

Data Archiving Strategies



Grid-Tools
The power of test data

Specialists in data creation, data masking and test data management

Contents

Archiving Strategies.....3

Data Archive - Architecture overview.....5

Designing your collection criteria.....8

Data Archive Advantages.....9

The Data Archive process.....11

Standard Format.....17

Archiving Strategies

Many databases, especially transactional systems, contain vast amounts of data that can be considered historical. No matter how much database tuning, disk and processing power you throw at databases there is a direct correlation between the amount of data and the performance.

It is important when designing new systems or trying to archive from existing applications that “Information Lifecycle Management” is considered. In addition new technology is now available to significantly improve the robustness of the archive process and accessibility of the archived data.

Archiving data from databases can be a complex process and should be approached with care. There are numerous methods to extract the data and then remove (purge) the data from the database. Some of the more popular methods used are as follows:

1. Take a physical backup of your database prior to the purge then issue deletes against the database.
2. Use database utilities to extract data to proprietary extract files and then issue deletes against the database.
3. Create copy tables within the database and move data to the copy tables and then issue deletes against the live tables.
4. Write bespoke extract programs to dump the data to flat file in an open format, ASCII for example and delete the data as the extracts run.
5. Run standard reports for the period to be deleted, store these offline and then issue deletes against the data.

When archiving data it is important to consider:

1. Access requirements to the archived data including: legal issues; data warehousing and add hoc requests.
2. Standard reporting, do you need to be able to run reports that include history data?
3. How often data needs to be purged.
4. How often data needs to be restored.
5. Can the application function if transaction data is removed?

6. How stable is the database model, are version changes very common and if so how relevant is the archived data in the old table format?
7. How easy is it to identify which data is historical, for example do you have chains of customer orders that cross date boundaries?
8. How complex are the table relationships, for example: are there numerous foreign keys? Such that data has to be removed in a particular sequence.

Once you have gathered information about the application you can begin designing your archiving strategy. Data Archive provides a straight forward and secure method of archiving.

Data Archive - Architecture overview

Data Archive offers multiple methods of archiving data. You may decide to use several of the different architectural approaches as appropriate for your data as it moves through the “Information Lifecycle”. Each of the architectures is summarized below with a detailed description of each architecture following.

- Application Transparency

Allows you to relocate data within your database to a separate schema. The combined data can be accessed using a generated application transparency layer. Users can use existing reports and applications to view the joined live and history schema together.

- BridgeHead’s Filestore

Transactional data and associated reference data is extracted into “encapsulated transaction” files. These files are passed to Bridgehead’s Filestore product which securely archives the extracted data. Once the data is secured by Filestore it is removed from the production database.

The archived data is indexed and can be selectively restored as required.

- Sand Searchable Archive

Data is relocated to the Sand Searchable Archive from the production database. Once the data has been relocated the data is removed from production. Archived data can be queried and accessed via SQL using ODBC and JDBC drivers.

The archived data is indexed automatically in the Sand Archive and can be selectively restored as required. The joined live and history can also be viewed using reports based on “Federated data access.”

- Mobius ViewDirect

Transactional data and associated reference data is extracted into “encapsulated transaction” files. These files are passed to Mobius’s ViewDirect which securely archives the extracted data using generated input and output policies.

Once the data is secured by ViewDirect it is removed from the production database.

The archived data is indexed and can be selectively restored as required. This technique is useful as an extension to VDR clients who already archive application reports.

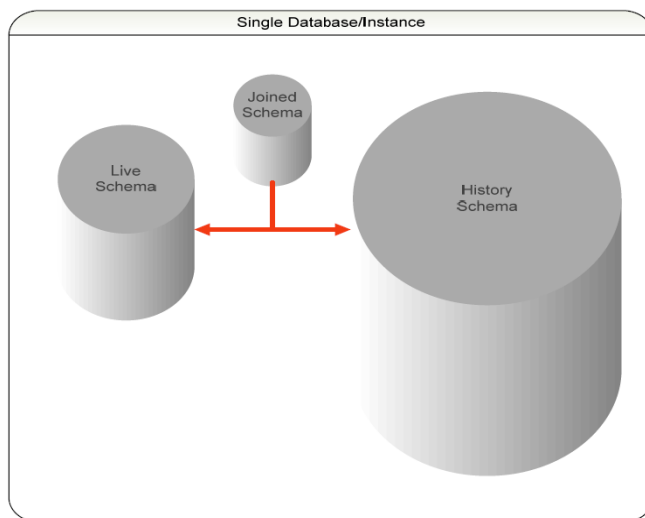
• **Application transparency**

The Data Archive architecture is designed to store your history data in another schema within your database. The data is copied to the other schema prior to being purged.

