Avalanche Migration Guide

Teradata to Avalanche

Prepared By: Lynn Hedegard

Date: 9th September 2019

Table of Contents

Introduction

Actian Avalanche (hereafter Avalanche) is a **fully managed cloud data warehouse service**. Avalanche enables you to easily create and use data analytic services in the Actian Cloud (Amazon Web Services or Microsoft Azure) without having to deal with traditional operational management of software, hardware, and all the associated management and monitoring systems. Using Avalanche involves a simple five-step process:

1. Set up an AWS or Azure account
2. Set up an Avalanche account
3. Create a cluster
4. Load data
5. Query your data

Avalanche is scalable — from modest-sized data warehouse deployments to massive data repositories. Built with component cluster architecture, your organization can optimize the IT expense structure by provisioning only those compute resources necessary for the current workload. Moreover, if you need to temporarily suspend your analytic service, the compute resources can be disabled, while the data remains in AWS or Azure storage systems.

The purpose of this guide is to provide a high-level overview of the steps necessary to migrate an existing Teradata environment to the Avalanche service. Enough detail is provided to achieve this goal without getting too deep into the myriad technical details and options. Detailed instructions for initial setup, configuration, ETL, management, etc. are covered in various Avalanche User Guides, and will be referenced wherever appropriate.

The intended audience of this content includes DBAs, migration architects, program managers, and others who need to understand the methodology regarding migration of their existing environment to the Avalanche service.

Plan Your Migration to Avalanche

Create Your Migration Team

You must identify the people in your organization who will participate in the migration effort. Some key roles include:

- Program manager
- Migration architect
- Application architect
- Data architect
- Quality assurance manager

One person may act in one or more of the roles mentioned above, but someone should be assigned to each of the roles.

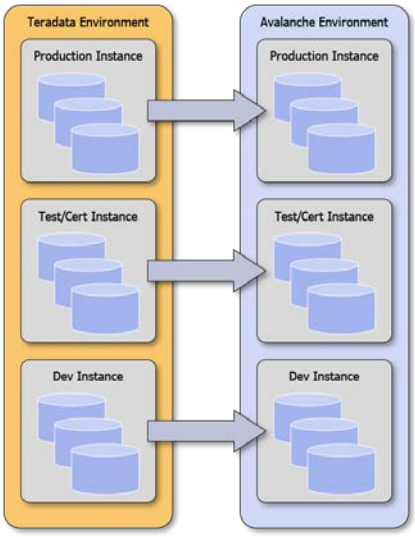
High Level Migration Planning

The planning you make prior to executing the migration process will directly affect the success you achieve with migration project.

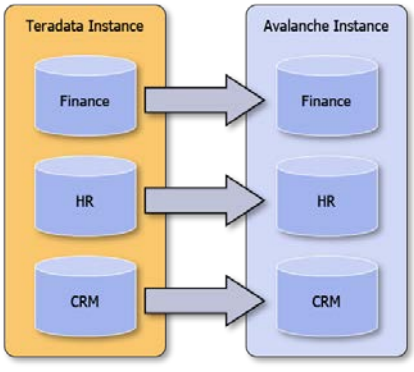
Actian recommends that your first migration have a minimal amount of change to high-level systems and processes. As substantial changes are made to processes in the new environment, it becomes more difficult to identify the source of problems – are they the result of poorly made architectural assumptions or decisions, or are they the result of errors in the porting of a given process?

The migration architect should identify which analytic environment will migrate to Avalanche. Each analytic environment should have a separate migration plan.

Within each operational environment, there will be separate databases that support various applications such as Finance, HR, CRM, Marketing, etc. You should create a separate database (i.e. schema) within Avalanche that maps to the associated database from Teradata.



Migration of Operational Environments



Migration of Databases

Enterprise Level Considerations

Here is a high-level list of topics to consider during your planning stage. We will delve into each topic in more depth in subsequent sections.

- Applications: Identify which applications will migrate to the Avalanche environment. Determine how these applications interact with their respective databases. Most "analytic" applications are primarily read-only (although they might persist temporary or intermediate data). "Operational" applications (e.g. call center apps, manufacturing control, or delivery logistics) contain a mixture of analytical operations coupled with record-keeping operations. Identify any application processes that require transactional semantics.
- Databases: Identify which database instance(s) and database(s) support these applications. These databases will be the focal point of the migration to Avalanche. For any given data set in the database, are there special security requirements (e.g. restricted access, need for encryption, data obfuscation, etc.)? If you are a multi-national organization, are there any special regulatory consideration for data that might leave the region?
- Database Objects: Identify which database objects will be migrated (e.g., tables, views, stored-procedures, etc.).
- Data-Sources: Identify the data source(s) that feed these databases. You should consider the geographical location of these databases and the network bandwidth available between these systems and Avalanche.
- Data Integration: Identify which data integration tools are currently in use. Do you perform ETL, ELT, special data cleansing processes, or other operations? How often is this data acquired and loaded into the database? Are there any SLAs regarding when a given data load job begins, and needs to be complete?
- Users: Identify the users that will interact with the applications being migrated. In many cases, an end-user interacts with the database through a "system-of-engagement" (e.g., internal systems, such as finance, marketing, CRM, etc., as well as BI tools, such as QLIK, Looker, Tableau, etc.). How will applications access Avalanche? What user ID(s) will the applications use? Will the applications use session pooling? Will certain applications have restricted access to certain data items in Avalanche? Also, you should identify those human users who will access Avalanche directly.
- Security: Identify user ID(s), and their associated roles and permissions. Identify what methods will be used for authentication, authorization, accounting, and logging.
- Network: How will your applications and end users connect to the Avalanche service? Based on your organization's internal security policies, changes may be required to firewall settings, port access, etc. in order to facilitate access to cloud-based services.
- Operational Procedures: Operational procedures sometimes can be hard to quantify, but this is why you should examine them. Will any operational procedures need to be modified in order to execute in a cloud-based environment? To ensure that your migration completes with minimum impact to the enterprise, walk through these procedures with an eye for adapting to a cloud-based environment.

