# AN ENTERPRISE DATABASE-CENTRIC APPROACH FOR GEOSPATIAL IMAGE MANAGEMENT AND PROCESSING

Qingyun (Jeffrey) Xie, Siva Ravada, Weisheng Xu, Zhihai Zhang

Oracle USA, Inc., One Oracle Drive, Nashua, NH 03062, USA -
(qingyun.xie, siva.ravada, weisheng.xu, zhihai.zhang)@oracle.com

**KEY WORDS:** Spatial Databases, Raster Data, Data Management, Image Processing, Query Processing, Database, Software

**ABSTRACT:**

Geoimagery and raster gridded data are growing exponentially. As a result, we face many challenges. This paper discusses two of the major challenges. One is how to effectively and efficiently archive, manage, process and deliver all those geoimage data. Another challenge is how to make the geoimages and professionally extracted information available to broader audiences so that enterprises and mass consumers can benefit more from our work. This paper focuses on the database server technology, which is one of the key areas that is essential and foundational for solving the above two challenges and beyond. It proposes a new enterprise database-centric approach for geospatial image and raster data management and processing. The key concept of this approach is to enhance and leverage enterprise IT technologies and provide a database-centric solution for image data management as well as key image processing operations. It consists of three major components: a new native database data type for raster data, a server-side image processing and raster operation engine and a standard-based user-friendly interface. It's designed to work in a client-server environment as well as in any multi-tier architecture. The advantages and benefits of this approach are discussed. The Oracle Spatial GeoRaster was designed based on this approach. A series of tests and research using the Oracle GeoRaster technology were conducted and are partially presented in this paper. The results show that this approach is practical, easy-to-use and truly scalable and performant. This database-centric approach is a viable solution for geospatial image management and processing.

## 1. INTRODUCTION

Geoimagery and raster gridded data are growing exponentially. Numerous remote sensors of different types on various platforms are collecting real time data about the Earth and our environment for different purposes on a daily basis. As a result, we face many technical challenges, two of which are the major ones we need to address carefully. One is how to effectively and efficiently archive, manage, process and deliver all those geoimage data. This real-time processing, management and distribution task is becoming overwhelming and we have to solve it intelligently. Another challenge is how to make the geoimages and professionally extracted information available to broader audiences so that a variety of businesses and mass consumers can benefit more from our work. In other words, geospatial and geoimaging technologies need a good platform to go mainstream. New research and technologies are needed in order to better solve those problems and the existing database and application server and client technologies must work synergistically in order to achieve those goals. This paper focuses on the database server technology, which is one of the key areas that is essential and foundational for solving the above two challenges and beyond.

It's well known that enterprise database management systems provide great data security, reliability, availability, recovery and backup, transactional features, versioning and concurrency, to name just a few. Because of these benefits, spatial database technologies based on standard relational database management systems (RDBMS) have become very popular in recent years. Good examples include RasDaMan/RasGEO (Baumann, 2001) and ArcSDE (ESRI, 2005). They typically take a middleware approach by storing data inside a standard RDBMS system and processing the data in a middleware or client software package. Most RDBMS's don't have image data types defined. So this approach requires a relational database schema to be designed to store the imagery inside RDBMS. However, a fixed set of relational tables specified in such application schema doesn't offer a good flexibility when it comes to integrate geoimage datasets with other enterprise datasets. The middleware acts as a processing engine, which queries the data from the RDBMS, retrieves the data out and then process the data in the middleware to serve the clients. Some extra data might have to be retrieved and delivered. Data transportation is expensive and could be insecure. So performance and data security are concerns with this approach. The other downside is either the lack of standard database SQL interface or the decoupling of its interface from the RDBMS system, which significantly limits the usability and enterprise integration efforts.

In this paper, we describe an enterprise database-centric approach for geospatial image and raster data management and processing. The key concept of this approach is to enhance and leverage enterprise IT technologies and provide a database-centric solution for image data management as well as key image processing operations. It not only offers the aforementioned standard database benefits but also goes one step further to provide more advantages to tackle the specific requirements derived from the two major challenges facing the geoimaging and geospatial industry.

## 2. THE ENTERPRISE DATABASE-CENTRIC APPROACH AND THE BENEFITS

While it's been proven to have many advantages and have become the industry trend to manage geoimagery data in commercial RDBMS systems instead of directly using file systems, we think the traditional RDBMS system itself should be enhanced to specifically handle geoimagery within the databases as well.

