DB-UNIFIER 使用指南

用于 db-unifier v0.5

作者：虞越

Source code availability:
https://github.com/wuyuetiger/db-unifier
https://code.csdn.net/tigeryu/db-unifier
https://git.oschina.net/db-unifier/db-unifier

前言

db-unifier 是一种对各种关系型数据库的统一包装工具，使用简便高效。

db-unifier 旨在保障数据库操作的简便高效的同时，以统一的接口屏蔽各种数据库 JDBC 操作的差异性，从而方便实现不同类型数据库之间的移植，使基于 db-unifier 开发的产品适应数据库环境的能力更强。

db-unifier 高效且统一的接口主要包括以下几个方面：

数据库字段类型被映射为统一的五种基本类型：String、Number、Timestamp、Clob、Blob。

自动管理数据库连接，无需用户硬代码关闭，避免连接泄漏的问题。

数据库表和视图信息获取。

数据库表分页查询。

数据库其它常用 SQL 操作。

数据库 LOB 字段的处理。

数据库序列处理。

数据库编码的处理。

与 Spring 的完美整合，支持基于 AOP 的声明式事务，以及与 Hibernate、iBatis 的混用。

另外 db-unifier 还提供了两个不错的工具：

一个是代码生成工具，用户通生成器可以快速生成单表或单视图的标准 BO 类和 DAO 类。

另一个是导入导出工具，用户可以在不同数据库之间进行数据高效的导入导出操作。

配置

使用环境

db-unifier 必需在 JDK1.6 及以上版本环境下使用。

依赖 JAR 包

db-unifier 的核心 jar 包名 db-unifier-v0.5.jar，可以从官方网站的 dist 目录中下载。

db-unifier 运行时所依赖的 jar 包如下：

dom4j-1.6.1.jar：用于 db-unifier 配置的读取。

freemarker.jar：用于代码生成，该功能非核心功能，不使用该功能可以不加载此 jar。

persistence.jar：用于生成的 BO 类代码，以符合 JPA 规范，该功能非核心功能，不使用该功能可以不加载此 jar。

db-unifier 经过测试的数据库 JDBC 驱动 jar 包如下：

ojdbc6.jar：Oracle 11g 驱动，兼容 Oracle 10g 和 Oracle 9i。

sqljdbc4.jar：Microsoft SQL Server 2012 驱动，兼容 SQL Server 2010、SQL Server 2008、SQL Server 2005。

mysql-connector-java-5.1.5-bin.jar：MySQL 5.x 驱动。

db2jcc4.jar：DB2 10.x 驱动。

jconn4.jar：Sybase 15.x 驱动。

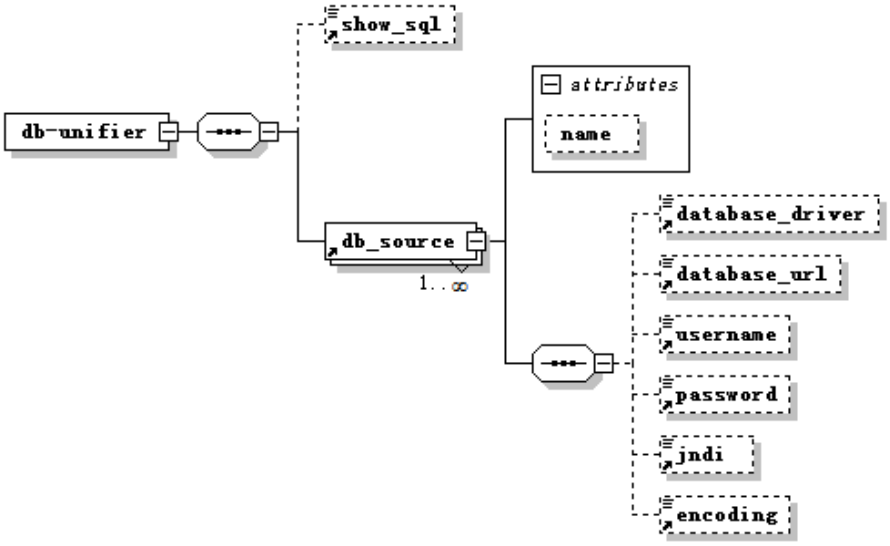
postgresql-9.3-1101.jdbc4.jar：PostgreSQL 9.3 驱动。

