CSIRO PUBLISHING

www.publish.csiro.au/journals/pasa *Publications of the Astronomical Society of Australia*, 2007, **24**, 13–20

Large Catalogue Query Performance in Relational Databases

Robert A. Power[A]

[A] CSIRO ICT Centre, Canberra ACT 2601, Australia. Email: robert.power@csiro.au

Received 2006 November 24, accepted 2007 February 19

Abstract: The performance of the MYSQL and ORACLE database systems have been compared for a selection of astronomy queries using large catalogues of up to a billion objects. The queries tested are those expected from the astronomy community: general database queries, cone searches, neighbour finding and cross matching. The catalogue preparation, SQL query formulation and database performance is presented. Most of the general queries perform adequately when appropriate indexes are present in the database. Each system performs well for cone search queries when the Hierarchical Triangular Mesh spatial index is used. Neighbour finding and cross matching are not well supported in a database environment when compared to software specifically developed to solve these problems.

Keywords: astronomical data bases: miscellaneous — catalogs — surveys

1 Introduction

An astronomy catalogue lists a collection of objects and their features, for example location, magnitude and colour. All sky catalogues have been constructed from systematic surveys of the night sky where one or more telescopes are dedicated to perform repeated observations. During this process, the same patch of sky will be observed multiple times and a catalogue constructed by merging data.

This paper reports an investigation comparing the performance of two popular database systems, MYSQL and ORACLE, using SQL queries on catalogues consisting of approximately a billion objects. This is the size expected of the catalogue generated from the Stromlo Southern Sky Survey (S4) using the SkyMapper telescope currently under construction[1]. Catalogues of this size currently exist and benchmarking tests have been carried out on a selection of these as well.

The investigation shows that large catalogues can be handled effectively with a modest computing environment. It is advisable, however, to be aware of specific deployment and implementation options available with each of the database systems adopted.

The paper starts with a review of related work before describing the computing environment and catalogues used. The first tests are a selection of typical queries expected by the astronomy community. This is followed by cone search queries, which highlight the need for a spatial index to improve query performance. The task of finding neighbours within a catalogue is then detailed along with the related operation of catalogue cross matching. A summary and discussion of the results concludes the paper.

An extended version of this investigation is available as a CSIRO ICT Centre Technical Report (Power 2006).

[1] www.mso.anu.edu.au/skymapper/

2 Related Work

The performance of general catalogue queries is covered in Gray et al. (2002) where they detail the implementation of twenty queries for the Sloan Digital Sky Survey (SDSS) (Szalay et al. 2000) using the Microsoft SQL SERVER database. These tests were however conducted on a dataset much smaller, 14 million rows. Their results cannot be directly compared with those reported here since every aspect is different: hardware, software and data.

The Hierarchical Triangular Mesh (HTM) spatial index (Kunszt et al. 2000) can be used to efficiently perform cone search and other location based queries. While there are other spatial indexing schemes for astronomy datasets,[2] detailed performance comparisons have not been reported.

Modern database systems provide support for data types other than numbers and strings. The type system of a database may be extensible allowing new user defined types and operations to be introduced. All the major database vendors support spatial data types in some form or other with standardisation efforts evolving[3]. The user base for these activities is from the Geographical Information Systems community which uses Earth based datasets. The coordinate systems are spheroidal rather than spherical, making it inefficient to measure distances when the data points use a spherical polar coordinate system. There are no known attempts to utilise these capabilities with astronomy datasets.

Neighbour finding is addressed in the field of computational geometry where optimal solutions use techniques that assume a complex data structure, for example the Voronoi tessellation or the Delaunay triangulation (Preparata 1985). These structures are computationally expensive to establish and the algorithms would need to be

[2] www.star.le.ac.uk/∼cgp/ag/skyindex.html
[3] www.opengeospatial.org/

10.1071/AS06026 1323-3580/07/01013

modified to work in a spherical polar coordinate system. The adaptation of these algorithms and structures for use in the astronomy domain remains untested.

