A01.1

The Maia project: The development of a very large geospatial database

Brigitte Colombo, Senior Consultant, Oracle Corporation and Ed Parsons, Chief Technical Officer, Ordnance Survey of Great Britain

Abstract

The Maia database is a world first. It will be the largest spatially-enabled database storing large scale topographic data and attribute information for editing and digital product generation purposes. It forms the backbone of the Ordnance Survey's business programme and supports the eDelivery initiatives.

This paper highlights the technical direction of the Ordnance Survey and discusses the issues involved with implementing a very large spatial database. The Maia database provides a unique case study of very large spatial database design and implementation. Technical issues include building the database for performance, availability, manageability and scalability, and designing the database structures to support complex data requirements and the evolution of the database.

Introduction

The Ordnance Survey of Great Britain and Oracle Corporation UK have been working together to develop a master, editing database called Maia.

This paper covers the strategic direction behind this development, the technical issues in Very Large Spatial Database (VLSDB) design and implementation, and finally, the results from the Maia prototype.

These issues are highly relevant from a strategic perspective to National Mapping Agencies or similar institutions, as well as, from a technical perspective, those wishing to implement a VLSDB, such as a national MasterMap coverage.

What is Maia?

Maia is the database system that will store the seamless digital mapping base of Great Britain that underpins the current and future range of MasterMap products. Maia will replace the existing tile-based Digital Data Management System (DDMS). Initially it will store the internal format of MasterMap large-scale topographic mapping, expanding to include other MasterMap layers, for example, 1:50,000 Generalised Topographic and the Integrated Transportation Network layers. Thus Maia must be designed to be highly scalable to support the future expansion of data complexity, data volume and functionality.

To support these requirements, the Ordnance Survey has embarked upon an assessment and pilot study of the Oracle9*i* Spatial database system. This follows several prototype systems developed during 2001.

Two databases will underpin the Ordnance Survey digital mapping systems: the Maia editing database and the Mercury publishing database. The Mercury database is the current Geospatial Object Server (GOS), based on Excelon Object Store technology. Oracle9*i* Spatial has been selected as the platform for Maia. Once Maia is operational, the field object editors (FOE) will directly edit the data stored in Maia. Updates will be pushed out from Maia to Mercury at regular intervals and be available to subscribers within a short period of being accepted into Maia. In summary, Maia is a read-write database while Mercury is a read-only publishing system, and the data in Mercury may be fully derived from Maia.