In a typical application the top 5% of tables store 95% of the data.

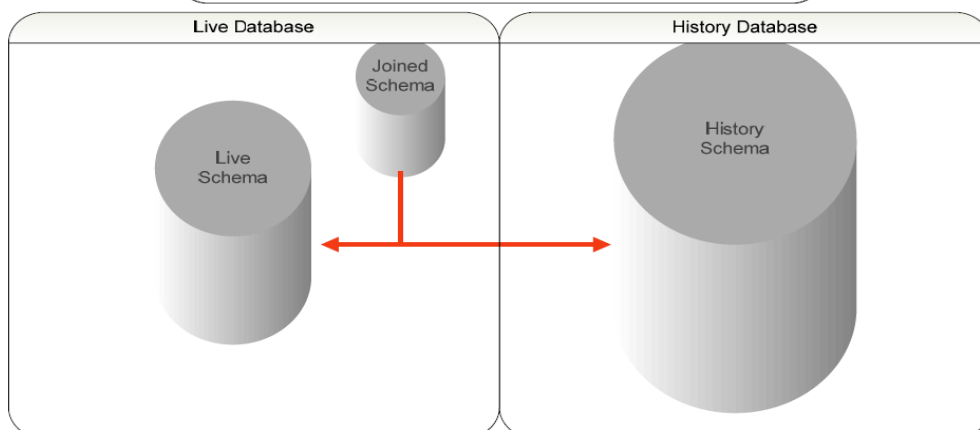
Only tables that are going to be archived are copied into the history schema, NOT ALL tables are copied. Reference, Work and Temporary tables remain in production only.

In Live	In History	Type of Table
CUSTOMER		Key table
ORDER	ORDER	Transaction table
ITEM	ITEM	Transaction table
LEDGER	LEDGER	Transaction table
STATE_TAX		Reference table
PO_PROCESS		Temporary
6 Tables in Live	3 Tables in History	



Views are created that join together the tables in Live and History such that a combined view of the data is provided for reporting and inquiry.

The joined schema definitions can be in the same database or instance or in a remote instance depending on the underlying RDBMS technology. The history tables can be located via device or tablespace definitions on more cost effective disk.



The history schema can be located in the same database (instance) or in a remote database.

Designing your collection criteria

To decide which rows are to be archived use the front end to explore the data and verify any assumptions you have made. In addition you may populate the history schema and verify the data is correct prior to purging the data.

Quite often dates or status codes can be used to identify which rows are candidates however sometimes more complex pre selection may be required. If this is the case it is recommended that you build a set of candidate archiving policies these policies can then be added at run time to pre-select the list of candidate rows to be archive.

In some circumstances if your application is bespoke you may add columns to key tables, for example ARCHIVE_IDENT, this column can then be used to drive the selection.

Once you have decided on your criteria add the appropriate SQL to the driving table, you may for example wish to use system date minus 2 years as the driving criteria to build your extract definitions.

If you do not have any documentation for your application or you are unfamiliar with it's operation you may wish to use GTPolicy which will explore the data build a central repository of relationships. GTPolicy can extract relationships from application code, naming standard, database constraints as well as allowing direct entry. All relationships can be verified against the database to verify the integrity of the data and of the relationship.

