Designing IT Solutions

Data Chamber / Teradata Performance Report

Tony Bolt
Principal Consultant
Teradata Certified Master

12 Jan 2010

Table of Contents

Background

Background

Overview

Designing IT Solutions have an approach to database row level security known as the Data Chamber. Teradata was engaged to run a series of independent tests to determine the performance impact, of the Data Chamber algorithms in a Teradata environment.

This report summarises the results of those tests and provides some high level commentary on the results obtained and the Data Chamber approach in general.

The tests were carried out on the 7th and 8th Jan 2010 in the Teradata Canberra office.

Tester Background

The testing was performed by Tony Bolt. Tony has been involved with Teradata for the past 17 years and has worked directly for Teradata for 10 years. He is currently a principal consultant in the Teradata office in Canberra. He is a Teradata certified master. He works mainly in the areas of data warehousing architecture and methodology, Teradata performance and capacity planning and security integration.

Data Chamber Overview

Data Chamber adds a column to each protected table. This column, known as the security key, encodes the security level of each row. This may be based upon one or more attributes of the row, or may be user or application assigned. To enable comparable testing, all security keys in this testing were derived from attributes of the rows.

A User Security table records a similar security key for each user which specifies the access privileges for each user.

Proprietary algorithms allow the system to determine whether a given user security key dominates a given row security key. The dominance function is calculated in a view that is created on each protected table. If the user dominates then they are permitted to see the row, otherwise the row is eliminated from the view.

Testing Overview

The test database was created from the TPCD test database specification. Tables used in the Tests were:

- CUSTOMER (15m rows)
- ORDERTBL (75m rows)
- LINEITEM (300m rows)
- PARTTBL (10m rows)
- SUPPLIER (0.5m rows)

The table definitions are included at Appendix A

The Teradata system used in the tests was a 2 node 2550 data warehouse appliance.

Each test consisted of a query based on the Data Chamber views and one or two equivalent queries based on:

a. hard-coded WHERE clauses such as might be generated by an application-based security scheme[1], and/or

b. an alternative user security scheme based on views and a user security table containing explicit column values.

These alternative query formulations gave us a baseline upon which to compare the performance of the Data Chamber queries.

For example, the CUSTOMER table has a column called C_MktSegment, containing values such as "Machinery", "Furniture" and "Automobile". If a given user only has access to the automobile market segment then, in the hard coded query formulation, the query would contain a clause such as:

[1] It should be noted that in a real world situation, this approach would require the where clause to be built by the querying application, and then added to any query generated by the user. The overhead that would be required to do this has not be taken into account in any of the tests performed.

C_MktSegment = 'AutoMobile'.

In the alternative user security table there would be a row such as:

UserName	TableColumn	AllowedValue
User001	Customer.MktSegment	Automobile

This table is joined to the Customer table via the column, C_MktSegment to ensure that the user, User001, can only see rows where C_MktSegment equals the Allowed value(s), 'Automobile'.

In the Data Chamber there is also a user security table but it only contains the user name and the user's security key, such as:

UserName	SecurityKey
User001	30

The SecurityKey, 30, encodes the fact that this user, User001, can only see rows from the Customer table where C_MktSegment = 'Automobile'. By joining this table to the Customer table using the dominance function the user's access is restricted to the automobile market segment.

Test Outcomes

Conclusion

The testing undertaken showed that the Data Chamber approach works efficiently and predictably in the Teradata environment. There is a small performance overhead associated with the join to a user security table and the calculation of the dominance function. This overhead is fixed for a given table size and Teradata configuration. It is independent of the query complexity, and independent of the complexity of the security labels.

There is an additional performance cost when the data security keys exceed 19 digits associated with the way that Teradata performs arithmetic on these large numbers. This overhead, whilst larger, is again fixed and independent of the query or label complexity.

The encoding of the security keys shifts part of the cost of security processing from query execution time to the ETL process.

The space used by the addition of security keys to large data tables can be mitigated using Teradata compression.