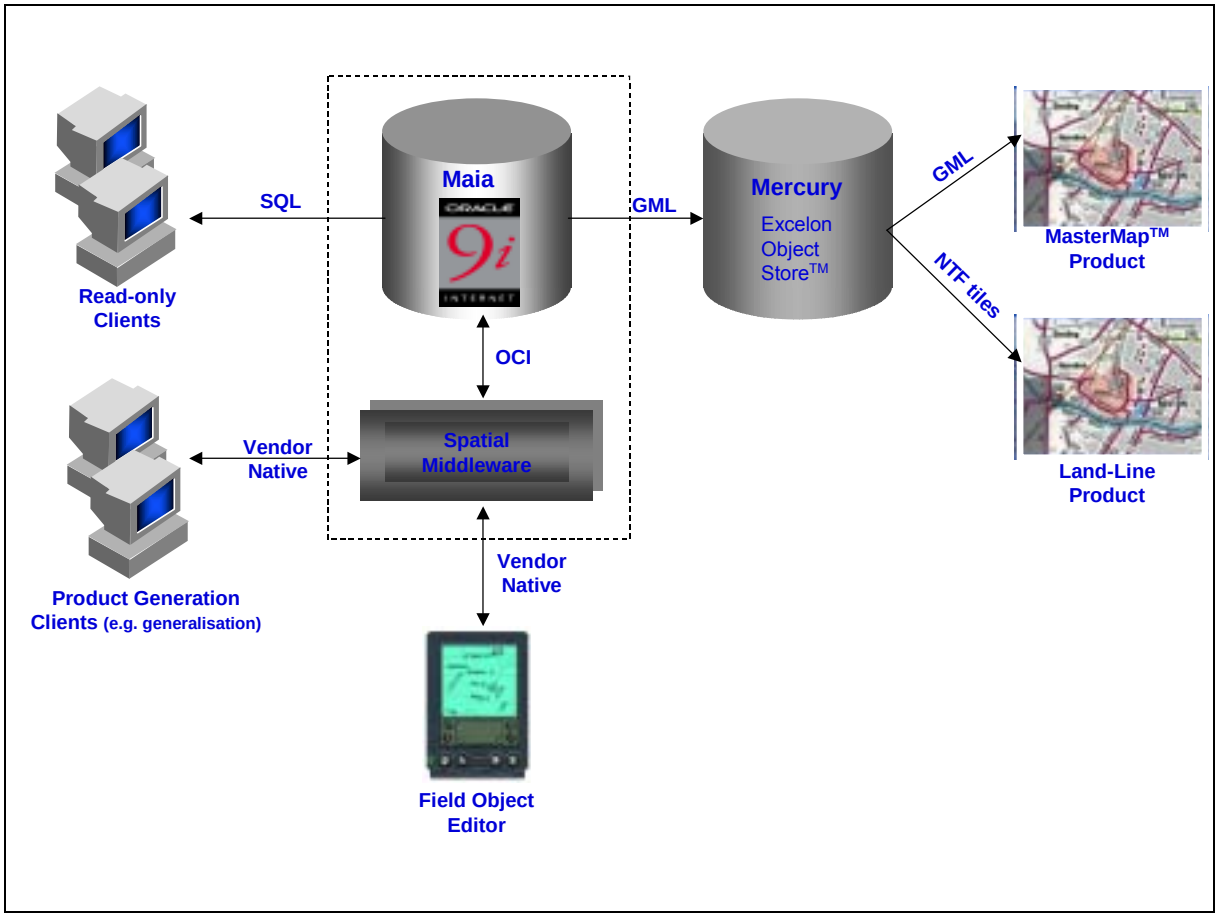

Figure 1. Ordnance Survey Target Architecture

Technical Strategy of the Ordnance Survey

Maia is a major part of the Ordnance Survey's move from a tiled to a seam-free digital mapping base for data maintenance, storage and supply. Other factors driving the Maia project are the requirement to modernise internal systems to support a new FOE, a technical strategy based upon e-delivery and a requirement to derive added value from existing digital data sources.

Key to realising these changes is a move away from a "Best of Breed" bespoke development environment to the exploitation of Standards based Commercial of the Shelf Software (SCOTS).
The use of a SCOTS based system considerably reduces the long-term cost of ownership of new systems and increases the speed and flexibility of new system development. For example, Oracle Spatial will allow the use of multiple vendor GIS applications as clients to the Maia database. As the defacto industry standard, Oracle Spatial is seen as advantageous for the development of the new FOE and for future initiatives that will act on the database such as automated generalisation. The Ordnance Survey has adopted Oracle as its enterprise database platform and has the technical infrastructure to implement this change.

Technical Issues concerning the implementation of a VLSDB

This section of the paper provides a brief introduction to Oracle Spatial and a discussion of design issues for very large spatial databases (VLSDB). This presents the concepts, which clarify the design and results of the Maia prototype.

Brief Overview of Oracle Spatial

Oracle Spatial is an integrated set of functions and procedures that use object-relational technology to manage, access and query spatial data within the Oracle database. This Spatial technology is part of the database kernel, as opposed to being an extension or plug-in.

In technical terms Oracle Spatial is:

- A data type called SDO_GEOMETRY, which enables point, line and polygon features as well as complex elements made up of a combination of these, to be stored just like any other number, character, or date data.
- A fast access method to the data via spatial indexes.
- Extensions to the query language SQL to be able to write and execute spatial queries such as, "Find me all the street lights in each Ward within the County of Hampshire."

```
SELECT /*+ ordered */ s.id, s.ulrn, w.ward, s.geom
FROM street_lights s, wards w
WHERE w.county='HAMPSHIRE'
AND SDO_RELATE (w.geoloc, s.geom, 'mask=ANYINTERACT
                querytype=window') = 'TRUE';
```

All of the Oracle Relational Database Management System (RDBMS) features are available with Spatial; these include replication, distributed processing, security and parallel processing. The same utilities are available, such as SQL*Loader, Oracle Enterprise Manager, backup and recovery, export and import. The same development languages such as SQL, PL/SQL, Java, XML, and OCI, can be used to insert, delete, update and select the spatial data. This means that you can have a single database for spatial and non-spatial data, and lower the cost of ownership by using mainstream I.T. resources to manage this data within Oracle Spatial.

Why VLSDB?

What is a VLSDB?

