



A REVELATION SOFTWARE LTD

**A quick look at the Data Warehouse
facility in OpenInsight**



CONTENTS

DATA WAREHOUSING	3
PREPARING TO WAREHOUSE LINEAR HASH DATA	4
CREATING A WAREHOUSE PROCEDURE	6
Procedure: To Start and Setup Options for a Warehouse Procedure	6
PROCEDURE: TO CREATE THE PRIMARY TABLE AND SCRIPTS	8
PROCEDURE: TO CREATE THE SECONDARY TABLE(S)	10
PROCEDURE: TO CREATE THE WAREHOUSE PROCEDURE AND DATASETS	12
EXECUTING A WAREHOUSE PROCEDURE	13
DATA WAREHOUSING UPDATES-ONLY	14
SAMPLE WAREHOUSING APPLICATION	14

DATA WAREHOUSING

This paper has been produced to enable the reader to fully understand the Data Warehouse functionality in OpenInsight. The information contained herein is taken directly from the help files in OpenInsight version 7.2.1 and the reader is advised to make reference to the help files in the version of OpenInsight that the reader is using.

Data warehousing describes the process of depositing all of an organizations data to a single location known as a warehouse. Most organizations have data in a variety of applications and databases, and although the data is often interrelated, it is not all accessible from the same place or from a single tool. Data warehousing solves this by extracting specific data from each application and storing it in a data warehouse; the data can then be queried by decision support tools, reporting applications, and other analysis tools.

Data warehousing tools are also useful for archiving a snapshot of an applications data, exporting data to a client/server database, and assisting in the migration of data from one database to another.

The data warehousing tools in OpenInsight for Workgroups, the Create Warehouse Procedure, Warehouse Manager and the Configure Warehouse tools, support application developers and information technology specialists as they build or contribute to departmental and corporate data warehouses. The data warehousing tools assist in relational database design, and provide reusable DataSet objects, and automatically generated Warehouse Procedures. The warehouse procedures contain the logic for constructing the relational database tables, maintaining those tables, and warehousing OpenInsight native table data into those tables.

PREPARING TO WAREHOUSE LINEAR HASH DATA

Prior to warehousing your data you need to:

- 👁 understand your data
- 👁 know your target database
- 👁 learn about the OpenInsight warehousing tools

First of all, you have to know what data you want to warehouse. Depending on the reason for warehousing the data, you may be extracting sparse or summary information from your OpenInsight native tables, or you may be warehousing every single bit of information. Secondly, the OpenInsight native table dictionary contains a great deal of information about your data, including its data type, a description of the data, whether or not it is multi-valued, etc. Unfortunately, many dictionaries are not kept up to date or were never fully designed, so that information may be either incorrect or incomplete. The dictionary is used by the warehousing tools to look for hints about how to warehouse the data. If the dictionary is not correct, it will make the process of warehousing much more difficult. Make the dictionary complete and correct; verify that all multi-valued fields are marked as multi-valued; and make sure that the data type assigned to each field is correct.

With regard to the target database, first, you must know the syntax for creating, dropping (deleting), and clearing tables. For example, almost all client/server database engines still require their own proprietary syntax for defining tables, although the ANSI SQL standard has helped standardize the syntax somewhat. Second, you should know the data types that are supported by the database engine in order to provide appropriate target data types for native table fields. These data types are used in the table creation script. Also, determine the DataSet type that corresponds to each of the database engines data types. DataSets are automatically created by the warehousing tools to be used by the Warehouse Procedures to warehouse the data. Several objects in OpenInsight use the DataSet types, which are based on the SQL Access Groups (SAG) Call Level Interface (CLI) type definitions. (ODBC is an implementation of the SAG CLI, which is one of the reasons for standardizing on these data types.) The objects that use the DataSet types are: The Connection Object, the Query Object, and the DataSet object.

The *Query Tool* in OpenInsight for Workgroups *Client/Server Workspace* is an excellent resource for trying out table creation syntax and determining type mappings. For example, if you try creating a table in Microsoft Access, you will find that the primary key clause is not supported and the int or integer data type is not supported; try using the number type instead. After creating the table, choose *List Objects* from the *Query* menu, select the table you created, and then choose *List Columns* from the *Query* menu to see the columns with their corresponding DataSet and native table types. In the Microsoft Access example, for the Access number type, you will see that the DataSet type is a DOUBLE (an 8-byte floating point number), and the native table type is FLOAT.