The work of Gray et al. (2004) describes an approach to finding neighbours in SQL that uses declination partitions termed *zones*. Each object is uniquely associated with a zone: neighbours are first determined within a zone and, only when necessary, neighbours between zones evaluated. This approach reduces the search space for an individual object to a more manageable size, but within zones it uses a *nested loop* approach where each object is compared with all others to find neighbours.

OPENSKYQUERY[4] allows astronomy catalogues to be cross matched. For example, a personal catalogue of objects can be used for cross matching against a selected database. The algorithm used is the foundation for a distributed query engine allowing catalogues at different locations to be merged using a likelihood analysis to determine if objects match or not. Cross matching performance using OPENSKYQUERY for large catalogues is not documented.

3 Environment

A brief description of the computing environment and the catalogues used is presented below.

3.1 Computing

The machine used for benchmarking was a Dell Power Edge 2650 server with dual Pentium Xeon 2 GHz CPUs, 2 GB main memory and 6 TB disk. The maximum transfer rate between the disk and the server is 160 Mbytes/sec. The operating system is Debian Linux 2.6.8.

The database systems used are MYSQL 5.0 Community Edition, Generally Available (GA) Release[5] and ORACLE 10.2.0[6]. The databases were installed 'out of the box'. The table and column names were kept identical so the same SQL query syntax could be used interchangeably for each database. All tests are measured in terms of elapsed time, the total time taken to complete the test as reported by the computer's internal clock.

The benchmarking tests were performed using the same Java application for both databases. All tests are performed on a 'cold' database, one that has just been shut down and restarted and the operating system's disk cache flushed. The neighbour finding and cross matching code is implemented in C++.

3.2 Catalogues

The original purpose of benchmarking was to profile the expected database performance using a large synthetic catalogue constructed to simulate the characteristics expected from SkyMapper. Code to generate a realistic stellar S4 dataset was provided (P. Francis 2004, private communication) for this purpose. Large non-synthetic astronomy catalogues are also available and a small selection were also tested. These extra datasets provide a comparison of the synthetic dataset with realistic ones. They also allow the catalogue cross matching operation to be investigated.

Table 1. Catalogue summary

Catalogue	Size (GB)	# recs (10^6)
S4	91	819
USNO B1	78	1045
2MASS	147	471
SSA	261	1071

A brief description of each catalogue follows with a summary in Table 1.

3.2.1 S4

The synthetic S4 dataset simulates a stellar population that is realistic in its magnitude and sky distribution for the southern hemisphere. The randomly generated magnitudes adopt an extension of the *UGRIZ* system used in the SDSS and is described at the SkyMapper homepage[1].

3.2.2 USNO B1

The US Naval Observatory (USNO) has published a series of catalogues all based on digitising photographic plates. The B1.0 (Monet et al. 2003) is the latest, referred to as USNO B1 in this paper, and contains over one billion objects compiled from 7435 Schmidt plates, describing their position, proper motion and magnitudes in various optical passbands.

Each record in the catalogue is derived from up to five surveys and each survey includes a star/galaxy estimator flag. The original focus of this report was to investigate stellar datasets, so it was considered to extract the stellar records. Using the star/galaxy estimator, if only one flag is set to indicate the record is a star, then there are around 615 million stars, approximately 60% of the dataset. If all flags are required to agree that a record is stellar, then roughly 21 million (2%) are found.

The benchmarking reported here concerns database performance, not science astronomy results, and so the whole dataset was loaded and issues of identifying stellar records not considered further.

3.2.3 2MASS

The Two Micron All Sky Survey (2MASS) (Skrutskie et al. 2006) contains point and extended source catalogues of the near infra-red sky. The point source catalogue contains positions and uniformly calibrated photometry of nearly 500 million point sources while the extended source catalogue consists of over 1.5 million spatially extended sources, mainly galaxies.

Only the point source catalogue has been investigated here.