Databases of around 1 Terabyte (TB) or more are typically described as very large databases (VLDB) because special techniques are usually applied to maintain and manage these systems. A very large spatial database (VLSDB) is similar, but includes additional considerations for geospatial data management and processing. Because Oracle Spatial is part of the Oracle RDBMS, the considerations for designing a VLSDB include the same issues as non-spatial VLDB plus additional factors for the spatial element.

Initially, the Maia database will be in the order of 815 GB, but the addition of other data sets is expected to double or triple this size within the first year of production.

VLDB techniques include table partitioning, parallelism and data warehousing techniques for database design, population and indexing. VLSDB techniques include the above, plus Spatial Partitioning, Spatial Index Partitioning and Spatial Clustering.

Partitioning

Partitioning addresses the problem of supporting very large tables and indexes by decomposing tables and indexes into smaller pieces called partitions. SQL statements can access and manipulate the partitions rather than the entire table. The partitions should be roughly balanced in size to benefit the most from partition elimination. Significant performance gains can be achieved when using partition-based queries.

For example, a National coverage can be partitioned into Administrative Areas.

If users only ever want information for one or more administrative areas (that is, the county name is specified in the WHERE clause of the query) then only the partition that pertains to the specified administrative area is queried; the vast majority of the data (that in all other administrative areas) is not processed. This is termed "partition elimination."

Indexes may be "global" across all partitions in a table, or "local" with one index for each partition. Partitioning data and, optionally, partitioned local indexes provides additional performance gains via parallel execution when inserting / updating table data or creating / updating indexes. Partitioning data and indexes also provides some very important manageability options for moving data, splitting and exchanging partitions. Note that, partitioning may lead to a decrease in performance if the table is not partitioned based upon the criteria used in the typical query profile.

Parallel Execution

Parallel execution or parallel processing can dramatically reduce query response times for data-intensive operations on large databases. Instead of using a single process for one statement the work is spread among multiple processes. Parallel execution is useful for many types of operations, including:

- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes and large tables
- Bulk inserts, updates and deletes
- Aggregations

Symmetric multiprocessing (SMP), clustered systems, and massively parallel systems (MPP) gain the largest performance benefits from parallel execution because statement processing can be split among many CPUs. Parallel execution helps systems scale in performance by making optimal use of hardware resources and the performance improvement can be significant.

Note that parallel execution is not suited to all systems and can reduce system performance on over utilized systems or systems with insufficient I/O bandwidth.

Data Warehousing Techniques

Data Warehouses contain very large quantities of data, have few concurrent users, are primarily Read-Only and are used for Decision Support Systems and analytical processing. To improve performance when querying very large quantities of data, the schema is de-normalised, for example, by storing optional attributes or pre-computed summaries in the same table.

Data is typically loaded in large batch operations using SQL*Loader.

Oracle9*i* provides the External Tables feature which can provide faster bulk loading of text files than SQL*Loader. The files containing the data are linked to a structure (similar to a SQL*Loader control file), and can then be read in parallel and written in parallel.

Partitioning is employed for performance and management reasons.

Bitmapped indexes, Function-Based Indexes, Index Organised Tables and Materialised Views are powerful techniques used to improve query performance in a data warehouse environment. Function-based indexes provide a rapid method of using existing point data sets within an Oracle Spatial and a GIS environment.

The other methods should be considered with VLSDB for the non-spatial attribute data that frequently accompanies each spatial feature. Refer to the bibliography for references that detail these features.

What's so Special About Spatial Data?

Spatial data has some special characteristics that may impose additional constraints on the design of a database system. These constraints apply to the characteristics of the data, technical architecture, performance and tuning, and include:

- the space required to store a spatial feature may be much greater than typical number or character data, requiring careful selection of database block size
- Oracle stores spatial objects as LOBs which have special characteristics
- choosing a spatial index type

- choosing a partitioning key for spatial data
- clustering spatial data on disk based upon proximity in the real world

Database Block Size

There are a number of factors to balance when setting the block size for a database (or at tablespace level in Oracle9*i*). The goal is to minimise unnecessary I/O by ensuring that a record is wholly contained within a database block (and not "chained" to another block resulting in additional I/O) but that there is not wasted space in the block. If the block size is too large relative to average record size you may retrieve many additional records in which you are not interested (depending on query selectivity) consuming memory resources and you may suffer block contention as many processes try to access the same block. Note that the database block size must be a multiple of the operating system block size.

