Table of Contents

1. INTRODUCTION
   1. QC Overview
   2. Technical Requirements
2. DATA FORMATS
3. SYSTEM ACCESS

3.1 Network Options

3.1.1 Company Network

3.1.2 Internet

3.2 Transport Options

3.2.1 Access Methods

3.2.2 Compression

3.3 Accessing data stores

3.3.1 Accessing Databases

3.3.2 Accessing remote servers

3.3.3 Accessing cloud stores

3.4 Security

3.4.1 FTP

3.4.2 SFTP

3.4.3 SCP

3.4.4 HTTP

3.5 Data Flow Examples

1. JOB EXECUTION

4.1 Overview

4.2 QC Job Types

4.2.1 Database Synch

4.2.2 File Transfer

4.2.3 Cloud Bound

4.3 Starting a Job

4.4 Job Cancellation

1. DATA COPY SCENARIOS
   1. Local Data Copy
   2. Remote Data Copy
   3. File Loading
   4. File Extraction
   5. Oracle Data Copy
      1. Table Data Copy
      2. Query Data Copy
2. SESSIONS

5.1 Creating New Session

5.2 Modifying a Session

1. FEEDBACK AND ERRORS
   1. Job Status
   2. Failures
      1. QC Errors
      2. Database Errors
2. TESTING
3. ADDITIONAL INFORMATION

APPENDIX A. DATA DICTIONARY A-1

APPENDIX B. ERROR MESSAGES B-1

APPENDIX C. DATA PLATFORMS C-1

APPENDIX D. GLOSSARY D-1

Audience: Operations/Technology

Reporting

ETL

Enclosed is the latest edition of the QCSM-GUI *Technical Specifications* dated July 7, 2015. This document was originally published December 9, 2014 for data experts developing data integration interfaces to Oracle RDBMS. Updated versions were published as follows:

* 2014 – December 9

All updates to this document regarding QC are posted to team sharepoint.

The changes described in this edition of QCSM-GUI *Technical Specifications* will be available in the QCSF Production environment in Aug 15, 2015.

Effective July 15, 2014:

* Added table, partition, and sub-partition lists.

The following is an overview of the enhancements to QCSM-GUI effective with the July 15, 2015 release.

|  |  |
| --- | --- |
| Chapter 1 | No change |
| Chapter 2 | No change |
| Chapter 3 | No change |
| Chapter 4 | Added argument description |
| Chapter 5 | Added SQL Server versions |
| Chapter 6 | Changed Copy Vector menu |

To obtain the latest information or answers to questions about QCSM-GUI contact Alex Buzunov. Metrics Team provides expertize and necessary information to use QCSM-GUI and implement data move.

General information is maintained on the QCSM Web page.

Technical Support Business Support

Phone: Phone:

Fax: Fax:

E-mail: E-mail:

Web: Web:

1. **Introduction**

The Query Copy Session Manager (QCSM-GUI) is part of integrated set of tools used for seamless bi-directional data integration between major data stores. In addition it supports self-service data extraction, and load by Data Experts. This application is developed by Metrics Team in a response to the technical challenges presented by ETL and reporting requirements. Final product presents means for ad-hoc, on-demand data transfer for upstream and downstream Oracle environments including table, partition, and query defined datasets. In addition data transfer sessions can be synchronized with other events and orchestrated.

Figure 1.1. An Overview of the QCSM-GUI Process.

Oracle

QCSM

User Creates User initiates QCSM-GUI Creates Scripts QC Performs Data is Loaded

Query Copy Session Data Transfer Required for Data Copy Data Transfer to Target DB

QCSM-GUI *Technical Specifications* provide Data Experts with resource for on-demand data transfer between staging and/or reporting environments allowing better accident resolution, query performance tuning, data debugging, and application testing.

This document is not intended to provide information on how to develop a system for data transfer or how to do database development. Instead it’s intended to describe what such a system must deliver to your organization.

Table 1.1 describes the contents and purpose of the QCSM-GUI *Technical Specifications*

|  |  |
| --- | --- |
| **Section** | **Description** |
| 1. Introduction | Describes the purpose of the system and provides an overview of QCSM-GUI templates and technical requirements. |
| 1. Parameter Entry | Describes the purpose and requirements of parameter entry of the session. |
| 1. System Access | Describes data transfer configuration |
| 1. Data copy scenarios |  |
| 1. Data Flows |  |
| 1. Data Sources |  |
| 1. Feedback and Errors |  |
| 1. Testing |  |
| 1. Additional Information |  |
| 1. Appendix-A – Arguments | Describes each argument including permitted values. |
| 1. Appendix-B - Templates | Defines detailed aspects of argument templates |
| 1. Appendix-C – Error Messages | Defines all messages generated by QC-CLI |
| 1. Appendix-D- Glossary | Defines the words and phrases with meaning unique to Data Copy and QCSM-GUI |

3. TEMPLATES

When you plan a data move you have to define certain information about it. You must assign an alpha-numeric identifier to each copy request. The order identifier is a Batch Sequence Number or timestamp. It must be unique to your organization. The order identifier along with user name allow QC to maintain unique code snapshot and log files related to this particular order.

Order lifecycle from origination to completion includes metadata entry by Data Experts. To facilitate order definition process Data Experts are required to define source and target templates to help QC identify a method of data copy, *host\_map* to set execution host, and connection credentials for source and target databases.