There is a absolute limit on the size of security keys of 38 digits. This is the maximum supported integer size in Teradata. If a given security scheme requires more than 38 digit security keys then the problem has to be partitioned in some way.

General Observations and Summary

Summary

The tests undertaken compared the query performance of the Data Chamber approach, an alternative user security model and a simple WHERE clause implementation. The results are summarized in Table 1 below.

The WHERE clause option assumes that the front-end application can dynamically add clauses to the SQL to limit the rows accessible by a given user. In this approach, the row-level security is implemented in the application and requires that all access to the data be via the application. This is unlikely to be practical in a data warehousing environment. If used, there is a query preparation cost (t_{QP}) associated with determining the user's access rights and generating the necessary SQL clauses.

In the alternative security model that we tested, where restricted attribute values were explicitly held in a user security table, the query time cost increases with the number of attributes in the security scheme ($t_Q + n^e k_1$ where n is the number of attributes, e is an exponential constant (slightly greater than 1) and k_1 is a constant representing the cost of joining to the security table)

In the Data Chamber approach some of the cost of determining the security classification of a given row is off-loaded to the ETL process (t_{ETL}). In the other approaches, the cost is borne at query time for every query. There is a small but constant cost associated with joining to the user security table and calculating the dominance function (k_2). When the number of values causes the security key to exceed 19 digits, the cost of calculating the dominance function increases (k_3).

Method	Number of attributes / values *(n)*	ETL Cost	Query prep Cost	Query time cost
Explicit WHERE Clauses				
	1	0	t_{QP}	t_Q
	2	0	t_{QP}	t_Q
	3	0	t_{QP}	t_Q
	n	0	t_{QP}	t_Q
Alternative Security Model				
	1	0	0	$t_Q + k_1$
	2	0	0	$t_Q + n^e k_1$
	3	0	0	$t_Q + n^e k_1$
	n	0	0	$t_Q + n^e k_1$
Data Chamber				
	1	t_{ETL}	0	$t_Q + k_2$
	2	t_{ETL}	0	$t_Q + k_2$
	3	t_{ETL}	0	$t_Q + k_2$
	n	t_{ETL}	0	$t_Q + k_{2/3}$

Table 1 - Performance Summary

General Observations

Without discussing the proprietary aspects of the Data Chamber approach, there are several observations that can be made:

Workload shift to ETL

When a query is executed, the Data Chamber solution compares the user's security key to the data security key stored in each row of the target table using the Data Chamber dominance function. The complexity of the underlying security scheme is completely encoded in the security keys.

In a data warehousing application the encoding of the security keys would be undertaken as part of the ETL process. Each row's security attributes are encoded once and then the resulting security key is used by all queries. This approach fits well with the usual data warehousing approach: "write once – read many" It also means that the query time overhead is constant, regardless of the complexity of the access rules.

There is a small overhead in joining the base tables to the user security table and with calculating the dominance function compared to reading the base tables directly and applying simple WHERE clause filters. However, because the Data Chamber approach moves some of the overhead of dealing with a complex access rules to the ETL process, the tests showed that, as the complexity increases, eventually the Data Chamber queries outperform explicit WHERE clause queries.

Compression

The Data Chamber approach adds a single security column to each protected table. This has the potential to increase the size of each row and hence the size of the table. However, the nature of the security encoding is such that two rows with the same security classification will have the same value of their respective security keys. Thus, in most practical instances, there are a relatively small number of unique security keys in any table.

This situation is well suited to Teradata's value-based compression. By compressing the security key, the space overhead will be negligible.

Optimizer Estimates

The Teradata query optimizer is a cost-based optimizer. It makes extensive use of statistics and heuristics to estimate the number of rows that will be output from each step in the query plan.

Because the Data Chamber uses mathematical functions to compare the user security key to the data security key the optimizer is unable to accurately predict the number of rows that will be eliminated. It is therefore possible that the optimizer will generate sub-optimal plans for very complex queries. There was, however, no evidence of this behaviour in the tests.

