

SQL Engine 16.20.x Overview

John Gill, Engineering Manager, Teradata
Phil Benton, Lead Product Architect, Teradata

- **16.20.x Feature Overview**
 - Performance
 - Security
 - Ease of Use
 - Analytics
 - Data Stream Architecture
- **Time Series**
- **Vantage Architecture – SQL / ML Engine Integration**
- **17.0+ Futures**



- **In-Memory Bulk Qualification**

- Bulk qualification can now be used for use on single column qualification with ROW tables (RET, SUM, Join steps) and Column Partitioned tables (SUM, Join steps)

- **JOIN INDEX Enhancements**

- Enhancements allow JIs to be used with decomposed aggregations, constant arithmetic, casting and partially covering queries using OUTER JOIN or EXISTS
- AJIs can be used with COUNT (*) , compressed AJIs can be used with OUTER JOINS

- **Column Partitioned table performance improvements** (update 2)

- VLC aware single column aggregations
- N-Way join optimization for star schema joins
- INSERT ... SELECT performance improvement

- **QueryGrid optimizations**

- Pipelining of format conversion & retrieve into a single step
- Common covering remote table optimization
- Remote table support with UDT (update 2)

- **Dynamically Parameterized literals**

- Literals parsed to allow requests to be cached like we can with USING parameterization

- **JSON Web Token Authentication**

- SSO capability with JWT through AppCenter for JWT Authentication via appcenter.
- JWT support for JDBC and .NET client (update 2)

- **Kerberos everywhere**

- All client interfaces support Kerberos as of SQL engine 16.10 release
- UDA SSO will be available in future with new capability to forward Kerberos tokens from SQL engine

- **Improved Quality of Protection Algorithms**

- New algorithms meet current security standards for message encryption and integrity.
- AES in GCM, CCM, and CTR modes, SHA-512 message integrity codes, Keyed HMAC on full cipher text

- **DBQL**
 - Multi-statement `INSERT / UPDATE / DELETE` row counts populated in `DBQLogtbl`
- **MAPS**
 - `ResusageSMHM` table to track MAPs level resource consumption
 - MAPs analyzer can be run prior to system expansion (update 2)
- **SHOWCOMPRESS**
 - Table level compression information exposed via SQL interface
- **JSON (update 2)**
 - New `INSERT INTO ... JSON` and `SELECT AS JSON` statements for shredding and composing JSON

- **New analytic functions**
 - DecisionTreePredict, GLMPredict, DecisionForestPredict, NaiveBayesPredict, NaiveBayesTextClassifierPredict, SVMSParsePredict
 - Sessionize, nPath, Attribution
 - AntiSelect, Pack, Unpack, StringSimilarity, Moving Average, nGramSplitter (update 2)
- **CSV DATASET Type**
 - Store CSV files in a DATASET CSV column
- **Temporal**
 - Support of [VALID/TRANSACTION/SYSTEM TIME] AS OF < Column name >
 - Allow CURRENT/NONSEQUENCED VALID/TRANSACTION TIME in FROM clause
- **PIVOT enhancements**
 - Allow aggregation of PIVOT generated columns (update 2)
 - Ability to generate PIVOT columns via IN-LIST (update 2)

- **Support for Down AMP in Backup and Restore**
 - No Fallback Tables will be Skipped
 - CBB jobs will still be rejected as we require WLSN from every AMP
- **Empty Table Restore Enhancements**
 - During Backup track empty tables, then during Restore skip Data and Build Phases
- **Compressed on Restore Table Settings**
 - Enforce System BLC settings on restore for tables
- **Support Multiple DSC Systems with one DBS at the same time**
 - Support Data Mover DSC and BAR DSC communication with single DBS
- **Restore Restart Capability**
 - Currently, when a DSA restore job does not complete successfully, the entire job must be restarted from the beginning.
 - Allows failed restore jobs to restart without having to run the whole job again

Time Series Tables

Primary Time Index

Time-aware Primary Index with respect to

- Storage distribution strategies and
- In-table local ordering strategy.



Today data is distributed on the basis of scalar column values: {A,B,C};

With a Primary Time Index the data distribution can include time-segments: {A,B, ΔT }

The Primary Time Index feature leads to “time-range” efficient queries

Create Table with PTI

Depending on the options specified in the PTI clause, Teradata **auto-generates up to 3 columns**:

- TD_TIMEBUCKET BIGINT NOT NULL
 - Values are populated and managed by Teradata
 - Hidden column: Cannot be loaded, updated, selected or referenced in a query
- TD_TIMECODE TIMESTAMP(6) NOT NULL
 - Value must always be provided by the user
 - Should be the same as timecode_dt in PTI clause
- TD_SEQNO INT NOT NULL
 - Generated when the SEQUENCED clause is specified
 - Used to order the rows along with TD_TIMECODE
 - TD_SEQNO can be between 1 to 2147483647 inclusively

The user can explicitly specify these columns in the CREATE TABLE (to alter the FORMAT or TITLE associated with the TD_TIMECODE or TD_SEQNO columns) or they will be auto-generated.

```
CREATE [SET|MULTISET] [GLOBAL TEMPORARY |  
VOLATILE ] TABLE series_table_name<  
[, <table options> ]  
(  
    [ <auto_generated_columns>,]  
    <column definitions> )  
PRIMARY TIME INDEX <optional_index_name>  
    ( <timecode_dt>  
    [, <timezero_date> ]  
    [, <timebucket_duration> ]  
    [, <columns_clause,> ]  
    [, <sequenced_flag> ] )  
[ <as clause> ]  
[ <index definitions> ]  
[ <commit options> ] ;
```

Create Table with PTI

```
timecode_dt      { TIMESTAMP |  
                  TIMESTAMP WITH TIME ZONE |  
                  DATE }
```

```
timezero_date    DATE  
specifying the "time zero" associated with table.  
Default timezero_date is January 1st, 1970 @ 00:00:00  
hours.
```

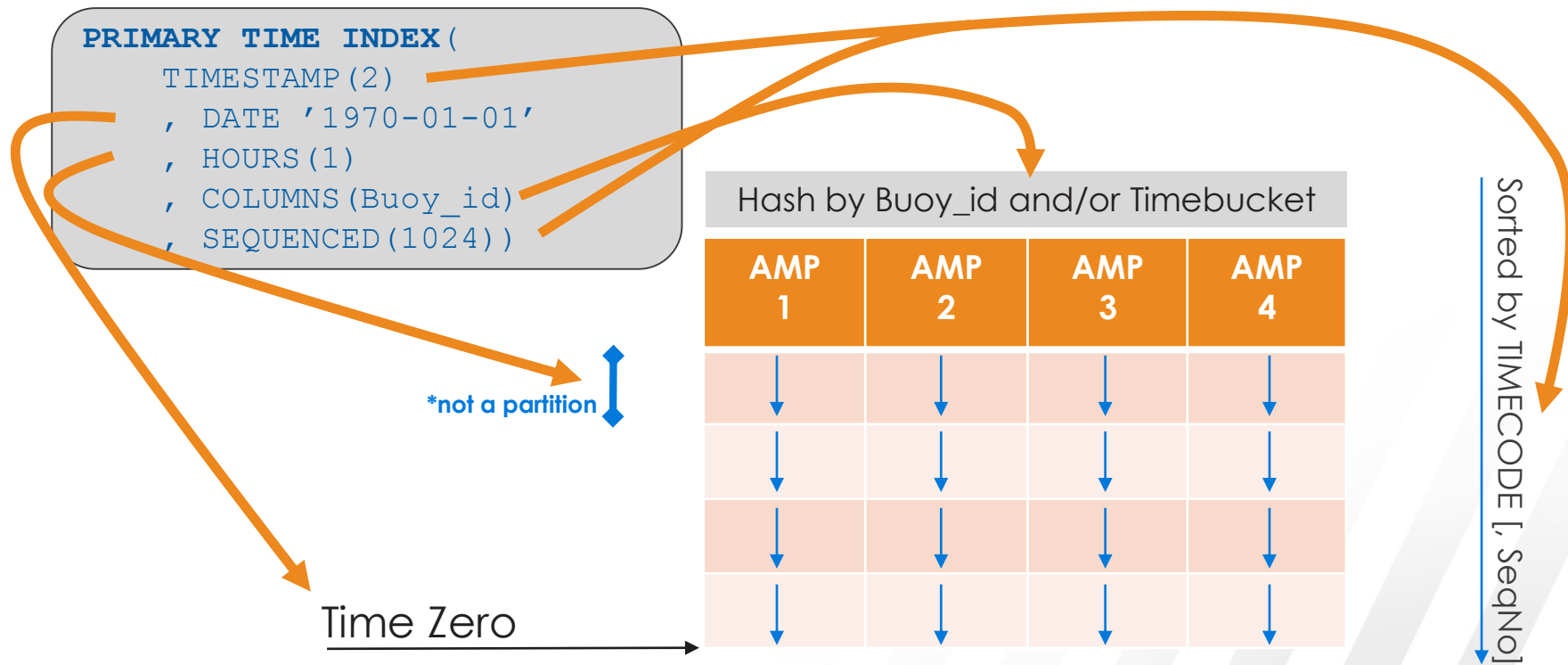
```
timebucket_duration <time duration>  
specified by CAL_YEARS, CAL_MONTHS, CAL_DAYS, WEEKS,  
DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS,  
MICROSECONDS
```

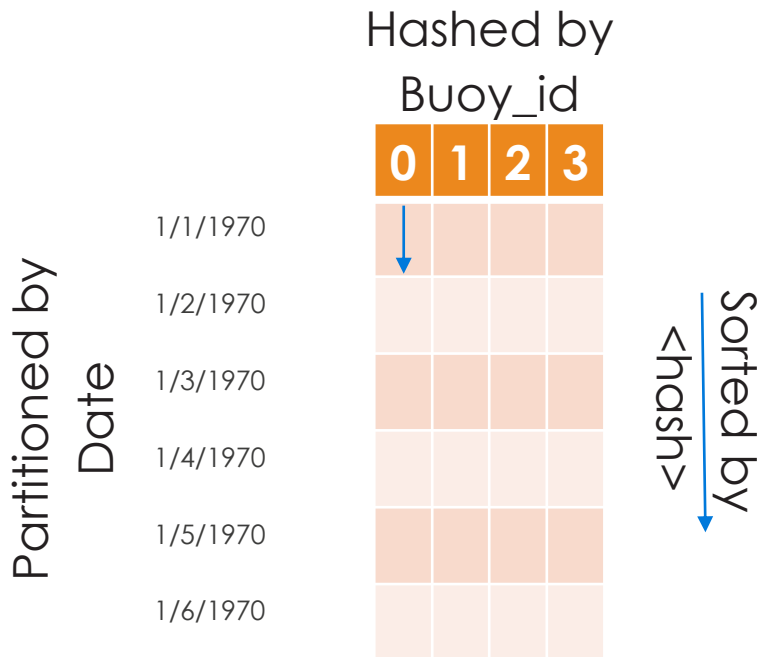
```
columns_clause   COLUMNS ( <column_list> )
```

```
sequenced_flag   { SEQUENCED <optional_maximum> | NONSEQUENCED }
```

```
... PRIMARY TIME INDEX  
<optional_index_name>  
  ( <timecode_dt>  
    [, <timezero_date> ]  
    [, <timebucket_duration> ]  
    [, <columns_clause,> ]  
    [, <sequenced_flag> ] ) ...
```

PTI Table





Physical Layout

- Data hashed by Buoy_id across AMPs
- Partitioned by Date (or Date and Hour, ...)
- Sorted by <hash> within the partition

Query

- Average temperature by buoy_id and Day (or Hour or Minute or 12 Seconds) for readings between 1/2/1970 13:00:00 and 1/4/1970 11:00:00

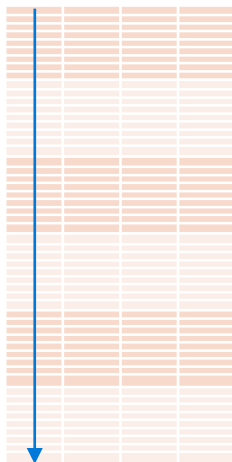
Plan

- Read 3 Partitions with condition
- Sort
- Group By and Aggregate

Few Slides on Partitions – PTI

Hashed by
Buoy_id, <timebucket>

0 1 2 3



Partitioned by
<timecode>

Sorted by
<timecode>

Physical Layout

- Data hashed by Buoy_id, <timebucket> across AMPs
- Partitioned by <timecode>; implied sort
- Sorted by <seq#> within the partition

Query

- Average temperature by buoy_id and Day (or Hour or Minute or 12 Seconds) for readings between 1/2/1970 13:00:00 and 1/4/1970 11:00:00

Plan

- TimeRangeScan
Def: Position 1st row, sequential read until last row
- Group By and Aggregate

Aggregations

```
SELECT AVG(EngineTemp)
FROM MyEngine
GROUP BY Country, Car_id
WHERE TIMECODE BETWEEN TIMESTAMP '10-01-2016 08:00'
                        AND TIMESTAMP '10-01-2016 09:00';
```

Returns **single** result per Country and Car_Id with AVG(EngineTemp) for rows found between **8:00 and 9:00**

```
SELECT $TD_TIMECODE_RANGE, AVG(EngineTemp)
FROM MyEngine
GROUP BY TIME (MINUTES(15) AND Country, Car_id)
        USING TIMECODE(TD_TIMECODE)
WHERE TIMECODE BETWEEN TIMESTAMP '10-01-2016 08:00'
                        AND TIMESTAMP '10-01-2016 09:00';
```

Returns **4 results** per Country and Car_Id with AVG(EngineTemp) for rows found **8:00-8:15, 8:15-8:30, 8:30-8:45, 8:45-9:00**