1. DATA SYNCHRONIZATION

Transactional data can be synchronized using **QC-CLI**

Some useful source on the internet include.

This list about data replication is not exclusive. Further information about data synchronization is available from selected DB vendor and internet.

QCSM-GUI does not implement real-time replication. It makes no representations regarding the accuracy integrity or completeness of your data replication.

Data procedures.

-Pre-determined intervals throughout the day.

Data examinations include:

- maintain your synchronization procedures

- maintain a log of synchronization

- compliance examinations to existing adequacy procedures

- audit trial information against your organizational standards.

In order to facilitate examinations, Data Experts should document and maintain a log of synchronization procedures.

This log should contain date and time of synch whether target was truncated. This log should be maintained for the period of the time and accessibility specified by your DBA.

1. QCSM-GUI persists session metadata and creates command used to invoke QC in CLI window.
2. QC generates Linux shell script and archives it.
3. Archived code is transferred to Linux system for execution.
4. QC opens SSH session and executes data copy using bash. Session stays open until copy process completion.
5. Logs are compressed and transferred to calling environment.
6. QC CLI window stays open until exit confirmation. Data Expert can review data copy on-screen log and close CLI window.

After review session can be modified and resubmitted.

For additional information about QCSM-GUI technical specs and requirements contact dedicated support.

* 1. Data Copy Overview.
     1. Table to Table copy.

This type of copy requires you to set the following arguments in Session Manager: source and target database, field separator, host\_map, and table names.

* + 1. Partition to Table Copy.

You have to set source and target logins, source table and partition name, target partition name (if any), field separator, and host\_map.

* + 1. Sub-Partition to Table Copy.

You have to set source and target logins, source table and sub-partition name, target sub-partition name (if any), field separator, and host\_map.

* + 1. Query to Table copy.

You have to set source and target logins, query file name, target table name (if any), field separator, and host\_map.

* + 1. Table to CSV file extract.

You have to set source login, source table name, field separator, and target file name.

* + 1. Partition to CSV files extract.

You have to set source login, source table name, partition name, field separator, and target file name.

* + 1. Sub-Partition to CSV file extract.

You have to set source login, source table name, sub-partition name, field separator, and target file name.

* + 1. Query to CSV file extract.

You have to set source login, query file name, field separator, and target file name.

* + 1. CSV file to Table load.

You have to set target login, target table name, field separator, and source CSV file name.

* + 1. Query load with delete (no table name).

You have to set source login, skip target table name, field separator, and CSV file name (file name has to include target schema, and table name).

* + 1. Query to target load (no table name).

You have to set source login, skip target table name, field separator, and CSV file name (file name has to include target schema, and table name).

* + 1. CSV to target load (no table name).

You have to set source login, skip target table name, field separator, and Query file name (file name has to include target schema, and table name).

* 1. **Technical Requirements.**

*QCSM Technical Specifications* has been created majorly to describe the technical requirements for transferring data between databases.

It provides detailed information about required source and target arguments and data formats. The most basic technical requirement is defining data vector and source and target templates.

Session metadata has to be packaged into one archived file and sent to remote server for execution. More than one session can be executed at the same time.

QC does not have to be executed manually from command line. QC Sessions are executed directly from QCSM interface. QC bat files can be generated from QCSM for external execution.

Bash script files are transmitted to Linux using SFTP protocol. Remote execution is performed using SSH.

Data copy shell script generates log files which are compressed and send back to calling environment. (For more information, see Section 7 – Feedback and Errors)

You can use QCSM GUI interface to access log files generated by remote data copy script.

For an overview of QCSM Technical Requirements andInformation Flow, see Figure 1.2

Figure 1.2 Overview of QCSM Technical Requirements and Information Flow.

4

2

3

1

Remote

Exec

SSH

BASH

QCSM

QC

6

5

1. Session Manager opens QC CLI window.
2. QueryCopy generates shell script.
3. Generated code is sent to remote server for execution.
4. Data Copy script is executed on remote Linux server.
5. Log file is sent back to user desktop.
6. User can check remote job status and logs using QCSM GUI.
7. **Data Flow.**

This section provides the Data Flow specification which Data Experts can use to perform.

The goal of QCSM in developing these specifications is to provide simple and reliable information flow mechanisms which allow Data Experts to minimize development and operational complexity by using off-the-shelf data synchronization templates, while providing generic access to all available on the market data stores. This section discusses template options, security considerations, network options, design requirements needed to achieve this goal.

* 1. **Execution Options**

QCSM supports two execution types: local (User’s desktop) exec and remote (Linux) exec. Figure 3-1 shows the very high level details of these data load options.

Figure 3-1. Execution Options.

2

1

Spooler

QueryCopy

Loader

Oracle

Oracle

Spooler

Loader

Remote host

QCSM

QCSM

QueryCopy

Windows

Linux

User has two options to copy data using QCSM.

1. Local execution. Used for prototyping of your data transfer.
2. Remote exec used for high performance transfer.

Data copy job performance depends on network connectivity and database server load. User can switch between remote and local load using QCSM interface.

3.1.2 **Linux Access**

Remote load server are configured in host\_map file using QCSM interface. Multiple servers can be configured.

