R-Tree Index and Bitmap Index Performance Comparisons of Three Query Categories on Geographic Data in Re-constructed Year 2000 TIGER Database - New York State Portion

Kan Zhou

Sihua Jin

Dept of Computer Science

New York University

December 9th, 2002

ABSTRACT

In a relation schema with significant number of records, explore the performance difference between the R-tree spatial index and bitmap index on geographic data sets. Three categories of SQL queries are used in order to obtain comparison results: Multipoint query, Extremal query and Range query. For each query category, we obtained the time required to obtain ten continuous query results from the spatial database. Comparisons are presented in the form of bar charts.

INTRODUCTION

Oracle Spatial

Oracle Spatial is an integrated set of functions and procedures that enable spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle9i database. Spatial data represents the essential location characteristics of real or conceptual objects as those objects relate to the real or conceptual space in which they exist. Oracle Spatial provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features in an Oracle9i database. Spatial consists of the following components:

A schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric data types;
A spatial indexing mechanism;
A set of operators and functions for performing area-of-interest queries, spatial join queries, and other spatial analysis operations;
Administrative utilities.

The spatial component of a spatial feature is the geometric representation of its shape in some coordinate space. This is referred to as its *geometry*. A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. In our experiment, a TIGER database that we created from US Census Bureau TIGER/Line files for the State of New York is used as such an example of road map.

TIGER Database / TIGER-Line Files

The TIGER/Line files are extracts from the United States Census Bureau TIGER (Topologically Integrated Geographic Encoding and Referencing) database of selected geographic information. The TIGER/Line files contain information about the spatial objects distributed over a series of record types. The Census TIGER database uses points, lines, and areas to provide a disciplined, mathematical description of the features of the earth's surface. Spatial objects in the Census TIGER database are interrelated. A sequence of points defines line segments, and line segments connect to define polygons. The TIGER/Line file documentation uses the terminology from the Spatial Data Transfer Standard (SDTS). The spatial objects in TIGER/Line belong to the "Geometry and Topology" (GT) class of objects in SDTS. In the construction of our database, we pay attention to classification of three important objects:

Node	A zero-dimensional object that is a topological junction of two or more links or chains, or an end point of a link or chain;
Complete Chain	A chain that explicitly references left and right polygons and start and end nodes. The shape points combine with the nodes to form the segments that make a complete chain;
GT-Polygon	An area that is an atomic two-dimensional component of a two-dimensional manifold, one and only one planar graph and its two-dimensional objects.

The UA Census 2000 TIGER/Line files consist of 17 record types that collectively contain geographic information. In our construction of database, we only referred to Record Type 1 file (RT1 briefly), RT2 and RTI files. Record Type 1 provides a single record for each unique complete chain in the TIGER/Line files. The basic data record contains the end nodes for the complete chain. Record Type 2 provides an additional series of latitude and longitude coordinate values describing the shape of each complete chain in Record Type 1 that is not a straight-line segment. That is, not all complete chains in Record Type 1 have shape points and therefore not all have an associated Record Type 2. Where a complete chain in Record Type 1 is not a straight line, Record Type 2 may have a many-to-one relationship with Record Type 1. Record Type I has a one-to-one relationship with Record Type 1. When Record Type I is linked to a single-sided Record Type 1 (county boundary), it will provide only the left- or the right-polygon identifier. Defining a complete chain requires information from Record Types 1, 2, and I.

Record Linkage in Construction of Database

Plotting a complete chain requires using the nodes from Record Type 1 and all of the shape point records in Record Type 2 with the same TLID, if any. Plot the start node first, and then search Record Type 2 for any matching records. If there is a match, the record will contain from 1 to 10 shape points. If all 10-point fields are filled with non-zero values, there may be an additional matching Type 2 record. Type 2 records are not sorted by TLID, but all records with the same TLID should appear together in sequence by the record sequence number (RTSQ). Plot the shape points from all Type 2 records and end the complete chain by plotting the end node. There is a one-to-one relationship between the GT-polygons constructed from Record Types 1 and 2 and those appearing in the Record Type I.

We slightly modified the construction process of 2-D GT polygons given all shape points' coordinates in a complete chain. A complete chain has a unique TLID to

reference in Record Type I files. Each TLID could correspond to as many as two POLYIDs, which are referred as POLYIDL and POLYIDR. We created therefore two polygons for each complete chain compared to creating one or two after joining all *boundary complete chain segments* as what other Mapinfo software did. So in this sense, our GT polygons constructed are not atomic; they could be divided into smaller polygons. However, as our main purpose of this project is to explore the R-tree index performance issues, not to re-creating a complete and accurate road map for NY State, this method is reasonable, and it did greatly simplify our logic to create a polygon. Although they are not atomic, they are actually at different spatial index levels compared to same level for all atomic polygons. We need not find all complete chain segments forming a polygon's boundary. In fact, the Mapinfo software processing on creation of polygons is very tough to implement using simple table joins based on our current database environment and Record Type files.

Indexes and Queries

We created one big table containing one field of GT Polygons, and created R-Tree index on these polygons. In order to compare performance of bitmap index, we need to put detail coordinate info of shape points in one complete chain specific to this polygon. Since most of complete chains have no more than 10 shape points, we choose to put 10 intermediate shape points plus the starting and ending points. Therefore, there are 24 longitude/latitude fields. The bitmap index is created against these 24 fields.

Three types of SQL queries have been tested: Multipoint query, Extremal query and Range query. The rationale for the selection of these types of queries is based on our big table's compositions. It is obvious that Prefix Match Query does not fit in this scenario, because all related fields are either GT objects or numeric fields; also, to get an "average" on GT objects is ambiguous and difficult to implement, so Grouping query is not applicable here.

