QlikView

Incremental Load Scenarios

Overview

Incremental load is a very common task in relation to data bases. It can dramatically reduce the time needed to reload a QlikView application. This document presents some QlikView incremental load scenarios that can help you develop a strategy to design and develop the optimum reload approach for your QlikView deployment.

What is incremental load?

It is defined as loading only the new or changed records from the database. The remaining records should already be available, one way or another. With the help of QVD files it is possible to perform incremental load in most cases.

What is a QVD file?

A QVD (QlikView Data) file is a file containing a table of data exported from QlikView. QVD is a native QlikView format. It can only be written to and read from QlikView. The file format is optimized for speed when reading data from a QLikView script but it is also very compact. Reading data from a QVD file is typically 10-100 times faster than reading from other data sources.

What are QVD files good for?

QVD files can be used for many purposes. At least four major uses can be easily identified. In many cases two or more of them will be applicable at the same time.

Increasing Load Speed

By buffering non-changing or slowly-changing parts of input data in QVD files, script execution can become considerably faster for large data sets. For large data sets it will thus be easier to meet reload time-window limitations. When developing applications it is often necessary to run the script repeatedly. By using QVD buffering in such situations repeated waiting times can be reduced significantly even if the data set is not that large.

Decreasing Load on Database Servers

By buffering non-changing or slowly-changing parts of input data in QVD files, the amount of data fetched from external data sources can be greatly reduced. This reduces load on external databases and network traffic. When several QlikView scripts share the same data it is only necessary to load it once from the source database. The other applications can make use of the data from a QVD file.

Consolidating Data from Multiple QlikView Applications

Consolidation of data from multiple QlikView applications is possible with the help of QVD files. With the Binary script statement you can only load data from only one single QlikView application into another. With QVD files a QlikView script can combine data from any number of QlikView applications. This opens up possibilities e.g. for applications consolidating similar data from different business units etc.

Incremental Load

In many common cases the QVD functionality can be used to facilitate incremental load, i.e. only loading new records from a growing database.

QlikView

Incremental Load Scenarios

Basic incremental load process

1. Load new data from Database table (slow, but few records)
2. Load old data from QVD file (many records, but fast)
3. Create new QVD file
4. Repeat procedure for each table

The actual complexity of the solution depends on the conditions of the source database and the application requirements, but can be broken down into a few basic cases, as outlined below. Most incremental approaches focus on loading new and changed records, some can also address removing deleted records from QlikView that have been deleted from the source database.

1. Daily Incremental Reloads (Insert/Update)
2. Multi-QVD Incremental Reloads
3. Frequent Incremental Reloads (Insert Only) – Utilizes binary load approach
4. Historical data management using Daily/Monthly Stacked QVDs
5. Incremental Reloads (Insert/Update/Delete) with Deletion Flag available
6. Incremental Reloads (Insert/Update/Delete) with No Deletion Flag available

On the following pages you will find outlined solutions for each of these cases.

Scripting Techniques

After the scenarios are sample QlikView scripts illustrating these techniques.

Additional Resources

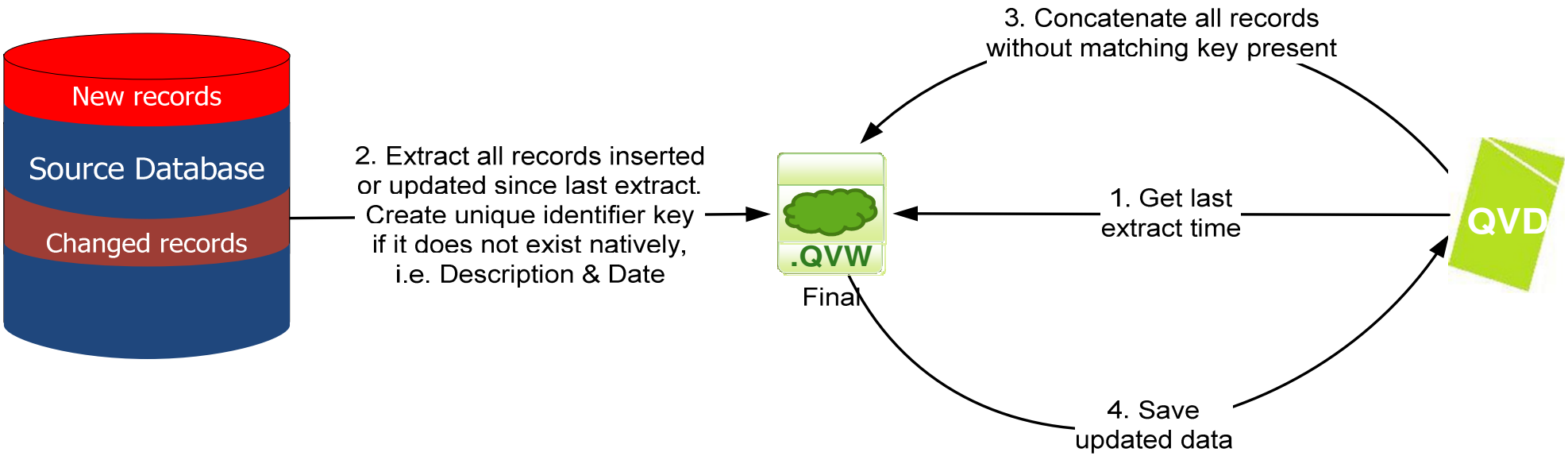
White Paper: How QlikTech uses QlikView - SalesForce.com
8QV10___Simple_Incremental_Load.pdf
IncrementalLoad.zip (Sample QVW and data file)