3.2 **Transport Options**

QCSM provides tree options for User to transfer data between Windows desktop and POSIX compliant remote host (Linux/Unux). If transfer is via FTP it is set with *CSV->FTP* copy vector. If transfer is via SFTP it is set with *CSV->SFTP* copy vector. If transfer is via SCP it is set with *CSV->PSCP* copy vector. Figure 3-2 illustrates tree basic types of data transfer using QueryCopy.

Figure 3-2. Data Transfer Flows.

FTP : CSV, JSON

SFTP: CSV, JSON

SCP : CSV, JSON

Exec

Host

QC

FTP : CSV, JSON

SFTP: CSV, JSON

SCP : CSV, JSON

**3.2.1 Access Methods**

Table 3-1 summarizes the transfer methods and file formats that are pertinent toeach transfer method.

|  |  |  |  |
| --- | --- | --- | --- |
| **Assess Method** | **FTP** | **SFTP** | **SCP** |
| CSV data files |  | + | + |
| JSON data files |  | + | + |
| logs | + | + | + |
| code | + | + | + |

The specifications for each access method are described below:

* SFTP – available for bi-directional data transfer POSIX and NT systems. Binary mode is used for transmissions. Encyption is used for file transfer.
* FTP – allows bi-directional data transfer POSIX and NT systems. Binary mode is used for transmissions. There’s no encryption.
* SCP – capable of bi-directional data transfer POSIX and NT systems. Binary mode is used for transmissions. Encyption is used for file transfer.

Table 3-2 Summarizes thespecifications for each transfer method.

|  |  |  |
| --- | --- | --- |
| FTP | SFTP | SCP |
| No file size limit.  RFC 959 Compatible  No encryption  Binary Mode  US ASCII code | No file size limit.  RFC 959 Compatible  SSL Encryption  Binary Mode  US ASCII code | Encrypted  Server side certificates  SSL Encryption |

|  |  |
| --- | --- |
| **Benefits of On-demand Data Sychronization Architecture** | |
| Data loaded in Bulk manner using native loader provided by database manufacturer. | User invokes data synch using variety of different predefined and tested templates. |
| Atomicity of load | The data synch copy-vector identifies the smallest unit of work (data load) providing separation of data transport and load from application logic, allowing user to focus on business requirements and not technical implementation. |
| Performance | Local data load uses local Hard Drive to persists User data. High performance load is done by switching to remote (server) load. |
| Data Investigation | Accessing temporary dump file produced by QC copy job provides additional method to access raw data in CSV or JSON files. |
| Scalability | Scalability is achieved by adding new remote servers and running parts of the load on multiple hosts in parallel. |

QCSM Real Time Data Synch.

Loading Schedules and Data Stores.

**5.3 Typical data strategy**

The following objectives must be accomplished to implement data synchronization strategy.

* Identify data requirements
* Build Data Model
* Mapeach type of data to File Type in QC.
* Lay out the downloading strategy for each typeof data from each system or data store.
* Investigate the delta procedures for each feed.
* Establish a historical data maintenance plan for each data source.
* Create QC Sessions to copy files or copy table data.
* Determine the job scheduling process.
* Define metadata for data load from multiple systems. When QC Sessions are created and all connectors are set – load the data.

6.6 **Job Scheduling – Control-M**

Control-M can be used as dedicated scheduling tool for all jobs including QC. Control-M can be used to schedule and monitor both the batch and real time jobs within QC.

Control-M allows basic orchestration (job dependencies)

**Real time Data Sync**

QC can be configured to continuously capture Oracle transactional data. This service can be stopped on demand for cutoff batch processing or DW maintenance.

**Near real time data copy.**

QC job can be sceduled to start data copy at particular intervals during the day.

**Batch Data Processing.**

QCSM creates QC Sessions allowing User full control over data integration process. QC job can be scheduled to run as part of a larger dependency of data-centric ETL and batch pipeline which already exists in your DW environment.

1. Infrastructure

QCSM is architected as multi-database system. Version 0.3.5 of QCSM supports all major RDBMS including number on noSQL stores. Asadditional data sources are added to QC lineup – more copy vectors are defined and added to menus.

**3.3 Access to QueryCopy logs.**

**3.3.1 QC Feedback**

Feedback items include information conveyed from data spooler and loader. Allfeedback items available via QCSM-GUI. The paragraphs below discuss each of the feedback items (see Section 6 – Feedback and Errors)

Job Status – Job status can be obtained from CLI window or QCSM session status. If job was submitted using QCSM job status will be displayed on RUN button. (Also see QCSM GUI User Guide)

Job Failure – Failure of a QC job is displayed in CLI window or Qcsm session status (Run button). Failed job can be resubmitted using QCSM-GUI or QC-CLI.

Job Logs – Application log file and Loader log files can be accessed in QCSM via “Output” tab.

**3.4 Security**

Copy Job submissions require valid usir ID and password combination for source and target databasees. In addition validLinuxlogin required if job is executed remotelly.

**3.4.1 FTP**

QC logging to company servers using supplied user IDand password. User ID and password are stored in host\_mapfile for each Linux server configured. FTP sessions are not persistent so new session is open for each file transmission. Some files(shards) are transmitted in parallel using multiple FTP sesions. (See QC-CLI User Guide).