derby.jar：Derby 10.10 Server 及嵌入式驱动。

配置文件

db-unifier 的配置文件为一个 XML 文件，名称为 db-unifier.config.xml，该文件必需放置于类的根路径下在会生效（一般置于 classes 目录中）。

db-unifier.config.xml 的 schema 描述如下：



下面给出配置的示例文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<db-unifier xmlns="http://www.sosostudio.org/db-unifier/config">
	<show_sql>true</show_sql>
	<db_source>
		<database_driver>org.apache.derby.jdbc.EmbeddedDriver</database_driver>
		<database_url>jdbc:derby:db;create=true</database_url>
	</db_source>
	<db_source name="encoding_dbsource">
		<database_driver>org.apache.derby.jdbc.EmbeddedDriver</database_driver>
		<database_url>jdbc:derby:db;create=true</database_url>
		<encoding>UNICODE</encoding>
	</db_source>
	<db_source name="autocode">
		<database_driver>org.apache.derby.jdbc.EmbeddedDriver</database_driver>
		<database_url>jdbc:derby:db;create=true</database_url>
	</db_source>
	<db_source name="exporter">
		<database_driver>org.apache.derby.jdbc.EmbeddedDriver</database_driver>
		<database_url>jdbc:derby:db;create=true</database_url>
	</db_source>
	<db_source name="importer">
		<database_driver>org.apache.derby.jdbc.EmbeddedDriver</database_driver>
		<database_url>jdbc:derby:db;create=true</database_url>
	</db_source>
	<db_source name="oracle">
		<database_driver>oracle.jdbc.driver.OracleDriver</database_driver>
		<database_url>jdbc:oracle:thin:@localhost:1521:orcl</database_url>
```

```
            <username>system</username>
            <password>password</password>
        </db_source>
        <db_source name="mssqlserver">
            <database_driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</database_driver>
            <database_url>jdbc:sqlserver://localhost:1433;DatabaseName=master</database_url>
            <username>sa</username>
            <password>password</password>
        </db_source>
        <db_source name="mysql">
            <database_driver>com.mysql.jdbc.Driver</database_driver>
            <database_url>jdbc:mysql://localhost:3306/test</database_url>
            <username>root</username>
            <password>password</password>
        </db_source>
        <db_source name="db2">
            <database_driver>com.ibm.db2.jcc.DB2Driver</database_driver>
            <database_url>jdbc:db2://localhost:50000/SAMPLE</database_url>
            <username>db2admin</username>
            <password>password</password>
        </db_source>
        <db_source name="sybase">
            <database_driver>com.sybase.jdbc4.jdbc.SybDriver</database_driver>
            <database_url>jdbc:sybase:Tds:localhost:5000/master</database_url>
            <username>sa</username>
            <password>password</password>
        </db_source>
        <db_source name="postgresql">
            <database_driver>org.postgresql.Driver</database_driver>
            <database_url>jdbc:postgresql://localhost:5432/postgres</database_url>
            <username>postgres</username>
            <password>password</password>
        </db_source>
        <db_source name="jndi">
            <jndi>java:test</jndi>
        </db_source>
</db-unifier>
```

show_sql：用于控制是否打印 SQL 语句。

db_source：用于连接指定的数据库，该配置的名字为一个 name 属性，如果未设置 name 属性则表示该配置为 db-unifier 的默认连接数据库。

获取连接默认数据库的 DbUnifier 对象创建如下：