Large keys and integer calculations

The Data Chamber encoding algorithm can generate very large security keys. In the tests, the security keys were defined as DECIMAL(38,0) which is the largest integer data type available in the Teradata system.

If the desired security scheme is very complex, the generated security keys (especially the user security key) could exceed the upper limit of the data type. In this case the simple model would have to be extended to allow for, for example, multiple user security keys.

In the tests, we observed a discontinuity in the performance of mathematical calculations on very large integers. It appears that Teradata uses a different (and more CPU intensive) method for certain calculations when the size of the numbers exceeds the maximum size of a BIGINT. This maximum is a 19 digit number: 9,223,372,036,854,775,807. In the tests we observed that the cost of calculating the Data Chamber dominance function had two values: one when the magnitude of the data security keys was less than this value and another when the magnitude was greater.

In a Data Chamber implementation the size of the security keys is related to the complexity of the underlying security model. There are two thresholds that apply:

1. If the magnitude of the data security keys is below 19 digits, the cost of the dominance function is constant and minimal. If the magnitude of the data security keys exceeds 19

digits the cost of the dominance function is again constant but higher.

2. If the magnitude of the user security keys exceeds the maximum size of a Teradata integer (i.e. 38 digits) then a partitioning scheme is required.

Detail Results

Individual Test Results

Test 1

Description. Restrict access based on one value of one attribute in one table (CUSTOMER).

Compare Data Chamber (DITS) access against explicit SQL formulation (WHERE).

(Note. The explicit SQL (WHERE) formulation represents the performance baseline. SQL such as this could be used to implement row-level security but would have to be generated in the front-end application based on the privileges of the user. In these tests the overhead of determining the user's privileges and generating the appropriate SQL are not included.)

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM dit_lib.customer_sec
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.customer
WHERE c_mktsegment = 'AUTOMOBILE'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test001	DITS	0:00:00.71	29,519	10.16	53,644,288
Test001	WHERE	0:00:00.46	25,987	5.34	1,024

Observations

1. The Data Chamber view implementation is transparent to the user.

2 The Data Chamber view includes a join to a user security table. The overhead of this join is demonstrated in this test (e.g. spool usage and CPU).

Test 2

Description. Restrict access based on one value of two attributes in one table (ORDERTBL).

Compare Data Chamber (DITS) access against explicit SQL formulation (WHERE).

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
WHERE o_orderpriority = '1-URGENT'
AND o_orderstatus = 'O'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test002	DIT	0:00:02.93	93,864	43.14	132,085,760
Test002	WHERE	0:00:01.47	88,388	20.02	1,024

Observations

As in Test 1, there is some overhead in the join required by the Data Chamber query.

Test 3

Description. Join two secured tables and restrict access based on three values of three attributes across the two tables.

Compare Data Chamber (DITS) access against explicit SQL formulation (WHERE).

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec
JOIN dit_lib.customer_sec
ON c_custkey = o_custkey
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
JOIN dit_lib.customer
   ON c_custkey = o_custkey
WHERE o_orderpriority = '1-URGENT'
   AND o_orderstatus = 'O'
   AND c_mktsegment = 'AUTOMOBILE'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test003	DIT	0:00:05.97	144,848	84.68	433,473,536
Test003	WHERE	0:00:03.35	124,073	49.54	185,729,536

Observations

Results are consistent with Tests 1 and 2.

Test 4

Description. To isolate the cost of the Data Chamber dominance function from the overhead of the required join, we repeat Test 3, but explicitly provide the value of the user security key.

Note. This is an artificial test and does not reflect the expected Data Chamber usage model.

Compare Data Chamber (DITS) dominance function against explicit SQL formulation (WHERE).

