Hibernate 关于 OpenGauss 方言包

Quick Start

1. **FOCK 原仓库**:

项目原仓库：https://gitee.com/opengauss/examples

Fock 项目仓库：https://gitee.com/solisamicus/examples

2. **克隆个人仓库**:

```
1 $ git clone git@gitee.com:solisamicus/examples.git
```

3. **添加 `upstream` 源**：将原项目添加为 `upstream` 源，以便以后同步更新。

```
1 $ git remote add upstream https://gitee.com/opengauss/examples.git
```

4. **禁止向 `upstream` 源 push**：确保无法向 `upstream` 源，以避免误操作：

```
1 $ git remote set-url --push upstream no_push
```

$ git remote -v

origin git@gitee.com:solisamicus/examples.git (fetch)

origin git@gitee.com:solisamicus/examples.git (push)

upstream https://gitee.com/opengauss/examples.git (fetch)

upstream no_push (push)

需求分析与设计

需求分析

- 确定 OpenGauss 数据库的特性，包括 SQL 方言、特定函数、数据类型等。

- 参考现有的 Hibernate PostgreSQL 方言包，了解它是如何实现的。
- 明确项目产出要求，包括代码、测试、文档以及社区合并要求。

设计方案

设计方言包的基本架构，包括哪些类需要继承或重写，如何进行 SQL 方言支持、DDL 生成、数据类型映射等。

环境搭建

服务器配置

```
================================================
 System Information
================================================
Hostname          : VM-8-12-centos
Operating System  : Linux
Kernel Version    : 3.10.0-1160.119.1.el7.x86_64
Architecture      : x86_64
OS Release        : CentOS Linux release 7.6.1810 (Core)
Uptime            : up 4 days, 20 hours, 39 minutes

================================================
 CPU Information
================================================
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Model name:            Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
CPU MHz:               2394.364
L3 cache:              28160K

================================================
 Memory Information
================================================
Total Memory      : 3.6G
Used Memory       : 757M
Free Memory       : 218M

================================================
 Disk Space Information
================================================
Root Filesystem   : 5/59GB (Used/Total)
```

```
33
34 ================================================
35  Network Configuration
36 ================================================
37 Interface         : lo
38 IP Address        : 127.0.0.1/8
39
40 Interface         : eth0
41 IP Address        : 10.0.8.12/22
42
43
44 ================================================
45  Active Network Connections
46 ================================================
47 Netid  State      Recv-Q Send-Q                        Local Address:Port
                                 Peer Address:Port
48 udp    UNCONN     0      0                                    *:68
                                     *:*
49 udp    UNCONN     0      0                               10.0.8.12:123
                                     *:*
50 udp    UNCONN     0      0                               127.0.0.1:123
                                     *:*
51 udp    UNCONN     0      0          [fe80::5054:ff:fef4:485f]%eth0:123
                                   [::]:*
52 udp    UNCONN     0      0                                   [::1]:123
                                   [::]:*
53 tcp    LISTEN     0      128                                  *:22
                                     *:*
54 tcp    LISTEN     0      1200                                 *:25432
                                     *:*                    users:
   (("gaussdb",pid=12090,fd=8))
55 tcp    LISTEN     0      1200                                 *:25433
                                     *:*                    users:
   (("gaussdb",pid=12090,fd=10))
56 tcp    LISTEN     0      100                             127.0.0.1:25
                                     *:*
57 tcp    LISTEN     0      128                                 [::]:22
                                   [::]:*
58 tcp    LISTEN     0      1200                                [::]:25432
                                   [::]:*                     users:
   (("gaussdb",pid=12090,fd=9))
59 tcp    LISTEN     0      1200                                [::]:25433
                                   [::]:*                     users:
   (("gaussdb",pid=12090,fd=11))
60 tcp    LISTEN     0      100                                 [::1]:25
                                   [::]:*
61
```

```
================================================
 Disk IO Statistics
================================================
./server_config.sh: line 47: iostat: command not found

================================================
 Mounted Filesystems
================================================
Mount Point       : /
Filesystem Type   : ext4
```

OpenGauss 安装部署

opengauss:5.0.0 LTS

环境准备

安装依赖包

```
$ yum install libaio-devel flex bison ncurses-devel glibc-devel patch redhat-lsb-core readline-devel libnsl
```

修改 selinux

```
$ cat /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

关闭防火墙

```
$ systemctl stop firewalld
$ systemctl disable firewalld
```

```
$ systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)
```

关闭 swap

```
$ swapoff -a
```

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:           3.6G        206M        2.0G        576K        1.4G        3.2G
Swap:            0B          0B          0B
```

关闭 RemoveIPC

```
$ cat /etc/systemd/logind.conf | grep RemoveIPC
#RemoveIPC=no
```

修改字符集参数

`vim /etc/profile` 在行尾添加 `export LANG=en_US.UTF-8`

```
$ echo $LANG
en_US.UTF-8
```

修改 kernel.sem 值

```
$ sysctl -w kernel.sem="250 85000 250 330"
kernel.sem = 250 85000 250 330
```

安装 openngauss

新建用户组

```
$ groupadd dbgroup
```

新建用户

```
$ useradd -g dbgroup omm
```

设置用户

omm@123

```
$ passwd omm
Changing password for user omm.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
```

下载解压

```
$ curl -o opengauss.tar.gz https://opengauss.obs.cn-south-1.myhuaweicloud.com/5.0.0/x86/openGauss-5.0.0-CentOS-64bit-all.tar.gz
$ mkdir -p /opt/software/openGauss
$ tar -xvf opengauss.tar.gz -C /opt/software/
$ tar -jxf openGauss-5.0.0-CentOS-64bit.tar.bz2 -C /opt/software/openGauss
```

更改目录所有者

```
$ chown -R omm /opt/software/
```

设置目录权限

```
$ chmod -R 755 /opt/software/openGauss
```

切换数据库管理用户

```
$ su omm
```

```
$ cd /opt/software/openGauss/simpleInstall
```

```
$ sh install.sh -w "openGauss@123" -p 25432
```

验证安装部署

openGauss 进程

```
$ ps ux | grep gaussdb
```

```
[omm@VM-8-12-centos ~]$ ps ux | grep gaussdb
omm       8919  0.0  0.0 110480   904 pts/2    S+   18:14   0:00 grep --color=auto gaussdb
omm      30209  1.6 19.7 5995532 748568 ?      Ssl  18:08   0:05 /opt/software/openGauss/bin/gaussdb -D /opt/software/openGauss/data/single_node
```

openGauss 实例

```
$ gs_ctl query -D /opt/software/openGauss/data/single_node
```

工具远程连接

```
$ cat ~/.bashrc
# .bashrc

```

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging
feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
export GAUSSHOME=/opt/software/openGauss
export PATH=$GAUSSHOME/bin:$PATH
export LD_LIBRARY_PATH=$GAUSSHOME/lib:$LD_LIBRARY_PATH
export GS_CLUSTER_NAME=dbCluster
# ulimit -n 1000000
```

在 Vim 中，找到与 `ulimit` 相关的那一行，并在行首添加 `#` 以注释掉它。

配置兼容/外网访问

切换目录

```
$ cd  /opt/software/openGauss/data/single_node
```

`postgresql.conf` 添加/取消注释以下行：

```
listen_addresses = '*'

local_bind_address = '0.0.0.0'

password_encryption_type = 0
```

`pg_hba.conf` 添加以下行：

```
host    all             all             0.0.0.0/0               md5
```

启动或重启数据库（omm用户）：

```
$ gs_ctl restart -D $GAUSSHOME/data/single_node
```

进入数据库命令行（omm用户）：

```
$ gsql -d postgres -p 25432
gsql ((openGauss 5.0.0 build a07d57c3) compiled at 2023-03-29 03:07:56 commit
0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
```

创建用户：

```
openGauss=# CREATE USER postgres PASSWORD 'osapp_openGauss@123';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# GRANT ALL PRIVILEGES TO postgres;
ALTER ROLE
openGauss=# \q
```

JDBC

JDK

Java SE Development Kit 8u202

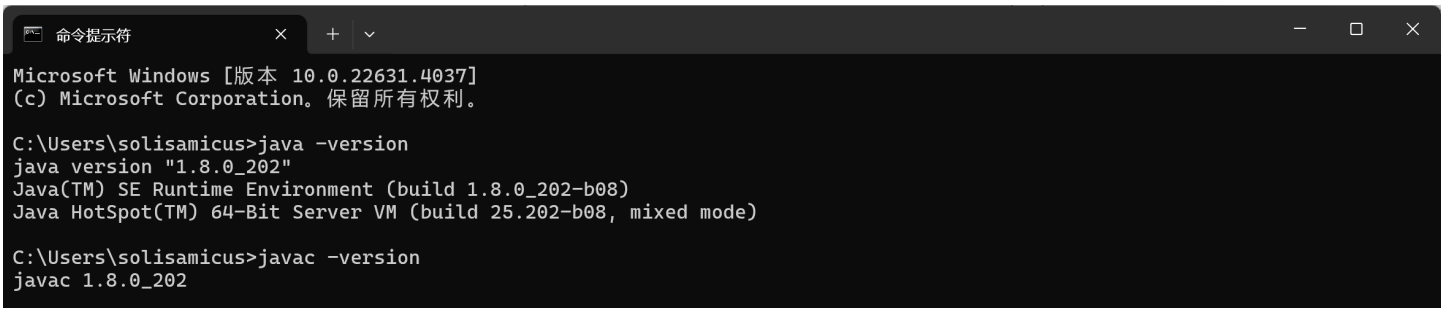
https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html

Search Oracle.com

The JDK is a development environment for building applications using the Java programming language. The JDK includes tools useful for developing and testing programs written in the Java programming la

配置环境变量：

变量名	变量值
JAVA_HOME	D:\SDK\Java\jdk1.8.0_202
Path	%JAVA_HOME%\bin %JAVA_HOME%\jre\bin
CLASSPATH	.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar



opengauss-jdbc

OpenGaussJDBCUtil.java：

```
1  import java.sql.*;
2
3  public class OpenGaussJDBC {
4      private static final String DRIVER = "org.opengauss.Driver";
5      private static final String URL =
   "jdbc:opengauss://43.138.80.125:25432/postgres";
6      private static final String USERNAME = "postgres";
7      private static final String PASSWORD = "osapp_openGauss@123";
8
9      static {
10         try {
11             Class.forName(DRIVER);
```