Traditional commercial RDBMS or ORDBMS systems and the SQL standards (such as SQL2 and SQL-99) typically support only simple data types and BLOB (Binary Large Objects). None of those standard data types meets the management requirements of geoimagery and raster gridded data. Even though the geoimage data can be directly stored as BLOB's in the databases, there is no standard operations developed to manipulate them inside the database system, and the standard SQL query language doesn't work the same way as for other simple data types. SQL/MM defines a data type SI_StillImage to store and manage still 2D images (ISO, 2001). However it stores images in one of the standard image file formats, which are best used to store only smaller images and are not specifically designed to support geoimagery and geospatial raster data types. Scalability and performance are concerns for both the simple BLOB approach and the SI_StillImage data type.

Geospatial imagery and raster data are typically huge in size and have many special metadata associated with them, such as coordinate system and georeferencing information. The operations on them are also different from other standard data types. In order to meet those special requirements, we propose an enterprise database-centric approach. This approach has uniquely a database-centric focus and uses server-based image processing concepts. By database-centric it means the raster data are stored and managed inside the database natively and the management and processing functionalities are implemented and embedded inside the database and closely and securely associated with the raster data itself. It's basically an enhancement of the RDBMS from inside.

More specifically, we think it should consist of three major components: a new native database data type for storing raster datasets, a server-side image processing and raster operation engine, and a standard user-friendly interface. It's designed to work in a client-server environment as well as in any multi-tier architecture.

The native object data type in this approach is specifically designed so that it can be used similarly as other standard database data types. The data model is generic for most raster data types, including geoimagery, so that each image can be stored as an object in any relational table. The specific format of the object type fits well into the enterprise RDBMS so that it's truly scalable and performant. For example, it allows flexible user-specified blocking, which means each image stored can be unlimited in size and adaptable to various applications. One database table can contain virtually unlimited number of images and various internal spatial indexing mechanisms enable fast metadata query and raster data retrieval.

This approach emphasizes a server-side image processing and raster operation engine. By doing that it offers true security for the data because the data no longer needs to be retrieved and loaded into a middleware or client through an insecure network and processed in an unmanaged computer memory. The processing engine is also closest to the data and so runs faster by avoiding data transferring cost. The processes can be run concurrently and deployed onto many powerful servers to reduce the burden on the desktop image processing systems. The processing engine can be coupled with middleware and client-side processing systems to fully leverage the power of enterprise distributed computing systems.

The approach offers a single data format and a SQL or PL/SQL API, which dramatically improves usability and simplifies data access. Usability is one of the key drivers behind this database-centric approach. SQL is the standard for enterprises and enterprise application developers are most familiar with it. By storing and managing the data inside the database, offering various indexing and query capabilities, and providing many basic processing operations through an easy-to-use and standard interface, this approach allows non-geoimaging experts easily integrate geospatial data with enterprise data, quickly leverage geospatial technologies, and deploy powerful IT resources so that the geoimagery and related information can be quickly delivered, distributed and used by different enterprises and mass consumers.

Oracle GeoRaster, an enterprise database management system for geospatial raster datasets, was designed based on this approach. To prove the concept of such a native database-centric approach, part of the design and some key benefits of Oracle GeoRaster are further described in the floowing sections of this paper. Some tests and research using the Oracle GeoRaster technology were conducted and are partially presented as well.

In the tests we used Oracle Database 10g Release 1, which was installed on Asianux 1.0 Service Pack 1. The Linux server has 4x 1G RAM, 4x 2.4GHz CPU, and 1x 72G internal hard disk. Network Appliance NearStore R200 system was used for database storage. It is a disk-based nearline storage system and provides near-primary storage performance at near-tape storage costs. The NetApp Storage consists of 16 disks (14 data disks + 2 parity disks, each disk is 292GB) combined into one global disk by RAID4. The test dataset includes 50 digital Color Ortho Images, courtesy of the Office of MassGIS, Commonwealth of Massachusetts Executive Office of Environmental Affairs. These 50 images cover the greater Boston area and can be seamlessly mosaicked into one large image. Each image has 8000 rows, 8000 columns and 3 bands and has a size of 183MB stored in TIFF format.

