Teiid - Scalable Information Integration

1

Teiid Caching Guide

7.0.0

Overview

Teiid provides three capabilities for caching data: materialized views, result set caching, and code table caching. These can be used to significantly improve performance in many situations.

Following is a summary comparison of these three data caching options.

	Materialized Views	Result Set Caching	Code Table Caching
Best For	Complex transformations	Complex frequently issued user queries	Key based lookups on small, frequently accessed tables
Data Change Rate	Static	Static	Static
Data Size	Any size	Small-Medium	Small
Cache Defined By	External scripts	Query execution	Scalar function (lookup) execution
Cache Scope	VDB	VDB or session	VDB within Query service
Cache Key		Based upon the user query	VDB + table + key column + value column
Cache Removal Policy	None (discretion of administrator)	Least-recently used removed first	None (discretion of administrator)
Cache Configuration	Model properties	DQP Configuration	DQP Configuration
Cache Administration	External to Teiid		
Access Method	Query against materialized view	Query with caching enabled	Query containing scalar “lookup” function
User Override?	Yes	Yes	No – must explicitly specify whether to use

Materialized Views

MetaMatrix supports Materialized Views. These are Relational virtual tables and views ('virtual groups') for which the transformations are pre-computed and the results are stored in an external database. When queries are issued against these virtual groups through the MetaMatrix Server, the cached results are used. This saves the cost of accessing all the underlying data sources and re-computing the virtual group transforms each time a query is executed against the group.

This strategy is appropriate when the underlying data does not change rapidly, or when it is acceptable to retrieve data that is "stale" within some period of time, or when it is preferred for end-user queries to access staged data rather than placing additional query load on operational sources. MetaMatrix provides a utility to refresh materialized tables. This utility uses the MetaMatrix batched update functionality.

2.1. Support Summary

1. Caching of relational table or view records (pre-computing all transformations)
2. Model-based definition of virtual groups to cache
3. User ability to override use of materialized view cache for specific queries
4. Administrative utility to initially load and refresh cached data

2.2. User Interaction

When client applications issue queries against a Relational table or view that has been defined as a materialized view, the MetaMatrix query engine automatically routes that query to obtain the results from the cache database.

Individual queries may override the use of materialized views by specifying OPTION NOCACHE on the query. This parameter must specify one or more virtual groups to override (separated by commas, spaces optional). If no virtual groups are specified, then it is the same as if the override option is not specified. Note that only virtual groups specified in the user or transformation query (in the FROM clause) can be specified to be overridden. If there are materialized virtual groups below that level (referenced in transformations), then the materialized view for those tables will be used.

Examples:

SELECT * from vg1, vg2, vg3 WHERE … OPTION NOCACHE vg1, vg3

SELECT * from vg1, vg2, vg3 WHERE … OPTION NOCACHE

The second query is equivalent to:

SELECT * from vg1, vg2, vg3 WHERE OPTION NOCACHE vg1, vg2, vg3

The materialization override option may be specified in virtual group transformation definitions. In that way, transformations can specify to always use real-time data obtained directly from a source. The use of caching and non-caching can be mixed in transformation definitions, just as with user queries.

2.3. Cache Configuration

Materialized View Definition

Materialized views are defined in the MetaBase Modeler by setting the materialized property on a table or view in a relational model. Setting this property's value to true (the default is false) allows the data generated for this virtual table to be treated as a materialized view.

The Name In Source property for this table determines the name of the physical Materialized Views tables. If left blank the default naming scheme of MV10000001 is used, otherwise the name provided is used.

Virtual Database Definition

Materialized views are relational tables (or views) in one or more models that have their materialized property set to true. When a set of virtual relational models containing materialized views is bundled into a virtual database (VDB) in the MetaBase Modeler, a physical relational model is automatically created and put into that VDB to represent the physical cache. The physical model is given the well-known name Materialization. The physical cache model will contain one physical table for each virtual group marked as a materialized view.

Note that if no virtual groups are marked as materialized views, then the Materialization physical cache model will not be created or included in the VDB.