[4] openskyquery.net/Sky/skysite/
[5] www.mysql.com
[6] www.oracle.com

Table 2. Query performance (mm:ss)

Query	# recs	MYSQL	ORACLE
1. *R* mag	733 846 916	19:43	4:25
2. *R* mag	108 446	0:01	0:03
3. *UGRIZ* mag	32 954	9:01	0:25
4. 1 colour cut	19 635 062	0:32	0:09
5. 4 colour cut	47	0:37	0:51
6. White dwarf	19 411 077	3081:10	16:25
7. Quasar	0	9:59	10:04
8. Rare colours	205	116:32	41:53
9. Rare colours	91	98:41	35:46
10. Cone colour	822	3:13	1:19
11. HTM group	262 145	108:07	86:25
12. Neighbours	557	0:33	0:28
13. Binaries	20 687	10:44	9:51
14. Gravit. lens	1 172 336	1171:09	491:37

3.2.4 SSA

The SuperCOSMOS Science Archive (SSA) (Hambly et al. 2001) comprises data extracted from scans of photographic Schmidt survey plates, similar to the USNO catalogues. The resulting object detections form a catalogue of over one billion records covering the southern celestial hemisphere.

A copy of the SSA Source Table consisting of stellar and galaxy data was obtained from the Royal Observatory of Edinburgh. As mentioned for USNO B1, the entire dataset was loaded and issues of identifying stellar objects ignored.

4 Catalogue Queries

Database performance was initially assessed using examples motivated from the SDSS and SSA. Only the S4 was tested since this corresponds to the size and content of a catalogue expected from SkyMapper, the focus of this investigation. Appendix A contains a brief description of the queries with a complete listing available in Power (2006).

The query performance is shown in Table 2. Note that database indexes have been created on the appropriate table columns to ensure efficient query processing.

The first two queries are the same except for the tested *R* magnitude value, resulting in more records found which significantly increases its elapsed time. The third query includes extra magnitude conditions to check, reducing the number of records found, but requires more work by the database.

Queries 4 and 5 use magnitude differences, a 'colour cut', and are efficiently performed by each database.

The performance of the search for white dwarf stars was a surprise. It is a simple variation from the previous query, but was the worst for MYSQL and one of the slowest for ORACLE.

The quasar query involves a complicated stellar model and contains the most complicated SQL *where* clause tested.

```
select id, ra, de
from   s4_sub
where  acos(sin(radians(-15.137))*
            sin(radians(de))       +
            cos(radians(-15.137))*
            cos(radians(de))*
            cos(radians(134.154 - ra)))
       <= radians(0.25)
```

Figure 1 Cone search SQL.

The next four queries, 8–11, test aggregation using the SQL *group by* clause. These are slow queries but when a cone search region is included (Query 10) the spatial index makes the query faster.

The last three queries are examples of self-joins to find neighbours. Queries 12 and 13 target close objects by testing a small angular separation and are able to find a result within an acceptable time. The same can not be said for Query 14 where it takes over eight hours to complete for ORACLE and nearly 20 hours for MYSQL.

In summary, ORACLE is significantly faster (by a factor of 3) than MYSQL for half the queries, and about the same for the remainder. All the queries were performed adequately by each database, other than the last (a time-consuming join operation) and Query 6 for MYSQL.

5 Location Queries

The next set of tests were cone searches, a location query defined as: find all the objects within a certain radius of a given location on the celestial sphere.

5.1 Cone Search

A cone search query is performed by calculating the great circle distance from the cone centre to each target object and testing if this distance is within the required radius. An example SQL query that uses the spherical law of cosines to find the records within a 0.25-degree radius of the location 134.154, −15.137 in degrees is shown in Figure 1.

This query uses the s4_sub table, an S4 subset of approximately 74 million records used to check the queries before running them on the full S4 table. In all 9309 matching records are found and it takes just under three minutes for MYSQL and 9 and a half hours with ORACLE. This large disparity in elapsed time is due to ORACLE being very slow when using trigonometric functions and the use of a user defined *radians* function, a provided function in MYSQL, but not ORACLE.

An efficient cone search requires indexing.

5.2 Spatial Indexing

There are numerous indexing schemes to support spatial data, see Gaede & Gunther (1998) for a survey. The Hierarchical Triangular Mesh (HTM) spatial index (Kunszt et al. 2000) uses area decomposition. The celestial sphere is initially divided into eight spherical triangles: four for the north and four south. Each triangle is then divided into four and the process repeated recursively to a predetermined

depth. The triangles are numbered with the aim of preserving spatial proximity: areas close spatially are labeled with numbers that are also numerically close. The labels are mapped to a base-10 integer value termed the 'HTM id'.