Third, if the target database uses transactions, find out if tables can be created and dropped within transactions. Some databases, including certain versions of SQL Server, don't allow table creation within a transaction. Also find out how mass deletes affect the databases transaction management. For example, instead of using `delete * from <table>` to clear a SQL Server table, you can issue a `truncate <table>` statement, which is not transaction-logged. This keeps the transaction log from getting too big, (a problem which is easy to encounter on SQL Server if you are doing mass operations

The OpenInsight data warehousing tools include the *Create Warehouse Procedure* wizard and the *Warehouse Manager*. Both tools are accessed from the *Tools* menu in the *Client/Server Workspace*.

The *Create Warehouse Procedure* wizard steps you through a series of screens that prompt you for the input required to create a warehouse procedure, a BASIC+ function containing the logic to warehouse a native table. Included in this code is the ability to create, clear, and drop the necessary

warehouse tables, to select native table information to warehouse, and to perform the actual process of data warehousing. The warehouse procedure knows how to extract data from one native table and store that information in one or more relational tables in the data warehouse. You may need to create one warehouse procedure or several. The *Warehouse Manager* is then used to select the warehouse procedures you want to execute.



CREATING A WAREHOUSE PROCEDURE

The *Create Warehouse Procedure* is a wizard that prompts you for information that is used to generate the code and DataSets necessary to convert linear hash data to relational data. The wizard process is divided into the following four procedures. All four procedures must be performed at one time to create a warehouse procedure. Conceptual information is included in these procedures to explain the reasons for specifying the required data inputs.

Procedure: To Start and Setup Options for a Warehouse Procedure

In this procedure you specify options which are used throughout the process of using the warehouse wizard.

1. Choose *Create Warehouse Procedure* from the Client/Server Workspace *Tools* menu to start the wizard.
2. Select the name of the linear hash table to be warehoused from the drop-down list. The tables listed are all the attached non-system tables that have dictionaries.
3. Click the **OPTIONS** button to set up case sensitivity to be used during this procedure. This is the first of the *Options* page. Most databases work with lower case, but some older database engines require uppercase to be used. If you don't know, use lower case. Every time a name is generated from now on, it will use the case you specified here, but you can always change the case of the name (or the name itself) manually.
4. Click the **NEXT** button to display the second *Options* page which specifies the defaults for LH (OpenInsight native tables) data types, including the corresponding DataSet data types, SQL data types, LH to DataSet type conversions, LH to DataSet argument type conversions, and DataSet argument quoting defaults. For more information, refer to the topics on DataSets earlier in this chapter.

The LH data types are supplied for you; everything else can be modified. It is important that the information in this window be correct as it is used to generate the definitions of the warehouse SQL tables and the DataSets used to populate those tables. Define the SQL data type first, then the DataSet type.

As an example, let's say that the LH data type is VARCHAR. Depending on the target database, the SQL data type may be char, varchar, string, text, memo, or any of a number of proprietary data type names. If the target data type requires a length to be specified, like char(20), then provide a default length. (You will be able to provide the exact length when you define individual columns later in this procedure.) Then provide the DataSet type based on the target data type. In this example, the DataSet type would be CHAR(20) as well. The DataSet may be used to stored data from a SQL table (for example, during an update), therefore its type is based on the SQL data type and not the LH data type.

The Conversion column (displayed in the figure below) specifies the conversion (used by the BASIC+ `fmt()` or `oconv()` functions) necessary to turn the native table data into a format that the DataSet understands. Typically, no conversion is necessary, since the DataSet by default expects values in OpenInsight format. The main exception to the rule is masked decimal types. The DataSet expects the data to be passed without a masked decimal. For example, the money value \$1.55 would be stored in masked decimal (MD2) as 155, and the money value £20.00 would be stored as 2000. To remove the masked decimal, those values must be converted using the MD2 conversion, resulting in 1.55 and 20.00 respectively.

With regard to the Arg Conversion and Arg Quoted columns, DataSet arguments are direct string substitutions, and are used primarily for executing the DataSet (retrieving values *from* the target database). The arguments are used to retrieve one records worth of information from a SQL database. Using a purchase order example picture the structure of the LH table that stores purchase order information with columns for position, single or multi-value data, and field name.