Data Chamber SQL (DITS)

Withheld

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
JOIN dit_lib.customer
   ON c_custkey = o_custkey
WHERE o_orderpriority = '1-URGENT'
   AND o_orderstatus = 'O'
   AND c_mktsegment = 'AUTOMOBILE'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test004	DIT	0:00:04.23	124,072	64.34	185,729,536
Test004	WHERE	0:00:03.36	124,072	49.74	185,729,536

Observations

With the underlying join removed from the Data Chamber query, we see that there is a small CPU and elapsed time overhead in the calculation of the dominance function. The magnitude of this overhead is assumed to be dependent on the size of the tables and the Teradata configuration.

Test 5

Description. This is the first of a series of tests that compare the Data Chamber approach with an alternative security model (ALT) based on a user security table that explicitly lists users' access privileges in terms of table/column and value.

In this test, user access is based on one value of one attribute in one table (CUSTOMER).

This alternative user security table is described in the Testing Overview section above.

For this test, an alternative security view was created as follows:

```
REPLACE VIEW dit_lib.Customer_sec_alt AS
SELECT   C_CUSTKEY
     ,   C_NAME
     ,   C_ADDRESS
     ,   C_NATIONKEY
     ,   C_PHONE
     ,   C_ACCTBAL
     ,   C_MKTSEGMENT
     ,   C_COMMENT
     ,   SEC_KEY
FROM   CUSTOMER C
       JOIN Sec_user_lookup l
       ON     c.C_Mktsegment = l.AvailableChar
WHERE  l.tablecolumnName     =
'Customer.MktSegment'  AND      l.username  = USER
```

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.customer_sec
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.customer_sec_alt
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test005	DITS	0:00:00.72	29,519	10.3	53,644,288
Test005	ALT	0:00:00.66	29,519	9.33	53,644,288

Observations

In this test both the Data Chamber (DITS) and alternative model (ALT) employ a join to a user security table. The small difference between the Data Chamber and Alternative model is assumed to be the calculation associated with the dominance function.

Test 6

Description. This test extends the security scheme to include 2 values across two attribute on one table (ORDERTBL). The test compares the Data Chamber with the alternative security model introduced in Test 5

For this test, an alternative security view was created as follows:

```
REPLACE VIEW DIT_LIB.ORDERTBL_sec_alt AS
SELECT  O_ORDERKEY
    ,  O_CUSTKEY
    ,  O_ORDERSTATUS
    ,  O_TOTALPRICE
    ,  O_ORDERDATE
    ,  O_ORDERPRIORITY
    ,  O_CLERK
    ,  O_SHIPPRIORITY
    ,  O_COMMENT
    ,  sec_key
FROM   dit_db.ORDERTBL OT
      JOIN dit_db.Sec_user_lookup l1
      ON     ot.O_orderpriority = l1.AvailableChar
      JOIN dit_db.Sec_user_lookup l2
      ON     ot.O_orderstatus = l2.AvailableChar
WHERE  l1.tablecolumnNAme      =
'Ordertbl.OrderPriority'  AND
      l1.username           = USER  AND
      l2.tablecolumnNAme      = 'Ordertbl.OrderStatus'
AND
      l2.username           = USER
```

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test006	DIT	0:00:02.83	93,864	42.80	132,085,760
Test006	ALT	0:00:03.49	94,036	53.08	132,085,760

Observations

The Data Chamber query is more efficient in this case. This is assumed to be because the Data Chamber view only requires a single join to its user security table, whereas the alternative mode requires two.

We note that, in the alternative model, the complexity of the user restriction is directly reflected in the definition of the view. In the Data Chamber case, the view definitions do not change as the complexity of the restrictions increases.

Test 7

Description. In this test we extend the user restriction to include 3 values across 2 attributes on one table. We compare the Data Chamber (DITS), alternative security model (ALT) and an explicit WHERE clause (WHERE).

The alternative security view on ORDERTBL is the same as for Test 6.

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test007	DIT	0:00:03.21	98,040	47.94	263,903,744
Test007	ALT	0:00:03.86	98,212	59.02	263,903,744
Test007	WHERE	0:00:01.36	88,388	21.00	1,024

Observations

This test shows that the Data Chamber model is more efficient than the alternative model as the complexity of the user restriction increases. Both models incur join overheads when compared to the explicit SQL WHERE clause formulation.