For vector data, line and polygon features may be complex comprising many hundreds or hundreds of thousands of coordinate pairs. The largest known feature in British digital data products is in MasterMap and is comprised of approximately 400,000 coordinate pairs. This is an extreme example, but given a data set of environmental management areas, each with an average of 500 coordinate pairs defining its boundary, the number of bytes within Oracle required to store this geometry is shown below. The example geometry is a polygon with two voids, the coordinate system is British National Grid and an average precision of 8 digits per ordinate is assumed [1].

SDO_GEOMETRY	Example Data	Estimation	Size (bytes)
SDO_GTYPE	2003	3 bytes plus 1 byte for every two numeric places	5 bytes
SDO_SRID	81989	1 byte if NULL; or 3 bytes plus 1 byte for every two numeric places	6 bytes
SDO_POINT	NULL	1 byte if NULL; or for each of the three numeric values, 1 byte if NULL, or 3 bytes plus 1 byte for every two numeric places	1 byte
SDO_ELEM_INFO	(1,1003,1), (1189,2003,1), (1871,2003,1)	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.	(4 + 5 + 4) + (5 + 5 + 4) + (5 + 5 + 4) + 40 = 81 bytes
SDO_ORDINATES	215130.15, 301000.05, 215135.16, 301000.05, etc	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.	(7 * 1000) + 40 = 7040 bytes
Total			**7133 bytes (7.0 KB)**

Table 1. Estimating the size of a polygon geometry in Oracle Spatial

For this example, there will be additional attribute data such as Management Area ID, Management Area Name, District Name, etc. Assuming these additional attributes require 200 bytes, the average record length for the table will be approximately 7.2 KB. Therefore a block size of 16 KB may be appropriate,

storing two geometries per block, with less than 2 KB of empty space (which may or may not be used if the data is updated).

Point data and simple lines or rectangles require much less space. In this example, the geometry is a point, the coordinate system is British National Grid and an average precision of 8 digits per ordinate is assumed [1].

SDO_GEOMETRY	Example Data	Estimation	Size (bytes)
SDO_GTYPE	2001	3 bytes plus 1 byte for every two numeric places	5 bytes
SDO_SRID	81989	1 byte if NULL; or 3 bytes plus 1 byte for every two numeric places	6 bytes
SDO_POINT	(215130.15, 301000.05, NULL)	1 byte if NULL; or for each of the three numeric values, 1 byte if NULL, or 3 bytes plus 1 byte for every two numeric places	7 + 7 + 1 = 15 bytes
SDO_ELEM_INFO	NULL	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.	1 byte
SDO_ORDINATES	NULL	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.	1 byte
Total			28 bytes

Table 2. Estimating the size of a point geometry in Oracle Spatial

Assuming this database is a decision support system, a block size of 8 KB would be the choice for storing this data set (4 KB block size is only suitable for some online transaction processing systems).

As can be seen, the space requirement of spatial data varies dramatically with the complexity of the data. Oracle9*i* supports the concept of tablespace block sizes such that data with similar characteristics can be grouped together in a tablespaces and an appropriate block size set for that tablespace. This flexibility provides many advantages but raises the requirement for a good understanding of the data and the way in which it will be used.

LOB Segments and Indexes

Data of type SDO_GEOMETRY is stored as a LOB (Large Binary Object). By default LOBs will be stored "in-line", that is, with the other table data. If the LOB column is greater than 3,964 bytes, it will be stored "out-of-line" in another segment of the same tablespace. This results in at least one additional read or write operation as the original block is read and then the block(s) containing the LOB segment. LOB segments are also created for spatial indexes.

By default in Oracle, LOB segments are not cached in the database buffer cache. This behaviour can be changed by specifying the STORE AS CACHE option of the CREATE TABLE or ALTER TABLE command. Caution should be exercised as read of a long LOB could effectively flush the cache. This problem may be reduced in Oracle9i with the use of multiple buffer caches and the ability to assign tables to specific buffer caches. On Unix systems, the operating system buffer cache can be used to store LOBs. This provides

another mechanism for tuning LOB performance if the database buffer cache is not used. Discuss the options with the system administrator.

Database Buffer Cache

Due to the size of spatial geometries, the database buffer cache size can have a significant impact on performance. As a general rule, this should be increased, with the optimum size determined by monitoring and tuning.