```
DbUnifier unifier = new DbUnifier();
```

获取连接指定名称数据库的 DbUnifier 对象创建如下：

```
DbUnifier unifier = new DbUnifier("name");
```

db_source 有两种配置方式：

JDBC 方式，需要提供 database_driver（数据库驱动）、database_url（数据库连接串）、username（用户名）和 password（口令），在一些特殊的情况用户名和口令可以不提供。

JNDI 方式，需要提供 jndi（JavaEE JNDI），在一些特殊的情况下，还需要提供 username（用户名）和 password（口令）。

这两种方式还有一个非必需的参数 encoding，该参数为数据库的内部字符集编码，仅使用于 BO 类对字符串截断的处理，一般没有太大的作用。db-unifier 对各类数据库都有一个默认的字符集编码，如下表：

数据库	默认字符集编码
Oracle	GBK
Microsoft SQL Server	GBK
MySQL	UNICODE（无论数据库采用什么编码，请都指定成 UNICODE）
DB2	UTF8
Sybase	GBK
PostgreSQL	UNICODE（无论数据库采用什么编码，请都指定成 UNICODE）
Derby	UNICODE（无论数据库采用什么编码，请都指定成 UNICODE）

如果实际的数据库字符集编码与默认的不符，可以在 db_source 中配置。

Spring 整合

db-unifier 可以很方便的与 Spring 整合，利用 Spring 的 IoC 功能进行 DbUnifier 类的装配，利用 Spring 的 AOP 功能进行事务的控制。具体的 Spring 参考配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tx="http://www.springframework.org/schema/tx"
```

```
:schemaLocation="
        ://www.springframework.org/schema/beans
        ://www.springframework.org/schema/beans/spring-beans.xsd
        ://www.springframework.org/schema/tx
        ://www.springframework.org/schema/tx/spring-tx.xsd"
lazy-init="true">

        <bean id="dataSourceTarget"
        ="org.springframework.jdbc.datasource.DriverManagerDataSource">
                <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
                <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
                <property name="username" value="system" />
                <property name="password" value="password" />
        </bean>

        <bean id="dataSource"
        ="org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy">
                <property name="dataSource" ref="dataSourceTarget" />
        </bean>

        <bean id="transactionManager"
        ="org.springframework.jdbc.datasource.DataSourceTransactionManager">
                <property name="dataSource" ref="dataSource" />
        </bean>

        <tx:annotation-driven transaction-manager="transactionManager" />

        <bean id="dbSource" class="org.sosostudio.dbunifier.dbsource.WrappedDbSource">
                <property name="dataSource" ref="dataSource" />
                <property name="encoding" ref="GBK" />
        </bean>

        <bean id="dbUnifier" class="org.sosostudio.dbunifier.DbUnifier">
                <constructor-arg ref="dbSource" />
        </bean>


</beans>
```

如果要在 Spring 中混用 Hibernate 和 db-unifier，可以先创建 Hibernate 的 SessionFactory Bean，并在此基础上将 org.springframework.jdbc.datasource.DataSourceTransactionManager 的实现换成 org.springframework.orm.hibernate3.HibernateTransactionManager 即可。

数据类型

在 db-unifier 中数据库字段类型被映射为统一的五种基本类型：String、Number、Timestamp、Clob、Blob。其中 String 又可以分为 National 和非 National 两个亚种，National 的 String 类型长度以字符计算，非 Nationall 的 String 类型长度以字节计算。

db-unifier 的各种类型到 JAVA 类型的映射关系见下表：

db-unifier 类型	JAVA 类型
String	String
Number	java.math.BigDecimal
Timestamp	java.sql.Timstamp
Clob	String
Blob	Byte[]

db-unifier 的各种类型到数据库字段类型的映射关系见测试章节。

核心类 DbUnifier

构造 DbUnifier

DbUnifier 是 db-unifier 中最核心的类，几乎所有的功能方法都位于该类中。

构造 DbUnifier 有以下几种方式：

从 DbSource 构造 DbUnifier，这种构造方式是所有 DbUnifier 构造方式的基础，其它方式归根结底都是以这种方式来构造的。在 db-unifier 中有四种 DbSource 可用于 DbUnifier 的构造。这几种方式的区别见下表：

DbSource 实现类	作用	连接处理
JdbcDbSource	通过 JDBC 获取数据库连接	由 DbUnifier 自动处理，当连接不再使用时由 DbUnifier 自动关闭或交由上层容器处理。
JndiDbSource	通过 JNDI 获取数据库连接	
WrappedDbSource	通过 DataSource 获取数据库连接	

ConnectionDbSource	沿用外部代码传入的数据库连接	由外部代码控制连接关闭。

从配置文件构造 DbUnifier，这是开发时一般使用的方式。这种方式在配置章节已有描述，具体内容请参见配置章节，这里不再赘述。

沿用外部代码传入的数据库连接来构造 DbUnifier，这种方式运用在需要高性能且又无 Spring 事务环境的情况下，具体参考代码片段如下：

```
DbSource dbSource = XmlConfig.getDbSource("name");
Connection con = dbSource.getConnection();
dbSource = new ConnectionDbSource(con);
DbUnifier unifier = new DbUnifier(con);
```