EXPERIMENTAL PROCEDURE

According to above discussions, our project was divided into the following *seven* stages:

1	Record Type file downloading and analysis of field positions;
2	Coding to extract fields based on RT file definitions specified in TIGER/Line File Document;
3	Creation of related RT* tables and importing of all related RT* files into database;
4	Coding to create data file which is to be imported into our primary table using Oracle's Bulk Loader;
5	Bulk Load data file into Oracle, and create R-tree and bitmap index on related field(s);
6	Coding to issue queries for each index, each query category. Get their starting execution time and completion time so as to get time interval.
7	Plot bar charts to reflect performance difference for two indexes and different query types.

Hardware / OS / DBMS Specifications

Most of our experiments are performed on a Hewlett-Packard notebook PC (except in stage 4 described above, in which we used a secondary Microsoft SQL Server concurrently, which is installed in another desktop PC to assure faster data file generation. This will be described in more detail later):

Intel Pentium III-Mobile 1.20GHz CPU featuring 32K L1 cache and 512K L2 cache, 512 MB PC133 SDRAM, 20.0 GB Ultra DMA Mode 5 HD (100MB/sec maximum transfer rate, 4200 RPM, 20ms average seek time, average rotational latency 7.1ms. 2MB cache).

OS: Windows XP Professional.

DBMS: Oracle 9i, Microsoft SQL Server 2000 pack 2. Buffer sizes are dynamically increasing.

Programming language settings: PERL 5.6.0 on Windows, with DBI module. ODBC is connection method for both Oracle and MS SQL Server.

Stage 1-3: Record Type file preparation and importing into DB RT* tables

Record Type ASCII files are downloaded from US Census web site: http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html for 2000 version. Each of NY State's 62 counties has corresponding 17 RT* files, in the format of tgr36XXX.RT*. XXX is their serial number and is odd only, and RT* ::= RT[(1-9)|(ACHIPRSZ)]. According to TIGER document, each line in any RT* file will generate one record in its corresponding table, and the table's field properties are defined. Therefore, 17 tables are created in MS SQL Server for each RT* file, with names as RT[(1-9)|(ACHIPRSZ)]. We used PERL to generate CSV file for each RT* file, and used SQL Server's bcp utility to bulk insert CSV into it.

Total size of RT files downloaded*: 827 MB for 62 counties.
Size of .MDF data file in SQL Server: 1.23 GB, with log file size of 400MB.
Total number of rows in all 17 tables: 7,576,247.
Biggest table: RT1, with 1,333,949 rows.

Stage 4-5: Primary Table Generation and Loading into Oracle 9i

Will create table "POLYALL_V3' featuring all 2-D polygons in NY state, including all counties. The table is defined as follows:

```
CREATE TABLE POLYALL_V3 (PolyID     int NOT NULL,
      Shape       MDSYS.SDO_GEOMETRY,
      FRLONG      NUMERIC(10) NOT NULL,
      FRLAT       NUMERIC(9) NOT NULL,
```

```
LONG1       NUMERIC(10) NOT NULL,
LAT1        NUMERIC(9) NOT NULL,
… …
LONG10      NUMERIC(10) NULL,
LAT10       NUMERIC(9) NULL,
TOLONG      NUMERIC(10) NOT NULL,
TOLAT       NUMERIC(9) NOT NULL );
```

We then wrote Perl script to get table join results from SQL Server's RT1, RT2 and RTI, and exported to a data file to be imported into Oracle primary table. Since RT1 and RT2's sizes are large, to ensure faster execution, we used same tables from another SQL Server installed on a desktop featuring Athlon XP 2000+ CPU, 384 MB memory and 30 GB HD. The 2 PCs are in the same network so connection and data exchange rates are fast enough.

The logic for table join is: Each complete chain defined by RT1 and RT2 will determine 2 polygons, in table RTI, which has PolyID "POLYIDL" and "POLYIDR". According to "Right" and "Left" definitions in TIGER/Line File Specification document, all "Right" polygons' complete chains are in counterclockwise order in RT2, and all "Left" polygons' complete chains are in clockwise order in RT2, so when they are created in POLYALL_V3 table, MDSYS.SDO_ELEM_INFO_ARRAY has SDO_ETYPE 1003 for all counterclockwise chains, and 2003 for clockwise chains. After all polygons have been created in POLYALL_V3 table, an R-tree index will be created regarding 'shape'. A bitmap index then was created on FRLAT / FRLONG / Long1..10 / LAT1…10 / TOLAT / TOLONG.

```
CREATE INDEX POLYALL_V3_shape_idx
ON POLYALL_V3(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

Create bitmap index POLYALL _V3_bitmapindex on POLYALL_V3
(FRLONG, FRLAT, LONG1, LAT1,
      … …
LONG10, LAT10, TOLONG, TOLAT);
```

The data file will be loaded into PolyAll_V3 using Oracle's Bulk Loader utility sqlldr.

The control file for Bulk Loader is:

```
LOAD DATA
INFILE PolyAll_V3_OracleSqlldr.csv
APPEND INTO TABLE PolyAll_V3
FIELDS TERMINATED BY    ','
TRAILING    NULLCOLS
(PolyID     INTEGER EXTERNAL,
shape COLUMN OBJECT(sdo_gtype INTEGER EXTERNAL, sdo_srid INTEGER EXTERNAL,
SDO_POINT COLUMN OBJECT
  ( X INTEGER EXTERNAL,
    Y INTEGER EXTERNAL,
    Z INTEGER EXTERNAL),
SDO_ELEM_INFO     VARRAY terminated by    ';'
(SDO_ORDINATES char(64)),
SDO_ORDINATES     VARRAY terminated by    ':'
(SDO_ORDINATES char(4095))    ),
FRLONG      INTEGER  EXTERNAL,
FRLAT       INTEGER  EXTERNAL,
LONG1       INTEGER  EXTERNAL,
LAT1        INTEGER  EXTERNAL,
… …
LONG10      INTEGER  EXTERNAL,
LAT10       INTEGER  EXTERNAL,
TOLONG      INTEGER  EXTERNAL,
TOLAT       INTEGER  EXTERNAL )
```

Each row in data file will be one row in the table. One row example is as follows:

```
70,2003,,,,,1,2003,1;-73734705,42686036,-73734391,42686564, -
73734232,42686861,-73734009,42687327,-73733700,42688021,-73733733,42688098,-
73734006,42688448,-73734134,42688623,-73734374,42688785,-73734550,42688855,-
73734931,42688807,-73735144,42688823,-73734705,42686036:-73734705,42686036,-
73734391,42686564,-73734232,42686861,-73734009,42687327,-73733700,42688021,-
73733733,42688098,-73734006,42688448,-73734134,42688623,-73734374,42688785,-
73734550,42688855,-73734931,42688807,-73735144,42688823,
```

Data file size: 343 MB.