The HTM source distribution includes SQL SERVER specific functions as stored procedures (non standard SQL extensions) to efficiently use the HTM spatial index. For example, the *fGetNearbyObjEq* function implements cone search.

In order to use the HTM spatial index, one solution would be to port the stored procedure code to each database system. This was not pursued because, apart from not wanting to port the code twice, the version of MYSQL originally used in this investigation did not support stored procedures. While the current version of MYSQL does, the decision had been taken early on that HTM support could be manifested externally to the database system by modifying the SQL queries as explained below. This has the advantage that databases which don't support stored procedures can be included in the assessment.

The functionality of the HTM spatial index can be used without stored procedures. Using the same cone search example from Figure 1, a 0.25 degree radius around the location 134.154, −15.137, an area on the celestial sphere encompassing the cone region is specified in terms of HTM id's by the condition:

```
htmId between 10853734678528
          and 10853785010175 or
htmId between 10853852119040
          and 10853868896255 or
htmId between 10853885673472
          and 10853919227903
```

This query fragment is evaluated by the function *fGetNearbyObjEq*. By modifying a user query to include the above *where* clause fragment, the benefit of the HTM spatial index can be utilised without requiring extensions to standard SQL.

In summary, a cone search query can be efficiently implemented in a database by generating a single HTM id for each point object location, creating an index on the HTM id column and generating queries that use HTM ranges in the SQL *where* clause.

5.3 Performance

Power (2006) reports twelve SQL query versions that perform a cone search using HTM id's in conjunction with various cone search expressions. For example using a haversine (Sinnott 1984), normalised Cartesian coordinates or the expression used in Figure 1. For each SQL query version, 500 random RA and Dec locations are generated and eight different cone search radii are used: 5, 15, 30 and 60 arcsec; then 5, 15, 30 and 60 arcmin. In all, over 40 000 queries are tested for each database.

Since the queries are placed randomly over the southern hemisphere, differing numbers of records are found, but the same cone search should find the same records regardless of which of the twelve query versions is used. This was true for ORACLE, but not always for MYSQL. This is believed to be due to issues of precision in floating point arithmetic or the implementation of trigonometric functions.

```
select id, ra, de
from   s4
where (htmId between 10853734678528
                 and 10853785010175 or
       htmId between 10853852119040
                 and 10853868896255 or
       htmId between 10853885673472
                 and 10853919227903)
  and  ra >= 133.8950142240700245
  and  ra <= 134.4129857759299682
  and  de >= -15.387
  and  de <= -14.887
  and  x * -0.6724205844444091 +
       y *  0.6925768998123125 +
       z * -0.2611279293024881
    >= 0.9999904807207345
```

Figure 2 Fastest-cone search.

Table 3. Cone search, radius 5 arcmin

Catalogue	Average records	MYSQL (sec)	ORACLE (sec)
S4	901	0.408	0.203
USNO B1	725	0.659	0.245
2MASS	309	0.323	0.179
SSA	1152	7.708	2.498

The consistently best performing query version was to use normalised Cartesian coordinates and include the cone's Minimum Bounding Rectangle (MBR) as well as the HTM id's. This is necessary in ORACLE in order to achieve acceptable query response times (avoiding trigonometric functions) and is advisable in MYSQL since it produces consistent results.

An example of such a query is shown in Figure 2, the same cone search as Figure 1, but expressed differently in SQL. Note the successive refinement in the query. The HTM id's are used as a coarse filter, then the MBR of the cone used as a further refinement before using the dot product between the two normalised Cartesian coordinates to test against the cosine of the cone search radius. There are no trigometric or radian functions used in the SQL, all expressions are pre-computed when generating the query.

Table 3 reports the average elapsed times for each catalogue using a search radius of 5 arcmin. ORACLE consistently performs two to three times faster than MYSQL.