Test 8

Description. In this test we further extend the user restriction to include 5 values across 3 attributes on one table. We compare the Data Chamber (DITS), alternative security model (ALT) and an explicit WHERE clause (WHERE).

Because we introduced a third attribute, the view for the alternative model had to be modified, as follows:

```
REPLACE VIEW DIT_LIB.ORDERTBL_sec_alt AS
SELECT  O_ORDERKEY
     ,  O_CUSTKEY
     ,  O_ORDERSTATUS
     ,  O_TOTALPRICE
     ,  O_ORDERDATE
     ,  O_ORDERPRIORITY
     ,  O_CLERK
     ,  O_SHIPPRIORITY
     ,  O_COMMENT
     ,  sec_key
FROM    DIT_DB.ORDERTBL OT
       JOIN DIT_DB.Sec_user_lookup l1
       ON     ot.O_orderpriority = l1.AvailableChar
       JOIN DIT_DB.Sec_user_lookup l2
       ON     ot.O_orderstatus = l2.AvailableChar
       JOIN DIT_DB.Sec_user_lookup l3
       ON     EXTRACT(YEAR FROM ot.O_orderdate) =
l3.AvailableNum
WHERE  l1.tablecolumnName = 'Ordertbl.OrderPriority'
AND
       l1.username = USER  AND
       l2.tablecolumnName = 'Ordertbl.OrderStatus'  AND
       l2.username = USER  AND
       l3.tablecolumnName = 'Ordertbl.OrderDate'  AND
       l3.username = USER
```

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
AND EXTRACT(YEAR FROM O_OrderDate) IN (1994,
1996)
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test008	DIT	0:00:02.85	92,568	42.05	82,510,848
Test008	ALT	0:00:20.64	4,772,887	315.62	1,262,012,928
Test008	WHERE	0:00:01.49	88,388	20.82	1,024

Observations

The performance of the Data Chamber query (DITS) remained constant but the performance of the alternative model deteriorated markedly. This deterioration was caused by a sub-optimal query plan generated by the optimizer.

While the performance of the alternative model's query could be tuned (for example, by gathering appropriate statistics), the result demonstrates that the performance of the Data Chamber model is independent of the complexity of the implemented access restriction.

Test 9

Description. This test extends Test 8 by introducing 2 joins: one to another secure table (CUSTOMER) and one to an unsecured table (LINEITEM).

The alternative security view on ORDERTBL is the same as for Test 8.

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec o
JOIN  dit_lib.customer_sec c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt o
JOIN  dit_lib.customer_sec_alt c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderke
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl o
JOIN  dit_lib.customer c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
AND EXTRACT(YEAR FROM O_OrderDate) IN (1994,
1996)
AND c_mktsegment = 'AUTOMOBILE'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test009	DIT	0:00:15.01	483,412	96.56	354,461,184
Test009	ALT	0:00:34.54	779,401	355.91	1,627,534,848
Test009	WHERE	0:00:12.51	463,219	61.25	172,788,224

Observations

This query joins three large tables. Compared to Test 8, we see that the explicit SQL (WHERE) has increased from 1.5 to 12.5secs. The Data Chamber (DITS) query has increased from 2.9 to 15secs. We note that the difference between the two queries is approximately 1.5 secs in both this test and Test 8. This suggests that the overhead of the Data Chamber processing is a small fixed cost (related to the size of the source secure tables) and not proportional to the overall cost of the query.

Test 10

Description. This test extends Test 9 by introducing 2 additional joins to unsecured tables (PARTTBL and SUPPLIER).

The alternative security view on ORDERTBL is the same as for Test 8 and 9.