```
12          } catch (ClassNotFoundException e) {
13              throw new RuntimeException("Failed to load JDBC driver", e);
14          }
15      }
16
17      public static Connection getConnection() throws SQLException {
18          return DriverManager.getConnection(URL, USERNAME, PASSWORD);
19      }
20
21      public static void executeByStatement(String sql) throws SQLException {
22          Connection conn = getConnection();
23          Statement statement = conn.createStatement();
24          statement.execute(sql);
25      }
26
27      public static void executeByPreparedStatement(String sql) throws
    SQLException {
28          Connection conn = getConnection();
29          PreparedStatement preparedStatement = conn.prepareStatement(sql);
30          preparedStatement.execute();
31      }
32
33      public static ResultSet executeQuery(String sql) throws SQLException {
34          Connection conn = getConnection();
35          PreparedStatement preparedStatement = conn.prepareStatement(sql);
36          return preparedStatement.executeQuery();
37      }
38
39      public static void printResultSet(ResultSet resultSet) throws SQLException
    {
40          ResultSetMetaData metaData = resultSet.getMetaData();
41          int columnCount = metaData.getColumnCount();
42          while (resultSet.next()) {
43              for (int i = 1; i <= columnCount; i++) {
44                  String columnValue = resultSet.getString(i);
45                  String columnName = metaData.getColumnName(i);
46                  System.out.print(columnName + ": " + columnValue + "  ");
47              }
48              System.out.println();
49          }
50      }
51
52      public static void closeResources(Connection conn, Statement statement,
    ResultSet resultSet) throws SQLException {
53          if (resultSet != null) resultSet.close();
54          if (statement != null) statement.close();
55          if (conn != null) conn.close();
```

```
56      }
57  }
```

方言包开发

Hibernate Dialect

方言（Dialect）在 Hibernate 中的主要作用是**告诉 Hibernate 如何将 HQL（Hibernate Query Language）或标准 SQL 转换为特定数据库的 SQL 语句**。由于不同数据库的 SQL 语法和数据类型有所不同，Hibernate 通过方言实现对多种数据库的兼容性。每个方言定义了数据类型、函数、特性等，并对数据库的特定行为提供支持。

Hibernate 运行流程和机制：

1. **配置**：Hibernate 通过 `hibernate.cfg.xml` 或者 `persistence.xml` 文件加载配置，其中包括数据库连接信息、映射文件或实体类的路径等。
2. **初始化**：根据配置，Hibernate 创建 `SessionFactory` 。

> `SessionFactory` 负责维护 Hibernate 与数据库之间的会话（`Session`）。

3. **生成 SQL 语句**：Hibernate 根据映射文件或注解，读取实体类的结构，包括类名、属性、关系映射等。
4. 会话管理：应用程序通过 `SessionFactory` 创建 `Session` 对象。

> `Session` 负责管理持久化操作，如保存、更新、删除实体，并维护 Hibernate 缓存。

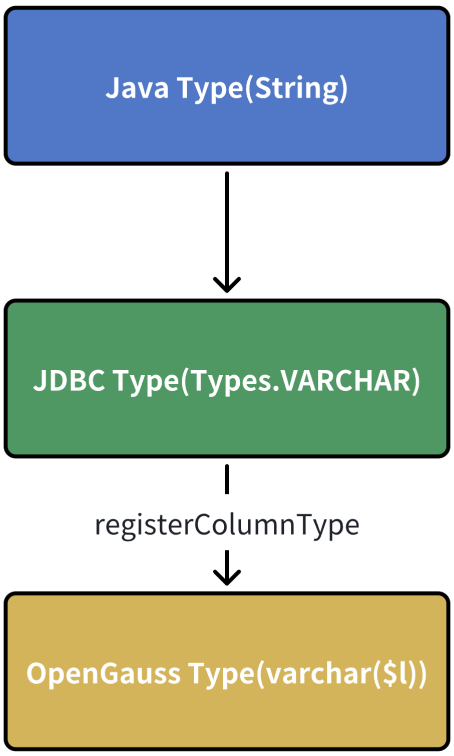
5. **事务控制**：Hibernate 的事务管理器会控制数据库事务的边界。方言也会影响事务的具体实现细节，例如使用哪种方式锁定行或如何处理批量操作。
6. **执行 SQL**：在 `Session` 中操作实体对象时，Hibernate 会根据方言生成合适的 SQL 语句，并通过 JDBC 执行这些 SQL 语句。
7. **返回结果**：当查询返回结果时，Hibernate 会将结果集通过方言和实体映射关系，转换为相应的 Java 对象。

Dialect

type mapping

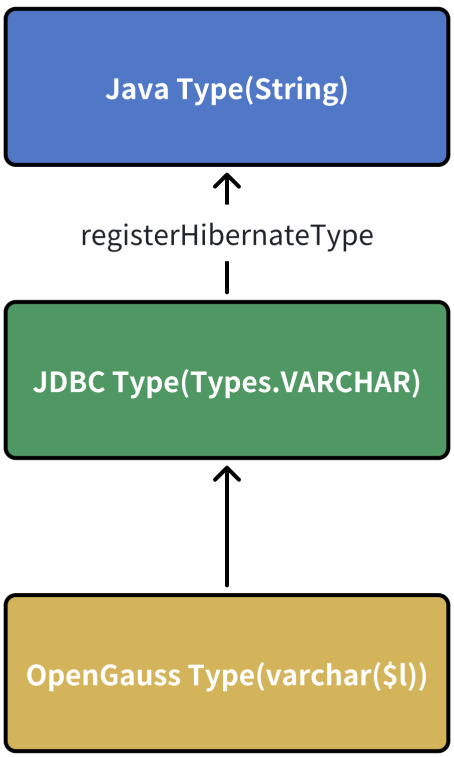
`registerColumnType`：将 **JDBC 类型**映射为**数据库的 SQL 列类型**，用于生成正确的 SQL 语句，确保 Java 类型在数据库中表示为合适的列类型。

例：`registerColumnType(Types.VARCHAR, "varchar($l)");`



`registerHibernateType`：将 **JDBC 类型**映射为 **Hibernate 类型**，用于从数据库读取数据时，正确地将数据库类型转换为 Java 类型，确保数据的完整性和一致性。

例：`registerHibernateType(Types.VARCHAR, StandardBasicTypes.STRING.getName());`



使用 DatabaseMetaData 动态获取类型信息，查看 OpenGauss 支持的 JDBC Type：

```
public static void collectionDeduplication() throws SQLException {
    Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
    DatabaseMetaData metaData = connection.getMetaData();
    ResultSet typeInfo = metaData.getTypeInfo();
    Set<Integer> jdbcTypes = new HashSet<>();
    while (typeInfo.next()) {
        int dataType = typeInfo.getInt("DATA_TYPE");
        jdbcTypes.add(dataType);
    }
    for (Integer jdbcType : jdbcTypes) {
        System.out.println("JDBC Type: " + jdbcType);
    }
}
```

```
JDBC Type: -2
JDBC Type: 1
JDBC Type: 2
JDBC Type: -5
JDBC Type: 4
JDBC Type: 5
JDBC Type: -6
JDBC Type: -7
JDBC Type: 7
JDBC Type: 8
JDBC Type: 12
JDBC Type: 2001
JDBC Type: 2002
JDBC Type: 2003
JDBC Type: 2004
JDBC Type: 2005
JDBC Type: 1111
JDBC Type: 2009
JDBC Type: 91
JDBC Type: 92
JDBC Type: 2012
JDBC Type: 93
```

```
public static void JDBCTypesMappingSQLTypes() throws SQLException {
    Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
    DatabaseMetaData metaData = connection.getMetaData();
    ResultSet typeInfo = metaData.getTypeInfo();
    Map<Integer, Set<String>> jdbcTypeToSqlTypes = new HashMap<>();
    while (typeInfo.next()) {
```

```
            String typeName = typeInfo.getString("TYPE_NAME");
            int dataType = typeInfo.getInt("DATA_TYPE");
            jdbcTypeToSqlTypes.computeIfAbsent(dataType, k -> new HashSet<>
()).add(typeName);
        }
        for (Map.Entry<Integer, Set<String>> entry :
jdbcTypeToSqlTypes.entrySet()) {
            int jdbcType = entry.getKey();
            Set<String> sqlTypes = entry.getValue();
            System.out.println("JDBC Type: " + jdbcType + ", SQL Types: " +
sqlTypes);
        }
}
```

JDBC Code	JDBC Types
1	java.sql.Types.CHAR
-2	java.sql.Types.BINARY
2	java.sql.Types.NUMERIC
4	java.sql.Types.INTEGER
-5	java.sql.Types.BIGINT
-6	java.sql.Types.TINYINT
5	java.sql.Types.SMALLINT
-7	java.sql.Types.BIT
7	java.sql.Type.REAL
8	java.sql.Types.DOUBLE

12	java.sql.Types.VARCHAR
2001	java.sql.Types.DISTINCT
2002	java.sql.Types.STRUCT
2003	java.sql.Types.ARRAY
2004	java.sql.Types.BLOB
2005	java.sql.Types.CLOB
1111	java.sql.Types.OTHER
2009	java.sql.Types.SQLXML
91	java.sql.Types.DATE
2012	java.sql.Types.REF_CURSOR
92	java.sql.Types.TIME
93	java.sql.Types.TIMESTAMP

布尔类型

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
-7	java.lang.Boolean	java.sql.Types.BIT	bit
16	java.lang.Boolean	java.sql.Types.BOOLEAN	bool

数值类型

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
-6	java.lang.Byte	java.sql.Types.TINYINT	int2
5	java.lang.Short	java.sql.Types.SMALLINT	int2
4	java.lang.Character.Integer	java.sql.Types.INTEGER	int4
-5	java.lang.Long	java.sql.Types.BIGINT	int8
7	java.lang.Float	java.sql.Type.REAL	float4
6	java.lang.Float	java.sql.Types.FLOAT	float8

8	java.lang.Double	java.sql.Types.DOUBLE	float8
3	java.math.BigDecimal	java.sql.Types.DECIMAL	decimal($p,$s)
2	java.math.BigDecimal	java.sql.Types.NUMERIC	numeric($p, $s)

字符/字符串类型

supportsNationalizedTypes()
作用: 检查是否支持国定类型（如 `NCHAR`、`NVARCHAR`）。
默认返回值: `true`

NVARCHAR2：OpenGauss 支持 `NVARCHAR2` 数据类型，用于存储 Unicode 字符串。

NCHAR：OpenGauss 支持 `NCHAR` 数据类型，用于存储固定长度的 Unicode 字符串。

Hibernate 将 `@Nationalized` 注解的字段映射为国家化类型：

`String` 类型的字段在 OpenGauss 中映射为 `nvarchar2`。

`Character` 类型的字段在 OpenGauss 中映射为 `nchar`。

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
1	java.lang.Character	java.sql.Types.CHAR	char(1)
12	java.lang.String	java.sql.Types.VARCHAR	varchar($l)
-1	java.lang.String	java.sql.Types.LONGVARCHAR	text
-15	java.lang.Character	Types.NCHAR	nchar($l)
-9	java.lang.String	Types.NVARCHAR	nvarchar2($l)

日期和时间类型

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
91	java.sql.Date	java.sql.Types.DATE	date
92	java.sql.Time	java.sql.Types.TIME	time
93	java.sql.Timestamp	java.sql.Types.TIMESTAMP	timestamp
2013	java.sql.Timestamp	java.sql.Types.TIME_WITH_TIMEZONE	timetzField
2014	java.sql.Timestamp		timestamptzField

		java.sql.Types.TIMESTAMP_WITH_TIMEZONE	

二进制类型

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
-2	byte[]	java.sql.Types.BINARY	bytea
-3	byte[]	java.sql.Types.VARBINARY	bytea
-4	byte[]	java.sql.Types.LONGVARBINARY	bytea

大对象类型

openGauss 提供了对 `BLOB` 和 `CLOB` 类型的原生支持：