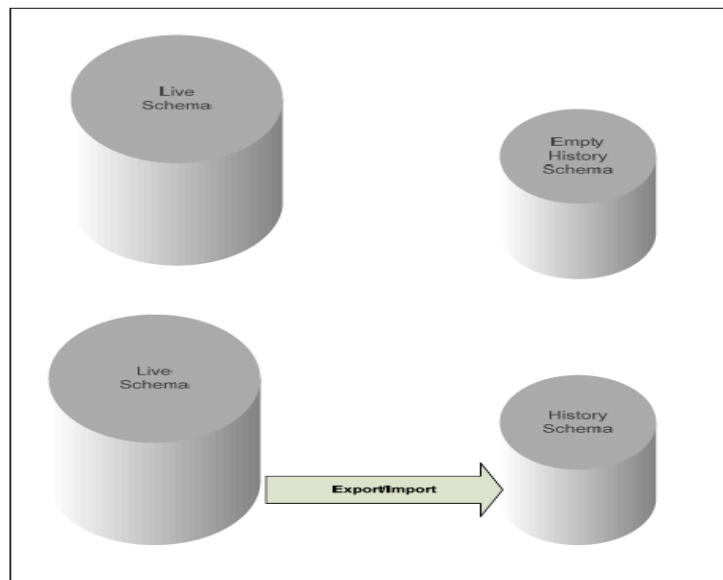
Data Archive Advantages

Feature	Benefit
Data cannot be deleted unless the data exists in the history schema.	This ensures no data can be removed by accident.
Live and history data can be viewed together.	Reports can use the joined view as required when history data needs to be included.
Application transparency	Synonyms, aliases or proxy tables will be created such that applications can be selectively switched to view live and history. Thus allowing reports and inquiry screens to function with no code changes.
Data can be restored from history.	If too much data or incorrectly identified data has been moved to history it can be selectively restored.
The copy process is separate from the purge.	This allows users to verify the correct data has been archived prior to removal.
Data is removed using the joined schema.	This allows for high performance parallel purging of data.
Foreign keys relationships dictate which tables are removed first.	Child tables are removed prior to parent tables.
The commit frequency of the deletes can be controlled.	This allows performance to be balanced against database resources.
If table definitions change in production the same changes are identified and made in history.	This allows the history data to remain at the same release level as production. If data is exported to files and needs to be restored it requires mapping of the old structure onto the production structure every time.

<p>The history data is on the same technology stack.</p>	<p>Writing data into a different technology requires that interfaces be maintained from the old to the new to ensure that the history data can be restored at any time.</p>
<p>The Data Archive technology can be used as part of your existing archiving strategy.</p>	<p>Data Archive builds command scripts which can easily be incorporated as part of your existing batch processes.</p>

The Data Archive process

You will firstly have to populate a history schema with your data.



Step 1 - Build your history schema

Create a history schema to hold your archived data using your standard database utilities and tools. This can be within the same instance/database or in a remote instance/database. You may wish to alter the device or file locations of the underlying schema or table space such that it is on a lower cost device.

Step 2 - Populate the history schema

Data Archive provides a tool designed to enable the navigation of relationally intact sets of data. Data Archive works by using a database's in built foreign key constraints as well as permitting the user to define their own business rules between schema tables.

Once a set of related tables is designed it can be saved to file in order to be used as the basis for Data Archive's extract utility.

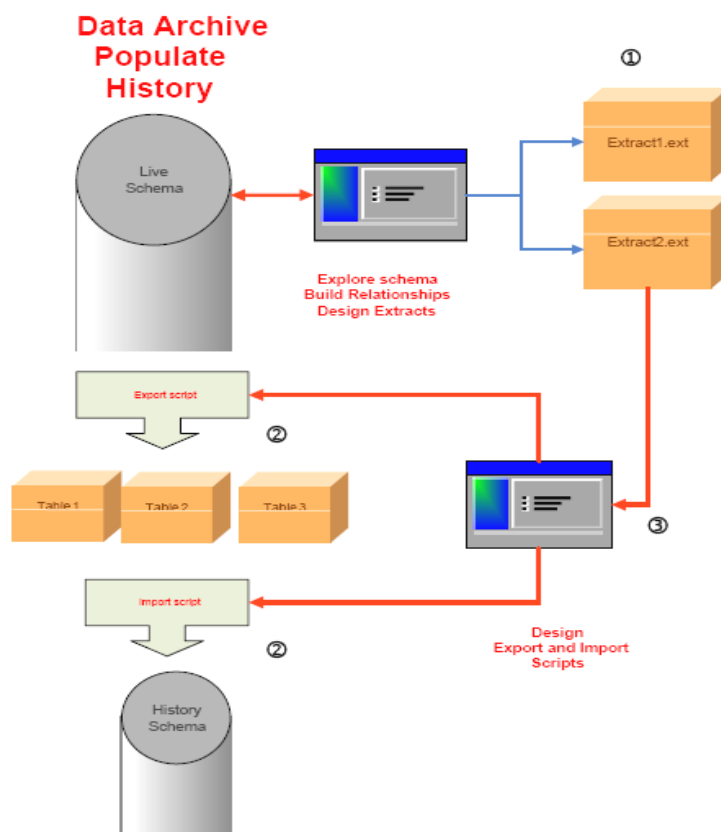
To populate your subset schema you must firstly design extract definitions, once you have completed the design you must save the definition using the save extract button:

You may build many extract definitions, the extract definitions are saved with a suffix of .ext. Once you have designed your extract you must build your export import scripts using the design export button.

You may also preview the record counts by running the generated _preview.sql which is created at the same time as the export/import scripts are generated.

The export/import process

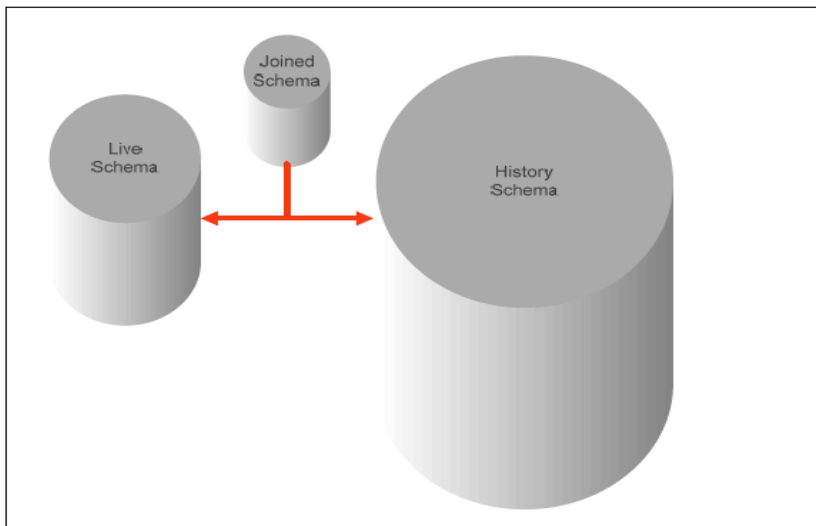
- Explore the schema and build data relationships
- Design and build multiple extract definitions
- Design and build export/extract and import load scripts
- Execute the export and import scripts in batch



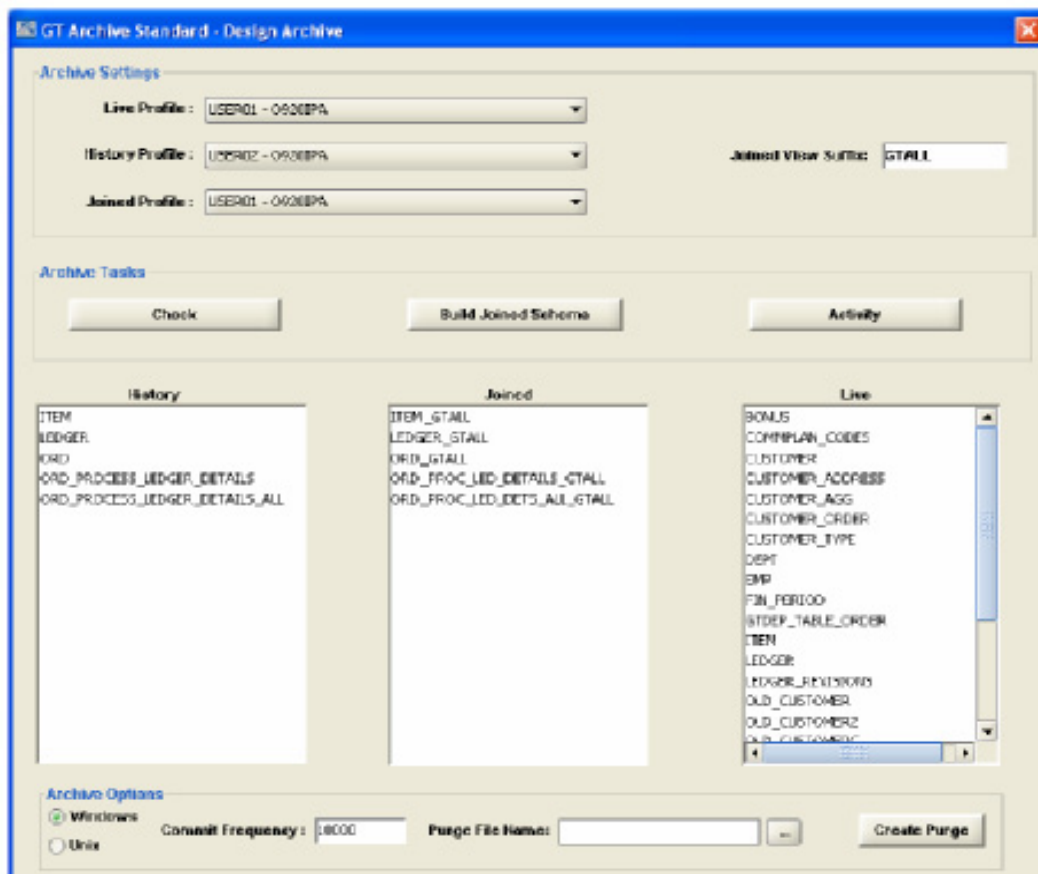
1. You may create many extract definitions
2. The export and import scripts will run in parallel on Unix or Linux.

