数据，提升JOIN效率。

单表

DDM管理控制台不提供单表创建操作，用户可以通过MySQL客户端或应用程序连接到DDM实例后自行创建。

如果一张表的数据，数据量预估在1000万以下，且没有与其他拆分表进行关联查询的需求，建议将其设置为单表类型，存储在默认分片中。

广播表

在业务数据库中，存在一些数据量不大，更新频度低，但常常需要用来做关联查询的表。

为了支持这类表与拆分表进行JOIN操作，DDM设计了一类“广播表”，具有以下特点：

- 广播表在各分片中数据一致。数据插入、更新与删除会实时在每一个分片中执行一次。
- 对广播表的查询，仅在一个分片中执行。
- 任何表都可以与广播表进行JOIN操作。

例如：

电商企业的订单管理系统，需要查询统计广东地区的订单数据。假如涉及到省份地区表与订单流水表进行JOIN查询，由于订单数据量庞大，订单流水表需要分片存储，因此可以考虑将省份地区表设计为“广播表”，避免跨库JOIN操作。

须知

广播表操作 、不带分片条件的SQL语句等全表扫描类语句，并发不要太高（或者选择在业务空闲时进行），否则可能报“后端rds连接数可能不够用”的错误。

3 如何选择事务模型

分布式数据库中间件当前支持单机、FREE最大努力提交、XA三种分布式事务模型。

详情如表3-1所示。

表 3-1 分布式事务模型特点

事务模型	优势	不足
单机	执行效率高，由底层数据库提供强一致性的保证。	如果事务涉及到多分片，DDM将拒绝执行，返回错误。
FREE最大努力提交	支持跨分片事务，执行结果相对独立，互不干涉。	执行效率低于单机。 如果有分片提交失败时，不提供事务补偿，因此会出现部分执行成功部分执行失败。
XA	保证了数据的强一致，适合对数据强一致要求很高的关键领域。	实现复杂，牺牲了可用性，对性能影响较大，不适合高并发高性能场景。

说明

- DDM的分布式事务模型建立在逻辑库层面上，一个逻辑库只能选择一种事务模型。在创建逻辑库时需要规划好存储的数据结构与SQL的需求，如是否需要跨分片，是否必须保证事务强一致性等，根据需求选择合适的分布式事务模型。
- 目前自定义hint类型只支持select和update语句。
- 在最终原子性场景下，建议不要删除表中字段或在已有字段中添加字段。若能确认当前事务未出异常或已正常回滚补偿完毕，则可以添加字段，添加时请在表字段的最后进行追加。

单机

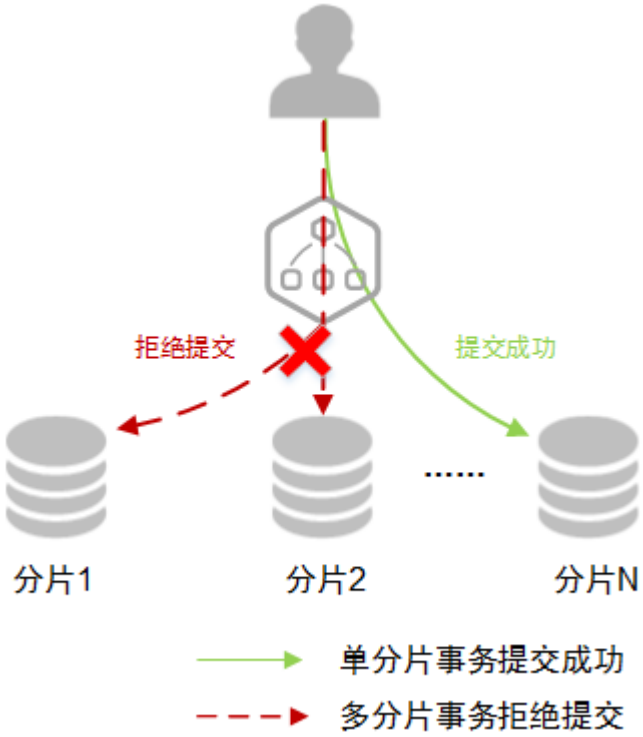
基本概述

只能在单个分片上执行，如果事务涉及到多分片，DDM将拒绝执行，返回错误。原理如图3-1所示。

适用场景

适合业务拆分比较合理，在应用层有独自完善的事务处理框架，到DDM的事务都是单分片事务，单分片事务由底层数据库提供强一致性的保证。单机事务模型下，如果出现跨分片的事务，会报错进行提示，避免达不到预期目的。