Files sent va FTP aro not encrypted and can onlybe sent via company network.

**3.5 Data Flow Examples**

The below diagrams present typical exchanges between QC-CLI and execution server.

In the Examples, Step 1, depicts a QC preparing code and submitting it to remote server. Step 2 includes copy job submission with a possible jeneration of error. Step 3 demonstrates QC resubmitting failed job using Session file (Details are in Section 5, and Appendix C)

Typical QC-CLI information flow via FTP.

Server

Client

* Uncompress code file
* Start data extract
* Verify data dump exists
* Start data load
* Delete temp data file
* Verify file existence
* Open FTP connection
* Login with user/pwd
* *Bin*
* *put <filename>*

Signal about job completion to QC-CLI

If rejected fix problems and resubmit

Accessing Sensitive Data via Internet.

To access financial data strong encryption is required. Due to confidential nature of the market data you transfer all client software used to communicate with execution server must be 128-bit Strong Encryption. Strong Encryption provides the highest level of protection for information transmitted over the Internet and is particularly suited for confidential on-line transactions. NOTE: no encryption needed for in-house transfers.

Data Perspective.

The QCSM-GUI provides up to three data perspectives with which to view the data. Perspective allows User to secure data persisted in different ways. The available perspectives are:

Remote – data located on remote server

Cloud – data located in cloud storage

Local – data persisted on local drive.

When User creates a session QSCM decides which of the above perspectives is applicable.

“New Session” form provides mechanism to select preferred source and target templates which define data perspective. Once selected, the perspective will be applied to all data copy submissions.

To change a perspective open “New Session” form and chose new session templates. Argument values can be optionally inherited by new session.

1. **Job Execution.**

The objective of this section is to describe QCSM data copy facilities. It provides an overview of data copy execution, describes copy vectors, templates, and provides representative sample of copy scenarios.

* 1. **Overview**

The mission of QCSM is to establish and maintain the complete life cycle of data copy in your Data Warehouse. To achieve this goal QCSM must be able to uniquely identify all related events including data extract and load. Furthermore it has to be able to link related artifact like load logs to QCSM Graphical User Interface (GUI).

* + 1. Creating new Session in QCSM GUI.

User creates new Session which is used to execute data copy job.

QCSM

Session

Job

Session contains information required to perform physical data copy. Job is a metadata, related to performed data transfer like logs, timestamp, and job status.

Table 4.1 lists available Session types. Each format is fully defined in Appendix C.

|  |  |  |
| --- | --- | --- |
| **Session type** | **Copy Vector** | **Description** |
| Spool and Load | ORA11G-ORA11G | Copy from Oracle to Oracle |
| Load from File | CSV-ORA11G | Load CSV file into Oracle table |
| Extract to File | ORA11G-CSV | Extract table data to CSV file |
| Inbound file transfer | CSV-PSCP | Transfer file to remote host |
| Outbound file transfer | PSCP-CSV | Transfer file from remote host |
| File splitter | CSV-SPLIT | Split CSV file to many. |
| File merger | CSV-MERGE | Merge CSV file into one |

* 1. **Transfer types**

**3.2.1 Data Copy Execution**

When User originates a copy request to transfer data QCSM records certain information about the order, also it assigns an alphanumeric identifier (job ID) to each transfer. The Job ID may be a batch Sequence Number or another identifier meaningful to the process. Job ID must be unique across the company within Business Day.

The nature of the transfer must be recorded along with date and time of the job. Interdepartmental data copy must be approved by Compliance Control guarding the flow across Chinese Walls.

**3.2.2 Data Copy Request**

When User originates a Data Copy Job and then subsequently transmits data certain information is required to be reported including copy vector, timestamp, and status.

The Data Transfer Desk must be run as a place within the firm where transfer request can be executed either automatically or with assistance of Data Expert. Finally extra-departmental requests have to be handled with extra care and should require gatekeeper’s approval and manager’s sign off with specific instructions attached.

**3.2.3 Data Security.**

Along with requirements to protect information User must apply rigorous information security protocols.

Here are two obvious but vital principles User has to employ:

Passwords. Always keep your passwords in secret. Never write them down or give them out to anyone, even to IT stuff, for any reason. Data dumps located on your Hard Drive may contain sensitive information. If you think there is a chance someone else knows your password – change it immediately.

Locking your PC. You are responsible for any actions taken when your user ID is in use. You should ensure that you log off or lock your workstation whenever you are away from your desk.

**4.2.4 Session Modifiction**

Since ech Job must hve unique identifier, user initited modification od a job id is typically done via QCSM-GUI.

* Modified Sessions are stored in User Desktop
* User can “Save As” as existing Session.
* When uses resets job ID – old logs will not be accessible via QCSM-GUI.
* It’s used responsibility to keep session files private.

**4.2.5 Job Execution**

When User starts a copy job and QC-CLI subcequentrly executes it, certain information about this job is logged.

User must identify source and target connect arguments. Passwords are hidded in QCSM-GUI. In addition User must identify source and target table names, delimiter, andparallel config. NOTE: if target table is truncated QCSM will prompt for user confirmation.

**4.2.6 Job Cancellation**

To end job execution on Widows User can close QC-CLI window. It will terminate all spoler and loader processes immediately and rollback all loader transactions. Remote loads will terminate because when ssh session is closed it effectively terminates all child processes initiated by data copy shell.

