GOOM Quick Reference Guide -

Managing Oracle Spatial Data for GeoMedia

July 2016

Chuck Woodbury
Senior Systems Consultant, VCA-DCV
Hexagon Safety and Infrastructure/Hexagon Geospatial
T +1 256.730.7755 M +1 256.616.9113
E **chuck.woodbury@intergraph.com**

TABLE OF CONTENTS

1. INTRODUCTION

The packages and scripts in GOOM.zip are my own personal toolkit and were initially created to help maintain the Oracle servers and spatial data warehouses used internally at Hexagon. These scripts and procedures help simplify spatial data maintenance when working with GeoMedia applications can be very useful for DBA's and database users new to Oracle's geospatial data storage. Just remember though, in all cases, these packages and scripts just calling procedures and functions that already exist in Oracle, primarily in the Locator component. These scripts and packages are *Open Source* and you are free to use and modify them to suit your needs as long as you always include the OS legal disclaimer in the header. The disclaimer makes us all a little safer when dealing with open source software

This document also provides reference information on configuring and maintaining an Oracle spatial schema for use with GeoMedia applications. It contains basic instructions for working with the GOOM and GDOBKP packages and with the accompanying SQL scripts contained in GOOM.zip. It is by no means comprehensive and is strictly designed to be a quick reference guide.

I hope you find this document and accompanying packages and scripts useful when working with Oracle spatial schemas in a GeoMedia environment.

Note: The contents of GOOM.zip are not officially sanctioned by Hexagon Safety and Infrastructure or Hexagon Geospatial. DO NOT log support tickets if you run into issues. For bug reports, general questions, or even comments, please Email Me directly – chuck.woodbury@hexagonsi.com.

GOOM.ZIP

If you have not already done so, you should create a directory that will hold all of the Oracle SQL and PL/SQL scripts you plan to use. I use *D:\DBAORA\SQL* but any directory that DOES NOT CONTAIN SPACES IN ITS PATH will work. Unzip G*OOM.zip* in your local SQL directory and it will create a GOOM directory containing a collection of scripts as well as six directories:

• Datatypes:	Scripts regarding data type mapping and overrides.
• DBAOnly:	Scripts for DBA's.
• ImpExp:	Scripts for exporting and importing schemas with GeoMedia metadata.
• Packages:	The main packages: GOOM, GDOBKP, GDODATA, GDORaster, and GTMUTIL.
• Tracing:	Oracle tracing procedures and scripts, and TKProf Scripts for analyzing traces.
• Triggers:	GeoMedia trigger examples.

The procedures and functions found in the GOOM packages can be used from anywhere in the database, including from your own scripts. The SQL scripts in GOOM are designed to be used in SQLPlus and may not work correctly in TOAD or SQL Developer but the packages should work anywhere. Both the packages and scripts will provide feedback as long as you have SERVEROUTUT turned on in SQLPlus (SET SERVEROUTPUT ON).

Configuring SQLPlus

If you plan on using the scripts, avoid using SQL*Developer or TOAD as these have limitations when using scripts that require prompting. The SQL command window is probably the easiest to use and provides for drag & drop, cut & pastes, and other editing capability that is missing from the SQLPlus interface. For simple database querying and DDL, make use of SQL*Developer or any third party app like TOAD, they make browsing the database much easier and faster. .

To configure the command window for use with Oracle, you can do the following:

1. Create a shortcut on the desktop and enter **cmd** in the location field. The shortcut should automatically point to cmd.exe.
2. Name the shortcut *SQLPlus* and select OK.
3. Right Click on the shortcut and select *Properties* from the drop down menu.
4. To make sure SQLPlus is initialized, append the following to the target field:
   /K sqlplus.exe
   The target field should look like:
   %windir%\system32\cmd.exe /K sqlplus.exe
5. Under the **Shortcut** tab, set the *Start in:* field to the directory containing your scripts. In my case, the *Start in:* field is set to *D:\DBAORA\SQL.*
6. Under the **Options** tab, enable the **Quick Edit** mode.
7. Under the **Layout** tab, increase the screen buffer size to a width of *300* and a height of *300*. This is optional but helps when examining data in the window.

Now you can connect to you database anytime by selecting the SQLPLUS shortcut and entering your connection parameters.

Another useful script is *login.sql*. If this is in the startup directory for SQLPlus, it will automatically be used to set SQL parameters for you. The *login.sql* that I use is in the GOOM directory. If you want to use it, copy it to the startup directory used above and then set the parameters in this file to suit your needs.

Using SQL Developer

Oracle SQL Developer is a **free** java based graphical tool for database development. With SQL Developer, you can browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements. You can also run any number of provided reports, as well as create and save your own. If you have an 11g (or later) client, you already have SQL Developer otherwise; you can get SQL Developer from Oracle. Whether you are a novice or an experienced Oracle DBA, SQL Developer can be a real useful tool.

2. GDOSYS – GEOMEDIA'S METADATA SCHEMA

GeoMedia Applications utilize a metadata schema called **GDOSYS**. For read-only connections, the GDOSYS schema is not officially required, but in my experience you should always use GDOSYS. For read-write connections, the GDOSYS schema is required and must contain a minimum set of required metadata tables. Only one GDOSYS schema is allowed per database instance and it holds the metadata information for all the schemas that will be used with GeoMedia applications.

Creating GDOSYS

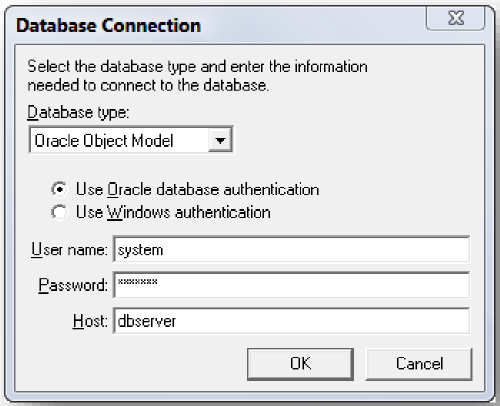
Typically, the GDOSYS schema is created using **Database Utilities** which calls the SQL script *CreateGDOSYS.sql* that is delivered in the *C:\Program Files (x86)\Common Files\Intergraph\GeoMedia\Scripts* directory (post 2012). If you are interested in what is going on behind the scenes, you can review this file. You can also make modifications to a copy of this file if you need to make some adjustments. Use care when making changes though, GeoMedia expects specific metadata tables and you can have unpredictable results if those tables are not present.

To create the GDOSYS schema using **Database Utilities**, do the following:

1. Go to **Start>>GeoMedia Professional >Utilities>Database Utilities** to access this utility.
2. Set the Database type: *Oracle Object Model*
3. Since you are creating the GDOSYS schema, you need to connect as DBA. For the User name, Password, and Host string, enter the following:

   System/password@dbserver

   where dbserver is the TNS name of the database server you are using. The easy connect method will also work: *user/password@server:port/SID*



Database Connection

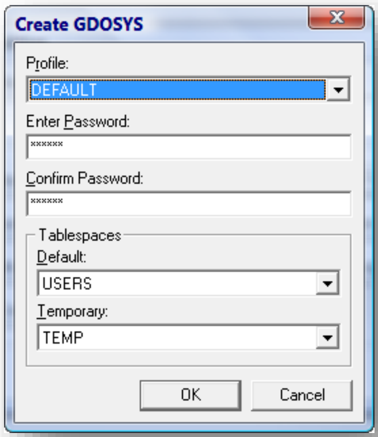
Select the database type and enter the information needed to connect to the database.

Database type: Oracle Object Model

(●) Use Oracle database authentication
( ) Use Windows authentication

User name: system
Password: *******
Host: dbserver

4. To create the GDOSYS metadata schema, select the **Create Metadata Tables** button.
5. You will be prompted for the following information:
6. The profile is the Oracle user profile to use, typically you would use the default profile DEFAULT.
7. The password field is for the GDOSYS user password. By default, the password for GDOSYS is GDOSYS but you can change it to suit your needs. You will need to confirm the password entry for security.
8. You can use the USERS tablespace or any other named tablespace for the GDOSYS schema. The temporary tablespace is TEMP by default.



Create GDOSYS

Profile: DEFAULT
Enter Password: ******
Confirm Password: ******
Tablespaces
Default: USERS
Temporary: TEMP

NOTE: For best results, create a tablespace to hold the GDOSYS schema. The tablespace can be fairly small; I usually start with 20MB. Make allowances for growth though.

9. Run the process by selecting OK then verify the creation of GDOSYS via SQLPlus or some other tool.

DBA's can also use the *CreateGDOSYS.sql* script to manually create the GDOSYS schema. Move the script to your SQL directory, connect as a DBA user, and then run:

```
SQL> @CreateGDOSYS <GDOSYS_PSWD> <ORA_PROFILE> <USER_TABLESPACE> <TEMP_TABLESPACE>
```

For Example:

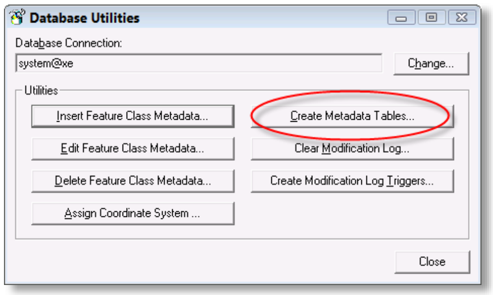
```
SQL> @CreateGDOSYS GDOSYS DEFAULT USERS TEMP
```

Or if you are using a GDOSYS tablespace:

```
SQL> @CreateGDOSYS GDOSYS DEFAULT GDOSYS TEMP
```

Updating GDOSYS

As new versions of GeoMedia applications are released, there may be changes to the GDOSYS schema definition. Basic changes are incorporated into **Database Utilities** which can be used to update GDOSYS as new releases come out. Whenever you get a new release of GeoMedia, you should perform the following steps:


Database Utilities
Database Connection:
Change...
Utilities
Insert Feature Class Metadata...
Create Metadata Tables...
Edit Feature Class Metadata...
Clear Modification Log...
Delete Feature Class Metadata...
Create Modification Log Triggers...
Assign Coordinate System ...
Close

system@xe

1. Go to **Start>>GeoMedia Professional>>Utilities>> Database Utilities** to access this utility.
2. Set the Database type: *Oracle Object Model*
3. Connect as a DBA user like SYSTEM.
4. To update the *GDOSYS* metadata schema, select the **Create Metadata Tables** button.
5. Since GDOSYS already exists, the process will check the existing metadata and update it if required. You will be notified whether or not an update occurs. The same process can be used to repair GDOSYS metadata if there are missing tables. This will not repopulate any of the tables but will add them back to the structure as needed.

A script called *UpdateGDOSYS.sql* is also included in the *C:\Program Files (x86)\Common Files\Intergraph\GeoMedia\Scripts* directory that a DBA user can use to upgrade GDOSYS. *UpdateGDOSYS.sql* is usually a more comprehensive upgrade than using **Database Utilities** but it will not detect or repair any metadata problems.

Note: Review the upgrade script before using it. This script may make changes that you may find undesirable. These include GDOSYS privilege changes as well as changes to GEXCLUSIONS.

3. INSTALLING UTILITY PACKAGES - GOOM AND GDOBKP

The various scripts described in this document depend on a set of functions and procedures created specifically for use with GeoMedia and Oracle. These procedures and functions collectively reside in an Oracle package called *GOOM* (GeoMedia Oracle Object Model). The *GOOM* package contains procedures and functions that simplify the process of configuring and maintaining data in an Oracle spatial schema. These include the repetitive tasks involved in assigning metadata, validating data, spatial indexing, and verifying metadata.

Because the metadata required by GeoMedia applications resides in the centralized schema GDOSYS, retaining the metadata settings for tables in individual schemas can be difficult during backup or export operations. The GDOBKP package was created to facilitate metadata backups by storing the metadata settings for individual tables (and views) in the schema that owns the tables. The GDOBKP procedures apply to standard GDOSYS metadata; it will not backup industry application specific metadata (like AFM).

Installing GOOM

The *GOOM* package needs to be installed by a DBA user before you can use any of the scripts mentioned in this document. Prior to installing GOOM, there are a number of parameters you can set in the package header, or you can use the values that are already set by default:

Parameter	Datatype (with default)	Description
`c_indxspace`	`VARCHAR2(30):='USERS';`	Change this variable to tablespace you want spatial indexing to use if INDX is not available. You can also use SetGOOMIndexSpace to set the index tablespace to use for spatial indexes before you run the SpatialIndex procedure.
`c_tol`	`NUMBER:=0.00005;`	Sets the spatial tolerance used by Oracle. If you are using GM, c_tol should never be larger than 0.0001 or smaller than 1/2 GM: 0.00005
`c_gtol`	`NUMBER:= 0.05;`	Sets the tolerance used by Oracle in Geodetic Calculations. Oracle looks at this value in meters, 0.05 is smallest allowed for most calculations.
`c_defDim`	`PLS_INTEGER  := 3;`	Default feature class dimension: 2 for 2D, 3 for 3D, user when working with empty geometry columns.
`c_logging`	`VARCHAR2(3)  :='NO';`	Turns on process logging to the table OWNER.GOOM_LOG. Set to YES enables logging.
`c_valext`	`VARCHAR2(4)  := '_ERR';`	Geometry validation table name extension.
`c_idxext`	`VARCHAR2(4)  := '_SI';`	Spatial index name extension (table_SI)

Parameter	Datatype (with default)	Description
c_PKext	VARCHAR2(4) := '_PK';	Primary Key constraint name extension. Max 4 chars!
c_sample	PLS_INTEGER := 1000;	Commands like GetSRID have to look at more than one record but they do not have to look at all records. Use this parameter to limit the number of records sampled. A smaller number means better performance but less accurate results.

c_XLO
c_YLO
c_ZLO
c_XHI
c_YHI
c_ZHI
	NUMBER := -2147483648;
	NUMBER := -2147483648;
	NUMBER := -2147483648;
	NUMBER := 2147483648;
	NUMBER := 2147483648;
	NUMBER := 2147483648;
		Default Extents for SetMBR and SetMBRProj (USER_SDO_GEOM_METADATA).
		
		Set these if you only work in a specific area and you want to utilize Zoom to Extents capability with your data. GM apps do not use this so in that case, you do not need to change these.

To install the GOOM package, connect to Oracle as a system DBA and execute the following SQL:
@GOOM_PKG.sql

The package procedures and functions will be loaded to the *GDOSYS* schema with execution granted to PUBLIC. A PUBLIC synonym called GOOM is created to ensure that the *GOOM* package procedures and functions are available to all users that require the capability. This is not required but makes things easier.

IMPORTANT: Most of the procedures in GOOM will provide feedback to the SQL user, to enable feedback, make sure you enter the following at the SQLPlus prompt (or set this in login.sql):

SET SERVEROUTPUT ON

GOOM undergoes frequent updates, to find out the version you are currently using, you can enter the following at the SQL prompt:

EXEC GOOM.VERSION;

When creating spatial indexes, the *GOOM* package will use the *INDX* tablespace by default, if INDX does not exist, it will default to the tablespace specified in *c_tablespace*. If you have another preference, you can specify the tablespace where you want GOOM to create spatial indexes by running the following at the SQL prompt while connected as a DBA user:

EXEC GOOM.SetGOOMIndexSpace('<INDX_SPACE>');

where *<INDX_SPACE>* is the name of the tablespace where you want the spatial indexes to be created

To determine what tablespace is currently being used, you can run the following:

SELECT GOOM.GetGOOMIndexSpace FROM DUAL;

For a list of available commands and functions, enter the following at the SQL prompt:
EXEC GOOM.HELPME;

For detailed descriptions and syntax for these procedures, enter the following:

```
EXEC GOOM.HelpDATA;           -- Spatial Data Manipulation
EXEC GOOM.HelpMBR;            -- Oracle metadata procedures.
EXEC GOOM.HelpGDOSYS;         -- GeoMedia metadata (GDOSYS) procedures.
EXEC GOOM.HelpTUNING;         -- Spatial Indexing and tuning.
EXEC GOOM.HelpUTILITIES;      -- Miscellaneous useful stuff.
EXEC GOOM.HelpVALIDATION;     -- Validation and repair procedures.
EXEC GOOM.HelpFUNCTIONS;      -- GOOM utility functions.
```

Installing GDOBKP

The GDOBKP package requires the GOOM package which must be installed first. The *GDOBKP* package needs to be installed by a system DBA. To do that, connect to Oracle as a system DBA and execute the following SQL:

@GDOBKP_PKG.sql

The package procedures and functions will be loaded to the *GDOSYS* schema with execution granted to PUBLIC. A PUBLIC synonym called GDOBKP is created to ensure that the package procedures are available to all users that require the capability. This is not required but makes things easier. To find out the version of GDOBKP you are currently using, you can enter the following at the SQL prompt:

EXEC GDOBKP.VERSION;

For a list of available commands and functions, enter the following at the SQL prompt:
EXEC GDOBKP.HELPME;

Installing GTMUTIL

The GTMUTIL package is a collection of procedures and functions that are meant as additions to the GTM package used by GeoMedia Transaction Manager. Both the GOOM and GTM packages are required in order to use it. This package includes extra revision set procedures, spatial indexing procedures for version-enabled tables, and privilege assignment procedures.