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec o
JOIN  dit_lib.customer_sec c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt o
JOIN  dit_lib.customer_sec_alt c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl o
JOIN  dit_lib.customer c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
AND EXTRACT(YEAR FROM O_OrderDate) IN (1994,
```

```
1996)
AND c_mktsegment = 'AUTOMOBILE'
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test010	DIT	0:00:14.43	561,734	134.51	923,403,776
Test010	ALT	0:03:41.42	3,250,046	2,605.66	6,982,094,848
Test010	WHERE	0:00:16.24	487,039	91.67	172,788,224

Observations

In this relatively complex query, the Data Chamber query outperformed the explicit SQL query. This was due to a small difference in the respective query plans. (In a critical step, the DITS query using a 6 partition hash join where the explicit SQL query used a single partition hash join.)

We also note that the alternative security model again performed poorly, relative to the other two queries.

Test 11

Description. This test builds on Test 8 by adding an additional secure attribute. This test is against a single large table (ORDERTBL) with 4 attributes and 7 values.

The alternative security view on ORDERTBL had to be extended to account for the 4th attribute, as follows:

```
REPLACE VIEW DIT_LIB.ORDERTBL_sec_alt AS
SELECT  O_ORDERKEY
    ,  O_CUSTKEY
    ,  O_ORDERSTATUS
    ,  O_TOTALPRICE
    ,  O_ORDERDATE
    ,  O_ORDERPRIORITY
    ,  O_CLERK
    ,  O_SHIPPRIORITY
    ,  O_COMMENT
    ,  sec_key
FROM   DIT_DB.ORDERTBL OT
      JOIN DIT_DB.Sec_user_lookup l1
      ON     ot.O_orderpriority = l1.AvailableChar
      JOIN DIT_DB.Sec_user_lookup l2
      ON     ot.O_orderstatus = l2.AvailableChar
      JOIN DIT_DB.Sec_user_lookup l3
      ON     EXTRACT(YEAR FROM ot.O_orderdate) =
l3.AvailableNum
      JOIN DIT_DB.Sec_user_lookup l4
      ON     ot.O_ShipPriority = l4.AvailableNum
WHERE  l1.tablecolumnNAme = 'Ordertbl.OrderPriority'
AND
      l1.username = USER  AND
      l2.tablecolumnName = 'Ordertbl.OrderStatus'  AND
      l2.username = USER  AND
      l3.tablecolumnName = 'Ordertbl.OrderDate'  AND
      l3.username = USER  AND
      l4.tablecolumnName = 'Ordertbl.ShipPriority'  AND
      l4.username = USER
```

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
AND EXTRACT(YEAR FROM O_OrderDate) IN (1994,
1996)
AND O_ShipPriority IN (5,7);
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test011a	DIT	0:00:31.44	91,056	492.40	13,943,808
Test011a	ALT	0:00:09.93	208,931	140.46	1,502,422,016
Test011a	WHERE	0:00:01.38	88,388	18.83	1,024
Test011b	DIT	0:00:02.55	93,143	38.79	13,943,808
Test011b	ALT	0:00:09.92	211,237	141.89	1,502,422,016
Test011b	WHERE	0:00:01.38	90,475	18.93	1,024

Observations

In the first execution of this test, labeled Test011a in the table above, we observed an unexpected increase in the CPU and elapsed time of the Data Chamber query.

Analysis of the security keys used by the Data Chamber showed that their magnitude had exceeded the maximum value supported by the BIGINT data type. The data type used for the security keys was not BIGINT but DECIMAL(38,0). It appears that Teradata uses a more CPU intensive mechanism when performing arithmetic on very large integers.

The second execution of this test, labeled Test011b in the table, provides support for this conclusion. In this test we were able to reduce the magnitude of the security keys to less than that supported by BIGINT. With this change the elapsed time and CPU consumption returned to expected values.

Tests 12 & 13

Description. Following the results of Test 11, Test 12 and 13 were designed to explore the impact of very large security key values in the user security table.

Using the setup for Test 11b, the size of the security key in the user security table was increased to greater than INTEGER (Test 12) and greater than BIGINT (Test13).