QlikView

Incremental Load Scenarios

Scenario #1 – Daily Incremental Reloads (Insert/Update)

On a daily basis, extract new and updated records (delta set) from source database. If no QVDs exist and also on a regular basis (such as weekly), extract all data from source database to create QVD files. Use native key field if available or create unique row key. Once the delta set is downloaded, the QVD that was created from the previous load is then loaded into the application. A WHERE NOT EXISTS clause is used to concatenate data from the current QVD to the delta set that was downloaded. If the Id of the delta set exists within the QVD, the assumption is that the data in the delta set is the relevant data and so any data within the QVD that matches the unique identifier in the delta set is not loaded. All additional data is concatenated to the delta set table. This approach requires an Modified Date or Create Date in source database tables.



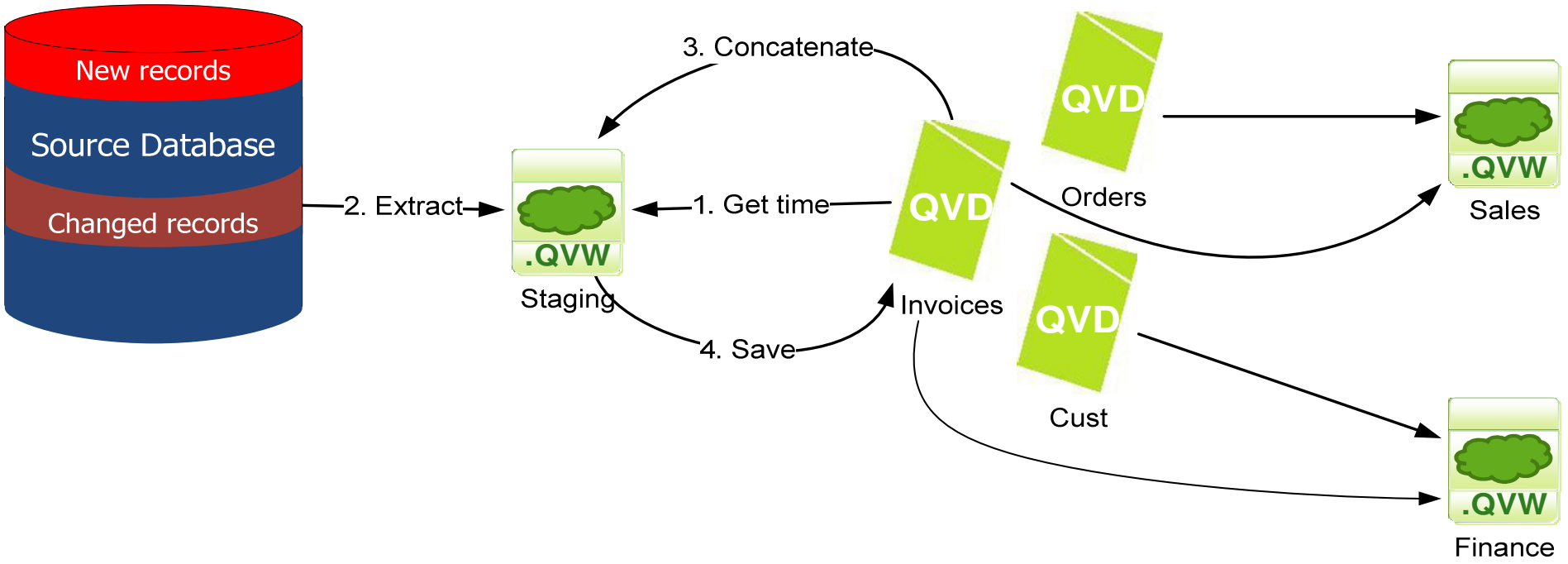
QlikView

Incremental Load Scenarios

Scenario #2 – Multi-QVD Incremental Reloads

Conceptually identical to previous scenario, but appropriate for enterprise applications sharing many data files. This approach has the incremental extract creating individual QVDs for later consumption. These individual extracts can be run in parallel by Publisher to reduce load time vs. a serial approach and to also allow multiple final applications to consume the produced QVDs.

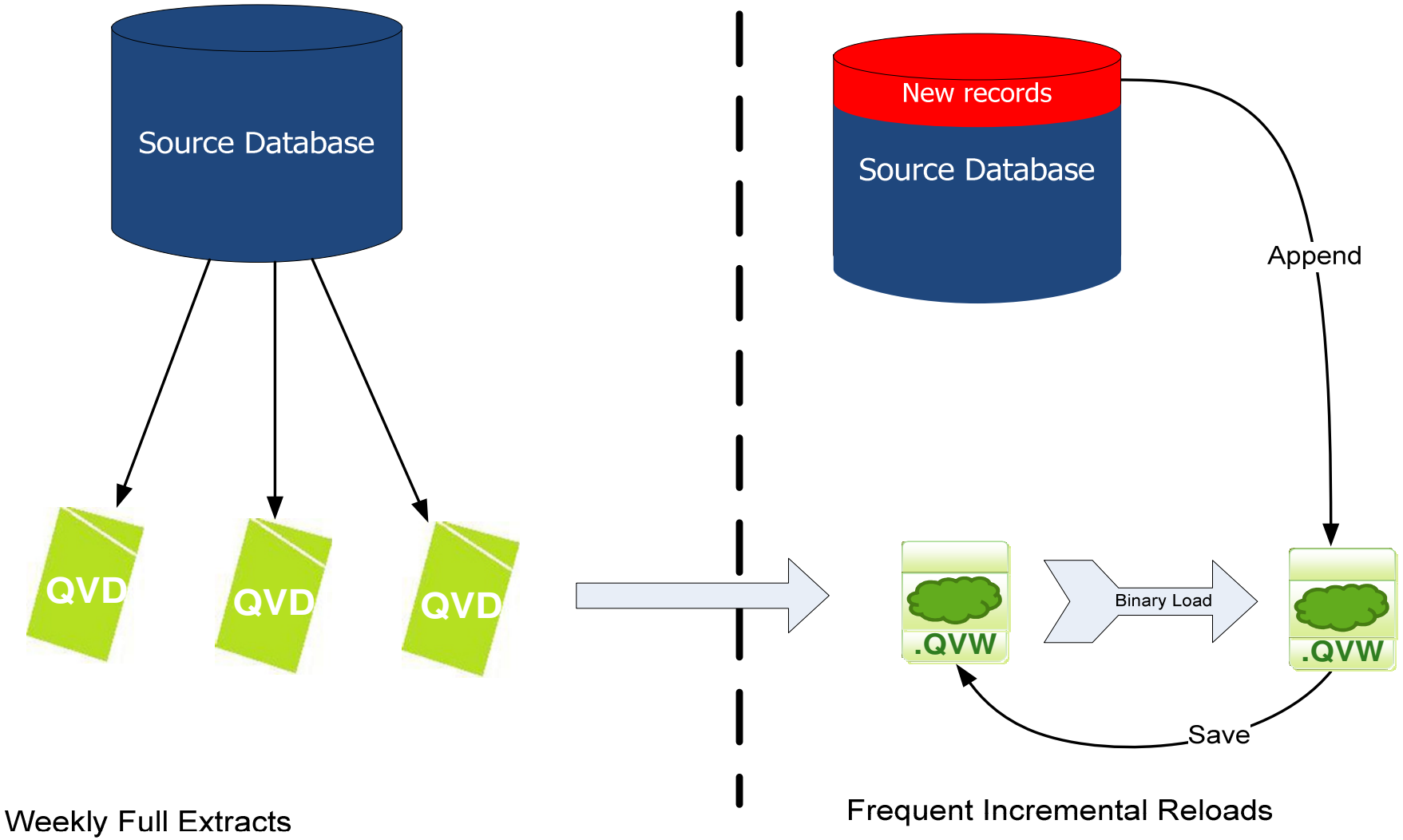
.