**4.3 Copy Vector Types**

This section describes te types of Copy Vectors and related data elements. Copy Vector defines general direction of data move. Check figure 4.3 for example description.

Figure 4.3. Data Vector Example.

ORA11G – ORA11G

Source - Target

There are 4 types of entitied which can be used on either side of Copy Vector: database, file type, file transfer tool, file transformation tool.

Combination of listed types give full set of available Copy Vectors in QC.

Table 4.3. List of Copy Vector types in QC.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Copy Vector Type** | **From** | **To** | **Result** | **Description** |
| Database | Database | Database | Data copied | Defines data transfer direction between databases. |
|  | Database | File Type | Data extracted | Defines data extraction direction. |
|  | File Type | Database | Data loaded | Defines data load direction |
| File Transfer | File Type | File Transfer Tool | Data ransferred |  |
|  | File Transfer Tool | File Type | Data trasferred |  |
| File Manger | File Type | File transformation Tool | File mangled |  |
|  |  |  |  |  |

**4.3.1 Database Copy Vectors**

Each database is identified by short identifier. For example Oracle 11G database is configured using following identifier:

Figure 4.3.1 Oracle identifier

ORA11G

Oracle database 11G Oracle versions

If there are multiple versions of the same database they are grouped into database family. For example Oracle family includes following Oracle database versions.

Table 4.3.1 Oracle Database identifiers.

|  |  |
| --- | --- |
| **Identifier** | **Full database name** |
| ORA12C | Oracle 12c |
| ORA11G | Oracle 11g |
| ORA10G | Oracle 10g |
| ORA9I | Oracle 9i |
| ORA8I | Oracle 8i |
| ORA733 | Oracle 7.3.3 |
| ORAXE | Oracle XE |

Oracle –

|---

|---

|---

|--

In order to define a copy vector User should utilize “short” databaseidentifier. Copy vector consists of 3parts: source db, pipe type,, targetdb

For example:

Source target

ORAC11G-ORA12C

Short dash short

Copy vector setsdirection of data transfer and type of transport used to do actual data copy.

**4.3.2 File Types**

Currently there are 2 data file types and one metadata file type in QC.

This means QCcan import andexport data in 2 formats: CSV and JSON-LINE and one metadata format: DDL

CSV abbreviates”coma-separated values” and is a format used topersist data in a flat file. JSON-LINE is a flat file format also called newline-delimited JSON. Exch line in such file is a valid JSON value. DDL is a metadata record about your database object.

User can crete copy vector for data extracts and load from and into CSV and JSON-LINE formats. Inaddition it is possible to create copy vector for metadataextraction and load in DDl format.

**4.3.6 Data transfer types**

There are 3 data transfer types defined in QC: FTP, SFTP, SCP. User can define copy vector for files to be ransferred to and from POSIX compliant systems. In this case copyvectorwill look like this:

Example 1:

Local remote

CSV – SFTP

CSV file Posix compliand host

Example 2:

Remote local

SCP – JSON

Posix host JSON-LINE file

4.3.7 **Data transformation tools**

The purpose of data munglin tools in to perform data transformation. Some examples are: file split and file merge. All data transformations are local to User PC.

Copy vector example:

Example 1:

Local Local

CSV- PLIT

One file Multiple files

Example 1:

Local Local

MERGE- JSON\_LINE

Multiple files One file

5.0 **Session Argument Lists**

Session holds metadataabot your transfer, listof arduments needed toexecute qc.exe. Arguments are divided into 3 groups – common, source andtarget. Common aguments are “shared” by Spooler and Loader. Source anrguments areset exclusively for Spooler, and Target arguments are defined for Loader.

Some Common arguments are used by QC to set internal structures.

Major data elements of a new session are:

|  |  |  |
| --- | --- | --- |
| **Data Copy Session (Sub-partition copy)** | | |
| **Common Arguments** | **Source Arguments** | **Target Arguments** |
| -w [--copy\_vector]  -o [--pool\_size]  -r [--num\_of\_shards]  -t [--field\_term  -l [--lame\_duck]  -K [--keep\_data\_file]  -V [--validate]  -U [--truncate\_target]  -E [--ask\_to\_truncate]  -X [--key\_on\_exit]  -L [--email\_to]  -M [--log\_dir]  -F [--default\_spool\_dir]  -B [--job\_name]  -Y [--time\_stamp]  -0 [--column\_buckets]  -1 [--job\_pre\_etl]  -2 [--shard\_pre\_etl]  -3 [--job\_post\_etl]  -4 [--shard\_post\_etl]  -C [--loader\_profile]  -5 [--host\_map]  -6 [--spool\_type]  -dbg [--debug\_level]  -spID [--status\_pipe\_id] | -q [--query\_sql\_file]  -Q [--query\_sql\_dir]  -c [--from\_table]  -P [--from\_partition]  -S [--from\_sub\_partition]  -j [--from\_user]  -x [--from\_passwd]  -b [--from\_db\_name]  -e [--nls\_date\_format]  -m [--nls\_timestamp\_format]  -O[--nls\_timestamp\_tz\_format]  -A [--header]  -W [--keep\_whitespace] | -u [--to\_user]  -p [--to\_passwd]  -d [--to\_db\_name]  -a [--to\_table]  -G [--to\_partition]  -N [--to\_sub\_partition]  -k [--skip\_rows] |

In order to execute new Job the following criteria should be met.