## 3. THE NATIVE RASTER DATA TYPE AND THE SCALABILITY

As described above, the first key component of this database-centric approach is a new native raster data type, which is called the GeoRaster data type in Oracle 10g and 11g databases (Oracle, 2004; Xie, 2008). Oracle GeoRaster defines a component-based logically layered multidimensional raster data model. A raster data object consists of raster cell data and associated metadata. The raster cell data is a multidimensional matrix of raster cells. Each cell stores a value, referred to as the cell value. The number of bits used to store the cell value is called the cell depth. The matrix has a number of dimensions, a cell depth, and a size for each dimension. As a multi-dimensional matrix, the core data can be blocked and compressed for optimal storage, retrieval and processing. In the GeoRaster data model, all associated information (other than the raster cell matrix) for the raster object is stored as "metadata", which include raster information, spatial reference system information, date and time information, layer information, and spatial extent (footprint) etc.

More specifically, a raster data (an image or a grid) is stored in Oracle as an object of the SDO_GEORASTER data type, called the GeoRaster object. This object type is the core data type for users and it stores all metadata and necessary information for indexing and raster data query. The type is defined as below:

```
CREATE TYPE sdo_georaster AS OBJECT (
rasterType NUMBER,
spatialExtent SDO_GEOMETRY,
rasterDataTable VARCHAR2(32),
rasterID NUMBER,
metadata XMLType);
```

The other data type is called SDO_RASTER, which is used to define the Raster Data Tables (RDT). The actual raster cell data of a large GeoRaster object are blocked into small blocks and each block is stored as one row in its RDT. The relationship between a GeoRaster object and its RDT and the raster cell data inside the RDT table are maintained and managed internally and automatically by enhancing the database server. Users only need to understand and deal with the SDO_GEORASTER object type (Figure 1).

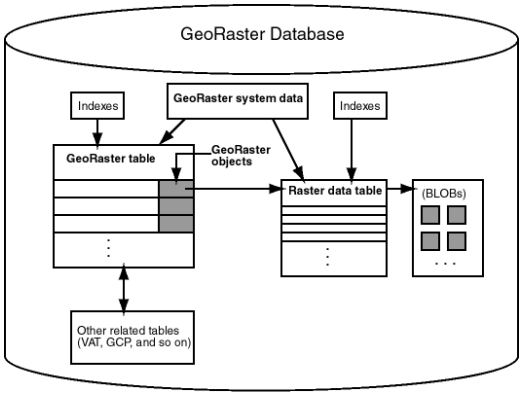


Figure 1. GeoRaster objects in an Oracle database (Oracle, 2004; Xie, 2008)

Using standard SQL language, a user can define any table and create one or more columns inside that table using the SDO_GEORASTER type. In Oracle database, a GeoRaster data table is any user-defined table, which has at least one data column of type SDO_GEORASTER. From a user perspective, a GeoRaster database is thus a list of GeoRaster tables, in which each image or raster grid is stored as a GeoRaster object in one row as shown in Figure 1. Users can build appropriate indexes on various columns of the GeoRaster tables e.g., a spatial R-tree index on the raster extent and B-tree indexes on other columns so that queries and other operations on the tables can be supported efficiently.

To build a GeoRaster database users simply create one or a list of GeoRaster tables using standard SQL statement. One example follows:

```
CREATE TABLE my_table

( id NUMBER PRIMARY KEY,
name VARCHAR2(50),
my_image SDO_GEORASTER);
```

Users are required to create RDT tables. The reason is purely to give users full control of the storage of the raster cell data so that appropriate tuning and partitioning can be applied to improve scalability and performance by leveraging the power of Oracle database server.

As described earlier, the first challenge is to make sure the image database management systems truly scalable. GeoRaster is completely built inside Oracle database server and the GeoRaster type is a native Oracle data type. Any table could contain one or more columns of the GeoRaster type. There is no limit on how many rows of an Oracle table could hold. A GeoRaster table is just a regular oracle table and so it could have unlimited number of rows. In each row a GeoRaster column can store one image. So, there is virtually no limit on the number of images you can store inside Oracle databases and the total size of such image database could be in petabytes.