The key is the purchase order number; there is both single-valued and multi-valued information and one symbolic. Assuming a similar structure for the data warehouse, there probably will be two relational tables: one for purchase orders and one for purchase order lines. To select all of the purchase order information corresponding to purchase order 17, the SQL statements have to contain something like where po_no = 17. Assuming the tables are named purchase_orders and purchase_order_lines, the scripts would look like

```
select * from purchase_orders where po_no = 17  
  
select * from purchase_order_lines where po_no = 17
```

The number 17 is called an argument because it is the value needed in the script to specify some particular search criteria. If that value needs to be quoted (for example, if it is a character type), then the Arg Quoted column needs to be set to Yes, otherwise to No. Likewise, if the value in the LH table needs to be formatted to result in that argument value, specify the conversion in the Arg Conversion column. In addition to masked decimals, the LH types that typically have to be converted are date, time, and date/time types, since it is unlikely that a target database understands dates and times in the OpenInsight internal format

Do not assume that the defaults on this window are correct. Know and understand your target databases data types. Some of this information is available dynamically from an OpenInsight Connection Object using the low level XO API function XOGetInfo. For more information, refer to Connection Object topics earlier in this chapter and the DSXO_API insert record.

Click the FINISH button when all defaults have been set and verified.

PROCEDURE: TO CREATE THE PRIMARY TABLE AND SCRIPTS

In this procedure the primary table, which contains single-value data, is defined and scripts are automatically generated which you need to verify.

1. Click the NEXT button to continue creating the Warehouse Procedure and display the second page of the wizard.
2. Specify the number of tables that data is to be warehouse to. Specify one table for single-value LH data and one additional table for each associated multi-value (AMV) group of data. Using the Purchase Order example, there is one AMV group (including the entire purchase order line value column) so the number entered here is two.
3. Click the NEXT button to continue and specify key information.

The primary table refers to the table that stores the single-valued data; in the example, purchase order information. The primary key for the primary table refers to the column or columns that uniquely identify a row in that table. Further more, the primary key for the primary table is part of the key for each of the tables storing multi-value data. In the example, this is the purchase order number (PO_NO). If the information is not correct, you could add, remove, and re-order the LH fields to define the correct key. Also, if you want more details about any one of the LH fields, select the field and click the DETAILS button

The first, second, third and fifth columns, (Pos, S/M, Field Name, and LH Type), are filled in for you and cannot be changed. The Pos column tells you the position in the dictionary; 0 denotes a key field, for example 0*0 is the full key, and 0*1 is the first part of a multi-part key. The second column is S for single-valued and M for multi-valued. The third column is the name of the LH field, and the fifth column is the LH data type for the field.

The fourth column is the name for the column of the SQL table that will store the value from this field. The SQL column name is based on the LH field name, with the upper/lower/mixed case option applied. The eighth column specifies whether or not the column in the SQL table can contain nulls. (Since key columns can't contain nulls, this is defaulted to No for you and cannot be changed.)

The rest of the columns (DataSet type, SQL type, Conversion, Arg Conversion, Arg Quoted) are defaulted from the information specified in the *Warehouse Options* wizard. You should make sure that this information is correct, because it will be used as the basis for all of the SQL tables created to warehouse the data from this LH table.

4. Click the next button to continue. The next page displays two undefined SQL tables.
5. Select the first undefined SQL table then click the DEFINE button. Each SQL table has to be defined individually; you are now starting the process of defining the first SQL table. When you are finished defining this table, you will return to the page in the figure above to select the second undefined table and complete its definition process. Notice that the default SQL table name is based on the LH table name and the case option you supplied. The first table defined stores the single-valued information.

6. Accept or change the default table name then click the NEXT button to display the key information for the first SQL table.
7. Accept the default key then click the NEXT button to continue. The wizard knows you are defining the SQL table for the single-value columns of the LH table.
8. Click the ADD button to display single-value fields.
9. Select the single-value fields to be warehoused, then click the OK button.

Again, verify that the data types and conversions are correct. Since these are data fields, you can also specify whether nulls are allowed. In the purchase order example, the purchase order date cannot be null.

10. Click the NEXT button to display the scripts that are automatically generated.

Verify all scripts.

There are seven scripts that are created: Create Table, Clear Table, Drop Table, Select Rows, Insert Row, Update Row, and Delete Row. The last four scripts are very familiar to developers who use the DataSet designer; these four scripts are the basis for the DataSet definition. The other three scripts are used in the Warehouse Procedure to create, clear, and drop the SQL tables.

This is a very important step to verify, since the SQL syntax varies from database to database. For example, neither the null/not null nor the primary key clauses are supported by Microsoft Access. SQL Server has its own peculiarity - it likes primary keys to be defined after the table is created by using the Transact-SQL stored procedure `sp_primarykey`.

The select statement has a where clause with a colon and then the argument `po_no_arg`. The arguments start with a colon and (in this wizard) are all suffixed with `_arg`. If you had specified that the argument should be quoted, you would see `:po_no_arg` instead of `:po_no_arg`. When the select script is executed, the entire argument (`:po_no_arg`) is replaced with the value of the argument, for example 17.