- Operational Environments: Most organizations have a development environment, test/certification environment, and a production environment. While these environments will be similar in nature, each will have a few unique characteristics. Thus, there should be a separate migration plan for each environment.

Test and Certification

After you have defined the high-level systems and processes that will migrate to Avalanche you need to develop a test plan to validate their functionality in the new Avalanche environment. You should devise a methodology to validate that a given data set in Avalanche matches the associated data set in Teradata.

In the current state of practice, few database systems remain unchanged for very long. Thus, it may be worthwhile to consider making a checkpoint of a given source database so that the comparison process between the Teradata data and the Avalanche data can be completed with a minimal number of exogenous variables.

Description of Existing Environment

Identify Components to Migrate to Avalanche

In this step you will examine your existing environment and determine which components will migrate from your existing environment to the Avalanche environment.

1. Collection of Applications: The migration architect who is planning this migration must identify those application that will be migrated to the Avalanche service. Examples include:
   - Executive sales dashboard
   - Internal applications (e.g. Finance, HR, Logistics, Factory Management, etc.).
   - Customer sales portal
   - Customer service portal
   - Others
2. Data Objects Used by Applications: For each of the applications identified above, determine the data set(s) are required for the application to perform its intended services. These will include lists of
   - Existing database tables used by the applications
   - Stored Procedures used by the applications
   - Existing non-database objects used by the applications
3. Data Acquisition Processes:
   - Current data integration tools used
   - Current data load and/or refresh rates
   - Network considerations
4. Management Subsystem:
   - List of existing user and/or application profiles and roles
   - List of current security policies
   - Description of expected up times, reduced access, maintenance periods, etc.)
   - Current SLA policies (e.g. response time, throughput, etc.)

Description of Applications

The migration architect should create an inventory of applications that will migrate to the new Avalanche environment. These applications will fall into one or more of the following categories;

- Dashboard Applications: These applications are generally "report-based" applications used to understand the current state of various organizational processes. Users may have the ability to tailor a report to their specific need, but in general, these applications do not modify the database in any significant fashion. Examples include sales reports with dimensional attributes, such as revenue by product, revenue by region, revenue by date, defects by product, etc.
- Customer Centric applications: These applications can be considered as systems-of-engagement, interacting with the customer to provide reporting services, marketing campaigns, and a limited number of transactional services, such as user profile updates, scheduling on-site work, etc. These systems should not be confused with high-volume OLTP systems.
- Data Science Based Environments: A significant amount of data mining and data science activity can migrate to the Avalanche environment. These applications are dominated by bespoke tools written in python, Scala, R, etc. It is imperative that the migration architect identify the data sets used by the data science teams and ensure that historical data is migrated to Avalanche – and that current data from various operational processes is propagated to the new Avalanche environment.
- ERP Applications: These applications are generally "business planning" applications used to control and manage various organizational processes. As such, they have both frequent updates, as well as frequent query activity (Avalanche is well-suited for these usage profiles).

Description of Source Data (Teradata Systems)

This section provides an inventory of Teradata elements that should be identified during the planning stage.

Identify Teradata Database Tables and Sizes

The first step of this process is to identify each Teradata database that will migrate to Avalanche. For each database, identify the set of tables that are included in the migration. For these tables you should determine number of rows, and the approximate size of a row. This data will be used to calculate the approximate size of the data set when it migrates to Avalanche.

You may use the following query to identify all user tables in a given Teradata system.

```
/*
 * For Teradata Systems
 *
 * DBC.TABLESIZE and DBC.DISKSPACE are the systems tables that
 * contain system information regarding space utilization.
 *
 * This query returns a list of tables with the
 * 1. Name of the person who created the table
 * 2. When it was created
 * 3. Who last altered the table
 * 4. When it was last altered
 * 5. When it was last accessed -- Note:
 * 6. How big the table is in MB.
 *
 * Result is returned in GB.
 *
 * Note: If an object existed prior to turning on usecount,
 *       and it has not been accessed since usecount was turned on,
 *       then the last access will be null.
 *
 * Lynn Hedegard
 */
SELECT
    tbl.databasename,
    tbl.TABLENAME,
    tbl.CreatorName,
    tbl.CreateTimeStamp,
    tbl.LastAlterName,
    tbl.LastAlterTimeStamp,
    tbl.LastAccessTimeStamp,
    SUM(currentperm)/(1024*1024) AS Table_Size_MB    -- The tables are spread over
the AMPs, so we have to sum the values
FROM dbc.tablesV tbl
    LEFT JOIN dbc.tablesize mYsize
                ON tbl.databasename = mYsize.databasename
                AND tbl.TABLENAME = mYsize.TABLENAME
GROUP BY 1,2,3,4,5,6,7
WHERE tbl.databasename = 'My_Database'
    AND tbl.TableKind <> 'V'
ORDER BY tbl.TABLENAME;


SELECT b.databasename, a.tablename, a.CreatorName, a.CreateTimeStamp,
a.LastAccessTimeStamp,
       sum(currentperm)/(1024*1024)
FROM dbc.tables a
inner join
dbc.tablesize b
on a.tablename=b.tablename
and a.databasename=b.databasename
group by 1,2,3,4,5 ;
```

Extracting DDL from Teradata Systems

One method to obtain the Data Definition Language (DDL) for tables you plan to migrate to Avalanche is to use Teradata's Teradata Studio.