```
@Override
public SqlTypeDescriptor getSqlTypeDescriptorOverride(int sqlCode) {
    SqlTypeDescriptor descriptor;
    switch (sqlCode) {
        case Types.BLOB: {
            descriptor = BlobTypeDescriptor.BLOB_BINDING;
            break;
        }
        case Types.CLOB: {
            descriptor = ClobTypeDescriptor.CLOB_BINDING;
            break;
        }
        default: {
            descriptor = super.getSqlTypeDescriptorOverride(sqlCode);
            break;
        }
    }
    return descriptor;
}
```

NCLOB：OpenGauss **不支持** `NCLOB` 数据类型。

`Clob` 类型的字段将映射为 `NCLOB`。但由于 OpenGauss 不支持 `NCLOB`，需要进行特殊处理。

```
registerColumnType( Types.NCLOB, "text" ); // 或者使用 "clob"
```

Hibernate 将 `@Nationalized` 注解的字段映射为国家化类型：

`NCLOB` 类型的字段在 OpenGauss 中映射为 `text（或 clob）`。

JDBC Code	Java Type	JDBC Types	OpenGauss 类型
2004	java.sql.Blob	java.sql.Types.BLOB	blob
2005	java.sql.Clob	java.sql.Types.CLOB	clob
2011	java.sql.NClob	java.sql.Types.NCLOB	text

json/jsonb 类型

openGauss 提供了对 json 类型的原生支持，但是 Hibernate 4/5 不支持任何 json 映射，为了使用 openGauss `JSON` 和 `JSONB` 类型，通过自定义 `UserType` 的实现来将 Java 对象与数据库中的 `JSON` 和 `JSONB` 类型相映射。

```
1  public class JsonType implements UserType {
2
3      private static final ObjectMapper objectMapper = new ObjectMapper();
4
5      @Override
6      public int[] sqlTypes() {
7          return new int[] { Types.JAVA_OBJECT};
8      }
9
10     @Override
11     public Class<Object> returnedClass() {
12         return Object.class;
13     }
14
15     @Override
16     public Object nullSafeGet(ResultSet rs, String[] names,
   SharedSessionContractImplementor session, Object owner) throws
   HibernateException, SQLException {
17         String json = rs.getString(names[0]);
18         if (json != null) {
19             try {
20                 return objectMapper.readValue(json, returnedClass());
21             } catch (Exception e) {
22                 throw new HibernateException("Failed to convert JSON to
   Object: " + json, e);
```

```
            }
        }
        return null;
    }

    @Override
    public void nullSafeSet(PreparedStatement st, Object value, int index,
SharedSessionContractImplementor session) throws HibernateException,
SQLException {
        if (value == null) {
            st.setNull(index, Types.OTHER);
        } else {
            try {
                String json = objectMapper.writeValueAsString(value);
                st.setObject(index, json, Types.OTHER);
            } catch (JsonProcessingException e) {
                throw new HibernateException("Failed to convert Object to
JSON: " + value, e);
            }
        }
    }

    // other method
```

function mapping

indentity

- 使用 `IDENTITY` 列

```
CREATE TABLE identity_test (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY,
    name VARCHAR(100)
);
```

查询	消息	查询时间	获取时间
CREATE TABLE identity_test (id BIGINT GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(100))	ERROR: syntax error at or near "BY" LINE 2: id BIGINT GENERATED BY DEFAULT AS IDENTITY, ^	0.074s	0.000s

执行失败，说明 OpenGauss 不支持 `IDENTITY` 列。

- 使用 `SEQUENCE`

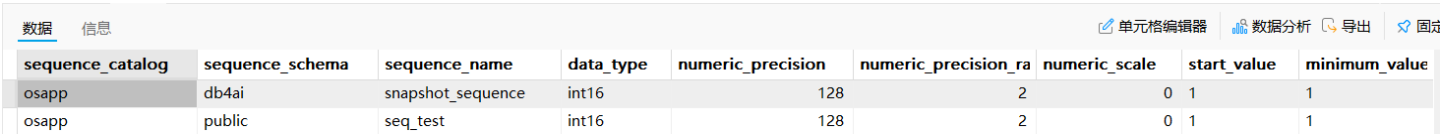
```
CREATE SEQUENCE seq_test START 1;
```

```
2 CREATE TABLE sequence_test (
3     id BIGINT DEFAULT nextval('seq_test'),
4     name VARCHAR(100)
5 );
```

```
1 INSERT INTO sequence_test (name) VALUES ('test_name1');
2 INSERT INTO sequence_test (name) VALUES ('test_name2');
3 INSERT INTO sequence_test (name) VALUES ('test_name3');
```

执行成功，OpenGauss 可以使用序列生成主键。

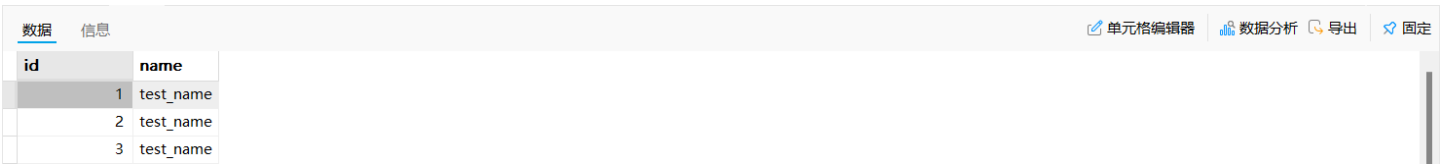
```
1 SELECT * FROM information_schema.sequences;
```



数据 信息 单元格编辑器 数据分析 导出 固定

sequence_catalog	sequence_schema	sequence_name	data_type	numeric_precision	numeric_precision_ra	numeric_scale	start_value	minimum_value
osapp	db4ai	snapshot_sequence	int16	128	2	0	1	1
osapp	public	seq_test	int16	128	2	0	1	1

```
1 SELECT * FROM sequence_test;
```



数据 信息 单元格编辑器 数据分析 导出 固定

id	name
1	test_name
2	test_name
3	test_name

```
1 DROP TABLE IF EXISTS sequence_test;
2 DROP SEQUENCE IF EXISTS seq_test;
```

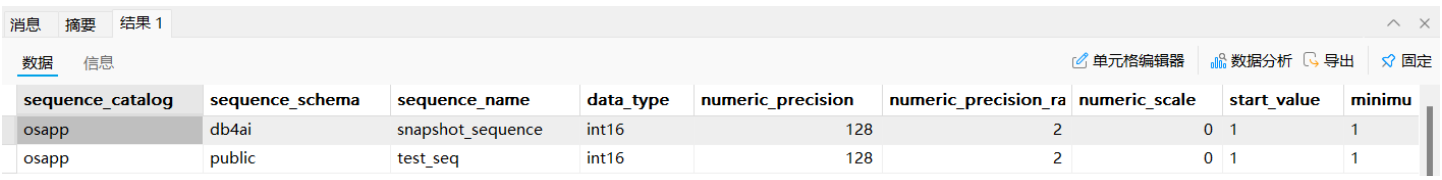
sequence

方法名称	功能描述	Dialect(defaule)	Dialect(OpenGauss)
supportsSequences()	是否支持序列	false	重载
supportsPooledSequences()	是否支持“池化”序列	false	重载
		MappingException	重载

getSequenceNextValString(String)	获取下一个序列值的 SQL 语句		
getSelectSequenceNextValString(String)	获取序列的下一个值的 SQL 表达式	MappingException	重载
getCreateSequenceStrings(String)	创建序列的 SQL 语句	Depending on getCreateSequenceString	继承
getCreateSequenceString(String)	创建序列的 SQL 语句	MappingException	重载
getCreateSequenceString(String, int, int)	创建初始值和增量值的序列的 SQL 语句	Depending on supportsPooledSequences	重载
getDropSequenceStrings(String)	删除序列的 SQL 语句	Depending on getDropSequenceString	继承
getDropSequenceString(String)	删除序列的 SQL 语句	MappingException	重载
getQuerySequencesString()	获取所有序列名称的 SQL 查询	null	重载
getSequenceInformationExtractor()	序列信息提取器	Depending on getQuerySequencesString	继承

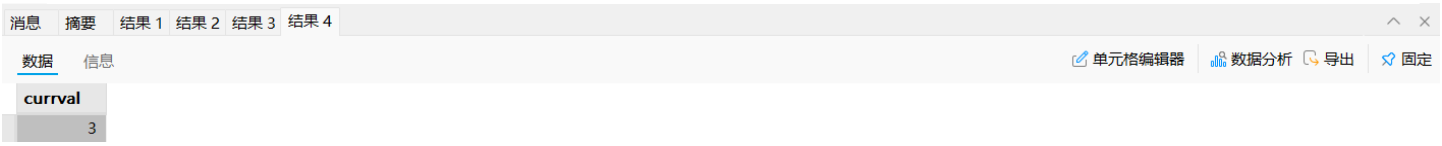
- 创建序列：