PolyAll_V3 table size after data file has been loaded: 1,188,262 rows.

Stage 6-7: Query to DB and obtain time difference

We wrote Perl scripts to issue 6 types of queries:

	Sample Oracle R-tree Index	Sample Oracle bitmap index
Extremal Query	SELECT MAX(SDO_GEOM.SDO_MAX_MBR_ORDINATE(c .Shape, m.diminfo, 1)) FROM POLYALL_V3 c, user_sdo_geom_metadata m WHERE m.table_name = 'POLYALL_V3' AND m.column_name = 'SHAPE'; -- Maximum X ordinate value (max among all TOLONG, Long1..10 and FRLONG)	SELECT MAX(TOLONG), MAX(LONG1), MAX(LONG2), MAX(LONG3), MAX(LONG4), MAX(LONG5), MAX(LONG6), MAX(LONG7), MAX(LONG8), MAX(LONG9), MAX(LONG10), MAX(FRLONG) FROM POLYALL_V3; -- assume at this point, it takes no time to find out the maximum Longitude value among these maximums.
MultiPoint Query	SELECT COUNT(*) FROM POLYALL_V3 WHERE SDO_RELATE(SHAPE, MDSYS.SDO_GEOMETRY(2001, NULL, MDSYS.SDO_POINT_TYPE(42635158, 7327429, NULL), NULL, NULL), 'mask=**touch** querytype=WINDOW') = 'TRUE';	SELECT COUNT(TOLAT) FROM POLYALL_V3 WHERE TOLAT = 42635158 and TOLONG = 7327429;
Range Query	SELECT COUNT(*) FROM POLYALL_V3 WHERE SDO_RELATE(SHAPE, MDSYS.SDO_GEOMETRY (2003, NULL, NULL, MDSYS.SDO_ELEM_INFO_ARRAY (1,1003,3), MDSYS.SDO_ORDINATE_ARRAY (-71000000, 40000000, -74000000, 44000000)), 'MASK=INSIDE QUERYTYPE=WINDOW') = 'TRUE'; -- found all Geometry objects that fall in a rectangle	SELECT COUNT(TOLAT)FROM POLYALL_V3 WHERE TOLONG > - 74000000 and TOLONG < - 71000000 AND TOLAT > 40000000 AND TOLAT < 44000000;

For each query type, 10 queries are issued continuously, while in the beginning of each run, a CleanBuffer() will be called to clean the cache, so each query is based on “cold buffer”. In MultiPoint query, each run will generate a random longitude/latitude pair from -74000000 ~ -71000000 / 40000000~44000000 range. For each query, the starting and ending system time are recorded and are logged into a specific log file.