The GTMUTIL package requires that both the GTM package (delivered with GeoMedia) and the GOOM package be installed first. The *GTMUTIL* package needs to be installed by a system DBA. To do that, connect to Oracle as a system DBA and execute the following SQL:

@GTMUTIL_PKG.sql

The package procedures and functions will be loaded to the *GDOSYS* schema with execution granted to PUBLIC. A PUBLIC synonym called GTMUTIL is created to ensure that the package procedures are available to all users that require the capability. This is not required but makes things easier. To find out the version of GDOBKP you are currently using, you can enter the following at the SQL prompt:

```
EXEC GTMUTIL.VERSION;
```

For a list of available commands and functions, enter the following at the SQL prompt:

```
EXEC GTMUTIL.HELPME;
```

Other Packages

Two other packages are available in the GOOM/Package directory:

- GDODATA: Spatial data generation procedures and functions
- GDORaster: Procedures and functions for GeoMedia's GeoRaster capability.

These packages are optional and you do not need to load them unless you want them. These packages all require GOOM to be installed first. GDORaster will require Oracle spatial and not just Oracle locator. In all cases, online help is available via the respective HELPME procedures (this only works if SERVEROUTPUT is ON):

```
EXEC GDODATA.HELPME;
EXEC GDORASTER.HELPME;
```

The GDODATA package contains procedures and functions that will create test tables and automatically populate them with random data based on the default coordinate system of the schema. The package can generate random attribute data as well as point, line, and area data. For example, to generate a quick test table with 100 records and an integer based primary key, you can run one of the following:

```
EXEC GDODATA.GeneratePtTable('MY_PT_TABLE',100);
EXEC GDODATA.GenerateLineTable('MY_LINE_TABLE',100);
EXEC GDODATA.GeneratePolyTable('MY_POLY_TABLE',100);
```

Both GDOSYS and Oracle metadata are added automatically and the data is also spatially indexed.

The GDORASTER package has procedures and functions to simplify the loading of GEOTIFF images into Oracle's SDO_GEORASTER format. This package requires Oracle 11G or later and the Oracle Spatial component must also be present.

4. CREATING ORACLE USER ACCOUNTS

As a DBA, you have a lot of latitude in the creation of user accounts in Oracle. What privileges you assign to the user account is dependent on what purpose the account will serve. To create a user account that will hold spatial data (a master user account), you typically will need to do the following while connected to SQL as a DBA:

```
CREATE USER <user> PROFILE "DEFAULT" IDENTIFIED BY <pswd>
    DEFAULT TABLESPACE USERS
    TEMPORARY TABLESPACE TEMP
    ACCOUNT UNLOCK;
```

where <user> is the username you want to create and <pswd> is the password for that user account. In the example above, I am using the USERS tablespace but any standard tablespace will work.

Since this is the account that will contain the spatial data, tables, views and other objects will have to be created in this schema. The following privileges will give you that capability; you will need to enter them as a DBA using the user name created above:

```
GRANT UNLIMITED TABLESPACE TO <user>;  DBA's discretion
GRANT CONNECT TO <user>;
GRANT RESOURCE TO <user>;
GRANT CREATE TABLE TO <user>;
GRANT CREATE SEQUENCE TO <user>;
```

You need to include the following privilege for spatial data users, or, set it at the database initialization level:

```
GRANT MERGE ANY VIEW TO <user>;
```

You can avoid having to set MERGE ANY VIEW for each schema by changing the database initialization parameter *optimizer_secure_view_merging* to FALSE (TRUE is default). This is highly recommended if you regularly use spatial data.

The UNLIMITED TABLESPACE role is optional; use it if you want. The CONNECT and RESOURCE roles are common when creating objects in a schema and are required. The CREATE TABLE and CREATE SEQUENCE privileges are required in order to use the PL/SQL functions and procedures in both MDSYS and in the GOOM package. These privileges cannot be added to a role, they have to be explicitly assigned. MERGE ANY VIEW is a security enhancement that affects the performance of spatial data. You must set the either the privilege or the global initialization parameter. Failure to do this will result in terrible spatial performance.

Caveats

Two main rules apply to user account creation:

- Never create a user account in the SYSTEM Tablespace. Typically, a user is created in the USER tablespace with temporary storage in TEMP. Any named tablespace will work here. How you design your tablespace structure depends on the hardware you are using and how the database will actually be used.

- Never assign DBA privileges to a master or satellite user account if you plan to use it with GeoMedia. This has serious performance implications because GeoMedia's Oracle data server will scan all tables and views that the user has privileges to see (a DBA sees everything in the database).

Master/Satellite Account Privileges

Satellite user accounts by definition contain no database objects. They exist to allow a specific user varying degrees of privilege on the master schema, which hold the database objects (tables, views, etc.). All users that need to access a spatial schema need to have, at a minimum, SELECT privilege on all the objects in GDOSYS and in the master schema. In addition to SELECT, here are some typical configuration and the required privileges:

- Master User and Schema – owns all objects
  - `@DBAOnly/dbadduser` You can use this script to create a master schema, make sure you change the password though.
- Admin User – Has full control of master schema
  - GRANT ALL on all database objects in GDOSYS and in the master schema.
  - `@GrantRW` You can use this script to grant the required privileges.
- Viewing User – Read-Only access to all objects
  - Select on all GDOSYS objects
  - Select on all master schema objects
  - `@GrantRO` You can use this script to grant the required privileges.
- Editing User - Has edit capability on some feature classes but not others and cannot remove or delete data.
  - Grant INSERT and UPDATE on GDOSYS's MODIFIEDTABLES AND MODIFICATIONLOG
  - Insert and update on all Read-Write tables and sequences in master schema.
  - `@GrantEdit` You can use this script to grant the required privileges.

Run the following from the master schema to revoke privileges for satellite accounts:

```
@GrantRevoke
```

You will be prompted to enter the user you want to revoke privileges from.

Creating a user account via a script

The script *DBADDUSER.sql* (in the DBAOnly directory) provides an automated way of creating a master schema. This script must be run from a DBA account:

```
@DBAddUser <user>
```

where *<user>* is the name of the user account you want to create. By default, this script will create a user account that has the same password as the username and put it in the USERS tablespace. The password can be changed later if you want (using SQL's PASSWORD command). Modify the *dbadduser.sql* script to suit your needs. *Dbcreateuser.sql* is also available and it will let you specify specific tablespace for the user account.

Privileges on GDOSYS are usually granted to PUBLIC however, you can use @GrantGDO to grant specific privileges on GDOSYS to a specified user. The options here are Full, Editor, and Read-only.

Use the above privileges as a guideline, you have a lot of leeway here as DBA.

5. EXPORTING AND IMPORTING SPATIAL DATA VIA GEOMEDIA

GeoMedia Professional provides two methods for moving data into Oracle; **Export to Oracle Object Model** and **Output to Feature Classes**. For large datasets, the quickest way to migrate data to Oracle is to use GeoMedia Pro's **Export to Oracle Object Model** command. If the original source data is in a format that GeoMedia can understand, then the export command will allow you to quickly transfer the data into an Oracle spatial schema.

When using **Export to Oracle Object Model**, you will need to connect to the warehouse containing the data you want to bring into Oracle spatial. If it is already in a GeoMedia warehouse, it is probably in good shape. If it is coming from an MGE based project or from CAD based data, the data must be clean before it is loaded into Oracle. The effect of bad data will be magnified in the Oracle and may cause spatial indexing to fail. The most import thing to remember is to start with clean data. Point and linear features are generally not an issue but Area features will need to be *complexed* before you can bring them over.

NOTE: Pay attention to the coordinate system you are using during the export process! **Export to Oracle Object Model** uses the coordinate system of the *GeoWorkspace*. Verify that the coordinate system you intend to use is assigned to the GeoWorkspace before proceeding with the export process. **Output to Feature Classes** allows some more flexibility when selecting what coordinate system to use, just make sure to verify it before running the process.

The **Output to Feature Classes** command in GeoMedia Pro is a lot more flexible than **Export to Oracle Object Model** but it is a lot slower. This is the tradeoff you get with flexibility. When using **Output to Feature Classes**, you can choose the columns you want, the coordinate system to use, and can even make modifications to both the table name and the column names. Options are available for preserving the definition and values of existing primary key columns, or creating new primary key columns. New tables can be created and loaded or existing tables can be appended to.

If the input data violates any of Oracle's rules for storing spatial data, OTFC will return an error when the spatial index is updated. **Export to Oracle Object Model** will load data regardless of the data's condition and the table containing the spatial data will not be indexed. Spatial indexing is required though and that indexing will fail if any of the data is invalid. Use GOOM's *ValidateGeom* procedure or run the script *Validate* to check for problems in the imported data. You have to fix any data problems before the imported data is usable by both Oracle and GeoMedia. Depending on the problems found, this can be a difficult task and may require you to delete the offending row.

Note: Do not use Export to OOM or OTFC to move data from one Oracle schema to another. There is no round trip data type conversion. The data type you start with will not necessarily be the same data type you end up with. There are better methods within Oracle for doing this type of data transfer. Look at using the GDOBKP package to preserve the GDOSYS metadata for your schema and then use Oracle's imp/exp or impdp/expdp commands to migrate existing Oracle tables.

6. VALIDATING SPATIAL GEOMETRIES

If you load data using Oracle's SQL*Loader which is the format used by GeoMedia's **Export to Oracle Object Model**, you need to validate the data before it can be spatially indexed. The validation process ensures that the data loaded correctly and that there are no data related problems that would inhibit the creation of a spatial index. Oracle provides several commands for validating data; the best is `SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT.`

This procedure will identify the *ROWID* containing the offending element. The GOOM procedure, ValidateGeom, simplifies the use of *validate_layer_with_context* and will generate a *<table>_ERR* table, containing all the errors it encountered for the specific feature class. There are two ways to use GOOM's ValidateGeom procedure, both validate your data and create the table for validation results:

```
EXEC GOOM.ValidateGeom('<table_name>','<column_name>');
```

or use a script; in SQLPlus, connect to your Oracle master schema and run the following:

```
@Validate
```

You will be prompted to enter the table name and then the geometry column name that you wish to validate. All geometry columns that exist in the database must be validated prior to indexing the data. The results will be loaded in <table>_ERR (e.g. COUNTIES_ERR).

NOTE: ValidateGeom will now detect 11g and then utilize a flag10G parameter to ensure that any use of arcs in a 3D geometry will pass validation.

Oracle recommends fixing all issues found by the validate process. To review the *validation results,* use the following script:

`@ListValResults` for all validation results

This will prompt you to enter the name of the table containing the validation results (typically *<table>_err*).

To examine any row containing a bad geometry, you could use an SQL statement similar to the following example:

```
SELECT GEOMETRY from COUNTIES where ROWID='<rowid>';
```

AnalyzeGeometry is a new function that looks at a single geometry value and tests it for validity. If it's invalid, a detailed summary will be returned to standard output. If you want the detailed analysis regardless of an error condition, set the v_vebose flag to TRUE (the default).

```
SELECT GOOM.AnalyzeGeometry(g_geometry) FROM table WHERE PKID=val;
```

Another way of analyzing a row of geometry is to use the following SQL script, which formats the geometry column for easier reading:

```
@ShowGeomByROWID;
```

SDO_GEOMETRY columns have to be initialized before they can be used. An uninitialized geometry is considered an atomic NULL. A geometry column can be empty but atomic NULLs will cause an error in spatial indexing. This is a common problem that occurs when loading geometry data in using SQL*Loader. SQL*Loader will create uninitialized geometry objects when the record it is trying to insert is NULL. To fix this problem, you need to reinsert a NULL into the entire geometry object. The following SQL is an example of how to do this for an individual table:

```
UPDATE CITIES A SET A.GEOMETRY=NULL
 WHERE A.GEOMETRY.SDO_GTYPE IS NULL;

COMMIT;
```

Since all tables loaded via SQL*Loader can potentially have this problem, the easiest way to solve this problem is to automatically correct all feature classes in the master schema. The following GOOM procedure is provided for this purpose (run this while connected to your master schema):

```
EXEC GOOM.FixNullGeoms;
```

Another problem that can occur when importing data into Oracle via GeoMedia is the right padding of character columns. This is especially prevalent if the source data is an older Access warehouse. To fix this, you will need to right trim the character columns using the SQL RTRIM function. You can do this automatically using the following:

```
EXEC GOOM.FixTrailingSpaces;
```

Very small arcs can also cause problems. These arcs are typically generated from CAD data and are usually less that 1 cm in diameter. If these are an issue, you can use the following procedure:

```
EXEC GOOM.FixSmallArcs('<TABLE_NAME>','<GEOMETRY_COL>',[LENGTH],[TOLERANCE]));
```

Another common issue is redundant points. This just means that you have a lot of vertices in the geometry that fall within the specified Oracle tolerance. Removing redundant points will improve performance and in Oracle 11g, it is required. Both Oracle and GeoMedia Professional provide a method to remove redundant points. To use Oracle's procedure, run the GOOM package procedure:

```
EXEC GOOM.FixRedundantPoints('<TABLE_NAME>','<GEOMETRY_COL>',[TOLERANCE]);
```

or use the script:

```
@FixRedundantPoints
```

and follow the prompts.

For other data related problems, you may be able to use GeoMedia's geometry validation commands to clean up the problems. There is a product called GeoMedia Fusion that contains more advanced geometry cleaning capability.

Some critical geometry problems may not be solvable. Since you know the ROWID's of the bad features, you can attempt to display them in GeoMedia using an attribute query or by using an Oracle view. If you can display the feature in the map window, you can use GeoMedia's validation and repair commands to fix the bad geometry. GeoMedia Fusion also contains advanced geometry validation and repair routines that may help here.

If the geometry is so bad that even GeoMedia will not read it, then you have only two options; 1) fix the problem in the original data and re-import or 2) remove the offending geometry. To delete a bad geometry from your table, you could use the following syntax`:

```
UPDATE <feature_class> A SET A.<geom_column>=NULL
WHERE ROWID=<rowid>;
```

The benefit of this SQL statement is that it removes the offending geometry but preserves the contents of the other attribute columns that may exist in the row. One example of this type of problem is collinear arcs. They can be very difficult to fix.

7. ORACLE METADATA – SETTING THE MBR AND TOLERANCE FOR GEOMETRIES

All columns of type SDO_GEOMETRY require entries in Oracle's spatial metadata table: SDO_GEOM_METADATA_TABLE which is stored in MDSYS. This table is used to store the spatial tolerance and spatial extents (minimum bounding rectangle – MBR) of each spatial column in the database. Oracle provides access to this metadata table via 2 views:

- ALL_SDO_GEOM_METADATA - Lists metadata for all tables the user has privilege to see (SELECT). This view is read-only which means that a user must own the table in order to populate Oracle's metadata for that table.
- USER_SDO_GEOM_METADATA – Lists metadata for the tables the user owns. This view is fully writable allowing users to insert metadata for tables in their schema.

Normally, applications update Oracle's metadata via USER_SDO_METADATA, which is writable, and access the metadata via ALL_SDO_GEOM_METADATA, which is read-only (as is its parent table). You can get around this restriction by granting insert privilege in MDSYS.SDO_GEOM_METADATA_TABLE to the user (or to PUBLIC) that requires the capability. This is not recommended but may be necessary in cases where you want a user to create tables in the master schema.

GeoMedia requires all feature classes (even those that are views) to have entries in the MDSYS.SDO_GEOM_METADATA_TABLE. During a connection, it will read from ALL_SDO_GEOM_METADATA to get the information it requires. When running a GeoMedia command that creates a table in Oracle, GeoMedia will first try to populate the MDSYS.SDO_GEOM_METADATA_TABLE and if that fails, it will next try to populate USER_SDO_GEOM_METADATA. If you are connected as a user that does not own the table you are creating and proper privileges have not been established, the process will fail. All columns of type SDO_GEOMETRY require Oracle metadata. This includes views (a GeoMedia requirement, not Oracle).

For projected data, the default extents of +-2147483648 for X,Y, and Z work best for most data cases. The tolerance is the most important consideration here as it can impact both validation and spatial analysis. GeoMedia's map tolerance is hard coded at 0.1 mm which means map coordinates less than .1 mm apart are considered to be equal. The equivalent tolerance in Oracle is 0.00005 meters and typically, this is the value that should be used. You should not make the tolerance greater than GeoMedia's and typically you should not be smaller than ½ of GeoMedia's tolerance.

For geodetic data, the default range for X and Y MUST BE +-180, +-90 and the tolerance is always expressed in meters. DO NOT USE the projected range if your data is geodetic. The default tolerance for geodetic data is 0.05 meters.

The GOOM package contains several functions that will automatically populate USER_SDO_GEOM_METADATA with default extents for all the geometry columns contained by the feature classes in your schema.

- To automatically set the metadata based on analysis of the geometry and SRID, use the following:
  ```
  EXEC GOOM.SetMBR('<table_name>','<geom._column>');
  ```
  Or
  ```
  EXEC GOOM.SetMBRAll; or EXEC GOOM.SetMBRAll(c_owner);
  ```

to automatically set the MBR for all feature classes in a schema. The use of OWNER.TABLE syntax in SetMBR or the use of c_owner in SetMBRAll requires write privilege on MDSYS.SDO_GEOM_METADATA_TABLE to the user on executing the command. This is not a problem for the DBA.

- For geometry columns that will contain projected data, use the following:
  `EXEC GOOM.SetMBRProj;` Sets the extents to +-2147483648 for all feature classes in the schema.
- For geometry columns that will contain non-projected (Long./Lat.) geometries, use the following:
  `EXEC GOOM.SetMBRGeo;` Sets the extents to +-180, +-90 for all feature classes in the schema
- For individual table, just use the table name, for example:
  `EXEC GOOM.SetMBRProj('<tablename>'); or`
  `EXEC GOOM.SetMBRGeo('<tablename>');`

If you want to copy the MBR values from one geometry column to another, you can use the following:

```
EXEC GOOM.CopyMBR('source_table','source_geom','target_table','target_geom');
```

This is particularly useful for views.