The setup and queries were otherwise identical to those for Test 11b

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test012	DIT	0:00:02.59	93,143	39.42	13,943,808
Test012	ALT	0:00:09.39	211,294	140.87	1,502,422,016
Test012	WHERE	0:00:01.33	90,475	18.68	1,024

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test013	DIT	0:00:02.63	93,143	40.18	13,943,808
Test013	ALT	0:00:09.51	211,323	139.91	1,502,422,016
Test013	WHERE	0:00:01.29	90,475	18.7	1,024

Observation

The performance of the Data Chamber (DITS) queries in both tests were very similar to those in Test 11b. It appears therefore that the performance of the dominance function calculation is not impacted by the magnitude of the user security keys. This is an important characteristic because, in a real world implementation it is expected that the user security keys will be larger than the data security keys.

Test 14

Description. This test used a combination of very large security keys in both the data and user security tables.

It used the setup for Test 13 (user security keys greater than BIGINT) and modified the data security keys to be also greater than BIGINT. Results similar to Test 11a were expected.

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test014	DIT	0:00:31.02	93,143	486.96	13,943,808
Test014	ALT	0:00:09.78	211,273	140.99	1,502,422,016
Test014	WHERE	0:00:01.33	90,475	18.74	1,024

Observations

As expected, with very large data security keys, the performance of the Data Chamber queries returned to the same level as in Test 11a

Test 15

Description. This test combined the very large security key values of Test 14, with the multiple table join of Test 10. The aim of this test was to see whether the additional overhead of the very large security keys was a constant or dependent on the complexity of the overall query.

The setup and queries combined aspects of Test 10 (5-way join) and Test 11 (4 attributes and 7 values on ORDERTBL). The alternative security view on ORDERTBL was the same as that used in Test 11

Data Chamber SQL (DITS)

```
SEL COUNT(*)
FROM  dit_lib.ordertbl_sec o
JOIN  dit_lib.customer_sec c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
```

Alternative SQL (ALT)

```
SEL COUNT(*)
FROM dit_lib.ordertbl_sec_alt o
JOIN  dit_lib.customer_sec_alt c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
```

Explicit SQL (WHERE)

```
SEL COUNT(*)
FROM dit_lib.ordertbl o
JOIN  dit_lib.customer c
ON o.o_custkey = c.c_custkey
JOIN  dit_db.lineitem l
ON l.l_orderkey = o.O_orderkey
JOIN dit_db.parttbl p
ON p.P_partkey = l.l_partkey
```

```
JOIN dit_db.supplier s
ON s.s_suppkey = l.l_suppkey
WHERE o_orderpriority IN ('1-URGENT','2-HIGH')
AND O_orderstatus = 'O'
AND EXTRACT(YEAR FROM O_OrderDate) IN (1994,
1996)
AND c_mktsegment = 'AUTOMOBILE'
AND o.o_shippriority IN (5,7);
```

		Elapsed Time	TotalIOCount	AMPCPUTime	SpoolUsage
Test015	DIT	0:00:41.43	357,379	526.88	845,196,800
Test015	ALT	0:00:40.70	816,645	337.86	1,743,141,376
Test015	WHERE	0:00:11.07	288,285	37.47	73,759,744

Observations

As expected, the performance overhead related to the very large data security keys is a fixed amount based on the size of the secured tables and the system configuration. In the case of our test environment it represents about 30 elapsed secs and 480 CPU secs. It is independent of the number of attributes, values and the complexity of the queries.

We observe that there are two tiers of overhead. If the data security keys stay within the bounds of BIGINT arithmetic, the overhead on our sample database and test configuration was only about 1.5secs elapsed time.