Time Aware Aggregation Functions

Existing Aggregate Functions

Average	Count
Describe	Kurtosis
Maximum	Minimum
Percentile	Rank
Skew	Sum
Std. population deviation	Std. sample deviation
Population variance	Sample variance

If not in the list above, then function is not time aware and cannot be used with the GROUP BY TIME clause

New Aggregate Functions

Bottom	Delta_T
First	Last
Median	Mode
Top	Mean absolute deviation

These new aggregate functions are only invocable with the GROUP BY TIME clause

Group By Time Clause

- Enhancement to SELECT clause.
- Allows an aggregate function to be computed on data that is grouped in terms of time
- Supported for both PTI & non-PTI tables
 - When used with non-PTI tables, the USING TIMECODE clause must be used
- Grouping is done by the time bucket number and then by any other fields specified in the GROUP BY TIME clause
- Both GROUP BY TIME & GROUP BY <other_element> cannot be used in the same query

GROUP BY TIME

```
'(<timebucket_duration> [  
AND  
<other_grouping_elements>]')'
```

```
SELECT MODE (col1)  
FROM aggr_table  
GROUP BY TIME(cal_years(100) AND  
bouy_id)  
USING TIMECODE(ts)  
ORDER BY 1
```

DELTA_T calculates the time difference between the occurrence of a starting event and ending event

Which vendors missed their promised delivery times on orders in 2016?

```
SELECT Orders.Order_Number, Orders.Vendor, Orders.Part_Name, Orders.Part_Number
      , INTERVAL(OrderPeriod) DAY TO SECOND AS Duration, OrderPeriod
FROM (SELECT Order_Number
      , DELTA_T((WHERE Status='Ordered'), (WHERE Status='Delivered')) AS OrderPeriod
      FROM Order_Status_Log
      WHERE TD_TIMECODE BETWEEN TIMESTAMP '2016-01-01 00:00:00'
                        AND TIMESTAMP '2016-12-31 23:59:59'

      GROUP BY TIME(* AND Order_Number)
      ) AS TSQ
      , Orders AS Orders
WHERE TSQ.Order_Number = Orders.Order_Number
      AND Duration > INTERVAL '4 00:00:00' DAY TO SECOND
ORDER BY Orders.Order_Number;
```

Missing Value Imputation: FILL

If the values are missing for an entire time bucket, the FILL Clause can be used with the GROUP BY TIME clause.

The options are:

- Ignore the missing value (NULL)
- Remove the row with the missing values
- Use the value from previous or next row
- Update the missing values with a constant or an estimate value.

```
SELECT $TD_TIMECODE_RANGE, $TD_GROUP_BY_TIME,  
       bouy_id, AVG(temperature) FROM OCEAN_BUOYS  
WHERE TD_TIMECODE BETWEEN TIMESTAMP '2014-01-06 09:45:00' AND  
                                     TIMESTAMP '2014-01-06 11:45:00'  
  
AND bouy_id=44  
GROUP BY TIME (MINUTES(15) AND bouy_id) FILL (PREV)  
ORDER BY 2,3;
```

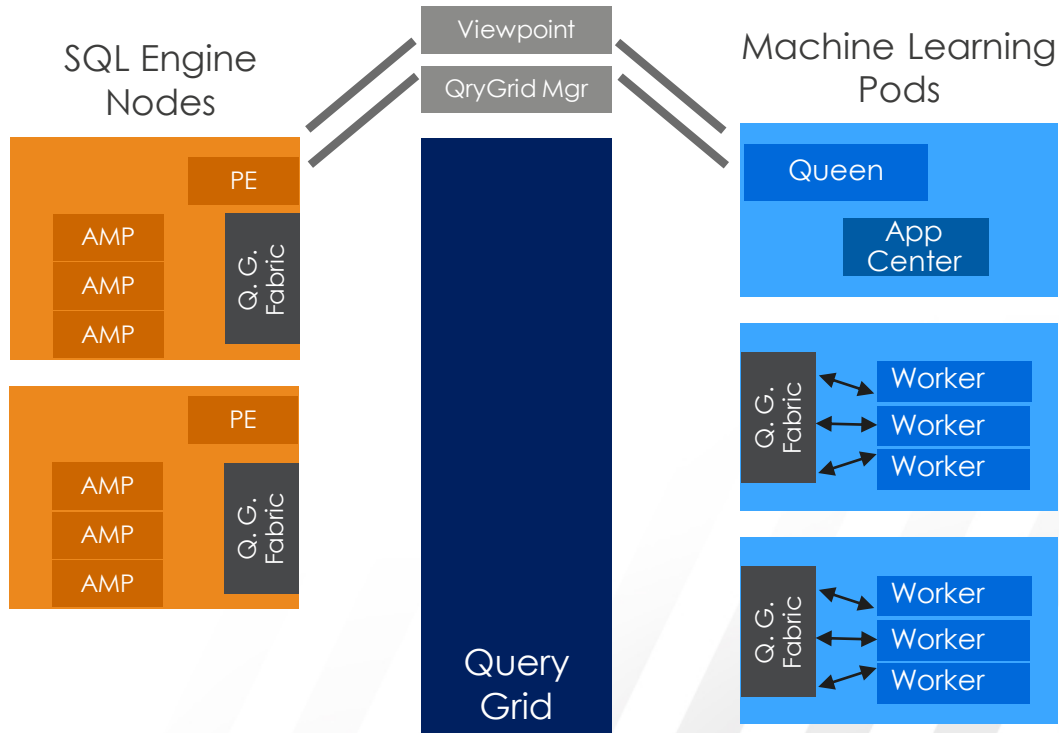



Vantage Architecture

SQL / ML Engine Integration

New Machine Learning (MLE) coprocessor nodes are deployed via Stacki on top of Kubernetes

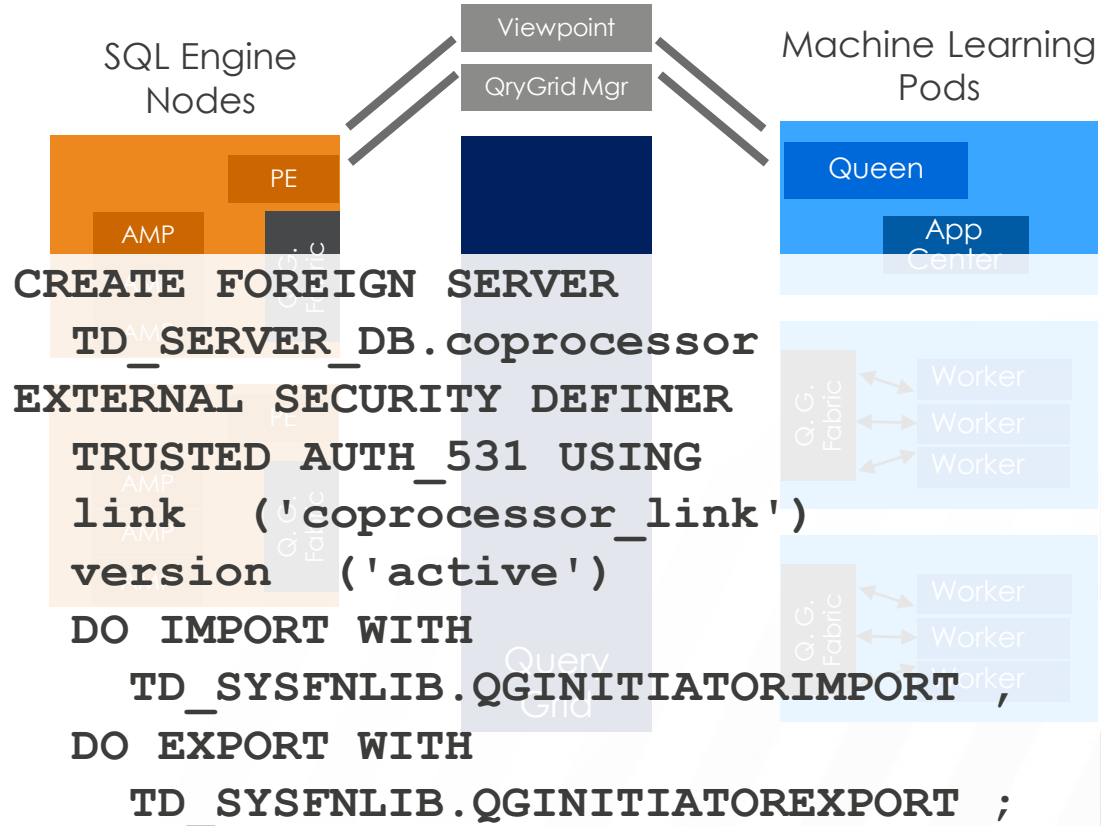
Querygrid is configured between the new MLE pods and the TD machine. MLE pods are defined as a foreign server on Teradata (SQL Engine)



SQL / ML Engine Integration

New Machine Learning (MLE) coprocessor nodes are deployed via Stacki on top of Kubernetes

Querygrid is configured between the new MLE pods and the TD machine. MLE pods are defined as a foreign server on Teradata (SQL Engine)

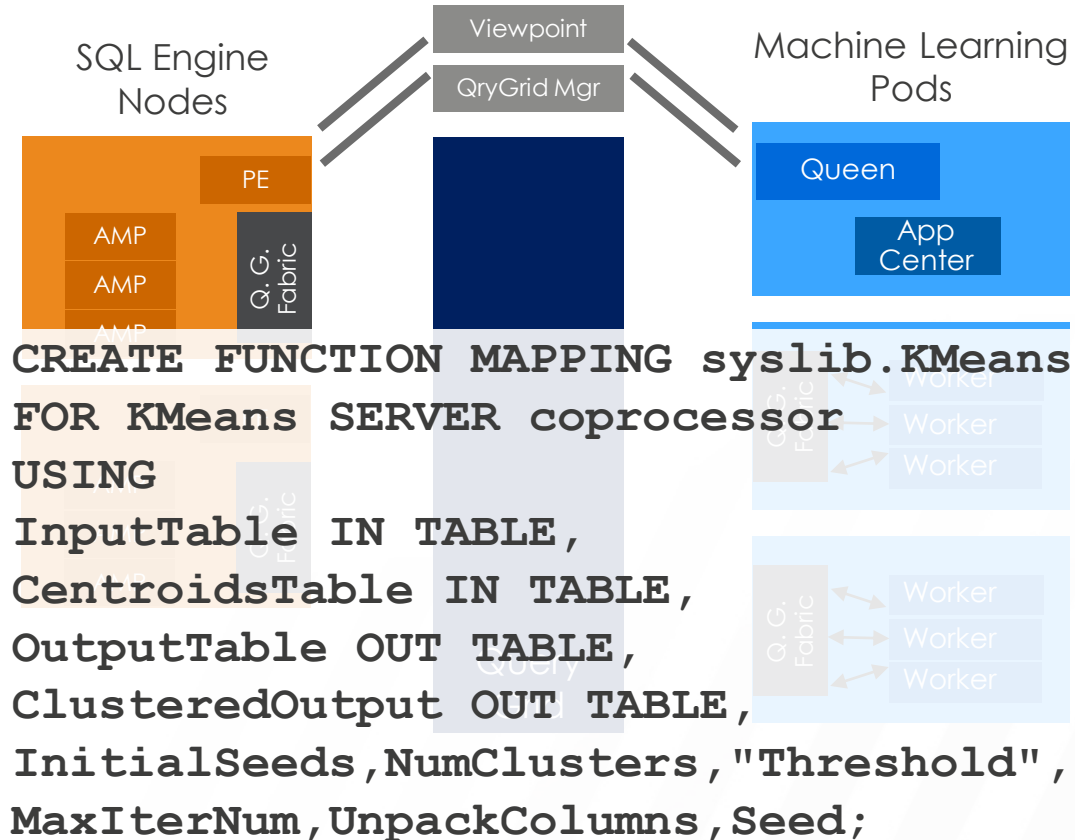


SQL / ML Engine Integration

New Machine Learning (MLE) coprocessor nodes are deployed via Stacki on top of Kubernetes

Querygrid is configured between the new MLE pods and the TD machine. MLE pods are defined as a foreign server on Teradata (SQL Engine)

New foreign functions are created on the SQL Engine nodes that map to functions on MLE



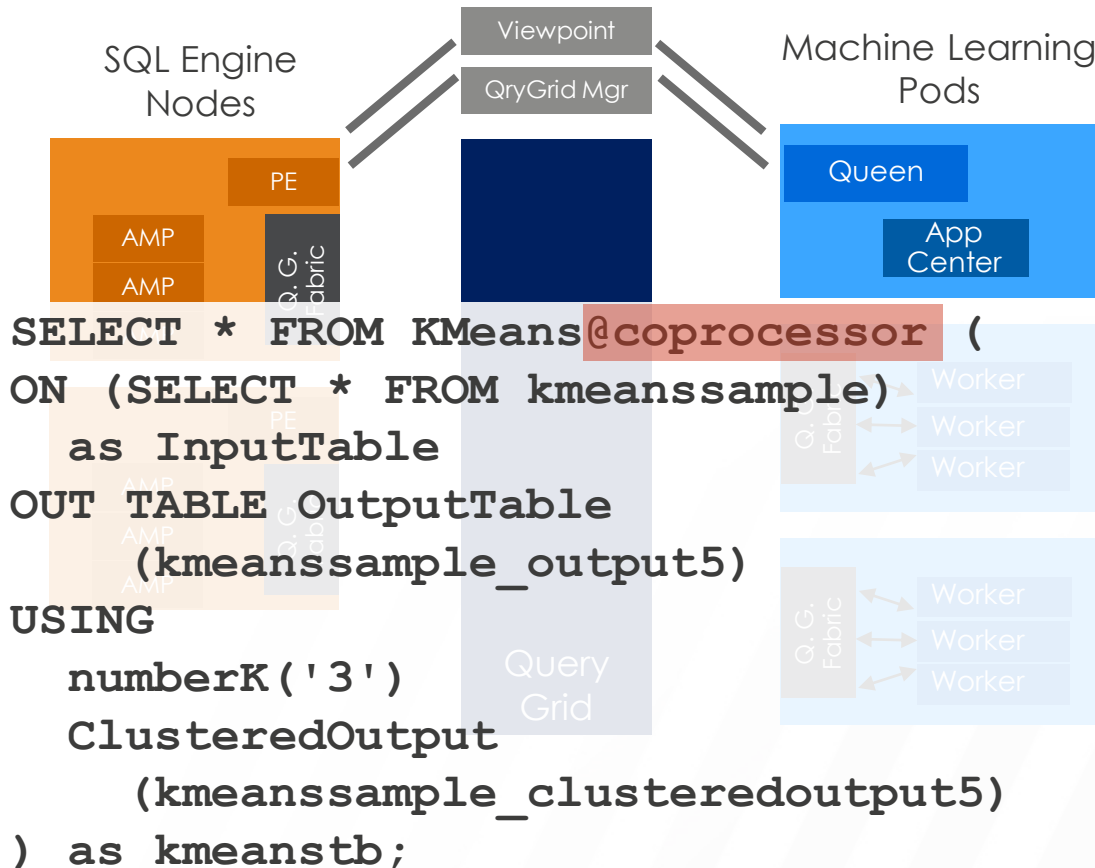
SQL / ML Engine Integration

New Machine Learning (MLE) coprocessor nodes are deployed via Stacki on top of Kubernetes