Another key question is how big a single GeoRaster object (a single image inside the database) could be. For this we specifically did some tests and described them in (Xie, Li and Xu, 2006). We loaded the 50 DOQ images into the database and stored each of them as one GeoRaster object in the Oracle database. Then we mosaicked them into a single GeoRaster object with a size of 9.6GB. We enlarged the mosaicked image by using the GeoRaster procedure scaleCopy with "scaleFactor=11" along both the row and column dimensions. The size of the resulting image is 1.1616TB. Finally we generated the pyramids for the result image using the GeoRaster procedure generatePyramid. So we successfully ended up with a huge GeoRaster object of about 1.5 terabytes in size. All GeoRaster functions (the SQL API) passed tests on this huge image.

This test shows that with a proper database configuration, this approach allows creating, storing, and processing large raster datasets. Single GeoRaster objects can be in the scale of terabytes while the whole database can contain virtually unlimited number of images with a total size in the scale of petabytes.

## 4. THE SERVER-SIDE PROCESSING ENGINE AND THE PERFORMANCE

A robust data manipulation engine is another essential part of this approach and of any large-scale image database management system. This engine should cover data loading, exporting, insertions, updating, queries, deletions, analysis and processing. Besides leveraging the standard enterprise database features, GeoRaster provides over 100 raster data and metadata operations through the SQL API to optimally manage and manipulate GeoRaster objects in support of various application requirements (Oracle, 2004; Xie, Sharma and Ihm, 2007; Xie, 2008). The data processing includes internal blocking and interleaving format change, pyramiding, compressing, mosaicking, enlarging and shrinking, subsetting, band copying and merging, partial raster updates, and generating statistics.

The success of such a data processing engine relies on its security, scalability, and performance. The database-centric approach emphasizes a server-side image processing and raster operation engine that means all the processing and manipulations are implemented inside the Oracle database server and use the protected database memory system. By doing that the data no longer needs to be retrieved and loaded into a middleware or client through an insecure network and processed in an unmanaged computer memory. In other words, better performance and true security can be achieved. The ~1.5TB GeoRaster object was generated through the engine, by calling functions such as the loader, mosaic, enlarging and pyramiding. All existing functions can run on this big image and so demonstrated great scalability of this processing engine. It's achieved by adding to the database server a robust and scalable caching and memory management system specifically for GeoRaster object manipulation, no matter how big the physical image is.

To get an idea of how GeoRaster performs, we tested on the key tuning function, called changeFormatCopy. It can adjust the internal storage format based on user specifications, such as blocking size, interleaving type, cell depth and compression type. The size of the GeoRaster object used is 96M. It has 8000 rows, 4000 columns and 3 bands. The cell depth is 8 bit unsigned. The interleaving type is BIP. They are all three-band true-color images. In the test, the database buffer cache and shared pool are flushed to make sure every measure on the procedure is of a new execution of the procedure itself. The results are shown in Table 1. The tests only do reblocking of the GeoRaster object (no interleaving and other changes combined) and "noblock" means the image is not blocked, or in other words, the whole image is one block without padding. The blocking size is specified in the order of row, column, and band. For example, a blocking size of 512x512x3 means each block would have 512 pixels in row dimension, 512 pixels in column dimension and 3 bands.

| Original raster block size | New raster block size after changeFormatCopy | | | |
|---|---|---|---|---|
| | 256x256x3 | 1024x1024x3 | 2048x2048x3 | noblock |
| 64x64x3 | 59.90 | 57.94 | 57.97 | 58.54 |
| 128x128x3 | 49.31 | 48.96 | 59.29 | 55.50 |
| 256x256x3 | 42.74 | 54.51 | 49.43 | 54.05 |
| 512x512x3 | 52.47 | 42.72 | 44.68 | 47.63 |
| 1024x1024x3 | 47.00 | 37.65 | 40.48 | 45.98 |
| 2048X2048x3 | 46.10 | 41.09 | 37.61 | 42.90 |

Table 1. Average execution time in seconds to change internal raster block sizes

The changeFormatCopy procedure consists of two major processes. One is to change the format of the GeoRaster object. Another one is to make a copy of the original data. As you can see from Table 1, the speed is very fast. One observation is that when we call the procedure to change the block size to the same block size of the original object, the procedure is basically equivalent to copying the original blocks directly to the new object. Based on the data in Table 1, it means that most of the execution time is spent on simply copying data from one GeoRaster object to another while the data manipulation inside the database server is actually much faster. Like most other functions, this one is obviously I/O intensive and better I/O throughput would help in general. Efficient I/O operation is one of the key directions in improving performance.