Virtual Database Deployment

When deploying a virtual database (VDB) in the MetaMatrix Console, the MetaMatrix administrator must define connector bindings for all physical models in the VDB. This is true for the well-known materialization cache model also. In the New VDB or New VDB Version wizard, the materialization model will show up in the Connector Binding definition panel. The administrator should select the appropriate JDBC connector for the data source where the materialized view cache will reside.

When deploying a VDB containing materialized views, the MetaMatrix administrator will be presented with the option to save administration scripts to the local file system. This is in the New VDB, New VDB Version, and Import VDB wizards. The administrator should save the scripts to a location where they can be accessed by a database administrator responsible for creating, loading, and refreshing the cache database. See Cache Administration section.

This is a summary of what an administrator must do when deploying a VDB containing materialized views:

1. Define connector binding from the materialization model to the physical cache database.

1. This defines the DBMS type, subsequently used to save cache creation and loading scripts appropriate for that database type
2. The username and password used in the connector binding must have privileges for the cache database
3. The cache database may not exist at this point, in which case the connector binding cannot be started, and the VDB cannot be activated.

1. Define username and password for creating and manipulating (swapping and truncating) cache tables in the cache database.

1. The username and password must directly connect to a schema or catalog in the database that is appropriate for the VDB version. This cache may be used by multiple VDB versions, but this is in general not recommended.
2. This username and password will be used for executing the Create, Truncate, and Swap scripts.

1. Define username and password for connecting to MetaMatrix, to issue insert commands against the virtual groups during loading and reloading.

1. This user information is not necessarily the same as the user information used in the connector binding, although it can be.

1. Save DDL scripts to the file system.

1. The administrator specifies the location to save the files.
2. The location should be accessible to the DBA or the scheduling process that will execute the scripts against the database
3. Four scripts will be saved with names appropriate to the VDB and version.

Cache Characteristics

Cache Persistence Mechanism

Materialized views cache their data in an external database system. This database may be the same as the database used as the MetaMatrix repository, or it may be a different one. In general it is recommended that the MetaMatrix repository not be used for materialized view caching, as heavy use could impact the performance of other core functionality of the MetaMatrix Server.

The following DBMS systems are supported for this purpose:

1. Oracle 8i or 9i or 10g

2. SQL Server 2000

3. DB2 8

Cache Operational Policy

Since the actual physical cache for materialized views is maintained external to the MetaMatrix system, there is no pre-defined policy for clearing and managing the cache. These policies will be defined and enforced by administrators of the MetaMatrix system.

Cache Administration

The cache used by materialized views is administered external to MetaMatrix. This provides a great deal of flexibility on how that cache is managed, refreshed, backed up, and otherwise administered along with other enterprise sources.

Administrator Responsibilities

These are the responsibilities of administrators with respect to MetaMatrix's materialized view data caching.

Operation	Run Against	Required Rights for Cache Database	Required Rights for MetaMatrix	Scripts Used	Frequency
Create cache database	Cache database	Create	None	Create	Once per VDB version
Initially populate cache	Cache database and MetaMatrix	Alter, Insert	CRUD	Truncate Load Swap	Once per VDB version
Refresh cache	Cache database and MetaMatrix	Alter, Insert	CRUD	Truncate Load Swap	Desired cache refresh rate

The refresh cache operation is recurring, basically at whatever frequency is appropriate for maintaining the desired data "freshness" for the VDB version. This operation can be scheduled to execute the MetaMatrix scripts, using standard scheduling mechanisms such as cron on UNIX or the Windows scheduler.

The truncate and swap scripts are used in the initial load. The truncate is a no-op in this case, and the swap happens because data is always loaded to the staging table, which must then be swapped with the real cache table.

MetaMatrix Administrative Scripts

To perform the administrative operations, MetaMatrix provides a set of 4 scripts. Here is a summary of the scripts. All the scripts use a similar template for their file names: VDBName_VDBVersion_Name.ddl).