Spatial Indexes

Oracle provides two types of spatial indexes: a (Linear) Quadtree and an R-tree. Spatial index creation and updates against spatially indexed data are slow relative to standard index types. A spatial index re-build is different to rebuilding a B-tree index in that it is a total re-creation. The performance of the spatial index types are compared for query, insert and index creation operations in [2]. A summary of the relative features is provided in the table below:

R-tree	Quadtree
Use for geodetic data	Cannot be used for geodetic data
Use for local spatial indexes	Do not support local spatial indexes
Can index data in two, three or four dimensions	Can index two-dimensional data
Do not require tuning	Require tuning of the tessellation level
Requires additional space in index tablespace during creation	Fixed-tiling level recommended. Hybrid indexing rarely suitable
Requires allocating sort area space in memory, and undo / rollback space and temporary space on disk for creation	Generally larger storage space required; point data requires same storage as R-tree
For large polygon data sets index creation time and updates can be faster than Quadtree	For point data sets index creation time and updating the index is faster than R-tree
Update times are almost linear in the number of updates	Index creation time is fairly linear
On average, for large polygon data, R-tree queries are faster for all relate query masks	For larger query windows (>10 mile radius), Quadtree may provide equal or better performance than R-tree.
Faster for nearest neighbour queries because spatial proximity is preserved	For “touch” queries Quadtrees are faster because they better approximate the “boundary” area of the window
Faster for distance queries	
Support incremental nearest neighbour queries (where results for a nearest neighbour query keep getting returned until some non-spatial criteria is met)	Do not support incremental nearest neighbour queries
May become less efficient with frequent updates (can rebuild the index)	

Table 3. Comparison of R-tree and Quadtree spatial index features

For most applications R-tree indexes provide better performance, require no tuning, less disk space and are the only spatial index type supported for local spatial indexes, two- or three-dimensional data and geodetic data. A Quadtree index may be advantageous for update-intensive applications using simple polygon geometries, for high concurrency update databases, or where specialised query masks such as "touch" are frequently used [2].

Partitioning Spatial Data and Indexes

For spatial data and indexes, Oracle supports range partitioning. This uses a scalar partition key and the data is allocated to a partition based upon whether the partition key is between user-specified values. For location based billing applications for mobile telecommunications a good partition key may be billing regions as the cost of making a call is computed based upon the location of the caller relative to their "home" area. This also has the advantage of storing all data that is located in the same geographic area in the real world, in the same partition and relatively contiguous on disk. The partition key should be based upon an attribute that is fairly static as re-partitioning the data is resource intensive.

For linear features it is difficult to find a geographical partitioning key for which every feature will be entirely contained by the area corresponding to a particular value. Real world features can cross the boundaries of administrative areas. Therefore, partition elimination can be therefore less straightforward with spatial data.

For partitioned table data, the spatial index may be either global or local partitioned. Local partitioned spatial indexes are supported for R-Tree indexes only and can provide the following benefits:

- reduced response times for long-running queries
- reduced response times for concurrent queries as I/O operations may run concurrently on each partition
- reduced index creation times as builds can be run in parallel
- easier index maintenance, because of partition-level create and rebuild operations (reduces time and permits operations in parallel)
- indexes on partitions can be rebuilt without affecting the queries on other partitions enhancing data availability
- storage parameters for each local index can be changed independent of other partitions

Partitioned local spatial indexes are a feature of Oracle9*i* Spatial. Exchanging partitions (with local spatial indexes) is supported with 9*i* release 2 (9.2.0).

Spatial Clustering Data on Disk

Spatial clustering is the organisation of data on physical storage media such that features that are close to each other in the real world are stored close to each other on disk. This requires the data to be sorted into an order that preserves spatial proximity using a space-filling curve such as a z-ordered algorithm. Studies show that for large data sets data retrieval performance can be significantly improved when implementing spatial clustering over randomly clustered data [3]. The performance benefit achieved is dependent upon the table or partition size and the selectivity of the query. Spatial index creation performance should also be improved as the data is already sorted.

Once each spatial feature has been assigned a spatial cluster key, the data can be clustered on disk using a query with the ORDER BY clause or loading pre-sorted data. Note that a published limitation in Oracle prevents records with a column greater than 30,000 bytes being sorted. Thus not all data, but certainly the majority, may be sorted in this way. However, as new data is inserted in to the table or the geometry is updated, the spatial clustering will degrade. The performance benefits of spatially clustering data on disk must be balanced against the increased processing and maintenance cost.