Querygrid is configured between the new MLE pods and the TD machine. MLE pods are defined as a foreign server on Teradata (SQL Engine)

New foreign functions are created on the SQL Engine nodes that map to functions on MLE

25

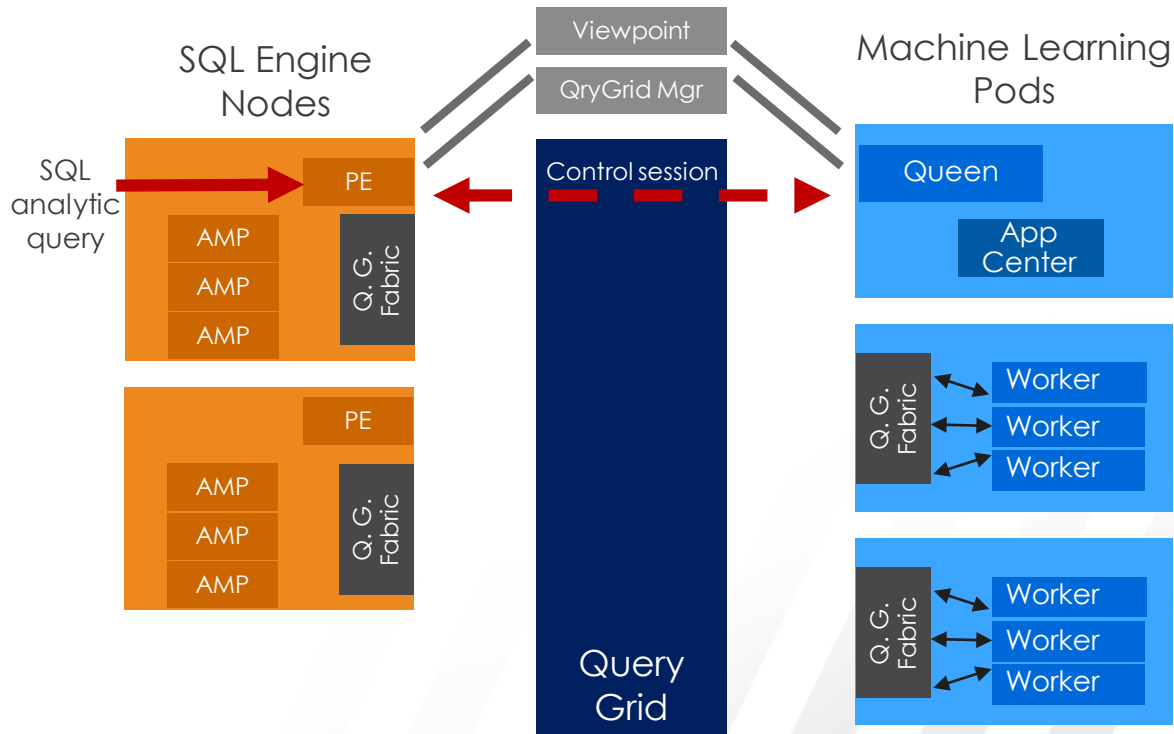


SQL / ML Engine Integration

When the foreign function is called, the “contract phase” coordinates execution between the two engines.

New with Analytic Platform is idea of the Collaborative Optimizer, where we coordinate function characteristics between the two engines.

Having tighter integration This allows us to increase performance of the contract phase and optimize function execution.



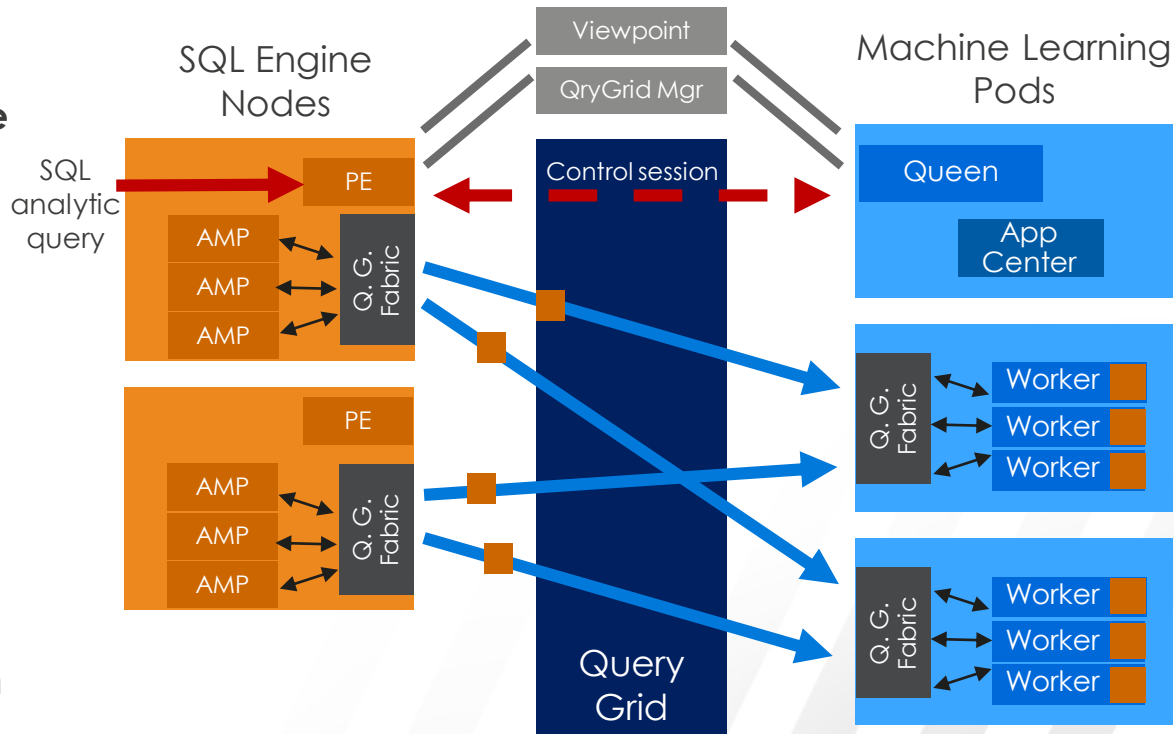
SQL / ML Engine Integration

During execution phase, the data is sent to MLE, the input data reduced by predicates that can be applied on the SQL engine side of things.

In-memory copies of the data is used for processing on MLE.

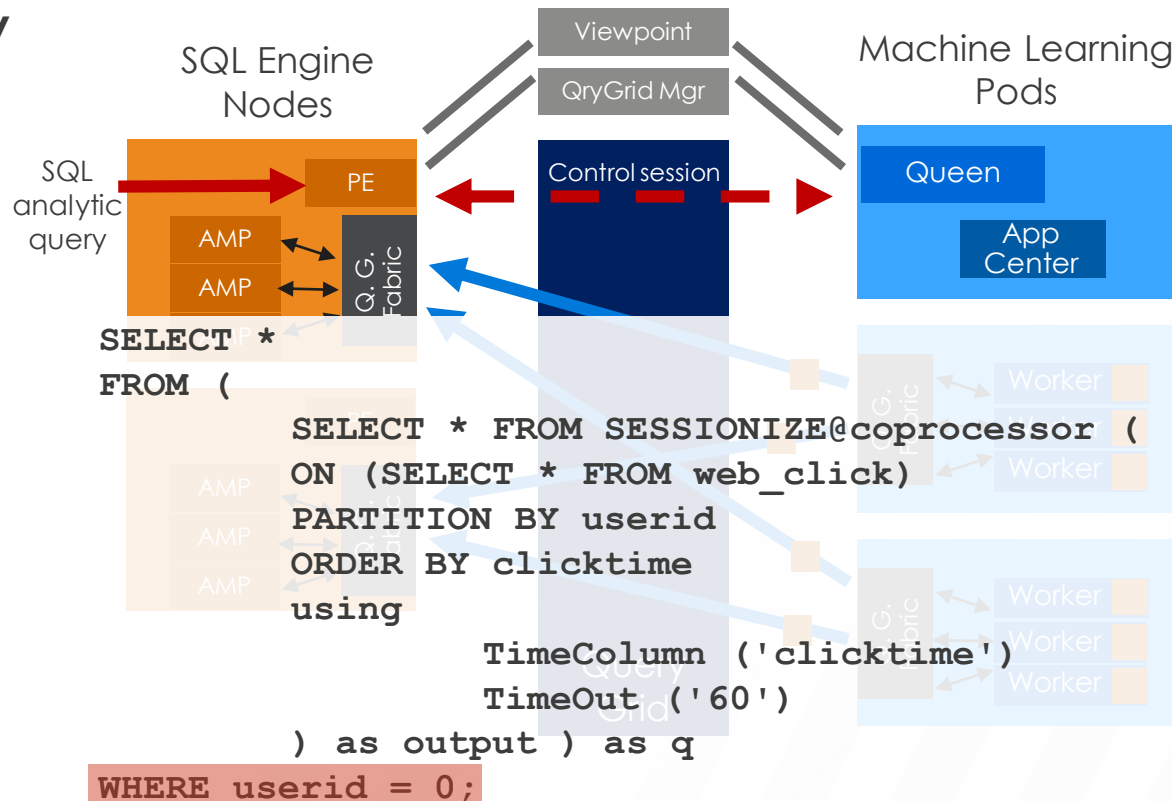
MLE workload management rules are mapped to TASM for a given request.

MLE workload management manages CPU & I/O prioritization as well as memory soft & hard limits and concurrency for a given workload.



**Collaborative optimizer
reduces data returned
across the fabric via
predicate pushdown.**

Query and Server level information is captured in Teradata, aligned with DBQL & Resusage.

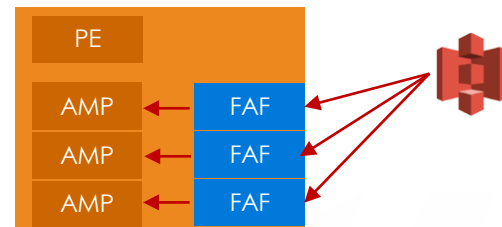


Native Object Store (17.0)

Allows access to S3 files natively through SQL Engine

- **Foreign Access Function (FAF) provides direct access to S3**
 - Integrated into SQL Engine source
 - Parallel, Scalable performance
- **Query access via foreign table**
 - Location as well as optional manifest and path pattern parameters
- **Initial release supports JSON, gzipped files and both LATIN-1 & UTF8 encoding.**

SQL Engine
Nodes

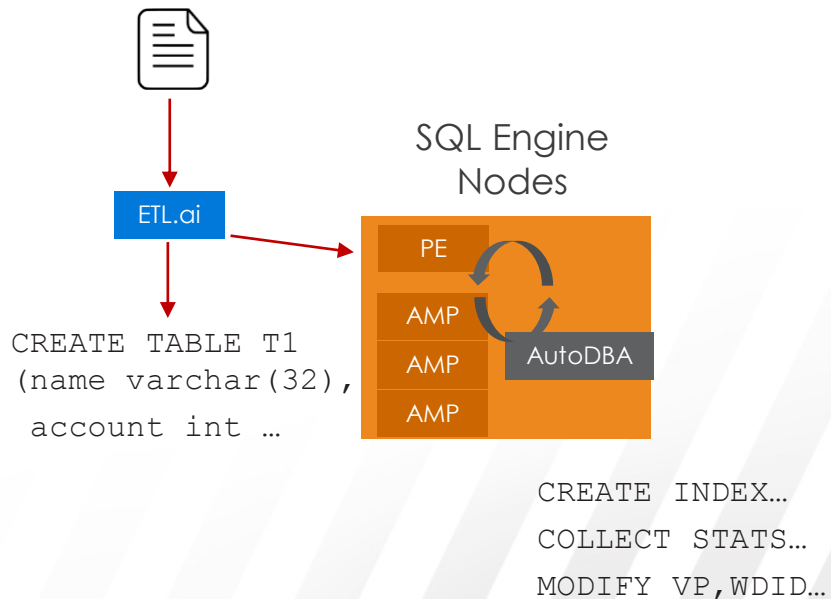


```
CREATE MULTiset FOREIGN TABLE EXTFSDB1.rivers
(
  Location VARCHAR(2048),
  Payload JSON(8388096) INLINE LENGTH 64000)
USING
(
  LOCATION    ('/s3/s3.amazonaws.com/flow-analysis/Data')
  PATHPATTERN ('$Data/$state/$river/$year/$month')
  < stuff deleted >
```

Autonomous Database (17.0)

Simplifies system management and data loading

- **ETL.ai**
 - Auto generate schema & identify keys
- **Stats.ai & Index.ai**
 - Monitor queries and access paths, tune system via physical design changes
- **SLG.ai**
 - Autotune resource allocations to align with SLGs
- **AutoDBA**
 - Background tuning service that supports Stats.ai, Index.ai and SLG.ai



Want More?