5.4 Sorting

Database query performance can be improved by sequencing the table data in the same order as the index. This allows the table data to be accessed sequentially, improving disk I/O. When the index is on the *htmId* column then this also groups spatially close objects nearby on disk. For example, given a SQL query with the following *where* clause:

```
htmId between 10853734678528
          and 10853785010175
```

Table 4. HTM sorted data radius 5 arcmin (sec)

Catalogue	MYSQL	ORACLE
S4	0.142	0.195
USNO B1	0.259	0.193
2MASS	0.108	0.134
SSA	0.218	0.484

```
select a.id, b.id
from   s4 a, s4 b
where  a.id != b.id
  and  a.x * b.x +
       a.y * b.y +
       a.z * b.z
    >= cos(radians(15/3600))
```

Figure 3 SQL neighbour finding.

The index on the *htmId* column is used to find the disk references for the corresponding table data. If the table and index records are stored in ascending order of HTM id, then the table data will be read from disk sequentially, making better use of disk reads.

The effect of re-sequencing the data was tested by sorting each dataset by the *htmId* values before loading into the database and repeating the cone search queries. The results for each catalogue when using a cone search radius of 5 arcmin is shown in Table 4, the same tests as reported earlier in Table 3.

The improvement is dramatic for MYSQL, around three times faster than the original queries. There is only a small speedup for ORACLE, apart from the SSA result, and now both systems have similar performance.

6 Finding Neighbours

The next problem tested was neighbour finding: for each object in the catalogue, find the neighbours within a specified angular distance occurring in the same catalogue.

Various approaches to identifying neighbours were performed on a subset of the synthetic dataset. These variations are explained in Sections 6.1 and 6.2. The best approach of those tested is to use software to generate a list of neighbours, recording for each pair their record ids and angular separation, then loading the result into the database. This is the approach used in the SuperCOSMOS and SDSS archives.

6.1 SQL

Finding the nearest neighbours for all objects within a catalogue can be achieved using standard SQL. The query in Figure 3 finds all pairs of records in S4 within 15 arcsec of each other using an expression involving normalised Cartesian coordinates to determine the angular separation between points. Note the record ids are tested (!=) to ensure the same record is not matched with itself.

Six SQL variations of this neighbour finding query were tested in both databases on an S4 data subset with less than 100 000 records. The fastest query took an hour to complete and the slowest more than four.

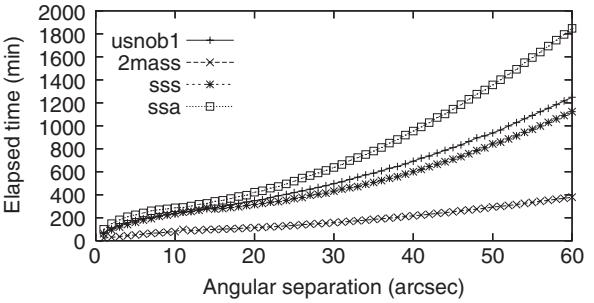

Figure 4 Neighbour-finding times.

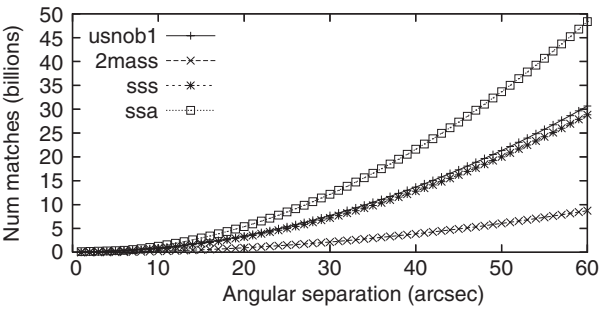

Figure 5 Neighbour-finding records.

6.2 Plane Sweep

A plane sweep solution was implemented next. This is an algorithmic technique used extensively in computational geometry to solve problems in two dimensional datasets. The data is first sorted in one dimension then an imaginary line is 'swept' along the sorted dimension and points processed as they are encountered.

The standard algorithm has been adapted for astronomy catalogues by accommodating the spherical geometry as described in Devereux et al. (2005). Notably:

- care must be taken at the poles;
- the sweep line distance requires cos(Dec) corrections;
- RA wrap around must be considered.

The implementation in a database can not be performed using standard SQL: procedural language extensions are required. This was only done for ORACLE, with a full description provided in Power (2006).