The Maia Pilot

Hardware

Model	Sun Enterprise V880
CPU	4 x SPARC-II 750 MHz
Memory	8 GB RAM
Disks	2 x T3 disk arrays, RAID 5, • 400 GB storage space
LAN Protocol	TCP/IP

Software

Operating System	Sun Solaris 2.8
Oracle Products	Oracle9*i* Enterprise Edition (9.0.1.3.0) Spatial, with Partitioning Option

Database Configuration

Locally managed tablespaces were used for all the data and index tablespaces. Automatic Undo management was configured for the database. To enable the parallel query options the following initialisation parameters were set:

```
parallel_automatic_tuning=TRUE

parallel_max_servers=16

parallel_min_servers=4
```

The Oracle SGA was configured as follows: -

Total System Global Area	806127636 bytes
Fixed Size	280596 bytes
Variable Size	167772160 bytes
Database Buffers	637534208 bytes
Redo Buffers	540672 bytes

Of the available 4 GB of memory, Oracle used just under 1 GB. The SGA could be sized much bigger and improve the overall query response times, however, database tuning was not in the scope of the Pilot.

Format of the Maia Pilot Study

The rationale for the Pilot study was to load and spatially index as much data as possible (the constraint being disk space) then test query performance and functionality important for the project.

This was not a formal benchmark. Many different processes may have been running against the server at the same time as tests were being conducted.

Maia Pilot Results

Maia Logical Design

Database design should be driven by business needs and encompass both the functional and non-functional requirements. The key factors that drove the design the Maia database were the large number of complex records that the first route into the data would be via a spatial query and that optimisation of data structures for spatial analysis was not required. That it to say, the question asked would be "Find me everything in the area", and **not** "Find me all the yellow bike sheds in London".

A schema was designed based upon three large central tables, which contain all the point, line and polygon features respectively and all their attributes. This design was chosen for performance reasons to reduce the number of table joins that would be required to query the data. A single table for storing all features was considered but discounted as some out-of-the box GIS require points, lines and polygons to be in separate tables. In the next phase of the project, the three feature table model will be refined to ten tables to reflect topological structuring such that each of the topological layers is placed in a separate table e.g. a road network line is in a different table to a topographic line or an administrative boundary line. This is required to implement a topology model within the database.

Optional feature attributes were modelled as columns in the main feature tables. In Oracle, trailing columns in a table will take up a maximum of 1 byte storage regardless of the number of columns, providing they are all NULL. To minimise record length the columns were ordered such that the least frequently populated columns were located at the end of the table definition.

There is also a business requirement to be able to query the database as it was at a point in time in the past: requiring on-line storage of historical data. Given that the editing applications or product generation processes will operate from the current version of the data, it was decided to keep the historic data in a separate tables to reduce the amount of data being searched. Data views across the current and historic data will enable the reconstruction of data snapshots at a point in time in the past. This design favours queries of the current version because historical queries are expected to be infrequent.

Each feature is stamped with a from_date and a to_date. The to_date is null for the current version. When features are updated database triggers will copy the current version of the feature to the corresponding history table and to time stamp the to_date attribute. The record in the main feature table will then be updated and the system time entered in the from_date attribute. The history to_date and the main feature table from_date will be set to the same time for all features in a given transaction.

Maia Physical Design

As each of the feature tables will be very large (lines 276,640,000 (66.5 %), areas 96,512,000 (23.2 %) and points 42,848,000 (10.3 %)) the main concern for the physical data model, was finding a suitable partition key. As the primary access method to the data would be geographic it was decided to implement a geographic-based partitioning regime. A reference grid similar to that used to partition the data in Mercury was conceptually laid over Britain and the centroid of each geographic feature was used to assign it to a grid cell. By using the centroid a feature could only ever appear once. The grid cell id became the partitioning key.

One hundred and sixty-two partitions were defined based upon feature density. Thus the partition grid cells are of smaller geographical extent in urban areas and become much larger in rural areas where feature density is low. Each partition was placed in its own tablespace within the database.

As explained above the performance benefit of partitioning is obtained when the partition key is passed as a parameter to the query. To achieve this using the current spatial partitioning model, an API layer could be introduced to interpret a query window into one or more of the grid cells. As a feature may straddle a partition boundary, the "partition map" used by the API layer to identify relevant partitions would also have to record and maintain the aggregate minimum bounding rectangles (MBR) of all features assigned to the partition. The query would then be re-written to include the grid cell id(s) and benefit from partition elimination. This would be transparent to the application. Development of such an API was out-of-scope for the project, however, this partitioning method was chosen because of the maintenance benefits and the

fast elimination of partitions available via the use of local partitioned spatial indexes (the root node of each index partition can be scanned very rapidly to determine those partitions that need to be examined).