QlikView
Incremental
Load Scenarios

Scenario #3 – Frequent Incremental Reloads (Insert Only)

When required to frequently refresh data to stay current (hourly, every 10 minutes, etc), it may be preferred to binary load QVW file and supplement it with newly created records. This approach only requires a Create Date field in the source data.

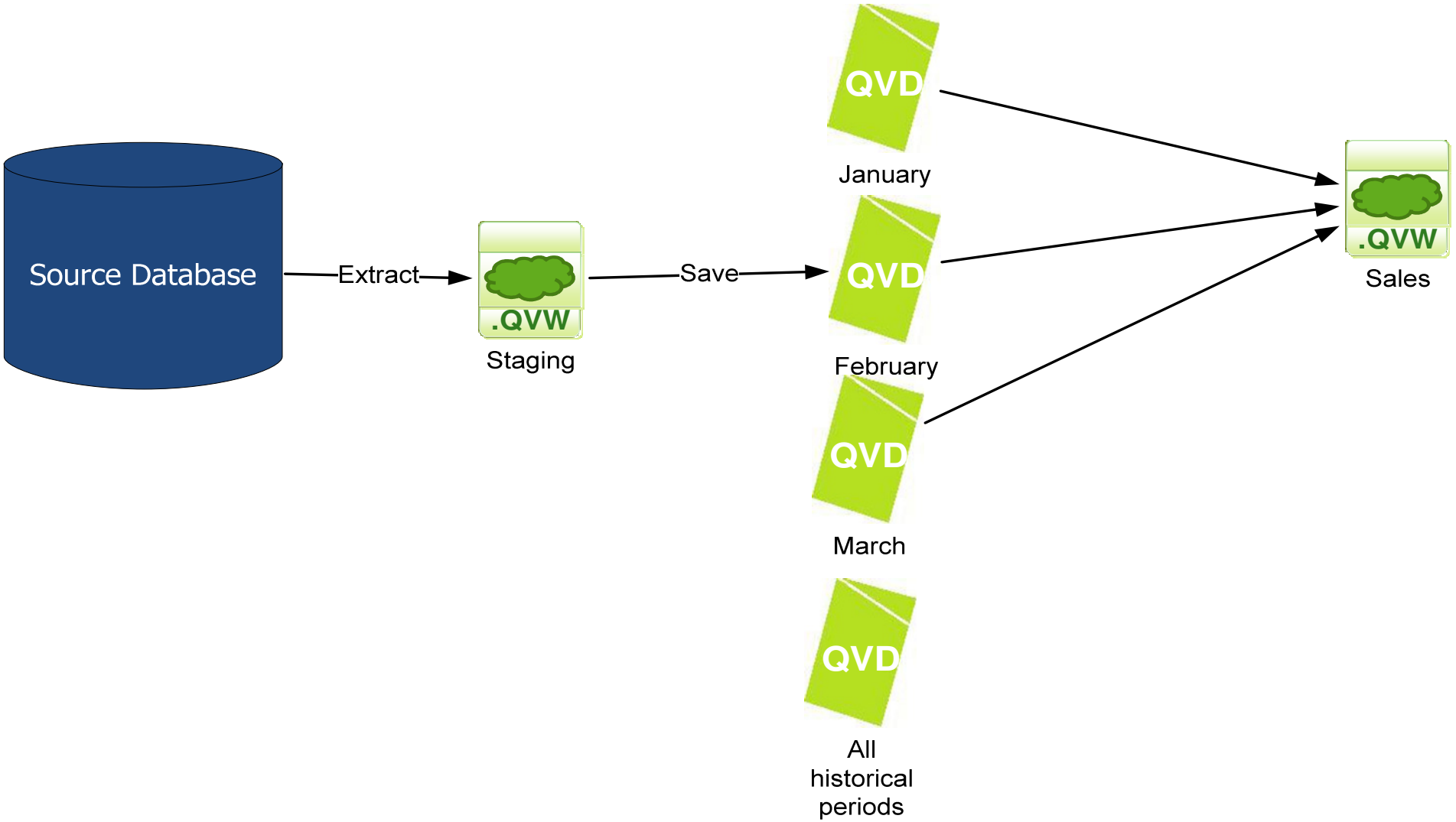


QlikView

Incremental Load Scenarios

Scenario #4 – Daily/Monthly Stacked QVDs

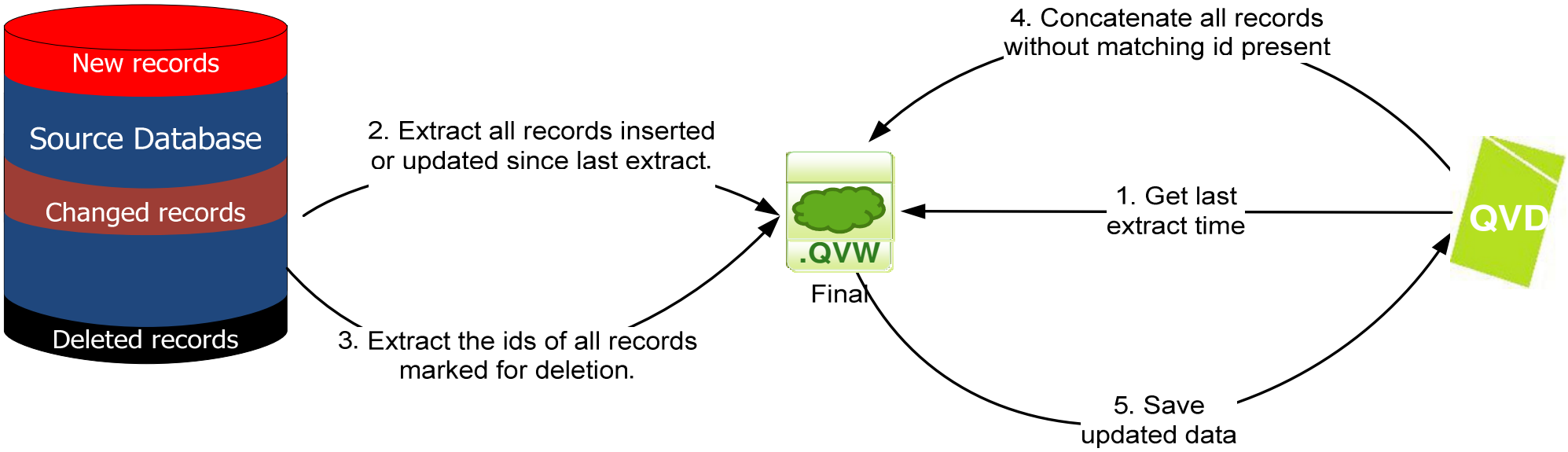
At the close of each day or month, extract and save source data to a distinct QVD file. The final application can be structured to load needed history records without going back to the source database. This approach allows maximum flexibility in managing data present in the final application. It also can be used to address data sources where there is no history directly kept by archiving daily/monthly records info QVDs.



QlikView
Incremental
Load Scenarios

Scenario #5 – Incremental Reloads (Insert/Update/Delete) Deletion Flag available

If deleted records are not physically deleted but only marked as deleted entries, then this approach can be used. It follows the previous approach, but also checks to make sure that records marked for deletion are not loaded from stored QVD.