The tolerance setting in Oracle's metadata is the most important value and should be no less than ½ of GeoMedia's fixed tolerance of 0.0001. To see what you current tolerance setting is, you can use the following:

```
SELECT GOOM.GetSpatialTolerance(c_tablename,c_geometry,[c_dim]) FROM DUAL;
```
c_dim is optional and can be 'X', 'Y' or 'Z' if tolerances are different.

To set the tolerance, the following procedure is available:

```
EXEC GOOM.SetSpatialTolerance(c_tablename,c_geometry,[n_tol]);
```
Leave n_tol blank to set the default of 0.00005

There are also a number of scripts available that you can use here:

- To calculate the exact extents of a single feature class, use the following script:
  `@MBR_Calc`
  Using the results from *MBR_Calc*, you can manually enter the appropriate extents for the single feature by using the INSERT statement provided by the *MBR_CALC* command.
- You can call the GOOM.CopyMBR procedure by using the script:
  `@MBR_Copy`

GDOSYS metadata is maintained by a trigger called *DELETEMETADATAGMT*. If you drop a table, the associated metadata in GDOSYS is also removed for that table. This is not true of Oracle’s metadata. If you drop a table using GeoMedia's tools, the Oracle metadata is also dropped but if you drop the table from Oracle, the metadata will be orphaned. Orphans in USER_SDO_GEOM_METADATA can be removed by using the following:

```
EXEC GOOM.DeleteOrphanMBR;
```

If you get an error about an existing entry, you need to manually remove the entry for the table from USER_SDO_GEOM_METADATA using the following command:

```
DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='<table_name>';
COMMIT;
```

If you want to utilize a trigger to maintain USER_SDO_GEOM_METADATA, you can run the following script in your spatial schema:

```
@Triggers/TrigUpdateUSGM.sql
```

Direct queries on the components of USER_SDO_GEOM_METADATA can be difficult. For example, the following query will fail:

```
SELECT USER_SDO_GEOM_METADATA.DIMINFO.SDO_TOLERANCE
  FROM USER_SDO_GEOM_METADATA
 WHERE TABLE_NAME = 'MYGEOMTABLE'';
```

```
ORA-00904: "USGM"."DIMINFO"."SDO_TOLERANCE": invalid identifier
```

For this type of query, you need to use Oracle's TABLE function to unpack the elements that make up the DIMINFO data type. Here are a couple of examples:

```
SELECT DIM.SDO_TOLERANCE
  FROM USER_SDO_GEOM_METADATA, TABLE (DIMINFO) DIM
 WHERE TABLE_NAME = 'MYGEOMTABLE'';
```

or this

```
SELECT DIM.*
  FROM USER_SDO_GEOM_METADATA, TABLE (DIMINFO) DIM
 WHERE TABLE_NAME = 'MYGEOMTABLE'';
```

Inserts can also be difficult; here is an example for a 2D geometry:

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES
('MYGEOMTABLE','GEOMETRY', MDSYS.SDO_DIM_ARRAY(
MDSYS.SDO_DIM_ELEMENT('X',-2147483648, 2147483648, 0.0005),
MDSYS.SDO_DIM_ELEMENT('Y',-2147483648, 2147483648, 0.0005)),NULL);
```

To review the USER_SDO_GEOM_METADATA table, you can use the following script:

```
@ListMBR
```

or, just use the following query:

```
SELECT * FROM USER_SDO_GEOM_METADATA;
```

8. ORACLE'S SRID AND GEODETIC DATA

The SRID is the integer value that Oracle uses to identify the coordinate system of the data stored in the SDO_GEOMETRY column. The SRID can be found in the SDO_GEOMETRY.SDO_SRID column of the geometry object and in the USER_SDO_GEOM_METADATA.SRID column in Oracle's metadata. If used, the SRID must exist in both locations and be identical for the given geometry column. GeoMedia does not use Oracle's SRID and will ignore the value. Any table created with GeoMedia will be created with a NULL SRID, however; any existing SRID's will be preserved as tables are edited.

If you are using projected data strictly within the GeoMedia environment, you do not need to set an SRID. If your feature class is based in Geographic coordinates, you are required to set an SRID for Oracle to properly handle the geometry data. Without the SRID, Oracle cannot correctly process spatial filters or queries. A list of available SRID's can be found from the following query in SQLPlus:

```
COLUMN NAME FORMAT A60;
SELECT CS_NAME NAME, SRID FROM MDSYS.CS_SRS;
```

Use the following script to list just the SRID's associated with geographic coordinate systems:

```
@ListSRIDbyGeo.sql
```

Use the following script to identify an SRID by a partial name (for example, WGS):

```
@ListSRIDbyName.sql
```

One of the most common SRID's in use is 8307 which corresponds to Longitude / Latitude WG84. Once you have found the SRID that you wish to use, you can run the following GOOM procedures:

```
EXEC GOOM.SetSRID(tablename, geometry_col_name, n_srid);
```

```
EXEC GOOM.SetSRIDAll (n_srid, [c_owner]);
```

The default for c_*owner* is always the current USER. You can also use the following scripts which call the GOOM package; you will be prompted for the required values:

```
@SetSRID.sql
```

If you make a mistake, re-run *SetSRID* and set the *SRID* to 0. This will assign a NULL to the SRID. If you want to assign the same SRID to all feature classes in the schema, use the following:

```
@SetSRIDAll.sql
```

Both processes require that the spatial index be dropped and recreated. This is done automatically using RTree indexes.

All of these commands make use of the GOOM package command:

```
EXEC GOOM.SetSRID('<table>','<geom_column.>',n_srid);
```

Where:

<table>	- Name of the table (or owner.table) in single quotes.

<geometry_col>	- Name of the column containing the geometry
n_srid	- The SRID from select SRID from MDSYS.CS_SRS. Set n_srid=0 to set NULL value (Default)

Note: Changing the SRID will invalidate the spatial index, the procedure will automatically re-index.

9. SPATIALLY INDEXING GEOMETRY DATA

Currently R-Trees are the preferred index type for spatial data. RTree indexing uses the following syntax:

```
CREATE INDEX <indexname> ON <table>(<geom_column>)
       INDEXTYPE IS MDSYS.SPATIAL_INDEX
       [PARAMETERS ('index_params [physical_storage_params]')];
```

where

<indexname>	- the name of the index you want to create.
<table>	- the name of your feature class
<geom_column>	- the name of the geometry column in your feature class
PARAMETERS	- there are a lot of optional parameters you can use, refer to the *Oracle Spatial Developer's Guide* for more information.

Use the following script to automatically index all your feature classes using RTrees (Note: In most cases, *table_name* can be in the format *owner.tablename* or just *tablename; UPPERCASE of course*):

```
EXEC GOOM.SpatialIndex('<table_name>', b_option);
```

Or

```
EXEC GOOM.SpatialIndexAll('schema_name', b_option);
```

schema_name	- The default schema is the current schema (USER);
b_option	- Set to TRUE (DEFAULT) to constrain the geometry to the GTYPE and FALSE to ignore the GTYPE. If you constrain the GTYPE, no other geometry type can be digitized into this geometry. It will also improve performance on point features.

Both of these use the following internal GOOM procedure:

```
EXEC GOOM.Rtree('<table_name>','<geometry_col>');
```

Do not index views unless you are using materialized views. Normal views will automatically pick up the spatial index from the base table's geometry.

To list existing spatial indexes (these are known as domain indexes) use the following:

```
@ListSidx
```

To delete all existing spatial indexes, use the following:

```
EXEC GOOM.DelSidx; or EXEC GOOM.DelSidx('USER'>;
```

To delete a single spatial index for a specified table/column, use the following:

```
EXEC GOOM.DROPSIDX('<table_name>','<column_name>');
```

You can also delete indexes manually. If a problem occurs during the indexing procedures, the spatial indexes will be in a corrupt state and will have to be dropped before you can re-create them. The only way to do this is to use the force option:

```
DROP INDEX <indexname> FORCE;
```

If your data was imported into Oracle from the **Export to Oracle Object** command, you are now ready to use your data in GeoMedia.

If you want to automate the entire process of fixing minor problems, setting the extents, and indexing your data use the ***AutoTune*** procedure from GOOM. For example:

`EXEC GOOM.AutoTune;` or `EXEC GOOM.AutoTune('SCHEMA');`

Typically though, to re-index, just use the indexing procedure:

`EXEC GOOM.SpatialIndexAll;` or `EXEC GOOM.SpatialIndexAll('SCHEMA');`

10. CONFIGURING GEOMEDIA'S METADATA

The **Export to Oracle Object** process creates scripts that automatically update the GDOSYS metadata schema with the required information about the feature classes being imported. **Output to Feature Classes** also takes care of this for you. Normally, this is sufficient for most uses, however you may find it necessary to make modifications as needed. For example, you may want to hide specific columns from GeoMedia, assign autonumber sequences to primary key columns, or alter the specific geometry types. If you create or edit tables manually in Oracle or you create views, you will need to create new metadata entries for those new tables or views. In both cases, **Database Utilities** is the tool you need to edit and update GeoMedia's GDOSYS metadata schema. The GOOM package also contains procedures that will set GDOSYS metadata for tables and views.

Inserting Feature Class Metadata

The **Insert Feature Class Metadata** in **Database Utilities** command allows you to create the metadata required for GeoMedia to see a table or view as a feature class. This applies to both graphic and non-graphic feature classes. Tables that do not have metadata entries will not be visible to GeoMedia. Here is the basic workflow for inserting a feature class:

1. Select the Insert Feature Class Metadata button on the Database Utilities dialog.
2. Your schema should already be selected in the *Schemas* list but check to make sure. By default, the schema owned by the connected user will be selected. Select the tables/or views you wish to add from the left side of the dialog and use the > button to shift them to the right side.
3. Each feature class in the right hand list will require specific properties to be set depending on the type of feature class. For each feature class in the list you will need to perform the following.
4. Select the feature class (it should be highlighted) and then select the **Properties** button. This activates the **Feature Class Properties** dialog, which lets you control description, attribute, and geometry information about the selected feature class.
5. On the **Attribute** tab, find your primary key column (it will have a key symbol). If your data does not have one, the process will not work.
6. If your primary key is a standard integer sequence, you will benefit by assigning it the datatype of autonumber. This tells GeoMedia Pro to automatically update this column using an Oracle sequence when inserting new records into this feature class. A right mouse click in the Datatype field will pop up a list of available datatypes, select the autonumber type and a list of available sequences will appear.
7. Select an existing sequence to assign or create a new sequence. The sequences are stored in your schema. GeoMedia makes use of them, but Oracle does not. For Oracle to utilize these, you would also have to create a trigger. The sequences are automatically set to the next available value after finding the current maximum for the column in question.
8. You can use the *Displayable* column to hide any columns from view inside of GeoMedia. Use this if you have columns you don't need or want to use within the GeoMedia environment.
9. The **Geometry** tab is where you set the Geometry Type for your Geometry column. The default is *SpatialAny*. Left click on the field to choose from available Geometry Types. Pick the type that is

appropriate for your geometry. With the exception of *GraphicText*, you should only use the *SpatialPoint*, *SpatialLine, SpatialAny*, or *SpatialArea* types, avoid the *GraphicTypes*. You can have more than one geometry column but you can have only one primary geometry. That one is assigned by the Primary Geometry Field.

10. After all the properties have been assigned, select **OK** on the **Insert Feature Class Metadata** dialog. At this point, you will need to re-assign the existing coordinate system to the feature classes or create a new coordinate system to assign. If one already exists, select and assign it, otherwise, you will have to create a new one.
11. When you select Ok on **the Select Coordinate System** dialog, the selected coordinate system will be assigned to all the feature classes in the right hand list on **the Insert Feature Class Metadata** dialog. If this is a problem, i.e. these feature classes are in different coordinate systems, you can use the Assign Coordinate System… button to change the coordinate system for each individual feature class.

At this point, your data is ready to be displayed and analyzed in GeoMedia. If you want to verify this, open GeoMedia, connect to your Oracle Schema (make sure you use the right data server) and display your data.

If you have simple feature classes and do not require any specific settings, you can use the following procedure to assign metadata for a single table that contains only one primary key column (associated with a sequence):

```
EXEC GOOM.SetGDOSYSMetadata('<table_name>','<seq_name>','<v_cs>',
                             [<v_geomtype_in>],['<v_keycolumn>'],
                             ['<num38>'],['v_num10']);
```

- *<table_name>* - The name of the table or view, upper case in single quotes.
- *<seq_name>* - Optional sequence name, use 'AUTO' to automatically generate and assign sequence for the primary key or use NULL for no sequence assignment.
- *<v_cs>* - The coordinate system to assign the geometry columns. Use CSGUID or enter 'D' to use the default warehouse CS. The default is 'D'.

Optional parameters include:

- *<v_gtype_in>* - Optional geometry type for the table particular useful for empty tables. If omitted the geometry will be retrieved from existing geometry. Use one of the following types: 10-point, 1-line, 2-Area, 3-compound, 5-text
- *<v_keycolumn>* - Optional, for views only, enter the key column from the base table. Use only if feature class is actually a view. If omitted/NULL, the feature class is treated as a normal table.
- *[<num38>]* - Optional override for NUMBER(38) PKEY assignments. Any value here except 'LONG' (the default) forces DOUBLE override. Use only if you need interoperability with ESRI NUMBER (38) for key columns.

- *[<v_num10>]* - Optional override for all NUMBER(10) assignments. Any value here except 'DOUBLE' (the default) forces LONG override. Use only if you need interoperability with G/TECH NUMBER(10) for G3E columns.

If you want the primary key to be autonumber, you need to include the sequence name. The use of a sequence is optional and may be NULL. *<cs>* is the coordinate system identifier that will be assigned to the geometry columns. It can either be the CSGUID of an existing CS or you can enter DEFAULT to use the warehouse's default coordinate system. If there is no default coordinate system assigned, the process will assign a dummy coordinate system. This will have to be changed using **Database Utilities' Assign Coordinate System** command.

For example, for a simple table called ROADS using a sequence called ROADS_SQ, here is the syntax:

```
EXEC GOOM.SetGDOSYSMetadata('ROADS','ROADS_SQ','D');
```

For a view called ROADS_VIEW with preserved key column ID, the syntax might be:

```
EXEC GOOM.SetGDOSYSMetadata('ROADS_VIEW','ROADS_SQ','D',NULL,'ID');
```

Editing Existing Feature Class Metadata

If you need to alter the metadata of existing feature classes, you can use Database Utilities to edit the existing metadata for your feature classes. The following will step you through the edit metadata process:

1. Invoke Database Utilities from the Start menu (it will locate under the GeoMedia Professional Program Group).
2. Change the Database Type to Oracle Object Model
3. Enter the user name, password, and net service name (server) for the Oracle schema you wish to work with. This activates the main Database Utilities menu:
4. Select the **Edit Feature Class Metadata** button. A list of feature classes in this schema that have metadata will be listed in the format *owner.feature_class*. Select the feature class to edit and then select Properties. Make the required changes and then select OK.

NOTE: If you are making DDL changes to a table or view (changes to the table/view definition), you cannot use EDIT to modify the metadata. Instead, you will have to delete the metadata for the table/view prior to making the DDL changes and then re-insert the metadata after the changes are made. This will ensure that the column changes will be added to the metadata.

Deleting Existing Feature Class Metadata

If you need to change the structure of a table by adding or removing a column, or you need to change an existing column's datatype, you must first delete the existing metadata for your feature class and then insert new metadata back into GDOSYS. The Edit command edits existing metadata, it will not add to existing metadata. The following will step you through the delete metadata process:

1. Invoke **Database Utilities** from the Start menu (it will be under the GeoMedia Professional Program Group).
2. Change the *Database Type* to *Oracle Object Model*
3. Enter the user name, password, and net service name (server) for the Oracle schema you wish to work with. This activates the main Database Utilities dialog.
4. Select the **Delete Feature Class Metadata** button. A list of feature classes in this schema that have metadata will be listed in the format *owner.feature_class*. Select the feature classes whose metadata will be removed. You will be prompted to verify your action.

You can also use delete the metadata entries for a table by using the following procedure:

```
EXEC GOOM.DelGDOSYSMetadata('<TABLE_NAME>');
```

As tables are deleted, the metadata present in GDOSYS should be maintained automatically (via a trigger in GDOSYS). If you think you are collecting orphans, you can run the following script to list them:

```
@ListGDOSYSorphans;
```

And then the following to delete them:

```
Exec GOOM.DelGDOSYSOrphans;
```

This will check for and delete any orphan records found in GDOSYS. This rarely occurs but it does happen, particularly in test environments.

11. METADATA BACKUP USING THE GDOBKP PACKAGE

The GDOBKP package allows you to backup up the metadata stored in GDOSYS for a specific schema. This is useful is you need to move the schema to another server or simple want to preserved the metadata as you have set it.

Main Procedures

When backing up metadata, you have the option of storing the metadata for all tables and views in the schema or for just one table. The backup is table based and should work with all standard GM metadata as well as the AFM portion of AFM metadata (it will NOT backup AFM metadata).

There are 2 main procedures:

- SaveGDOSYSMetadata
- RestoreGDOSYSMetadata:

The following tables are created by the backup process:

- BKP_SDO_GEOM_METADATA – Oracle metadata values for this schema
- GCOORD_BKP – Stores the coordinate systems used by the schema.
- GCOORD_DEF – Stores the default coordinate system for the schema if it exists.
- GDOSYS_BKP – Stores the GDOSYS metadata for this schema

To backup up the metadata for an entire schema, run the following:

```
EXEC GDOBKP.SaveGDOSYSMetadata;
```

while connected to the schema or run:

```
EXEC GDOBKP.SaveGDOSYSMetadata('SCHEMA_NAME');
```

while connected as a DBA user.

The GDOBKP package will backup and restore GDOSYS metadata for all the feature classes in the given schema. Once the metadata has been backed up, you can use Oracle's **IMP** and **EXP** utilities (or data pump) to save, transfer, and restore the Oracle schema. It is not necessary to restore the data to a schema with the same name, the metadata will always use the current schema name.

Following the backup process, a verification check will be run. Make sure that the verification check passes and that the metadata stored in GDOSYS matches the metadata backed up to your local schema.

Once the metadata has been saved to local tables, you can use Oracle's EXPDP/IMPDP (or the old EXP/IMP) commands to export the schema, move the data, and re-import as needed. This is particular useful if you need to send a dataset (or even a single table) to Intergraph customer support.

To restore the metadata from a schema that contains the backup tables, you must be connected in SQL as the owner of the schema. The GDOSYS metadata schema must also exist prior to restoring any backup and you must have write privileges on GDOSYS. Run the following to restore the metadata:

```
EXEC GDOBKP.RestoreGDOSYSMetadata;
```

There are no input options; it will simply restore the metadata as it exists in the backup tables. At the end of the restore process, the restoration will be verified. Make sure everything checks out, if an error occurs, you will need to fix it before moving on. Sequence ownership is also verified at this time as is picklist ownership.

Backup Caveats:

- The backup/restore procedures handle basic metadata only; they do not work with AFM specific metadata, LTT specific metadata, or any other metadata outside of the standard GeoMedia GDOSYS metadata. Metadata specific to other industry application are not backed up.
- Metadata Description fields are backed up as long as they do not contain any special characters. This includes single and double quotes. Simple alpha numeric text only.
- Original metadata is not deleted during the process though there is a procedure that will do that if desired (see `DeleteGDOSYSMetadata`).

Restore Caveats:

- The GDOSYS metadata schema must already exist in the target database.
- The restore process will delete existing metadata for the same feature class in the same schema.
- The restore process is user independent, you can load the data and restore the metadata into a different user account if needed (even on a different database). This is most useful if you are getting a dataset from someone else or are moving test data to a production system.
- The GDOSYS metadata versions must match – 6.1, 2013, and 2014 all have the same metadata version.

Verification of the backup is automatic but you can re-verify the GDOSYS backup data to existing GDOSYS metadata at any time using the following:

```
EXEC GDOBKP.VerifyGDOSYSBkp;
```

This is useful if you want to determine whether your backup copy is still valid. If there have been any metadata changes since your last metadata backup, it will show up here. This would indicate a need to run the metadata back up again.

Other Procedures and Functions

- **PROCEDURE DeleteGDOSYSMetadata;**
  This procedure deletes all the GDOSYS metadata related to the schema where the command is run. It does not touch GCOORDSYSTEM entries. The syntax is:

  ```
  EXEC GDOBKP.DeleteGDOSYSMetadata;
  ```

- **PROCEDURE DeleteBackupTables;**
  This procedure removes the backup tables created by the other procedure in this package. The

syntax is:

```
EXEC GDOBKP.DeleteBackupMetadata;
```

- **PROCEDURE SaveGCoordSystem;**
  This procedure backs up all the coordinate systems that are in use by the feature classes in the schema. The coordinate systems are stored in a local table called GCOORDSYS_BKP. If a Default schema coordinate system is found, it will be stored in GCOORDSYS_DEF. The syntax is:

  ```
  EXEC GDOBKP.SaveGCoordSystem;
  ```

- **PROCEDURE RestoreGCoordSystem;**
  This procedure restores the information stored in the GCOORDSYS_BKP table and if present, GCOORDSYS_DEF. The syntax is:

  ```
  EXEC GDOBKP.RestoreGCoordSystem;
  ```

Typical Workflow

To backup an Oracle dataset, first backup the metadata using:

```
EXEC GDOBKP.SaveGDOSYSMetadata;
```

then, use Oracle's exp utility to output the schema to an Oracle dump file:

```
EXP user/pswd@dbservice file=user.dmp
```

The user.dmp file contains a complete dump of your schema.

To restore your schema, first create a new user account that will hold the data, then run Oracle's imp utility:

```
IMP dba/pswd@dbservice file=user.dmp fromuser=user touser=newuser
```

Connect to the database as this new schema and then run:

```
EXEC GDOBKP.RestoreGDOSYSMetadata;
```

Follow this by running:

```
EXEC GOOM.SpatialIndexAll;
```

This assumes that GDOSYS already exists in the target instance and both GOOM and GDOBKP package must already be installed.

12. STANDARD MAINTENANCE OPERATIONS

Oracle schemas do not maintain themselves and there are a lot of things a DBA has to do to ensure healthy database operation.   Here are the recommended maintenance operations:

Tuning

As DML operations are performed on a table, the RTree indexes on the geometry fields will degrade and the performance of spatial filters will begin to suffer. Eventually, the spatial filters will not work at all. To maintain spatial filter performance, the RTree indexes have to be regenerated on a regular basis. How often this is necessary depends on the amount of editing that takes place. There is no good rule of thumb here.

Most sites regenerate the spatial indexes every weekend which is usually sufficient. This can be done by setting up an Oracle job that runs the GOOM package procedure `SpatialIndexAll.`

Standard indexes also need to be rebuilt periodically.   How and when this is done depends on the data in question and the design of the schema.

Metadata

The MODIFICATIONLOG table in GDOSYS can also have negative impact on performance. If you connection performance seems to be slowing down, it is usually due to the size of this table. To solve this, the MODIFICATIONLOG table should be truncated whenever there is downtime, such as evenings and weekends. Again, you can use an Oracle job for this.   Entries in the MODIFICATIONLOG are based on MODFIEDTABLES. For best results, you should truncate the MODIFIEDTABLES table at the same time you truncate the MODIFICATIONLOG table.

GOOM has a procedure to do this:

```
EXEC GOOM.ClearModLog;
```

The best way to create a job is via the EMCA console but you can also use the following script:

```
CreateModLogJob.sql
```

Backups

The easiest way to backup a schema is to use the procedures and methods outlined in above.   Oracle offers a lot of different way to back up data depending on the size of the data and how critical that data is.   A schema backup is the simplest way but you may still lose data depending on how often the schema backups are made. Check your Oracle documentation for alternate methods.

Maintenance Windows

Oracle utilizes maintenance windows to run automated procedures in the database. A maintenance window can be any contiguous time interval, such as "between midnight and 6 a.m., every Saturday", or a more complex interval such as "between midnight and 6 a.m., on the last workday of every month, excluding company holidays". These windows are used by the Oracle Scheduler to run automated procedures. When a maintenance window opens, Oracle Database creates an Oracle Scheduler job for each maintenance task that is scheduled to run in that window. Each job is assigned a job name that is generated at runtime.

To keep your database running at optimal performance and efficiency, the database should be running routine maintenance tasks whenever it is idle. Oracle also has built in maintenance tasks that you should not modify, if these built in tasks are not run, the database will have serious performance degradation.

Typical maintenance tasks should include the following:

- Statistics Gathering – Statistics should be gather regularly and are crucial to maintaining performance.
- Rebuilding Spatial Indexes – Typically weekly but this really depends on the amount of editing that is occurring. Static tables rarely need re-indexing.
- Maintaining metadata – For GeoMedia apps, truncating the GDOSYS.MODIFICATIONLOG and MODIFIEDTABLES is necessary. This should be done nightly. Failure to do this can cause editing and map refresh performance to suffer.

Database Statistics

Oracle's Cost Base Optimizer (CBO) determines how to process all SQL. The CBO requires complete and up to date statistics on database objects in order to efficiently process queries. The real key to maintaining performance is maintaining statistics. Here are the points to remember:

- If any object changes by more than 10%, then new statistics are required.
- Full Database Statistics should be run automatically in off peak hours.
- System statistics is also important but need to be collected less often.
- Init parameter TIMED_STATISTICS should be set to TRUE. This does not hurt performance.
- Init parameter STATISTICS_LEVEL = (TYPICAL,ALL,BASIC)
  - Must be at least TYPICAL – collects all major statistics required by the CBO and provides best overall performance
  - ALL – Adds timed OS statistics and plan execution statistics to TYPICAL
- To get an idea of your current level of statistics gathering use:

```
SELECT STATISTICS_NAME, DESCRIPTION, SYSTEM_STATUS
  FROM v$statistics_level;
```

The SYSTEM_STATUS should be ENABLED for all STATISTICS_NAME values listed.

On my servers, I use the following as a general guide to gathering statistics:

- Run weekly during non peak hours:
    - EXEC DBMS_STATS.GATHER_DATABASE_STATS; Gathers complete database statistics
    - EXEC DBMS_STATS.GATHER_DICTIONARY_STATS; Gathers statistics on SYS, SYSTEM, and other dictionary schemas
- Run during heavy workload for at least one hour:
    - EXEC DBMS_STATS.GATHER_SYSTEM_STATS; Gathers host system OS statistics
- Run during peak loads as elapsed time
    - EXEC DBMS_STATS.GATHER_SYSTEM_STATS('START'); min one hour delay during high workload
    - EXEC DBMS_STATS.GATHER_SYSTEM_STATS('STOP');
- Run rarely, typically after every upgrade.
    - DBMS_STATS.GATHER_FIXED_OBJECTS_STATS;
- Run nightly during non peak hours
    - DBMS_STATS.GATHER_SCHEMA_STATS (USER, CASCADE=>'TRUE');
- Run as needed for high use schemas:
    - DBMS_STATS.GATHER_TABLE_STATS(USER, 'TABLE', CASCADE=>'TRUE');
    - DBMS_STATS.GATHER_INDEX_STATS(USER, 'INDEXNAME');

These all have a variety of parameters and can greatly vary in execution speed. These can help performance if there is a lot of editing occurring at the DB level. What you set up and utilize on your system is up to you.

Initialization Parameters

Setting initialization parameters used by an Oracle database is highly dependent on the systems configuration. Below are the parameters we use in-house for Oracle 12C, most would also apply to 11G.

```
ALTER SYSTEM SET db_file_multiblock_read_count = 8 SCOPE=BOTH;
ALTER SYSTEM SET open_cursors=2000 SCOPE=BOTH;
ALTER SYSTEM SET recyclebin='OFF'  SCOPE=BOTH;
ALTER SYSTEM SET optimizer_secure_view_merging=FALSE SCOPE=BOTH;
ALTER SYSTEM SET optimizer_index_caching=50 SCOPE=BOTH;
ALTER SYSTEM SET optimizer_index_cost_adj=20 SCOPE=BOTH;
ALTER SYSTEM SET spatial_vector_acceleration=TRUE SCOPE=BOTH; -- 12C only
```

These are dynamic and can be applied to a running instance however, the may or may not improve your overall performance.

13. MISC. SCRIPTS AND PROCEDURES

There are quite a few scripts available as examples. If you are unsure what a script does, check its header. It should tell you what the script is for. Many of the scripts are just prompts for GOOM package procedures.

Coordinate System Commands

If you want to get a list of the coordinate systems in GDOSYS, enter the following SQL:

```
@ListCS
```

If you want to see the GDOSYS coordinate system assignments for all feature classes in the database, enter the following SQL:

```
@ListCSAll
```

For a list of coordinate systems, including the default, assigned to a specific user account, use:

```
@ListCSUser
```

Entries in GCOORDSYSTEM are never deleted and over time, you can get a lot of coordinate system orphans. The following command will give you a list of the orphaned coordinate system entries:

```
@ListCSOrphans
```

Removing orphans can improve performance. To remove the orphan coordinate system entries:

```
EXEC GOOM.DelCSOrphans;
```

To set a default coordinate system for an existing schema:

```
EXEC GOOM.SetDefaultCS(csguid,schema_name);
```

Or use the script:

```
@SetDefaultCS
```

and follow the prompts.

Cataloging a Schema

The following script will create a catalog of your data and store it in a CATALOG table. The contents include feature types, counts and Oracle/GeoMedia feature mapping:

```
@CreateCatalog
```

The following script lists the contents of the CATALOG table allowing you to review it later:

```
@ListCatalog
```

14. TROUBLE SHOOTING

If you run into problems using the Oracle Object Model with GeoMedia or GeoMedia Pro, you have a couple tools to help you troubleshoot the problem. To see what the client is sending to Oracle, you can use GeoMedia's GORACLE.log. On the database server, you can use Oracle's built in trace capability.

GORACLE.LOG

The *GORACLE.log* file will be used automatically if the data server detects its existence. This log file is a simple text file called *goracle.log* and must be located in the *C:\temp* directory.

Note: Only use *goracle.log* when trouble shooting a problem. It will affect overall performance.

If you are getting a database error from an operation in GeoMedia, you need to reproduce the problem with *GOracle.log* in place. If you log a support request, this log file will be the first thing the support analyst will request. If you can provide it when you log the worksheet, you will likely get a resolution much faster.

GOracle.log will contain all the sql passed to Oracle via GeoMedia's Oracle data server. Any GeoMedia problems that occur will be flagged with a GOERROR – hex value. If the error happened in Oracle, you will also see a standard ORA-ERRNUMBER and a brief description of the problem. The Oracle data server also sends SQL to the database to deliberately generate an error; these are used to test specific conditions and can be ignored. Here is an example of the type of error you should be looking for:

```
pSQLStatement:
'insert into GDOSYS.GFEATURES(FEATURENAME,GEOMETRYTYPE,PRIMARYGEOMETRYFIELDNAME,
FEATUREDESCRIPTION) values ('GRegistrationGuides',1,'LineGeometry','GRegistrationGuides')'

Database Error: ORA-00001: unique constraint (GDOSYS.SYS_C001711) violated

GOERROR - 8004076f
```

In this particular case, the insert statement generated an Oracle error which caused the insert to failure. This failure then led to a problem in GeoMedia as indicated by the GOERROR. The hex number can be used by development to determine what module the error occurred in. This is only useful when using a debugger.

Session Traces

Oracle tracing allow you to examine SQL statements and how they are being processed in Oracle. This is useful for tracking down performance issues and other problems relating to queries. This is your second line of defense in determining the cause of problems. For best results, initialize the trace just before you run the specific GeoMedia command.

1. Initialize GeoMedia Professional and connect to your spatial schema.
2. Open SQLPlus and connect to SYS.
3. Enter the following to set the required parameters for tracing:

```
ALTER SYSTEM SET TIMED_STATISTICS=TRUE;
ALTER SYSTEM SET MAX_DUMP_FILE_SIZE=200000;
```

4. To determine the session to trace, enter the following:

```
SELECT USERNAME, SID, SERIAL#, PROGRAM, OSUSER, TERMINAL
  FROM V$SESSION
 WHERE OSUSER NOT LIKE 'SYSTEM%';
```

5. You should see a SID and a SERIAL number associated with GeoMedia. Write these down.
6. Now initialize the trace:

```
EXEC DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION (&SID,&SERIAL,TRUE);
```

   when prompted, enter the SID and SERIAL for GeoMedia.
7. If you receive the message *PL/SQL procedure successfully completed*, the trace is initiated.
8. In GeoMedia, display a feature class and run an attribute query.
9. Now turn off the trace. There are 2 ways to do this. If you end your GeoMedia session i.e. exit from GeoMedia or close the database connection, the trace will end automatically. Otherwise, you will need to do the following to end the trace:

```
EXEC DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION (&SID,&SERIAL,FALSE);
```

   using the same SID and SERIAL you used to start the trace.
10. The trace file is generated on the server in a directory specified in the initialization parameters. By default this is in: *C:\oracle\admin\<DBNAME>\udump* directory.
11. To interpret the trace file, you will need to use *tkprof*. Go the directory containing the dump file and open a command window.
12. Enter the following (all one line):

```
TKPROF.EXE <CURRENT_TRACE>.TRC MYTRACE.TXT SYS=NO
SORT=(PRSELA,EXEELA,FCHELA)
```

   The Tracing directory contains 2 bat files that will simplify this operation. *TraceOrder.bat* will automatically convert any trace file found to a readable format based execution order. *TracePerformance.bat* does the same thing but places the slowest SQL first in the list.
13. The output file *MyTrace.txt* will hold the trace statistics. Open this file and examine the contents. This file can help you trouble shoot bottlenecks in the database related to I/O and to execution of queries

The tracing directory contains utilities to simplify the tracing process. These utilities should only be used by the DBA. These trace procedures require the use of DBMS_MONITOR. You will need to grant EXECUTE on DBMS_MONITOR to the DBA user from the user SYS. I use SYSTEM for my DBA account but any account with DBA privileges will work here.

After granting the EXECUTE privilege on DBMS_MONITOR, load the *DBTraceSessionProc* into your DBA user account.