Script Base Name	Use	Description	Run Against	User Info
Create	Creation	Create tables for cache	Cache database	Script user
Truncate	Reload	Truncate temporary table when inserting new records	Cache database	Script user
Load	Load or reload	Execute queries against materialized virtual groups, insert results into cache	MetaMatrix	Access user
Swap	Reload	Swap temporary table with cache table after new records inserted	Cache database	Script user

In addition to these scripts, MetaMatrix provides a connection properties file with encrypted passwords.

Materialized Database Creation

The cache for materialized views is maintained in a separate database system. Therefore, tables must be defined in the database to hold the cached information for the materialized views in a VDB. Materialized views are scoped to a specific VDB version. Therefore, the cache tables should in general be segregated in the cache database by VDB and version. It is the responsibility of the MetaMatrix and database administrators to ensure that the database is defined appropriately for the VDB version, and to ensure that two users are created with the appropriate privileges:

1. Script user – Used when executing Create/Truncate/Swap scripts [note used for Load scripts, since that executes directly against MetaMatrix with MetaMatrix user info]. Requires create/ load/alter privileges against the materialized view tables in the database.

2. Access user – Used in connector binding definition both when reading the cache for user queries, and when refreshing the cache using the Load script. Requires read and insert privileges against the materialized view tables in the database.

MetaMatrix provides a DDL script to create all the materialized tables for a VDB. This script is DBMS-specific. When deploying a VDB, the type of the DBMS is determined from the connector

binding used for the materialization physical model, and the appropriate script for that DBMS is extracted from the VDB and saved to the local file system.

Script name: <VDBName>_<VDBVersion>_Create.ddl

Example for “CustomerInfo” VDB, version 7: CustomerInfo_7_Create.ddl

The create script is a standard DDL script that can be executed against the DBMS system using whatever mechanisms it supports. The script must be executed against a database schema/ catalog with the appropriate privileges as specified in the MetaMatrix Console.

Materialized Database Loading

MetaMatrix provides a set of 3 scripts to initially populate and subsequently reload (“refresh”) all the materialized tables for a VDB. These scripts are DBMS-specific. They are extracted from the VDB when deploying that VDB, and saved by an administrator to the local file system. These scripts all assume that the cache tables have been created, using the “Create” script described above.

These three scripts do the following for each materialized view:

1. Truncate records in a temporary table (executed directly against DBMS)
2. Load – execute query against virtual group, and insert into temporary table (executed against MetaMatrix)
3. Swap temporary table with cache table (executed directly against DBMS)

The second script uses the MetaMatrix batched inserts functionality. Therefore, the MetaMatrix Server must be running, and the VDB must be deployed and activated. The queries that are executed all use the “cache override” option, so that the transformations are executed.

The first and third scripts execute DBMS-specific operations, to more efficiently update the cache records. These scripts are standard DDL scripts that can be executed against the DBMS system using whatever mechanisms it supports. The scripts must be executed against a database schema/catalog with the appropriate privileges as specified in the MetaMatrix Console.

The three scripts can be executed together to perform initial loading and subsequent refresh. This can be done using the loadscript.cmd (.sh on UNIX) command script, located in the server materializedviews directory. This script can be used to schedule database loading activities.

2.4. Limitations

1. Materialization works only with Relational tables and views. It does not work with Data Access virtual groups or procedures.
2. A user cannot specify OPTION NOCACHE on virtual groups that are not at the top level (in the user query). The design does not currently support this.

2.5. Outstanding Issues

Specifying OPTION NOCACHE within a transformation query is the same as specifying it a user query – the identified virtual groups will always be re-computed in the context of the virtual group containing the transformation query with the override option.

Result Set Caching

MetaMatrix provides the capability to store the results of specific queries. MetaMatrix can be configured to store the results for end-user queries, or to cache the results of atomic queries issued to data sources in response to end-user queries, or both. When the exact same user query is submitted to the MetaMatrix Server, the cached results will be returned. Similarly, if the exact same atomic query is encountered while processing an end-user query, the cached results will be used in processing that query, even if the end-user query is different than the original one. These caching techniques can yield significant performance gains if users of the system submit the same queries often, or if user queries result in the same queries being issued to the underlying data sources.