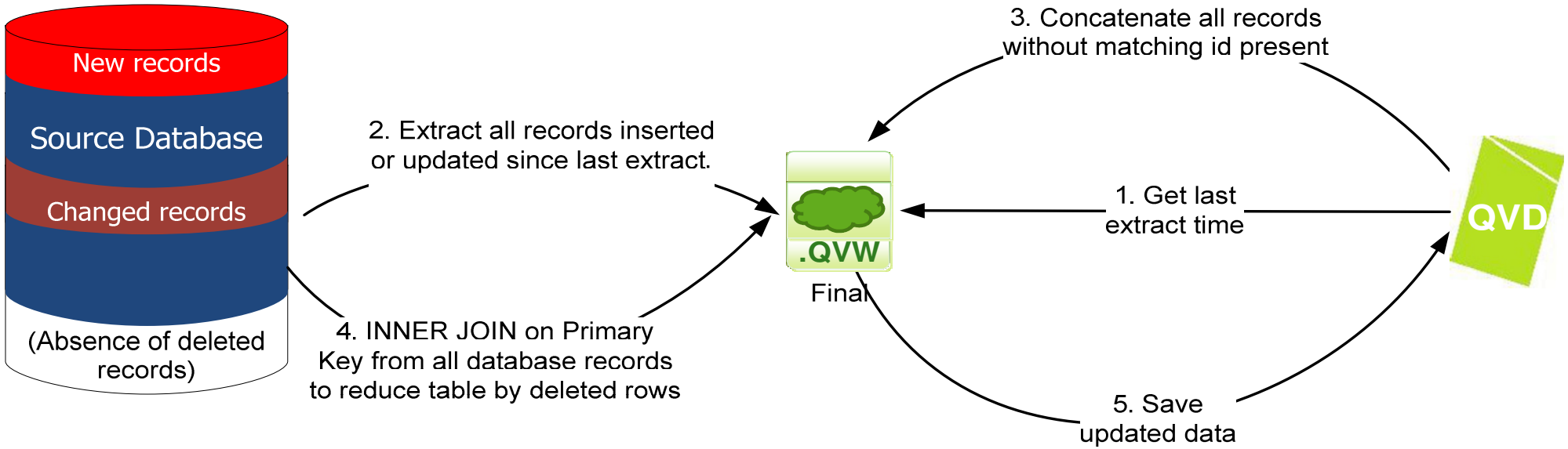


QlikView

Incremental Load Scenarios

Scenario #6 – Incremental Reloads (Insert/Update/Delete) No Deletion Flag available

If no Deletion Flag is available, then an extra step is needed to reduce the data set by no longer present records. First perform the DB extract and QVD concatenate, then take this combined set and INNER JOIN it against the source DB to remove those deleted records. The INNER JOIN only needs to evaluate the key field, not all fields.



QlikView

Incremental Load Scenarios

Basic incremental load process

Insert and Update. (No Delete)
The standard case is when data in previously loaded records may have been changed between script executions.
The conditions are as follows:
• The data source an be any database
• QlikView loads records inserted into the database or updated in the database after the last script execution
• A field ModificationDate (or similar) is required for QlikView to know which records are new.
• A primary key field is required for QlikView to sort out updated records from the QVD file.
• This solution will make the reading of the QVD file to be made in the standard mode rather than the super-fast mode. The end result will however still be considerably faster than if the entire database had to be read.
Script example:

```
LET vQVDPath = 'Qvd\'; // Set QVD storage Directory
LET vExecTime = UTC();
SET vLastExecTime = 0; // resetting vLastExecTime
// As long as a QVD already exists, find the latest timestamp
for modified records. This will be used to generate the delta
set.
if not isnull(QVDCreateTime('$(vQVDPath)<tablename>.qvd'))
then
LoadTime:
Load Max(LastModifiedDate) as LastModifiedDate
From $(vQVDPath)<tablename>.qvd (qvd);
Let vLastExecTime =peek('LastModifiedDate',0,'LoadTime');
Drop Table LoadTime;
end if
```

```
SQL SELECT Id,
…
FROM <tablename>
WHERE LastModifiedDate >=$(vLastExecTime) and
LastModifiedDate < $(vExecTime);
// Check to see if this is the first reload. If it is, skip this step
if Not isnull(QvdCreateTime('$(vQVDPath)<tablename>.qvd'))
then
Concatenate (<tablename>)
LOAD *
FROM $(vQVDPath)<tablename>.qvd (qvd)
WHERE Not(Exists (Id));
end if
//If data exists within table, store to QVD.
if NoOfRows('<tablename>') > 0 then
STORE <tablename> INTO $(vQVDPath)<tablename>.qvd;
Drop Table <tablename>;
end if
```

QlikView

Incremental Load Scenarios

Advanced incremental load process

Insert, Update and Delete

The most difficult case to handle is when records are actually deleted from the source database between script executions. The conditions are as follows:

• The data source an be any database
• QlikView loads records inserted into the database or updated in the database after the last script execution
• QlikView removes records deleted from the database after the last script execution

504

• A field ModificationDate (or similar) is required for QlikView to know which records are new.
• A primary key field is required for QlikView to sort out updated records from the QVD file.
• This solution will make the reading of the QVD file to be made in the standard mode rather than the super-fast mode. The end result will however still be considerably faster than if the entire database had to be read.

Simplified Script example:

```
LET ThisExecTime = Now();
QV_Table:
SQL SELECT PrimaryKey, X, Y FROM DB_TABLE
WHERE ModificationTime >= #$(LastExecTime)#
AND ModificationTime < #$(ThisExecTime)#;
CONCATENATE
LOAD PrimaryKey, X, Y FROM File.QVD
WHERE NOT EXISTS(Primary Key);
Inner Join SQL SELECT PrimaryKey FROM DB_TABLE;
If ScriptErrorCount = 0 then
STORE QV_Table INTO File.QVD;
Let LastExecTime = ThisExecTime;
End If
```