数据库基本操作

getTableList、getTable 方法用于获得数据库表的信息。其中 getTableList 有参数 sortByFk，该参数默认为 false，当该参数为 true 时，数据库表将按外键关系，主表在前，从表在后的顺序返回列表，这在某些场合是非常有用的，但这会使该方法执行消耗更多的时间。

getViewList、getView 方法用于获得数据库视图的信息。

createTable、dropTable 分别的创建数据库和删除数据库表的方法。

数据库查询操作

executeSelectSql 方法用于数据库查询。查询的参数主要有以下几个：

sql：查询的 SQL 语句。

values：查询条件中所包含的数据值。Values 只能对应支持 db-unifier 支持的五种数据类型，分别为 String、Number、Timestamp、Clob 和 Blob，用户不用关心数据库字段类型，只要在这五种类型中做出选择就行了。

containsLob：该布尔值表示是否要返回 Lob 字段的实际内容。当 Lob 字段内容很大或者返回的 Lob 字段内容很多时，建议设置 containsLob 为 false，避免过大或过多的 Lob 造成内存不足的异常。当 containsLob 为 false 时，Clob 和 Blob 永远返回 null。

pageSize：分页时每页的大小（记录数）。

pageNumber：分页时指定的页数。

下面是一个使用 executeSelectSql 进行查询的例子：

```
RowSet rowSet = unifier.executeSelectSql("select * from TAB", 100, 50);
```

一种经过包装的 SelectSql 对象可以辅助用户来拼写 SQL 语句，避免用户自己拼写时错误造成异常。下面是一个使用 SelectSql 对象进行查询的示例：

```
SelectSql selectSql = new SelectSql()
    .setTableName("USER")
    .setColumns("*")
    .setOrderByClause(new OrderByClause()
        .addOrder("USERNAME", Direction.ASC));
RowSet rowSet = unifier.executeSelectSql(selectSql, 3, 2);
```

查询的结果为一个 RowSet 对象，其所包含的字段有以下几个：

rowList：返回所有记录行的列表，每行为一个 Row 对象。Row 对象只提供五种数据类型的获取方法。Row 还提供了到 Java Bean 类的转换方法 toBean（RowSet 提供了 toBeans 的方法），其主要依赖于 JPA 的 annotation。下面是一个使用 toBean 的示例：

```
SysTest sysTest = (SysTest) row.toBean(SysTest.class);
```

pageSize：分页时每页的大小（记录数）。

pageNumber：分页时指定的页数。

totalRowCount：查询结果的总记录数，并不是当前页的记当数。

totalPageCount：查询结果的总页数。

RowSet 还提供了一组 getOnlyXXXXX 的方法，假设结果集中只包含一条记录且只有一个字段，可以很方便的通过此方法直接获取该字段的值。

遍历 RowSet 一般采用以下示例中的类似代码：

```
for (int i = 0; i < rowSet.size(); i++) {
    Row row = rowSet.getRow(i);
    String username = row.getString("USERNAME");
    System.out.println(username);
}
```

数据库其它 SQL 操作

executeOtherSql 方法用于其它非数据库查询的 SQL 操作。使用方法上同

executeSelectSql。executeOtherSql 的返回值是成功操作所影响的记录数。

多种对 SQL 语句包装的类分别可用于对数据库表的增（InsertSql）、删（DeleteSql）、改（Update）操作。

下面是一个通过 InsertSql 进行数据插入的操作示例：

```
unifier.executeInsertSql(new InsertSql()
    .setTableName(tableName)
    .setInsertKeyValueClause(new InsertKeyValueClause()
        .addStringClause(columnName, i + "")));
```

数据库 LOB 操作

getSingleClob、setSingleClob、getSingleBlob 和 getSingleBlob 这四个方法主要用于对单个 Lob 字段的流式处理，通过这种方式可以最大程度减少内存的开销，提高操作的效率。

下面是一段对数据库 Blob 字段读出存成文件的操作。

```
OutputStream os = new BufferedOutputStream(
    new FileOutputStream("/file.dat"));
unifier.getSingleBlob("select content from process_file where id=?",
    new Values().addStringValue("1"), os);
os.close();
```

数据库序列操作

getSequenceNextValue 方法用于获取序列值。其采用了数据库表模拟的方式来生成序列，保证了各类数据库操作的一致性和兼容性。其还采用了乐观锁的方式实现了序列值获取的算法，保证了在集群、多线程的情况下能正确顺序取号，不重号、不跳号。