```
1 CREATE SEQUENCE test_seq;
2 SELECT * FROM information_schema.sequences;
```



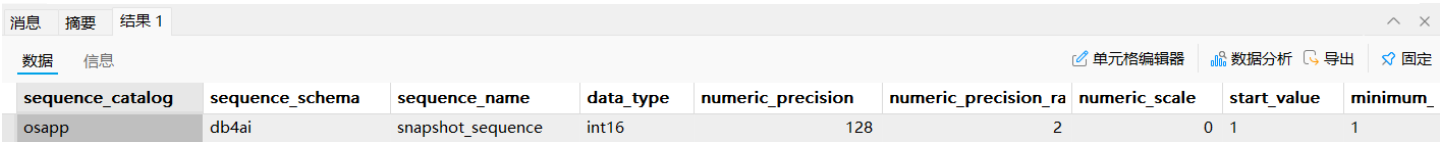
- 使用序列：

```
1 SELECT nextval('test_seq');
2 SELECT nextval('test_seq');
3 SELECT nextval('test_seq');
4 SELECT currval('test_seq');
```



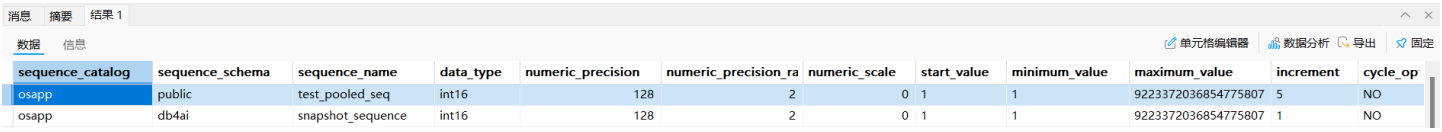
- 删除序列：

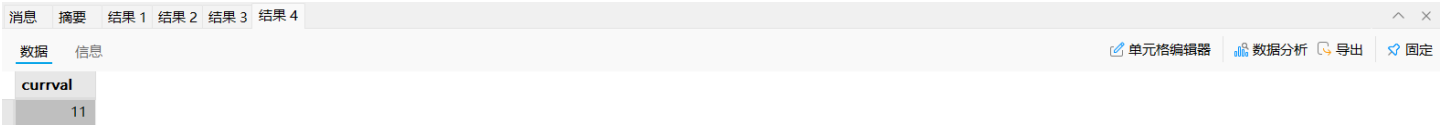
```
1 DROP SEQUENCE IF EXISTS test_seq;
2 SELECT * FROM information_schema.sequences;
```

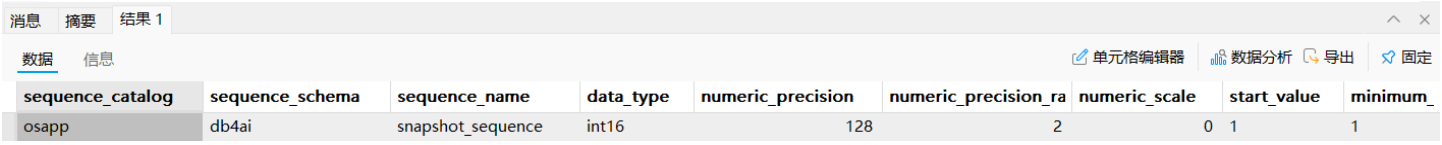


- 池化序列（同上）：

```
1  CREATE SEQUENCE test_pooled_seq START WITH 1 INCREMENT BY 5;
2  SELECT * FROM information_schema.sequences;
3
4  SELECT nextval('test_pooled_seq'); // 1
5  SELECT nextval('test_pooled_seq'); // 1 + 5 = 6
6  SELECT nextval('test_pooled_seq'); // 6 + 5 = 11
7  SELECT currval('test_pooled_seq');
8
9  DROP SEQUENCE IF EXISTS test_pooled_seq;
10 SELECT * FROM information_schema.sequences;
```







> - 查看所有序列

limit/offset

方法名称	功能描述	Dialect(defaule)	Dialect(OpenGauss)

getLimitHandler()	提供分页支持	LegacyLimitHandler(this)	重载

```
private static final AbstractLimitHandler LIMIT_HANDLER = new
AbstractLimitHandler() {
    @Override
    public String processSql(String sql, RowSelection selection) {
        final boolean hasOffset = LimitHelper.hasFirstRow(selection);
        return sql + (hasOffset ? " limit ? offset ?" : " limit ?");
    }

    @Override
    public boolean supportsLimit() {
        return true;
    }

    @Override
    public boolean bindLimitParametersInReverseOrder() {
        return true;
    }
};

@Override
public LimitHandler getLimitHandler() {
    return LIMIT_HANDLER;
}
```

lock acquisition

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
supportsLockTimeouts()	指示方言是否支持在获取锁时指定锁超时。	true	继承
isLockTimeoutParameterized()	指定锁超时是否作为参数呈现在 SQL 字符串中。	false	继承
getLockingStrategy(Lockable, LockMode)	返回适用于指定 `LockMode` 的数据库级锁的 `LockingStrategy` 实例。	LockingStrategy	继承
getForUpdateString()	返回默认的 `FOR UPDATE` SQL 片段。	`" for update"`	继承

getForUpdateString(String)	针对特定的列别名，返回 `FOR UPDATE` SQL 片段。	`" for update of aliases"`	重载
getForUpdateString(LockMode, int)	核心方法，给定 `LockMode` 和超时，返回适当的锁定 SQL 片段。		继承
getForUpdateString(LockMode)	根据给定的 `LockMode` 确定适当的 `FOR UPDATE` SQL 片段，假设默认的超时。		继承
getForUpdateString(LockOptions)	根据提供的 `LockOptions`（包含 `LockMode` 和超时设置），确定适当的 `FOR UPDATE` SQL 片段。		继承
getForUpdateString(String, LockOptions)	针对特定的列别名和 `LockOptions`，确定适当的 `FOR UPDATE` SQL 片段。		重载
getWriteLockString(int)	提供获取 WRITE 锁的 SQL 片段，考虑指定的超时。		重载
getWriteLockString(String, int)			
getReadLockString(int)	提供获取 READ 锁的 SQL 片段，考虑指定的超时。		重载
getReadLockString(String, int)			
forUpdateOfColumns()	指示方言是否支持 `FOR UPDATE OF` 语法。	false	继承
supportsOuterJoinForUpdate()	指定是否可以对外部连接的行使用 `FOR UPDATE`。	true	继承
getForUpdateNowaitString()	返回适用于该方言的 `FOR UPDATE NOWAIT` SQL 子句。		重载
getForUpdateSkipLockedString()	返回适用于该方言的 `FOR UPDATE SKIP LOCKED` SQL 子句。		重载
getForUpdateNowaitString(String)	根据给定的列别名，返回适用于该方言的 `FOR UPDATE OF column_list NOWAIT` SQL 子句。		重载

getForUpdateSkipLockedString(String)	根据给定的列别名，返回适用于该方言的 `FOR UPDATE OF column_list SKIP LOCKED` SQL 子句。		重载
appendLockHint(LockMod, String)	如果方言支持，则将锁提示直接附加到 `FROM` 子句中的表名。		继承
appendLockHint(LockOptions, String)			继承
applyLocksToSql(String, LockOptions, Map<String, String[]>)	根据提供的锁选项和关键列修改 SQL，以应用适当的锁。		继承

callable statement

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
registerResultSetOutParameter(CallableStatement, int)	注册一个返回 `ResultSet` 的 OUT 参数，使用参数位置。	throw new UnsupportedOperationException	重载
registerResultSetOutParameter(CallableStatement, String)	注册一个返回 `ResultSet` 的 OUT 参数，使用名称位置。	throw new UnsupportedOperationException	继承
getResultSet(CallableStatement)	从先前注册的 OUT 参数（按位置）中获取 `ResultSet`，默认为 1。	throw new UnsupportedOperationException	重载
getResultSet(CallableStatement, int)	从先前注册的 OUT 参数（按位置）中获取 `ResultSet`。	throw new UnsupportedOperationException	重载
getResultSet(CallableStatement, String)	从先前注册的 OUT 参数（按名称）中获取 `ResultSet`。	throw new UnsupportedOperationException	继承

```sql
CREATE TABLE test_table (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    amount NUMERIC(10,2)
);

INSERT INTO test_table (id, name, amount) VALUES
(1, 'Alice', 1000.00),
(2, 'Bob', 2000.00),
(3, 'Charlie', 3000.00);

CREATE OR REPLACE PROCEDURE get_test_table_cursor(out_cursor OUT refcursor) AS
BEGIN
    OPEN out_cursor FOR SELECT * FROM test_table;
END;
```

```java
// public ResultSet getResultSet(CallableStatement statement, int position) throws SQLException
@Test
public void registerResultSetOutParameterByPositionTest() {
    session.doWork(connection -> {
        CallableStatement callableStatement = connection.prepareCall("{CALL get_test_table_cursor(?)}");
        callableStatement.registerOutParameter(1, Types.REF_CURSOR);
        callableStatement.execute();
        ResultSet rs = (ResultSet) callableStatement.getObject(1);
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            BigDecimal amount = rs.getBigDecimal("amount");
            System.out.println("ID: " + id + ", Name: " + name + ", Amount: " + amount);
        }
        rs.close();
        callableStatement.close();
    });
}

// public int registerResultSetOutParameter(CallableStatement statement, String name) throws SQLException
@Test
public void egisterResultSetOutParameterByNameTest() {
    session.doWork(connection -> {
```

```
        CallableStatement callableStatement = connection.prepareCall("{CALL get_test_table_cursor(?)}");
        callableStatement.registerOutParameter("out_cursor", Types.REF_CURSOR);
        callableStatement.execute();
        ResultSet rs = (ResultSet) callableStatement.getObject(1);
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            BigDecimal amount = rs.getBigDecimal("amount");
            System.out.println("ID: " + id + ", Name: " + name + ", Amount: " + amount);
        }
        rs.close();
        callableStatement.close();
    });
}