You can generate DDL for a database, database objects, or both using the *Generate DDL wizard*. The generated DDL will be placed into a .sql file. Use the following steps to generate the DDL:

1. Display the Query Development perspective.
2. In the Data Source Explorer, right-click a database, table, user-defined function, user-defined datatype, or a database view.
3. Select the model elements (e.g., DROP statements, CREATE statements, comments) to include in the DDL script and click Next.
4. Select the model objects (e.g., databases, tables, views, etc) to include in the DDL script and click Next. The object type determines which model object options are available.
5. Select Save and run DDL options (e.g., Folder Name, File Name, etc.), then click Next.
6. Click Finish

Teradata Database Objects NOT Needed in Avalanche

The following list of tables are used by the Teradata system, but should not be part of the migration process.

- DBC
- CRASHDUMPS
- DBCMNGR
- EXTERNAL_AP
- EXTUSER
- LOCKLOGSHREDDER
- QCD
- SQLJ
- SYS_CALENDAR
- SYSADMIN
- SYSBAR
- SYSJDBC
- SYSLIB
- SYSSPATIAL
- SYSTEMFE
- SYSUDTLIB
- SYSUIF
- TD_SERVER_DB
- TD_SYSFNLIB
- TD_SYSGPL
- TD_SYSXML
- TDPUSER
- TDQCD
- TDSTATS
- TDWM

Define the Avalanche System

Define Platform Attributes

Defining the Avalanche system is a straightforward process. Begin by estimating the total UNCOMPRESSED size of the complete data to be processed by Avalanche. A convenient and conservative estimate for the compression ratio is 4. This number is a conservative estimate and is divisible by 2. For example, if you currently have 20TBs of data, the S_{Comp} value would be 20/4 or 5TB of compressed data.

The following calculations summarize the calculations

$$S_{Total} = Estimated\ TOTAL\ size\ of\ all\ UNCOMPRESSED\ data\ to\ be\ loaded\ into\ Avalanche.$$
$$Comp = The\ expected\ compression\ Ratio\ for\ your\ source\ data$$
$$S_{Comp} = S_{Total}\ /Comp$$
$$AUs = \lceil S_{Comp}\ /2 \rceil$$

The number of AUs that you have just calculated will be used when you provision the cluster.

Set Up Cloud Storage

On AWS, create an S3 bucket. It is important to locate your S3 bucket in the same AWS region as your Avalanche service. S3 is used to stage your table data before it is loaded from S3 into Avalanche.

On Azure, create an Azure Data Lake Storage (ADLS) Gen2 storage account with a hierarchical namespace which provides a native directory-based file system. ADLS is used to stage your table data in Azure before it is loaded into Avalanche.

Define Avalanche Schema

Avalanche uses the namespace `SchemaName.ObjectName.`

In Avalanche, users and schemas are essentially the same thing. You can consider that a user is the account you use to connect to a database, and a schema is the set of objects (tables, views, etc.) that belong to that account.

Databases from Teradata maps to schemas in Avalanche. Typically, production schemas will belong to secure user accounts where access is controlled.

Define Users

In Avalanche, groups and roles can simplify control of database access. Groups are used to apply permissions to a list of users, while roles are used to associate subject privileges and permissions with an application.

To create a group, use the SQL statement:

```
CREATE GROUP inside_sales ;
```

Users can be added to a group by specifying:

```
ALTER GROUP inside_sales ADD USERS (dannyh, helent) ;
```

Users can be assigned a default group:

```
CREATE USER bspring WITH PASSWORD='secret', GROUP = engineering ;
```

or

```
ALTER USER bspring WITH GROUP = engineering ;
```

Data Distribution

Avalanche tables are either partitioned or not. Unpartitioned tables are read into cache by each node and are sometimes referred to as “replicated tables.” Unpartitioned tables are generally smaller tables by the number of rows in the schema (for example, dimension tables).

Partitioned tables (sometimes referred to as “sharded” tables) are managed as multiple “chunks” in which a given chunk (partition) resides on a given node. Medium-to-large-sized tables (for example, large dimension and fact tables) are more efficiently processed as partitioned tables because each node needs to only deal with the partitions of the table that are stored on that node.

Following is the syntax for partitioning:

```
CREATE TABLE employee (
   emp_no INTEGER NOT NULL NOT DEFAULT,
   emp_name CHAR(32) NOT NULL NOT DEFAULT,
   dept_no INTEGER,
   emp_rating INTEGER)
WITH
PARTITION = (HASH ON emp_no DEFAULT PARTITIONS);
```

The DEFAULT keyword denotes the pre-configured number of partitions by cluster based on Avalanche Units size for high performance. To specify no partition, use the phrase WITH NOPARTITION.

Constraints

You can specify constraints to ensure that the contents of columns fulfill your database requirements.

Ordinary (enforced) constraints are always checked at the end of every statement that modifies the table. If the constraint is violated, an error is returned and the statement is aborted. If the statement is within a multi-statement transaction, the transaction is not aborted.