The insert script uses similar syntax. When the DataSet is created by the wizard, it has the same column names as the SQL table does, (it is less confusing that way). For example, `po_no`, `po_date`, and `po_desc`. When the insert script is executed by the DataSet, it fills in `:po_no`, `:po_date`, and `:po_desc` from the column values of the row being inserted. When this script is executed, it looks something like:

```
insert into purchase_orders
(
  po_no, po_date, po_desc
)
values
(
  17, 30 Mar 1997, We need a new expensive jacuzzi
)
```

11. Click the FINISH button to end the definition of the primary SQL table.

PROCEDURE: TO CREATE THE SECONDARY TABLE(S)

The secondary table contains your multi-value data. If you have more than one group of AMV data, you will have to perform this part of the wizard process for each group.

1. Now you must define the other SQL table(s). These are the tables that are to store the multi-value data.
2. Select the remaining undefined table and press the DEFINE button.

The wizard always defaults the SQL table name to the name of the LH table with the case option applied. As this is the second SQL table, provide a different name for this table. In the example, this is changed to `purchase_order_lines`, a name that reflects what is to be warehoused in this table.

3. Click the NEXT button to continue the definition of the second SQL table.

This table is to contain AMV group data (in the example, purchase order lines), so you need to build a key that uniquely identifies a particular value (a particular purchase order line) in the multi-value group. In the example, the purchase order number (`PO_NO`) is used, as it is the primary key of the primary table. Next, the multi-valued field called `PO_LINE` that uniquely identifies each line, is added to the key.

What if there is no multi-value data field like the `PO_LINE` that uniquely identifies each value? For example, without the `PO_LINE` column, what if the first purchase order line is the *first* purchase order line only because it's the first value in the AMV group?

Uniquely identifying some value by where it appears in a table (or an AMV group, which is like a table within a table) is called identification by position. A relational database does not support identification by position, so to provide that information in a relational table, you need to have a column in the SQL table that contains the position (1, 2, 3, ...) of that row. In LH, this is called `@MV`, so instead of selecting the `PO_LINE` field, you would have pressed the ADD `@MV` button. By doing so, you are telling the wizard to create an additional counter column, which is automatically incremented (by the Warehouse Procedure) for every purchase order line on any given purchase order.

4. After the key has been defined for the second SQL table, click the NEXT button to continue.
5. Click the ADD button to display the fields that can be warehoused in this table. Notice that there are both single- and multi-valued fields. Select the multi-valued fields that contain the appropriate information (in the example, the purchase order line information). Don't include the symbolic field, because it is simple enough to do that calculation in SQL.
6. Click the NEXT button to display the automatically generated scripts.

Note: The *Select Rows* script only checks for part of the key (for example, the purchase order number). The entire key is not used in the where clause. The purpose of the *Select Rows* script is to select all (and only) those rows from the table that correspond to one LH record (in the example, a purchase order). The script correctly retrieves all of the AMV data for a particular purchase order. It knows to search only the columns from the primary key of the primary table, since those columns are (by default) common to each SQL table for the LH table.

7. Verify that the scripts are correct, click the FINISH button and return to the *Create Warehouse Procedure* wizard and see that both tables are defined.

In the next procedure the wizard end products, the warehouse stored procedure and two DataSets, are created.

PROCEDURE: TO CREATE THE WAREHOUSE PROCEDURE AND DATASETS

1. Click the NEXT button to continue. The default name for the Warehouse Procedure is the name of the table being warehoused.
2. Click the FINISH button and the wizard creates a BASIC+ function called WP_xxx where xxx is the name you supplied for the Warehouse Procedure. In addition, a DataSet object is created for each warehoused SQL table. The name of the DataSet object is WP_xxx where xxx is the name of the SQL table. In the example, the following entities are created:

WP_PURCHASE_ORDERS - a stored procedure

WP_PURCHASE_ORDERS - a DataSet, and

WP_PURCHASE_ORDER_LINES - a DataSet.

These end products of the Create Warehouse Procedure wizard are customizable. In fact, for complex data structures, like sub-multi-values, which aren't covered by the wizard, you will have to customize the resulting warehouse procedure.

Tip: One way you may be able to avoid changing the Warehouse Procedure for more complicated data structures is by setting up symbolic fields in the dictionary which are designed to massage the data into standard multi-value groups to simplify warehousing.

For more information about data warehousing specifics related to SQL Server, refer to the white paper on this topic found in the Knowledge Base at the Revelation Software web site: www.revelation.com.