Step 3 - Build the joined schema

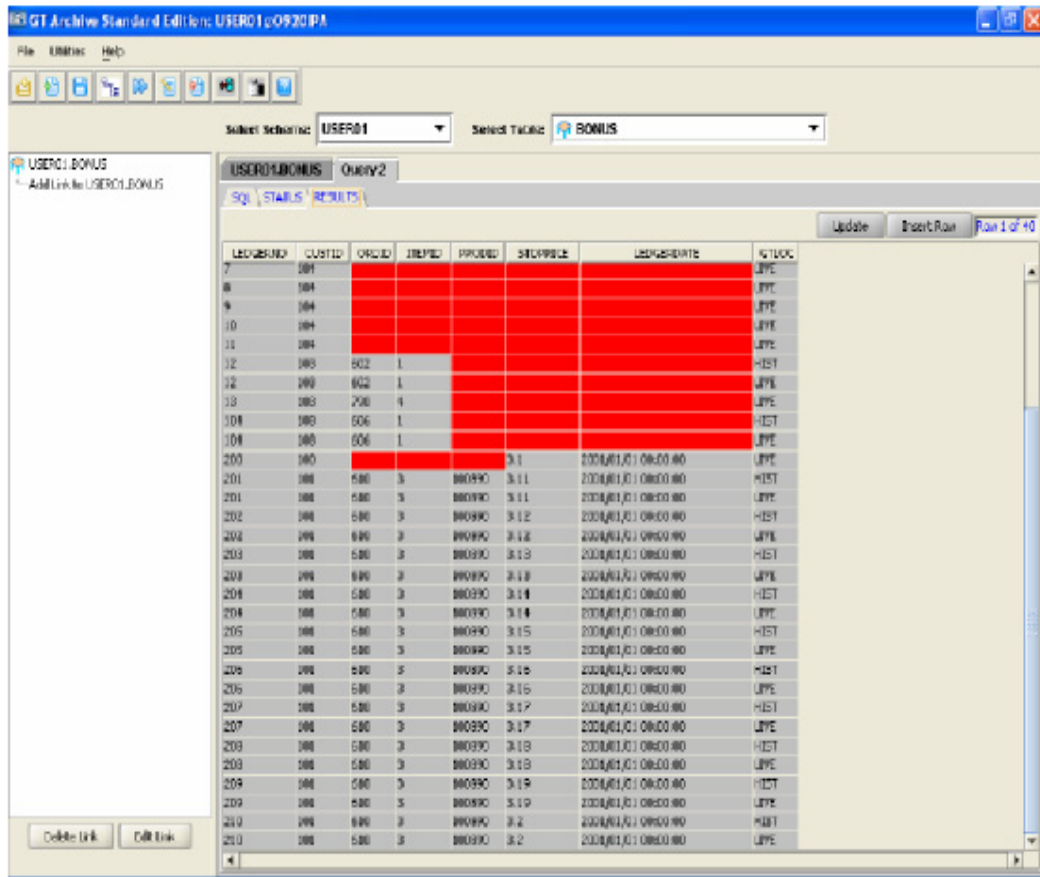
For the purge process to run, the live schema uses the joined schema to identify rows that have been archived and then purges them from live. This guarantees that the data has been archived correctly and also allows data to be selectively restored.



