Tables

In this lecture

- Different table types.
- Useful system tables.

Teradata Data Dictionary

- Teradata data dictionary tables are metadata tables present in the DBC database.
- It can be used for variety of things such as checking table size, query bottleneck and database size etc.

Derived Table

- Derived tables are created, used and dropped within a query.
- These are used to store intermediate results.

Example

The following example builds a derived table `EmpSal`

```
SELECT
Emp.EmployeeNo,
Emp.FirstName,
Empsal.NetPay
FROM
Employee Emp,
(select EmployeeNo , NetPay
from Salary
where NetPay >= 75000) EmpSal
where Emp.EmployeeNo=Empsal.EmployeeNo;
```

Volatile Table

- Created, used and dropped within a user session.
- Their definition is not stored in data dictionary.
- They hold intermediate data of the query which is frequently used.

```
CREATE [SET|MULTISET] VOALTILE TABLE tablename
<table definitions>
<column definitions>
<index definitions>
ON COMMIT [DELETE|PRESERVE] ROWS
```

Example

```
CREATE VOLATILE TABLE dept_stat
(
dept_no INTEGER,
avg_salary INTEGER,
max_salary INTEGER,
min_salary INTEGER
)
PRIMARY INDEX(dept_no)
ON COMMIT PRESERVE ROWS;
```

Global Temporary Table

- The definition of Global Temporary table is stored in data dictionary and they can be used by many users/sessions.
- Data loaded into global temporary table is retained only during the session.
- You can materialize up to 2000 global temporary tables per session.

```
CREATE [SET|MULTISET] GLOBAL TEMPORARY TABLE tablename
<table definitions>
<column definitions>
<index definitions>
```

Example

```
CREATE SET GLOBAL TEMPORARY TABLE dept_stat
(
dept_no INTEGER,
avg_salary INTEGER,
max_salary INTEGER,
min_salary INTEGER
)
PRIMARY INDEX(dept_no);
```

`SET` vs `MULTISET`

- Teradata classifies the tables as `SET` or `MULTISET` tables based on how the duplicate records are handled.
- A table defined as `SET` table doesnÃ¢â‚¬â„¢t store the duplicate records.
- A `MULTISET` table can store duplicate records.
- **Tip**: If you are sure rows *will* be unique, you can set the table as multiset. This avoids unnecessary uniqueness checks, for instance, for `INSERT`.

`CREATE TABLE`

The syntax is:

```
CREATE <SET/MULTISET> TABLE <Tablename>
< Table Options>
<Column Definitions>
<Index Definitions>;
```

Syntax decoded

- `<Table Options>` specifies the physical attributes of the table such as `JOURNAL` and `FALLBACK`.
    - `JOURNAL`: Maintain data integrity in the event of component or process failure (restore to prescribed point in time).
    - `FALLBACK`: Storing a second copy of each row on a different AMP.
- `<Column Definitions>` specifies the list of columns, data types and their attributes.
- `<Index Definitions>` additional indexing options such as Primary Index.

Example

```
CREATE SET TABLE EMPLOYEE,FALLBACK
(
      EmployeeNo INTEGER,
      FirstName VARCHAR(30) ,
      LastName VARCHAR(30) ,
      DOB DATE FORMAT 'YYYY-MM-DD',
      JoinedDate DATE FORMAT 'YYYY-MM-DD',
      DepartmentNo BYTEINT
 )
UNIQUE PRIMARY INDEX ( EmployeeNo );
```

SHOW TABLE

```
 SHOW TABLE Employee;
 *** Text of DDL statement returned.
 *** Total elapsed time was 1 second.
------------------------------------------------------------
CREATE SET TABLE EMPLOYEE ,FALLBACK ,
     NO BEFORE JOURNAL,
     NO AFTER JOURNAL,
     CHECKSUM = DEFAULT,
     DEFAULT MERGEBLOCKRATIO
     (
      EmployeeNo INTEGER,
      FirstName VARCHAR(30),
      LastName VARCHAR(30),
      DOB DATE FORMAT 'YYYY-MM-DD',
      JoinedDate DATE FORMAT 'YYYY-MM-DD',
      DepartmentNo BYTEINT)
UNIQUE PRIMARY INDEX ( EmployeeNo );
```

ALTER TABLE

```
ALTER TABLE <tablename>
ADD <columnname> <column attributes>
DROP <columnname>;
```

Example:

```
ALTER TABLE employee
ADD BirthDate DATE FORMAT 'YYYY-MM-DD',
DROP DOB;
```