RESULTS

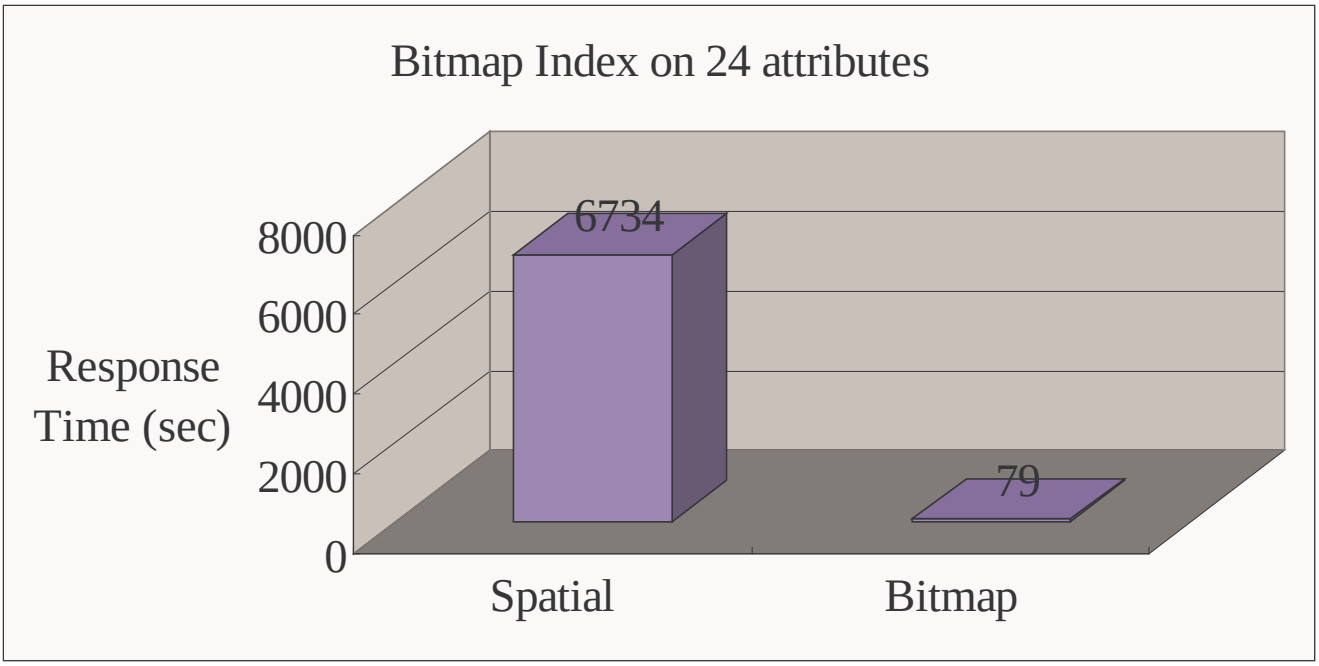

Figure 1. Performance Comparison For 10 consecutive Extremal Queries

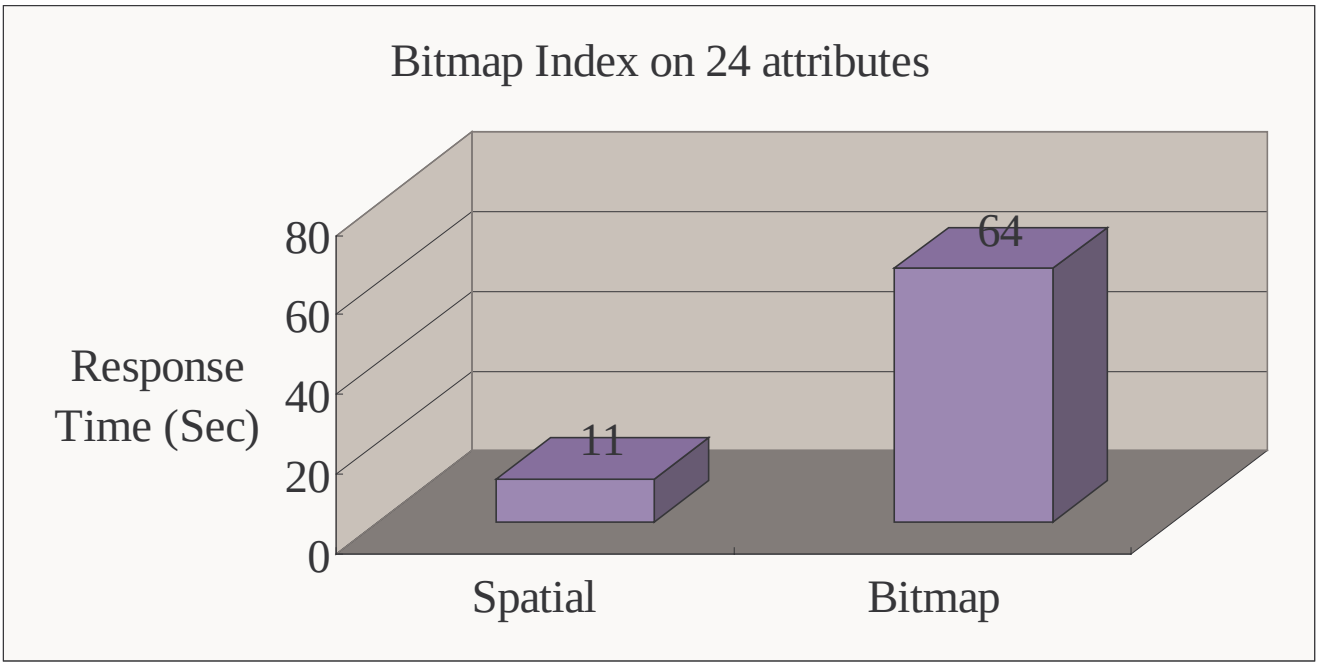

Figure 2. Performance Comparison For 10 consecutive MultiPoint Queries

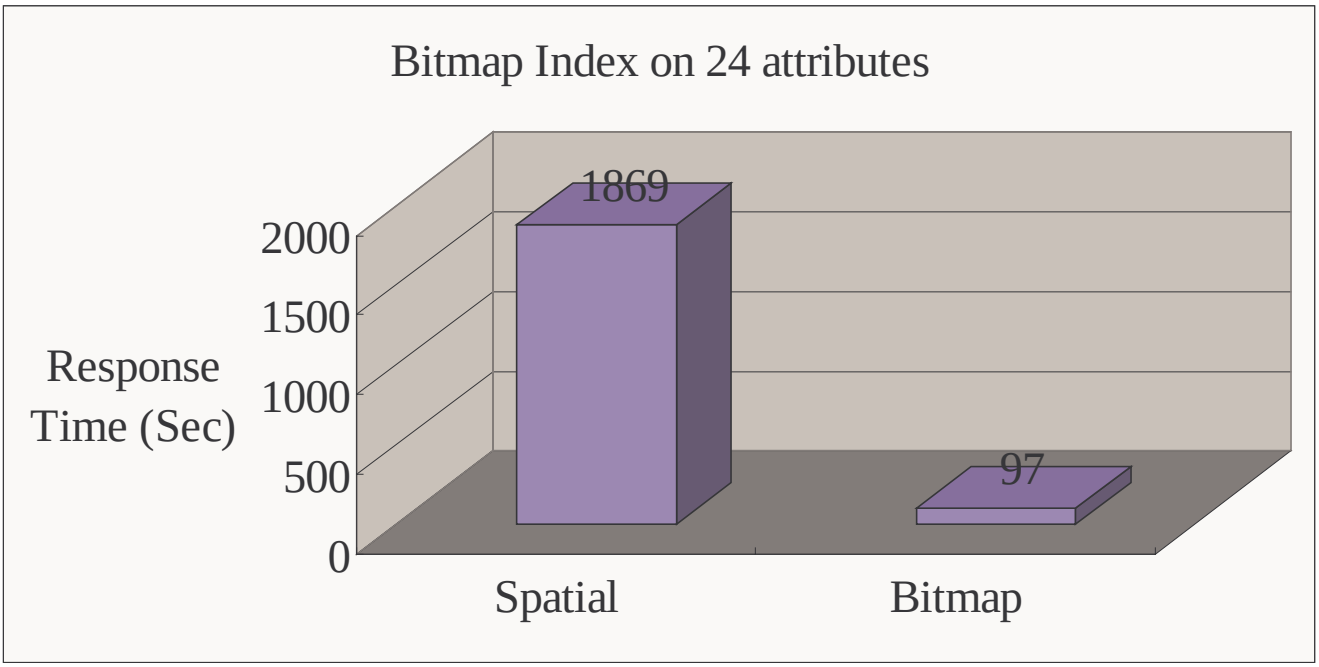

Figure 3. Performance Comparison For 10 consecutive Range Queries

CONCLUSION AND DISCUSSION

R-Tree index beats Bitmap index only in MultiPoint Query, but bitmap index beat R-tree in both Extremal query and Range query with a wide margin.