图 3-1 单机事务模型



最大努力提交

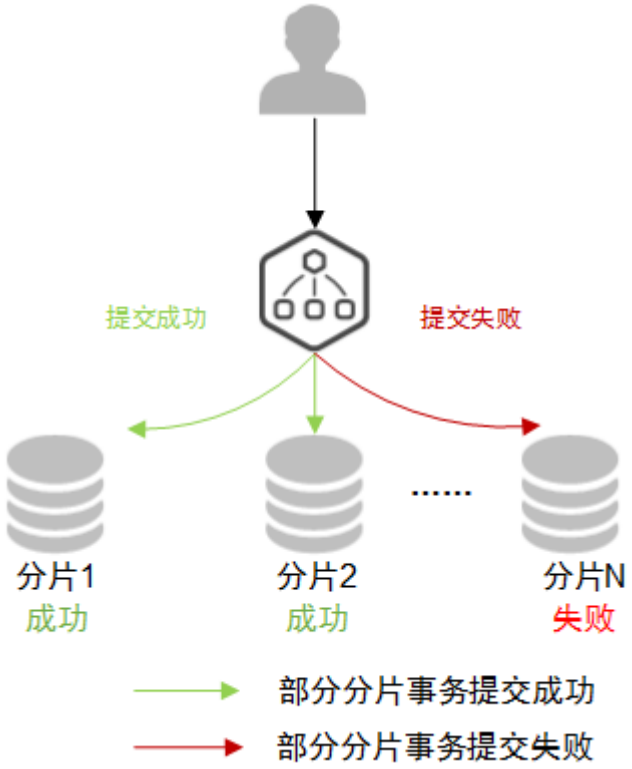
基本概述

事务在各分片上分别提交，互不干涉，提交结果尽最大可能保持一致，但有部分提交成功部分提交失败的可能，原理如图3-2所示。

适用场景

这种分布式事务适合绝大部分不涉及金钱往来的业务，在性能和一致性之间比较好的一个平衡。事务中的commit往多个节点发送执行，有部分commit成功部分commit失败的可能性，但是这种情况出现的可能性比较低，只有在commit的时间窗内出现异常才有可能出现。

图 3-2 最大努力提交事务模型



最终原子性

基本概述

事务在各分片上的提交结果不保证始终一致，如果有分片提交失败，DDM会对其他提交成功的分片提供补偿机制撤消之前的修改，从而确保各分片事务状态最终一致。

说明

最终一致性场景下，如有并发的查询请求，查询到的结果可能不是最终状态，如部分分片已完成提交，部分分片还在提交中，即存在中间状态。

适用场景

适合对一致性要求比较高的场景，最终一致性解决了最大努力提交模型下部分commit成功部分失败的问题。如果对部分读的sql一致性要求比较高，还可以通过**select for update**或者**lock in share mode**来避免读取到不一致状态（部分成功部分失败）。

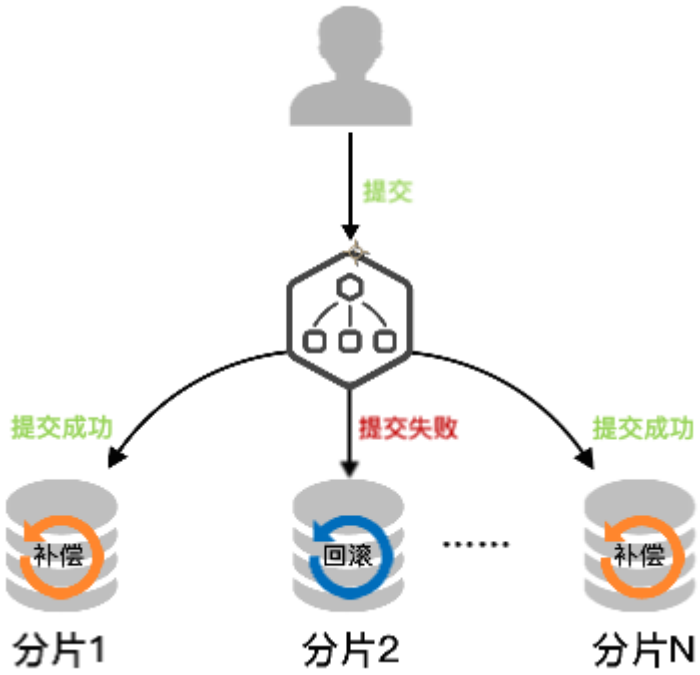
例如：

select col1, col2,...coln from table1 where col1={拆分键} for update;

select col1, col2,...coln from table1 where col1={拆分键} lock in share mode;

where条件中建议带上拆分键。

图 3-3 最终原子性事务模型



4 如何由业务侧实现全局唯一 ID

操作场景

本章节介绍如何在业务测实现全局唯一ID。

说明

假定业务表 test 的表结构如下：

```
id bigint
 order_id varchar(30)
 c1,
 c2,
 c3... 等业务字段
```