DROP TABLE

- `DROP TABLE <tablename>;`
- `DROP TABLE IF EXISTS <tablename>;` sadly not existant.
- From Teradata 13.10, you can use BTEQ syntax in SQLA:

```
SELECT 1 FROM dbc.TablesV
WHERe databasename = <your db>
AND TableName = '<table>';
.if activitycount = 0 then GoTo ok
DROP TABLE <table>;
.label ok
```

In general:

- `ALTER`
- `CREATE`
- `DROP`
- `MODIFY`
- `RENAME`
- `REPLACE`
- `SET ROLE`, `SET SESSION`, `SET TIME ZONE`

VIEWS

- Same DDL commands apply to **views**.
- Views are queries build on demand, they are useful to restrict data to a level of aggregation.

  Example

```
CREATE VIEW  products_more_than_3_sold AS
SELECT  productid, productname, productprice
FROM  product
WHERE  productid IN (
                        SELECT  productid
                        FROM  soldvia
                        GROUP BY productid
                        HAVING SUM(noofitems) > 3);
```

Useful system tables

Useful tables

- **dbc.tables:** Objects present in a database and their related information
- **dbc.columns:** Column informatiom of tables, views, join index & hash index etc.
- **dbc.indices:** Stores all the index related informatiom for tables, views, join index , hash index & secondary index etc.
- **dbc.errormsgs:** To error message for an error code.

Table / Database size

- **Table Size:** Table size can be determined from multiple tables for example : `dbc.allspace` & `dbc.tablesize` .
- **Database Size:** Database size can determined using `dbc.diskspace` .

Nodes and AMP info

```
/*Number of Nodes*/
SELECT COUNT(DISTINCT nodeid) FROM dbc.resusagescpu;

/*Number of Amps on each Node*/
SELECT nodeid,COUNT(DISTINCT vproc) number_of_amps
FROM dbc.ResCpuUsageByAmpView
GROUP BY nodeid;

/*Number of AMPs in the system*/
SELECT HASHAMP()+1;
```

- Number of rows in each AMP for a specific table:

```
SELECT
HASHAMP(HASHBUCKET(HASHROW(PIcolumn))), COUNT(*)
FROM tablename GROUP BY 1;
```

Skew factor

- Parallelism is good: many processes working at once. Right?
- **Dark side:** the task is completed when the slowest member has finished.
- If all AMPs have similar amount of work, task will finish sooner than if only few AMPs are working.
- For this we check the *skew factor*:

$$100 \times \left(1 - \frac{AVG(CurrentPerm)}{MAX(CurrentPerm)}\right)$$

- Recommended: Skew factor of 10 or less.

Example

```
SELECT TABLENAME,
       SUM(CURRENTPERM) CURRENTPERM,
       CAST(100*(1-(AVG(CURRENTPERM)/MAX(CURRENTPERM)))
       AS DECIMAL(5,2))
       "SKEWFACTOR(%)"
  FROM DBC.TABLESIZE
 WHERE DATABASENAME = 'MY_DB'
 GROUP BY 1
 ORDER BY 1
;
```

Exercise

Write queries to:

- Find number of tables in database.
- Find tables with specific column name.
- List all columns in specific table.
- Find a table by the name in Teradata database.
- Find tables with specific word in name.

Views

Views

- SQL statements put together.
- Recreated when queried = No extra physical space.
- **Limitations:**
  - Cannot create indices on top of them.
  - Cannot use `ORDER BY`.

Materialized Views:

- These hold a copy of the data, which takes physical space.
- Unlike Oracle(?), they are auto-refreshed in Teradata.
- Also known in Teradata as *join indices*.
- System overhead due to maintenance (but no user overhead).

When to use views

- Restrict granularity for certain users.
- More natural way to work with data for business users.
- Best practices for MicroStrategy / Teradata

View folding

```
CREATE VIEW totalsales AS
SELECT p.productname, Sum(s.noofitems) AS solditems
FROM Product p
JOIN soldvia s
ON p.productid=s.productid
GROUP BY p.productname
```

When you run:

```
SELECT productname, solditems
FROM totalsales
WHERE solditems>1
```

no reference to the view on Explain plan!

When was the view updated?

```
SELECT
databasename,
tablename, -- This will be the view name
version,  -- The times this view has been rebuilt
createtimestamp,  -- When the view first created
lastaltertimestamp  -- Last time view was rebuilt
FROM dbc.tables
WHERE
databasename = 'tutorial' -- isolates for my database only
AND
tableKind = 'V'  -- forces only views to be displayed
```