It is interesting to see that in MultiPoint query, R-tree beats bitmap index. Looking back to Philippe's tuning examples on Point query and Range query, we found in the Point query, the mask function value used in RELATE function between geometry objects and candidate point-type objects is 'equal', whereas in our MultiPoint query, we used mask value as 'touch'. Our reason is to find our any 2-D objects that in their forming complete chain, there is such a shape point that is identical in coordinates with those of the target point. This is based on our specific construction method, because the way we construct polygons ensures that there would be no point object as a GT-polygon entity in any record. All polygons are of real shape. Semantically, our implicit order of MultiPoint searching for R-tree means: "find all rows that contains a polygon which has a point on its forming complete chain that matches target point's coordinates." This is obviously many-to-one relationship, so it is a multipoint query.

We also even tested the situation that we switched mask value from 'touch' to 'equal', and it turns out to be the response time for R-tree is a little longer in this case, but still not longer than bitmap index. Their response times are almost identical.

In the Extremal and Range queries, the reason that R-tree lost to Bitmap index could be: Since all these queries requires R-tree index on longitude/latitude attributes alternate its searching criterion among different attributes as different levels of the tree, and since bitmap index requires no more levels of attributes than longitude/latitude, the overhead in constructing R-tree compared to bitmap index could be a defining factor to explain its performance lag.

REFERENCES

1. Oracle 9i Spatial User's Guide and Reference.
2. Other Oracle 9i Books (Release 9.2).
3. TIGER/Line File Documents, by US Census Bureau, U.S. Department of Commerce, Geography Division, 2000.

ATTACHMENTS

1. Log file of single-run query version, with all system time;
2. Log file of 10-run query version, with all system time;
3. PERL script to issue queries and do the system time management (10-run version).

ATTACHMENT 1

```
<<<<<< Started at:  Sun Dec  8 02:02:54 EST 2002

perl TestQueries2.pl

===> Starting EXTREMAL QUERY With Spatial Index at: Sun Dec  8 02:02:54 EST 2002


Extremal query (on Spatial index) result:     -71777491
<=== Finished EXTREMAL QUERY With Spatial Index at: Sun Dec  8 02:14:26 EST 2002


===> Starting EXTREMAL QUERY With Bitmap Index at:  Sun Dec  8 02:14:27 EST 2002


Extremal query (on Bitmap Index) result:      Max (-71778137 -71777491 -71777491 -
71782748 -71794139 -71784927 -71837088 -71856694 -71856982 -71857174 -71857270 -
71778137)
<=== Finished EXTREMAL QUERY With Bitmap Index at:  Sun Dec  8 02:14:34 EST 2002


===> Starting MULTIPOINT QUERY With Spatial Index at:Sun Dec  8 02:14:34 EST 2002


MULTIPOINT QUERY (on Spatial Index) result:   4
<=== Finished MULTIPOINT QUERY With Spatial Index at:Sun Dec  8 02:14:37 EST 2002


===> Starting MULTIPOINT QUERY With Bitmap Index at:Sun Dec  8 02:14:37 EST 2002


MULTIPOINT QUERY (on Bitmap Index) result:    4
<=== Finished MULTIPOINT QUERY With Bitmap Index at:Sun Dec  8 02:14:43 EST 2002


===> Starting RANGE QUERY With Spatial Index at:    Sun Dec  8 02:14:44 EST 2002


RANGE query (on Spatial Index) result: 316874
<=== Finished RANGE QUERY With Spatial Index at:    Sun Dec  8 02:27:43 EST 2002


===> Starting RANGE QUERY With Bitmap Index at:     Sun Dec  8 02:27:44 EST 2002


RANGE query (on Bitmap Index) result:  317454
<=== Finished RANGE QUERY With Bitmap Index at:     Sun Dec  8 02:27:52 EST 2002
```

ATTACHMENT 2

```
<<<<<< Started at:  Sun Dec  8 02:51:18 EST 2002

perl TestQueries2_10Sets.pl

===> Starting 10-runs of EXTREMAL QUERY With Spatial Index at:   Sun Dec  8 02:51:18
EST 2002

Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
Extremal query (on Spatial index) result:     -71777491
<=== Finished 10-runs of EXTREMAL QUERY With Spatial Index at:   Sun Dec  8 04:43:32
EST 2002

===> Starting 10-runs of EXTREMAL QUERY With Bitmap Index at:    Sun Dec  8 04:43:32
EST 2002

Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
Extremal query (on Bitmap Index) result:      Max  (-71778137  -71777491  -71777491  -
71782748  -71794139  -71784927  -71837088  -71856694  -71856982  -71857174  -71857270  -
71778137)
<=== Finished 10-runs of EXTREMAL QUERY With Bitmap Index at:    Sun Dec  8 04:44:51
EST 2002

===> Starting 10-runs of MULTIPOINT QUERY With Spatial Index at: Sun Dec  8 04:44:51
EST 2002

MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
```

```
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
MULTIPOINT QUERY (on Spatial Index) result:   0
<=== Finished 10-runs of MULTIPOINT QUERY With Spatial Index at: Sun Dec  8 04:45:02
EST 2002

===> Starting 10-runs of MULTIPOINT QUERY With Bitmap Index at:  Sun Dec  8 04:45:02
EST 2002

MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
MULTIPOINT QUERY (on Bitmap Index) result:    0
<=== Finished 10-runs of MULTIPOINT QUERY With Bitmap Index at:  Sun Dec  8 04:46:06
EST 2002

===> Starting 10-runs of RANGE QUERY With Spatial Index at:Sun  Dec   8  04:46:06  EST
2002

RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
RANGE query (on Spatial Index) result: 316874
<=== Finished 10-runs of RANGE QUERY With Spatial Index at:Sun  Dec   8  06:57:15  EST
2002

===> Starting 10-runs of RANGE QUERY With Bitmap Index at: Sun  Dec   8  06:57:15  EST
2002

RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
RANGE query (on Bitmap Index) result:  317454
<=== Finished 10-runs of RANGE QUERY With Bitmap Index at: Sun  Dec   8  06:58:52  EST
2002
```