Constraints can also be declared as NOT ENFORCED. This allows the database designer to describe a constraint (such as a referential relationship) without the overhead of checking the constraint. The assumption is that the constraint will be enforced externally in some way, and therefore the DBMS does not have to do it. The constraint description is available to the query optimizer and to external query generators, allowing better query plans or better queries to be generated. If the actual data violates the non-enforced constraint, incorrect results may occur.

Constraints can be specified for individual columns or for the entire table. For details, see table-level and column-level constraints.

The types of constraints are:

- Unique constraint–Ensures that a value appears in a column only once. Unique constraints are specified using the UNIQUE option.
- Referential constraint–Ensures that a value assigned to a column appears in a corresponding column in another table. Referential constraints are specified using the REFERENCES option.
- Primary key constraint–Declares one or more columns for use in referential constraints in other tables. Primary keys must be unique.

Additional column-level constraints for masking or encrypting sensitive PI data is available in Avalanche. Here are some examples:

Create a table in which the Social Security number is encrypted using AES 256-bit encryption:

```
CREATE TABLE socsectab (
    fname CHAR(10),
    lname CHAR(20),
    socsec CHAR(11) ENCRYPT )
    WITH ENCRYPTION=AES256, PASSPHRASE='decoder ring', NOPARTITION;
```

Create a table with the address and salary columns masked:

```
CREATE TABLE employee(
    name VARCHAR(20),
    address VARCHAR(20) MASKED,
    salary FLOAT MASKED AS 0)
```

```
WITH ENCRYPTION=AES256, NOPARTITION;
```

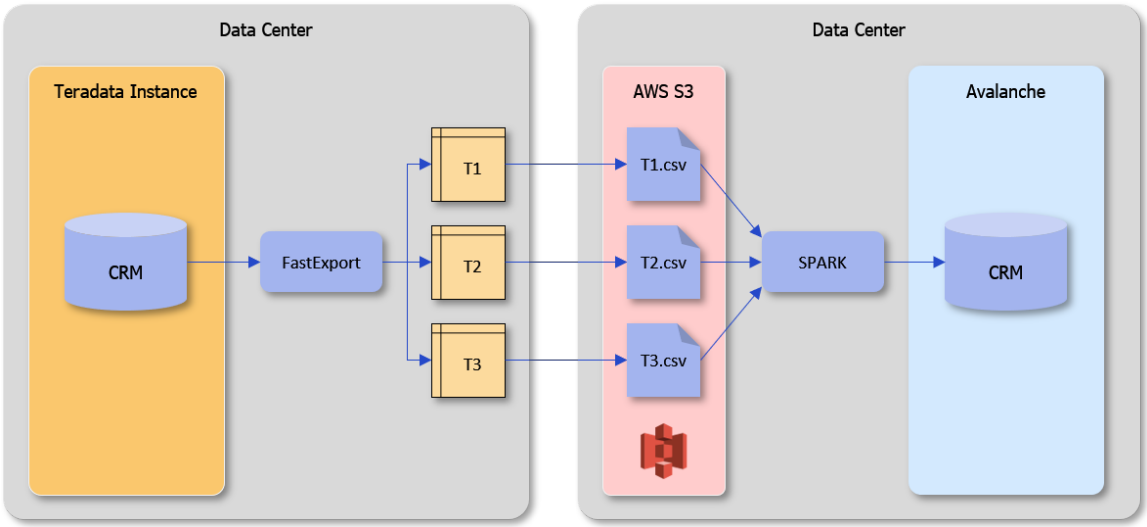
Define Tables

You will use the DDL statements extracted from the Teradata system(s) to create your Avalanche tables.

Data Load Process

Data Load Reference Architecture

This section describes how to perform an initial data load from an existing Teradata system to an Avalanche system. For this description we will use an example of data migration from a CRM database on Teradata to a CRM database on Avalanche. Furthermore, we presume there are three tables in the CRM database, labeled T1, T2 and T3.



The first step in this process is to export the three tables from the CRM database on Teradata to CSV files. The recommended method for this is to use the Teradata utility known as FastExport. This utility is used to export data from Teradata tables into flat files. Data can be extracted from one or more tables using a Join command. Since FastExport exports the data in 64K blocks, it is useful for extracting large volumes of data.

Consider the following Customer_Table:

Cust_ID	FirstName	LastName	BirthDate
101	Mike	James	1/5/1980
104	Alex	Stuart	11/6/1984
102	Robert	Williams	3/5/1983
105	Robert	James	12/1/1984
103	Peter	Paul	4/1/1983

The following code is an example of a FastExport script that exports data from the customer table and writes into a customerdata.csv file.

```
.LOGTABLE tduser.customer_log;
.LOGON 192.168.1.102/dbc, dbc;
   DATABASE tduser;
   .BEGIN EXPORT SESSIONS 4;
      .EXPORT OUTFILE customerdata.csv
      MODE RECORD FORMAT TEXT;
      SELECT CAST(Cust_ID AS CHAR(10)),
             CAST(FirstName AS CHAR(15)),
             CAST(LastName AS CHAR(15)),
             CAST(BirthDate AS CHAR(10))
      FROM
      Customer;
   .END EXPORT;
.LOGOFF;
```

The script can be saved with a name, such as export_customer.fx. The following command can be used to invoke the script.

```
fexp < export_customer.fx
```

Following the completion of the FastExport job, you will need to copy the CSV files to your AWS S3 bucket.

