Storing Reference Catalog

LSST Database

Assumptions (based on DataAccWG telecon discussions Oct 23, Oct 6, Oct 2):

- data will come from several sources (USNO-B, simCat, maybe SDSS, maybe 2MASS [**TBD**])
- some fields will go over the pole
- we should keep reference catalogs for simCat and all-the-rest separate (almost nothing will match)
- size of the master reference catalog (MRC) will be ~1 billion rows (USNO-B), augmented by some columns (maybe from SDSS, maybe 2MASS). For now, assuming 100 bytes/row, that is 100GB [**need to come up with schema**]
- we will need to extract a subset of rows to create individual reference catalogs:
    - astrometric reference catalog (simCat, non-simCat)
    - photometric reference catalog (simCat, non-simCat)
    - SDQA astrometric reference catalog (simCat, non-simCat). It will need only bright, isolate stars [**open question: maybe we don't need separate SDQA cat**]
    - SDQA photometric reference catalog (simCat, non-simCat) - this is beyond DC3b [**see open question above**]
- each reference catalog will be ~ 1% of the entire MRC, so ~1GB
- some objects will be shared by multiple reference catalogs
- we do not need to worry about updating reference catalogs (e.g., the input source catalogs are frozen) - this is true for at least USNOB and SDSS.
- typical access pattern "give me all objects for a given CCD" - this implies a cone search with a ccd radius in ccd center, then would need to clip the edges.

Building the Reference Catalogs

ObjectIds from different data sources will not match, and we will need to run association (a la Association Pipeline) to synchronize them (this is a heavy operation).

Given the assumptions above and the results of the tests below, the best option seems to be:

- build a MRC (join data from all sources into one large table)
- extract reference catalogs and store separately (they are small, ~1 GB each in DC3b)
- could push the MRC to tapes if we need to recover some disk space
- should we ever need to introduce a new version of a source catalog - we would create a new MRC.

SDQA Catalogs:

- want bright stars: need to apply filter on magnitude
- want isolated stars: need to identify object without near neighbors
- see SdqaWcsFailureCheckStage for more details

The former is easy. The latter will require some thinking. It is a one-time operation, and we are talking about few million rows (1% of 1 billion = 10 millions, cut on magnitude will further reduce it). Options:

- maybe use custom C++ code, generate on the fly overlapping declination zones and stream by zone
- maybe use DIF-spatial-index-assisted join

[**This needs investigation**]

Spatial index: we are considering using ?DIF. **Todo:**

- **write up DIF capabilities**
- **test it**
- **also try InnoDB with cluster index on healpixId**

Description of the Tests

The tests were done on lsst-dev01 (2 CPUs), using /u1/ file system which can do ~45 MB/sec sequential read (tested using "dd if=/u1/bigFile of=/dev/zero")

Input data: USNO-B catalog, 1 billion rows (65GB in csv form). Schema:

```
CREATE TABLE XX (
  # objectId  BIGINT NOT NULL PRIMARY KEY, -- added later
  ra         REAL NOT NULL,
  decl       REAL NOT NULL,
  muRA       FLOAT NOT NULL,
  muRAErr    FLOAT NOT NULL,
  muDecl     FLOAT NOT NULL,
  muDeclErr FLOAT NOT NULL,
  epoch      FLOAT NOT NULL,
  bMag       FLOAT NOT NULL,
  bFlag      FLOAT NOT NULL,
  rMag       FLOAT NOT NULL,
  rFlag      FLOAT NOT NULL,
  b2Mag      FLOAT NOT NULL,
  b2Flag     FLOAT NOT NULL,
  r2Mag      FLOAT NOT NULL,
  r2Flag     FLOAT NOT NULL
);
```

Quick Summary

- Speed of loading is reasonable (~1 h 20 min)
- Speed of building indexes is acceptable (~5 h)
- Selecting small number of rows via index is very fast
- Speed of full table scan is at ~85% of raw disk speed (30 min)
- Selecting large number of rows via index (full index scan) is *unacceptably slow* (8 h)
- Speed of join is relatively slow: at 15% of raw disk speed (~5 h)

Details are given below.

Data ingest

Loading data done via

```
LOAD DATA INFILE 'x.csv' INTO TABLE XX FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'"
```

took 1h 19 min.

iostat showed steady 27 MB/sec write I/O, disk was 18% busy, CPU 90% busy.

The db file size (MYD): 60 GB

Building key

Command:

```
alter table add key(bMag)
```

CPU ~30% busy. Disk ~75% busy doing 41 MB/sec write and 21 MB/sec read

Took 5h 11min

MYI file size: 12 GB

Adding primary key

(This is not a typical operation - we won't be doing it in LSST)

Command:

```
ALTER TABLE XX
ADD COLUMN objectId BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY FIRST;
```

Took 15h 36min

MYD grew to 68 GB, MYI grew to 26 GB

Index for objectId takes 14 GB. (theoretically, 1 billion x 8 bytes is 8 GB, so 43% overhead)

Full table scan

Command:

```
-- note, there is no index on r2Mag
select count(*) from XX where r2Mag > 5.6
```

Takes 30 min

iostat shows between 36-42 MB/sec read I/O, which makes sense (68 GB MYD file in 30 min = 38 MB/sec). This is 84% of the best this disk can do.