Result set caching will cache result sets based on an exact match of the incoming SQL string. It only applies to SELECT and EXEC statements; it does not apply to SELECT INTO statements, or INSERT, UPDATE, and DELETE statements.

3.1. Support Summary

1. Caching of end-user queries (on a per-query service basis)
2. Caching of data source queries (on a per-connector binding basis)
3. Scoping of caching to either VDB or session level
4. Caching of XML result sets
5. Users explicitly state whether to use a result set cache or not (if available)
6. Administrative clearing of caches

3.2. User Interaction

End users or client applications explicitly state whether to use result set caching for each query. This can be done by setting the JDBC ResultSetCacheMode execution property to true to enable the use of caching for that statement, or false to disable it. The default is true. Note that if this property is set to true, it only has an effect if caching has been enabled in the Server.

Specification of result set caching for ODBC and SOAP is through extra URL properties, on a per-connection basis.

Each query is re-checked for authorization for the user's permissions, regardless of whether or not the query results have been cached.

3.3. Cache Configuration

Result set caching in the MetaMatrix Server is configured in the MetaMatrix Console. By default, result set caching is disabled. When enabled, the default caching scope is restricted to a particular VDB.

Result set caching can be enabled and configured in two places:

1. End-user queries – In Configuration Deployment panel, QueryEngine PSC, QueryService properties (per MetaMatrix process)
2. Data source queries – In Configuration Connector Binding panel, properties

The following are the properties that can be set at the query service and connector binding levels.

Property	Description	Type	Default
ResultSet Cache Enabled	Enable result set caching for the source	Boolean	false
ResultSet Cache Maximum Age	Maximum time before the cache is automatically cleared	Integer – millsec	0 (no limit)
ResultSet Cache Maximum Size	Maximum size the cache will be allowed to grow to before objects are removed	Integer – MB	0 (no limit)
ResultSet Cache Scope	Whether caching is restricted to a specific session (‘session’) or a specific VDB version (‘vdb’)	String	vdb

End-User Cache Configuration

End-user query result caching is controlled in properties for the individual Query Services in each PSC. These properties can be accessed in the Configuration Deployment panel, by selecting the QueryEngine PSC, then selecting an individual QueryService. The properties are shown at the bottom.

Data Source Cache Configuration

Data source caching is controlled in properties for the individual connector bindings for each source. These properties can be accessed on the Configuration Connector Bindings panel, by selecting an individual connector binding and selecting the Properties tab. Note that these are all optional properties. Check the Include Optional Properties setting to see these properties.

3.4. Cache Characteristics

Cache Persistence Mechanism

Result sets are persisted in object caches local to individual Java processes (virtual machines or VMs). User query result sets are cached in the VM of the individual query services running

on host machines. Data source query results are cached in the VM of the individual connector bindings running on host machines.

Cache Operational Policy

Objects are removed from the cache on a least frequently used basis. When the cache reaches approximately 90% of its maximum size (as defined by the ResultSet Cache Maximum Size property), query result sets are removed to bring the cache back down to the 80% threshold.

3.5. Cache Administration

Result set caching is administered using the servershell script under the <server install>/util directory. This script is called servershell.cmd on Windows systems, and servershell.sh on Unix systems.

The following administrative operations can be performed for all caches of a given type, by specifying the appropriate parameters on the script's command-line:

1. Clear cache options
2. clearQueryServiceResultSetCaches – clears all the result set caches in all the query services
3. clearConnectorServiceResultSetCaches – clears all the result set caches in all the connector services

These can be listed in help by executing the script with the command-line argument expertmode on.

3.6. Limitations

1. BLOBs and CLOBs cannot be cached. Therefore, any query retrieving BLOB or CLOB information will not be cached.
2. Caches are not distributed; they are bound to a particular MetaMatrix process. So, if the same query is routed to two different host machines, the query will be executed twice, and cached independently on each machine.
3. Result set caching is not transactional and should not be used in the scope of XA transactions. Transactions depend on (and enforce) consistency of data, and cached data cannot be guaranteed to be consistent with the data store's data.
4. ResultSet Cache Scope property values are not constrained, so users may type in the incorrect values (only vdb and session are allowed). [defect 14444]

3.7. Outstanding Issues

Specification of result set caching for ODBC and SOAP is through extra URL properties, on a per-connection basis.