Load Data into Avalanche

Staging Data in a Data Lake

Transferring large amounts of data may be time-consuming and expensive. Use an AWS Snowball or Azure Data Box to easily and securely move your unloaded and converted UTF-8 files to the cloud.

Alternatively, use AWS Console or Azure Portal in a browser to upload the directories and files to an S3 bucket or ADLS directory.

Optionally, both AWS and Azure provide a command-line interface (CLI) that can be downloaded and installed to move data to cloud storage.

Choose your preferred transfer method and upload your data to cloud storage.

Load Data From the Data Lake

Avalanche uses external tables to load data from S3 or ADLS.

To create the external table for data with a header row (S3 example):

```
CREATE EXTERNAL TABLE parts_s3 (
   p_partkey INTEGER NOT NULL,
   p_name VARCHAR(55) NOT NULL,
   p_mfgr CHAR(25) NOT NULL,
   p_brand CHAR(10) NOT NULL,
   p_type VARCHAR(25) NOT NULL,
   p_size INTEGER NOT NULL,
   p_container CHAR(10) NOT NULL,
   p_retailprice DECIMAL(18,2) NOT NULL,
   p_comment VARCHAR(23) NOT NULL
) USING SPARK WITH
```

```
REFERENCE='s3a://<bucket>/part.tbl*',FORMAT='csv',OPTIONS=('header'='true','delimiter'='|');
```

Important! Ensure that your external table column names match the names used for columns in your source data header row.

You can now manipulate the data using standard SQL against the external table.

To load data into your table, use standard SQL. To load the parts table from external table parts_s3, use an INSERT statement:

```
INSERT INTO parts SELECT * FROM parts_s3;
```

This statement reads the data from the external table “parts_s3”, which points to the data on S3 and inserts it into the native Avalanche table “parts.”

Accessing Avalanche

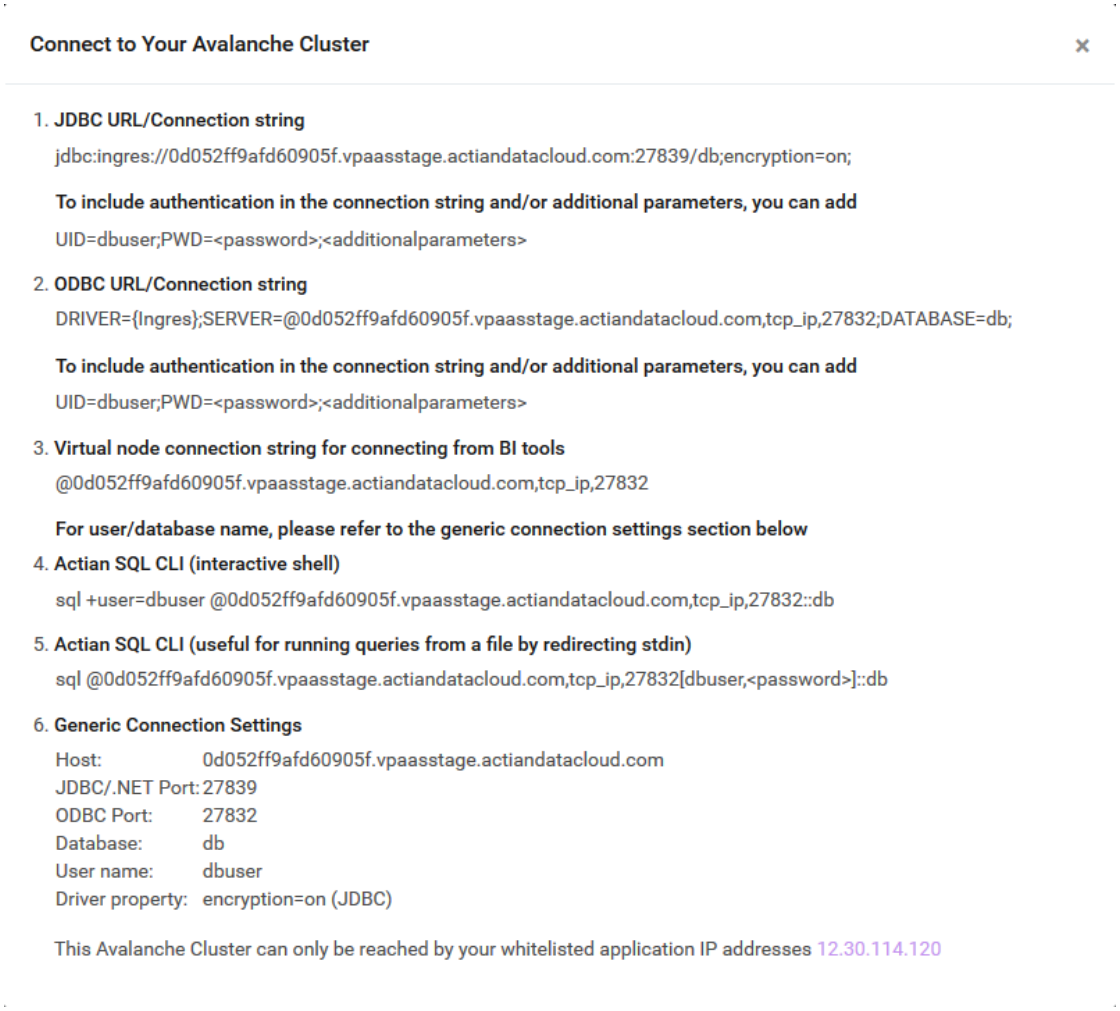
Connecting to Avalanche

Connecting to Avalanche on AWS

After your Avalanche cluster has been created, a hostname is provided. This is a unique label assigned to the cluster that identifies the device for web and database communications.