代码生成

db-unifier 的代码生成方法用于生成单表或单视图的标准 BO 类和 DAO 类。生成的类可以简化开发代码，通过编译器识别避免低级的书写错误，使开发者更专注于业务。另外，也可以在此基础上配合一定的代码来防止 SQL 注入的问题。

生成的 BO 类因为符合 JPA 的规范，除了可以用于自己对应的 db-unifier DAO 类，也可以用于 Hibernate。又因为 BO 类符合 JAXB 规范，还可用于 CXF 生成的 Web Service 方法。

假设我们要对 Oracle 中的 TEST1 和 TEST2 表进行代码生成，需要做以下几步：

首先安装 JDK1.6，并确保 JDK 的 bin 目录在系统环境变量 PATH 中。

然后，新建一个目录，将 db-unifier-v0.5.jar、dom4j-1.6.1.jar、freemarker.jar、persistence.jar、ojdbc6.jar 复制到其中，再在该目录中创建 src 目录。

接着，在 classes 目录中创建 db-unifier.config.xml 文件。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<db-unifier xmlns="http://www.sosostudio.org/db-unifier/config">
	<show_sql>true</show_sql>
	<db_source name="autocode">
		<database_driver>oracle.jdbc.driver.OracleDriver</database_driver>
		<database_url>jdbc:oracle:thin:@localhost:1521:orcl</database_url>
		<username>system</username>
		<password>password</password>
	</db_source>
</db-unifier>
```

最后通过执行以下命令就可以自动生成代码了：

```
java -classpath ".; db-unifier-v0.5.jar; dom4j-1.6.1.jar; freemarker.jar; persistence.jar; ojdbc6.jar" org.sosostudio.dbunifier.autocode.DaoGenerator src org.sosostudio.test TEST1 TEST2
```

这其中几个参数的含义如下：

src：生成目标 BO 和 DAO 类的根路径，这里的路径是相对于当前目录的相对路径。

org.sosostudio.test：生成目标 BO 和 DAO 类的 package。

TEST1 TEST2：需要生成的表，可以一直往后追加，一次生成多个表的 BO 和 DAO 类。

导入导出

db-Unifier 的导入导出方法用于在不同数据库之间进行数据高效的导入导出操作。由于该方法可以跨数据库导数据，支持自动侦测数据库表的主从关系，保障数据库的导入不受外键约束的影响，支持 LOB 字段，而且效率很高，具有很高的实用性。不过要注意的是此方法因为是基于 db-unifier 的，所以 db-unifier 所不支持的数据库字段类型，也不支持导入导出，具体可以参见测试章节。

假设我们要把 Oracle 中的所有表导入 MySQL 中需要做以下几步：

首先安装 JDK1.6，并确保 JDK 的 bin 目录在系统环境变量 PATH 中。

然后，新建一个目录，将 db-unifier-v0.5.jar、dom4j-1.6.1.jar、ojdbc6.jar、mysql-connector-java-5.1.5-bin.jar 复制到其中。

接着，在 classes 目录中创建 db-unifier.config.xml 文件。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<db-unifier xmlns="http://www.sosostudio.org/db-unifier/config">
	<show_sql>true</show_sql>
	<db_source name=" exporter ">
		<database_driver>oracle.jdbc.driver.OracleDriver</database_driver>
		<database_url>jdbc:oracle:thin:@localhost:1521:orcl</database_url>
		<username>system</username>
		<password>password</password>
	</db_source>
	<db_source name="importer">
		<database_driver>com.mysql.jdbc.Driver</database_driver>
		<database_url>jdbc:mysql://localhost:3306/test</database_url>
		<username>root</username>
		<password>password</password>
	</db_source>
</db-unifier>
```

最后通过执行以下命令先从 Oracle 中导出数据：

```
java -classpath ".; db-unifier-v0.5.jar; dom4j-1.6.1.jar; ojdbc6.jar"
org.sosostudio.dbunifier.pipe.DbExporter
```

导出过程中一直有提示信息输出，最终会生成一个 result 目录，并在其中生成一堆文件。

接着执行以下命令将这些数据导入 MySQL：

```
java -classpath ".; db-unifier-v0.5.jar; dom4j-1.6.1.jar; mysql-connector-java-5.1.5-bin.jar"
org.sosostudio.dbunifier.pipe.DbImporter
```