Query is very important for such databases and the speed of such query is critical. So, we did some tests on the major GeoRaster query function, called getRasterSubset. It returns a subset of a GeoRaster object based on the query window, no matter how big the GeoRaster object is or how it is stored internally

| Retrieving window size | GeoRaster block size | | | | |
|---|---|---|---|---|---|
| | 128x128x3 | 256x256x3 | 512x512x3 | 1024x1024x3 | 2048x2048x3 |
| 256x256x3 | 0.3528 | 0.3194 | 0.3256 | 0.3178 | 0.3722 |
| 512x512x3 | 0.3970 | 0.3666 | 0.3424 | 0.3804 | 0.4478 |
| 1024x1024x3 | 0.5388 | 0.5068 | 0.4602 | 0.4766 | 0.5882 |
| 2048x2048x3 | 1.1450 | 0.9680 | 0.9028 | 0.8924 | 0.8848 |

Table 2. Average execution time in seconds to retrieve different subsets of raster data from differently blocked GeoRaster objects

We called getRasterSusbet 50 times continuously to get the average execution time. In this test, the buffer cache and shared pool were flushed out before every execution of the getRasterSubset procedure to make sure the measures are the pure running time of each independent call of the getRasterSubset and do not take advantages of database caches. We used various retrieving window sizes to retrieve data from the GeoRaster objects that have various GeoRaster block sizes. Results are shown in Table 2.

From this test, the AOI (area of interest) queries took only sub-seconds and showed very good performance. Properly tuning the blocking size may improve the performance as well. GeoRaster provides such tuning tool through functions like changeFormatCopy and so you can easily adjust the internal storage to meet specific application requirements.

## 5. THE USABILITY

The other challenge we discussed earlier in this paper is how to attract more users and push geoimagery into enterprise systems so that more people and more businesses can benefit from the achievements of the geoimaging and geospatial professionals. So, enterprise integration and usability becomes another key of the design of this database-centric approach.

Obviously the native data type and building the processing system from inside the commercial database server enable easy integration of enterprise data and geospatial raster data. It allows users to define a relational table in which images and other enterprise data can be stored together and those tables can be joined by defining their relationships. In order to build a single data type for different data sources, the native GeoRaster data type uses a single and integrated data model and thus simplifies the understanding and usage of such raster datasets. Even more important is the SQL API provided with GeoRaster.

SQL is the language of commercial database systems. Most Oracle database users and system integrators are familiar with the usage of the standard SQL and PL/SQL language. The language is simple, easy-to-use and has good modularity. It gives user the flexibility of data manipulations either through simple data queries or by packaging many functions into one package to achieve more complex goals.

For example, users can query the total band number of a GeoRaster object as follows:

```
select sdo_geor.getbanddimsize(t.my_image)
   from my_table t where id=21;
```

Users can simply write the following PL/SQL block to tune the block size of a GeoRaster object (geor1) and apply JPEG compression and store it in another GeoRaster object (geor2).

```
declare
  geor1 sdo_georaster;
  geor2 sdo_georaster;
begin
  select my_image into geor1 from my_table where id = 1;
  select my_image into geor2 from my_table where id = 2
       for update;
  sdo_geor.changeformatcopy(geor1,
       'blocksize=(256,256,3) compression=JPEG-B',
        geor2);
  update my_table set my_image=geor2 where id=2;
  commit;
end;
```

A more complex task could be that the user wants to find out all images (maybe hundreds or more) inside a specific region and then generate full pyramids for each of the images. The following simple PL/SQL block would do the work automatically.

```
declare
  type curtype is ref cursor;
  my_cursor curtype;
  stmt varchar2(1000);
  id     number;
  gr    sdo_georaster;
  gm  sdo_geometry;
begin
  -- 1. define the query area in WGS84 coordinate system
  gm := sdo_geometry(2003, 8307, null,
            sdo_elem_info_array(1,1003,3),
            sdo_ordinate_array(5,6,30,30));

  -- 2. define the query statement on the georaster table
  stmt := 'select id, t.my_image from my_table t ' ||
    'where sdo_inside(t.my_image.spatialextent, :1)="TRUE"';

  -- 3. spatially query all images INSIDE the query area
  --    and generate full pyramids for each of the images
  open my_cursor for stmt using gm;
  loop
    fetch my_cursor into id, gr;
    exit when my_cursor%NOTFOUND;
    sdo_geor.generatePyramid(gr, 'resampling=bilinear');
    execute immediate 'update my_table set my_image=:1
      where id=:2' using gr, id;
    commit;
  end loop;
  close my_cursor;
end;
```

