Query Optimizations

These are some examples of the huge speed-ups available by optimizing MySQL queries and indexes.

Optimizing queries the select objects in a region

The qserv_XXX UDFs are deprecated in favor of the scisql UDFs, which are documented here ?http://lsst-web.ncsa.illinois.edu/schema/sciSQL_0.3/ including an introductory section on spatial geometry ?http://lsst-web.ncsa.illinois.edu/schema/sciSQL_0.3/#s2.

Example from Serge explaining how to find objects within a rectangular area on the sky:

```
-- Create a binary representation of the search polygon
SET @poly = scisql_s2CPolyToBin(359.9601, 2.5952,
                                0.0398, 2.5952,
                                0.0398, 2.6748,
                                359.9601, 2.6748);

-- Compute HTM ID ranges for the level 20 triangles overlapping
-- @poly. They will be stored in a temp table called scisql.Region
-- with two columns, htmMin and htmMax
CALL scisql.scisql_s2CPolyRegion(@poly, 20);

-- Select reference objects inside the polygon. The join against
-- the HTM ID range table populated above cuts down on the number of
-- SimRefObject rows that need to be tested against the polygon
SELECT refObjectId, isStar, ra, decl, rMag
FROM SimRefObject AS sro INNER JOIN
    scisql.Region AS r ON (sro.htmId20 BETWEEN r.htmMin AND r.htmMax)
WHERE scisql_s2PtInCPoly(ra, decl, @poly) = 1;
```

If you are selecting a rectangular box (with constant-RA and constant-decl sides), you may wish to use the box function instead: `scisql_s2PtInBox(ra,decl, raMin, declMin, raMax, declMax) = 1` (raMin, declMin, raMax, declMax are constants as used in s2CPolyToBin).

Optimizing queries that join large tables

If your query joins with a large table (e.g., Object, Source, ForcedSource, etc.), your query is likely to run slow. In order to speed things up, you may try pre-filtering the tables. Depending on the filtering conditions (WHERE clause), you may be able to significantly cut the number of joined rows. If only 10% (or less) of the large table will be involved, it is often faster to create a table consisting only of involved rows (slightly larger is okay if the rows can be later filtered out by column or join expressions). Here is an example.

Original query:

```
SELECT
      fsrc.ra, fsrc.decl,
      deepSourceId,
      deepForcedSourceId,
      exp.run,
      fsrc.timeMid,
      scisql_dnToAbMag(fsrc.psfFlux, exp.fluxMag0) as g,
      scisql_dnToAbMagSigma(fsrc.psfFlux, fsrc.psfFluxSigma,
```

```
                            exp.fluxMag0, exp.fluxMag0Sigma) as gErr
FROM
      DeepForcedSource AS fsrc,
      Science_Ccd_Exposure AS exp
WHERE
      exp.scienceCcdExposureId = fsrc.scienceCcdExposureId
  AND fsrc.psfFlux is not null
  AND fsrc.psfFluxSigma is not null
  AND fsrc.ra >= 30.0
  AND fsrc.ra < 60.0
```

It is going through all ForcedSources, and for each of them joins with exposure table. A faster way to do that:

```
-- Create your own database (replace xx with your mysql user name)
CREATE DATABASE xx_tmp1;


-- Select all rows of interest (bright stars in this case) without
-- doing any joins, and save the results in your table. It will
-- still take a long time, but at least it will be a nice sequential
-- scan, which is the best you can do. To give you some idea about
-- timing: our DC_W13_Stripe82.ForcedSource is 1.8 TB, going through
-- the entire table at 100 MB/sec would take 6 hours, so don't expect
-- magic. :)
CREATE TABLE xx_tmp1.BrightStars
SELECT
     ra, decl,
     deepSourceId,
     deepForcedSourceId,
     timeMid,
     scienceCcdExposureId,
     psfFlux,
     psfFluxSigma
FROM
     DeepForcedSource
WHERE
     psfFlux is not null
 AND psfFluxSigma is not null
 AND ra >= 30.0
 AND ra < 60.0;


-- Add an index on BrightStars.scienceExposureId
ALTER TABLE xx_tmp1.BrightStars ADD INDEX(scienceCcdExposureId);


-- And then do the joins you need to do. Hopefully by now your
-- data set will be much smaller than the initial one, so the
-- join will be fast.
CREATE TABLE xx_tmp1.MyResults
SELECT
     bs.ra, bs.decl,
     deepSourceId,
     deepForcedSourceId,
     exp.run,
     bs.timeMid,
     scisql_dnToAbMag(bs.psfFlux, exp.fluxMag0) as g,
     scisql_dnToAbMagSigma(bs.psfFlux, bs.psfFluxSigma,
                           exp.fluxMag0, exp.fluxMag0Sigma) as gErr
FROM
     xx_tmp1.BrightStars AS bs,
```

```
     Science_Ccd_Exposure AS exp
WHERE
     exp.scienceCcdExposureId = bs.scienceCcdExposureId;


-- your results can now be retrieved using:
SELECT * from xx_tmp1.MyResults;
```