其中 id是拆分字段，希望order_id全局唯一。

操作步骤

步骤1 创建索引表 test_uk_order，结构如下：

```
order_id varchar(30)
test_id bigint
```

其中 order_id是拆分字段，也是PK。

步骤2 针对test表增删改的场景分别确认。

- insert时，先 insert test，再insert test_uk_order。如果test_uk_order表因主键冲突插入失败，则认定存在test。order_id字段存在重复值，则判定test表的insert也是失败的。
- update时，如果不涉及order_id和id字段修改，无需扩展。只要涉及order_id和id的修改，直接报错不允许修改。
- delete时，先查出order_id，删除test表，再按照order_id删除test_uk_order表。

----结束

5 如何将单 RDS 数据整库迁移至 DDM

操作场景

本章节主要介绍将单RDS（非拆分，以下简称旧RDS）库中的数据整库迁移到DDM实例非拆分库中，仅使用DDM做读写分离场景。

说明

- 迁移过程中可能会出现业务中断情况，中断时长与迁移数据量大小、网络情况相关。
- 数据迁移是一项比较复杂的操作，建议在业务量较低时进行。本实践仅供参考，您需要根据自己业务场景、数据量、停机时间要求等情况，设计合适的迁移方案。
- 对于数据量较大的场景，建议通过工单或售后服务联系DDM技术支持人员进行支撑，在正式数据迁移前进行充分的迁移演练测试。

迁移前准备

- 准备可以访问旧RDS实例、目标DDM实例和目标DDM实例关联的RDS实例的ECS。
  a. 确保旧RDS实例、目标DDM实例和目标DDM实例关联的RDS实例都在同一个VPC下，保证网络互通。
  b. 旧RDS实例、目标DDM实例和目标DDM实例关联的RDS实例的安全组建议配置相同，如果不同则需要放开对应端口访问。
  c. ECS已安装MySQL官方客户端，MySQL客户端版本建议为5.6或5.7。
     - Redhat系列Linux安装命令：**yum install mysql mysql-devel**
     - Debian系列Linux安装命令：**apt install mysql-client-5.7 mysql-client-core-5.7**
  d. ECS磁盘空间足够存放临时转储文件；ECS内存空间足够，可以用来比较转储文件。
- 准备已关联RDS实例的DDM实例，并配置DDM账号、DDM逻辑库等相关信息。
- 如果目标DDM实例逻辑库为拆分库，则需要在迁移前，在DDM控制台先创建与旧RDS数据表结构相同的逻辑表。

约束限制

- 为了保持数据完整性，需要先停止旧RDS业务后再进行数据迁移。

- 该场景不支持通过DDM关联旧RDS实例进行数据关联，需要将旧RDS实例数据导出后再导入到DDM实例完成数据迁移。
- 目标DDM关联的RDS版本与旧RDS的MySQL版本需要保持一致。

从旧 RDS 导出数据

步骤1 登录ECS。

步骤2 执行如下命令导出结构数据，其中红色参数需根据实际情况配置，详细参数说明如表5-1所示。

```
mysqldump -h {DB_ADDRESS} -P {DB_PORT} -u {DB_USER} -p --skip-lock-tables --add-locks=false --set-gtid-purged=OFF --no-data {DB_NAME} > {mysql_schema.sql}
```

表 5-1 参数解释

参数	说明	备注
DB_ADDRESS	待导出数据的数据库连接地址。	必填
DB_PORT	数据库侦听端口	必填
DB_USER	数据库用户	必填
DB_NAME	数据库名称	必填
mysql_schema.sql	生成的表结构文件名。	每次导出表结构时文件名不同。 建议以“逻辑库名”+“_”+“schema”格式命名，以免数据被覆盖。如mysql_schema.sql。
mysql_data.sql	生成的整库数据文件名。	-

步骤3 执行如下命令导出整库数据，其中红色参数需根据实际情况配置，详细参数说明如表5-1所示。

```
mysqldump -h {DB_ADDRESS} -P {DB_PORT} -u {DB_USER} -p --hex-blob --complete-insert --skip-lock-tables --skip-tz-utc --skip-add-locks --set-gtid-purged=OFF --no-create-info {DB_NAME} > {mysql_data.sql}
```