ATTACHMENT 3

```
#!c:/perl/bin/perl.exe

## This file function is to issue 10 runs for each query category,
## and get their response time.

################################################################################
#                           MAIN BODY
#
################################################################################

use strict;
use DBI;
use Getopt::Std;

$SIG{'INT'} = 'CleanUp';
$SIG{'TERM'} = 'CleanUp';

my $db; #universal DB handler
my %Options;
getopts(":d", \%Options);

my $DBDSN= "OrclDB1";            #DSN for AlphaDB1
my $DBuser = "sa";
my $DBpasswd = "bcmdba1";
my $dbSa = DBI->connect("DBI:ODBC:$DBDSN",$DBuser, $DBpasswd, {
       RaiseError=>1,
       AutoCommit=>1
       }
) or die "Couldn't connect to database: " . DBI->errstr;
#connect to databse and now $db is the DB handle

$DBDSN= "OrclDBSysdba";          #DSN for AlphaDB1
$DBuser = "system";
$DBpasswd = "sandy";
my $dbSys = DBI->connect("DBI:ODBC:$DBDSN",$DBuser, $DBpasswd, {
       RaiseError=>1,
       AutoCommit=>1
       }
) or die "Couldn't connect to database: " . DBI->errstr;
#connect to database and now $dbSys is the DB handle

$db = $dbSa;
my $sql = qq{TRUNCATE TABLE TMaxX; };
DBSimpleAction($sql);
### preparing to insert various maximum X-coordinates into TMaxX, empty it first

my ($cmd) = $0;
$cmd =~ s|^.*/||;   #Program name
my $cmdLine = "perl $cmd " . join(' ', @ARGV);
my $logFile;
my $logDir = 'C:/CSCourses/AdvDB_fa02/FinalProject/Log';
my $yyyymmdd = ConvertSysDate(`date`);
mkdir $logDir;
$logFile = "$logDir/TestQueries_PointOnly_$yyyymmdd.log";
open(LOGFILE, ">>$logFile") or die "Can't open file $logFile\n"; #log file
select(LOGFILE); $| = 1; select(STDOUT);
print LOGFILE "<<<<<< Started at:\t" . `date` . "\n";
print LOGFILE "$cmdLine\n\n";    #records the whole command line

ExtremalQueryS();   #Extremal query on Spatial index
```

```
ExtremalQueryB();   #Extremal query on Bitmap index

MultiPointQueryS(); #MULTIPOINT QUERY on Spatial index

MultiPointQueryB(); #MULTIPOINT QUERY on Bitmap index

RangeQueryS();      #Range query on Spatial index

RangeQueryB();      #Range query on Bitmap index


$dbSys->disconnect;
$dbSa->disconnect;
close LOGFILE;
exit(0);


#############################################################################
#                            SUBROUTINES
#############################################################################


sub ExtremalQueryS() {
      ##### 1. Now Start the Extremal query to get maximum X-Coordinate in PolyAll_V3
table:
      print LOGFILE "===> Starting 10-runs of EXTREMAL QUERY With Spatial Index
at:\t" . `date` . "\n\n";

      for(my $i = 0; $i < 10; $i++) {
            CleanBuffer();

            my $sql = qq{
                  SELECT MAX( SDO_GEOM.SDO_MAX_MBR_ORDINATE(c.Shape, m.diminfo, 1) )
                  FROM POLYALL_V3 c, user_sdo_geom_metadata m
                  WHERE m.table_name = 'POLYALL_V3' AND m.column_name = 'SHAPE';
            };
            my $maxX = DBSingleSearch($sql);
            #print LOGFILE "Extremal query (on Spatial index) result:\t$maxX\n";
      }

      print LOGFILE "<=== Finished 10-runs of EXTREMAL QUERY With Spatial Index
at:\t" . `date` . "\n\n";
}


sub ExtremalQueryB() {
      print LOGFILE "===> Starting 10-runs of EXTREMAL QUERY With Bitmap Index
at:\t" . `date` . "\n\n";

      for(my $i = 0; $i < 10; $i++) {
            CleanBuffer();

            $sql = qq{
                 SELECT MAX(TOLONG), MAX(LONG1), MAX(LONG2), MAX(LONG3), MAX(LONG4),
MAX(LONG5),
                  MAX(LONG6),  MAX(LONG7),  MAX(LONG8),  MAX(LONG9),  MAX(LONG10),
MAX(FRLONG)
                  FROM POLYALL_V3;
            };
            my @max = DBRowSearch($sql);
            #print LOGFILE "Extremal query (on Bitmap Index) result:\t$maxX\n";
      }
```

```
	print LOGFILE "<=== Finished 10-runs of EXTREMAL QUERY With Bitmap Index at:\t" . `date` . "\n\n";
}


sub MultiPointQueryS() {
	##### 2. Now start the MULTIPOINT QUERY to get counts of certain point
	print LOGFILE "===> Starting 10-runs of MULTIPOINT QUERY With Spatial Index at:\t" . `date` . "\n\n";

	for(my $i = 0; $i < 10; $i++) {
		CleanBuffer();

		my $long = int(rand(3000000)) - 74000000; #between -71000000 and -74000000
		my $lat = int(rand(4000000)) + 40000000; #between 40000000 and 44000000

		$sql = qq{
			SELECT COUNT(*) FROM POLYALL_V3
			WHERE SDO_RELATE(SHAPE, MDSYS.SDO_GEOMETRY(2001, NULL,
			MDSYS.SDO_POINT_TYPE($long, $lat, NULL), NULL, NULL),
			'mask=touch querytype=WINDOW') = 'TRUE';
		};
		my $count = DBSingleSearch($sql);
		print LOGFILE "MULTIPOINT QUERY (on Spatial Index) result:\t$count\n";
	}

	print LOGFILE "<=== Finished 10-runs of MULTIPOINT QUERY With Spatial Index at:\t" . `date` . "\n\n"
}


sub MultiPointQueryB() {
	print LOGFILE "===> Starting 10-runs of MULTIPOINT QUERY With Bitmap Index at:\t" . `date` . "\n\n";

	for(my $i = 0; $i < 10; $i++) {
		CleanBuffer();

		my $long = int(rand(3000000)) - 74000000; #between -71000000 and -74000000
		my $lat = int(rand(4000000)) + 40000000; #between 40000000 and 44000000

		$sql = qq{
			SELECT COUNT(TOLAT) FROM POLYALL_V3
			WHERE TOLONG = $long and TOLAT = $lat;
		};
		my $count = DBSingleSearch($sql);
		print LOGFILE "MULTIPOINT QUERY (on Bitmap Index) result:\t$count\n";
	}