You will use the Data Archive – Build Joined Schema option to create the schema.

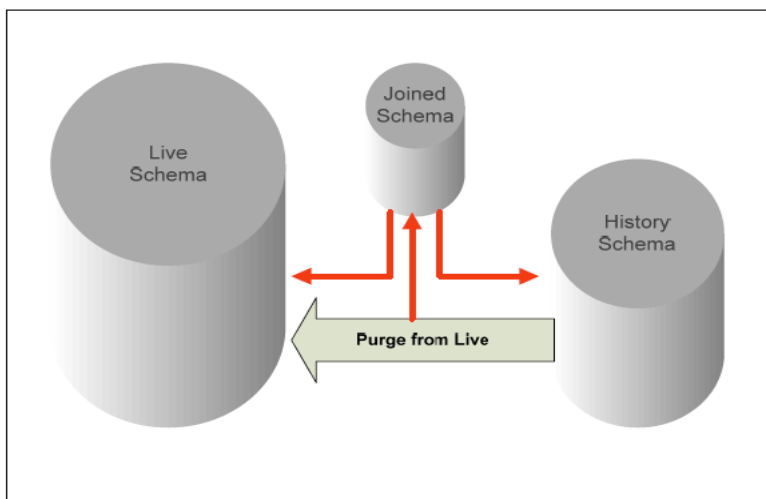


The Joined views of the data can be created in a separate schema or within the Live schema. Once the joined schema has been created you can view the joined data using the views.



An extra column is used to identify the location of the data. Verify that the correct data has been archived prior to issuing the delete.

Step 4 - Purge the data from Live

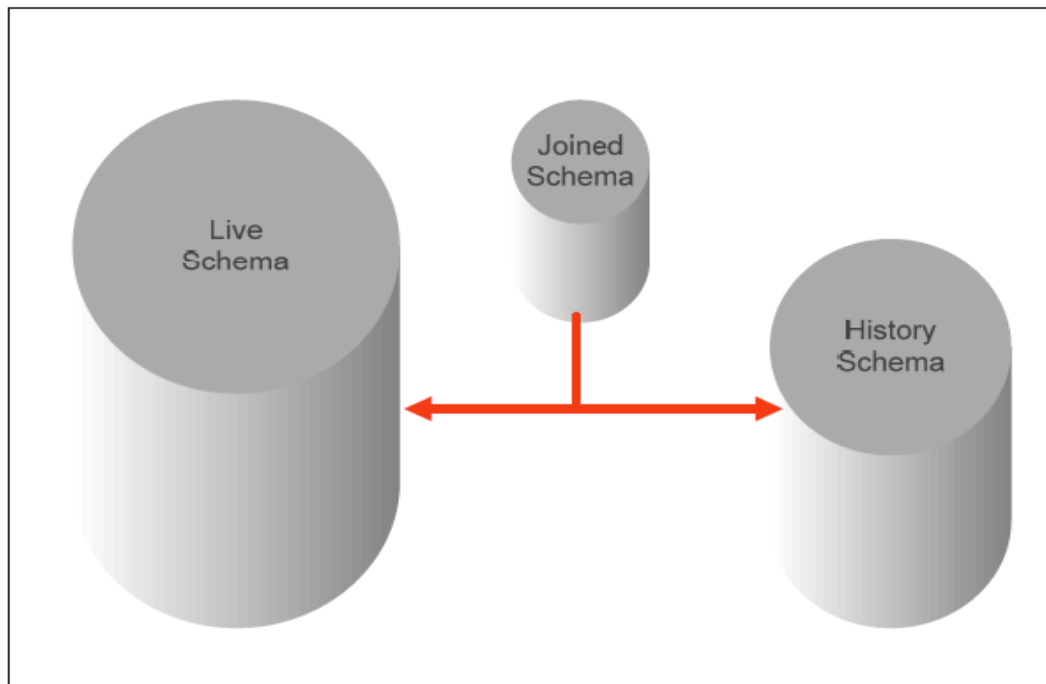


The purge utility GTPurge utility is called via a command script generated by Data Archive.