@Test
public void getResultSetTest() {
    session.doWork(connection -> {
        CallableStatement callableStatement = connection.prepareCall("{CALL get_test_table_cursor(?)}");
        callableStatement.registerOutParameter(1, Types.REF_CURSOR);
        callableStatement.execute();
        // public ResultSet getResultSet(CallableStatement statement, int position) throws SQLException
        ResultSet rsByPosition = (ResultSet) callableStatement.getObject(1);
        // public ResultSet getResultSet(CallableStatement statement, String name)
        // ResultSet rsByName = (ResultSet) callableStatement.getObject("out_cursor");
        while (rsByPosition.next()) {
            int id = rsByPosition.getInt("id");
            String name = rsByPosition.getString("name");
            BigDecimal amount = rsByPosition.getBigDecimal("amount");
            System.out.println("ID: " + id + ", Name: " + name + ", Amount: " + amount);
        }
        rsByPosition.close();
        callableStatement.close();
    });
}
```

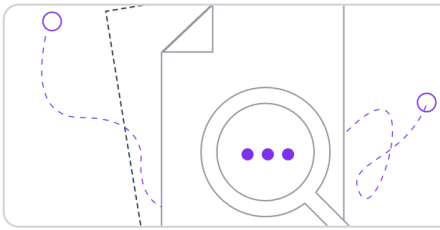
current timestamp

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
supportsCurrentTimestampSelection()	判断数据库是否支持检索当前时间戳。	false	重载
isCurrentTimestampSelectStringCallable()	判断检索当前时间戳的 SQL 语句是否需要使用 JDBC 转义语法（callable）。	throw new UnsupportedOperationException	重载
getCurrentTimestampSelectString()	返回用于获取当前时间戳的 SQL 语句。	throw new UnsupportedOperationException	重载
getCurrentTimestampSQLFunctionName()	返回数据库特定的当前时间戳 SQL 函数名。	`"current_timestamp"`	继承

exception

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
buildSQLExceptionConverter()	构建 `SQLExceptionConverter` 实例，用于将 `SQLException` 转换为 Hibernate 的 `JDBCException` 层次结构。	null	继承
buildSQLExceptionConversionDelegate()	构建 `SQLExceptionConversionDelegate` 实例，用于解释数据库特定的错误码或 SQLState 码。	null	重载
getViolatedConstraintNameExtracter()	返回 `ViolatedConstraintNameExtracter` 实例，用于从 `SQLException` 中提取违反的约束（如主键、唯一键、外键等）的名称。	EXTRACTER	重载

OpenGauss 的错误码参考：



https://docs-opengauss.osinfra.cn/zh/docs/5.0.0/docs/DatabaseReference/SQL…

Sql标准错误码说明

openGauss是一个高性能、高安全、高可用、高智能的企业级开源关系数据库。
openGauss也是一个鼓励社区贡献和协作的开源数据库平台。

错误码结构是由前两位字母或数字表示错误类别，后三位数字或字母表示具体错误。

处理方法：根据错误码的前两位字符对异常进行分类处理，而不需要了解所有具体的错误码，可以有效地覆盖大部分异常情况。对于无法分类或未知的错误码，返回通用的 `JDBCException`，或者让其他异常转换委托来处理。

```
-- 用于测试唯一约束
CREATE TABLE test_unique_constraint (
    id SERIAL PRIMARY KEY,
    unique_column VARCHAR(50) UNIQUE
);

-- 用于测试非空约束
CREATE TABLE test_not_null_constraint (
    id SERIAL PRIMARY KEY,
    not_null_column VARCHAR(50) NOT NULL
);

-- 用于测试 CHECK 约束
CREATE TABLE test_check_constraint (
    id SERIAL PRIMARY KEY,
    check_column INT CHECK (check_column > 0)
);

-- 创建引用表
CREATE TABLE test_references (
    id SERIAL PRIMARY KEY
);

-- 用于测试外键约束
CREATE TABLE test_foreign_key_constraint (
    id SERIAL PRIMARY KEY,
    foreign_key_column INT,
    CONSTRAINT fk_test FOREIGN KEY (foreign_key_column) REFERENCES
test_references(id)
);

```

OpenGauss 的异常消息格式：

- **23505**：违反唯一约束（UNIQUE_VIOLATION）

```
ERROR: duplicate key value violates unique constraint "constraint_name"
详细: Key (column_name)=(value) already exists.
```

- **23503**：违反外键约束（FOREIGN_KEY_VIOLATION）

```
ERROR: insert or update on table "table_name" violates foreign key constraint
"constraint_name"
详细: Key (column_name)=(value) is not present in table "referenced_table".
```

- **23502**：违反非空约束（NOT_NULL_VIOLATION）

```
ERROR: null value in column "column_name" violates not-null constraint
详细: Failing row contains (values...).
```

- **23514**：违反 CHECK 约束（CHECK_VIOLATION）

```
ERROR: new row for relation "table_name" violates check constraint
"constraint_name"
详细: N/A
```

union subclass

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
getSelectClauseNullString(int sqlType)	返回一个在 `SELECT` 子句中表示指定 SQL 类型的 `NULL` 值的字符串片段。	`"null"`	继承
supportsUnionAll()	指示数据库是否支持 `UNION ALL` 操作。	false	重载

```
SELECT 'value1' AS column1, 123 AS column2
UNION ALL
SELECT NULL AS column1, NULL AS column2;
```

```
4
5 SELECT 'value1' AS column1, 123 AS column2
6 UNION ALL
7 SELECT null::varchar AS column1, null::integer AS column2;
```

miscellaneous

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
createOuterJoinFragment()	创建 `JoinFragment` 策略，负责处理该方言在连接（join）处理方面的变体。	new ANSIJoinFragment()	继承
createCaseFragment()	创建 `CaseFragment` 策略，负责处理该方言在 `CASE` 语句处理方面的变体。	new ANSICaseFragment()	继承
getNoColumnsInsertString()	返回在未指定任何列值的情况下插入一行时使用的 SQL 片段。	"values ( )"	重载
supportsNoColumnsInsert()	指示数据库是否支持不指定任何列的 `INSERT` 语句。	true	继承
getLowercaseFunction()	返回将字符串转换为小写的 SQL 函数名称。	"lower"	继承
getCaseInsensitiveLike()	返回用于不区分大小写的 `LIKE` 比较的 SQL 操作符或函数。	"like"	重载
supportsCaseInsensitiveLike()	指示数据库是否支持不区分大小写的 `LIKE` 操作。	false	重载
transformSelectString(String)	允许在 SQL 语句发送到数据库之前对其进行修改。可用于在特定方言中添加提示或修改查询。	select	继承
getMaxAliasLength()	指定数据库中允许的 SQL 别名的最大长度。	10	继承
toBooleanValueString(boolean)	返回布尔值（`true` 或 `false`）对应的 SQL 字面值。	bool ? "1" : "0"	重载

keyword

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
registerKeyword(String)	注册一个数据库的保留字，将保留字转换为小写并添加到 `sqlKeywords` 集合中。	sqlKeywords.add(word.toLowerCase(Locale.ROOT));	继承
buildIdentifierHelper(IdentifierHelperBuilder , DatabaseMetaData)	构建 `IdentifierHelper` ，用于处理标识符的转换，包括大小写和引用。	IdentifierHelper	重载

buildIdentifierHelper()：

```
1  public IdentifierHelper buildIdentifierHelper(
2          IdentifierHelperBuilder builder,
3          DatabaseMetaData dbMetaData) throws SQLException {
4      builder.applyIdentifierCasing( dbMetaData );
5
6      builder.applyReservedWords( dbMetaData );
7      builder.applyReservedWords( AnsiSqlKeywords.INSTANCE.sql2003() );
8      builder.applyReservedWords( sqlKeywords );
9
10     builder.setNameQualifierSupport( getNameQualifierSupport() );
11
12     return builder.build();
13 }
14
```

1. **应用标识符大小写**：根据数据库元数据，确定标识符的大小写策略（大写、小写、混合）。
2. **应用保留字**：从数据库元数据、ANSI SQL 标准和 Dialect 注册的关键词中收集保留字。
3. **设置名称限定支持**：指定数据库是否支持限定符（如模式、目录）。
4. **构建 `IdentifierHelper`**：返回构建好的 `IdentifierHelper` 实例。

IdentifierHelperBuilder

类字段

```
1  private final JdbcEnvironment jdbcEnvironment;
2
3  private NameQualifierSupport nameQualifierSupport = NameQualifierSupport.BOTH;
4
5  private final TreeSet<String> reservedWords = new TreeSet<>(
   String.CASE_INSENSITIVE_ORDER );
6
7  private boolean globallyQuoteIdentifiers = false;
8  private boolean skipGlobalQuotingForColumnDefinitions = false;
9  private boolean autoQuoteKeywords = true;
10 private IdentifierCaseStrategy unquotedCaseStrategy =
   IdentifierCaseStrategy.UPPER;
11 private IdentifierCaseStrategy quotedCaseStrategy =
   IdentifierCaseStrategy.MIXED;
12
```

- **`jdbcEnvironment`**： 提供对 JDBC 相关设置和元数据的访问。
- **`nameQualifierSupport`**： 指示名称限定符支持的级别（例如，支持 schema、catalog、两者或都不支持）。
- **`reservedWords`**： 一个包含 SQL 保留字的集合，这些词不应作为未引用的标识符使用。
- **`globallyQuoteIdentifiers`**： 如果为 `true`，则所有标识符都将被引用。
- **`skipGlobalQuotingForColumnDefinitions`**： 如果为 `true`，即使 `globallyQuoteIdentifiers` 为 `true`，列定义也不会被全局引用。
- **`autoQuoteKeywords`**： 如果为 `true`，则将自动引用作为保留字的标识符。
- **`unquotedCaseStrategy`**： 未引用标识符的大小写策略（大写、小写、混合）。
- **`quotedCaseStrategy`**： 引用标识符的大小写策略。

构造函数

提供使用所需的 `JdbcEnvironment` 创建 `IdentifierHelperBuilder` 实例的方法。

```
1  public static IdentifierHelperBuilder from(JdbcEnvironment jdbcEnvironment) {
2      return new IdentifierHelperBuilder( jdbcEnvironment );
3  }
4
5  private IdentifierHelperBuilder(JdbcEnvironment jdbcEnvironment) {
6      this.jdbcEnvironment = jdbcEnvironment;
7  }
8
```

应用保留字

将 `DatabaseMetaData` 中的 SQL 保留字添加到 `reservedWords` 集合中。