Check out these other sessions for more information about these great SQL Engine Features.

Feature	Title	Day – Time	Location
DSA	Use DSA for Business Continuity of Mission Critical Teradata Systems	Monday – 3:30pm	Jasmine B
Native Object Storage	Data Lake Analytics with S3 and Azure BLOB storage	Monday – 4:05pm	Innovation Theater 2
Analytics Platform	The NEXT... Teradata Analytics Platform Architecture	Tuesday – 8:00am	South Seas J
Analytics Platform	Are You Losing Control? Managing Workloads on the Analytics Platform!	Wednesday – 11:30am	South Seas F
SQL Engine	The Power of Three - Time Series, Geospatial and Temporal	Thursday – 11am	South Seas D

Thank You!

**Rate This Session # 1468 – SQL Engine 16.20.x
Overview**

with the Teradata Analytics
Universe Mobile App

Follow Me

LinkedIn @**gilljohn**

@**philipjbenton**

Questions/Comments

Email: **John.Gill@Teradata.com**

philip.benton@Teradata.com

Adaptive Optimizer

- **Description**

- Next iteration of dynamic query planning leveraging IPE infrastructure
- Executing query fragments of a larger request and leverage results or statistics as dynamic feedback to plan the reminder of the query

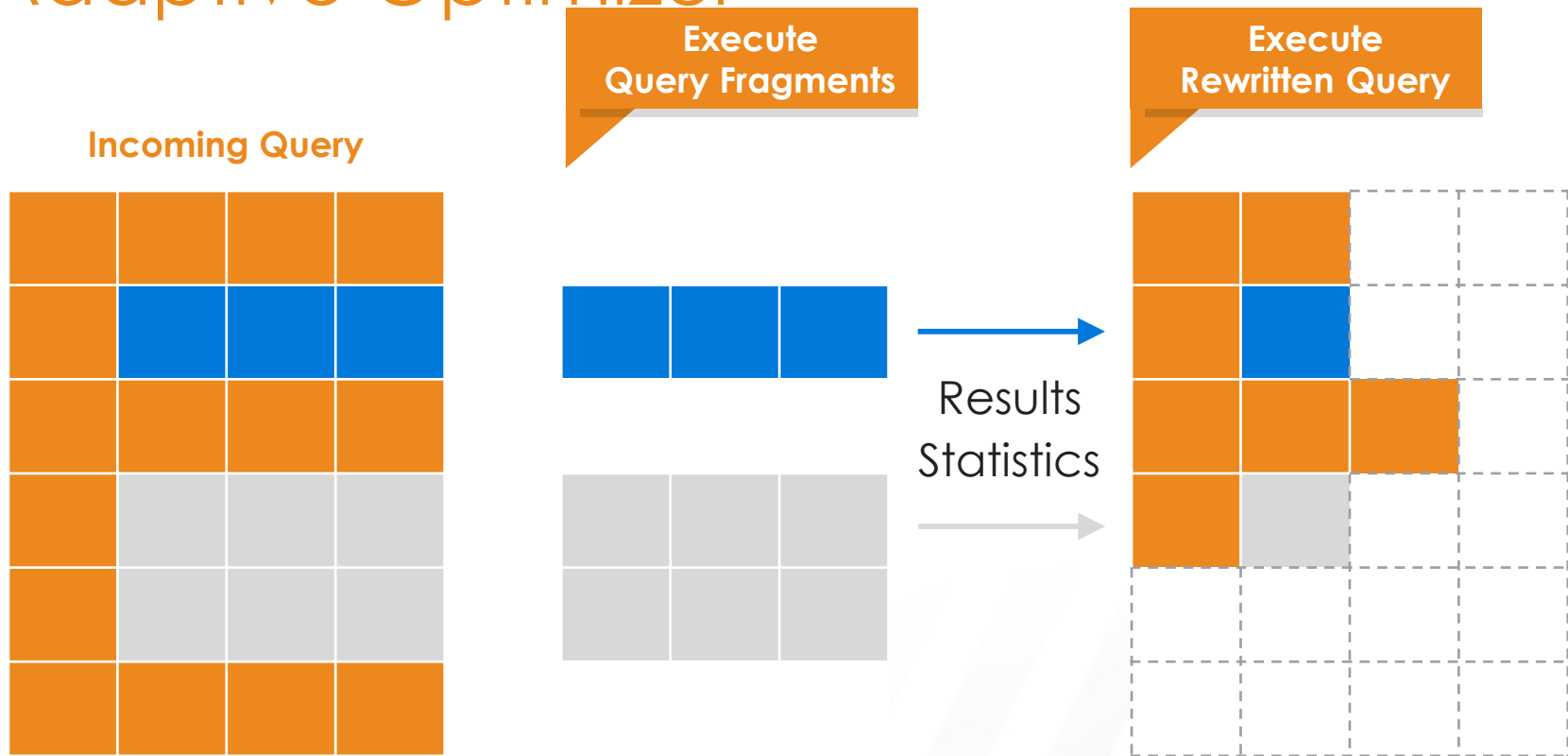
- **Benefits**

- Enables more query optimizations, including partition elimination, access path selection, join elimination, join index rewrites, predicate simplification, improved cost estimation, and more
- Results in a more optimal plan

- **Considerations**

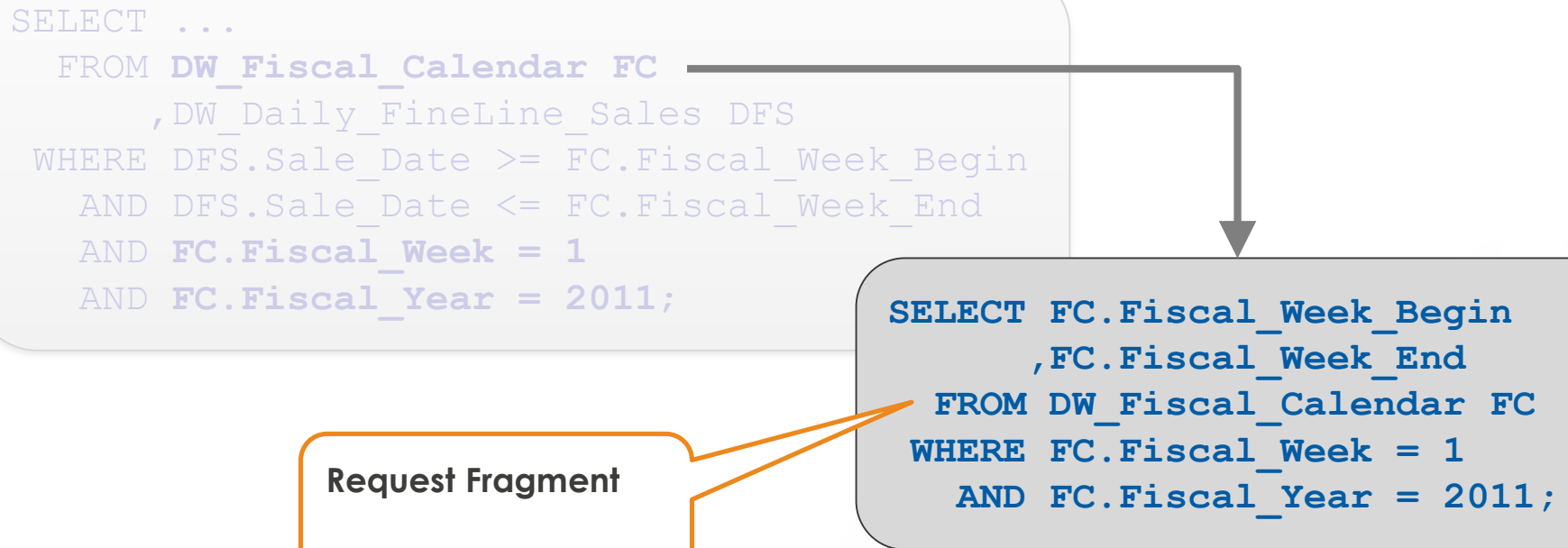
- Out-of-the-box improvements
- Eligibility requirements: >1 min runtime, runtime $< 10\%$ of stat. plan estimates
- Not eligible: MSR, recursive queries

Adaptive Optimizer



Single-Row Relation

```
SELECT ...  
  FROM DW_Fiscal_Calendar FC  
        ,DW_Daily_FineLine_Sales DFS  
WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin  
      AND DFS.Sale_Date <= FC.Fiscal_Week_End  
      AND FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2011;
```



Request Fragment

Single-row relation

```
SELECT FC.Fiscal_Week_Begin  
      ,FC.Fiscal_Week_End  
FROM DW_Fiscal_Calendar FC  
WHERE FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2011;
```

Single-Row Relation

```
SELECT ...  
  FROM DW_Fiscal_Calendar FC  
        ,DW_Daily_FineLine_Sales DFS  
WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin  
      AND DFS.Sale_Date <= FC.Fiscal_Week_End  
      AND FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2011;
```

Rewritten Query

After applying result
feedback

No large Product Join
Partition elimination

```
SELECT FC.Fiscal_Week_Begin  
      ,FC.Fiscal_Week_End  
FROM DW_Fiscal_Calendar FC  
WHERE FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2011;
```

```
SELECT ...  
  FROM DW_Daily_FineLine_Sales DFS  
WHERE DFS.Sale_Date >= '2012-01-01'  
      AND DFS.Sale_Date <= '2012-01-07';
```

Non-correlated Subquery with Small Result Set

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE)/  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
      AND Store.S_STOREKEY IN  
      (SELECT S_STOREKEY  
       FROM STORE  
       WHERE S_STATUS IN  
            ('OPEN',  
             'UNDER EXT REPAIR',  
             'UNDER INT REPAIR'))  
      AND OrderTbl.O_ORDERDATE BETWEEN  
            1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

Original Query

**OrderTbl - MLPPI table partitioned by
OrderDate and StoreKey**

Non-correlated Subquery with Small Result Set

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN  
  (SELECT S_STOREKEY  
   FROM STORE  
   WHERE S_STATUS IN  
     ('OPEN',  
      'UNDER EXT REPAIR',  
      'UNDER INT REPAIR'))  
AND OrderTbl.O_ORDERDATE BETWEEN  
  1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

↓

```
SELECT S_STOREKEY  
FROM STORE  
WHERE S_STATUS IN  
  ('OPEN',  
   'UNDER EXT REPAIR',  
   'UNDER INT REPAIR');
```

Request Fragment

Subquery

Non-correlated Subquery with Small Result Set

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN  
  (SELECT S_STOREKEY  
   FROM STORE  
   WHERE S_STATUS IN  
     ('OPEN',  
      'UNDER EXT REPAIR',  
      'UNDER INT REPAIR'))  
AND OrderTbl.O_ORDERDATE BETWEEN  
  1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

```
SELECT S_STOREKEY  
FROM STORE  
WHERE S_STATUS IN  
  ('OPEN',  
   'UNDER EXT REPAIR',  
   'UNDER INT REPAIR');
```

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN (100, 200, 300)  
AND OrderTbl.O_ORDERDATE BETWEEN  
  1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

Rewritten Query

After applying result feedback

Non-correlated Subquery with Small Result Set

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN  
  (SELECT S_STOREKEY  
   FROM STORE  
   WHERE S_STATUS IN  
     ('OPEN',  
      'UNDER EXT REPAIR',  
      'UNDER INT REPAIR'))  
AND OrderTbl.O_ORDERDATE BETWEEN
```

Rewritten Query

After transitive closure

Partition Elimination

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN (100, 200, 300)  
AND OrderTbl.O_ORDERDATE BETWEEN  
  1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

```
SELECT Store.S_STATUS,  
       sum(OrderTbl.O_TOTALPRICE) /  
       count(distinct OrderTbl.O_ORDERKEY)  
FROM Store, OrderTbl  
WHERE Store.S_STOREKEY=OrderTbl.O_STOREKEY  
AND Store.S_STOREKEY IN (100, 200, 300)  
AND OrderTbl.S_StoreKey IN (100, 200, 300)  
AND OrderTbl.O_ORDERDATE BETWEEN  
  1030301 AND 1030331  
GROUP BY Store.S_STATUS;
```

Statistics Feedback for Foreign Tables

```
CREATE FOREIGN SERVER MyHadoopServer
  USING server('153.64.96.249'),
        port('9083'),
        hosttype('hadoop')
        templeton_port('1880')
DO IMPORT WITH HarnessTblOpImp,
EXPORT WITH HarnessTblOpExp;
```

```
SELECT customer.last_name, CAST(HTbl.price
AS DECIMAL(8,2)) AS 'Price', HTbl.mileage
FROM Sales.Sales_hdr@MyHadoopServer AS HTbl
     ,customer
WHERE HTbl.make   = customer.users_make
     AND HTbl.model = customer.users_model;
```

Collecting light statistics as
Sales_hdr table is being spooled

- Row count
- Number of unique values
- High mode frequency

Update statistics information and
plan the join with Customer

Iterative Optimization

```
SELECT t2.ca_custkey ....
FROM (SELECT ca.ca_custkey
      ,t1.ca_acctkey
      ,t1.ncusts
      ,COUNT(ca_custkey)
        OVER (PARTITION BY t1.ca_acctkey ...) totcusts
      FROM (SELECT ca_acctkey
        ,COUNT(DISTINCT ca_custkey) as ncusts
        FROM tempchurn tc
        ,custaccounts ca
        WHERE tc.custkey = ca_custkey
        GROUP BY 1
        HAVING ncusts >= 2
      ) AS t1
      ,custaccounts ca
      WHERE ca.ca_acctkey = t1.ca_acctkey
    ) as t2
LEFT OUTER JOIN ordertbl ot
  ON   t2.ca_custkey = ot.o_custkey
     AND o_ORDERDATE < '2003-04-01'
     AND o_ORDERDATE >= DATE '2003-04-01' - INTERVAL '90' DAY
```

Statistics feedback
captured and used to
plan the rest of the
request

Join Index Enhancements:

Improve Rewrite of Aggregate Expressions

- **New enhancements that can increase the JI usage for certain types of queries**
 - Decompose SUM of the addition and/or subtraction of a set of non-nullable fields/expressions into individual SUMs, e.g.,
$$\text{SUM}(A+B) \rightarrow \text{SUM}(A) + \text{SUM}(B) \rightarrow \text{AJI.SumOfA} + \text{AJI.SumOfB}$$
 - Factor out constant or single-value field from the SUM of a multiplication expression, e.g.,
$$\text{SUM}(C*100) \rightarrow \text{SUM}(C)*100 \rightarrow \text{AJI.SumOfC} * 100$$
 - Lift the CAST from a CAST expression of SUM and apply that on the pre-computed SUM in AJI for compatible data types, e.g.,
$$\text{SUM}(\text{CAST}(\text{IntField AS BIGINT})) \rightarrow \text{CAST}(\text{AJI.SumOfIntField AS BIGINT})$$

Join Index Enhancements:

Partial Rewrite Coverage of Exist Expressions

- Removes the restriction that a partial covering join index cannot be used for query with correlated EXISTS.