Data Loading

Fifty-four percent of complete national cover or 250,000,000 features comprising sixteen 100 km British National Grid squares: NB, NC, NG, NH, NM, NN, NR, NS, NW, NX, SJ, SK, SO, SP, SU, TQ; were loaded into the Maia database. These areas were chosen as representative of all types of data in MasterMap, specifically the very large, complex features along the coast of Scotland, and the very dense urban areas in the south of England.

Ordnance Survey wrote routines to extract data from the Mercury database in a format suitable for loading into Oracle using SQL*Loader. Each spatial feature was augmented with a cluster key, which was used to both partition and to sort the data on disk. A primary key on the TOID was used to reject duplicate data. The duplicate features result from the 5 km "chunks" used in the extraction query, causing features that cross the chunk boundary to be repeated.

Load rates were highly variable depending predominately upon the density of the features. Load rates varied from 1,000 – 3,000 features per minute. The estimated minimum time to complete the national load is 85 hours (3 days, 13 hours)

Number of features	Time to Load
250,000,000	42 hours

Data loading was parallelised in four load processes to make full use of the 4 CPUs on the server. Alphabetically listed data files were distributed cyclically between the 4 load processes, resulting in a degree of randomisation. However, the load processes remained unbalanced, with some processes completing more than 15 hours after the first. Performance could be improved by using External Tables and by balancing the load streams based upon the number of records in each input file. Tables were set to NOLOGGING during the load to minimise the amount of redo generate and improve performance. Note that Direct Path loads are not supported for spatial data.

Sizing

The raw data (without indexes) required the following disk space:

Number of features	Size on disk
227,859,756 *	128 GB

** 249,697,673 features were loaded, but National Building Data Set polygons, cartographic pylons and archways were deleted resulting in 227,859,756 features.*

With an estimated total of 416,000,000 features, extrapolating from 54 % of the data, the Maia database is estimated to require 250 GB of raw data storage within the database, extending to around 815 GB for all data, indexes, database structures (redo logs, archived redo logs, undo, temporary and system tablespaces), allowance for maintaining history on-line and contingency.

Type of Data	Size (GB)
Raw data	250
Spatial indexes	80
Transient tables required during Spatial index creation / re-build (≈ 50% of final index size)	40
Non-Spatial indexes	50
History (10% change per year for 5 years)	125
History Spatial indexes	40
Transient tables required during Spatial index creation / re-build	20
History Non-Spatial indexes	25
Database structures (system, undo, redo, temp)	20
Sub-total	*650*
Contingency at 25%	165
Total	815 GB *

** This estimate does not take into account space for data re-organisation, a staging area for data undergoing quality assurance or for data sets other than the internal large-scale topographic MasterMap data set.*

Spatial Clustering Data on Disk

The grid-based partition key described above was refined to also serve as a spatial clustering key on which the data could be sorted. A Z-ordered curve was used to refine the grid cells generating 10-character cluster keys, appended with the 100 km grid reference e.g. Sdccadbacbdc, which provided a clustering resolution of just under 100 m.

The data was first loaded into the table partitions in random order and then sorted, partition by partition. Features greater than 30,000 bytes (0.0002% of lines and 0.02% of areas) were inserted into the partition following the sort.

Spatial Indexing

The following elapsed times where obtained for **global** spatial index creation: -

Table	No. Records	Elapsed Time	Spatial Index Final Size	Spatial Index Max. Build Size	Final Size as % of Build Size
Areafeature	52,946,832	24:13:00	4,480	8,382	53.45
Linefeature	151,464,847	59:14:25	12,727	16,849	75.54
Pointfeature	23,448,077	6:57:48	1,992	6,466	30.81

Global versus Local Spatial Indexes

Global spatial indexes were used due to a Bug with local spatial index creation in Oracle 9.0.1.3. Local partitioned spatial indexes may be chosen as more performant for the Maia database and are currently being tested using Oracle9*i* release 2.

Parallel Spatial Index Creation

The index creation process in 9.0.1.3 is a serial operation and cannot be parallelised. Parallelism was enabled by starting the spatial indexes for each of pointfeature, linefeature and areafeature simultaneously. Thus with 3 CPUs (for 3 spatial tables) the minimum time required to spatially index the data is the length of time of the longest operation, this being for the lines. If local partitioned spatial indexes are used, indexing of partitions in parallel is possible, enabling all 4 CPUs to be utilised. Oracle 9*i* release 2 provides parallel spatial index creation; the performance improvement has been shown to be up to 50%.