```
public void applyReservedWords(DatabaseMetaData metaData) throws SQLException {
    if ( metaData == null ) {
        return;
    }
    // 如果未启用自动引用关键字，则跳过加载关键词
    if ( !autoQuoteKeywords ) {
        return;
    }
    this.reservedWords.addAll( parseKeywords( metaData.getSQLKeywords() ) );
}
```

解析关键词

将逗号分隔的关键词字符串解析为列表。

```
private static List<String> parseKeywords(String extraKeywordsString) {
    return StringHelper.parseCommaSeparatedString( extraKeywordsString );
}

```

应用标识符大小写

根据元数据确定未引用和引用标识符的大小写策略。

- 检查元数据方法，如 `storesLowerCaseIdentifiers()` ，以确定数据库如何处理标识符大小写。
- 相应地设置 `unquotedCaseStrategy` 和 `quotedCaseStrategy` 。

```
public void applyIdentifierCasing(DatabaseMetaData metaData) throws
SQLException {
    if ( metaData == null ) {
        return;
    }

    final int unquotedAffirmatives = ArrayHelper.countTrue(
            metaData.storesLowerCaseIdentifiers(),
            metaData.storesUpperCaseIdentifiers(),
            metaData.storesMixedCaseIdentifiers()
    );

```

```
12      if ( unquotedAffirmatives == 0 ) {
13          log.debug( "JDBC driver metadata reported database stores unquoted
    identifiers in neither upper, lower nor mixed case" );
14      }
15      else {
16          // 如果多个为 true，则存在歧义，但我们仍将继续
17          if ( unquotedAffirmatives > 1 ) {
18              log.debug( "JDBC driver metadata reported database stores unquoted
    identifiers in more than one case" );
19          }
20
21          if ( metaData.storesUpperCaseIdentifiers() ) {
22              this.unquotedCaseStrategy = IdentifierCaseStrategy.UPPER;
23          }
24          else if ( metaData.storesLowerCaseIdentifiers() ) {
25              this.unquotedCaseStrategy = IdentifierCaseStrategy.LOWER;
26          }
27          else {
28              this.unquotedCaseStrategy = IdentifierCaseStrategy.MIXED;
29          }
30      }
31
32      // 对于引用的标识符，采用类似的逻辑
33      final int quotedAffirmatives = ArrayHelper.countTrue(
34              metaData.storesLowerCaseQuotedIdentifiers(),
35              metaData.storesUpperCaseQuotedIdentifiers(),
36              metaData.storesMixedCaseQuotedIdentifiers()
37      );
38
39      if ( quotedAffirmatives == 0 ) {
40          log.debug( "JDBC driver metadata reported database stores quoted
    identifiers in neither upper, lower nor mixed case" );
41      }
42      else {
43          if ( quotedAffirmatives > 1 ) {
44              log.debug( "JDBC driver metadata reported database stores quoted
    identifiers in more than one case" );
45          }
46
47          if ( metaData.storesMixedCaseQuotedIdentifiers() ) {
48              this.quotedCaseStrategy = IdentifierCaseStrategy.MIXED;
49          }
50          else if ( metaData.storesLowerCaseQuotedIdentifiers() ) {
51              this.quotedCaseStrategy = IdentifierCaseStrategy.LOWER;
52          }
53          else {
54              this.quotedCaseStrategy = IdentifierCaseStrategy.UPPER;
```

```
        }
    }
}
```

Setter 和 Getter 方法

用于设置和获取配置选项的各种方法：

- `setGloballyQuoteIdentifiers`，`isGloballyQuoteIdentifiers`
- `setSkipGlobalQuotingForColumnDefinitions`，`isSkipGlobalQuotingForColumnDefinitions`
- `setAutoQuoteKeywords`
- `setNameQualifierSupport`，`getNameQualifierSupport`
- `setUnquotedCaseStrategy`，`getUnquotedCaseStrategy`
- `setQuotedCaseStrategy`，`getQuotedCaseStrategy`
- `applyReservedWords`，`setReservedWords`，`clearReservedWords`

构建 `IdentifierHelper`

`IdentifierHelper` 将根据配置的策略处理标识符规范化，构建并返回一个 `IdentifierHelper` 实例。

```
public IdentifierHelper build() {
    if ( unquotedCaseStrategy == quotedCaseStrategy ) {
        log.debugf(
                "IdentifierCaseStrategy for both quoted and unquoted identifiers was set " +
                        "to the same strategy [%s]; that will likely lead to problems in schema update " +
                        "and validation if using quoted identifiers",
                unquotedCaseStrategy.name()
        );
    }

    return new NormalizingIdentifierHelperImpl(
            jdbcEnvironment,
            nameQualifierSupport,
            globallyQuoteIdentifiers,
            skipGlobalQuotingForColumnDefinitions,
            autoQuoteKeywords,
            reservedWords,
            unquotedCaseStrategy,
            quotedCaseStrategy
```

```
20      );
21  }
22
```

identifier quoting

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
openQuote()	用于开启引用标识符的该方言特有的字符。	`"`	继承
closeQuote()	用于关闭引用标识符的该方言特有的字符。	`"`	继承
quote(String name)	应用方言特定的引用。		继承

DDL

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
`getTableExporter`	获取表的导出器，用于创建和删除表的 SQL 语句	Exporter<Table>	继承
`getSequenceExporter`	获取序列的导出器，用于创建和删除序列的 SQL 语句	Exporter<Sequence>	继承
`getIndexExporter`	获取索引的导出器，用于创建和删除索引的 SQL 语句	Exporter<Index>	继承
`getForeignKeyExporter`	获取外键约束的导出器，用于创建和删除外键的 SQL 语句	Exporter<ForeignKey>	继承
`getUniqueKeyExporter`	获取唯一约束的导出器，用于创建和删除唯一约束的 SQL 语句	Exporter<Constraint>	继承
`getAuxiliaryDatabaseObjectExporter`	获取辅助对象的导出器，用于创建和删除辅助对象的 SQL 语句	Exporter<AuxiliaryDatabaseObject>	继承

目录（Catalog）管理

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
Catalog			
canCreateCatalog()	判断是否支持 `catalog` （数据库目录）的创建	false	继承
getCreateCatalogCommand(String)	获取创建 `catalog` 的 SQL 语句	throw new UnsupportedOperationException	继承
getDropCatalogCommand(String)	获取删除 `catalog` 的 SQL 语句	throw new UnsupportedOperationException	继承

模式（Schema）管理

- 模式创建
- 模式删除
- 当前模式信息

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
Schema			
canCreateSchema()	判断是否支持 `schema` 的创建	true	继承
getCreateSchemaCommand(String)	获取创建 `schema` 的 SQL 语句	`"create schema schema_name"`	重载
getDropSchemaCommand(String)	获取删除 `schema` 的 SQL 语句	`"drop schema shcema_name"`	重载
getCurrentSchemaCommand()	获取查询当前 `schema` 的 SQL 语句	null	重载
qualifyIndexName()	判断是否需要使用 `schema` 名来限定索引名称	true	继承

表（Table）管理

- 表创建
- 表删除
- 表修改

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
TABLE			
Create Table			
getCreateTableString()	返回创建表的 SQL 语句片段。	`"create table"`	继承
getCreateMultisetTableString()	返回用于创建多集表的 SQL 语句片段。	`"create table"`	继承
getTableTypeString()	获取创建表时指定存储引擎等类型的 SQL 片段	`""`	继承
Drop Table			
supportsIfExistsBeforeTableName()	判断是否支持在 `DROP TABLE` 语句中的表名前加 `IF EXISTS`	false	重载
supportsIfExistsAfterTableName()	判断是否支持在 `DROP TABLE` 语句中的表名后加 `IF EXISTS`	false	继承
getDropTableString(String)	生成 `DROP TABLE` 语句	`"drop table [if exists] table_name"`	继承
getCascadeConstraintsString	获取 `DROP TABLE` 或 `DROP CONSTRAINT` 语句中级联删除的 SQL 片段	""	重载
ALTER Table			
hasAlterTable()	判断是否支持 `ALTER TABLE` 语句	true	继承
supportsIfExistsAfterAlterTable()	判断是否支持在 `ALTER TABLE` 语句中加 `IF EXISTS`	false	重载

getAlterTableString(String tableName)	生成 `Alter TABLE` 语句	`"alter table [if exists] table_name"`	继承
Table Column			
getAddColumnString()	获取添加列的 SQL 片段	throw new UnsupportedOperationException	重载
getAddColumnSuffixString()	获取添加列时的 SQL 语句后缀	`""`	继承
getDefaultMultiTableBulkIdStrategy()	返回处理多表批量 ID 操作的默认策略类实例，用于处理批量插入或删除等操作。	PersistentTableBulkIdStrategy 实例	重载

约束（Constraint）管理

- 外键约束
- 主键约束
- 唯一约束
- 检查约束
- 约束删除

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
Constraints			
外键约束、主键约束、唯一约束			
getAddForeignKeyConstraintString(String, String[], String, String[], boolean)	获取添加外键约束的 SQL 语句	`"add constraint constraint_name foreign key (foreign_key) references [(primaryKey)]"`	继承
	获取添加外键约束的 SQL 语句		继承

getAddForeignKeyConstraintString(String, String)		`"add constraint constraint_name foreign_key_definition"`	
supportsCascadeDelete()	判断是否支持外键的级联删除	true	继承
dropConstraints()	判断在删除表之前是否需要删除表上的约束	true	继承
getDropForeignKeyString()	获取删除外键约束的 SQL 片段	`" drop constraint "`	继承
hasSelfReferentialForeignKeyBug()	判断数据库在处理自引用外键时是否存在 bug	false	继承
getAddPrimaryKeyConstraintString(String)	获取添加主键约束的 SQL 语句	`"add constraint constraint_name primary key"`	继承
supportsIfExistsBeforeConstraintName()	判断是否支持在 `DROP CONSTRAINT` 之前中加 `IF EXISTS`	false	重载
supportsIfExistsAfterConstraintName()	判断是否支持在 `DROP CONSTRAINT` 之后中加 `IF EXISTS`	false	继承
getCascadeConstraintsString()	获取 `DROP TABLE` 或 `DROP CONSTRAINT` 语句中级联删除的 SQL 片段	`""`	重载
检查约束			
supportsColumnCheck()	判断是否支持列级别的 `CHECK` 约束	true	继承
supportsTableCheck()	判断是否支持表级别的 `CHECK` 约束	true	继承

注释（Comment）支持

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)

supportsCommentOn()	判断是否支持在表、列等对象上添加注释	false	重载
getTableComment(String)	获取表注释的 SQL 语句片段	`""`	继承
getColumnComment(String)	获取列注释的 SQL 语句片段	`""`	继承

其他 DDL 支持

方法名称	功能描述	Dialect(default)	Dialect(OpenGauss)
Comment			
getNullColumnString()	获取可为空列的 SQL 片段	`""`	继承
getCrossJoinSeparator()	获取 `CROSS JOIN` 分隔符	`" cross join "`	继承
getColumnAliasExtractor	获取列别名提取器	ColumnAliasExtractor.COLUMN_LABEL_EXTRACTOR	继承

相关 SQL 尝试

以下在 OpenGauss 数据库进行 SQL 语句测试所用，主要用于熟悉 OpenGauss 特性：

Catalog/Schema