* All arguments have to have a value
* Host\_map has to be defined for local or remote execution
* Source and target tables has to exists
* Some of target has to exists
* Source partition or subpartition has to exists
* Source file has to exists

4.3.2 **Job Execution Log**

Whenever User transfers data betweendatabases “Ouptput” tab ispopulatedin QCSM GUI. It offers access to all application and SQL\*Loader logs. Users can open log files using default editor.

* Only job configured to run on host #0 will execute locally.
* Configure new remote host in host\_map file.
* Temporary dump file isperssted by default
* Truncate Job will open confirmation window.
  1. Non-Functional requirements.

It must be possible to 100% determine the correct functioning of all software modules and ETL scripts through the unit and regression test harness.

|  |  |  |
| --- | --- | --- |
| Capacity | Horizontal scale. | The design willallow the system to scale horizontally at the individual module level. Adding capacity to the system should always be achievable through adding new host servers. |
| Management | Operations Management | The routine running configuration ofthis system will notrequire any shell access to the production server or databases. All configurationand monitoring isnot on User PC |
| Performance |  | Data copy performance is limited only by network bandwidth. |
|  |  |  |

**4. Data Copy scenarios**

This section provides a representative sample of Data Copy Scenarios which exist in QC Framework. It defines and illustrates selective set of scenarios and their requirements. NOTE: QC is not a real-time system, thus the representations in the scenarios do not reflect the actual time when QC jobs are submitted. They are only intended to illustrate which arguments must be set in for a copy task to be completed.

Each scenario is accompanied by a list of arguments required for it to work. While each scenario illustrates key points of how QC works they are not exhaustive.

Components and Responsibilities.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Component name** | **Responsibilities** | **Non-functional Requirements** |
| Sqlldr.exe | Bulk loader | Load CSV data into Oracle table | Provides high performance bulk load protocol. |
| Sqlplus.exe | Db client shell | Execute SQL statements. | Provides access to data using SQL. |
| Split.exe | File splitter | Split large files into smaller pieces | Provides fast file split, byte or line based. |
| Pscp.exe | SCP client | Transfer binary files | Provides secure file transfer protocol for POSIX systems. |

Figure 6. QC Event Sequence Diagram.

QC

SQL\*Plus

SQL\*Plus

SQL\*Loader

Read table

Metadata.

Spool Data

Load Data

Table 4.2 Local Job Execution Events

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source databse to fetch table andcolumn metadata. |
| 4 | QC extracts data from source | QC creates SQL scripts and spools data using SQL\*Plus. |
| 5 | QC loads data to target | QC created extract scripts using column metadata and invokes loader process. |

Figure 4.1 Table Data Copy Job Events

QCSM-GUI

Oracle

2

Oracle

1

QC-CLI

This scenario illustrates common Event Flow for QC-CLI and includes the following assumptions.

1. Source and Target tables have to exits.
2. Only scalar data copy is supported.
3. Source table is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
4. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
5. Temporary CSV file(s) created during Table Data Copy and removed when load is done. User can keep the file for debugging by setting –*keep\_dump* arg.
6. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4-3. Remote Job Execution Events

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job using QCSM | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument count and values, parses them to well-formed bat command and opens QC-CLI process user defined arguments. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC generates shell scripts. | QC creates SQL and Bash Shell scripts for execution on remote host. |
| 5 | QC transfers compressed code file to remote host. | QC compresses generated shellcode and transfers it to remote host for execution. |
| 6 | QC triggers data spool and load | QC opens SSH connection to remote host and runs generated shell scripts. |
| 7 | QC downloads log files | QC transfers created log files back to User environment. |

Figure 4.2 Remote Table Data Copy Job Events

QCSM-GUI

QC-CLI

Oracle

2

Oracle

1

Remote Host

This scenario illustrates common Event Flow for QC-CLI and includes the following assumptions.

1. Source and Target tables have to exits.
2. Only scalar data copy is supported.
3. Source table is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
4. Copy scripts are executed in bash shell so it has to be set up on Remote Host.
5. Remote Host ha to have enough free space available for Copy Job to persist temporary CSV files.
6. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
7. Temporary CSV file(s) created during Table Data Copy and removed when load is done. User can keep the file for debugging by setting –*keep\_dump* arg.
8. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Sharded File Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of input files | Source | Target |
| multiple | multiple | single | SCV File | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 4 | QC loads file to target | QC shards input file and executes file load using multiple loader instances. |

QCSM-GUI

Loader

QC

Oracle

CSV

Loader

Loader

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. CSV file is well formed with or without HEADER record.
3. Input file is broken to logical shards. Shard size I set by –*shard\_size* arg.
4. QC creates a queue to process file shards. Loader threads are pooled until all shards are loaded. Pool size is set by –*pool\_size* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Multi-File Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of input files | Source | Target |
| multiple | One per file | multiple | CSV Files | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC loads files to target | QC executes file load using multiple loader instances. |