Users can also wrap up such blocks into a PL/SQL procedure and store it in the database, then call the stored procedure directly. Such features enable users to organize complex processes and automate database administration tasks easily. This API enables non-geospatial experts understand such geospatial databases and the data manipulations and thus dramatically help expand the raster data usages in broader areas.

## 6. LEVERAGING IT INFRASTRUCTURE

Raster data processing and manipulations are computationally complex and I/O intensive. Single process might not be fast enough to meet the real-time archiving and processing requirements. Concurrent processing offers one of the best answers to such requirements. Tests show that concurrent processes drastically improve performance and scalability, including GeoRaster data loading, queries and processing (Xie, Li and Xu, 2006).

Since GeoRaster allows users to store raster data natively inside Oracle databases, it enables all benefits from Enterprise Computing technologies, such as the Oracle Enterprise GRID. From this standpoint of view, GeoRaster is an Enterprise GRID Computing enabler for the geospatial and geoimaging applications. Oracle enterprise GRID computing technology provides the benefits of lower cost, higher quality and flexibility, greater scalability and performance, and so on. In addition, the server-side processing engine, as an integral part of the database server, can be coupled with middleware and client processing systems to fully leverage the power of enterprise distributed computing systems. With a multi-tier architecture and the power of GRID computing, concurrent processing and parallelization can be readily available for raster image database management and processing.

## 7. CONCLUSIONS

In summary, this enterprise database-centric approach provides a foundation to help solve the two major challenges in a truly secure, scalable and performant way and offers an easy-to-use interface to empower non-geospatial professionals to manage and process geospatial raster datasets. The implementation of Oracle GeoRaster based on this database-centric approach and the tests we conducted show that this database-centric approach is a viable solution for geospatial image management and processing.

This approach focuses on the database server, in which the future directions include content-based indexing and search, componentizing the server-side image processing and query engine and storing them as database models, enhancing raster data analysis and mining, leveraging computing clusters and parallelizing image processing operations. It is not to discount the middleware image servers and desktop image processing systems or GIS systems. Instead the spatially enabled database server, middleware application servers and desktop image processing and GIS systems should complement each other and built on top of them a distributed system with a multi-tier architecture is the right direction.

## ACKNOWLEDGEMENTS

The authors would like to thank Zhun Li for conducting some of the tests presented in this paper.

## REFERENCES

ESRI, 2005. Raster Data in ArcSDE® 9.1 - An ESRI White Paper. http://www.esri.com/library/whitepapers/pdfs/arcsde91-raster.pdf (accessed April 2008)

ISO, 2001. ISO/IEC 13249-5:2001, Information technology - Database languages - SQL Multimedia and Application Packages - Part 5: Still Image, International Organization For Standardization.

Oracle, 2004. *Oracle Spatial GeoRaster, 10g Release 1 (10.1).* Oracle Corporation.

Baumann, P., 2001. Web-enabled Raster GIS Services for Large Image and Map Databases, 5th Int'l Workshop on Query Processing and Multimedia Issues in Distributed Systems (QPMIDS'2001), Munich, Germany, September 3-4, 2001

Xie, J., Z. Li, and W. Xu, 2006. Using Enterprise Grid Computing Technologies to Manage Large-Scale Geoimage And Raster Databases. In: *the Proceedings of ASPRS 2006 Annual Conference*, Reno, Nevada, May 1 – 5, 2006.

Xie, Q., J. Sharma, and J. Ihm, 2007. Oracle Spatial 11g GeoRaster, An Oracle Technical White Paper. http://www.oracle.com/technology/products/spatial/pdf/11g_collateral/spatial11g_georaster_twp.pdf (accessed April 2008)

Xie, Q., 2008. Oracle Spatial, Raster Data. *Encyclopedia of GIS*, Shashi Shekhar and Hui Xiong (editors), Springer. pp. 826 - 832.