```
CREATE JOIN INDEX StoreSalesJI AS
  SELECT S.store_id, SS.sale_date, SS.sale_amt , SS.item_sku,
         S.rowid AS StoreRID
  FROM Stores S INNER JOIN Sales SS ON S.store_id = SS.store_id;
```

```
SELECT S.store_id, SS.sale_date, SS.sale_amt
  FROM Stores S INNER JOIN Sales SS ON S.store_id = SS.store_id
    and EXISTS (sel 1 from Campaign C WHERE S.district_id = C.district_id);
```



```
SELECT S1.store_id, SS.sale_date, SS.sale_amt
  From StoreSalesJI
    INNER JOIN Stores ON StoreRID=Stores.rowid
```


Join Index Enhancements:

Partial Rewrite Coverage of Outer Join Expressions

- Remove the restriction of not using a join index that partially covers the outer table of a left outer join

```
CREATE JOIN INDEX StoreSalesJI AS  
  SELECT S.store_id, SS.sale_date, SS.sale_amt , SS.item_sku,  
         S.rowid AS StoreRID  
  FROM Stores S INNER JOIN Sales SS ON S.store_id = SS.store_id;
```

```
SELECT S1.store_id, SS.sale_date, SS.sale_amt  
FROM Stores S INNER JOIN Sales SS ON S.store_id = SS.store_id  
LEFT JOIN Campaign C ON S.district_id = C.district_id;
```



```
SELECT S1.store_id, SS.sale_date, SS.sale_amt  
From StoreSalesJI  
INNER JOIN Stores ON StoreRID=Stores.rowid
```

Aggregate Join Index Enhancements (16.20 Update 1)

- **Compressed JI with outer joins**

- 16.20 Join Index optimization allowed OUTER JOINS with AJIs
- This enhancement removed restriction of OUTER JOINS on compressed AJIs

- **Enabling JOIN INDEXES COUNT(*) without GROUP BY**

- Prior to 16.2 update 1, queries that did not have same grouping column as the AJI would end up accessing base table
- With this enhancement, optimizer re-writes the queries to utilize AJI for COUNT(*) and aggregations that don't specify GROUP BY clause
- Note that COUNT(*) queries without WHERE clause will use COUNT(*) optimization against the base table.

QueryGrid – Common Remote Table Elimination

Optimize multiple remote tables with same predicates and projections referenced in the query by way of avoiding multiple fetches from the remote system.

```
SELECT ...
  FROM DW_Fiscal_Calendar@remote FC
        ,DW_Daily_FineLine_Sales_USA DFS
 WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin
        AND DFS.Sale_Date <= FC.Fiscal_Week_End
        AND FC.Fiscal_Week = 1
        AND FC.Fiscal_Year = 2017
UNION
SELECT ...
  FROM DW_Fiscal_Calendar@remote FC
        ,DW_Daily_FineLine_Sales_Germany DFS
 WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin
        AND DFS.Sale_Date <= FC.Fiscal_Week_End
        AND FC.Fiscal_Week = 1
        AND FC.Fiscal_Year = 2017;
```

QueryGrid – Common Covering Remote Table

**Common remote table
fetched with the covering
condition when multiple
references of the same
remote table occur and
conditions on one or
more references cover
different other conditions**

```
SELECT ...  
  FROM DW_Fiscal_Calendar@remote FC  
        ,DW_Daily_FineLine_Sales_USA DFS  
WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin  
      AND DFS.Sale_Date <= FC.Fiscal_Week_End  
      AND FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2017  
UNION  
SELECT ...  
  FROM DW_Fiscal_Calendar@remote FC  
        ,DW_Daily_FineLine_Sales_USA DFS  
WHERE DFS.Sale_Date >= FC.Fiscal_Week_Begin  
      AND DFS.Sale_Date <= FC.Fiscal_Week_End  
      AND FC.Fiscal_Week = 1  
      AND FC.Fiscal_Year = 2016;
```

Security (16.0)

- **Gateway API for 3rd party Security Tools to monitor traffic to/from Teradata**
- **Provide LDAP Search Mechanism (Lightweight LDAP)**
- **Force a single authentication mechanism from the TD Server (TDNEGO)**
 - Added support for JAVA & .NET (cli & ODBC were implemented in 15.10)
- **Recommended Reading**
 - Security Administration Release 16.0 (B035-1100-160K)
 - TDNEGO Single Mechanism to Log On (Orange Book: 541-0011029-A02)
 - Security: User Authentication and Directory Integration (Orange Book: 541-0004998-D02)

Packageless GSS (16.10)

- **What is it?**

- No change in functionality, moved TeraGSS code into CLI and ODBC packages.
- Prior to 16.10, TeraGSS was a separate package that needed to be installed w/ TTU
- Complications w/ multiple packages and ensuring compatibility w/ CLI, ODBC & TDICU

Kerberos Everywhere (16.20 Update 1)

- **What is it?**

- 16.20 (Update 1), we added capability for Teradata to store and forward Kerberos tokens
 - Enabled by `dbscontrol ForwardCredential` (General #120)
- Provides end to end Kerberos SSO capability from BI tools, through Teradata to rest of UDA
- Linux and Windows clients already supported Kerberos as of 16.00
- In 16.10 we added support for MacOS, HP-UX, AIX and Solaris clients.

New QOP Algorithms (16.20)

- **Provides a new set of TDGSS QoP algorithms to meet current security standards for message encryption and integrity**
 - AES in GCM, CCM, and CTR modes
 - SHA-512 message integrity codes
 - Keyed HMAC on full cipher text
- **Enables compliance with new National Institute of Standards and Technology (NIST) standards and recommendations for block cipher modes, secure hash standards, and keyed-hash message authentication codes**
- **Eliminates perceived AES-CBC vulnerability to padding oracle attacks**

New QOP Algorithms (16.20)

- Use of new QoPs can be configured to work in conjunction with current QoPs for full backward compatibility

```
<MechQop Value="Default">  
    AES-K256_GCM_PKCS5Padding_SHA512_DH-K2048  
    AES-K128_CBC_PKCS5Padding_SHA1_DH-K2048  
    AES-K192_CBC_PKCS5Padding_SHA1_DH-K2048  
    AES-K256_CBC_PKCS5Padding_SHA1_DH-K2048  
</MechQop>
```

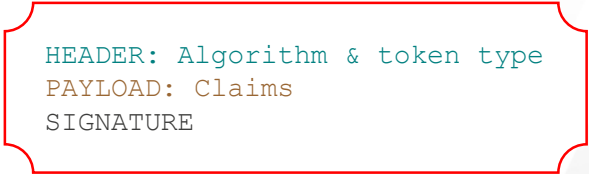
JWT Authentication (16.20)

- **Add support for a new TDGSS/TeraGSS authentication mechanism to enable single sign-on (SSO) to the database using JSON Web Token (JWT) produced upon successful user authentication to the UDA User Service.**
- **Improves integration with UDA tools/applications such as AppCenter, Query Service, QueryGrid, etc.**
- **Provides architecture to enable indirect support for other authentication protocols via the UDA User Service**

JWT Authentication (16.20)

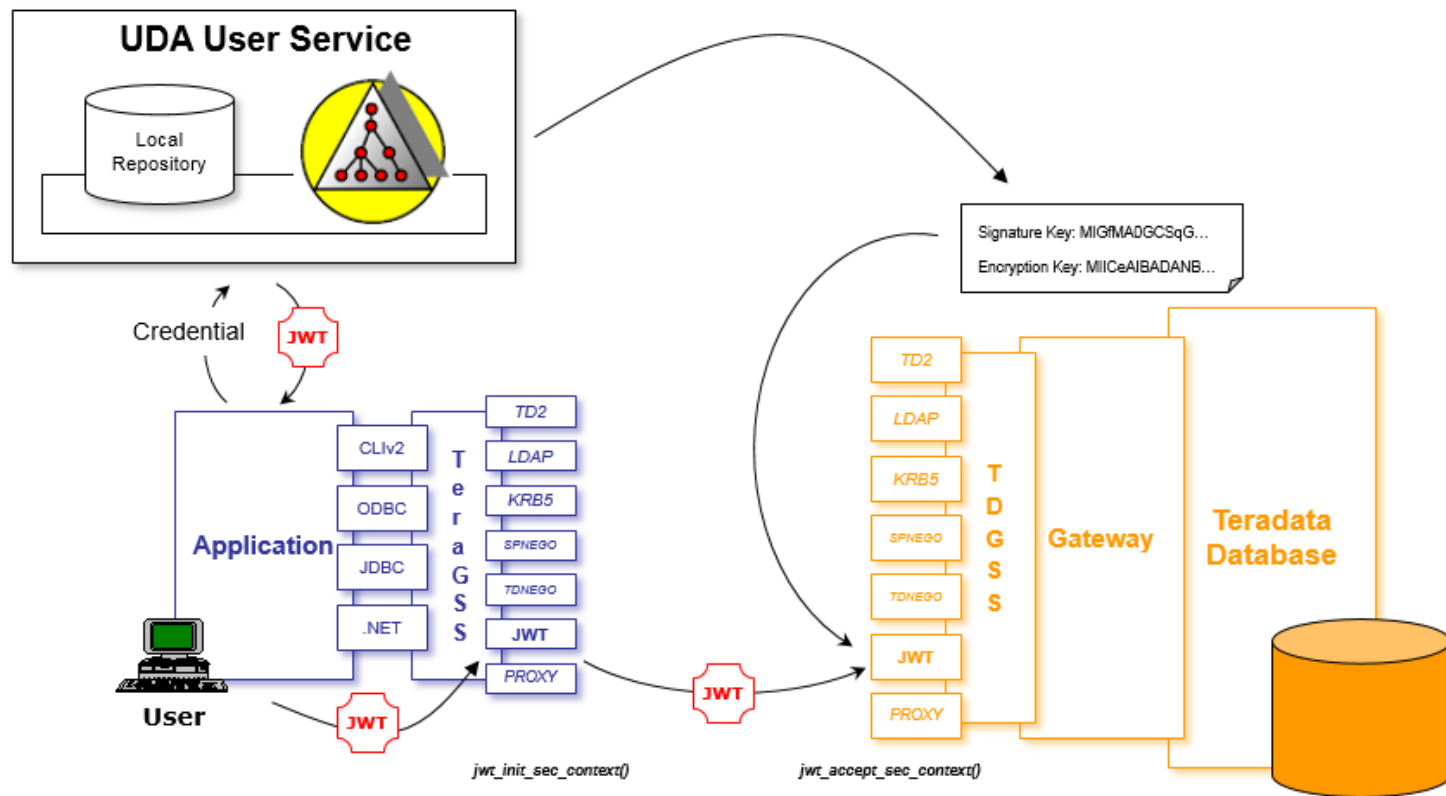
- **UDA User Service**

- UDA Shared Service
- Authentication server for UDA applications
- User authenticates by sending credentials (a username and password) and is returned a JSON Web Token (JWT)
 - JSON Web Tokens are an open, industry standard (IETF RFC 7519) method for representing claims securely between two parties
 - Signed using HMAC or RSA keys
- Supports two types of users
 - LDAP
 - Local
- Token Claims
 - **groups** - array of LDAP group membership
 - **roles** - array of application roles
 - **custid** - unique customer ID



```
HEADER: Algorithm & token type  
PAYLOAD: Claims  
SIGNATURE
```

JWT Authentication (16.20)



DBQL / Object Use Count

- **What is it?**

- Three minor enhancements to DBQL:
 - DBQL buffer size was increased from 64k to 16MB
 - Reduces express requests (AWTs & bynet traffic) and write I/O
 - Not recommended to change beyond 2MB default
 - Counts for multi-statement DML requests are now captured in a single column
 - USECOUNT logging can be flushed via QUERY LOGGING statement

DBQL Multi-Statement Request Logging

- Prior to 16.20 Update 1, there was little visibility into how many ROWS were INSERTED/UPDATED/DELETED via Multi-statement DML requests
- DBQLogtbl Extrafield52 (StmtDMLRowCount in view & 17.0) is now populated with these counts.
- Feature is enabled via default DBQL logging and supports SPs, XSPs, TPT, Mload, Fload.
- Values are stored as JSON datatype to allow for queries to parse out the values via DOT notation:

```
{"Insert":2,"Update":1,"Delete":2}
```

DBQL Multi-Statement Request Logging

- For single statement requests, only one value is populated per row.
- Consider these SQL statements against a table that already has 5 rows:

```
INSERT INTO t1 (1,2,3) ;
```

```
UPDATE t1 SET C1 = 10 ;
```

- StmtDMLRowCount would contain the following entries, each in the appropriate row for each request:

```
{ "Insert" : 1 }
```

```
{ "Update" : 6 }
```

DBQL Multi-Statement Request Logging

- Records associated with multi-statement requests will contain appropriate values for all operations the request.
- The following MSR being submitted against a table with 5 rows:

```
INSERT INTO t1 (1,2,3)  
;UPDATE t1 SET C1 = 10;
```

- StmtDMLRowCount column will be populated with the following entry in a single row:

```
{"Insert":1,"Update":6}
```

OUC Flush

- Syntax of **FLUSH QUERY LOGGING** statement supports a new option to flush the OUC cache:

```
FLUSH QUERY LOGGING WITH flush-option [...,flush-option];
```

where *flush-option* can now include keyword **USECOUNT**

- **USECOUNT** is not included with the **ALLDBQL** option which is short-hand for specifying all of the DBQL-related options.
- **USECOUNT** is incorporated into the wider **ALL** option.

SHOWCOMPRESS (16.20 Update 1)

- We introduced SHOWBLOCKS and SHOWWHERE in 15.0 & 15.10, respectively to expose filesystem metadata via SQL vs. legacy command line tool
- SHOWCOMPRESS reports same compression information that is done via legacy tool by table & subtable
- Macros are used to create & populate the table:
 - dbc.createfsysinfotable
 - dbc.populatefsysinfotable

SHOWCOMPRESS

- First run the macro to create the reporting table

```
EXEC dbc.CreateFsysInfoTable (  
TargetDatabaseName,   Database in which target table should be created  
TargetTableName,      Tablename of the target table  
TargetTableType,      ['PERM', 'GLOBALTEMP' or 'VOLATILE']  
IsFallback,           [ 'Y' | 'N' ] Sets report table to fallback, no effect if ALWAYS FALLBACK  
Command,              [ 'SHOWBLOCKS' | 'SHOWWHERE' | 'SHOWCOMPRESS' ]  
DISPLAYOPTION,        [ 'S' | 'M' | 'L' ] depending on the type of display needed.  
MapName               Map Name  
);
```

```
EXEC dbc.createfsysinfotable  
('targetdb', 'targettable' , 'PERM',  
 'Y','SHOWCOMPRESS', 'S', 'MapName' );
```


SHOWCOMPRESS

- Then second macro loads the table w/ the report

```
EXEC dbc.PopulateFsysInfoTable(  
  InputDatabaseName,    Name of the database in which the input table resides,  
  InputTableName,       Name of the input table on which the operation will be performed  
  Command,              [ 'SHOWBLOCKS' | 'SHOWWHERE' | 'SHOWCOMPRESS' ]  
  DISPLAYOPTION,        [ 'S' | 'M' | 'L' ] depending on the type of display needed.  
  TargetDatabaseName,   Name of the Target Database of the target table.  
  TargetTableName       Name of the Target table that will receive the FERRET Info rows.  
);
```

```
EXEC dbc.populatefsysinfotable  
  ('inputdb', 'inputtable', 'SHOWCOMPRESS',  
   'S', 'targetdb', 'targettable');
```

SHOWCOMPRESS

- **SHOWCOMPRESS /S reports**

- Compression state for each table & subtable
- Compression algorithm & level
- What MAP the table is in

TheDate	TheTime	DataBaseName	TableName	TableID	TableIDTAI	TLA_BLCOption
10/1/2018	11:51:21	EDW	CUSTOMER	00-00-49-0C-00-00	1,024	DEFAULT
10/1/2018	11:51:18	EDW	CUSTOMER	00-00-49-0C-00-00	2,048	DEFAULT

System_BLCOption	TLA_BLCALG	System_BLCALG	TLA_BLCLevel	System_BLCLevel	MapName
MANUAL	DEFAULT	ZLIB	DEFAULT	1	TD_MAP1
MANUAL	DEFAULT	ZLIB	DEFAULT	1	TD_MAP1

SHOWCOMPRESS

- **SHOWCOMPRESS /L reports**
 - Everything in /S report
 - Actual physical space consumed
 - Exact compression ratio (vs. estimate in showblocks)

TableIDTAI	CompressionState	CompressionAlgorithm	CompressionLevel	ExactCompRatio	ExactPctofBlocks	ExactPctofData	ExactUsedGB
-1	COMPRESSED	IPPZLIB	1	53.59	96.02	100.01	-8
-1	DISQUALIFIED	N/A	N/A	0	3.94	-1.77	0.14
-1	UNCOMPRESSED	N/A	N/A	0	0.03	-0.01	0
0	UNCOMPRESSED	N/A	N/A	0	100	100	0
1,024	COMPRESSED	IPPZLIB	1	53.68	99.99	100	-4
1,024	DISQUALIFIED	N/A	N/A	0	0	0	0
2,048	COMPRESSED	IPPZLIB	1	53.52	92.79	100	-4
2,048	DISQUALIFIED	N/A	N/A	0	7.2	-3.55	0.14

AVRO DATAFILE type support

- **What is it?**

- Self-describing AVRO files can now be loaded directly into Teradata and accessed similar to other CDTs
- No need to convert data from native AVRO type to JSON or relational database columns to store and access data
- Ability to use entire suite of Dot Notation functionality on AVRO data files
- Additional functionality to query, validate, convert and describe AVRO data files
- AVRO Schema can either be associated with a column or each instance

- **Recommended Reading**

- Teradata DATASET Data Type (16.0 Teradata Manual: B035-1198-160K)

AVRO DATAFILE type support

- **Avro requires each instance of data to be accompanied by a schema which conforms to a certain structure and describes the data**
- **Teradata supports two modes for handling/storing these schemas:**
 - **Column-level**
 - Use CREATE AVRO SCHEMA to register SCHEMA in DATA DICTIONARY
 - Then define an Avro DATASET COLUMN that references a known Avro SCHEMA.
 - **Instance-level**
 - The Dataset type with storage format Avro may be used without a column-level schema, meaning that each instance of the type must be composed of schema + data
- **Column-level SCHEMAs offer performance and space advantages**
- **Instance-level SCHEMAs offer data flexibility that is being inserted**

AVRO DATAFILE Type Support

- **Registering & Administration of DATASET SCHEMAS**

- New CREATE, DROP, HELP, SHOW SCHEMA DDL
- New WITH SCHEMA ACCESS RIGHT to be able to get READ access to a SCHEMA

```
CREATE Avro SCHEMA employeeSchema AS '{  
  "type": "array",  
  "items": {  
    "type": "record",  
    "name": "record",  
    "fields": [{"name": "name", "type": "string"}, {"name": "company", "type": "string"}]}  
}';
```

- **New dictionary tables to store information on registered SCHEMAS**

- DBC.SchemaInfo (via DBC.DatasetSchemaInfoV) returns information on registered SCHEMAS
- DBC.DatasetSchemaDependenciesV can be used to identify dependencies between registered SCHEMAS and DATASET columns TABLES with DATASET columns

AVRO DATAFILE Type Support

- **New Methods are available to support the new DATASET type**
 - Retrieve SCHEMA or all/part of the AVRO instance payload
 - Retrieve metadata from AVRO instances
 - Convert instances to JSON
 - Validate the AVRO instance

Name	Type	Description
getSchema()	Method	Retrieves the schema of any instance of the DATASET data type and returns its text representation
getRawData()	Method	Retrieves the raw data of any DATASET data type instance, returning as a non-LOB type
getRawDataLob()	Method	Retrieves the raw data of any DATASET data type instance and returns it as a LOB type
getSchemaSize()	Method	Retrieves the size of the schema for any instance of a DATASET data type
getRawDataSize()	Method	Retrieves the size of the raw data for any instance of the DATASET data type
toJSON()	Method	Converts any instance of the DATASET data type into a JSON text instance
AvroProject()	Method	Allows users to read specified portions of an Avro instance, without having to read the entire instance
AvroProjectToJSON()	Method	Identical to the AvroProject method, but returns a JSON-encoded Avro value instead of an Avro instance
Validate()	Method	Reports whether any Dataset type instance is valid or not

AVRO DATAFILE Type Support

- **New Functions and table operators to support the new DATASET type**
 - Compare SCHEMAs, retrieve DATASET KEY information
 - Convert CSV records to JSON & AVRO
 - Convert relational table to DATASET data type & vice-versa

Name	Type	Description
SchemaMatch()	Function	Allows CLOBs or JSON types used to represent an Avro schema to be compared for compatibility.
SchemaEqual()	Function	Allows CLOBs or JSON types used to represent an Avro schema to be compared for equality.
CreateDATASET()	Function	Allows users to create DATASET data type instances composed of Avro data when the schema and data are NOT included in the data payload.
AVRO_CHECK()	Function	Allows a user to pass in Avro data in our transform format to check it for validity
Dataset_Keys()	Table Operator	Like JSON_Keys, allows the database to provide a list of all queryable keys of a DATASET data type instance or a DATASET data type schema which is specified as a CHAR/VARCHAR/CLOB/JSON
Dataset_Table()	Table Operator	Takes a DATASET instance and creates a temporary table based on a subset (or all) of the data contained within.
AvroContainerSplit()	Table Operator	Splits a BLOB in the Avro Object Container File format into a table of individual Avro Dataset objects
CSV_TO_JSON()	Table Operator	Allows CSV data to be converted into JSON data
CSV_TO_AVRO()	Table Operator	Allows CSV data to be converted into Avro data
DATASET_PUBLISH()	Table Operator	Allows any row of data in a relational table to be composed into a DATASET data type.

AVRO DATAFILE Type Support

- **Converting relational table to Avro DATASET type**

```
CREATE table employee (name varchar(32), company varchar(32));
CREATE table employee_avro (
    id int,
    avroFile DATASET(1000) STORAGE FORMAT Avro WITH SCHEMA employeeSchema);
ins into employee ('phil','teradata');
ins into employee ('tom','teradata');
```

- **Using getSchema() to capture retrieve SCHEMA from Avro DATASET instance**

```
SELECT avroSchema.getSchema()
FROM
DATASET_PUBLISH(
    ON ( SELECT * FROM employee
        where name = 'phil')
    ) AS avroSchema;
```

```
{"type": "array",
 "items": {
   "type": "record",
   "name": "record",
   "fields": [{ "name": "name", "type": "string" },
               { "name": "company", "type": "string" } ] }
```

AVRO DATAFILE Type Support

- Using DATASET_PUBLISH to convert to Avro DATASET type

```
INSERT INTO employee_avro SELECT 1, avroInstance.* FROM DATASET_PUBLISH(
    ON (SELECT name, company FROM employee) ) AS avroInstance;
```

- **Normal SELECT retrieves data in Avro binary format**

```
SELECT avroFile FROM employee avro;
```

```
980000000000000000000000000000001062F00000009800000000000000000020C  
05B010000E00709070C252359000000000000000000000000000000000000000000000  
010001000000002000000000000000001532000100000000000000F00B  
9621B9A6B5877250000
```

- Using `toJSON()` to retrieve in readable format

```
SEL avroFile.toJSON() FROM employee avro;
```

```
[{"name": "tom", "company": "teradata"}]  
[{"name": "phil", "company": "teradata"}]
```

CSV DATAFILE Type Support

- Second storage format option to the Dataset data type
- Supports existing Dataset Schemas, both column-level (stored in Dictionary) and embedded (stored with the CSV data).
- Supports existing Dot Notation that is available for JSON and Dataset types.
- CSV schemas are optional and describe field and record delimiters and field names. This is different from Avro schemas, which are used to describe the structure of the data

```
CREATE CSV SCHEMA myCSVSchema AS
'{
    "field_delimeter" : "#",
    "record_delimeter" : ";"
}';
```

```
CREATE TABLE t1
( pkey INTEGER,
  csvCol DATASET STORAGE FORMAT CSV
WITH SCHEMA myCSVSchema
);
```

CSV – Single/Multi-Record Options

- **Insert a CSV value with three records into a table**

```
INSERT INTO t1 VALUES(0,  
  "Item ID#Item Name#Quantity#Price;55#bicycle#10#199.99;65#soccer  
ball#2#24.99");
```

- **Select a CSV value from a table**

```
SELECT csvCol FROM t1;
```

```
"Item ID#Item Name#Quantity#Price;55#bicycle#10#199.99;65#soccer  
ball#2#24.99");
```

- **A multi-record CSV file may be split into and stored as individual CSV values, one per record, via the CSV table operator, CSVSplit.**

CSV Methods, Functions, Table Operators and External Routine Support

Methods, Functions, Table Operators

getSchema()

GetRawData

getRawDataLob

getSchemaSize

getRawDataSize

Validate

header

numRecords

Dataset_Keys

CreateDATASET

DATASET_PUBLISH

DATASET_TABLE

- Support the CSV storage format as input/output parameters in the following external routines:
 - C UDF/XSP
 - C Aggregate UDF
 - C Table functions and operators
 - Java UDF/XSP
 - Java Aggregate UDF
 - Java table functions and operators
- All existing Dataset FNC functions are supported on the CSV storage format (e.g., FNC_GetDatasetInfo, FNC_GetDatasetSchema, etc.).
- FNC functions are Teradata Database-provided functions that users can call within their external routines.

AVRO DATAFILE type support

- **What is it?**

- Self-describing AVRO files can now be loaded directly into Teradata and accessed similar to other CDTs
- No need to convert data from native AVRO type to JSON or relational database columns to store and access data
- Ability to use entire suite of Dot Notation functionality on AVRO data files
- Additional functionality to query, validate, convert and describe AVRO data files
- AVRO Schema can either be associated with a column or each instance

- **Recommended Reading**

- Teradata DATASET Data Type (16.0 Teradata Manual: B035-1198-160K)

AVRO DATAFILE type support

- **Avro requires each instance of data to be accompanied by a schema which conforms to a certain structure and describes the data**
- **Teradata supports two modes for handling/storing these schemas:**
 - **Column-level**
 - Use CREATE AVRO SCHEMA to register SCHEMA in DATA DICTIONARY
 - Then define an Avro DATASET COLUMN that references a known Avro SCHEMA.
 - **Instance-level**
 - The Dataset type with storage format Avro may be used without a column-level schema, meaning that each instance of the type must be composed of schema + data
- **Column-level SCHEMAs offer performance and space advantages**
- **Instance-level SCHEMAs offer data flexibility that is being inserted**

AVRO DATAFILE Type Support

- **Registering & Administration of DATASET SCHEMAS**

- New CREATE, DROP, HELP, SHOW SCHEMA DDL
- New WITH SCHEMA ACCESS RIGHT to be able to get READ access to a SCHEMA