```
/* Schema */
CREATE SCHEMA test_schema;

SELECT current_schema();

DROP SCHEMA test_schema;
```

创建多个 schema 并在不同的 schema 中创建索引，看看是否需要 schema 前缀来引用索引。

1. **创建多个 Schema**

```
CREATE SCHEMA test_schema1;
CREATE SCHEMA test_schema1;
```

2. **创建多个索引**

```sql
-- 在 test_schema1 中创建表和索引
CREATE TABLE test_schema1.my_table (id INT PRIMARY KEY, name VARCHAR(100));
CREATE INDEX test_schema1.my_index ON test_schema1.my_table (name);

-- 在 test_schema2 中创建相同的表和索引名称
CREATE TABLE test_schema2.my_table (id INT PRIMARY KEY, name VARCHAR(100));
CREATE INDEX test_schema2.my_index ON test_schema2.my_table (name);
```

3. **查询索引信息**

```sql
-- 查看所有索引
SELECT * FROM pg_indexes WHERE tablename = 'my_table';
```

4. **环境清理**

```sql
-- 删除 test_schema1 和 test_schema2 中的索引
DROP INDEX test_schema1.my_index;
DROP INDEX test_schema2.my_index;
```

```sql
-- 删除 test_schema1 和 test_schema2 中的表
DROP TABLE test_schema1.my_table;
DROP TABLE test_schema2.my_table;
```

```sql
-- 删除 test_schema1 和 test_schema2
DROP SCHEMA test_schema1;
DROP SCHEMA test_schema2;
```

外键约束

Example1：

1. **清空环境 - 删除测试表（如果存在）**

```sql
DROP TABLE IF EXISTS child_table;
DROP TABLE IF EXISTS parent_table;
```

2. **创建测试表并添加外键约束**

```sql
-- 创建父表
CREATE TABLE parent_table (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);
-- 创建子表并添加外键约束
CREATE TABLE child_table (
    id INT PRIMARY KEY,
    parent_id INT,
    CONSTRAINT fk_parent FOREIGN KEY (parent_id) REFERENCES parent_table(id)
);
```

3. **删除外键约束**

```sql
-- 尝试删除外键约束
ALTER TABLE child_table DROP CONSTRAINT fk_parent;
```

4. **检查约束是否成功删除**

```sql
-- 查看表结构，确保外键约束已经删除
SELECT constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'child_table' AND constraint_type = 'FOREIGN KEY';
```

5. **清空环境 - 删除测试表**

```sql
DROP TABLE IF EXISTS child_table;
DROP TABLE IF EXISTS parent_table;
```

Example2：

1. **清空环境 - 删除测试表（如果存在）**

```sql
DROP TABLE IF EXISTS child_table;
DROP TABLE IF EXISTS parent_table;
```

2. **创建测试表**

```
CREATE TABLE parent_table (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE child_table (
    id INT PRIMARY KEY,
    parent_id INT
);
```

3. **添加外键约束**

```
ALTER TABLE child_table ADD CONSTRAINT fk_child_parent FOREIGN KEY (parent_id)
REFERENCES parent_table(id);
```

4. **检查约束是否成功添加**

```
SELECT constraint_name
FROM information_schema.table_constraints
WHERE table_name = 'child_table' AND constraint_type = 'FOREIGN KEY';
```

5. **清空环境 - 删除测试表**

```
DROP TABLE IF EXISTS child_table;
DROP TABLE IF EXISTS parent_table;
```

Example3：

1. **清空环境 - 删除测试表（如果存在）**

```
DROP TABLE IF EXISTS employee;
```

2. **创建测试表**

```
CREATE TABLE employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    manager_id INT,
    CONSTRAINT fk_manager FOREIGN KEY (manager_id) REFERENCES
employee(employee_id)
);
```

3. **添加测试数据**

```
INSERT INTO employee (employee_id, name, manager_id) VALUES (1, 'CEO', NULL);
INSERT INTO employee (employee_id, name, manager_id) VALUES (2, 'Manager', 1);
INSERT INTO employee (employee_id, name, manager_id) VALUES (3, 'Employee', 2);
```

4. **执行删除操作**

尝试删除某个自引用的行：

```
DELETE FROM employee WHERE employee_id = 1;
```

ERROR: update or delete on table "employee" violates foreign key constraint "fk_manager" on table "employee"

DETAIL: Key (employee_id)=(1) is still referenced from table "employee".

删除操作失败，并提示由于自引用外键问题无法删除。

5. **清空环境 - 删除测试表**

```
DROP TABLE IF EXISTS employee;
```

Example4：

1. **清空环境 - 删除测试表（如果存在）**

```
DROP TABLE IF EXISTS test_null_column;
```

2. **创建测试表**

不使用显式的 `NULL` 声明，创建一个可空的列，观察其默认行为：

```sql
CREATE TABLE test_null_column (
    id INT PRIMARY KEY,
    name VARCHAR(255)
);
```

3. **验证列的可空性**

```sql
SELECT column_name, is_nullable
FROM information_schema.columns
WHERE table_name = 'test_null_column';
```

返回结果显示 `name` 列是可空的，意味着 OpenGauss 不需要显式声明 `NULL`，默认情况下列是可空的。

4. **使用显式 `NULL` 声明创建列**

```sql
ALTER TABLE test_null_column ADD description VARCHAR(255) NULL;
```

5. **再次验证列的可空性**

```sql
SELECT column_name, is_nullable
FROM information_schema.columns
WHERE table_name = 'test_null_column';
```

6. **清空环境 - 删除测试表**

```sql
DROP TABLE test_null_column;
```

其他支持

SQL 语法支持

- supportsEmptyInList()
  作用: 检查是否支持空的 `IN` 列表。
  默认返回值: `true`

- areStringComparisonsCaseInsensitive()
  作用: 检查字符串比较是否区分大小写。
  默认返回值: `false`
- supportsRowValueConstructorSyntax()
  作用: 检查是否支持“行值构造器”语法。
  默认返回值: `false`
- supportsRowValueConstructorSyntaxInSet()
  作用: 检查是否支持在 `SET` 子句中使用“行值构造器”语法。
  默认返回值: `false`
- supportsRowValueConstructorSyntaxInInList()
  作用: 检查是否支持在 `IN` 列表中使用“行值构造器”语法。
  默认返回值: `false`
- supportsParametersInInsertSelect()
  作用: 检查是否支持 `INSERT ... SELECT ...` 语句中的参数。
  默认返回值: `true`
- replaceResultVariableInOrderByClauseWithPosition()
  作用: 检查是否需要将 `ORDER BY` 子句中的变量替换为列位置。
  默认返回值: `false`
- requiresCastingOfParametersInSelectClause()
  作用: 检查 `SELECT` 子句中的参数是否需要 `cast()`。
  默认返回值: `false`
- supportsSubselectAsInPredicateLHS()
  作用: 检查是否支持在 `IN` 谓词的左侧使用子查询。
  默认返回值: `true`
- supportsSubqueryOnMutatingTable()
  作用: 检查是否支持在 `UPDATE/DELETE` 查询中引用正在修改的表。
  默认返回值: `true`
- supportsExistsInSelect()
  作用: 检查是否支持在 `SELECT` 子句中使用 `EXISTS` 语句。
  默认返回值: `true`
- supportsTupleCounts()
  作用: 检查是否支持 `COUNT(a,b)` 形式的元组计数。
  默认返回值: `false`
- supportsTupleDistinctCounts()
  作用: 检查是否支持 `COUNT(DISTINCT a,b)` 形式的元组计数。
  默认返回值: `true`

- requiresParensForTupleDistinctCounts()
  作用: 检查在 `COUNT(DISTINCT a,b)` 时是否需要括号。
  默认返回值: `false`

```sql
-- Create test_table and insert sample data
CREATE TABLE test_table (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

INSERT INTO test_table (first_name, last_name) VALUES
('John', 'Doe'),
('Jane', 'Doe'),
('John', 'Smith'),
('Jane', 'Smith'),
('John', 'Doe'); -- Duplicate to test counting

-- Test COUNT(a, b)
SELECT COUNT(first_name, last_name) FROM test_table;

-- Test COUNT(DISTINCT a, b) without parentheses
SELECT COUNT(DISTINCT first_name, last_name) FROM test_table;

-- Test COUNT(DISTINCT (a, b)) with parentheses
SELECT COUNT(DISTINCT (first_name, last_name)) FROM test_table;
```

- getInExpressionCountLimit()
  作用: 返回 `IN` 谓词中元素数量的限制。
  默认返回值: `0`

```sql
-- Create test environment for IN expression count limit
CREATE TABLE test_in_limit (
    id INT PRIMARY KEY
);

-- Insert a large number of test rows
DO $$
BEGIN
    FOR i IN 1..10000 LOOP
        INSERT INTO test_in_limit (id) VALUES (i);
    END LOOP;
END $$;
```

```
13
14 -- Test IN predicate with a large number of elements
15 SELECT * FROM test_in_limit WHERE id IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
16
17 -- Clean up the environment
18 DROP TABLE test_in_limit;
```

- supportsTuplesInSubqueries()
  作用: 检查是否支持在子查询中使用元组。
  默认返回值: `true`