Clicking the hostname opens the Connect to Your Avalanche Cluster Dialog.

When you click this button, a pop-up dialog displays the various cluster connection strings for JDBC and ODBC connections. Note that you will see a pop-up box similar to the following diagram, but the values will be different for your instance.



Select and copy the connection string for the appropriate database access middleware you plan to use with your applications. Note that the connection string will work only from machines with a whitelisted IP address.

Appendix 1 – Mapping SQL

Core Data Types

Data type mapping between Teradata and Avalanche:

Teradata	Description	Avalanche
INTEGER, INT	32-bit integer	INTEGER
BYTEINT	8-bit integer, -128 to 127	TINYINT
SMALLINT	16-bit integer	SMALLINT
BIGINT	64-bit integer	BIGINT
DECIMAL(p, s), DEC(p, s)	Fixed-point number	DECIMAL(p, s)
NUMERIC(p, s)	Fixed-point number	DECIMAL(p, s)
FLOAT	Double precision floating-point number	FLOAT
DOUBLE PRECISION	Double precision floating-point number	FLOAT8
REAL	Single precision floating-point number	REAL

SQL Language Elements

Converting SQL Language Elements from Teradata to Avalanche

Teradata	Description	Avalanche
ACTIVITY_COUNT	Get the number of affected rows	SQL%ROWCOUNT
exp MOD exp2	Modulo (remainder) operator	MOD(exp, exp2)

Comparison Operators

Teradata	Description	Avalanche
exp EQ exp2	Equal	exp = exp2
exp LE exp2	Less than or equal	exp <= exp2
exp LT exp2	Less than	exp < exp2
exp NE exp2	Not equal	exp <> exp2
exp GE exp2	Greater than or equal	exp >= exp2
exp GT exp2	Greater than	exp > exp2

Built-in SQL Functions

Converting Functions from Teradata to Avalanche:

Teradata	Description	Avalanche
DATE	Get the current date (year, month and day)	SYSDATE
TIME	Get the current time with fraction	SYSTIMESTAMP
ZEROIFNULL(*exp*)	Replace NULL with 0	NVL(*exp*, 0)

SELECT Statement

Converting SQL Queries from Teradata to Avalanche:

Teradata	Avalanche
SEL keyword	Converted to SELECT
SELECT without FROM clause	SELECT … FROM *dual*

QUALIFY Clause Conversion:

Teradata	Avalanche
SELECT c1 FROM t1 WHERE c1='A' QUALIFY ROW_NUMBER() OVER (PARTITION by c1 ORDER BY c1) = 1	SELECT * FROM (SELECT c1, ROW_NUMBER() OVER (PARTITION by c1 ORDER BY c1) rn FROM t1 WHERE c1='A') WHERE rn = 1

CREATE TABLE Statement

Converting CREATE TABLE Statements from Teradata to Avalanche

Teradata	Description	Avalanche
MULTISET	Allows duplicate rows unless a unique is key defined	Keyword not required, removed
[NO] FALLBACK	Store a row copy	Clause removed
[NO] BEFORE JOURNAL	Store before and after images of data	Clause removed
[NO] AFTER JOURNAL		
CHECKSUM = DEFAULT | *val*	Calculate checksum	Clause removed
DEFAULT MERGEBLOCKRATIO	Combine small blocks	Clause removed
PRIMARY INDEX (*col*, …)	Hash partitioning	PARTITION BY HASH (*col*, …)
UNIQUE PRIMARY INDEX	Unique hash partitioning	UNIQUE and PARTITION BY HASH

Column Options and Attributes

Teradata	Description	Avalanche
TITLE 'title'	Column title	Removed from CREATE TABLE, can be used as alias in queries
FORMAT *format'*	Display format	Removed from CREATE TABLE
CHARACTER SET *name*	Column character set	Not supported, removed
CASESPECIFIC	Case sensitive comparison	Default, keyword removed
COMPRESS *val* | (*val*, …)	Column values to compress	Clause removed

Temporary Tables

Teradata	Description	Avalanche
CREATE VOLATILE TABLE	Temporary table that exists until the end of session	CREATE GLOBAL TEMPORARY TEMPORARY TABLE
PRIMARY INDEX	Temporal table can be partitioned	Temporary tables cannot be partitioned
ON COMMIT DELETE | PRESERVE ROWS		ON COMMIT DELETE | PRESERVE ROWS

CREATE PROCEDURE Statement

Converting Stored Procedures from Teradata To Avalanche

Teradata	Description	Avalanche
CREATE PROCEDURE | REPLACE PROCEDURE *name*		CREATE OR REPLACE PROCEDURE *name*
name()	When without parameters	name
IN | OUT | INOUT param datatype(length)		*param* IN | OUT | IN OUT *datatype*
DYNAMIC RESULT SETS *num*	Number of returned result sets	Clause not required, removed
No AS keyword before outer BEGIN END block		IS keyword added
Declarations are inside BEGIN END block		Declarations are before BEGIN END block

Procedural SQL Statements

Converting procedural SQL statements used in stored procedures, functions and triggers from Teradata to Avalanche:

Variable Declaration and Assignment

Teradata	Description	Avalanche
DECLARE var datatype(len) [DEFAULT value];	Variable declaration	var datatype(len) [DEFAULT value];
SET *var* = *value*;	Assignment statement	var := value;

Condition Handlers

Teradata	Description	Avalanche
DECLARE EXIT HANDLER FOR SQLEXCEPTION	SQL exception handler	EXCEPTION WHEN OTHERS
DECLARE CONTINUE HANDLER FOR SQLSTATE VALUE '23505'	Unique key violation	EXCEPTION WHEN DUP_VAL_ON_INDEX

Cursor Declarations and Operations

Teradata	Description	Avalanche
DECLARE *cur* CURSOR FOR SELECT …	Cursor declaration	CURSOR *cur* IS SELECT …
DECLARE *cur* CURSOR FOR *stmt_id*;	Cursor for prepared statement	*cur* SYS_REFCURSOR;
DECLARE *cur* CURSOR WITH RETURN	Result set	*cur* OUT SYS_REFCURSOR
PREPARE *stmt_id* FROM *sql_str*;	Prepare dynamic SQL statement	Linked with OPEN *cur* FOR *sql_str*;
OPEN *cur*;	Open a cursor	OPEN *cur*;
OPEN *cur* [USING *var,* …];	Open cursor for prepared statement	OPEN *cur* FOR *sql_str* [USING *var,* …];
FETCH *cur* INTO *var,* …;	Fetch a cursor	FETCH *cur* INTO *var,* …;
CLOSE *cur*;	Close a cursor	CLOSE *cur*;

Executing Dynamic SQL Statements

Teradata	Description	Avalanche
PREPARE *stmt_id* FROM *sql_str*;	Prepare dynamic SQL statement	Linked with EXECUTE IMMEDIATE *sql_str*;
EXECUTE *stmt_id* [USING *var*, …];	Execute prepared dynamic SQL	EXECUTE IMMEDIATE *sql_str* [USING *var*, …];
EXECUTE IMMEDIATE *sql_str*;	Execute dynamic SQL	EXECUTE IMMEDIATE *sql_str*;

Flow-of-Control Statements

Teradata	Description	Avalanche
FOR *var* AS SELECT … DO *stmts* END FOR;	For each row loop	FOR *var* IN (SELECT …) LOOP *stmts* END LOOP;
FOR *var* AS *cur* CURSOR FOR SELECT … DO *stmts* END FOR;		
IF *condition* THEN … END IF;	IF statement	IF *condition* THEN … END IF;
LEAVE *label*;	Leave a loop or block	EXIT *label*;
LEAVE outer_proc_label;	Leave the procedure	RETURN;
LOOP *stmts* END LOOP;	A loop statement	LOOP *stmts* END LOOP;
label: LOOP *stmts* END LOOP *label*;		<<*label*>> LOOP *stmts* END LOOP *label*;
REPEAT *stmts* UNTIL *condition* END REPEAT;	Conditional loop	LOOP *stmts* EXIT WHEN *condition*; END LOOP;
WHILE *condition* DO *stmts* END WHILE;		WHILE *condition* LOOP *stmts* END LOOP;

Other Statements and Procedural Language Elements

Teradata	Description	Avalanche
CALL proc(param, …)	Call a procedure	proc(param, …)
label:	Label declaration	<<label>>

Other SQL Statements

Converting other SQL statements from Teradata to Avalanche:

Teradata	Description	Avalanche
HELP STATISTICS *table_name*	Show number of unique values for columns	Commented
INS | INSERT INTO *table_name*	Insert a row	INSERT INTO *table_name*

Error Codes and Messages

Mapping error codes and messages from Teradata to Avalanche:

Teradata	Description	Avalanche
SQLSTATE = '02000'	Row not found	*cur*%NOTFOUND for FETCH statement
		SQL%NOTFOUND for UPDATE, DELETE statements
		SQLCODE = 100 in an exeption handler for SELECT INTO

Appendix 2 – Detailed Data Type Mapping

The following tables map the various Teradata Special data types to Avalanche

Numeric Data Types

Teradata Data Type	Description	Vector Data Type	ANSI SQL	TDAT Ext
BIGINT	64-bit integer	BIGINT	**X**	
BYTEINT	8-bit integer, -128 to 127	TINYINT		**X**
DATE*	Teradata has 2 different "DATE" data types. DATE is supported both in its Teradata internal form, and in the preferred ANSI DateTime form. You need to ensure you know the semantics used by the application	INTEGER	**X**	**X**
DECIMAL(p, s), DEC(p, s)	Fixed-point number	DECIMAL(p, s)	**X**	
DOUBLE PRECISION	Double precision floating-point number	FLOAT, or FLOAT8 (8-byte)	**X**	
FLOAT	FLOAT is a Teradata synonym for REAL and DOUBLE PRECISION.	FLOAT, or FLOAT8 (8-byte)	**X**	
INTEGER, INT	32-bit integer	INTEGER	**X**	
NUMBER	*NUMBER is a Teradata extension to the ANSI SQL:2011 standard.*			**X**
NUMBER(n,m)	Number of n total digits with m fractional digits.	DECIMAL(p, s)		**X**
NUMBER(*,m)	Number with up to m fractional digits.	DECIMAL(p, s)		**X**
NUMERIC(p, s)	Fixed-point number	DECIMAL(p, s)	**X**	
REAL	Double precision floating-point number	FLOAT4 (4-byte), or FLOAT8 (8-byte)	**X**	
SMALLINT	16-bit integer	SMALLINT	**X**	