```
CREATE Avro SCHEMA employeeSchema AS '{  
  "type": "array",  
  "items": {  
    "type": "record",  
    "name": "record",  
    "fields": [{"name": "name", "type": "string"}, {"name": "company", "type": "string"}]}  
}';
```

- **New dictionary tables to store information on registered SCHEMAS**

- DBC.SchemaInfo (via DBC.DatasetSchemaInfoV) returns information on registered SCHEMAS
- DBC.DatasetSchemaDependenciesV can be used to identify dependencies between registered SCHEMAS and DATASET columns TABLES with DATASET columns

AVRO DATAFILE Type Support

- **New Methods are available to support the new DATASET type**
 - Retrieve SCHEMA or all/part of the AVRO instance payload
 - Retrieve metadata from AVRO instances
 - Convert instances to JSON
 - Validate the AVRO instance

Name	Type	Description
getSchema()	Method	Retrieves the schema of any instance of the DATASET data type and returns its text representation
getRawData()	Method	Retrieves the raw data of any DATASET data type instance, returning as a non-LOB type
getRawDataLob()	Method	Retrieves the raw data of any DATASET data type instance and returns it as a LOB type
getSchemaSize()	Method	Retrieves the size of the schema for any instance of a DATASET data type
getRawDataSize()	Method	Retrieves the size of the raw data for any instance of the DATASET data type
toJSON()	Method	Converts any instance of the DATASET data type into a JSON text instance
AvroProject()	Method	Allows users to read specified portions of an Avro instance, without having to read the entire instance
AvroProjectToJSON()	Method	Identical to the AvroProject method, but returns a JSON-encoded Avro value instead of an Avro instance
Validate()	Method	Reports whether any Dataset type instance is valid or not

AVRO DATAFILE Type Support

- **New Functions and table operators to support the new DATASET type**
 - Compare SCHEMAs, retrieve DATASET KEY information
 - Convert CSV records to JSON & AVRO
 - Convert relational table to DATASET data type & vice-versa

Name	Type	Description
SchemaMatch()	Function	Allows CLOBs or JSON types used to represent an Avro schema to be compared for compatibility.
SchemaEqual()	Function	Allows CLOBs or JSON types used to represent an Avro schema to be compared for equality.
CreateDATASET()	Function	Allows users to create DATASET data type instances composed of Avro data when the schema and data are NOT included in the data payload.
AVRO_CHECK()	Function	Allows a user to pass in Avro data in our transform format to check it for validity
Dataset_Keys()	Table Operator	Like JSON_Keys, allows the database to provide a list of all queryable keys of a DATASET data type instance or a DATASET data type schema which is specified as a CHAR/VARCHAR/CLOB/JSON
Dataset_Table()	Table Operator	Takes a DATASET instance and creates a temporary table based on a subset (or all) of the data contained within.
AvroContainerSplit()	Table Operator	Splits a BLOB in the Avro Object Container File format into a table of individual Avro Dataset objects
CSV_TO_JSON()	Table Operator	Allows CSV data to be converted into JSON data
CSV_TO_AVRO()	Table Operator	Allows CSV data to be converted into Avro data
DATASET_PUBLISH()	Table Operator	Allows any row of data in a relational table to be composed into a DATASET data type.

AVRO DATAFILE Type Support

- Converting relational table to Avro DATASET type

```
CREATE table employee (name varchar(32), company varchar(32));
CREATE table employee_avro (
    id int,
    avroFile DATASET(1000) STORAGE FORMAT Avro WITH SCHEMA employeeSchema);
ins into employee ('phil','teradata');
ins into employee ('tom','teradata');
```

- Using getSchema() to capture retrieve SCHEMA from Avro DATASET instance

```
SELECT avroSchema.getSchema()
FROM
DATASET_PUBLISH(
    ON ( SELECT * FROM employee
        where name = 'phil')
    ) AS avroSchema;
```

```
{"type": "array",
 "items": {
   "type": "record",
   "name": "record",
   "fields": [{ "name": "name", "type": "string" },
              { "name": "company", "type": "string" } ] }
```

AVRO DATAFILE Type Support

- Using DATASET_PUBLISH to convert to Avro DATASET type

```
INSERT INTO employee_avro SELECT 1, avroInstance.* FROM DATASET_PUBLISH(
    ON (SELECT name, company FROM employee) ) AS avroInstance;
```

- **Normal SELECT retrieves data in Avro binary format**

```
SELECT avroFile FROM employee avro;
```

```
98000000000000000001062F000000980000000000000020C  
05B010000E00709070C2523590000000000000000000000  
01000100000002000000000001532000100000000000F00B  
9621B9A6B5877250000
```

- Using `toJSON()` to retrieve in readable format

```
SEL avroFile.toJSON() FROM employee avro;
```

```
[{"name": "tom", "company": "teradata"}]  
[{"name": "phil", "company": "teradata"}]
```

CSV DATAFILE Type Support

- Second storage format option to the Dataset data type
- Supports existing Dataset Schemas, both column-level (stored in Dictionary) and embedded (stored with the CSV data).
- Supports existing Dot Notation that is available for JSON and Dataset types.
- CSV schemas are optional and describe field and record delimiters and field names. This is different from Avro schemas, which are used to describe the structure of the data

```
CREATE CSV SCHEMA myCSVSchema AS
'{
    "field_delimeter" : "#",
    "record_delimeter" : ";"
}';
```

```
CREATE TABLE t1
( pkey INTEGER,
  csvCol DATASET STORAGE FORMAT CSV
WITH SCHEMA myCSVSchema
);
```

CSV – Single/Multi-Record Options

- **Insert a CSV value with three records into a table**

```
INSERT INTO t1 VALUES(0,  
  "Item ID#Item Name#Quantity#Price;55#bicycle#10#199.99;65#soccer  
ball#2#24.99");
```

- **Select a CSV value from a table**

```
SELECT csvCol FROM t1;
```

```
"Item ID#Item Name#Quantity#Price;55#bicycle#10#199.99;65#soccer  
ball#2#24.99");
```

- **A multi-record CSV file may be split into and stored as individual CSV values, one per record, via the CSV table operator, CSVSplit.**

CSV Methods, Functions, Table Operators and External Routine Support

Methods, Functions, Table Operators

getSchema()

GetRawData

getRawDataLob

getSchemaSize

getRawDataSize

Validate

header

numRecords

Dataset_Keys

CreateDATASET

DATASET_PUBLISH

DATASET_TABLE

- Support the CSV storage format as input/output parameters in the following external routines:
 - C UDF/XSP
 - C Aggregate UDF
 - C Table functions and operators
 - Java UDF/XSP
 - Java Aggregate UDF
 - Java table functions and operators
- All existing Dataset FNC functions are supported on the CSV storage format (e.g., FNC_GetDatasetInfo, FNC_GetDatasetSchema, etc.).
- FNC functions are Teradata Database-provided functions that users can call within their external routines.

PIVOT and UNPIVOT

- **PIVOT provides an easy mechanism to transform rows into columns.**
 - This feature is very useful for reporting purposes as it allows user to aggregate and rotate data so that the user can create meaningful tables that are easy to read.
- **The PIVOT operation is accomplished by a QUERY REWRITE using CASE expressions**
 - The number of CASE statements is directly related to the pivot specification i.e. the number of column values specified in the IN list of the PIVOT operator.
- **UNPIVOT introduced in 14.10 via TD_UNPIVOT table operator provided an easy mechanism to transform columns into rows**
 - Added grammar in 16.0 so that UNPIVOT grammar can be use to call TD_UNPIVOT table operator.

PIVOT

country	state	yr	qtr	sales
Canada	ON	2001	Q1	15
...
Canada	ON	2001	Q3	10
USA	AZ	2001	Q3	80
USA	CA	2001	Q3	30
Canada	ON	2001	Q4	20
USA	AZ	2001	Q4	60
USA	CA	2001	Q4	25
Canada	ON	2002	Q3	30
USA				
USA				

```
SELECT country, state, yr,  
sum(case when qtr = 'Q1' then sales end) as Q1,  
sum(case when qtr = 'Q2' then sales end) as Q2,  
sum(case when qtr = 'Q3' then sales end) as Q3  
FROM SALES GROUP BY country, state, yr  
ORDER BY yr, country, state;
```

```
SELECT *  
FROM SALES  
PIVOT(SUM(sales) FOR qtr  
      IN ('Q1' AS Q1, 'Q2' AS Q2,  
          'Q3' AS Q3, 'Q4' AS Q4)) dt  
ORDER BY yr, country, state;
```

country	state	yr	Q1	Q2	Q3	Q4
Canada	ON	2001	15	10	10	20
USA	AZ	2001	30	110	80	70
USA	CA	2001	50	50	30	25
Canada	ON	2002	5	30	30	?
USA	AZ	2002	20	70	90	?
USA	CA	2002	60	80	40	?

UNPIVOT

country	state	yr	Q1	Q2	Q3
Canada	ON	2001	15	10	10
USA	AZ	2001	30	110	80
USA	CA	2001	50	50	30
Canada	ON	2002	5	30	30
USA	AZ	2002	20	70	90
USA	CA	2002	60	80	40

```
SELECT * FROM v1
UNPIVOT(sales FOR qtr
        IN (Q1,Q2,Q3,Q4)) dt2;
```

```
CREATE VIEW v1(country,state,yr,Q1,Q2,Q3,Q4)
AS (
  SELECT country, state, yr, Q1, Q2, Q3, Q4
  FROM SALES PIVOT(SUM(sales)
    FOR qtr
      IN ('Q1' AS Q1,'Q2' AS Q2,
        'Q3' AS Q3,'Q4' AS Q4)) dt );
```

country	state	yr	qtr	sales
Canada	ON	2001	Q1	15
USA	AZ	2001	Q1	30
USA	CA	2001	Q1	50
Canada	ON	2001	Q2	10
USA	AZ	2001	Q2	110
USA	CA	2001	Q2	50
Canada	ON	2001	Q3	10
USA	AZ	2001	Q3	80
...

PIVOT then UNPIVOT

country	state	yr	qtr	sales
USA	AZ	2001	Q1	30
USA	AZ	2001	Q2	110
USA	AZ	2001	Q3	80
USA	AZ	2001	Q4	10
USA	AZ	2001	Q4	60
USA	AZ	2002	Q1	20
USA	CA	2002	Q1	60
USA	AZ	2002	Q2	70

```
SELECT * FROM v1
UNPIVOT(sales FOR qtr
        IN (Q1,Q2,Q3,Q4)) dt2;
```

```
CREATE VIEW v1(country,state,yr,Q1,Q2,Q3,Q4)
AS (
  SELECT country, state, yr, Q1, Q2, Q3, Q4
  FROM SALES PIVOT(SUM(sales)
    FOR qtr
      IN ('Q1' AS Q1,'Q2' AS Q2,
        'Q3' AS Q3,'Q4' AS Q4)) dt );
```

country	state	yr	qtr	sales
USA	AZ	2001	Q1	30
USA	AZ	2001	Q2	110
USA	AZ	2001	Q3	80
USA	AZ	2001	Q4	10
USA	AZ	2001	Q4	70
USA	AZ	2002	Q1	20
USA	CA	2002	Q1	60
USA	AZ	2002	Q2	70
USA	AZ	2002	Q3	90

PIVOT Enhancements (16.20 Update 2)

- **What is it?**

- PIVOT syntax was enhanced to simplify PIVOT queries for some common use cases.
- Additional aggregations over pivot columns is now allowed.
- IN-List expression to define the PIVOT columns is now allowed.

PIVOT Aggregation

- Prior to this feature, the following query would need to be used to aggregate all the columns

```
SELECT dt2.*, Q1 + Q2 + Q3 (named "TotalSales")  
  FROM (sel country, state, yr, qtr, sales from sales) dt1  
 PIVOT(SUM(sales) FOR qtr  
 IN ('Q1' AS Q1, 'Q2' AS Q2, 'Q3' AS Q3)) dt2 ORDER BY country, yr;
```

country	state	yr	Q1	Q2	Q3	TotalSales
Canada	ON	2001	15	10	10	35
USA	AZ	2001	30	110	80	220
USA	CA	2001	50	50	30	130
Canada	ON	2002	5	30	30	65
USA	AZ	2002	20	70	90	180
USA	CA	2002	60	80	40	180

PIVOT Aggregation

- Query is simplified with the new PIVOT WITH syntax and avoids a self-join.