步骤4 导出完成后，在ECS上执行如下可查看步骤2和步骤3导出的.sql文件。

```
ls -l
```

----结束

将数据导入至 DDM 实例

步骤1 在ECS上执行如下命令，将结构文件导入DDM。

```
mysql -f -h {DDM_ADDRESS} -P {DDM_PORT} -u {DDM_USER} -p {DB_NAME} < {mysql_schema.sql}
Enter password: **********
```


表 5-2 参数解释

参数	说明	备注
DDM_ADDRESS	待导入数据的DDM实例连接地址。	可在DDM管理控制台上，实例基本信息页面查看“连接地址”和“端口”。
DDM_PORT	待导入数据的DDM侦听端口。	
DDM_USER	访问DDM的用户。	创建DDM逻辑库时使用的账号，需具备读写权限。
DB_NAME	待导入数据的DDM逻辑库名称。	-
mysql_schema.sql	待导入结构文件的名称。	即导出数据中**步骤2**导出的文件名称。
mysql_data.sql	待导入整库数据文件的名称。	即导出数据中**步骤3**导出的文件名称。

步骤2 执行如下命令，将数据文件导入DDM。

```
mysql -f -h {DDM_ADDRESS} -P {DDM_PORT} -u {DDM_USER} -p {DB_NAME} < {mysql_schema.sql}
Enter password: **********
```

----结束

6 如何将 Mycat 数据整库迁移至 DDM

操作场景

本章节主要介绍将Mycat中的数据整库迁移到DDM中。

说明

- 迁移过程中可能会出现业务中断情况，中断时长与迁移数据量大小、网络情况相关。
- 数据迁移是一项比较复杂的操作，建议在业务量较低时进行。本实践仅供参考，您需要根据自己业务场景、数据量、停机时间要求等情况，设计合适的迁移方案。
- 对于数据量较大的场景，您可以在管理控制台右上角，选择“**工单 > 新建工单**”，完成工单提交。工作人员会为您进行相关处理。或售后服务联系DDM技术支持人员进行支撑，在正式数据迁移前进行充分的迁移演练测试。
- 由于DDM仅支持通过弹性云服务器（ECS）访问，因此需要先将数据库导出为文件到ECS，然后从ECS将文件中的数据导入到DDM。

迁移前准备

- 准备可以访问Mycat、目标DDM实例和目标DDM实例关联的RDS实例的ECS。
  a. 确保Mycat、目标DDM实例和目标DDM实例关联的RDS实例都在同一个VPC下，保证网络互通。
  b. 部署Mycat的ECS、目标DDM实例和目标DDM实例关联的RDS实例的安全组建议配置相同，如果不同则需要放开对应端口访问。
  c. ECS已安装MySQL官方客户端，MySQL客户端版本建议为5.6或5.7。
     - Redhat系列Linux安装命令：**yum install mysql mysql-devel**
     - Debian系列Linux安装命令：**apt install mysql-client-5.7 mysql-client-core-5.7**
  d. ECS磁盘空间足够存放临时转储文件；ECS内存空间足够，可以用来比较转储文件。
- 准备已关联RDS实例的DDM实例，并配置DDM账号、DDM逻辑库等相关信息。
- 本章节以Mycat 1.6版本进行迁移为例。

迁移策略

Mycat与DDM数据表类型不同，迁移策略也有所差异，详情如**表6-1**所示。

表 6-1 迁移策略

Mycat表类型	DDM表类型	迁移策略
非拆分表	单表	1. Mycat导出**表结构**和**表数据**。 2. 连接目标DDM关联的RDS**将数据导入至目标DDM（非拆分表场景）**。
拆分表：分片规则为hash类（含年月日等日期类）	拆分表：拆分算法为hash（含日期函数）	1. Mycat导出全部数据表结构。 2. 对照导出的表结构，在DDM控制台创建表结构完全相同的表。 3. **从Mycat整库导出数据**。 4. **连接DDM导入整库数据**。
拆分表：分片规则为按范围range类（含年月日等日期类）	拆分表：拆分算法为range（含日期函数）	
广播表	广播表	

约束限制

- 为了保持数据完整性，需要先停止Mycat业务后再进行数据迁移。
- 该场景不支持通过DDM关联Mycat关联的RDS进行数据关联，需要将Mycat数据导出后再导入到DDM完成数据迁移。
- 目标DDM关联的RDS版本与Mycat关联数据库的版本需要保持一致。