```
CONNECT SYSTEM/MANAGER@SERVER  @DBTraceSessionProc.sql
```

Once this procedure is loaded, there are 2 scripts that can be run:

- DBTraceConnect – Creates a trigger on the specified schema that will activate on connection and begin tracing until the schema disconnects. Remove the trigger when the trace is completed
- DBTraceSession – Traces a specified session. Use this to capture a snapshot of a specific process.

Make sure you have run SET SERVEROUTPUT ON to get feedback from these. You need to use TKPROF to interpret the trace file. I have included 2 batch files to help here (rename these to .bat from .bat1). Copy the trace file to the *C:\temp* director along with these 2 files then just double click on the one you want to use:

- TracePerformance.bat – The trace is sorted in order of the slowest SQL statement to the fastest statement. This is good if you are trying to track down performance issues.
- TraceOrder.bat – The trace is sorted in the order of execution. This is good for tracking problems.

Note: These batch commands operate on all *.trc* files in the directory where they are run.

Here is an example of a trace for a spatial filter query:

```
Select ID,ID,CITY_NAME,GEOMETRY,ANNULSNOW
  From SPATIALUSER.V_SNOW_CITIES A where (MDSYS.SDO_RELATE(A.GEOMETRY,
     :GeometryFilter, 'mask=INSIDE+COVEREDBY+EQUAL querytype=window') = 'TRUE')
 call     count       cpu    elapsed       disk      query    current        rows
-------------------------------------------------------------------------------
Parse         1      0.00       0.00          0          0          0           0
Execute       1      0.06       0.05          0        777          0           0
Fetch         2      0.03       0.01          0        139          2          81
-------------------------------------------------------------------------------
total         4      0.09       0.06          0        916          2          81
```

The output table is broken down into the 3 main sql processes:

- PARSE - This step translates the SQL statement into an execution plan. This step includes checks for proper security authorization and checks for the existence of tables, columns, and other referenced objects.
- EXECUTE - This step is the actual execution of the statement by Oracle. For INSERT, UPDATE, and DELETE statements, this step modifies the data. For SELECT statements, the step identifies the selected rows.
- FETCH - This step retrieves rows returned by a query. Fetches are only performed for SELECT statements.

The other columns of the SQL trace facility output are combined statistics for all parses, all executes, and all fetches of a statement. The sum of query and current is the total number of buffers accessed.

- COUNT – Number of times a statement was parsed, executed, or fetched.

- CPU – Total CPU time in seconds for all parse, execute, or fetch calls for the statement.
- ELAPSED – Total elapsed time in seconds for all parse, execute, or fetch calls for the statement.
- DISK – Total number of data blocks physically read from the datafiles on disk for all parse, execute, or fetch calls
- QUERY – Total number of buffers retrieved in consistent mode for all parse, execute, or fetch calls. Buffers are usually retrieved in consistent mode for queries.
- CURRENT – Total number of buffers retrieved in current mode. Buffers are retrieved in current mode for statements such as INSERT, UPDATE, and DELETE.
- ROWS – The total numbers of rows processed by the SQL Statement. For SELECT, rows are reflected for fetch, for INSERT, UPDATE, and DELETE, the rows are reflected for execute.

Figuring out what to do with all this information is the tough part. Typically, if you are reporting a performance issue, this will help us determine whether the problem in on the client, GeoMedia, or on the server, Oracle. If can also help us tune the database to run the queries more efficiently. If you are interested in more here, use the internet or get a good book on Oracle Performance Tuning.

APPENDIX A: THE GOOM PACKAGE

Run the *GOOM_PKG.sql* script from a DBA user to compile and load the GOOM package. The package will be installed into GDOSYS and the commands will be available to all PUBLIC users. They can also be used in your own scripts. Remember, if you are still using Oracle 10G, use the correct version of GOOM.

To find out what version of the GOOM package you are using, enter the following in SQL:

EXEC GOOM.VERSION;
To see a list of available commands and syntax after the GOOM package has been loaded, enter the following at the SQL Prompt:

EXEC GOOM.HELPME;
For syntax and command descriptions, enter:

EXEC GOOM.HelpDATA;	Spatial Data Manipulation
EXEC GOOM.HelpMBR;	Oracle metadata procedures
EXEC GOOM.HelpGDOSYS;	GeoMedia metadata (GDOSYS) procedures
EXEC GOOM.HelpTUNING;	Spatial Indexing and tuning
EXEC GOOM.HelpUTILITIES;	Miscellaneous useful stuff
EXEC GOOM.HelpVALIDATION;	Data Validation and repair procedures
EXEC GOOM.HelpFUNCTIONS;	GOOM utility functions

To specify the tablespace where you want GOOM to create spatial indexes, run the following at the SQL prompt while connected as a DBA user:

EXEC GOOM.SetGOOMIndexSpace('<INDX_SPACE>');

where *<INDX_SPACE>* is the name of the tablespace where you want the spatial indexes to be created

To determine what tablespace is currently being used, you can run the following:

SELECT GOOM.GetGOOMIndexSpace FROM DUAL;

GOOM HELP SYNTAX

Here is the syntax guide for the procedures and functions in GOOM:

v_	- indicates varchar2 literal in upper case - 'TABLENAME' or 'OWNER'
n_	- indicates numeric - 1.3 or 12432.0
i_	- indicates integer - 1 or 3 or 12432.
b_	- indicates boolean - TRUE or FALSE
[ ]	- indicates an optional value.
()	- indicates a default value

<geometry> or **g_** - indicates geometry object, this is not a literal.

Common usage:
v_ownertable - Name of the table to use: 'TABLENAME' or 'OWNER.TABLENAME'

v_geomcol - Name of the SDO_GEOMETRY column: Default is NULL in most cases. If the table contains only one geometry column, NULL will force the geometry column to be determined automatically.
v_schema - Name of schema to process: 'CHUCK'
g_geometry - You can pass the actual geometry column, a variable containing a sdo_geometry construct or the construct itself.

Usage Notes:

- OWNER.TABLE input is allowed for most procedure and functions except where indicated. If the owner is not specified, the default is always assumed to be the current USER or schema. Always remember to use single quotes around literals. You can also pass the USER built in variable.
- In most cases, it is assumed the DBA is the user running in OWNER.TABLE mode. Standard users may not have all the privileges required to run all procedures on tables they do not own directly but have limited access to.
- When specific tables are not used, the command generally applies to all feature classes in the schema.

Spatial Data Manipulation (EXEC GOOM.HelpData;)

Bearing returns bearing between 2 points.

```
FUNCTION Bearing(n_xcoord1 IN NUMBER,
                 n_ycoord1 IN NUMBER,
                 n_xcoord2 IN NUMBER,
                 n_ycoord2 IN NUMBER,
                 v_ctype   IN VARCHAR2 DEFAULT 'P',
                 v_rtype   IN VARCHAR2 DEFAULT 'D')
RETURN NUMBER;
```

Syntax: n_bearing := BEARING(x1,y1,x2,y2,v_ctype,v_rtype);
v_ctype - Default 'P' for projected. Otherwise calculates for long lat.
v_rtype - Default 'D' for degrees. Otherwise calculates radians.

ChangeOrdinatePrecision returns a geometry with the values in the ordinate array rounded to the number of specified decimal places. Permanently alters your data!

```
FUNCTION ChangeOrdinatePrecision(g_geom IN SDO_GEOMETRY,
                                 i_decimals IN INTEGER DEFAULT 6)
RETURN MDSYS.SDO_GEOMETRY;
```

Syntax: UPDATE table A SET A.geometry = GOOM.ChangeOrdinatePrecision(A.geometry, i_decimals);

g_geom is the actual geometry column (no quotes).
i_decimals is the number of decimal places of precision to preserve, the default is 6.

ContainsArcs is a check for arcs in a geometry. It returns 1 for true and 0 for false.

FUNCTION ContainsArcs(g_InputGeom IN SDO_GEOMETRY)
RETURN INTEGER;

Syntax: IF GOOM.ContainsArcs(g_geom) = 1 THEN
Select GOOM.ContainsArcs(a.geometrycol) FROM table A;
g_geom is the actual geometry column (no quotes).

ConvPoly2Line takes an input polygon feature class and converts it to a polyline geometry feature class. The entire feature class is converted. If you want to convert just a single geometry, use Oracle's SDO_UTIL.POLYGONTOLINE.

```
PROCEDURE ConvPoly2Line(v_tablename IN VARCHAR2,
                        v_geomcol IN VARCHAR2 DEFAULT NULL);
```

Syntax: EXEC GOOM.ConvPoly2Line(v_tablename,v_geomcol);
v_tablename is the table being modified.
v_geomcol is the column containing the geometry.

Note: This command does not support owner.table.

Convert2D returns a 2D geometry based on a 3D input geometry. The Z is stripped off.
If an error occurs, the original input geometry is returned.

FUNCTION Convert2D(g_3DGeom IN MDSYS.SDO_GEOMETRY)
RETURN MDSYS.SDO_GEOMETRY;

Syntax: SELECT GOOM.Convert2D(GEOMETRY) FROM TABLE;
UPDATE TABLE SET GEOMETRY=GOOM.Convert2D(GEOMETRY);
g_2DGeom:=GOOM.Convert2D(3dgeometry);

Note: This command does not support owner.table. Raster and Text are not supported.

Convert3D returns a 3D geometry based on a 2D input geometry. The Z is populated by the specified constant Z value (default = 0). If an error occurs, the original input geometry is returned.

```
FUNCTION Convert3D(g_2DGeom IN MDSYS.SDO_GEOMETRY,
                   n_Zval IN NUMBER DEFAULT 0)
RETURN MDSYS.SDO_GEOMETRY;
```

Syntax: SELECT Convert3D(GEOMETRY, 0) FROM TABLE;
UPDATE TABLE SET GEOMETRY=Convert3D(GEOMETRY,0);
g_3DGeom:=Convert3D(g_2dGeom,0);

Note: This command does not support owner.table. Raster and Text are not supported.

Convert2NativePt returns a 2D/3D native point geometry from an input 2D/3D oriented point geometry.

```
FUNCTION Convert2NativePt(g_geom IN MDSYS.SDO_GEOMETRY,
                          i_dim IN INTEGER DEFAULT 2)
RETURN MDSYS.SDO_GEOMETRY;
```

Syntax: g_geom:= GOOM.Convert2NativePt(g_geom,[i_dim]);
UPDATE table A SET A.geometry = GOOM.Convert2NativePt(A.geometry);
g_geom is the actual geometry (no quotes).
i_dim is the optional output dimension (2 or 3). Default=2.

Convert2OrientedPt returns a 2D/3D oriented point geometry from an input 2D/3D native pt geometry. The orientation matrix is set to 0.

```
FUNCTION Convert2OrientedPt(g_geom IN MDSYS.SDO_GEOMETRY)
RETURN MDSYS.SDO_GEOMETRY;
```

Syntax: g_geom:= GOOM.Convert2OrientedPt(g_geom);
UPDATE table A SET A.geometry = GOOM.Convert2NativePt(A.geometry);
UPDATE table A SET A.geometry = GOOM.Convert2OrientedPt(A.geometry);
g_geom is the actual point geometry (no quotes).

ConvGeom2Pts returns a point cluster geometry from an input line or area geometry.

```
FUNCTION ConvGeom2Pts(g_polygeom IN MDSYS.SDO_GEOMETRY)
RETURN MDSYS.SDO_GEOMETRY;
```

Syntax: g_geom:= GOOM.ConvGeom2Pts(g_polygeom);
UPDATE table A SET A.geometry = GOOM.ConvGeom2Pts(A.geometry);
g_geom is the actual point geometry (no quotes).

GetPtCoord returns the ordinate associated with the specified type: X, Y, or Z.

```
FUNCTION GetPtCoord(g_geom IN MDSYS.SDO_GEOMETRY, v_typ IN VARCHAR2)
RETURN NUMBER;
```

Syntax: n_coord:=GOOM.GetPtCoord(g_geom,v_type);
SELECT GOOM.GetPtGeom(A.geometry,v_type) FROM table A;

v_typ is the coordinate to return: 'X','Y', or 'Z'

GetLinearCoord returns the ordinate associated with the specified type: X,Y, or Z from either the start or the end point of the line.

```
FUNCTION GetLinearCoord(g_geom IN MDSYS.SDO_GEOMETRY,
                        v_typ IN VARCHAR2,
                        v_loc IN VARCHAR2 DEFAULT 'START')
RETURN NUMBER;
```

Syntax: n_coord:=GOOM.GetLinearCoord(g_geometry,v_typ,v_loc);
SELECT GOOM.GetLinearCoord(A.geometry,'X','START') FROM table A;
g_geometry is the actual geometry column (no quotes).
v_typ - Coordinate to return: 'X','Y', or 'Z'
v_loc - Location of Coord: 'START','END', START is DEFAULT

GetPtBearing returns the bearing (in radians or degrees) between input point 1 and input point 2.

```
FUNCTION GetPtBearing(g_geom1 IN SDO_GEOMETRY,
                      g_geom2 IN SDO_GEOMETRY,
                      v_rtype IN VARCHAR2 DEFAULT 'D')
RETURN NUMBER;
```

Syntax: n_bearing:= GOOM.GetPtBearing(g_ptgeom1,g_ptgeom1, v_rtype)
g_ptgeom1 and g_ptgeom2 are actual geometries, no quotes
v_rtype determine result in degrees or radians. D (degrees) is the default.

GetLinearBearing returns the bearing (in radians or degrees) between the start point and end point of the input linear geometry.

```
FUNCTION GetLinearBearing(g_geom IN SDO_GEOMETRY,
                          v_rtype IN VARCHAR DEFAULT 'D')
RETURN NUMBER;
```

Syntax: n_bearing:= GOOM.GetLinearBearing(g_geom,v_rtype);
Select GOOM.GetLinearBearing(<GEOMETRY>, v_rtype) FROM DUAL;
g_geom is the actual geometry column (no quotes).
v_rtype determines result in degrees or radians. D (degrees) is the default.

GetPointRotation returns the rotation (in radians or degrees) of an oriented point.

```
FUNCTION GetPointRotation(g_geom IN SDO_GEOMETRY,
                          v_type IN VARCHAR2 DEFAULT 'D')
RETURN NUMBER;
```

Syntax: n_rotation:= GOOM.GetPointRotation(g_geom,v_type);
Select GOOM.GetPointRotation(A.<GEOMETRY>, v_type) FROM table A;
g_geom is the actual geometry column (no quotes).
v_type determines result in degrees or radians. D (degrees) is the default.

SetPointRotation sets the potion in the specified geometry to the specified value in degrees.

```
FUNCTION SetPointRotation(g_geom IN SDO_GEOMETRY,
                          n_degrees IN NUMBER DEFAULT 0)
RETURN SDO_GEOMETRY DETERMINISTIC;
```

Syntax: g_geometry := GOOM.GetPointRotation(g_geom,n_degrees);
UPDATE <table> A SET A.<geometry>= GOOM.GetPointRotation(A.<GEOMETRY>, n_degrees) FROM table A;
g_geom is the actual geometry column (no quotes).
n_degrees is the angle of rotation in degrees (0 is default).

Oracle Metadata (EXEC GOOM.HelpMBR;)

CopyMBR copies the USER_SDO_GEOM_METADATA entries for one table/geometry to another one.

```
PROCEDURE CopyMBR(v_frmtable IN VARCHAR2,
                  v_frmgeom IN VARCHAR2,
                  v_totable IN VARCHAR2,
                  v_togeom IN VARCHAR2);
```

Syntax: EXEC GOOM.CopyMBR(v_source_table,v_source_geom,v_target_table,v_target_geom);
Input the source table/geometry and the target table/geometry. You can also specify a view.

DeleteOrphanMBR deletes orphan metadata from USER_SDO_GEOM_METADATA. Orphans would occur if you had entries for table/views that were no longer in the schema.

```
PROCEDURE DeleteOrphanMBR(v_schema IN VARCHAR2 DEFAULT USER);
```

Syntax: EXEC GOOM.DeleteOrphanMBR;

SetMBR sets the default MBR for a table's geometry column in USER_SDO_GEOM_METADATA.

```
PROCEDURE SetMBR(v_tablename IN VARCHAR2,
                 v_geomcol IN VARCHAR2 DEFAULT NULL,
                 i_dimension IN INTEGER DEFAULT NULL);
```

Syntax: EXEC GOOM.SetMBR(v_tablename,[v_geomcol],[i_dimension]);

v_geomcol is the geometry column. If not specified, the procedure will automatically use the first one it finds.
i_dimension is the dimension to use, only specify this (2 or 3) if the geometry is empty.

SetMBRAll sets the default MBR in USER_SDO_GEOM_METADATA for all geometry columns in the schema.

```
PROCEDURE SetMBRAll(v_schema IN VARCHAR2 DEFAULT USER,
                    i_dimension IN INTEGER DEFAULT NULL);
```

Syntax: EXEC GOOM.SetMBRAll([v_schema]);
v_schema is optional and should only be used when a DBA is running the command for the specified user.
i_dimension is the optional dimension to use, only specify this (2 or 3) if the geometry is empty.

SetMBRGeo sets USER_SDO_MEOM_METADATA to +-180, +-90 for all feature classes.

```
PROCEDURE SetMBRGeo(v_tablename IN VARCHAR2 DEFAULT 'ALL',
                    i_dimension IN INTEGER DEFAULT NULL)
```

Syntax: EXEC GOOM.SetMBRGeo([v_tablename],[i_dimension]);
v_tablename is optional. If set, only the table name is processed, otherwise all tables in the schema are processed automatically.
i_dimension is the optional dimension to use, only specify this (2 or 3) if the geometry is empty.

SetMBRProj set the default MBR in USER_SDO_GEOM_METADATA to the default for projected data +-2147483648,+-2147483648 for all feature classes.

```
PROCEDURE SetMBRProj(v_tablename IN VARCHAR2 DEFAULT 'ALL',
                     i_dimension IN INTEGER DEFAULT NULL);
```

Syntax: EXEC GOOM.SetMBRProj;
EXEC GOOM.SetMBRProj([v_tablename],[i_dimension]);
i_dimension is the optional dimension to use, only specify this (2 or 3) if the geometry is empty.

SetSRID sets the SRID value for the specified table/geometry to the specified SRID. SRID's are not required for projected data but they are required for geodetic data.