Appendix 1 – Test Database Definition

CUSTOMER

15m rows

```
CREATE SET TABLE DIT_DB.CUSTOMER ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     C_CUSTKEY INTEGER NOT NULL,
     C_NAME VARCHAR(25) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     C_ADDRESS VARCHAR(40) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     C_NATIONKEY INTEGER NOT NULL,
     C_PHONE CHAR(15) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     C_ACCTBAL DECIMAL(15,2) NOT NULL,
     C_MKTSEGMENT CHAR(10) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     C_COMMENT VARCHAR(117) CHARACTER SET LATIN CASESPECIFIC
NOT NULL,
     Sec_Key DECIMAL(38,0))
UNIQUE PRIMARY INDEX ( C_CUSTKEY );
```

ORDERTBL

75m rows

```
CREATE SET TABLE DIT_DB.ORDERTBL ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     O_ORDERKEY INTEGER NOT NULL,
     O_CUSTKEY INTEGER NOT NULL,
     O_ORDERSTATUS CHAR(1) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     O_TOTALPRICE DECIMAL(15,2) NOT NULL,
     O_ORDERDATE DATE FORMAT 'yyyy-mm-dd' NOT NULL,
     O_ORDERPRIORITY CHAR(15) CHARACTER SET LATIN CASESPECIFIC
NOT NULL,
     O_CLERK CHAR(15) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     O_SHIPPRIORITY INTEGER NOT NULL,
     O_COMMENT VARCHAR(79) CHARACTER SET LATIN CASESPECIFIC
NOT NULL,
     Sec_Key DECIMAL(38,0))
UNIQUE PRIMARY INDEX ( O_ORDERKEY );
```

LINEITEM

300m rows

```
CREATE SET TABLE DIT_DB.Lineitem ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     L_ORDERKEY INTEGER NOT NULL,
     L_PARTKEY INTEGER NOT NULL,
     L_SUPPKEY INTEGER NOT NULL,
     L_LINENUMBER INTEGER NOT NULL,
     L_QUANTITY DECIMAL(15,2) NOT NULL,
     L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
     L_DISCOUNT DECIMAL(15,2) NOT NULL,
     L_TAX DECIMAL(15,2) NOT NULL,
     L_RETURNFLAG CHAR(1) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     L_LINESTATUS CHAR(1) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     L_SHIPDATE DATE FORMAT 'yyyy-mm-dd' NOT NULL,
     L_COMMITDATE DATE FORMAT 'yyyy-mm-dd' NOT NULL,
     L_RECEIPTDATE DATE FORMAT 'yyyy-mm-dd' NOT NULL,
     L_SHIPINSTRUCT CHAR(25) CHARACTER SET LATIN CASESPECIFIC
NOT NULL,
     L_SHIPMODE CHAR(10) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     L_COMMENT VARCHAR(44) CHARACTER SET LATIN CASESPECIFIC NOT
NULL)
PRIMARY INDEX ( L_ORDERKEY );
```

PARTTBL

10m rows

```
CREATE SET TABLE DIT_DB.PARTTBL ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     P_PARTKEY INTEGER NOT NULL,
     P_NAME VARCHAR(55) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     P_MFGR CHAR(25) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     P_BRAND CHAR(10) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     P_TYPE VARCHAR(25) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     P_SIZE INTEGER NOT NULL,
     P_CONTAINER CHAR(10) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     P_RETAILPRICE DECIMAL(15,2) NOT NULL,
     P_COMMENT VARCHAR(23) CHARACTER SET LATIN CASESPECIFIC
NOT NULL)
UNIQUE PRIMARY INDEX ( P_PARTKEY );
```

SUPPLIER

500k rows

```
CREATE SET TABLE DIT_DB.SUPPLIER ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     S_SUPPKEY INTEGER NOT NULL,
     S_NAME CHAR(25) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     S_ADDRESS VARCHAR(40) CHARACTER SET LATIN CASESPECIFIC NOT
NULL,
     S_NATIONKEY INTEGER NOT NULL,
     S_PHONE CHAR(15) CHARACTER SET LATIN CASESPECIFIC NOT NULL,
     S_ACCTBAL DECIMAL(15,2) NOT NULL,
     S_COMMENT VARCHAR(101) CHARACTER SET LATIN CASESPECIFIC
NOT NULL)
UNIQUE PRIMARY INDEX ( S_SUPPKEY );
```