从 Mycat 导出数据表结构

步骤1 登录ECS。

步骤2 执行如下命令导出Mycat数据中的表结构数据，其中斜体变量参数需根据实际情况配置，详细参数说明如**表6-2**所示。

```
mysqldump -h {DB_ADDRESS} -P {DB_PORT} -u {DB_USER} -p    --skip-lock-tables --add-locks=false --set-gtid-purged=OFF --no-data --order-by-primary {DB_NAME} > {mysql_schema.sql}
Enter password: **********
```

表 6-2 参数解释

参数	说明	备注
DB_ADDRESS	待导出数据的数据库连接地址。	必填
DB_PORT	数据库侦听端口	必填
DB_USER	数据库用户	必填
DB_NAME	数据库名称	必填

参数	说明	备注
mysql_schema.sql	生成的表结构文件名。	每次导出表结构时文件名不同。 建议以“逻辑库名”+“_”+“schema”格式命名，以免数据被覆盖。如mysql_schema.sql。
mysql_data.sql	生成的整库数据文件名。	-

----结束

从 Mycat 整库导出数据

步骤1 登录ECS。

步骤2 执行如下命令导出Mycat整库数据，其中斜体变量参数需根据实际情况配置，详细参数说明如表6-2所示。

```
mysqldump -h {DB_ADDRESS} -P {DB_PORT} -u {DB_USER} -p   --hex-blob --complete-insert --skip-lock-tables --add-locks=false --set-gtid-purged=OFF --quick --no-create-info --order-by-primary {DB_NAME} > {mysql_data.sql}
Enter password: **********
```

步骤3 导出完成后，在ECS上执行如下命令可查看导出的.sql文件。

```
ls -l
```

----结束

将数据导入至目标 DDM（非拆分表场景）

在ECS上采用MySQL客户端直连目标DDM关联的RDS，直接执行以下命令导入表结构文本文件和数据文件。

如果是单表或普通表，

```
mysql -f -h {RDS_ADDRESS} -P {RDS_PORT} -u {RDS_USER} -p {DB_NAME} < {mysql_table_schema.sql}
Enter password: **********
mysql -f -h {RDS_ADDRESS} -P {RDS_PORT} -u {RDS_USER} -p {DB_NAME} < {mysq_table_data.sql}
Enter password: **********
```

- RDS_ADDRESS为待导入数据的RDS的地址。
- RDS_PORT为RDS实例的端口。
- RDS_USER为RDS实例的用户名。
- DB_NAME为RDS数据库名称，如果导入的是单表，DB_NAME为RDS第一个分片的物理数据库。
- mysql_table_schema.sql为待导入的表结构文件名。
- mysq_table_data.sql为待导入的表数据文件名。

将数据导入至目标 DDM（拆分表、全局表场景）

在ECS上采用MySQL客户端连接DDM执行如下命令，将整库数据文件导入DDM。

```
mysql -f -h {DDM_ADDRESS} -P {DDM_PORT} -u {DDM_USER} -p {DB_NAME} < {mysql_data.sql}
Enter password: **********
```

表 6-3 参数解释

参数	说明	备注
DDM_ADDRESS	待导入数据的DDM实例连接地址。	可在DDM管理控制台上，实例基本信息页面查看“连接地址”和“端口”。
DDM_PORT	待导入数据的DDM侦听端口。	
DDM_USER	访问DDM的用户。	创建DDM逻辑库时使用的账号，需具备读写权限。
DB_NAME	待导入数据的DDM逻辑库名称。	-
mysql_data.sql	待导入整库数据文件的名称。	即导出数据中**步骤2**导出的文件名称。

7 通过 JDBC 连接池连接 DDM

操作场景

连接池实现原理：在系统初始化的时候，将数据库连接作为对象存储在内存中，当用户需要访问数据库时，发出请求，直接从连接池中取出一个已建立的空闲连接对象。使用完毕后，再将连接放回连接池中，供下一个请求访问使用。连接的建立、断开都由连接池自身来管理。同时，还可以通过设置连接池的参数来控制连接池中的初始连接数、连接的上下限数以及每个连接的最大使用次数、最大空闲时间等等。也可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

本章节主要介绍了如何通过JDBC连接池与DDM对接，实现数据操作。如果是Java程序，建议您使用**HikariCP**。

- Java 8：建议使用3.3.1版本。
- Java 7：建议使用2.4.13版本。
- JDBC连接DDM不支持开启用户游标提取（useCursorFetch）参数。