```
PROCEDURE SetSRID(v_tablename IN VARCHAR2,
                  v_geomcol IN VARCHAR2 DEFAULT NULL,
                      i_srid IN INTEGER DEFAULT 0);
```

Syntax: EXEC GOOM.SetSRID(v_tablename,v_geomcol,i_srid)
i_srid=0 sets NULL value (Default) which is generally used for projected data.

SetSRIDAll sets the specified SRID for all tables/geometries in the schema.

```
PROCEDURE SetSRIDAll(i_srid IN INTEGER DEFAULT 0,
                     v_schema IN VARCHAR2 DEFAULT USER)
```

Syntax: EXEC GOOM.SetSRIDAll(i_srid, [v_schema]);
v_schema is optional and can be used by a DBA to operate on any schema.
i_srid=0 sets NULL value (Default) generally used for projected data.

SetSpatialTolerance sets the Oracle spatial tolerance to use for the specified table/geometry in USER_SDO_GEOM_METADATA. The default tolerance is 0.00005 m and is the recommended value to use for projected data.

```
PROCEDURE SetSpatialTolerance(v_tablename IN VARCHAR2,
                              v_geomcol IN VARCHAR2 DEFAULT NULL,
                                 n_tol IN NUMBER DEFAULT 0.00005);
```

Syntax: EXEC GOOM.SetSpatialTolerance(v_tablename,v_geometry,n_tol);

SetSpatialToleranceAll sets the Oracle spatial tolerance for all table/geometries in the schema. The default tolerance is 0.00005 m and is the recommended value to use for projected data.

```
PROCEDURE SetSpatialToleranceAll(n_tol IN NUMBER DEFAULT 0.00005,
                                 v_schema IN VARCHAR2 DEFAULT USER);
```

Syntax: EXEC GOOM.SetSpatialToleranceAll(n_tol, [v_schema]);
v_schema is optional and can be used by a DBA to operate on any schema.

GetSpatialTolerance returns the current tolerance setting for the table/geometry.

```
FUNCTION GetSpatialTolerance(v_tablename IN VARCHAR2,
                             v_geomcol IN VARCHAR2 DEFAULT NULL,
                                v_dim IN VARCHAR2 DEFAULT 'X')
RETURN NUMBER;
```

Syntax: SELECT GOOM.GetSpatialTolerance(v_tablename,v_geometry,[v_dim]) FROM DUAL;
v_dim is optional and can be 'X', 'Y' or 'Z' if tolerances are different.

GDOSYS Metadata (EXEC GOOM.HelpGDOSYS;)

SetGDOSYSMetadata sets default entries in GDOSYS for the specified table/view. This creates the feature class metadata used by GeoMedia. It is the command line equivalent of running Database Utilities and just accepting the defaults. Only tables with a single primary key are supported. A DBA user must use owner.table for the table name.

```
PROCEDURE SetGDOSYSMetadata(v_tablename   IN VARCHAR2,)
                            v_seq_name    IN VARCHAR2 DEFAULT NULL,)
                                 v_cs     IN VARCHAR2 DEFAULT 'DEFAULT',)
          i_geomtype_in IN INTEGER DEFAULT NULL,)
          v_keycolumn   IN VARCHAR2 DEFAULT NULL,)
          v_num38       IN VARCHAR2 DEFAULT 'LONG',)
          v_num10       IN VARCHAR2 DEFAULT 'DOUBLE'));
```

Syntax: EXEC GOOM.SetGDOSYSMetadata(v_tablename, [v_seqname],[v_cs],[v_geomType],[v_keycolumn],[v_num38],[v_num10]);

v_tablename is the name of the table or view. It is required.

v_seqname (optional) is the name of the sequence that populates the primary key. If it does not exist, it will be created.

v_cs (optional, default D) is the csguid of the coordinate system to assign. Use 'D' to use the default coordinate system assigned to the schema.)

v_geomType (optional) is the GTYPE to use for empty geometries. Otherwise the GTYPE will be picked up automatically by sampling the geometry gtype.

v_keycolumn (optional) used only if v_tablename is actually a view, then it must be the key preserved column used in the view.

v_num38 (optional) is an override for NUMBER(38) PKEY assignments. Any value here except 'LONG' forces override. Use only if you need interoperability with ESRI NUMBER(38).

v_num10 (optional) is an override for all NUMBER(10) assignments. Forces LONG. Any value here except 'DOUBLE' forces override. Use only if you need interoperability with GTECH NUMBER(10) for G3E columns.

SetGDOSYSMetadataAlL sets default entries in GDOSYS for all tables/views in your schema. Set the USER constants prior to compiling to control how the procedure works.

PROCEDURE SetGDOSYSMetadataAll(v_schema IN VARCHAR2 DEFAULT USER));

Syntax: EXEC GOOM.SetGDOSYSMetadataAll;
EXEC GOOM.SetGDOSYSMetadataAll([v_schema]);

v_schema is optional and can be used by a DBA to operate on any schema.

SetDefaultCS sets the default coordinate system used by the specified schema. Setting a NULL for v_csguid disables the default. The CSGUID comes from the GDOSYS.GCOORDSYSTEM table.

```
PROCEDURE SetDefaultCS(v_csguid IN VARCHAR2 DEFAULT NULL,
                       v_schema IN VARCHAR2 DEFAULT USER);
```

Syntax: EXEC GOOM.SetDefaultCS(c_csguid, [v_schema]);

v_schema is optional and can be used by a DBA to operate on any schema.

SetPrimaryGeom sets the primary geometry indication in GDOSYS for the specified geometry.

```
PROCEDURE SetPrimaryGeom(v_tablename IN VARCHAR2,
                         v_geomcol IN VARCHAR2);
```

Syntax: EXEC GOOM.SetPrimaryGeom(v_tablename, v_geomcol);
v_tablename can be owner.table or just table.
v_geomcol is the name of the geometry in v_tablename that will be primary.
This is not required for tables that contain only one geometry.

GetPrimaryGeom returns the primary geometry field name used by GeoMedia. This is only necessary when more than one geometry when more than one geometry is present in a feature class.

```
FUNCTION GetPrimaryGeom(v_tablename IN VARCHAR2) RETURN VARCHAR2
```

Syntax: EXEC GOOM.SetPrimaryGeom(v_tablename, v_geomcol);
v_tablename can be owner.table or just table. This is not required for tables that contain only one geometry.

OverrideNumDatatype allows you to set the data type matching override in GDOSYS.GFIELDMAPPING for the specified table/column.

```
PROCEDURE OverrideNumDatatype(v_tablename IN VARCHAR2,
                              v_column IN VARCHAR2,
                                 v_type IN VARCHAR2);
```

Syntax: EXEC GOOM.OverrideNumDatatype(v_tablename, v_column, v_type);
v_type can be 1 for BOOLEAN, 3 for INTEGER, 4 for LONG, or 7 for DOUBLE. This only works for numeric data types.

SetField2Hypertext allows you to set a field format to HyperText for the specified table/column for use in GeoMedia. The command modifies existing GDOSYS metadata.

```
PROCEDURE SetField2Hypertext(v_tablename IN VARCHAR2,
                             v_column IN VARCHAR2);
```

Syntax: EXEC GOOM.SetField2Hypertext(v_tablename, v_column);
This only works for VARCHAR2 and CHAR data types.

DelGDOSYSMetadata deletes all the GDOSYS metadata associated with the specified table/view.

```
PROCEDURE DelGDOSYSMetadata(v_tablename IN VARCHAR2,
                            b_respn IN BOOLEAN DEFAULT TRUE);
```

Syntax: EXEC GOOM.DelGDOSYSMetadata(v_table_name);

v_tablename can be passed as a table/view for the current user. A DBA user
  can pass owner.table.

DelGDOSYSOrphans searches for orphan entries in GDOSYS related to the specified schema.

PROCEDURE DelGDOSYSOrphans(v_schema IN VARCHAR2 DEFAULT USER);

Syntax: EXEC GOOM.DelGDOSYSOrphans;
EXEC GOOM.DelGDOSYSOrphans(v_schema);
v_schema is optional, normally the current user is used. A DBA can
  specify any schema.

DeleteOrphanCS deletes orphan coordinate systems from GDOSYS.GCOORDSYSTEM. If the coordinate system is used anywhere in GDOSYS, it will not be deleted.

PROCEDURE DeleteOrphanCS(b_respn IN BOOLEAN DEFAULT TRUE)

Syntax: EXEC GOOM.DeleteOrphanCS;

CleanModLog is used to truncate the GDOSYS.ModificationLog and ModifiedTables tables.

PROCEDURE ClearModLog(v_user IN VARCHAR2 DEFAULT NULL);

Syntax: EXEC GOOM.ClearModLog;
Generally this should be run by a DBA user or as a system level job. An option
  was added to allow EXEC GOOM.ClearModLog(USER); but this will be slow.

LogDataModification will log and inset, update, or delete operation into GDOSYS's MODIFIEDTABLES and MODIFICATIONLOG tables. This is useful when writing modlog triggers or doing customization that requires external notification for GM users.

```
PROCEDURE LogDataModification(v_tablename IN VARCHAR2,
                              v_keyname IN VARCHAR2,
                              v_keyvalue IN VARCHAR2,
                              i_modtype IN INTEGER);
```

Syntax: EXEC GOOM.LogDataModification(v_tablename, v_keyname, i_keyvalue, i_modtype);
v_tablename is the table being modified.
v_keyname is the primary key column name for the table.
v_keyvalue is the primary key row identifier being modified (integer or character).
i_modtype is the type of modification: 1 for insert, 2 for update, and 3 for delete.
Note: The procedure only works with single, numeric based primary keys.

Spatial Indexing and Analysis (EXEC GOOM.HelpTuning;)

Autotune is a multi-function that will tune a schema that has been loaded using GeoMedia's Export to Oracle Object Model data. The process repairs NULL geometries, sets default Oracle metadata (MBR), create spatial indexes, and generates statistics. You do not need to use it on existing schemas, it is meant to be used on newly imported schemas.

PROCEDURE AutoTune(v_schema IN VARCHAR2 DEFAULT USER);

Syntax: EXEC GOOM.Autotune([v_schema]);
v_schema is optional and can be used by a DBA to operate on any schema.

DelSIDX deletes all the spatial indexes in the specified schema.

PROCEDURE DelSidx(v_schema IN VARCHAR2 DEFAULT USER)

Syntax: EXEC GOOM.DelSIDX; for the current schema.
EXEC GOOM.DelSIDX(v_schema); as a DBA operation.

DropSidx drops the spatial index on the specified table/geometry pair.

```
PROCEDURE DropSidx(v_tablename IN VARCHAR2,
                   v_geomcol IN VARCHAR2 DEFAULT NULL)
```

Syntax: EXEC GOOM.DropSIDX(v_table_name,v_column_name)

SpatialIndex will create a spatial index on all geometry columns in the specified table.

```
PROCEDURE SpatialIndex(v_tablename IN VARCHAR2,
                       b_geomopt IN BOOLEAN DEFAULT TRUE);
```

Syntax: EXEC GOOM.SpatialIndex(v_tablename, [b_geomopt]);
b_geomopt will create a geometry optimized index by default. Set to FALSE if you do not want the geometry to be optimized. Optimization constrains the GTYPE so no other GTYPE can be introduced into the geometry.

SpatialIndexAll will create a spatial index for all geometry based columns in the schema. A DBA can specify any schema for this operation, otherwise it operates on the current schema.

```
PROCEDURE SpatialIndexAll(v_schema IN VARCHAR2 DEFAULT USER,
                          b_geomopt IN BOOLEAN DEFAULT TRUE);
```

Syntax: EXEC GOOM.SpatialIndexAll([c_owner],[b_geomopt]);
b_geomopt will create a geometry optimized index by default. Set to FALSE if you do not want the geometry to be optimized. Optimization constrains the GTYPE so no other GTYPE can be introduced into the geometry.

Stats calculates the statistics for the objects in the current (or specified) schema.

PROCEDURE Stats(v_schema IN VARCHAR2 DEFAULT USER);

Syntax: EXEC GOOM.STATS([v_schema]);
v_schema is optional and can be used by a DBA to operate on any schema.

DelStats deletes the database statistics associated with the current (or specified) schema.

PROCEDURE DELStats(v_schema IN VARCHAR2 DEFAULT USER);

Syntax: EXEC GOOM.DelSTATS([v_schema]);
v_schema is optional and can be used by a DBA to operate on any schema.

GetGOOMIndexSpace returns the INDX space currently used by the spatial indexing commands.

FUNCTION GetGOOMIndexSpace
RETURN VARCHAR2;

Syntax: Select GOOM.GetGOOMIndexSpace from dual;

SetGOOMIndexSpace sets the default spatial indexing tablespace.

PROCEDURE SetGOOMIndexSpace(v_indexspace IN VARCHAR2);

Syntax: EXEC GOOM.SetGOOMIndexSpace(v_indexspace);
Set v_indexspace to 'D' to revert to system default of USERS or INDX

Utilities (EXEC GOOM.HelpUtilities;)

VERSION returns package version and date information.

PROCEDURE VERSION;

Syntax: EXEC GOOM.version;

DBMsg is used to send DBMS_OUTPUT messages back to the console. Messages require SERVEROUTPUT to be enabled -> SET SERVEROUTPUT ON.
.

```
PROCEDURE DBMSG(v_Msg IN VARCHAR2,
                i_maxlen IN INTEGER DEFAULT 255);
```

Syntax: EXEC GOOM.DBMSG(c_text, [i_max_length]);
Input c_text, default max length is 255 characters.

Response uses DBMsg to send a formatted response from a procedure or function. The response is in two parts, the operation and then the result. The padding is … specified by the length. By default the response portion is left justified (best for text) but you can also specify right justification.

```
PROCEDURE Response(v_operation IN VARCHAR2,
                   v_results IN VARCHAR2,
                     i_pad IN INTEGER DEFAULT 30,
                 b_RtJustify IN BOOLEAN DEFAULT FALSE);
```

Syntax: EXEC GOOM.Response(v_operation,v_result,[i_pad],[b_rjustify];
v_operation can be the procedure name or anything else up to 30 characters.
v_results is the information you want to pass to the user.
i_pad is the number of spaces between operation and result (default 30).
b_RtJustify determines if the output should be right justified (default FALSE).

DashLine generates a dashed (---) line of specified length.

PROCEDURE DashLine(i_length IN INTEGER DEFAULT 80);

Syntax: EXEC GOOM.DashLine([i_length]);
Default length is 80.

DotLine generates a dotted (...) line of specified length.

PROCEDURE DotLine(i_length IN INTEGER DEFAULT 80)

Syntax: EXEC GOOM.DotLine([i_length]);
Default length is 80.

DblLine generates a double (===) line of specified length.

PROCEDURE DblLine(i_length IN INTEGER DEFAULT 80)

Syntax: EXEC GOOM.DblLine([i_length]);
Default length is 80.

GenLine generates a line of any character for any length.

```
PROCEDURE GenLine(v_char IN VARCHAR2,
                  i_length IN INTEGER DEFAULT 80);
```

Syntax: EXEC GOOM.GenLine(v_char, [i_length]);
Default length is 80.

TitleBlock generates a formatted title block; dash line, centered title, then a dot line.

```
PROCEDURE TitleBlock(v_title IN VARCHAR2,
                     i_length IN INTEGER DEFAULT 80,
                       v_sep IN VARCHAR2 DEFAULT '-');
```

Syntax: EXEC GOOM.TitleBlock(v_title,[i_length], [v_v_sep]);

c_title is the centered title line (required).
i_length is the length of the title block, default 80.
c_separator is the character separator, default is -.

TitleLine is similar to **TitleBlock** but only generate the centered title line.

```
PROCEDURE TitleLine(v_title IN VARCHAR2,
                    i_length IN INTEGER DEFAULT 80,
                      v_sep IN VARCHAR2 DEFAULT '-');
```

Syntax: EXEC GOOM.TitleLine(v_title,[i_length], [v_v_sep]);
v_title is the centered title line (required).
i_length is the length of the title block, default 80.
v_separator is the character separator, default is -.

ProcessStart generates a process start message that is useful when running a string of procedures.

```
PROCEDURE ProcessStart(v_cmdname IN VARCHAR2,
                       v_schema IN VARCHAR2 DEFAULT USER)
```

Syntax: EXEC ProcessStart(v_procname, v_schema);
v_cmdname is the name of the running process.
v_schema is the user/schema where process is running.

ProcessComplete is used in conjunction with ProcessStart to indicate that a process has ended (used when running a string of procedures).

```
PROCEDURE ProcessComplete(v_cmdname IN VARCHAR2,
                          v_schema IN VARCHAR2 DEFAULT USER)
```

Syntax: EXEC ProcessComplete(v_cmdname, v_schema);
v_cmdname is the name of the running process.
v_schema is the user/schema where process is running.

ProcessTerminate is similar to process complete but used in an exception block when an error has terminated the running processes.

```
PROCEDURE ProcessTerminate(v_cmdname IN VARCHAR2,
                           v_schema IN VARCHAR2 DEFAULT USER);
```

Syntax: EXEC ProcessTerminate(v_cmdname, v_schema);
v_cmdname is the name of the running process.
v_schema is the user/schema where process is running.

REPORT_ERROR outputs a formatted error message to the user.

```
PROCEDURE REPORT_ERROR(v_cmdname IN VARCHAR2,
                       v_process IN VARCHAR2,
                       v_debug IN VARCHAR2,
                       v_SQLCODE IN VARCHAR2,
                       v_SQLERRM IN VARCHAR2)
```

Syntax: EXEC GOOM.REPORT_ERROR (v_cmdname,v_process,v_debug,v_SQLCODE,v_SQLERRM);
v_cmdname is the name of the procedure/function generating the error.
v_process is the process being run when error occurred.
v_debug is a debug msg embedded in the procedure/function.
v_SQLCODE,v_SQLERRM are system level codes from Oracle, use SQLCODE and SQLERRM.

WRITE_RESULTS will write operations and results to the GOOM_LOG table for later review if logging is turned on. Logging is disabled by default.