	print LOGFILE "<=== Finished 10-runs of MULTIPOINT QUERY With Bitmap Index at:\t" . `date` . "\n\n";

}


sub RangeQueryS() {
	##### 3. Now start the Range query to get counts of all the points that fall in certain rectangle
	print LOGFILE "===> Starting 10-runs of RANGE QUERY With Spatial Index at:\t" . `date` . "\n\n";
```

```
	for(my $i = 0; $i < 10; $i++) {
		CleanBuffer();

		$sql = qq{
			SELECT COUNT(*) FROM POLYALL_V3
			WHERE SDO_RELATE(Shape, MDSYS.SDO_GEOMETRY (2003, NULL, NULL,
			MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),
			MDSYS.SDO_ORDINATE_ARRAY(-71000000,	40000000,	-74000000,
44000000)),
			'mask=inside querytype=WINDOW') = 'TRUE';
		};
		my $count = DBSingleSearch($sql);

		print LOGFILE "RANGE query (on Spatial Index) result:\t$count\n";
	}
	print LOGFILE "<=== Finished 10-runs of RANGE QUERY With Spatial Index at:\t" .
`date` . "\n\n";
}


sub RangeQueryB {
	print LOGFILE "===> Starting 10-runs of RANGE QUERY With Bitmap Index at:\t" .
`date` . "\n\n";

	for(my $i = 0; $i < 10; $i++) {
		CleanBuffer();

		$sql = qq{
			SELECT COUNT(TOLAT)FROM POLYALL_V3
			WHERE TOLONG > -74000000 and TOLONG < -71000000
			AND TOLAT > 40000000 AND TOLAT < 44000000;
		};
		my $count = DBSingleSearch($sql);

		print LOGFILE "RANGE query (on Bitmap Index) result:\t$count\n";
	}
	print LOGFILE "<=== Finished 10-runs of RANGE QUERY With Bitmap Index at:\t" .
`date` . "\n\n";
}




################################################################################
# Function: DBSimpleAction
#
# Description:
#	execute sql scripts generated inside previous functions
#	that are just simple action, i.e. no return values.
#	and there won't be duplicate rows in the DB, so no overwrites
#
# Return values:
################################################################################

sub DBSimpleAction {
	#the argument passed here is a SQL-script
	my ($str) = @_;

	print $str if defined $Options{'d'};;

	# Prepare the statement for immediate execution
	my $rv = $db->do( $str );
	if ( !defined $rv ) {
```

```
	print ERRFILE "DB simpleAction statement execution failed: $DBI::errstr\n";
	exit(1);
	}

}	#end DBSimpleAction


################################################################################
# Function: DBSingleSearch
#
# Description:
#	execute sql scripts generated inside previous functions
#	that are mostly searching queries which should return certain values.
#
# Return values:	ONE single value, depending on searching criteria
#	for example: search for AssetID given AssetSymbol
################################################################################

sub DBSingleSearch {
	my ($sql) = @_;

	my $sth = $db->prepare($sql) or die "Couldn't prepare statement: " . $db->errstr;

	$sth->execute or die "Couldn't execute statement: " . $db->errstr;

	my ($val) = $sth->fetchrow();
	#single result is expected so there should be just 1 row

	if ($sth->rows == 0) {	return -9999; }
	# -9999 as an arbitrary integer, meaning searching gives no match

	$sth->finish or die "statement finish error.\n";

	return $val;
}	#end DBSingleSearch


################################################################################
# Function: DBRowSearch
#
# Description:
#	execute sql scripts generated inside previous functions
#	that are mostly searching queries which should return certain values.
#
# Return values:	ONE ROW as an array to be returned, depending on searching criteria
#	for example: search * FROM Ttable WHERE ...
################################################################################

sub DBRowSearch {
	my ($sql) = @_;

	my $sth = $db->prepare($sql) or die "Couldn't prepare statement: " . $db->errstr;

	$sth->execute or die "Couldn't execute statement: " . $db->errstr;

	my (@val) = $sth->fetchrow();
	#single result is expected so there should be just 1 row

	if ($sth->rows == 0) {	return (-9999); }
	# -9999 as an arbitrary integer, meaning searching gives no match
```

```
	$sth->finish or die "statement finish error.\n";

	return @val;
}	#end DBRowSearch



################################################################################
# Function: CleanBuffer
#
# Description:
#	execute sql scripts to clean buffer using sysdba account
#
# Return values:
################################################################################

sub CleanBuffer {

	$db = $dbSys;
	$sql = qq{
		ALTER SYSTEM FLUSH SHARED_POOL;
	};
	DBSimpleAction($sql);
	$db = $dbSa;

} #end CleanBuffer


################################################################################
# Function: ConvertSysDate
#
# Description: based on the date string created by DOS command "date",
#	like 'Wed Jul 31 08:39:40 EDT 2002';
#	returns the datetime-format string for SQL.
#	function called: GetMonthEndDate();
#
# Return values:    yyyymmdd
################################################################################

sub ConvertSysDate {
	my ($dateStr) = @_;
	my (@dates) = split(/\s+/, $dateStr);  #6 units

	my ($str) = $dates[1] . '/' . $dates[2] . '/' . $dates[5];

	$str = GetMonthEndDate($str);
	#pass string like 'Jul-31-2002' to GetMonthEndDate();
	#get the returned string like '7/31/2002'.

	@dates = split('/', $str);
	$dates[0] = '0' . $dates[0] if int($dates[0]) < 10; #08 instead of 8