操作步骤

步骤1 配置Maven。

- Java 8：

```
<dependency>
   <groupId>com.zaxxer</groupId>
   <artifactId>HikariCP</artifactId>
   <version>3.3.1</version>
</dependency>
```

- Java 7：

```
<dependency>
   <groupId>com.zaxxer</groupId>
   <artifactId>HikariCP-java7</artifactId>
   <version>2.4.13</version>
</dependency>
```

步骤2 创建表。

表 7-1 创建表。

表名	字段	类型	是否主键
account	account_number	bigint	是
	account_type	varchar（45）	否
	account_name	varchar（50）	否

步骤3 连接DDM实例。

1. 配置连接数：JdbcUrl连接串中的参数和HikariCP参数。
2. 插入数据。

示例：

```
package com.huawei.ddm.examples;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

public class HikariCPDemo {
    private static DataSource datasource;
    private static DataSource getDataSource() {
        if (datasource == null) {
            HikariConfig config = new HikariConfig();

            // 配置JdbcUrl连接串中的参数
            config.setJdbcUrl("jdbc:mysql:loadbalance://192.168.0.10:5066,192.168.0.11:5066/db_name?
loadBalanceAutoCommitStatementThreshold=5&loadBalanceHostRemovalGracePeriod=15000&loadBal
anceBlacklistTimeout=60000&loadBalancePingTimeout=5000&retriesAllDown=10&connectTimeout=10
000&socketTimeout=60000');
            /*
            // 配置JdbcUrl连接串中的参数也可以通过以下方法进行:
            config.addDataSourceProperty("loadBalanceAutoCommitStatementThreshold",5);
            config.addDataSourceProperty("loadBalanceHostRemovalGracePeriod", 15000);
            config.addDataSourceProperty("loadBalanceBlacklistTimeout", 60000);
            config.addDataSourceProperty("loadBalancePingTimeout", 5000);
            config.addDataSourceProperty("retriesAllDown", 10);
            config.addDataSourceProperty("connectTimeout", 10000);
            */
            config.setUsername("username");
            config.setPassword("password");
            config.setMaximumPoolSize(10);
            config.setAutoCommit(true);

            // 配置HikariCP参数
            config.addDataSourceProperty("cachePrepStmts", true);
            config.addDataSourceProperty("prepStmtCacheSize", 250);
            config.addDataSourceProperty("prepStmtCacheSqlLimit", 2048);
            config.addDataSourceProperty("minimumIdle", 5);
            config.addDataSourceProperty("maximumPoolSize", 10);
            config.addDataSourceProperty("idleTimeout", 30000);

            datasource = new HikariDataSource(config);
        }
        return datasource;
    }

    public static void main(String[] args) {
        Connection connection = null;
        PreparedStatement pstmt = null;
```



```
        ResultSet resultSet = null;
        try {
            DataSource dataSource = getDataSource();
            connection = dataSource.getConnection();
            System.out.println("The Connection Object is of Class: " + connection.getClass());

            // 插入测试数据
            String insertSql = "insert into account(account_number, account_type, account_name)
values(?, ?, ?);";
            PreparedStatement insertStmt = connection.prepareStatement(insertSql);
            insertStmt.setLong (1, 1L);
            insertStmt.setString (2, "manager");
            insertStmt.setString (3, "demotest01");
            insertStmt.executeUpdate();
            connection.commit ();

            // 查询数据
            pstmt = connection.prepareStatement("SELECT * FROM account");
            resultSet = pstmt.executeQuery();
            while (resultSet.next()) {
                String accountNumber = resultSet.getString("account_number");
                String accountType = resultSet.getString("account_type");
                String accountName = resultSet.getString("account_name");
                System.out.println(accountNumber + "," + accountType + "," + accountName);
            }
        } catch (Exception e) {
            try {
                if (null != connection) {
                    connection.rollback();
                }
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        }
    }
}
```

----结束

8 通过 Navicat 客户端连接 DDM 实例

操作场景

本文将介绍如何获取弹性公网的IP地址，并通过Navicat客户端连接DDM实例。

Navicat 客户端连接 DDM 实例

步骤1 登录分布式数据库中间件服务，单击需要连接的DDM实例名称，进入实例基本信息页面。

步骤2 在“实例信息”模块的弹性公网IP单击“绑定”。选择任意地址进行绑定。

步骤3 在DDM管理控制台左侧选择虚拟私有云图标。单击“访问控制>安全组”

步骤4 在安全组界面，单击操作列的“配置规则”，进入安全组详情界面。在安全组详情界面，单击“添加规则”，弹出添加规则窗口。根据界面提示配置安全组规则，设置完成后单击“确定”即可。