QCSM-GUI

Loader

QC

Oracle

CSV

Loader

Loader

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. CSV files are well formed with or without HEADER record.
3. Input file(s) are broken to logical shards. Shard size I set by –*shard\_size* arg.
4. QC creates a queue to process file shards. Loader threads are pooled until all shards are loaded. Pool size is set by –*pool\_size* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Multi-File Sharded Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of input files | Source | Target |
| multiple | Multiple per File | multiple | CSV Files | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC loads files to target | QC shards all input files into logical partitions and executes file load using multiple loader instances. |

QCSM-GUI

Loader

QC

Oracle

CSV

Loader

CSV

Loader

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. CSV files are well formed with or without HEADER record.
3. Input file(s) are broken to logical shards. Shard size I set by –*shard\_size* arg.
4. QC creates a queue to process file shards. Loader threads are pooled until all shards are loaded. Pool size is set by –*pool\_size* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.4 Table Data Extract Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple per table | multiple | Table | CSV File |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC executes Table extract using single spooler instance. |

QCSM-GUI

CSV

Oracle

Spooler

QC

Spooler

Spooler

This scenario illustrates common Event Flow and includes the following assumptions.

1. All output files are local to User QCSM Desktop.
2. Source table is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
3. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
4. CSV files are well formed with or without HEADER record. Header controlled by *–header* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.
6. Target spool directory ha to exists.
7. Existing file conflicts are ignored and old files having the same name are getting overwritten.

Table 4.4 Sharded Table Data Extract Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple per table | multiple | Table | CSV Files |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC shards source Table and executes file extract using multiple spooler instances. |

QCSM-GUI

CSV

Oracle

Spooler

QC

Spooler

Spooler

This scenario illustrates common Event Flow and includes the following assumptions.

1. All output files are local to User QCSM Desktop.
2. Source table is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
3. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
4. CSV files are well formed with or without HEADER record. Header controlled by *–header* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.
6. Target spool directory ha to exists.
7. Existing file conflicts are ignored and old files having the same name are getting overwritten.

Table 4.5 Table Data Copy Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| single | single per table | single | Table | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source Table data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Source

Table

Target

Table

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Temporary CSV file

This scenario illustrates common Event Flow and includes the following assumptions.

1. Output file is local to User QCSM Desktop.
2. It’s a single threaded load. Shard count I set by –*num\_of\_shards* arg.
3. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
4. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.6 Sharded Table Data Copy Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple | multiple | Table | Table |

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source Table data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Target

Table

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Sharded Source table

Temporary CSV file

Temporary CSV file

Temporary CSV file

This scenario illustrates common Event Flow and includes the following assumptions.

1. All output files are local to User QCSM Desktop.
2. Source table is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
3. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
4. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.5 Partition Data Copy Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| single | single | single | Partition | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads Partition metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source Partition data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Target

Table

Partition

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Temporary CSV file

This scenario illustrates common Event Flow and includes the following assumptions.

1. Output file is local to User QCSM Desktop.
2. It’s a single threaded load. Shard count I set by –*num\_of\_shards* arg.
3. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
4. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.6 Sharded Partition Data Copy Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple | multiple | Partition | Table |

Todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source Table data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Target

Table

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Sharded Source Partition

Temporary CSV file

Temporary CSV file

Temporary CSV file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads** | **Shards** | **Number of temp files** | **Source** | **Target** |
| multiple | multiple | multiple | Partition | Table |

This scenario illustrates common Event Flow and includes the following assumptions.

1. All output files are local to User QCSM Desktop.
2. Source **Partition** is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
3. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
4. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.5 Sub-PartitionData Copy Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| single | single | single | Sub-Partition | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads Partition metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source **Sub-Partition** data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Target

Table

Partition

Sub-Partition

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Partition

Temporary CSV file

This scenario illustrates common Event Flow and includes the following assumptions.

1. Output file is local to User QCSM Desktop.
2. It’s a single threaded load. Shard count I set by –*num\_of\_shards* arg.
3. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
4. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

**4.4.4 Sharded Sub-Partition Data Copy**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple | multiple | Sub-Partition | Table |

todo

Table 4.6 Sharded Sub-PartitionData Copy Events

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch table and column metadata. |
| 4 | QC extract data to CSV file(s) | QC extracts source Table data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Target

Table

Partition

Sub-Partition

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Sub-Partition

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Sub-Partition

Sub-Partition

Partition

Sharded Source Sub-Partition

Temporary CSV file

Temporary CSV file

Temporary CSV files

This scenario illustrates common Event Flow and includes the following assumptions.

1. All output files are local to User QCSM Desktop.
2. Source **Sub-Partition** is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
3. QC creates a queue to process table data shards. Spooler threads are pooled until all shards are extracted. Pool size is set by –*pool\_size* arg.
4. CSV file is well formed with or without HEADER record. Header controlled by *–header* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Query Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| single | single | single | Query File | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch Query columns metadata. |
| 4 | QC extract Query data to CSV file | QC extracts source Table data to temporary CSV file using single Spooler. |
| 5 | QC loads data to target Table | QC loads temporary CSV file to target Table using single Loader. |

Oracle

SQL

Query File

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Temporary CSV file

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. SQL Query File can contain only one SQL Statement.
3. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Multi-Query Parallel Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple | Multiple | Query Files | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch Query columns metadata. |
| 4 | QC extract Query files data to CSV files | QC extracts source Queries data to temporary CSV files using multiple spooler instances. |
| 5 | QC loads data to target Table | QC loads temporary CSV files to target Table using multiple Loaders. |

Oracle

Query Files

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Temporary CSV file

Temporary CSV file

Temporary CSV files

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. SQL Query Files can contain only one SQL Statement per file.
3. Source **Sub-Partition** is broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
4. QC creates a queue to export input Query files and then load temporary files. Spooler threads are pooled in parallel until all Queries are processed. Pool size is set by –*pool\_size* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.