```
1  -- Create test environment for tuple support in subqueries
2  CREATE TABLE table1 (
3      col1 INT,
4      col2 INT
5  );
6
7  CREATE TABLE table2 (
8      col1 INT,
9      col2 INT
10 );
11
12 -- Insert test data into both tables
13 INSERT INTO table1 (col1, col2) VALUES (1, 1), (2, 2), (3, 3);
14 INSERT INTO table2 (col1, col2) VALUES (1, 1), (3, 3);
15
16 -- Test tuple usage in subqueries
17 DELETE FROM table1 WHERE (col1, col2) IN (SELECT col1, col2 FROM table2);
18
19 -- Verify deletion result
20 SELECT * FROM table1;
21
22 -- Clean up the environment
23 DROP TABLE table1;
24 DROP TABLE table2;
```

- supportsSelectAliasInGroupByClause()
  作用: 检查是否支持在 `GROUP BY` 子句中使用 `SELECT` 别名。
  默认返回值: `false`

```
1 -- Create test environment for GROUP BY alias support
2 CREATE TABLE test_group_by_alias (
```

```
    id INT,
    amount DECIMAL
);

-- Insert test data into the table
INSERT INTO test_group_by_alias (id, amount) VALUES (1, 100.0), (1, 200.0),
(2, 150.0);

-- Test GROUP BY with alias (this should fail in OpenGauss)
SELECT id AS identifier, SUM(amount) AS total_amount
FROM test_group_by_alias
GROUP BY identifier;  -- OpenGauss does not support alias in GROUP BY

-- Test GROUP BY using column names (this should work)
SELECT id AS identifier, SUM(amount) AS total_amount
FROM test_group_by_alias
GROUP BY id;

-- Clean up the environment
DROP TABLE test_group_by_alias;
```

- supportsNonQueryWithCTE()
  作用: 检查是否支持非查询语句中的 CTE。
  默认返回值: `false`

```
-- Create test environment for CTE with non-query statements
CREATE TABLE test_table_cte (
    id INT PRIMARY KEY,
    amount DECIMAL
);

-- Insert test data into the table
INSERT INTO test_table_cte (id, amount) VALUES (1, 100.0), (2, 200.0), (3,
300.0);

-- Test CTE in DELETE statement
WITH cte AS (
    SELECT id FROM test_table_cte WHERE id > 1
)
DELETE FROM test_table_cte WHERE id IN (SELECT id FROM cte);

-- Test CTE in UPDATE statement
WITH cte AS (
    SELECT id FROM test_table_cte WHERE id = 1
)
```

```
UPDATE test_table_cte SET amount = amount * 1.1 WHERE id IN (SELECT id FROM
cte);

-- Test CTE in INSERT statement
WITH cte AS (
    SELECT id, amount FROM test_table_cte WHERE id = 1
)
INSERT INTO test_table_cte (id, amount) SELECT id + 100, amount FROM cte;

-- Clean up the environment
DROP TABLE test_table_cte;
```

- supportsValuesList()
  作用: 检查是否支持 `VALUES` 列表。
  默认返回值: `false`

```
-- Create test environment for VALUES list support
CREATE TABLE test_values_list (
    value INT
);

-- Test VALUES list directly in a query
SELECT * FROM (VALUES (1), (2), (3)) AS t(value);

-- Test using VALUES list in an INSERT statement
INSERT INTO test_values_list (value) VALUES (1), (2), (3);

-- Verify the result of the insert
SELECT * FROM test_values_list;

-- Clean up the environment
DROP TABLE test_values_list;
```

LOB 操作支持

- useInputStreamToInsertBlob()
  作用: 检查是否应使用流操作插入 BLOB 数据。
  默认返回值: `true`
- supportsExpectedLobUsagePattern()
  作用: 检查是否支持正常的 LOB 使用模式。
  默认返回值: `true`

- supportsLobValueChangePropogation()
  作用: 检查 LOB 的更改是否会传播到数据库。
  默认返回值: `true`
- supportsUnboundedLobLocatorMaterialization()
  作用: 检查是否支持在事务外生成 LOB 定位符。
  默认返回值: `true`
- forceLobAsLastValue()
  作用: 检查是否需要将 LOB 值放在最后。
  默认返回值: `false`
- supportsJdbcConnectionLobCreation(DatabaseMetaData databaseMetaData)
  作用: 检查是否通过 JDBC 连接支持 LOB 创建。
  默认返回值: `true`

```
CREATE TABLE test_lob (
    id INT PRIMARY KEY,
    blob_data BLOB,
    clob_data CLOB
);
```

隔离级别

- doesReadCommittedCauseWritersToBlockReaders()
  作用: 检查 `READ_COMMITTED` 隔离级别是否会导致写者阻塞读者。
  默认返回值: `false`

> ✍️ **事务 A**：开启事务后，执行 `UPDATE` 操作，将 `value` 更新为 `200`，但不提交。
>
> **事务 B**：在另一个连接中，开启事务，执行 `SELECT` 查询，尝试读取 `value` 的值。
>
> **验证**：检查事务 B 是否被阻塞，以及读取到的值是否为 `100`。
>
> **结果**：如果事务 B 未被阻塞，且读取到的值为 `100`，则说明写操作未阻塞读操作，方法应返回 `false`。

- doesRepeatableReadCauseReadersToBlockWriters()
  作用: 检查 `REPEATABLE_READ` 隔离级别是否会导致读者阻塞写者。
  默认返回值: `false`

> 🏕️ **事务 A**：开启事务后，执行 `SELECT` 操作读取 `value` 的值，不提交。
>
> **事务 B**：在另一个连接中，开启事务，尝试 `UPDATE` 同一行数据。

验证：检查事务 B 是否被阻塞或抛出异常。

结果：如果事务 B 未被阻塞，且更新成功，则说明读操作未阻塞写操作，方法应返回 `false`。

提示和注释支持

查看文档：

- addSqlHintOrComment(String sql, QueryParameters parameters, boolean commentsEnabled)
  作用: 在 SQL 查询中添加提示或注释。
  默认返回值: 修改后的 SQL 字符串。
- prependComment(String sql, String comment)
  作用: 在 SQL 查询前添加注释。
  默认返回值: 带有注释的 SQL 字符串。
- escapeComment(String comment)
  作用: 转义注释中的特殊字符。
  默认返回值: 转义后的注释字符串。
- getQueryHintString(String query, List<String> hintList)
  作用: 在查询中应用提示。
  默认返回值: 带有提示的修改后的 SQL 查询字符串。
- getQueryHintString(String query, String hints)
  作用: 在查询中应用提示。
  默认返回值: 带有提示的 SQL 查询字符串。

约束和唯一性支持

查看文档：

- getUniqueDelegate()
  作用: 获取此 SQL 方言的 `UniqueDelegate`。
  默认返回值: `UniqueDelegate` 实例。
- supportsUnique()
  作用: 检查是否支持 `UNIQUE` 列语法。
  默认返回值: `true`
- supportsUniqueConstraintInCreateAlterTable()
  作用: 检查是否支持在 `CREATE/ALTER TABLE` 语句中添加唯一约束。
  默认返回值: `true`

- getAddUniqueConstraintString(String constraintName)
  作用: 获取添加唯一约束的 SQL 片段。
  默认返回值: 添加唯一约束的 SQL 片段字符串。
- supportsNotNullUnique()
  作用: 检查是否支持 `NOT NULL` 和 `UNIQUE` 组合的约束。
  默认返回值: `true`

锁

- useFollowOnLocking()
  作用: 检查是否通过后续的 `SELECT` 来应用悲观锁。
  默认返回值: `false`
- useFollowOnLocking(QueryParameters parameters)
  作用: 检查是否通过后续的 `SELECT` 来应用悲观锁，基于查询参数。
  默认返回值: `false`

滚动和查询模式支持

- defaultScrollMode()
  作用: 返回默认的滚动模式。
  默认返回值: `ScrollMode.SCROLL_INSENSITIVE`
- supportsResultSetPositionQueryMethodsOnForwardOnlyCursor()
  作用: 检查是否支持在只向前游标上查询结果集位置信息。
  默认返回值: `true`

其他数据库特性支持

- getNotExpression(String expression)
  作用: 返回一个表达式的取反形式。
  默认返回值: `not + expression`
- supportsBindAsCallableArgument()
  作用: 检查是否支持将 JDBC 绑定参数作为函数或存储过程的参数。
  默认返回值: `true`
- supportsPartitionBy()
  作用: 检查是否支持 `PARTITION BY` 子句。
  默认返回值: `false`
- **getDefaultBatchLoadSizingStrategy()**
  作用: 获取默认的批量加载大小策略。
  默认返回值: `50`

- **批量插入测试**：对不同的批量大小进行测试，记录每种批量大小的耗时。

- **确定最佳批量大小**：根据测试结果，选择耗时最短的批量大小作为默认值。
- **结果**：根据测试结果，在 `OpenGaussDialect` 中设置合适的批量加载大小。

批量加载大小的测试结果可能会因环境、硬件配置等因素而异，需要根据实际情况进行调整。

- supportsNamedParameters(DatabaseMetaData databaseMetaData)
  作用: 检查是否支持命名参数。
  默认返回值: `databaseMetaData.supportsNamedParameters()`
- supportsSkipLocked()
  作用: 检查是否支持 `SKIP LOCKED` 超时。
  默认返回值: `false`
- supportsNoWait()
  作用: 检查是否支持 `NO WAIT` 超时。
  默认返回值: `false`
- getCreateTemporaryTableColumnAnnotation(int sqlTypeCode)
  作用: 获取临时表列定义中的注释。
  默认返回值: `""`
- inlineLiteral(String literal)
  作用: 内联字符串字面值，并对其进行转义。
  默认返回值: 转义后的字符串字面值。
- escapeLiteral(String literal)
  作用: 转义字符串字面值中的特殊字符。
  默认返回值: 转义后的字符串字面值。

```
1  private static final Pattern SINGLE_QUOTE_PATTERN = Pattern.compile("'",
   Pattern.LITERAL);
2  private static final String TWO_SINGLE_QUOTES_REPLACEMENT = "''";
3
4  public String inlineLiteral(String literal) {
5      return String.format("'%s'", escapeLiteral(literal));
6  }
7
8  protected String escapeLiteral(String literal) {
9      return
   SINGLE_QUOTE_PATTERN.matcher(literal).replaceAll(TWO_SINGLE_QUOTES_REPLACEMENT)
   ;
10 }
11
```

- isLegacyLimitHandlerBehaviorEnabled()
  作用: 检查是否启用了传统的限制处理程序行为。
  默认返回值: `false`
- resolveLegacyLimitHandlerBehavior(ServiceRegistry serviceRegistry)
  作用: 确定是否启用传统的限制处理行为。
  默认返回值: 无返回值 (`void`)