基于 SDE API 的影像数据高效存储研究*

沈林芳，刘仁义，刘 南
(浙江大学 地理信息科学研究所，浙江 杭州 310028)

摘 要：对于目前与日俱增的影像数据，难以实现快捷高效的管理存储，应有更有效的管理方法来满足目前应用需要。阐述了 SDE 中影像数据存储机理。在此基础上，分析了用 API 开发影像管理系统时，存储影像数据的关键技术，并给出了实现代码。

关键词：SDE；API；影像数据；关系型数据库
中图法分类号：TP391.41 **文献标识码**：A **文章编号**：1001-3695(2005)02-0024-02

Study of Efficiently Storaging Images Data Based on SDE API

SHEN Lin-fang, LIU Ren-yi, LIU Nan
(Institute of Geographic Information Science Research, Zhejiang University, Hangzhou Zhejiang 310028, China)

Abstract: With the development of remote sensing technology, images data are becoming bigger and bigger at present. It is difficult to realize the storage and management of the massive data. Thus, we need an efficient management to meet the application demand. Based on the research of architecture and principle of ArcSDE for images data, the paper focuses on the key technology of storaging images data and the codes are given.

Key words: SDE; API; Images Data; Relational Database

1 引言

由于影像数据具有直观和信息量丰富等特点，在地质勘探、海洋监测等领域里备受青睐。传统的管理模式为文件和关系数据库混合模式，即影像数据以文件方式存储在本地磁盘上，而属性信息采用关系数据库表方式。然而，这种模式由于影像属性信息和影像数据文件分离管理，数据更新维护成本相当大，难以实现影像数据共享、分布式管理、多用户并发操作及安全恢复等，无法满足目前用户的需求。

SDE(Spatial Database Engine)是此问题的最好解决方案。SDE 将空间数据和与其相关的属性数据统一放到工业标准关系数据库中进行管理，同时采取开放策略，提供标准的应用程序接口(API)，使得海量影像数据的管理获得一种较为理想的模式。同时，使得面向多用户，在广域网式以真正的客户/服务器方式提供了强有力的数据服务，实现了数据统一管理。

本文在分析了 SDE 相关技术的基础上，提出了一种在 VC++环境中基于 SDE C API 的影像数据存储管理方法，并开发了一个影像数据库原型系统。

2 SDE 相关技术分析

SDE 实际是普通关系数据库的扩展，并支持最新的 SQL3.0标准，因此，享有快速高效存储、访问和更新数据能力。

收稿日期：2004-03-10；修返日期：2004-05-22
基金项目：国家“863”计划资助项目(2001AA630301)；国家自然科学基金项目(40271087)；浙江省自然科学基金项目(401006)

2.1 影像数据存储机理

SDE 主要通过用户业务表(Business Table)、栅格列表(Raster Columns Table)、栅格表(Raster Table)、波段表(Raster Band Table)、数据块表(Raster Blocks Table)、辅助表(Raster Auxiliary Table)六张表来实现对影像数据的存储和管理。

(1)栅格列表。它是影像数据在 RDBMS 中的管理表。该表主要存储影像列名、含有影像列的业务表名。在导入影像数据时，SDE 自动在表中添加一条记录，并分配一个唯一标志符 rastercolumn_id。

(2)用户业务表。该表名由用户定义，是用户可见表。业务表中必须有一个影像列，并在该列中存储一个影像数据的引用。通过影像列名与栅格列表建立联系。

(3)栅格表。存储表名为 SDE_RAS_<rastercolumn_id>，存储影像的基本描述信息和关键字 raster_id。此表中的记录和栅格列中的记录是一一对应的。

(4)波段表。存储表名为 SDE_BND_<rastercolumn_id>，存储影像栅格号、波段号、波段宽高、块宽高、边界范围以及创建日期等。每张影像图根据波段数分成多条记录存储在波段表中，并通过 raster_id 与栅格建立多对一的关系。