Table 4.3 Multi-Query Sharded Parallel Load Job Execution Events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Shards | Number of temp files | Source | Target |
| multiple | multiple | Multiple | Query Files | Table |

todo

|  |  |  |
| --- | --- | --- |
| **Ref** | **Event** | **Event Details** |
| 1 | User submits a Job | User select Session record and clicks “Run” button. |
| 2 | QCSM opens QC-CLI window. | QCSM verifies argument values, parses them to well-formed bat command and executes it in CLI window. |
| 3 | QC reads metadata. | QC makes SQL request to source database to fetch Query columns metadata. |
| 4 | QC extract Query files data to CSV files | QC extracts source Queries data to temporary CSV files using multiple spooler instances. |
| 5 | QC loads data to target Table | QC loads temporary CSV files to target Table using multiple Loaders. |

Oracle

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Loader

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Spooler

|  |  |  |
| --- | --- | --- |
| 4 | QC extract data to CSV file(s) | QC extracts source Table using single thread. |

Sharded Query Files

Temporary CSV file

Temporary CSV file

Temporary CSV files

This scenario illustrates common Event Flow and includes the following assumptions.

1. All input files are local to User QCSM Desktop.
2. SQL Query Files can contain only one SQL Statement per file.
3. Query datasets are broken to logical shards. Shard count I set by –*num\_of\_shards* arg.
4. QC creates a queue to export input Query files and then load temporary files. Spooler threads are pooled in parallel until all Queries are processed. Pool size is set by –*pool\_size* arg.
5. Job status is updated in QCSM-GUI. “Run” button label will change to “SUCCES” or “FAILURE” depending on the QC job status.
6. **Data Extraction Formats.**

This section and Appendix C describe the detailed specification for the layout of data extracts.

Out of all available extract formats, QC is using CSV and JSON-LINE data formats to extract data from database. CSV is universal format, JSON-LINE is used in Ajax and noSQL database line MongoDB.

* 1. **Record Formats**

QCSM supports 2 row formats – CSV and JSON-LINE.

**4.1.1 The following rules apply to** **CSV format:**

1. The sequence of fields within each record is fixed.
2. Each field must be terminated by a delimiter, even if the field is max length.
3. None of the fields in the record can contain the character used for the delimiter.
4. The last field in a record is not required to be terminated by a delimiter
5. The delimiter is defined by *–field\_term* argument
6. Each field is positional. The order of the fields within each record is fixed.
7. Fields must be equal or less of the full length of the column specified in database.
8. Mandatory fields (NOT NULL) must contain appropriate value and be terminated by a delimiter.
9. Numeric fields may be padded with leading zeros.
10. Alphanumeric fields may be optionally padded with trailing blanks.
11. Date fields are of varied length and format and defined by NLS\* parameters.
12. Fields labeled as Numeric(n,m) must contain not more than **n** characters, including the decimal and must contain not more than **m** characters to the tight of the decimal.

4.1.2 **JSON-LINE format rules.**

1. String should be delimited with double quotation marks and use backslash escaping syntax.
2. Arrays should use square bracket notation with elements being coma separated.
3. Objects should be delimited with curly brackets and use comas to separate each pair.
4. Whitespace is ignored in JSON and should be avoided.
5. Comments are ignored.
6. Do not use any control characters in JSON.
7. In JSON-LINE each line in data file has to be well formed JSON value.
8. Line separator is “\n”
9. JSON data has to be saved with extension \*.json. Stream compression can be used.
   1. **File Header**

HEADER record is not required for CSV file to be loaded into database. There’s no trailer record for CSV file. The rest of the records may be included in any order.

Header record is used to identify names and order of columns, separated by a delimiter.

The following is assumed while processing header record:

1. Header consists only of one record and it’s always the first record in CSV file
2. Number of columns defined in a Header line is equal the number of columns in CSV file and target Table.
3. By default there’s no header in CSV output. User can add Header record by setting *–header* arg.
4. **TESTING**

QCSM provides some testing facilities for User Jobs.

Users transmitting information should test their jobs thoroughly before they start full-sized data load.

All new Sessions in QCSM are created will test presets for trial transmissions. As test succeeds User can remove test limitations and execute full-size load.

Testing does not alleviate User from responsibility to do manual checks on validity of data loads.

Every new session is created with the following presets.

1. Limit number of rows transferred using *–lame\_duck* arg.
2. Single threaded loads are controlled with --*pool\_size* arg.
3. Type of Debug messages printed in QC-CLI windows are changed using *–debug* arg.
4. ADDITIONAL INFORMATION

The following information is important for user to meet data synch requirements.

It provides specifications not covered elsewere and gives info on where to seek assistance for understanding these tech specs.

* 1. QSCM Executions and SUCCESS criteria.
  2. QC Job Cancellations.
  3. QC Job failures