Energy Compensation System

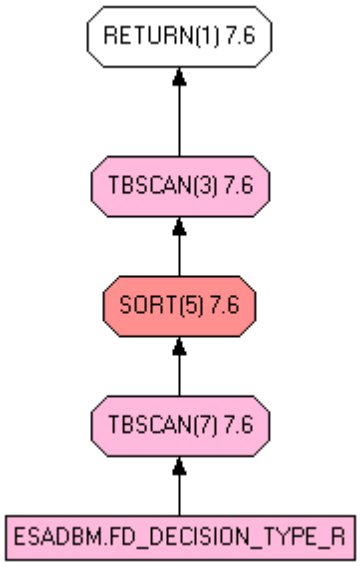
Database Optimization

Small and Static Tables

Fetching all data from a small table (like a lookup or reference table)

```
SELECT *
FROM ESADBM.FD_DECISION_TYPE_R
WHERE COALESCE(ENABLED_FLAG, 0) = 1
ORDER BY FD_DECISION_TYPE ASC;
```

Doing an access plan on that query yields:



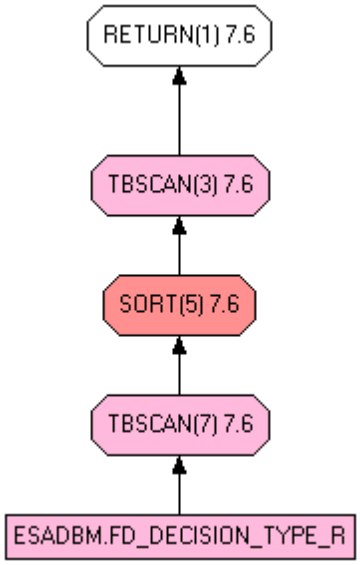
The cost (in timeron) associated with such trivial query is so low this table will most likely not benefit from the addition of an index on the ENABLED_FLAG column. Most records in lookup tables are enabled (i.e. have their ENABLED_FLAG fields set to 1) so most entries will be included in the index – which defeats the purpose of having an index in the first place.

To prove my point… let's add an index on FD_DECISION_TYPE_R.ENABLED_FLAG

```
CREATE INDEX ESADBM.FD_DECISION_TYPE_EF
ON ESADBM.FD_DECISION_TYPE_R(ENABLED_FLAG) ALLOW REVERSE SCANS;

RUNSTATS ON TABLE ESADBM.FD_DECISION_TYPE_R
WITH DISTRIBUTION ON ALL COLUMNS AND DETAILED INDEXES ALL;
```

The access plan on that newly-optimized table stays the same:



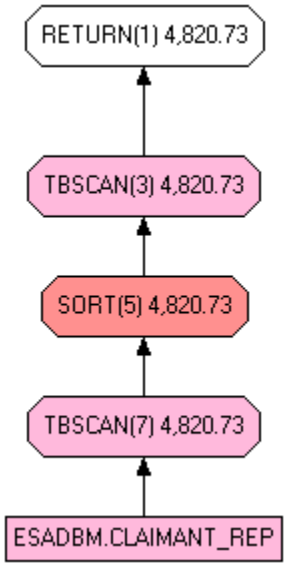
Conclusion: don't waste your time (and server bandwidth) trying to optimize small lookup tables.

Large Data Tables

Fetching specific data from a large table can be time-consuming if not indexed properly:

```
SELECT *
FROM ESADBM.CLAIMANT_REP
WHERE CLAIMANT_ID = ?;
```

Doing an access plan on that query yields:



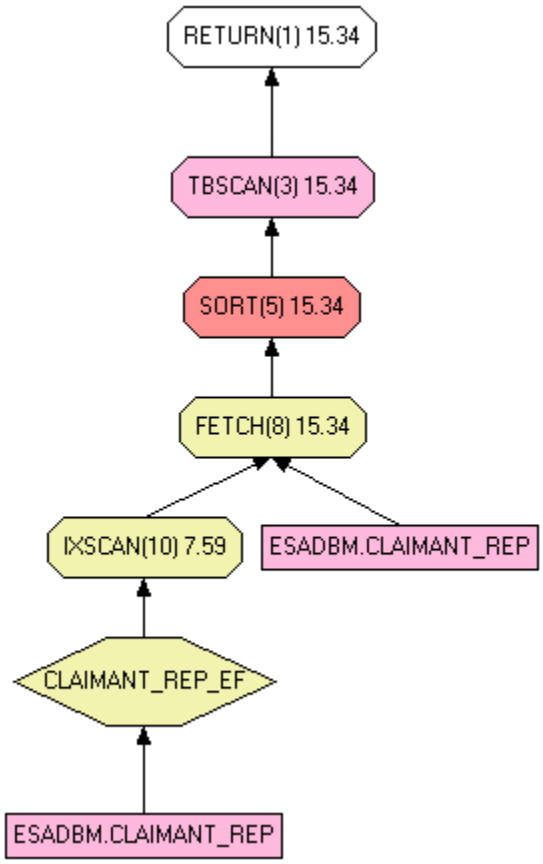
So clearly the database server is doing a full scan on that table to find a couple of claimant-related records.