	$dates[1] = '0' . $dates[1] if int($dates[1]) < 10; #08 instead of 8
	$str = $dates[2] . $dates[0] . $dates[1];     #yyyymmdd format
	return $str;

}


################################################################################
# Function: GetMonthEndDate
#
# Description:
```

```
#      based on the date string in the stat book file usually 1st column,
#      e.g. 'Mar-95', 'Mar-1995', 'October 1, 1998', '5/31/95', 'July-31-2002'...etc
#      gives the SQL format of datetime input
#      if no day specified, then return the last calendar day in that month
#
#      Function called: LeapYear();
#
# Return values:    string like '07/31/2002'
################################################################################

sub GetMonthEndDate
{
      my ($str) = @_;
      my ($year, $month, $day);
      my (@dates) = split(/[\-,\/ +]/, $str);#delimiters are - , and space
      my (%Months) = (
                              'Jan' => 1,
                              'January' => 1,
                              'Feb' => 2,
                              'February' => 2,
                              'Mar' => 3,
                              'March' => 3,
                              'Apr' => 4,
                              'April' => 4,
                              'May' => 5,
                              'Jun' => 6,
                              'June' => 6,
                              'Jul' => 7,
                              'July' => 7,
                              'Aug' => 8,
                              'August' => 8,
                              'Sep' => 9,
                              'September' => 9,
                              'Oct' => 10,
                              'October' => 10,
                              'Nov' => 11,
                              'November' => 11,
                              'Dec' => 12,
                              'December' => 12
            );
      #hash array for different Month inputs

      my (%daysPerMonth) = (
                              1 => 31,
                              2 => 28,
                              3 => 31,
                              4 => 30,
                              5 => 31,
                              6 => 30,
                              7 => 31,
                              8 => 31,
                              9 => 30,
                              10 => 31,
                              11 => 30,
                              12 => 31
            );
      #days in a month

      if ( $dates[$#dates] < 100 ) {
            #$dates[$#dates] is the last element in the @dates, which is the year
            #means the year is 2-digit number, so convert to 4-digits
            $year = 2000 + $dates[$#dates] if $dates[$#dates] <= 29;
            #2029 is the threshold year,
```

```
              #any year shown as 00-29 will be interpreted as 2000-2029.
              $year = 1900 + $dates[$#dates] if $dates[$#dates] > 29;
       } else {     #already 4-digit number
              $year = $dates[$#dates];
       }

       my ($leap) = LeapYear($year);   #call LeapYear function to see if it's leap
year

       if ($dates[0] =~ /^\d+$/) {
              $month = $dates[0];
       } else {
              $month = $Months{$dates[0]} ;    #$Months{$dates[0]}   should   get   the
month number
       }

       $day = $daysPerMonth{int($month)};     #get the last day in the month
       $day = 29 if $leap == 1 && int($month) == 2;

       if ((scalar @dates) == 2) {
              $str = "$month/$day/$year";
       } else  { #number of days in the month is presented
              $str = "$month" . '/' . $dates[1] . '/' . $year; #$dates[1] is day
              $str = "$month" . '/' . $day . '/' . $year if $dates[1] eq '';
              #if date is like 'June, 2001' must add day as $dates[1] is null,
              #but scalar @dates is 3

       }

       return $str; #now it should has the format like '7/31/1996'
}      #end GetMonthEndDate


################################################################################
# Function: LeapYear
#
# Description:
#      Tell if a 4-digit 1900+ year value is a leap year or not
#
# Return values:    0 if not a leap year, 1 if yes
################################################################################

sub LeapYear {
       my ($yr) = @_;

       return 0 if $yr < 1900 || $yr > 2099;
       #at this point, only deal with between 1900-2099

       return 0 if $yr == 1900;

       if  ($yr - int($yr / 4) * 4 == 0 ) {   #cannot mod 4
              return 1;
       } else {
              return 0;
       }
}#end LeapYear


################################################################################
#
#  Function:  Debug
#
#  Description: Prints statement if Debug flag was passed into program
```

```
#
################################################################################

sub Debug
{
    my ($msg) = @_;
    print "$msg\n";
}


################################################################################
# Function: CleanUp
#
# Description: close DB connection on signal interrupt
#
# Return values:
################################################################################

sub CleanUp {
    $db->disconnect if defined $db;
    close LOGFILE;
    exit(2);
}
```