(5)数据块表。存储表名为 SDE_BLK_<rastercolumn_id>，主要存储影像数据块、波段号、块行列号，金字塔等级。

(6)辅助表。存储表名为 SDE_AUX_<rastercolumn_id>，存储一般的辅助信息，如颜色表、统计信息等。

2.2 分块分级技术

根据上述对 SDE 影像数据在 RDBMS 中存储机理的分析，不难看出在 SDE 中，自动将大型的栅格数据集分割成若干的块(Tiles)，并以二进制大对象(BLOB)方式存储在关系数据库

的二维表中(表 1) 。一般情况下, 数据块大小 128 ×128 已经基本能满足应用。块大小与数据访问效率有着密切关系, 应根据数据类型、数据库和网络状况而定。在 SDE 中, 自左向右自上而下对影像数据进行分块处理, 不到一块的数据块作 128 × 128 来处理。这种模式的优势在于用户, 只需取当前屏幕所能显示的数据块即可, 大大地减少了数据传输量, 有利于访问效率的提高。

为了提高显示速度, SDE 采用了多分辨率的金字塔等级结构(图 1) , 允许客户端访问指定分辨率下的数据。其基本思想如下:

(1) 原始影像数据为金字塔的最低等级, 为 0 级;

(2) 第 1 级通过对 0 级影像数据重采样压缩为 1 /4 得到, 2 级通过对 1 级影像数据重采样压缩为 1 /4 得到, 以此类推;

(3) 直到最后影像数据的像素个数小于四个为止。

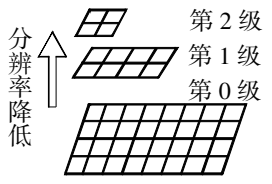

图 1　分级显示示意图

表 1　影像分块分级存储表

波段号	等级	行号	列号	数据
1	0	0	0	数据 1
1	1	2	3	数据 2
⋮	⋮	⋮	⋮	⋮

在实际应用中, 并非创建所有金字塔等级, 一般当重采样影像数据小于数据块时即停止。

因此, 这些额外等级会增加将近 20% 的存储空间, 但是对于大数据量, 特别是大于几百兆的影像数据, 显示效率有显著的提高。

SDE 由于分块分级技术的采用, 对海量影像数据具有很好的支持。在浏览影像时, 只需根据当前显示屏幕大小和将调度的影像数据大小, 计算出当前最佳显示分辨率, 然后提取相应尺度等级的影像数据, 这种访问方式在显示效率有明显提高。

2.3　SDE 影像数据操作

对于 RDBMS 中的影像数据, 用户可以通过许多客户端软件, 如 ArcView, ArcInfo 等, 即可完成对 SDE 中影像数据的导入、读取、修改和分析等操作, 也可以利用 SDE 自身提供的外部编程接口 API 函数来实现同样的操作。API 函数使应用程序开发者和用户不必从底层的 DBMS API 开发, 就可以方便地定制自己的 ArcSDE 客户端, 缩减数据库和应用程序的开发成本。API 函数分 C API 和 Java API 两种。但在目前的版本中 Java API 不支持对影像数据存储管理, 故本文就采用了 C API 开发方式。

C API 是 ESRI 为 C 或C++ 开发人员提供的访问 ArcSDE 接口, 用它可以快捷高效地存储访问 RDBMS 的数据, 还可以实现跨平台。ArcSDE 提供 700 多个函数, 其中包括: 数据库连接和服务实例函数、层管理函数、流和查询管理函数、表管理函数、栅格数据管理函数、日志管理函数等。

3　基于 C API 的影像数据存储

3.1　与 SDE 创建数据流

在客户端应用程序中访问 SDE 中数据时, 首先利用与 SDE 的连接创建数据流, 然后在客户端和服务器之间传送数据。将 SDE 客户端软件安装目录下的头文件和动态连接(pef. dll, sg. dll, sde30. dll) 引入工程。