```
SELECT * FROM SALES PIVOT(SUM(sales)
  FOR qtr IN ('Q1' AS Q1,'Q2' AS Q2,'Q3' AS Q3)
  WITH SUM(*) as TotalSales) dt
ORDER BY yr, country, state;
```

country	state	yr	Q1	Q2	Q3	TotalSales
Canada	ON	2001	15	10	10	35
USA	AZ	2001	30	110	80	220
USA	CA	2001	50	50	30	130
Canada	ON	2002	5	30	30	65
USA	AZ	2002	20	70	90	180
USA	CA	2002	60	80	40	180

PIVOT Aggregation

- New syntax also allows aggregations of a subset of the pivoted columns

```
SELECT * FROM sales PIVOT(SUM(SALES)
FOR qtr IN ('Q1','Q2','Q3','Q4')
    WITH SUM("'Q1'", "'Q2'") AS "1H_Sales",
    SUM("'Q3'", "'Q4'")
) DT order by country, state, yr;
```

country	state	yr	'Q1'	'Q2'	'Q3'	'Q4'	1H_Sales	SUM('Q3','Q4')
Canada	ON	2001	15	10	10	20	25	30
Canada	ON	2002	5	30	30	?	35	30
USA	AZ	2001	30	110	80	60	140	140
USA	AZ	2002	20	70	90	?	90	90
USA	CA	2001	50	50	30	25	100	55
USA	CA	2002	60	80	40	?	140	40

PIVOT IN-List

- Columns names can now be dynamically created at runtime using adaptive optimizer infrastructure with IN-List
- No longer need to explicitly define list of PIVOT column values.

```
SELECT * FROM sales PIVOT(SUM(SALES) FOR qtr IN  
(SELECT qtr FROM sales)) DT order by 1;
```

country	state	yr	Q1	Q2	Q3	Q4
Canada	ON	2001	15	10	10	20
USA	AZ	2001	30	110	80	60
USA	CA	2001	50	50	30	25
Canada	ON	2002	5	30	30	?
USA	AZ	2002	20	70	90	?
USA	CA	2002	60	80	40	?

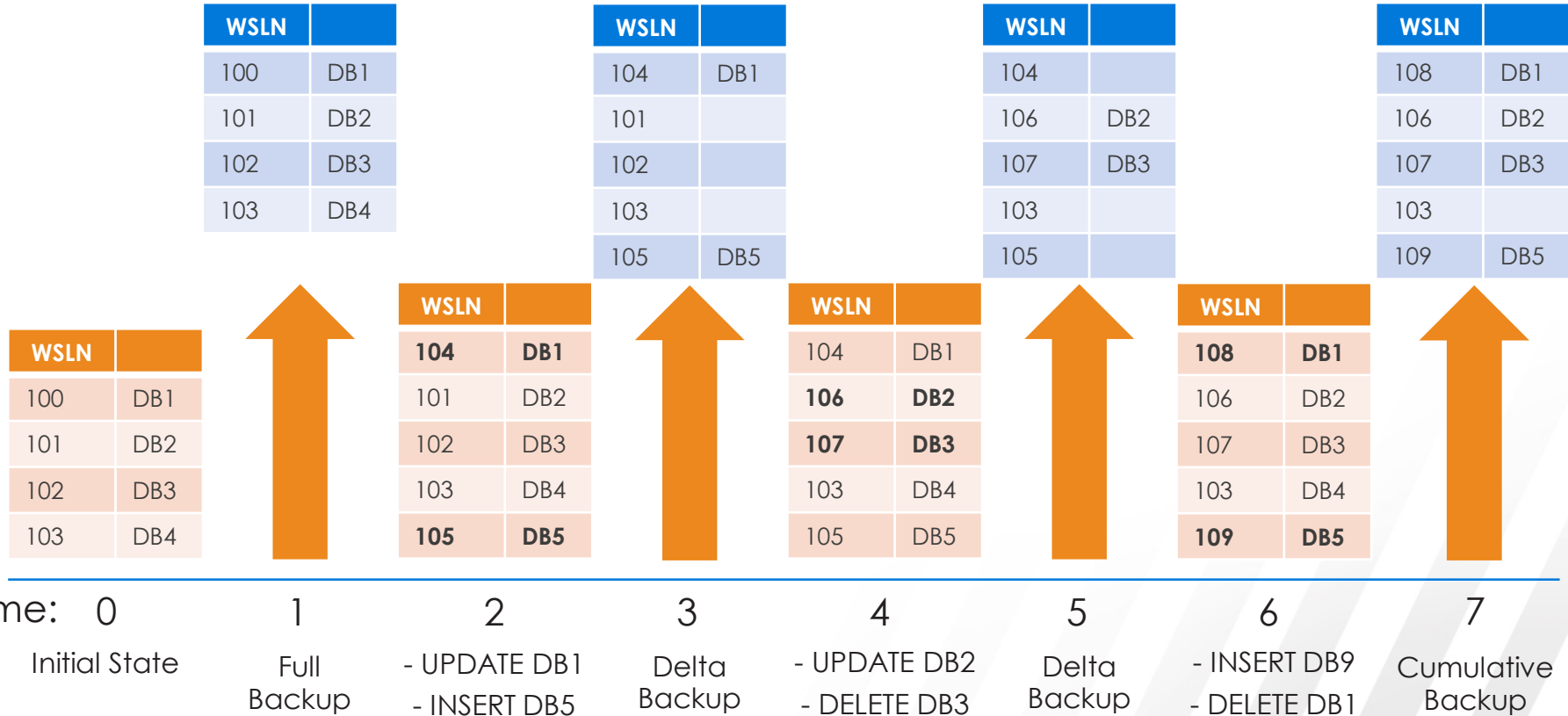
Incremental Restore (IR)

- **Completes second half of Change Block Backup (CBB) feature**
 - CBB Restore requires a rollback operation requiring a full restore
 - Incremental Restore allows rollforward operation allowing delta patches
- **To ensure source and DR stay in sync tables will be left in a read-only state**
- **When full read/write access to the data is needed, a new command is used to disable read-only access**

How Change Block Backup Works

- **Every data block on AMP has a unique number (WLSN)**
- **Full backup is required first**
 - All data blocks are backed up
 - Highest WLSN reported back to DSC as part of backup job
- **During a Change Block Backup, DSC tells DBS the highest WLSN**
 - For Delta, the WLSN is from the last backup
 - For Cumulative, the WLSN is from the last full backup
- **CBB only includes data in blocks that have WLSN higher than what DSC**
 - NOTE: All blocks are read to determine if they should be included in the backup

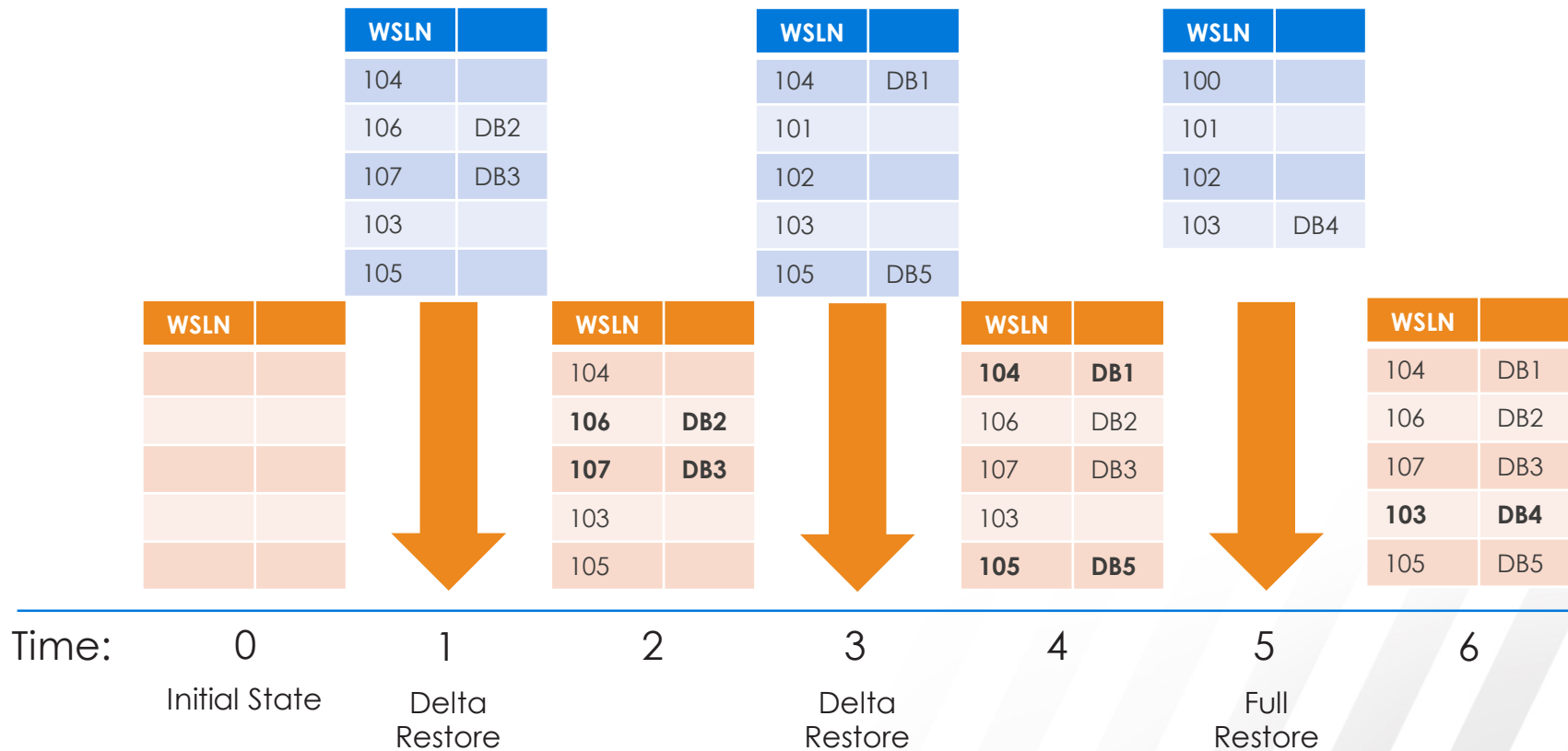
How Change Block Backup Works



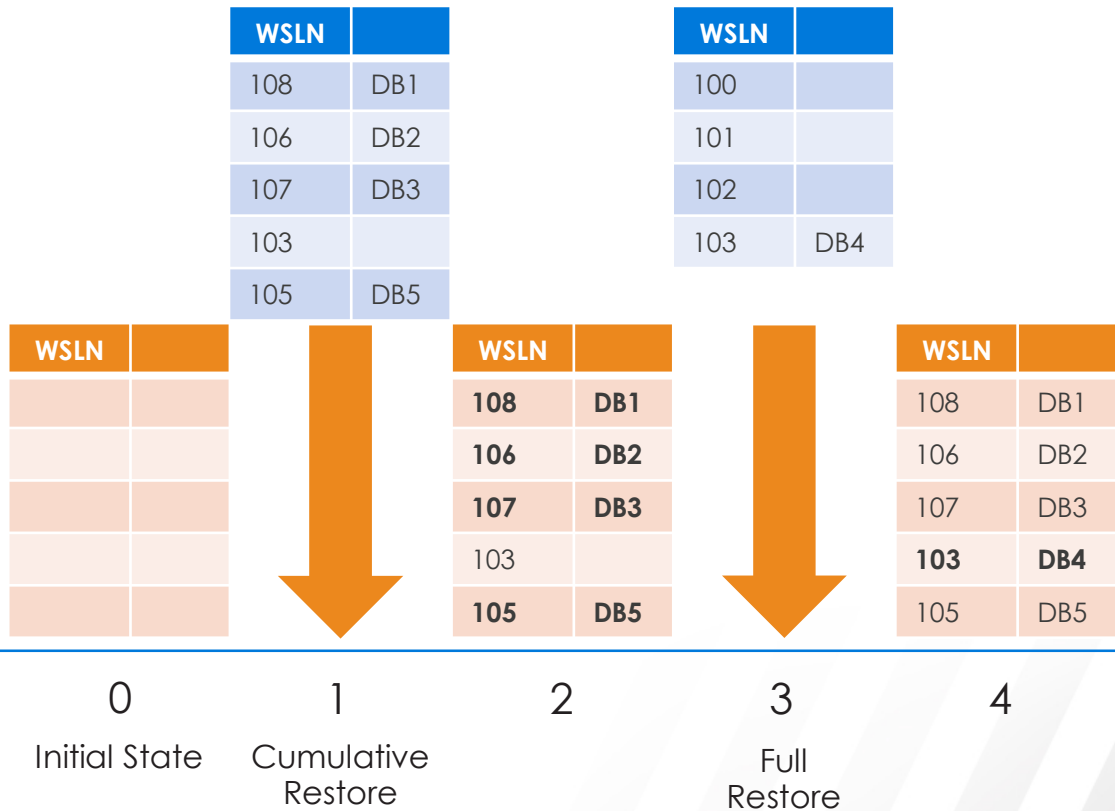
How CBB Restore Works Now

- **Only Full Restores are supported right now**
- **All the data is deleted from the table on the target system**
- **The restore will include a save set for each Change Block Backup and the last full backup**
 - The save sets will be restored in the opposite order they were archived
 - The save set from the last full backup is the last save set in the job plan

How Delta CBB Restore Works



How Cumulative CBB Restore Works



How Incremental Restore Works

Target:
Read Only State

WSLN	
500	DB1
501	DB2
502	DB3
503	DB4

WSLN	
504	DB1
101	DB2
102	DB3
103	DB4
505	DB5

WSLN	
504	DB1
506	DB2
507	DB3
503	DB4
505	DB5

Source:

WSLN	
100	DB1
101	DB2
102	DB3
103	DB4



WSLN	
104	DB1
101	DB2
102	DB3
103	DB4
105	DB5



WSLN	
104	DB1
106	DB2
107	DB3
103	DB4
105	DB5
108	DB3



Deleted Row
Log Subtable

Time: 0

Initial State

1

Full
Backup /
Restore

2

- UPDATE DB1
- INSERT DB5

3

Delta
Backup /
Restore

4

- UPDATE DB2
- DELETE DB3

5

Delta
Backup/
Restore