# Making SQL great again

# Collecting Statistics

## Why do we need statistics

- The execution strategy is based on the statistics collected on the tables used within the SQL query. Statistics on the table is collected using the `COLLECT STATISTICS` command.
- Optimizer requires environment information and data demographics to come up with the optimal execution strategy.

# Information collected

- **Environment Information**
  - Number of Nodes, AMPs and CPUs.
  - Amount of memory.
- **Data Demographics.**
  - Number of rows.
  - Row size.
  - Range of values in the table.
  - Number of rows per value.
  - Number of Nulls.

## Approaches

There are three approaches to collect statistics on the table.

- Random AMP Sampling.
- Full statistics collection.
- Using `SAMPLE` option.

# Collecting and viewing statistics

```
/* Retrieve statistics */
collect statistics
on tutorial.salestransaction column(customerid);

/* Show collected statistics*/
help statistics tutorial.salestransaction;
show statistics on tutorial.salestransaction;
```

## Recommendations for stats collection

- At least on Primary Index column/columns.
- Update collect stats after 10% change in data of a table.
- Non unique secondary indexes(NUSI) for optimizer to know total number of rows in NUSI subtable and make better decisions regarding the cost savings.
- Non-indexed columns that are frequently used in `WHERE` and `JOIN` clauses.
- Drop statistics that are not used anymore(i.e. stats on columns that no longer used to filter data or used in joins) as it consumes space and consumes CPU to update them.
- Stats collection should happen during non-peak hours, this is because stats collection takes high CPU usage.

# Too much to remember? No worries

```
/* Suggest which statistics to collect */
DIAGNOSTIC HELPSTATS ON FOR SESSION;
explain select sum(noofitems)
from tutorial.salestransaction sales
left join  tutorial.soldvia sold
on sales.tid = sold.tid;
```

- Collect the stats only if the optimizer is recommending high confidence.

# Set how statistics will be sampled

```
/* Random sample */
COLLECT STATISTICS
USING SAMPLE 10 PERCENT
COLUMN productprice
ON tutorial.productid;

/* Start with full-table scan, but stop at some point*/
COLLECT STATISTICS
USING SYSTEM SAMPLE
COLUMN productprice
ON tutorial.productid;

/*Full-table scan to collect the specified stats.*/
COLLECT STATISTICS
USING NO SAMPLE
COLUMN productprice
ON tutorial.productid;
```

```
EXPLAIN
```

## Explaining `EXPLAIN`

- Explain plan is the step-by-step description of a query plan generated by the parsing engine.
- Explain plan can tell you how Optimizer will execute a query.
- Although you can not directly modify it, you can modify your query to influence the execution plan.

## Explaining `EXPLAIN` (cont.)

- **Locking Information:** Explain plan provides information about locking. This lock can be like Pseudo table which is placed to avoid global deadlock conditions, read, write, access and exclusive lock.
- **Row retrieval Strategy:** Teradata may fetch rows by full table scan, using primary index, using secondary index or any other access path.
- **Time & size estimation information:** Explain plan also provides the information about estimated row counts and estimated time to complete a particular step and query.

## Explaining `EXPLAIN` (cont.)

- As all the AMP work independently, they can not access other AMP data directly. So for join processing rows should be in the same AMP. Teradata decides to redistribute, duplicate to bring the rows to be joined on the same AMP.
- **Join information:** In case of join operation, explain plan will show you what kind of join operation is chosen by the optimizer base on the situation. You will see something like product join, single partition hash join, merge join etc.

## Explaining `EXPLAIN` (cont.)

- **Confidence level:** In the explain of a query, you will find something like high confidence, low confidence, no confidence. These are obtained through the statistics collection phase.
- **AMPs involvement information:** During any kind of operation like retrieving rows, joining tables, aggregation you will get information about how many amps take part in that operation. Depends on the work, it can be single AMP, group AMP or all AMPs.

## Things to look after in `EXPLAIN`

- Join strategy used ($O(mn)$ complexity if product join).
- All-rows scan ($O(\log(n))$ complexity).
- Confidence level (that XIIth century map).

# SQL Tuning

## Other performance tips

- If `LIKE` used in a `WHERE` clause, it is better to try to use as many leading characters as possible.
- Avoid use of large list of values in `IN/NOT IN` clauses. Store them in some temporary table and use that table in the query.
- If values are nearly unique values then `DISTINCT` clause may outperform `GROUP BY`. When there are many duplicate value then `GROUP BY` performs better than `DISTINCT`.
- When using `CASE`, start with the most frequent values to avoid checking all conditions.