Of course you don't have to save your results in MyResults table, but saving results in a table has some benefits: you won't be flooded with output, you can reuse the results, etc...

Computation in DB instead of in Python

Using a GROUP BY to do aggregation in the database (when the aggregates are relatively simple to compute) is much faster than millions of individual queries to do aggregation object-by-object.

Original:

```
for objectIdResult in objectIdList:
    objectId = long(objectIdResult[0])

    query = """SELECT scienceCcdExposureId, filterId, psfFlux, psfFluxErr
               FROM Source
               WHERE objectId=%ld""" % (objectId,)
    ca.execute(query)

[... compute avgFlux array ...]

    query  = 'UPDATE Object SET'
    for filterId in valid[0]:
        filterName = filterMap[filterId]
        filterMagName = filterName + 'Flux_PS'
        query += ' %s = %f,'  % (filterMagName,
            fluxScale*avgFlux[filterId])

    query=query.rstrip(',')
    query += ' where objectId = %ld' % objectId
    ca.execute(query)
```

New:

```
CREATE TABLE NewFlux AS
SELECT objectId, S.filterId,
    SUM(IF(psfFluxErr<=0, 0, POW(psfFluxErr, -2)) * psfFlux / fluxMag0) /
    SUM(IF(psfFluxErr<=0, 0, POW(psfFluxErr, -2))) * 3.6307e-20 AS newFlux
FROM rplante_DC3b_u_pt11final.Source AS S
JOIN Science_Ccd_Exposure USING (scienceCcdExposureId)
GROUP BY objectId, S.filterId;

UPDATE Object, NewFlux
SET uFlux_PS = newFlux
WHERE Object.objectId = NewFlux.objectId AND filterId = 0;

[... and five more for the other filters ...]
```

Improvement:

From an estimated 36 hours to 1 hour.

Adding index (1)

Query:

```
SELECT m.refObjectId, m.objectId, r.uMag, r.gMag, r.rMag, r.iMag, r.zMag, r.yMag
FROM RefObjMatch AS m, SimRefObject AS r, Object AS o
WHERE m.refObjectId=r.refObjectId AND o.objectId=m.objectId
    AND m.angSep < 0.5
    AND m.nRefMatches=1 AND m.nObjMatches=1
    AND r.isStar=True AND r.isVar=False
    AND r.rMag<20 AND r.rMag>16
    AND o.rNumObs>1;
```

EXPLAIN showed that `SimRefObject` was being scanned.

New:

```
SET myisam_sort_buffer_size=1000000000;
ALTER TABLE SimRefObject ADD PRIMARY KEY (refObjectId);
```

Improvement:

From 1 hour to 1 minute.

Adding index (2)

Query:

```
SELECT s.psfFlux, zp.iMag, zp.objectId
FROM ZpData AS zp, Source AS s
WHERE s.scienceCcdExposureId = 4038489594
    AND s.objectId=zp.objectId
```

EXPLAIN output:

```
|1| SIMPLE | s  | ref | IDX_scienceCcdExposureId,IDX_objectId |
    IDX_scienceCcdExposureId | 9 | const | 1 | Using where |
|1| SIMPLE | zp | ALL | NULL |
    NULL | NULL | NULL | 59165 | Using where; Using join buffer |
```

No index is being used on `ZpData`, so the database is doing a full scan of it each time.

New:

```
ALTER TABLE ZpData ADD KEY IDX_objectId (objectId);
```

Improvement:

From at least 8 seconds per query to 0.15 seconds per query.