Selecting through index

Small # rows

Selecting a small number of rows through index takes no time

Large # rows (full index scan)

```
-- there is an index on bMag
select count(*) from XX where  bMag > 4;
```

Took 7h 55 min

The result = 370 milion rows (37% of all rows)

iostat shows there a lot of small io (1MB/sec), which is consistent with elapsed time: IDX file size = 27.4 GB --> 1MB/sec. It looks like mysql is inefficiently walking through the index (tree), fetching pieces from disk in small chunks.

Note that a similar query but selecting small number of rows

```
-- this returns 0 rows
select count(*) from XX where  bMag > 40;
```

Takes no time.

As expected, disabling indexes and rerunning query takes 29 min 14 sec (full table scan)

Speed of join

We may want to join USNO-B objects with eg SDSS objects on the fly (assuming objectId are synchronized)...

```
-- create dummy sdss table
CREATE TABLE sdss (
  objectId  BIGINT NOT NULL PRIMARY KEY,
  sdss1     FLOAT NOT NULL,
  sdss2     FLOAT NOT NULL,
  sdss3     FLOAT NOT NULL,
  sdss4     FLOAT NOT NULL,
  sdss5     FLOAT NOT NULL,
  sdss6     FLOAT NOT NULL
);

-- in this case it is 25% of usno rows
INSERT INTO sdss
SELECT objectId, bMag, bFlag, rMag, rFlag,b2Mag, b2Flag
FROM XX
WHERE objectId % 4 = 0;
```

The loading took 1 h 21 min.

Doing the join:

```
CREATE TABLE refCat1
SELECT * FROM XX
JOIN sdss using (objectId);
```

Takes 3 h 54 min

It involved reading 68+8 GB and writing 23 GB. That gives speed 7 MB/sec, which is ~15% of the raw disk speed.

Speed of spatial join

TODO (Serge?)

Full USNOB-1 Schema

In Informix format (easy to map to MySQL)

```
create table "informix".usno_b1
  (
    usno_b1 char(12) not null ,
    tycho2 char(12),
    ra decimal(9,6) not null ,
    dec decimal(8,6) not null ,
    e_ra smallint not null ,
    e_dec smallint not null ,
    epoch decimal(5,1) not null ,
    pm_ra integer not null ,
    pm_dec integer not null ,
    pm_prob smallint,
    e_pm_ra smallint not null ,
    e_pm_dec smallint not null ,
    e_fit_ra smallint not null ,
    e_fit_dec smallint not null ,
    ndet smallint not null ,
    flags char(3) not null ,
    b1_mag decimal(4,2),
    b1_cal smallint,
    b1_survey smallint,
    b1_field smallint,
    b1_class smallint,
    b1_xi decimal(4,2),
    b1_eta decimal(4,2),
    r1_mag decimal(4,2),
    r1_cal smallint,
    r1_survey smallint,
    r1_field smallint,
    r1_class smallint,
    r1_xi decimal(4,2),
    r1_eta decimal(4,2),
    b2_mag decimal(4,2),
    b2_cal smallint,
    b2_survey smallint,
    b2_field smallint,
    b2_class smallint,
    b2_xi decimal(4,2),
    b2_eta decimal(4,2),
    r2_mag decimal(4,2),
    r2_cal smallint,
    r2_survey smallint,
    r2_field smallint,
    r2_class smallint,
    r2_xi decimal(4,2),
    r2_eta decimal(4,2),
    i_mag decimal(4,2),
    i_cal smallint,
```

```
    i_survey smallint,
    i_field smallint,
    i_class smallint,
    i_xi decimal(4,2),
    i_eta decimal(4,2),
    x decimal(17,16) not null ,
    y decimal(17,16) not null ,
    z decimal(17,16) not null ,
    spt_ind integer not null ,
    cntr serial not null
  );

revoke all on "informix".usno_b1 from "public" as "informix";

create index "informix".usno_b1_cntr on "informix".usno_b1 (cntr)  using btree ;
create index "informix".usno_b1_dec on "informix".usno_b1 (dec)    using btree ;
create index "informix".usno_b1_spt_ind on "informix".usno_b1     (spt_ind) using btree ;
```

Proposed Reference Catalog Schema

Work in progress

```
    objectId BIGINT NOT NULL PRIMARY KEY,
    ra DOUBLE,      -- ra
    decl DOUBLE,    -- del
    muRa DOUBLE,    -- pm_ra
    muDecl DOUBLE,  -- pm_dec
    parallax FLOAT, -- ???
    b1Mag FLOAT,    -- b1_mag
    r1Mag FLOAT,    -- r1_mag
    b2Mag FLOAT,    -- b2_mag
    r2Mag FLOAT,    -- r2_mag
    iMag FLOAT      -- i_Mag
    -- what about errors for all the above???
```