```
PROCEDURE WRITE_RESULTS(v_schema IN VARCHAR2 DEFAULT USER,
                        v_type IN VARCHAR2,
                        v_cmd IN VARCHAR2,
                        v_feature IN VARCHAR2,
                        v_result IN VARCHAR2);
```

Syntax: EXEC GOOM.WRITE_RESULTS(v_type,v_cmd,v_feature,v_result);
v_type is the category of the process, use to sort log table.
v_cmd is the procedure or function that is running.
v_feature is the feature class being processed.
v_result is the result being logged.

Validation and Repair (EXEC GOOM.HelpValidation;)

FixRedundantPoints uses Oracle's REMOVE_DUPLICATE_VERTICES to remove redundant points based on the tolerance value.

```
PROCEDURE FixRedundantPoints(v_tablename IN VARCHAR2,
                             v_geomcol IN VARCHAR2 DEFAULT NULL,
                             n_tol IN NUMBER DEFAULT 0.00005);
```

Syntax: EXEC GOOM.FixRedundantPoints(v_table_name,v_geomcol,[n_tol]);
n_tol is optional. If not specified, the default of 0.00005 will be used.
Note: Remember that tolerance is based on units of storage.

FixSmallArcs will find and stroke arcs (into line strings) that meet the chord length and tolerance criteria. Small arcs can cause problems in both GeoMedia and in Oracle's own spatial calculations.

```
PROCEDURE FixSmallArcs(v_tablename IN VARCHAR2,
                       v_geomcol IN VARCHAR2 DEFAULT NULL,
                                   v_length IN NUMBER DEFAULT 0.05,
                                     n_tol IN NUMBER DEFAULT 0.001);
```

Syntax: EXEC GOOM.FixSmallArcs(c_table_name,c_geom_column,[n_chordlength],[n_tolerance]);
n_chordlength is the length of the chord that measures the tolerance. Default 0.05.
n_tol is the distance between the arc and the chord length. Default 0.001.
Note: The defaults are representative of small arcs and work well. Keep in mind that these values are based on the units of storage.

FixNullGeom repairs NULL (uninitialized) geometries caused by using SQL Loader. It does this by initializing the geometry column and making it empty.

PROCEDURE FixNullGeoms(v_schema IN VARCHAR2 DEFAULT USER);

Syntax: EXEC GOOM.FixNullGeom;
EXEC GOOM.FixNullGeom([v_schema]);
v_schema is optional, normally the current schema is used. A DBA can specify any schema. Note: This procedure does not harm existing data.

FixTrainingSpaces removes trailing spaces from all VARCHAR2 columns in a schema.
This is sometimes an issue when importing data from Microsoft Access based warehouses.

PROCEDURE FixTrailingSpaces(v_schema IN VARCHAR2 DEFAULT USER)

Syntax: EXEC GOOM.FixTrailingSpaces;
Note: This procedure does not harm existing data.

GetError returns a text description of the spatial error code (from ValidateGeom).

FUNCTION GetError(v_error VARCHAR2 DEFAULT 'EMPTY')
RETURN VARCHAR2;

Syntax: v_RESULT:=GOOM.GetError(v_error);
SELECT GOOM.GetError(v_error) FROM DUAL;

RepairGeometry uses Oracle's Rectify_Geometry to repair the geometry. Rectify fixes redundant points (ORA-13356), reversed polygons (ORA-13367), and self-intersecting polygons (ORA-31349).

```
PROCEDURE RepairGeometry(v_tablename IN VARCHAR2,
                                     v_geomcol IN VARCHAR2 DEFAULT NULL,
                                           n_tol IN NUMBER DEFAULT 0.00005)
```

Syntax: EXEC GOOM.RepairGeometry(v_tablename,v_geomcol, n_tol);
n_tol is optional. If not specified, the default of 0.00005 will be used.
Note: Remember that tolerance is based on units of storage. This procedure does not support owner.table. Be careful with this one as it may be data destructive.

ValidateGeom uses Oracle's VALIDATE_LAYER_WITH_CONTEXT to validate the geometry in the feature class. It handles the error table as well as other bookkeeping.

```
PROCEDURE ValidateGeom(v_tablename IN VARCHAR2,
                       v_geomcol IN VARCHAR2 DEFAULT NULL);
```

Syntax: EXEC GOOM.ValidateGeom(v_tablename,v_geomcol);
v_tablename CANNOT use OWNER.TABLE. The owner of the table must run the command.
v_geomcol is the geometry column to be validated.

AnalyzeGeometry looks at a single geometry value and tests it for validity. It will return a detailed error message. If an error condition exists, a detailed analysis will be returned to the console (if SERVEROUTPUT is ON). If you want the detailed analysis regardless of an error condition, set the v_vebose flag to TRUE.

```
FUNCTION AnalyzeGeometry ( g_geometry IN SDO_GEOMETRY,
                           v_verbose IN VARCHAR2 DEFAULT 'FALSE')
RETURN VARCHAR2;
```

Syntax: SELECT GOOM.AnalyzeGeometry(g_geometry) FROM table WHERE PKID=val;

Tables and Sequences (EXEC GOOM.HelpTables;)

DropTable drops the specified table and all its related metadata.

```
PROCEDURE DropTable( v_tablename IN VARCHAR2);
```

Syntax: EXEC GOOM.DropTable('TABNAME');

CopyTable create a copy of the source table. Options included spatial index creation, Data inclusion, and GDOSYS metadata creation. The default is True, True, True.

```
PROCEDURE CopyTable(v_srcownertable IN VARCHAR2,
        v_tgtownertable IN VARCHAR2,
        b_sidx      IN BOOLEAN DEFAULT TRUE,
        b_data      IN BOOLEAN DEFAULT TRUE,
        b_gdosys    IN BOOLEAN DEFAULT TRUE);
```

Syntax: EXEC GOOM.CopyTable(v_sourcetabname,v_copytabname);

CreateSequence creates a Sequence for specified table/column. Sequence will start at MAX(col)+1.

PROCEDURE CreateSequence(v_tablename IN VARCHAR2, v_Column IN VARCHAR2 DEFAULT NULL);
Syntax: EXEC GOOM.CreateSequence(v_tablename,[v_column]);
  v_tablename is required.
  v_column is optional but is generally the column using the sequence.

CreateNewSequence will create a new sequence for the specified table and optional columnname and will return the sequence name.

```
PROCEDURE CreateNewSequence(v_tablename IN VARCHAR2,
                            v_Column IN VARCHAR2 DEFAULT NULL,
                          v_seqname OUT VARCHAR2);
```

Syntax: EXEC GOOM.CreateNewSequence(v_tablename,[v_column],v_seqname);
  v_tablename is required.
  v_column is optional but is generally the column using the sequence.
  v_seqname must be declared in the calling procedure.

DelDupRows deletes duplicate records in a table based on the specified column.

PROCEDURE DelDupRows(v_tablename IN VARCHAR2, v_column IN VARCHAR2);

Syntax: EXEC GOOM.DelDupRows(v_tablename, v_column);

Warning: This procedure is destructive, make sure you understand what it is doing.

DropSequence drops the specified sequence if it exists.

PROCEDURE DropSequence(v_sequence IN VARCHAR2);

Syntax: EXEC GOOM.DropSequence(v_sequence);
  v_sequence is the sequence name**.**

GetSequenceName returns a unique sequence name based on the table/column name.

```
FUNCTION GetSequenceName(v_tablename IN VARCHAR2,
                        v_column IN VARCHAR2 DEFAULT NULL,
                         b_unique IN BOOLEAN DEFAULT FALSE)
RETURN VARCHAR2;
```

Syntax: v_seqname:=GOOM.GetSequenceName(v_tablename,[v_column]);
v_tablename is required.
v_column is optional but is generally the column using the sequence.

AddPrimaryKey adds a new integer based primary key column to a table, creates a sequence, and then updates the key with sequence values. You can choose to keep the current key (if it exists) as a normal column or delete it.

```
PROCEDURE AddPrimaryKey(v_table IN VARCHAR2,
                        v_keycolnew IN VARCHAR2 DEFAULT 'PID',
                          b_dropflag IN BOOLEAN DEFAULT TRUE);
```

Syntax: EXEC GOOM.AddPrimaryKey(v_table, [v_keycolnew], [TRUE]);
v_table is required and can be in the form of OWNER.TABLE.
v_keycolnew is the optional new column name. PID is default.
b_dropflag is the optional Boolean to drop the current key column if it exists.

Utility Functions (EXEC GOOM.HelpFunctions;)

FUNCTION Ang2Rad (n_degrees IN NUMBER) RETURN NUMBER;
Returns converts the input degrees to radians.
Syntax: n_radians := Ang2Rad(n_degrees);

FUNCTION Rad2Ang (n_radians IN NUMBER) RETURN NUMBER;
Returns the input radians in degrees.
Syntax: n_degrees := Rad2Ang(n_radians);

FUNCTION ROTINDEX (v_type IN VARCHAR2, n_degrees IN NUMBER) RETURN NUMBER;
Returns the rotation matirx index value for the input angle in degrees.
Syntax: n_irot := ROTINDEX('I',n_degrees);
n_jrot := ROTINDEX('J',n_degrees);

FUNCTION ROTANGLE (n_irot IN NUMBER, n_jrot IN NUMBER) RETURN NUMBER;
Returns the rotation angle in degrees from the input i, j rotation indices.
Syntax: n_degrees := ROTANGLE(n_irot, n_jrot);

FUNCTION DMS2DD(v_dms IN VARCHAR2) RETURN NUMBER;
Returns Degree:Minute:Seconds from Decimal Degrees.
Syntax: n_dec_degrees := DMS2DD(v_dms);
SELECT DMS2DD('40:20:50') FROM DUAL;
v_dms is colon (:) delimited by default.

FUNCTION DD2DMS(n_dd IN NUMBER, v_delim IN VARCHAR2 DEFAULT ':') RETURN VARCHAR2;

Returns Decimal Degrees from Lat or Lon: Degree, Minute, Seconds using delimiter.
Syntax: v_dec_degrees := DMS2DD(v_dms,v_delim);
v_dms := DD2DMS(n_dd,v_delim);
SELECT DD2DMS(40.3472222,':') FROM DUAL;
v_delim can be any single character like ':'

FUNCTION RandInRange(i_lo, i_hi) RETURN NUMBER;
RandInRange returns a random real number between the integer values i_lo and i_hi.
Syntax: n_value:=GOOM.RandInRange(i_lo, i_hi);

FUNCTION Integer2Text(i_integer IN INTEGER) RETURN VARCHAR2;
Returns 4 bytes of text from the input integer.
Syntax: v_text:=GOOM.Integer2Text(i_integer);

FUNCTION Text2Integer(v_string IN VARCHAR2) RETURN INTEGER;
Returns an integer value from 4 bytes of text. Anything more will be truncated.
Syntax: i_integer:=GOOM.Text2Integer(v_string);

FUNCTION Integers2String(v_string IN VARCHAR2) RETURN VARCHAR2;
Returns a text string from a comma delimited string of integers.
Syntax: v_string:=GOOM.Integers2String(v_integerList);

FUNCTION String2Integers(v_string IN VARCHAR2) RETURN VARCHAR2;
Returns a comma delimited string of integers, every 4 bytes (incl. spaces) is represented by one integer.
Syntax: i_integerList:=GOOM.String2Integers(v_string);

FUNCTION SplitOwnerObject(v_tablename IN VARCHAR2, v_type IN VARCHAR2) RETURN VARCHAR2;
Returns either the Owner name or the Object name from the owner.object format.
Syntax: v_schema:=GOOM.SplitOwnerObject(v_tablename,v_type);
v_schema:=GOOM.SplitOwnerObject(v_tablename,'OWNER');
v_table:=GOOM.SplitOwnerObject(v_tablename,'OBJECT');
Valid c_types are OWNER/USER or TABLE/OBJECT (use TABLE for views).

Get Functions (EXEC GOOM.HelpFunctions;)

FUNCTION GetDBNAME RETURN VARCHAR2;
Returns the global database name.
Syntax: Select GOOM.GetDBNAME from DUAL;

FUNCTION GETDBVERSION RETURN VARCHAR2;
Returns the Database Version number.
Syntax: Select GOOM.GetDBVersion from DUAL;

FUNCTION GetOwnerObject(v_tablename) RETURN VARCHAR2;
Returns the tablename in the format or OWNER.TABLE
Syntax: v_ownertable:=GOOM.GetOwnerObject(v_tablename);

FUNCTION GetDim(v_tablename,v_geometry) RETURN VARCHAR2;
Returns the dim of a layer - 2 or 3 as VARCHAR2(4)
Syntax: v_dim:=GOOM.GetDim(v_tablename,v_geometry);

FUNCTION GetGeom(v_tablename) RETURN VARCHAR2;
Returns the 1st geometry column for a layer as VARCHAR2
Syntax: g_geometry:=GOOM.GetGeom(v_tablename);

FUNCTION GetSRID(v_tablename,v_geometry) RETURN INTEGER;
Syntax: i_srid:=GOOM.GetSRID(v_tablename,v_geometry);
Returns the 1st srid found or NULL.

FUNCTION GetCSName(i_srid) RETURN VARCHAR2;
Returns the CS name of the provided SRID.
Syntax: v_csname:=GOOM.GetSRID(i_srid);

FUNCTION isGeographic (i_srid IN INTEGER DEFAULT 0) RETURN BOOLEAN;
If the SRID is a geographic srid return TRUE.
Syntax: If isGeographic THEN

FUNCTION GetINDXNAME(v_tablename, v_extend) RETURN VARCHAR2;
Returns a unique index name based on tablename VARCHAR2(30) and a 3 character extension. The default extension is _SI
Syntax: v_index:=GOOM.GetINDXNAME(v_tablename, v_extend);

FUNCTION GetSequenceName (v_tablename , v_column, b_unique) RETURN VARCHAR2;
Returns sequence name based on input table and column.
Syntax: v_seqname:=GOOM.GetSequenceName(v_tablename,v_column,v_unique);
v_column is optional and the default is NULL.
b_unique is optional and the default is FALSE, if TRUE, a unique name will be generated.

FUNCTION GetGTYPE(v_tablename, v_geomcol) RETURN VARCHAR2;
Returns the GTYPE as VARCHAR2(4)
Syntax: v_GTYPE:=GOOM.GetGTYPE(v_tablename,v_geomcol);

FUNCTION GetGeomType(v_tablename, v_geomcol) RETURN VARCHAR2;
Returns the GeomType as VARCHAR2(16)
Syntax: v_GeomType:=GOOM.GetGeomType(v_tablename,v_geomcol);
Possible types are POINT, LINE, POLYGON, COLLECTION, MULTIPOINT, MULTILINE, MULTIPOLYGON and UNKNOWN.

FUNCTION GetGDOTableName (v_tabletype) RETURN VARCHAR2;
Returns the GDOSYS.tablename for the given geomedia gdo tabletype.
Syntax: v_gdotable:=GOOM.GetGDOTableName(v_tabletype);
Valid v_tabletype values are:
'INGRFieldLookup','GOracleFieldMapping','INGRAttributeProperties'

'INGRGeometryProperties','INGRFeatures','INGRPickLists'
'GCoordSystemTable','GOracleIndexColumns','GParameters'

FUNCTION GetCount(v_tablename) RETURN INTEGER;
Returns the Row count as an INTEGER
Syntax: i_count:=GOOM.GetCount(v_tablename);

FUNCTION GetTableSize(v_tablename) RETURN NUMBER;
Returns the segment size for the specified table in MB
Syntax: n_size:=GOOM.GetTableSize(v_tablename);

FUNCTION GetError(v_error) RETURN VARCHAR2;
Returns a text description of spatial error codes as VARCHAR2(255)
Syntax: v_RESULT:=GOOM.GetError(v_error);

FUNCTION GetKeyCol(v_tablename) RETURN VARCHAR2;
Returns the primary key column of a table assuming it only has one.
Returns NO_KEY if a key does not exist or there is more than 1.
Syntax: v_keycol:=GOOM.GetKeyCol(v_tablename);

FUNCTION GetViewKeyCol(v_viewname) RETURN VARCHAR2;
Returns the primary key column of a view assined in GDOSYS.
Returns 'NO_KEY' if a key does not exist or there is more than 1.
Syntax: v_keycol:=GOOM.GetViewKeyCol(v_viewname);

FUNCTION GetColumnType (v_tablename, v_keycol) RETURN VARCHAR2
Returns the data type for the specified primary key.
Syntax: v_keytype:=GOOM.GetColumnType(v_tablename,v_keycol);

FUNCTION GOOM.GetPrimaryGeom(v_tablename) RETURN VARCHAR2
Returns the primary geometry used by GeoMedia when a FC contains more than one geometry field.
Syntax: v_primarygeom:=GOOM.GetPrimaryGeom(v_tablename);

Check Functions (EXEC GOOM.HelpFunctions;)

FUNCTION chkTable(v_tablename) RETURN BOOLEAN;
Returns TRUE if table exists otherwise FALSE as BOOLEAN
Syntax: IF GOOM.ChkTable(v_tablename) THEN

FUNCTION chkView(v_tablename) RETURN BOOLEAN;
Returns TRUE if this is a view otherwise FALSE as BOOLEAN
Syntax: IF GOOM.ChkView(v_tablename) THEN

FUNCTION chkTrigger(v_trigname) RETURN BOOLEAN;
Returns TRUE if trigger exists otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkTrigger(v_trigname) THEN

FUNCTION chkGeometry(v_tablename, v_geomcol IN VARCHAR2) RETURN BOOLEAN;
Returns TRUE if this is a geometry otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkGeometry(v_tablename,v_geomcol) THEN

FUNCTN chkSequence(v_seqname) RETURN BOOLEAN;
Returns TRUE if sequence exists otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkSequence(v_seqname) THEN

FUNCTION chkIndex(v_indexname) RETURN BOOLEAN;
Returns TRUE if index exists otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkIndex(v_indexname) THEN

FUNCTION chkSpatialIndex(v_tablename, v_geomcol) RETURN BOOLEAN;
Returns TRUE if spatial index exists otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkSptaialIndex(v_tablename,v_geomcol) THEN

FUNCTION chkMetadata(v_tablename) RETURN BOOLEAN;
Returns TRUE if metadata exists otherwise FALSE as BOOLEAN.
Syntax: IF GOOM.ChkMetadata(v_tablename) THEN

FUNCTION ChkMBR(v_tablename, v_geomcol) RETURN BOOLEAN;
Returns TRUE if Oracle metadata exists otherwise FALSE as BOOLEAN.
Syntax: IF GOOM.ChkMBR(v_tablename,v_geomcol) THEN

FUNCTION chkPKEY(v_tablename,v_column) RETURN BOOLEAN;
Returns TRUE if column is a key column otherwise FALSE as BOOLEAN
Syntax: IF GOOM.chkPKEY(v_tablename,v_column) THEN

FUNCTION chkInsertOnMDSYS RETURN BOOLEAN;
Returns TRUE if user has insert privileges on MDSYS.SDO_GEOM_METADATA_TABLE, otherwise FALSE. If TRUE, a user can insert Oracle metadata for a table they do not own but have privileges on.
Syntax: IF GOOM.chkInsertOnMDSYS THEN

APPENDIX B: THE GDOBKP PACKAGE

The GDOBKP package provides procedures that will back up the metadata stored in GDOSYS for the schema where the procedures are run. The metadata is stored in the schema in a table called GDOSYS_BKP. Coordinate system information is stored in GCOORDSYS_BKP and default coordinate system information is stored in GCOORDSYS_DEF.

Run the script *GDOBKP_PKG.sql* from a DBA user to compile and load the package. The package will be installed into GDOSYS and the commands will be available to all PUBLIC users. They can also be used in your own scripts. To see a list of available commands and syntax after the GDOBKP package has been loaded, enter the following at the SQL Prompt:

```
EXEC GDOBKP.HELPME;
```

Main Procedures

EXEC GDOBKP.VERSION

- Gets the version and date of this package.

EXEC GDOBKP.HELPME

- Displays the procedure syntax.

EXEC GDOBKP.SaveGDOSYSMetadata; or exec GDOBKP.SaveGDOSYSMetadata(c_owner);

- Save GDOSYS Metadata for all feature classes in the schema.
- DBA users can run this remotely by specifying the schema name.
- Run this prior to exporting your schema.

EXEC GDOBKP.RestoreGDOSYSMetadata;

- This MUST be run by the schema owner (sorry)
- Restore saved GDOSYS Metadata for all feature classes in the schema.

Misc Procedures

EXEC GDOBKP.DeleteGDOSYSMetadata; or exec GDOBKP.DeleteGDOSYSMetadata(c_owner);

- DBA users can run this remotely by specifying the schema name.
- Deletes GDOSYS metadata for all the feature classes in the schema.
- Careful here, this is not recoverable. Make a backup first.

EXEC GDOBKP.DeleteBackupTables; or exec GDOBKP.DeleteBackupTables(c_owner).;

- DBA users can run this remotely by specifying the schema name.
- Deletes the tables used by this package to backup metadata.

EXEC GDOBKP.RestoreGCoordSystem; or exec GDOBKP.RestoreGCoordSystem(c_owner);

- DBA users can run this remotely by specifying the schema name.
- Restores the coordinate systems stored in GCOORDSYS_BKP.
- The default coordinate system, if any, is also restored.

EXEC GDOBKP.SaveGCoordSystem; or exec GDOBKP.SaveGCoordSystem(c_owner);

- DBA users can run this remotely by specifying the schema name.
- Backs up just the coordinate systems in use to GCOORDSYS_BKP.
- The default coordinate system, if any, is stored in GCOORDSYS_DEF.

EXEC GDOBKP.VerifyGDOSYSBkp; or exec GDOBKP.VerifyGDOSYSBkp(c_owner);

- DBA users can run this remotely by specifying the schema name.
- Verify the backup. Compares existing metadata to the backup and checks that the counts match.

EXEC GDOBKP.VerifySequenceOwner; or exec GDOBKP.VerifySequenceOwner(c_owner);

- DBA users can run this remotely by specifying the schema name.
- Verify the ownership of sequences. Compares sequence ownership in the schema with those in GFIELDMAPPING and corrects any problem.

EXEC GDOBKP.VerifyLibraryObjectTypeRef;

- Verify the ownership of tables in the TABLENAME field of the LIBRARYOBJECTTYPE table. This only works for Library schemas. If you are restoring metadata for a library schema to a schema using a different name, this is required.

APPENDIX C: AVAILABLE SQL SCRIPTS

Here is a list of the scripts available in OOM and a brief description of its use. These scripts are meant as examples and can be modified to suit your own needs.

SQL Script	Description
AddChgKeyColumn.sql	Change the key column of a table
ChngGMpt2OOpt.sql	Convert 9i GM oriented points to 10G Oracle oriented points.
ChngPrimaryGeom.sql	Change the primary geometry field of a table.
CreateCatalog.sql	Create a catalog of the current schema. The catalog lists all the tables and the geometry types stored in those tables.
CreateMBRFeatureClass.sql	Creates a feature class that is the MBR geometry for another feature class.
CreateReferenceGrid.sql	Creates a feature class that is a reference grid of area features. Useful for spatial filter areas and reference features.
CreateSeqTriggers.sql	Creates a sequence trigger for the specified table.
DeleteTab.sql	@DeleteTable <tabname> Delete a table.
FixRedundantPoints.sql	Calls the GOOM Package procedure to fix redundant points in an input geometry.
FixSmArcs.sql	Calls the GOOM Procedure. Fixes tiny arcs so GeoMedia can handle them correctly. It does this by stroking the arcs into line strings.
GrantEDIT.sql	Grant Edit Only privileges to specified user
GrantGDO.sql	Grants privileges to GDOSYS based on 3 types of users; master, editor, or reader.
GrantRevoke.sql	Revoke All privileges from specified user. Run in the user requesting the revocation.
GrantRO.sql	Grant Read Only privileges to specified user. Run in the grantor user account.
GrantRW.sql	Grant RW privileges to specified user. Run in the grantor user account.
InsertXY.sql	This script extracts the X and Y values from a table and writes them to a geometry column that has been added to that table.
InsertXYZ.sql	This script extracts the X, Y, and Z values from a table and writes them to a geometry column that has been added to that table.

SQL Script	Description
ListCatalog.sql	Decode contents of catalog table created with the *Createcatalog* script.
ListCS.sql	List the current coordinate systems stored in GDOSYS.GCOORDSYSTEM.
ListCSAll.sql	List the current coordinate system assignments and the default. This script lists coordinate systems that have no associated feature class. These are considered orphaned coordinate systems and may be deleted to improve overall performance.
ListCSorphans.sql	This script lists coordinate systems that have no associated feature class. These are considered orphaned coordinate systems and may be deleted to improve overall performance.
ListCSUser.sql	Coordinate system assignments for current schema.
ListDefaultCS.sql	List the current assigned default coordinate system for this user.
ListGDOSYSOrphans.sql	List any orphan metadata in GDOSYS the current user.
ListGOOMLOG.sql	Generate a nice output format for the GOOM_LOG file.
ListMBR.sql	Return the MBR information from USER_SDO_GEOM_METADATA
ListMetadata.sql	List the GDOSYS metadata assigned to a table.
ListModifications.sql	List Modifications from the GDOSYS.ModificationLog table.
ListSIDX.sql	List the spatial indexes
ListSRIDsGeo.sql	List available SRID's for geographic data.
ListSRIDsProj.sql	List available SRID's for projected data.
ListSRIDsUnit.sql	List the Unit conversion SRID's
ListValResults.sql	List validation results generated by the GOOM.ValidateGeom command. The Validation results should be in the table_val table.
ListUserPrivs.sql	List the privileges associated with a user
MBR_Calc.sql	Returns the extents of a layer for use in USER_SDO_GEOM_METADATA.
MBR_COPY.sql	Copy MBR values (from USER_SDO_GEOM_METADATA) from one feature class to another.
SetDefaultCS.sql	Set the Default Coordinate System to use in this schema. Lists existing CS's and allows user to pick the one to use.

SQL Script	Description
SetSRID.sql	Set the SRID for a specified table.geometry.
SetSRIDAll.sql	Set the SRID for all the table.geometry combinations in the schema.
ShowGeomByKey.sql	Returns a readable SDO_GEOMETRY based on key column
ShowGeomByROWID.sql	Returns a readable SDO_GEOMETRY based ROWID
SpatialIndex.sql	Calls the GOOM process to spatially index a table.
Validate.sql	Data validation script.

Scripts for DBA's only:

SQL Script	Description
DBAdduser.sql	Creates a user account where user and password are the same. Automatically uses default tablespace.
DBCreateUser.sql	Creates a user account where user and password are the same. Lets you choose tablespace.
DBInstallInfo.sql	List what you have installed in Oracle and its version. Also list possible invalid objects.
DBKeepPoolCanidates.sql	List possible candidates for the KEEP Pool
DBKeepPoolStartupTrig.sql	Set up a trigger to load KEEP pool.
DBVerifySpatial.sql	Scans and verifies all Locator/Spatial objects.
DBViewMods.sql	Creates a view in GDOSYS that allows you to read and review information in the MODIFICATIONLOG table

Data Type Scripts (Use only if instructed to do so or you know what you are doing):

SQL Script	Description
AutoNumber2Double_trg.sql	This trigger should be installed in GDOSYS and is AFM and SFM compatible. In order for the trigger to work, GDOSYS must have the following privilege: grant select any table to GDOSYS;

SQL Script	Description
	This trigger checks incoming feature classes whose primary key should be double but is set to integer due to the presence of an autonumber sequence. The type 4 integer in GDOSYS.GFIELDMAPPING is converted to a type 7 double for the inserted feature class.
Number10toLong.sql	This script overrides the type 7 Double for primary keys in GDOSYS.GFIELDMAPPING and forces a conversion to a type 4 LONG INTEGER. This procedure should be run once in the USER that owns the feature classes in question.
OverrideGField_trig.sql	This trigger should be installed in GDOSYS and is NOT AFM compatible. This trigger checks incoming feature classes whose primary key should be double but is set to integer due to the presence of an autonumber sequence. The type 4 integer in GDOSYS.GFIELDMAPPING is converted to a type 7 double for the inserted feature class.
OverrideGFieldAFM_trig.sql	Same as above but AFM compatible.
SetAuto2Double.sql	Check for feature classes whose primary key is double but is set to integer due to the presence of an autonumber sequence. The type 4 integer in GDOSYS.GFIELDMAPPING is converted to a type 7 double. This procedure should be run once in the USER that owns the feature classes in question.
SetAuto2DoubleAFM.sql	Same as above but AFM compatible.
SetNum10toDouble.sql	The type 4 integer in GDOSYS.GFIELDMAPPING is converted to a type 7 double. This procedure should be run once in the USER that owns the feature classes in question.

APPENDIX D: THE SDO_GEOMETRY OBJECT

Oracle Spatial defines the object type SDO_GEOMETRY as:

```
CREATE TYPE SDO_GEOMETRY AS OBJECT (
SDO_GTYPE NUMBER,
SDO_SRID NUMBER,
SDO_POINT SDO_POINT_TYPE,
SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

SDO_GTYPE

SDO_GTYPE indicates the type of the geometry. In the following table, the d is the number of dimensions: 2, 3, or 4. For example, a value of 2003 indicates a 2-dimensional polygon.

Value	Geometry Type	Description
d000	UNKNOWN_GEOMETRY	Spatial ignores this geometry.
d001	POINT	Geometry contains one point.
d002	LINESTRING	Geometry contains one line string.
d003	POLYGON	Geometry contains one polygon
d004	COLLECTION	Geometry is a heterogeneous collection
d005	MULTIPOINT	Geometry has multiple points.
d006	MULTILINESTRING	Geometry has multiple line strings.
d007	MULTIPOLYGON	Geometry has multiple, disjoint polygons

SDO_SRID

SDO_SRID can be used to identify a coordinate system (spatial reference system) to be associated with the geometry. If SDO_SRID is null, no coordinate system is associated with the geometry. If SDO_SRID is not null, it must contain a value from the SRID column of the MDSYS.CS_SRS table and this value must be inserted into the SRID column of the USER_SDO_GEOM_ METADATA view. All geometries in a geometry column must have the same SDO_SRID value. GeoMedia ignores this value; you can set it to NULL or use it as the needs arises.

SDO_POINT

SDO_POINT is defined using an object type with attributes X, Y, and Z, all of type NUMBER. If the SDO_ELEM_INFO and SDO_ORDINATES arrays are both NULL, and the SDO_POINT attribute is non-null, then the X and Y values are considered the coordinates for point geometry. Otherwise the SDO_POINT attribute is ignored by Spatial. GeoMedia uses oriented points so this should be set to NULL. GeoMedia can read this field if it exists.

SDO_ELEM_INFO_ARRAY

SDO_ELEM_INFO is defined using a varying length array of numbers set in groups of 3 (called triplets). The array interprets how the ordinates are stored in the SDO_ORDINATES array. Each triplet in the SDO_ELEM_INFO array is interpreted as follows (Offset, EType, Interpretation):

- OFFSET - Indicates the offset within the SDO_ORDINATES array where the first ordinate for this element is stored. Offset values start at 1.
- ETYPE - Indicates the type of the element. Valid values are
- 0 – User defined
- 1 – Point
- 2 – Line
- 1003 and 2003 – Outer and Inner Polygons
- 4 – Compound Line
- 1005 and 2005 – Outer and Inner Compound Polygons
- INTERPRETATION - Means one of two things, depending on whether or not ETYPE is a compound element:
- If the ETYPE is a compound element (4 or #005), this field specifies how many subsequent triplet values are part of the element.
- If the ETYPE is not a compound element (1, 2, or #003), the interpretation attribute determines how the sequence of ordinates for this element is interpreted. For example, a line string or polygon boundary may be made up of a sequence of connected straight-line segments or circular arcs.
- For polygon ring elements in a single geometry must use a 4-digit ETYPE value for all elements:
- 1003: exterior polygon ring (must be specified in counter clockwise order)
- 2003: interior polygon ring (must be specified in clockwise order)
- SDO_ETYPE values 4 and #005 are considered compound elements. They contain at least one header triplet with a series of triplet values that belong to the compound element.

If the geometry consists of more than one element, then the last ordinate for an element is always one less than the starting offset for the next element. The ordinates from its starting offset to the end of the ORDINATE array describe the last element in the geometry.

For compound elements (SDO_ETYPE values 4 and #005), a set of n triplets (one per sub-element) is used to describe the element. It is important to remember that sub-elements of a compound element are contiguous. The last point of a sub-element is the first point of the next sub-element. For sub-elements 1 through n-1, the end point of one sub-element is the starting point for the next.

The semantics of each ETYPE element and the relationship between the SDO_ELEM_INFO and SDO_ORDINATES varying length arrays for each of these ETYPE elements are given below:

ETYPE	INTERPRETATION	Oracle Meaning	GeoMedia
0	0	Unsupported element type.	RW
1	1	Point type.	RW
1	n > 1	Point cluster with n points.	RO
2	1	Line string (straight line segments)	RW
2	2	Line string made up of a connected sequence of circular arcs.	RW
#003	1	Simple polygon whose vertices are connected by straight line segments	RW
#003	2	Polygon made up of a connected sequence of circular arcs	RW
#003	3	Rectangle type. LL/UR	RO
#003	4	Circle type. Described by three points on circumference	RO
4	n > 1	Line string with some vertices connected by straight line segments and some by circular arcs. The value, n, specifies the number sub-elements.	RW
#005	n > 1	Compound polygon with some vertices connected by straight line segments and some by circular arcs.	RW

For ETYPE's 4 and #005, the next n triplets in the SDO_ELEM_INFO array describe each n sub-element. The sub-elements can only be of ETYPE 2. The end point of a sub-element is the start point of the next sub-element, and it must not be repeated. The start and end points of the polygon must be the same.

Each circular arc is described using three coordinates: the arc's start point, any point on the arc, and the arc's end point. The coordinates for a point designating the end of one arc and the start of the next arc are not repeated. For example, five coordinates are used to describe a polygon made up of two connected circular arcs. Points 1, 2, and 3 define the first arc, and points 3, 4, and 5 define the second arc. The coordinates for points 1 and 5 must be the same, and point 3 is not repeated.

SDO_ORDINATE_ARRAY

SDO_ORDINATES is defined using a varying length array (1048576) of NUMBER type that stores the coordinate values that make up the boundary of a spatial object. This array must always be used in conjunction with the SDO_ELEM_INFO varying length array. The values in the array are ordered by dimension. For example, a polygon whose boundary has four 2-dimensional points is stored as:

```
{X1, Y1, X2, Y2, X3, Y3, X4, Y4, X1, Y1}.
```

If the points are 3-dimensional, then they are stored as:

```
{X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4, X1, Y1, Z1}
```