This solution improves performance from over an hour using the standard SQL solution of Section 6.1 to 6.5 min but is still much slower, by orders of magnitude, when compared to a C++ implementation that generates the same results in less than 10 sec.

This code has been adopted by the Royal Observatory Edinburgh where they use it to generate lists of neighbours. Their previous implementation took 48 hours to process a billion rows of data whereas now it takes five hours (R. Collins 2005, private communication).

6.3 Performance

Figures 4 and 5 show the performance of the C++ implementation (Devereux et al. 2005) of the plane sweep neighbour finding algorithm for each catalogue using angular separations ranging from 1 to 60 arcsec. The results of Figure 4 do not include the time to write the result to disk: the tests simply count the number of matching records.

Note that the USNO B1 and SSA tables both have around one billion objects, but SSA takes longer to process. This is because the SSA data is spatially denser than the USNO B1: the SSA table only covers the southern hemisphere whereas the USNO B1 covers the entire celestial sphere. The impact is that more neighbours will be found for SSA which requires more data processing as the sweep line encounters points in the sorted dataset.

The plane sweep algorithm to find neighbours requires the catalogue to be sorted in ascending sequence of Dec. Only the object location (RA and Dec) and unique identifier are required. The location is needed since it is used to determine proximity to other objects, while the id is used to record the result.

The results reported in Figure 4 do not include the time to sort the data by Dec. This pre-processing is accomplished by extracting the requisite information then sorting using specifically written software and takes between 1:30 and 2:15 hours to complete for each of the four catalogues tested.

7 Cross Matching

Cross matching is similar to finding neighbours, except now there are two datasets. The task is: given two catalogues, for each object in one, find the objects in the other that are within a specified angular separation.

The algorithm reported in Devereux et al. (2005) requires location error information to establish a statistical measure of the likelihood that two objects match. The matching reported here used the same software, modified to test a fixed radius, reporting all objects within the circle that match from the other catalogue.

7.1 Performance

The cross matching tests are performed as a pair wise comparison of the four test catalogues, each prepared by extracting the spatial description along with their unique identifier and sorted as done for neighbour finding (the same input files used for neighbour finding are used for cross matching). Table 5 summaries the elapsed time in hours for cross matching the various catalogues while Table 6 shows the number of records found, in billions. These tests include the time taken to write the results to disk.

SQL solutions were not attempted given the previous poor performance for neighbour finding and the similarity in algorithm.

These results show that the cross matching algorithm is more efficient when the smaller catalogue is used against a larger one. For example, it takes 4:38 to match 2MASS against SSA and 6:02 to match SSA against 2MASS. This is due to the processing performed when the sweep line encounters a point and relates to the density of the catalogue in the dimension in which the dataset is processed.

Table 5. Cross-matching times (hh:mm)

Catalogues	S4	USNO	2MASS	SSA
S4	8:24	7:04	4:59	8:52
USNO	6:19	9:37	6:32	8:10
2MASS	3:56	5:15	3:14	4:38
SSA	8:43	8:51	6:02	11:19

Table 6. Cross-matching records (10^9)

Catalogues	S4	USNO	2MASS	SSA
S4	4.42	2.57	1.43	3.88
USNO	2.57	4.94	2.07	4.38
2MASS	1.43	2.07	1.51	2.05
SSA	3.88	4.38	2.05	7.08

When catalogues of similar size are cross matched using the Devereux et al. (2005) algorithm the dataset with lower spatial density should be used 'first'. For example, matching USNO B1 against the SSA takes 8 hours 10 minutes whereas doing it in reverse takes 8 hours 51 minutes.

8 Discussion

One of the underlying assumptions not addressed here is that a database is the right environment for publishing a large catalogue such as the S4. The justification extends beyond astronomy specific issues and relates to the ease of data access available through the SQL query language. An alternative would be to develop software in C or FORTRAN to process the data with the aid of libraries for efficiently accessing astronomy datasets, such as WCSTOOLS[7]. However, the expressive power and simplicity of SQL can not be equalled in such a design.

A relational database provides a simple means of accessing astronomy catalogues. Important operations allowing data selection are feasible, making thorough exploration of the data to be achieved.