以下C++程序片段是用于与 ESRI ArcSDE 管理的空间数据库引擎进行连接, 并创建流的程序。

```
SE_CONNECTION Connection; // 与 SDE 的 SE_CONNECTION 对象
SE_STREAM Stream; // 与 SDE 的 SE_STREAM 对象
SE_ERROR error; // 与 SDE 的 SE_ERROR 对象
rc = SE_connection_create( server, instance, database, user, passwd, &error, &Connection) ; // 建立与空间数据库的连接
rc = SE_stream_create ( Connection, &Stream) ; //创建 Stream
…
rc = SE_stream_free ( Stream) ; //释放 Stream
SE_connection_free( Connection) ; //断开连接
```

执行完以上代码后, 空间数据字段被保存在 Stream 对象中, 剩下的任务只需从 Stream 中创建数据或抽取数据。

3.2　影像数据的导入

影像数据入库就通过数据流来实现。SDE 可以处理大数据影像, 在入库时, 它可以通过用户自定义的回调函数动态地逐块上载, 并在服务器端建立金字塔结构。影像导入的基本流程如下: 首先创建数据流, 然后利用它绑定栅格列并设置影像数据基本属性信息, 最后执行数据流, 断开连接。具体实现代码如下:

```
// 创建栅格列
ret = SE_rascolinfo_create ( &hRasCol) ;
// 创建 SE_RASCOLINFO 对象
ret = SE_rascolinfo_set_raster_column ( hRasCol, table, column) ;
//栅格列名和业务表名
ret = SE_rastercolumn_create ( handle, hRasCol) ; //创建影像层
SE_rascolinfo_free ( hRasCol) ; //释放 SE_RASCOLINFO 对象
```

执行完上述代码后, SDE 自动在 RDBMS 中创建栅格表、波段表、数据块表、辅助表。但只是空的表结构, 接下来只需在相应的表中插入数据即可。如果影像数据有坐标信息, 可以通过对应提供的 API 函数, 设置空间参考系统。限于篇幅, 本文不再赘述。

以下是影像数据上载的核心部分代码:

```
ret = SE_stream_insert_table( stream, table, 2, ( const CHAR** ) columns) ; //在业务表中插入两列数据
ret = SE_stream_bind_input_column ( stream, 1, ( void* ) fileName, &notNull) ; //绑定第一列
context = ( CLIENT_DATA* ) calloc ( 1, sizeof( CLIENT_DATA) ) ;
bits_per_pixel = SE_PIXEL_TYPE_GET_DEPTH ( pixelType) ;
//获得像素深度
bytes = ( imageWidth * bits_per_pixel + 7) / 8;
context -> buffer = ( CHAR * ) calloc ( 1, bytes) ;
context -> length = bytes;
context -> fd = open ( fileName, O_RDONLY | O_BINARY) ;
ret = SE_rasterattr_create ( &hRasAttr, TRUE) ;
//创建 SE_RASTERATTR 对象
ret = SE_rasterattr_set_callback ( hRasAttr, RasterDataCallback, ( void* ) context) ; //设置自定义回调函数
ret = SE_rasterattr_set_image_size ( hRasAttr, imageWidth, imageHeight, numBands) ; //设置影像宽高和波段数
ret = SE_rasterattr_get_extent( hRasAttr, &envelope) ;
//设置影像数据范围
ret = SE_rasterattr_set_tile_size ( hRasAttr, tileWidth, tileHeight) ; //设置块宽高
ret = SE_rasterattr_set_pixel_type ( hRasAttr, pixelType) ;
//设置导入像素类型
//绑定影像数据列
ret = SE_stream_bind_input_column ( stream, 2, ( void* ) hRasAttr, &notNull) ;
ret = SE_stream_execute ( stream) ; //执行流, 上载数据
…
```

如果用户想在导入数据后, 在服务器端建立金字塔结构, 需在绑定栅格列之前调用设置金字塔等级接口。该接口中第二个参数设置金字塔最高等级数, 第三个参数设置是否跳过第 0 级, 第四个参数是设置额外等级重采样方式。(下转第 28 页)