**Postgre数据库中关于大对象的存取**

作者：霍琦

目录

[1. 数据库定义 2](#_Toc421088859)

[1.1. bytea 2](#_Toc421088860)

[1.2. oid 2](#_Toc421088861)

[1.3. 比较 2](#_Toc421088862)

[2. Java代码 3](#_Toc421088863)

[2.1. bytea 3](#_Toc421088864)

[2.2. oid 3](#_Toc421088865)

# 数据库定义

## bytea

在PostgreSQL中存储二进制数据，例如存储Word、Excel文档，图片文件等，可以使用bytea类型的列。bytea类型是PostgreSQL特有的存储二进制数据的字段类型，与SQL标准中的BLOB和BINARY LARGE OBJECT类型异曲同工。这在PostgreSQL文档的bytea类型介绍中有所说明。

## oid

除了标准的bytea(等同于SQL中的blob)类型外,还有一个pg独有的大对象(largeobject)类型.大对象类型实际上是个整型地址类型,它指向一张系统表pg\_largeobejct的主键loid, 实际的内容都存在这张系统表里,而且这内容是按分块存储的.

## 比较

两者的区别和利弊:

1, 用大对象类型时,二进制数据统一存储在一张系统表里, 与其他数据分开,提高其他数据读写性能. 并能和其他数据分开备份.

2, 由于二进制数据是分块(大概每8kb一块)存储在大对象系统表里, 并且这系统表建有索引. 所以尺寸大的数据, 随机读写的性能将提高(比如只要读取后面多少字节), 若存储在bytea, 就没有这方面的优势.

3, 大对象类型的读取需要有专门的操作函数, 必须要写程序才能读取,比较烦琐,而bytea相对简单.

# Java代码

## bytea

public String fetchFileByBytes(String path) {

 String sql = "select \* from blob\_table";

 Connection conn = null;

 PreparedStatement ps = null;

 ResultSet rs = null;

 File file = null;

 OutputStream fos = null;

 try {

 conn = Util.getConn();

 ps = conn.prepareStatement(sql);

 rs = ps.executeQuery();

 byte[] buffer = null;

 file = new File(path);

 fos = new FileOutputStream(file);

 while (rs.next()) {

 buffer = rs.getBytes("file");

 fos.write(buffer);

 }

 fos.flush();

 fos.close();

 } catch (SQLException e) {

 e.printStackTrace();

 return null;

 } catch (IOException e) {

 e.printStackTrace();

 } finally {

 Util.close(null, ps, conn);

 }

 return null;

 }

 public String saveFileByByteas(String path) {

 Connection conn = null;

 PreparedStatement ps = null;

 File file = null;

 FileInputStream fis = null;

 InputStream is = null;

 try {

 file = new File(path);

 fis = new FileInputStream(file);

 is = new BufferedInputStream(fis);

 System.out.println(file.length());

 String sql = "insert into blob\_table (id,name,dsc,file) values(?,?,?,?)";

 conn = Util.getConn();

 ps = conn.prepareStatement(sql);

 ps.setInt(1, 1);

 ps.setString(2, file.getName());

 ps.setString(3, file.getAbsolutePath());

 ps.setBinaryStream(4, fis, (int) file.length());

 ps.executeUpdate();

 System.out.println("插入成功！");

 } catch (FileNotFoundException e) {

 e.printStackTrace();

 } catch (SQLException e) {

 e.printStackTrace();

 } finally {

 Util.close(null, ps, conn);

 try {

 fis.close();

 } catch (IOException e) {

 e.printStackTrace();

 }

 }

 return null;

 }

相应的，如果大对象不是存放在文件中，比如虚拟机中的类对象，可以通过ByteArrayInputStream实现。

## oid

public String fetchFileByBlob(String path) {

 String sql = "select \* from blob\_table ";

 Connection conn = null;

 PreparedStatement ps = null;

 ResultSet rs = null;

 File file = null;

 OutputStream fos = null;

 try {

 conn = Util.getConn();

 conn.setAutoCommit(false);

 // 获取大对象管理器以便进行操作

 LargeObjectManager lobj = ((org.postgresql.PGConnection) conn)

 .getLargeObjectAPI();

 ps = conn.prepareStatement(sql);

 rs = ps.executeQuery();

 byte[] buffer = null;

 file = new File(path);

 fos = new FileOutputStream(file);

 while (rs.next()) {

 Long oid = rs.getLong("file");

 LargeObject obj = lobj.open(oid, LargeObjectManager.READ);

 buffer = new byte[obj.size()];

 obj.read(buffer, 0, obj.size());

 fos.write(buffer);

 obj.close();

 }

 fos.flush();

 fos.close();

 } catch (SQLException e) {

 e.printStackTrace();

 return null;

 } catch (IOException e) {

 e.printStackTrace();

 } finally {

 Util.close(null, ps, conn);

 }

 return null;

 }

 public String saveFileByBlob(String path) {

 Connection conn = null;

 PreparedStatement ps = null;

 File file = null;

 FileInputStream fis = null;

 try {

 file = new File(path);

 fis = new FileInputStream(file);

 System.out.println(file.length());

 String sql = "insert into blob\_table (id,name,dsc,file) values(?,?,?,?)";

 conn = Util.getConn();

 conn.setAutoCommit(false);

 PGConnection pgCon = (PGConnection) conn;

 // 获取大对象管理器以便进行操作

 LargeObjectManager lobj = pgCon.getLargeObjectAPI();

 // LargeObjectManager lobj =new LargeObjectManager((BaseConnection)

 // conn);

 // // 创建一个新的大对象

 int oid = lobj.create(LargeObjectManager.READ

 | LargeObjectManager.WRITE);

 System.out.println("oid:" + oid);

 // 打开一个大对象进行写

 LargeObject obj = lobj.open(oid, LargeObjectManager.WRITE);

 byte buf[] = new byte[(int) file.length()];

 int s, tl = 0;

 while ((s = fis.read(buf, 0, (int) file.length())) > 0) {

 obj.write(buf, 0, s);

 tl += s;

 }

 obj.close();

 ps = conn.prepareStatement(sql);

 ps.setInt(1, 1);

 ps.setString(2, file.getName());

 ps.setString(3, file.getAbsolutePath());

 ps.setInt(4, oid);

 ps.executeUpdate();

 conn.commit();

 System.out.println(ps.executeUpdate());

 System.out.println("插入成功！");

 } catch (FileNotFoundException e) {

 e.printStackTrace();

 } catch (SQLException e) {

 e.printStackTrace();

 } catch (IOException e) {

 e.printStackTrace();

 } finally {

 Util.close(null, ps, conn);

 try {

 fis.close();

 } catch (IOException e) {

 e.printStackTrace();

 }

 }

 return null;

 }