EXECUTING A WAREHOUSE PROCEDURE

Warehouse Procedures are executed using the Warehouse Manager in the Client/Server Workspace.

1. To start the *Warehouse Manager* choose *Warehouse Manager* from the Client/Server Workspace *Tools* menu.
2. Select the warehouse procedures to process then click the NEXT button.
3. Choose the appropriate option for coordinating data warehousing activities with other workstations, and then click the NEXT button.
4. Select the *Yes, please create...* option to create the SQL tables if necessary, then click the NEXT button.
5. Select an option specifying how the warehouse tables are cleared before executing a Warehouse Procedure, then click the NEXT button.
6. Click the CONNECT button to connect to a SQL database.
7. The *Connect to Data Source* dialog box, shown below, is displayed. Select the data source where the data warehouse resides and provide any additional information necessary to establish a connection. For more information, refer to Connection topics earlier in this chapter. Click the OK button to establish the connection.
8. When the connection has been established the data source name is entered as shown in the figure below. Click the NEXT button to continue.
9. When the established connection uses transactions and you need to create tables, but the database containing the warehouse doesn't allow tables to be created or dropped within transactions, you need to create a second, non-transacted connection. This second connection is used to create and drop the tables. Select this option as shown in the figure below, then click the NEXT button.
10. Select a scheduling option. The default is the *Default to the last used schedule* option. The second option requires an interval. When specifying the interval use the format: hh:mm:ss (hours, minutes, seconds).
11. Click the FINISH button to continue. The Warehouse wizard is closed and the Warehouse Manager is displayed.
12. The Warehouse Managers scheduling window is displayed. Click the START button.
13. This queues up the Warehouse Procedure to be processed and displays the waiting window.
14. When the time comes to run the next scheduled Warehouse Procedure, the running page is displayed. This page shows the progress of the processing of the Warehouse Procedure. Processing can be paused or stopped by clicking the appropriate button. If you want the currently executing warehouse procedure to complete and then want to halt processing, select the *Stop warehousing after this procedure completes* option. The STOP button on the running page interrupts the warehouse procedure.

Warning: Using the STOP button can leave the data in a partially warehoused state. Using the STOP button with live data is not recommended.

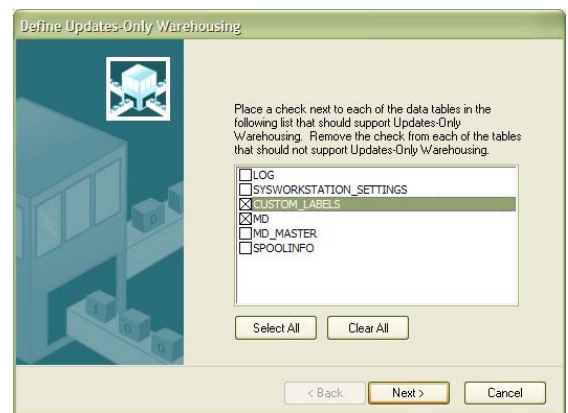
15. When the processing has completed, you are returned to the scheduling. Click the REVIEW LOG button to review any errors that occurred during processing.
16. Click the RETURN button to close the log. Click the EXIT button on the scheduling page to exit the Warehouse Manager.

DATA WAREHOUSING UPDATES-ONLY

OpenInsight can update a data warehouse with only those LH records which have changed; this is referred to as *Updates-Only Warehousing*.

To configure *Updates-Only Warehousing*, choose *Configure Warehouse* from the Client/Server Workspace *Tools* menu to launch the *Define Updates-Only Warehouse* wizard. This wizard is used to select the linear hash tables for which you want to warehouse only updates.

In addition, the *Changes only* option has been enabled in the *Warehouse Procedure* scheduling window, which is accessed from the SCHEDULE button in the Warehouse Manager wizard.



SAMPLE WAREHOUSING APPLICATION

A sample warehousing application based on a linear hash database has been added to the existing EXAMPLES application. This sample includes three tables (PURCHASE_ORDERS, STOCK, and RECEIPTS), three forms (PO_ENTRY, STK_ENTRY, and RCPT_ENTRY), three Warehouse Procedures created by the warehousing wizard (WP_PURCHASE_ORDERS, WP_STOCK, and WP_RECEIPTS), and five DataSet Objects created by the warehousing wizard (WP_PURCHASE_ORDERS, WP_PURCHASE_ORDER_LINES, WP_STOCK, WP_RECEIPTS, WP_RECEIPT_LINES). The sample warehousing is designed to be used with the SQL Server direct Connection Object, as described by the white paper covering SQL Server warehousing. This white paper can be found at the Revelation web site: www.revelation.com.