Code Table Caching

MetaMatrix provides a means of caching small, frequently used tables of data. This is referred to as “code table caching” or “reference data caching”.

Code table caching is done by using the lookup scalar function, provided as a standard function with MetaMatrix. The lookup function provides a way to get a value out of a table when a key value is provided. The function automatically caches all the values in the referenced table for the specified key/value pairs. The cache is created the first time it is used in a particular MetaMatrix process. Subsequent lookups against the same table using the same key and value columns will use the cached information.

This caching solution is appropriate for integration of “reference data” with transactional or operational data. Reference data are static data sets – typically small – which are used very frequently in most enterprise applications. Examples are ISO country codes, state codes, and different types of financial instrument identifiers.

4.1. Support Summary

1. Caching of small, frequently accessed tables

2. Administrative clearing of cached tables through command line utility

4.2. User Interaction

This caching mechanism is automatically invoked when the lookup scalar function is used. Each time this function is called with a unique combination of referenced table, key element, and returned element (the first 3 arguments to the function), the MetaMatrix System caches the entire contents of the table being accessed. Subsequent lookup function uses with the same combination of parameters uses the cached table data.

Note that the use of the lookup function automatically performs caching; there is no option to use the lookup function and not perform caching. Once the values for a particular lookup are cached, they will be used until the cache is cleared through the Console.

Specification:

The following provides information on the use of the lookup function.

Function	Definition	Datatype Constraint
LOOKUP(codeGroup, returnElement, keyElement, keyValue)	In the lookup group codeGroup, find the row where keyElement has the	codeGroup, returnElement, and keyElement must be string literals containing metadata identifiers, keyValue datatype

Function	Definition	Datatype Constraint
	value keyValue and return the associated returnElement.	must match datatype of the keyElement, return datatype matches that of returnElement

See the MetaMatrix Query Support Booklet for more information on use of the lookup function.

Examples (can be used wherever scalar literal can be in query):

lookup('ISOCountryCodes, 'CountryName', 'CountryCode', 'US')

lookup('StatePostalCodes, 'StateDisplayName', 'PostalCode', '63131')

lookup('EmpIDs', 'Name', 'ID', 'm204815')

4.3. Cache Configuration

Cached lookup groups might consume significant memory. You can limit the number and maximum size of these code groups by setting properties of the Query Service through the MetaMatrix Console.

The following are the properties that can be set at the query service level to control reference data caching.

4.4. Cache Characteristics

Cache Persistence Mechanism

Reference tables are persisted in object caches local to individual Java processes (virtual machines or VMs). They are cached in the VM of the individual query services running on host machines.

Cache Loading Policy

When a user calls the lookup function for a unique combination of table, key element, and returned element, the MetaMatrix System caches all key-value pairs of that table for the specified elements.

The MetaMatrix System uses this cached map for all queries, in all sessions, that later access this lookup group.

Cache Operational Policy

Cached tables are never proactively cleared. They can only be cleared through using the svcmgr utility script.

4.5. Cache Administration

The lookup cache can be cleared using the svcmgr utility script. To clear cached code tables, set expertmode on, and then pass in the clearCodeTableCaches argument.

4.6. Limitations

1. Cached code tables are not tied to specific virtual databases. This may result in an error when the same model is used in multiple virtual databases while it has bindings to different sources of data. This may pose a security issue if a cached code table is loaded in the context of one user but then accessed by another user who should not have access to that data. [defect 11629]
2. The use of the lookup function automatically performs caching; there is no option to use the lookup function and not perform caching. [defect 14445]

Cached code tables are never proactively cleared by the Server. If a lot of code tables are loaded, or large code tables are loaded, the Server's available memory could be exceeded.