测试

db-unifier 采用 JUnit 进行单元测试，下表是对各类数据库各种数据类型的测试结果，其中各种颜色所代表的含义如下：

白色：该数据库不支持的字段类型。

红色：该数据库支持，但是 db-unifier 从移植兼容性考虑不予以支持的字段类型。

绿色：该数据库支持，且 db-unifier 能保证其移植兼容性的字段类型。db-unifier 建议使用者没有特殊情况，应该尽量使用这些字段类型来建数据库表。

数据库类型	DB-UNIFIER 类型	Oracle	MS SQL Server	MySQL	DB2	Sybase	PostgreSQL	Derby
bigint	Number							
binary	Blob							
binary varying	Blob							
bit	Number							
blob	Blob							
bool	Number							
boolean	Number							
bytea	Blob							
char	String							
char binary	String							
char varying	String							
character	String							
character varying	String							
clob	Clob							
date	Timestamp							
datetime	Timestamp							
datetime2	Timestamp							
dec	Number							
decimal	Number							
double	Number							
double precision	Number							
enum	String							
fixed	Number							
float	Number							
image	Blob							
int	Number							
integer	Number							
long	Clob							
long raw	Blob							
long varchar	Clob							
longblob	Blob							
longtext	Clob							

mediumblob	Blob							
mediumint	Number							
mediumtext	Clob							
money	Number							
national char	String(National)							
national char varying	String(National)							
national character	String(National)							
national character varying	String(National)							
nchar	String(National)							
nchar varying	String(National)							
nclob	Clob							
number	Number							
numeric	Number							
nvarchar	String(National)							
nvarchar2	String(National)							
raw	Blob							
real	Number							
set	String							
smalldatetime	Timestamp							
smallint	Timestamp							
smallmoney	Number							
text	Clob							
time	Timestamp							
timestamp	Timestamp							
tinyblob	Blob							
tinyint	Number							
tinytext	Clob							
varbinary	Blob							
varchar	String							
varchar2	String							

数据库方面的移植能力

并不是使用了 db-unifier 就能完完全全的保证应用的移植性了，db-unifier 毕竟是一个开放性的工具，不会严格控制用户的操作。为了保证基于 db-unifier 在数据库方面的移植能力，请用户注意以下几个方面：

请使用 db-unifier 推荐的数据类型，没有特殊理由，不要使用别的数据类型，更不要使用测试清单上都没有的数据类型。详见下表：

	String	**String**	**Number**	**Timestamp**	**Clob**	**Blob**

		(National)				
Oracle	varchar2	nvarchar2	numeric	timestamp	clob	blob
MS SQL Server	varchar	nvarchar	numeric	datetime	text	image
MySQL	varchar	nvarchar	numeric	datetime	longclob	longblob
DB2	varchar	nvarchar	numeric	timestamp	clob	blob
Sybase	varchar	nvarchar	numeric	datetime	text	image
PostgreSQL	varchar	nvarchar	numeric	timestamp	clob	blob
Derby	varchar	varchar	numeric	timestamp	clob	blob

数据库对象命名的长度不要超过 30。

数据库对象命名和字段命名时不要加双引号强制名称大小写。

数据库对象命名和字段命名时不要使用中文，不要使用特殊符号，仅使用英文字母及下划线。不要使用单个英语单词命名，应以多个英文单词复合起来命名。

字符串字段长度不要超过 2000。

不要使用存储过程、触发器、定时器、专有函数（拼接字符串、类型转换）等等基于特别数据库实现的 PL 语言，这些会对移植带来超大的麻烦，所以在 db-unifier 里没有提供对这些数据库对象支持的方法。

不要设置级联删除和级联更新功能。

不要在同一张表上自己引用自己，建立外键关系。

不要使用数据库自带的自增值和序列。

不要以日期、LOB 类型的字段作为主键。

尽量不要拼写过于复杂的 SQL 语句。

把 select 语句作为子查询时要为该子查询起别名。

使用 SQL92 标准书写内、外连接查询，不要使用特定数据库的语法。

不要向数据库的字符串字段插入空字符串。因为在 Oracle 中这样做最终会插入一个 null 值，虽然这很不合理，但确实存在。