说明

- 绑定弹性公网IP后，建议您在内网安全组中设置严格的出入规则，以加强数据库安全性。

步骤5 打开Navicat客户端，单击“连接”。在新建连接窗口中填写主机IP地址（弹性公网IP地址）、用户名和密码（DDM帐号、密码）。

步骤6 单击“连接测试”，若显示连接成功，单击“确定”，等待1-2分钟即可连接成功。连接失败会直接弹出失败原因，请修改后重试。

----结束

说明

通过其他可视化的MySQL工具（例如 Workbench）连接DDM实例的操作与此章基本一致，不做详细描述。

9 如何选择 DDM 逻辑库分片数

在DDM实例创建逻辑库时，对需要水平拆分的表，需要选择合适的RDS实例和单RDS分片数量，通过预估拆分表的数据量，规划出逻辑库分片数量，可以避免各个分片的上单张表数据容量过高。

- 逻辑库选择“非拆分”模式时，一个逻辑库仅对应一个RDS实例，在该RDS实例上仅创建1个分片。
- 逻辑库选择“拆分”模式时，一个逻辑库可对应多个RDS for MySQL实例，在每个RDS for MySQL实例上，您可创建1-64个分片。

  一个分片即为MySQL实例的一个Database，逻辑库的总分片数等于所关联的RDS分片数总和。

  说明

  - 您对当前逻辑库所进行的**扩容**操作，不会改变逻辑库总分片数。即平滑扩容的实质：用更多的RDS来分流一部分分片的数据（如：原来是1个RDS for MySQL实例共8个分片，扩容后，每个RDS for MySQL实例4分片，2个实例共计8个分片）。
  - DDM后续将会推出翻倍扩容的功能，以满足分片裂变的需求。

如何选择单 RDS 分片数量

单RDS分片数需根据需要合理规划，当单RDS实例规格超过规格上限时将会直接系统影响性能，在总分片数固定的前提下，要评估单RDS实例规格，权衡RDS实例数量和单RDS分片数的关系。

分片估算公式如下：

- 总分片数={总记录数}/{单表记录数}={RDS实例数}*{单RDS分片数}
- {单RDS实例记录数}={单表记录数}*{单RDS分片数}
- 单RDS存储容量={单RDS实例记录数}*{单记录字节数}
- 单RDS实例规格=存储容量、吞吐性能、响应延迟、连接数、物理资源等

对需要水平拆分的表，评估未来1-2年后的业务规模，可参考**表9-1**评估单RDS分片数。

表 9-1 业务规格评估参考

项	计算方法参考	用户取值	示例
总记录数	根据业务规模评估	按实际填写	10亿条 =1,000,000,000
单表记录数	根据业务规模评估	按实际填写	1000万条
总分片数	{总记录数}/{单表记录数} (等价{RDS实例数}*{单RDS分片数})	按实际填写	100
单RDS分片数	{总分片数}/{ RDS实例数} 支持取值范围枚举: [8,16,32,64,128] **说明** ● 单RDS分片数需根据实际情况合理规划，当分片数过多导致单RDS实例规格超过规格上限时将会直接系统影响性能。 ● 低规格实例的分片数若过多，建议增加调整实例规格或者实例数量。	按实际填写	32

10 如何选择拆分键和拆分算法

本章节主要介绍拆分键和拆分算法的使用场景，以及如何合理的选择拆分键和拆分算法。

相关介绍请参见表10-1。

表 10-1 拆分键与拆分算法使用简介

拆分算法	hash类		range类	
拆分键	表字段	表字段+日期函数	表字段	表字段+日期函数
详细说明	根据指定的表字段将数据平均拆分到各个分片上。	根据指定的表字段+日期函数将数据平均拆分到各个分片上。 表字段必须是日期类型（date、datetime、timestamp）	将数据表内的记录按照算法元数据定义的规则将数据拆分到指定的分片上。	根据指定的表字段+日期函数将数据按照算法元数据的规则将数据拆分到各个分片上。 表字段必须是日期类型（date、datetime、timestamp）