In this case an index will obviously be beneficial:

```
CREATE INDEX ESADBM.CLAIMANT_REP_CLMT_ID
ON ESADBM.CLAIMANT_REP(CLAIMANT_ID ASC) ALLOW REVERSE SCANS;

RUNSTATS ON TABLE ESADBM.CLAIMANT_REP
WITH DISTRIBUTION ON ALL COLUMNS AND DETAILED INDEXES ALL;
```

The access plan on that newly-optimized table shows a huge improvement:



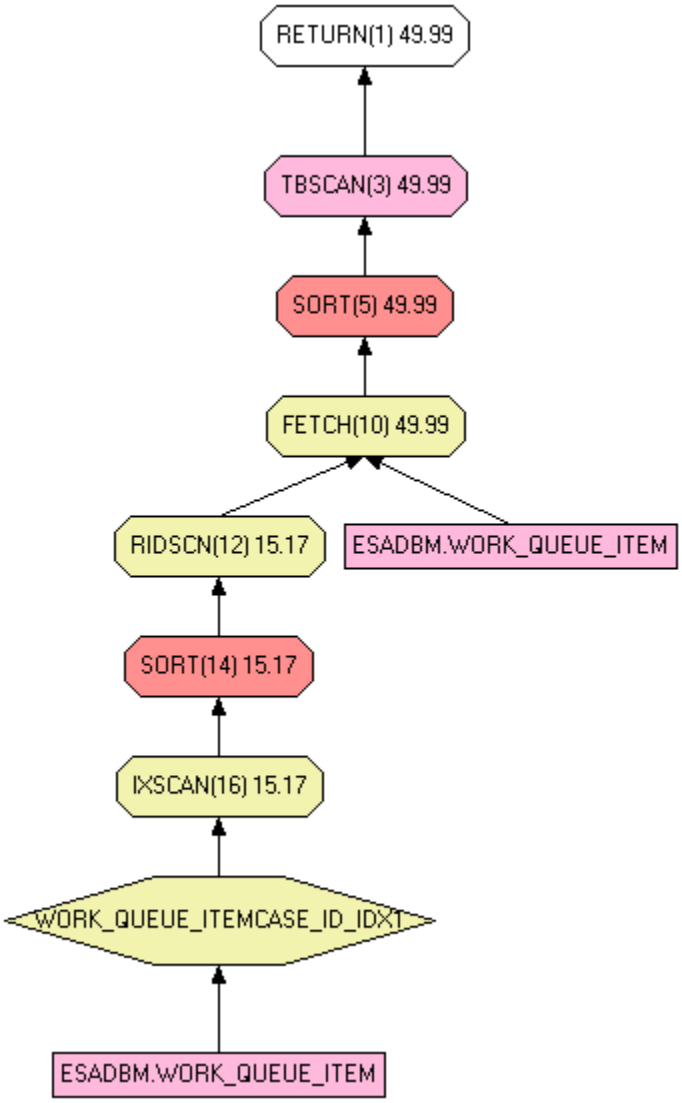
Conclusion: make sure child tables have the relationship to their parent table properly indexed.

Large Data Tables That Won't Benefit From Further Optimization

Fetching specific data from a large table can be time-consuming if not indexed properly:

```
SELECT *
FROM ESADBM.WORK_QUEUE_ITEM t0
WHERE (t0.CASE_ID = ? AND t0.WORK_CATEGORY_CD = ?
AND t0.WORK_ACTION_CD = ? AND t0.WORK_ITEM_STATUS_CD = ?)
ORDER BY t0.START_DT DESC;
```

Doing an access plan on that query yields:



The cost associated with such query is fairly low as a composite index on CASE_ID and WORK_ITEM_STATUS_CD already exists.

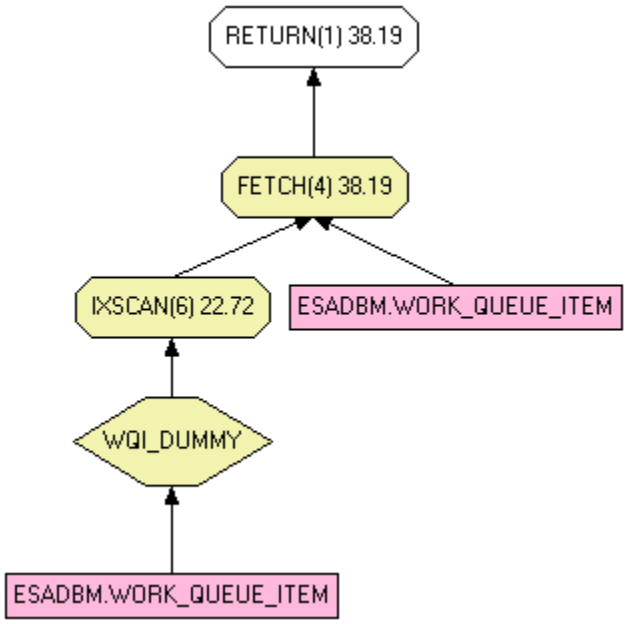
Table WORK_QUEUE_ITEM is a huge table there is no doubt that performing any depth of table scan in it is a very expensive process. Considering that the query provided above also includes filters on WORK_CATEGORY_CD, WORK_ACTION_CD and START_DT the question becomes: can we optimize this even further?!?

Let's find out!

```
CREATE INDEX ESADBM.WQI_DUMMY
ON ESADBM.WORK_QUEUE_ITEM(CASE_ID, WORK_CATEGORY_CD,
    WORK_ACTION_CD, WORK_ITEM_STATUS_CD, START_DT)
ALLOW REVERSE SCANS;

RUNSTATS ON TABLE ESADBM.WORK_QUEUE_ITEM
WITH DISTRIBUTION ON ALL COLUMNS AND DETAILED INDEXES ALL;
```

Things got a little better, but considering that this rather complex index now has to get updated upon DELETE/UPDATE, this new index is probably not worth it…



Conclusion: some tables are large by nature and can only be optimized to some degree.