This report profiles queries for large catalogues residing in the MYSQL and ORACLE database systems. The tests were a selection of typical queries expected by users from the astronomy community, cone search, neighbour finding and cross matching. Most queries, other than neighbour finding and cross matching, were handled adequately by each database system.

ORACLE was consistently faster than, or the same as, MYSQL for the general queries tested on the S4 table. Cone search is well supported using the HTM as a spatial index in conjunction with an MBR constraint. ORACLE was originally between two and three times faster than MYSQL for

[7] tdc-www.harvard.edu/wcstools/

cone search queries. However, sorting the catalogues as a pre-processing step before database loading improves query response times for cone search, dramatically for MYSQL, such that the databases now report similar query response times as each other.

Neighbour finding for a catalogue of a billion objects can be achieved in around 6.5 hours: 2 hours preparation (extract location and record id, then sort by Dec) and around 4.5 hours to find the neighbours, depending on the target angular separation required. Similarly, cross matching two catalogues consisting of a billion objects can be achieved in around 12 hours: under 4 hours to prepare the two catalogues and just over 8 hours to do the matching itself. These two operations are not well supported for large catalogues in a database environment when only using SQL.

There are further avenues that could be explored. Parallelism of neighbour finding and cross matching using the zone approach from Gray et al. (2004) could yield improvements to the plane sweep algorithm. ORACLE supports parallelism using partitioning, where a single table is distributed across file systems in a way understood by the query optimiser. Query execution can then be distributed across file systems utilising multi-processor machines, as it is completely transparent from the user perspective. MYSQL can be installed on a loosely coupled computing cluster as a means of managing scalability.

The tests reported here define a baseline performance; there could be optimisations available that would improve response times. Databases are complex systems and optimal performance is often only achieved after careful tuning of the disk used, operating system configuration, database specific parameters and sometimes the SQL queries themselves. The results presented here have been achieved with minimal attention to these aspects.

Acknowledgements

This work would not have been possible without access to large astronomy catalogues. Thanks go to David Monet and his colleagues at the USNO for providing a copy of the USNO B1 catalogue; Nigel Hambly and Bob Mann for access to the SuperCOSMOS data; Paul Francis for the script to generate a realistic S4 synthetic catalogue; and to the 2MASS team for responding to an email with a set of five double-sided DVDs.

Thanks also to CSIRO staff Dave Abel, Drew Devereux and Peter Lamb for the original work on neighbour finding and cross matching.

This investigation was undertaken as partial fulfillment of the requirements for the Graduate Diploma in Science at the Australian National University supervised by Dr. Paul Francis.

References

Devereux, D., Abel, D. J., Power, R. A. & Lamb, P. R., 2005, ASP Conf. Ser. 347, ADASS XIV, Eds. Shopbell, P., Britton, M. & Ebert, R. (San Francisco: Astronomical Society of the Pacific), 346

Gaede, V. & Gunther, O., 1998, ACM Computing Surveys Volume 30 Number 2, 170

Gray, J., Slutz, D., Szalay, A. S., Thakar, I. A., vandenBerg, J., Kunszt, P. Z. & Stoughton, C., 2002, Microsoft Technical Report MSR-TR-2001-01

Gray, J., Szalay, A. S., Thakar, A., Fekete, G., O'Mullane, W., Heber, A. & Rots, A., 2004, Microsoft Technical Report MSR-TR-2004-32

Hambly, N. C., et al., 2001, MNRAS, 326, 1279

Kunszt, P. Z., Szalay, A. S., Csabai, I. & Thakar, A. R., 2000, ASP Conf. Ser. 216, ADASS IX, Eds. Manset, N., Veillet, C. & Crabtree, D. (San Francisco: Astronomical Society of the Pacific), 141

Monet, D. G., et al., 2003, AJ, 125, 984

Power, R. A., 2006, CSIRO ICT Centre TR 06/209

Preparata, F. & Shamos, M., 1985, Computational Geometry an Introduction (NY: Springer)

Skrutskie, M. F., et al., 2006, AJ, 131, 1163

Sinnott, R. W., 1984, S&T, 68(2), 159

Szalay, A. S., Gray, J., Thakar, A., Kunszt, P. Z., Malik, T., Raddick, J., Stoughton, C. & vandenBerg, J., 2000, ACM SIGMOD, Eds. Chen, W., Naughton, J. F. & Bernstein, P. A., 451

A S4 Catalogue Queries

The 14 queries tested in Section 4 are briefly described below with the SQL commands for a few presented.