适用场景	适用于需要将数据均匀分布的场景，例如：银行类客户业务应用，业务逻辑主体是客户，可使用客户对应的表字段（例如客户号）作为拆分键，详情参见如下示例。	需要按时间（年、月、日、周及其组合）对数据进行拆分的场景，例如：游戏类的应用，可使用玩家对应的表字段（例如玩家注册时间）作为拆分键，按日、月、年等函数分片，方便统计和查询某日、月玩家的操作数据，帮助游戏厂家做大数据分析。	适合范围类操作较多的场景，例如：电商类应用，如果业务场景是围绕商家做活动进行，业务逻辑主体是活动日期，可使用活动日期对应的表字段（例如活动名称、日期范围）作为拆分键，方便统计某周期内销量等情况。	例如日志分析场景，日志系统中可能包含各类复杂的信息，这时您可以选择时间字段作为拆分键，然后对拆分键使用日期函数拆分。 为了方便日志清理和转储，采用range拆分算法，对时间字段用日期函数转换成年，表示按年存储到各个分片上，详情参见如下示例。

如何选择拆分算法

拆分算法即将逻辑表中数据拆分到多个数据库分片上的算法，DDM支持hash和range两种拆分算法。

- hash类

  将数据均匀分布在各个分片。

  适用于SQL查询条件使用“=”、“IN”之类运算符相对较多的场景。

- range类

  将数据表内的记录按照算法原数据的取值范围进行分片存储。

  适用于SQL查询条件使用“>”、“<”、“BETWEEN ... AND ...”之类运算符相对较多的场景。

拆分算法的使用，需要结合业务查询场景进行评估，以选择合适的拆分算法，提升DDM效率。

说明

在对逻辑库进行扩容时：

- 按hash算法分片的逻辑表，表数据涉及到迁移，会被重新均匀分布所有分片上。
- 按range算法分片的逻辑表，表数据默认不做迁移，如果您修改了拆分规则，重新定义范围分布，请自行直连RDS分片完成数据迁移。

如何选择拆分键

拆分键是在水平拆分逻辑表的过程中，用于生成路由结果的表字段，指定表字段后，可以进一步选择日期函数，也可以手动输入“日期函数(字段名)”，数据表字段必须是日期类型（date、datetime、timestamp），日期函数适用于需要按时间（年、月、日、周及其组合）对数据进行拆分的场景。

DDM根据拆分键与拆分算法计算路由结果，对拆分表的数据进行自动水平拆分，分发到数据分片中。

选取拆分算法与拆分键一般遵循以下规则：

- 尽可能使数据均匀分布到各个分片上。
- 该拆分键是最频繁或者最重要的查询条件。
- 优选主键作为拆分键，因为主键作为查询条件时，查询速度最快。

有明确主体的业务场景

拆分表的数据量一般都达到千万级，因此选择合适的拆分算法和拆分键非常重要。如果能找到业务主体，并且确定绝大部分的数据库操作都是围绕这个主体的数据进行的，那么可以选择这个主体所对应的表字段作为拆分键，进行水平拆分。

业务逻辑主体与实际应用场景相关，下列场景都有明确的业务逻辑主体。

1. 银行类客户业务应用，业务逻辑主体是客户，可使用客户对应的表字段（例如客户号）作为拆分键。部分业务系统的业务场景为围绕银行卡/账号的，可以选取卡/账号作为拆分键。
2. 电商类应用，如果业务场景是围绕商品进行操作，业务逻辑主体是商品，可使用商品对应的表字段（例如商品编码）作为拆分键。
3. 游戏类的应用，主要围绕玩家数据进行操作，业务逻辑主体是玩家，可使用玩家对应的表字段（例如玩家id）作为拆分键。

以银行类客户业务为例，建表SQL语句如下：

```
CREATE TABLE PERSONALACCOUNT (
ACCOUNT VARCHAR(20) NOT NULL PRIMARY KEY,
NAME VARCHAR(60) NOT NULL,
TYPE VARCHAR(10) NOT NULL,
AVAILABLEBALANCE DECIMAL(18,2) NOT NULL,
STATUS CHAR(1) NOT NULL,
CARDNO VARCHAR(24) NOT NULL,
CUSTOMID VARCHAR(15) NOT NULL
) ENGINE=INNODB DEFAULT CHARSET=UTF8;
```

无明确主体的业务场景

如果业务场景中找不到合适的主体，也可以选择那些数据分布较为均匀的属性所对应的表字段作为拆分键。

例如日志分析场景，日志系统中可能包含各类复杂的信息，这时您可以选择时间字段作为拆分键。

选择时间字段作为拆分键时，支持对拆分键使用日期函数拆分。

为了方便清理和转储，采用range拆分算法，对时间字段用日期函数转换成年，表示按年存储到各个分片上。

建表SQL语句：

```
CREATE TABLE LOG (
LOGTIME DATETIME NOT NULL,
LOGSOURCESYSTEM VARCHAR(100),
LOGDETAIL VARCHAR(10000)
);
```