Date/Time Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
DATE	DATE with a DateForm of ANSIDate is ANSI SQL:2011 compliant.	ANSIDATE	**X**	
TIME [(n)]		TIME [time_precision] [time_zone_spec] (WITH TIME ZONE, or WITH LOCAL TIME ZONE)	**X**	
TIMESTAMP [(n)]		TIMESTAMP [(timestamp_precision)] [time_zone_spec]	**X**	

Interval Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
INTERVAL DAY [(*n*)]	Identifies a field as an INTERVAL value defining a period of time in days.	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL DAY [(*n*)] TO HOUR	Identifies a field as an INTERVAL value defining a period of time in days and hours.	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL DAY [(*n*)] TO MINUTE	Identifies a field as an INTERVAL value defining a period of time in days, hours, and minutes.	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL DAY [(*n*)] TO SECOND	Identifies a field as an INTERVAL value defining a period of time in days, hours, minutes, and seconds.	INTERVAL DAY TO SECOND [(second_precision)]	**X**	

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
INTERVAL HOUR [(*n*)]	Identifies a field as an INTERVAL value defining a period of time in hours.	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL HOUR [(*n*)] TO MINUTE	Identifies a field as an INTERVAL value defining a period of time in hours and minutes	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL HOUR [(*n*)] TO SECOND	Identifies a field as an INTERVAL value defining a period of time in hours, minutes, and seconds	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL MINUTE [(*n*)]	Identifies a field as an INTERVAL value defining a period of time in minutes	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL MINUTE [(*n*)] TO SECOND [(*m*)]	Identifies a field as an INTERVAL value defining a period of time in minutes and seconds	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
INTERVAL MONTH	Identifies a field as an INTERVAL value defining a period of time in months	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL SECOND [(*n*,[*m*])]	Identifies a field as an INTERVAL value defining a period of time in seconds	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL YEAR [(*n*)]	Identifies a field as an INTERVAL value defining a period of time in years	*Can be derived via a combination of the [INTERVAL DAY TO SECOND] & [INTERVAL YEAR TO MONTH] data types, and the TIMESTAMPADD(), TIMESTAMPDIFF(), DATE_ADD(), DATE_SUB() functions.*	**X**	
INTERVAL YEAR [(*n*)] TO MONTH	Identifies a field as an INTERVAL value defining a period of time in years and months	INTERVAL YEAR TO MONTH	**X**	

Character Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
CHAR[(*n*)]	Represents a fixed length character string for Teradata Database internal character storage	CHAR[(n)]	**X**	
CHARACTER(*n*) CHARACTER SET GRAPHIC	CHARACTER is ANSI SQL:2011 compliant. GRAPHIC is a Teradata extension to the ANSI SQL:2011 standard	CHAR(size) (Character-Sets are not supported)		**X**
CLOB	Represents a large character string. A character large object (CLOB) column can store character data, such as simple text or HTML. Note: A CLOB column can store XML or JSON documents.	VARCHAR(size)	**X**	
CHAR VARYING(*n*)		VARCHAR(size) (Charactersets are not supported)	**X**	
LONG VARCHAR		VARCHAR(size)		**X**
LONG VARCHAR CHARACTER SET GRAPHIC		VARCHAR(size) (Charactersets are not supported)		**X**
VARCHAR(*n*)	Represents a variable length character string of length 0 to n for Teradata Database internal character storage	VARCHAR(size)	**X**	
VARCHAR(*n*) CHARACTER SET GRAPHIC		VARCHAR(size) (Character-Sets are not supported)		**X**

Period Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
PERIOD(DATE)	A data type that has two DateTime elements associated with it; BEGINNING & ENDING	PERIOD is not supported		**X**
PERIOD(TIME [(*n*)])		PERIOD is not supported		**X**
PERIOD(TIMESTAMP [(*n*)])		PERIOD is not supported		**X**

Byte Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
BLOB[(*n*)]		BLOB is not supported	**X**	
BYTE[(*n*)]		NCHAR(size)		**X**
VARBYTE[(*n*)]		NVARCHAR(size)		**X**

UDT Data Types

Teradata Data Type	Description	Avalanche Data Type	ANSI SQL	TDAT Ext
UND_NAME		Not supported		**X**

Appendix 3 – Glossary of Terms

Different vendors have specific meanings for commonly used terms found in the Analytics / Database space. This section covers the meaning of terms we will use in this guide.

- **Database Instance**: This is the set of executables used to manage one or more "databases."
- **Database**: This is a collection of tables, objects, indexes, views, stored-procedures, etc. used to support a given application.
- **TiB**: The TebiByte specifies a given number of bytes used for digital information. It is a member of the set of units with binary prefixes defined by the International Electrotechnical Commission (IEC). Its unit symbol is TiB. The prefix *tebi* (symbol Ti) represents multiplication by 1024^4, therefore:

  $1 \text{ tebibyte} = 2^{40} \text{ bytes} = 1{,}099{,}511{,}627{,}776 \text{bytes} = 1024 \text{ gibibytes}$

  $1024 \text{ TiB} = 1 \text{ pebibyte (PiB)}$

  The tebibyte is closely related to the TeraByte (TB), which is defined as 10^{12} bytes = 1,000,000,000,000 bytes. One tebibyte (1 TiB) is approximately equal to 1.1 TB. In some contexts, the terabyte has been used as a synonym for tebibyte.