The first two queries examine the impact of a query that counts fewer/more rows in the database by adjusting the *rmag* threshold. Query 1 tests for *rmag* values less than 21.75 while Query 2 tests for values less than 12.6.

Query 3, shown below, tests all *UGRIZ* magnitudes for values less than 12.6.

Query 3 UGRIZ magnitude query.

```
select count(*)
from   s4
where  umag < 12.6
  and  gmag < 12.6
  and  rmag < 12.6
  and  imag < 12.6
  and  zmag < 12.6
```

Queries 4–6 use magnitude differences to express colour constraints. The colour cut used in Query 6 below is from Gray et al. (2002) where it is described as a search for white dwarf stars.

Query 6 A search for white dwarf stars.

```
select count(*)
from   s4
where  umag - gmag < 0.4
  and  gmag - rmag < 0.7
  and  rmag - imag > 0.4
  and  imag - zmag > 0.4
```

Query 7 is also from Gray et al. (2002) and aims to find stars that match a quasar at redshift between 5.5 and 6.5. The query constraint is representative of a user query corresponding to a non-trivial stellar model and is the most complicated selection query tested.

Query 7 User-defined stellar model to find quasars.

```
select count(*)
from   s4
where (umag - gmag > 2.0 or
       umag > 22.3)
  and  imag between 0 and 19
  and  gmag - rmag > 1.0
  and (rmag - imag < (0.08 +
       0.42*(gmag - rmag - 0.96))
   or  gmag - rmag > 2.26)
  and  imag - zmag < 0.25
```

Query 8 is adapted from Gray et al. (2002) where binning the magnitude differences using the *group by* clause to convert them to integers is discussed. The resulting data is sorted using the *order by* clause and the output column names have been relabelled.

Query 9 is the same as Query 8 except that it includes the clause: *having count(*) <500* to restrict the number of results found per group.

Query 10 uses the HTM index to query a large region of the sky in conjunction with a simple colour cut. Query 11 is similar except that a different colour cut is defined and there is no cone search restriction.

Query 8 Find objects rare in colour space.

```
select round(umag - gmag) ug,
       round(gmag - rmag) gr,
       round(rmag - imag) ri,
       round(imag - zmag) iz,
       count(*)          pop
from   s4
group by round(umag - gmag),
         round(gmag - rmag),
         round(rmag - imag),
         round(imag - zmag)
order by pop
```

The remaining three queries, 12–14, are examples of a self-join. The neighbour table (s4_nn) records pairs of close stars by listing the records id's and angular separation in arc seconds up to a threshold of 10 arcsec. The neighbours are calculated as described in Section 6 and the result loaded into the databases.

Query 12 finds stars very close together, within 0.0083 arcsec, with one of the *UGRIZ* magnitudes having a variation greater than 0.1. Query 13 finds binary stars where one of the stars has the colour of a white dwarf using the same colour selection as Query 6.

Query 14 Gravitational-lens query.

```
select count(*)
from   s4 a,
       s4 b,
       s4_nn nn
where  a.id = nn.id1
  and  b.id = nn.id2
  and  nn.id1 < nn.id2
  and  nn.sep <= 5.0
  and  abs((a.umag - a.gmag) -
           (b.umag - b.gmag)) < 0.05
  and  abs((a.gmag - a.rmag) -
           (b.gmag - b.rmag)) < 0.05
  and  abs((a.rmag - a.imag) -
           (b.rmag - b.imag)) < 0.05
  and  abs((a.imag - a.zmag) -
           (b.imag - b.zmag)) < 0.05
```

The last query finds objects that are close to each other and have similar colours, which could be used to identify gravitational lens events. Whereas the previous two queries use small angular separations to find objects very close together, this time a large portion of the neighbour's table will be selected since one-quarter of the records are within 5 arcsec.