# Neighbors

LSST Database

Many queries, for example dbQuery010?, dbQuery013?, dbQuery028?, or dbQuery029? involve near-neighbor search. To address neighbor-related queries, SDSS maintains a pre-computed Neighbors table which keeps information about all neighbors of each object within 0.5 arcmin radius. The Neighbors table contains these fields: objID, neighborObjID, distance, type, neighborType and some SDSS specific "modes".

A typical query that uses the Neighbor table looks like:

```
SELECT *
FROM   Object    o1
JOIN   Neighbors n  ON (o1.objectId = n.objectId)
JOIN   Object    o2 ON (o2.objectId = n.neighborId)
WHERE  n.distance < :distance
  AND  "o1 attribute restrictions"
  AND  "o2 attribute restrictions"
  AND o1.objectId <> o2.objectId
```

This query typically involves a full Object table scan to process the o1 attribute restrictions (since they often select a large fraction of the Objects), a full scan of the Neighbor table (since the selectivity is again low), plus random reads from the Object table (unless the distance is very selective in which case an index on distance could be used).

Maintaining the Neighbor table has some drawbacks

- storage cost (~40 TB in first Data Release, ~130TB in last Data Release - see below for details)
- maintenance costs

### Maintenance cost

The maintenance cost of the Neighbor table arises not so much from creating its rows but instead from keeping the information in the table up to date. The more precomputed information we keep, the more difficult it becomes to maintain this table. For example if we choose to store classification types and probabilities (this neighbor is a galaxy...), we would have to update many rows in the Neighbors table every time an object is reclassified or the classification probability changes.

### Storage cost

In LSST the Neighbor table would be ~40 TB in size in data release 1, and ~130 TB in the last data release. This is based on these assumptions:

- 4 million objects per FOV on average, and 10 sq deg FOV implies 111 objects per sq arc min
- 0.5 arc min radial neighbor limit implies pi/4 sq arc min neighbor area
- 15 billion Objects in DR1
- row size 22 bytes (objectId 8, neighborId 8, distance 4, type 1, typeProbability 1, neighborType 1, neighborTypeProbability 1).

# Alternative, partition-based approach

An alternative approach that relies on data partitioning could be used. Suppose the Object table is partitioned into 1 million non-overlapping sub-partitions as shown below.

Sub-partitions are marked in black, size of a sub-partition is at least 1 x 1 arc min. Each object belongs to exactly one sub-partition. To find objects' neighbors we need to scan up to 4 adjacent sub-partitions as shown (green, purple and blue lines), so let's call such group of 4 an ObjectPartition, and make ObjectPartition a unit of processing. Then for each ObjectPartition a query would look like

```
SELECT *
FROM   ObjectPartition o1
JOIN   ObjectPartition o2 USING (objectId)
WHERE  "distance between o1 and o2" < :distance
  AND  "o1 attribute restrictions"
  AND  "o2 attribute restrictions"
  AND  o1.objectId <> o2.objectId
```

Suppose we have 1000 processing nodes. Each node processes 1000 sub-partitions (250 ObjectPartitions) sequentially. Notice that

- an ObjectPartition can now easily fit into memory, so self-join is fast
- each sub-partition is processed 4 times (except edges) but we don't need to load each 4 times into memory, since the previous ObjectPartitions already loaded most of them. So this query is essentially equivalent to one full table scan of the Object table, assuming that the database cache allows previously-used partitions to remain memory-resident until they are no longer needed.