Elapsed Time versus CPU Time

CPU intensive operations that take place concurrent to spatial index creation will significantly slow down the creation of spatial indexes. For example, at the same time as the spatial index on the linefeature table was creating, foreign key constraints were being created and schema objects analysed. By comparing elapsed time with CPU time, we found that these additional operations caused the elapsed time to be more than 8 hours longer than the CPU time.

Spatial Index Tablespace Size

More space is required during spatial index creation than the size of the final spatial index because a large number of transitory tables are created during the creation; these are then dropped on completion. The figures above show that the amount of additional space required during spatial index creation vary for point, line and polygon features.

Enable Foreign Key Constraints

It is also useful to present the results obtained when completing typical database operations such as enabling foreign key constraints as the time taken to complete these tasks must also be considered in a large data load exercise.

Type of Operation	Database Size	Elapsed Time
Enable all foreign key constraints (63 indexes)	250,000,000 records 3 ancillary tables 9 lookup tables	Actual = 19 hours
Enable all foreign key constraints (63 indexes)	Full national data set 3 ancillary tables 9 lookup tables	Estimate = 35 hours

The tables, indexes (spatial and non-spatial) and spatial index tables should be analysed. A measurement is not available for this operation but when estimating statistics with a sample size of 5 per cent, is expected to take no more than 10 hours for the full National data set.

Other Tests

MapInfo Professional 6.5 was used to visualise the data. A database view was used to format the data e.g. with a MI_PRINX column, suitable for MapInfo. This provided a visual check of the data.

Other tests are being progressed by the Ordnance Survey to gain experience in the operations that will typically be performed against the Maia database, such as generating a change-only update query and implementing the automatic population of the history table using database triggers.

It is expected that the system will go into production on Oracle9*i* release 2 to take advantage of features such as automatic segment space management, parallel spatial index creation, local partitioned spatial indexes and improved R-tree index update performance. Further tests are being conducted within the Ordnance Survey to measure the benefits of these new features.

The Ordnance Survey is closely following the development of a Topology Manager proposed for release with Oracle 10*i* Spatial. The Ordnance Survey has already demonstrated that it can populate the draft topology model within the Maia database.

Conclusion

This paper has described how the Maia master editing database project exemplifies the Ordnance Survey's technical strategy, the specific issues that drive the design of a very large spatial database in the Oracle environment and the results obtain from the Maia prototype.

The Ordnance Survey continues to test and explore the possibilities offered by the Oracle9*i* Spatial database (release 2) to support its advanced storage, editing and product generation requirements.

The future development of this system will push the boundaries of current spatial database technology and realise substantial business benefits for the Ordnance Survey. To ensure the successful adoption of Oracle Spatial for current and future projects, Oracle Corporation is collaborating with Ordnance Survey and its technology partners. Ordnance Survey, in turn, is feeding enhancement requests and new requirements to Oracle for incorporation in future releases of the Spatial product. This close relationship is sure to benefit the geospatial community in the UK, and internationally.

Acknowledgements

Many thanks to Mark Richardson of Ordnance Survey for his contribution to this paper.

References

[1] "Oracle Spatial 8.1.6 Performance-Related Characteristics," Oracle Technical White Paper, April 2000 (54 p).

[2] "Quadtree and R-Tree Indexes in Oracle Spatial: A Comparison using GIS Data", Kothuri R, Ravada S, Abugov D, ACM SIGMOD '2002 June 4-6 Madison, Wisconsin USA (12 p.).

[3] "Spatial DBMS testing with data from the Cadastre and TNO-NITG," Tijssen T, Quak C and van Oosterom P, GISt Report No. 7, Delft, March 2001.

Bibliography

[4] "Optimal Storage Configuration Made Easy", Juan Loaiza, Oracle Open World, 2000 (13 p).

[5] "Oracle9i Materialized Views: an Oracle White Paper", Oracle Corporation, May 2001 (23 p).

[6] "Oracle 9i Database Concepts Release 1 (9.0.1)" Oracle Corporation, Part No. A88856-02

[7] "Oracle Spatial User's Guide and Reference Release 9.2" Oracle Corporation, Part No. A96630-01