The GTPurge utility will delete the data in the correct referential sequence and in parallel. The commit frequency can be changed as required to control database resources and run times.

Step 5 - Accessing the live and history data

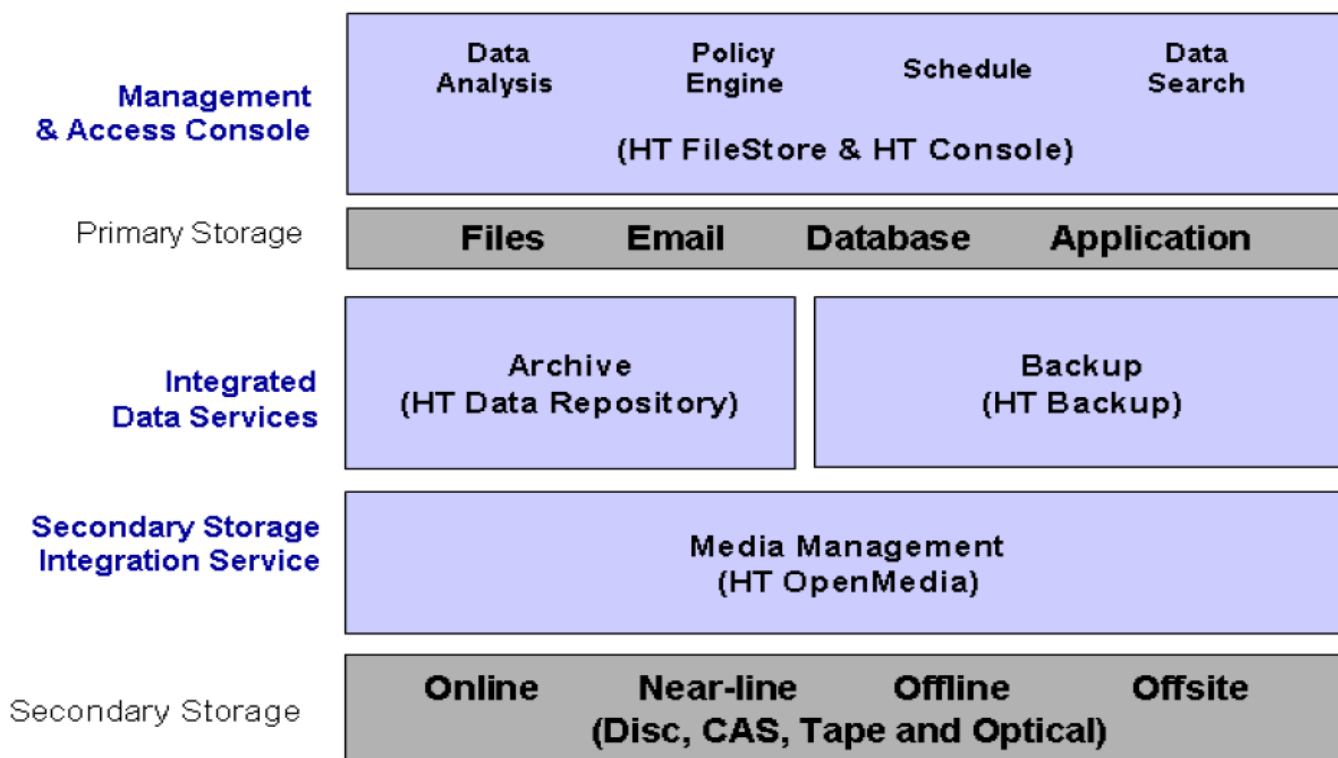
The joined schema views can be used via synonyms, aliases and proxy tables to allow existing applications to access the data with no code changes. Contact support for details of how to set this up.



- **BridgeHead’s HT Filestore**

HT FileStore provides policy driven storage management allowing an organization to configure its own storage management policies. These policies define what files should be retained, for how long and the most appropriate media type for their storage.

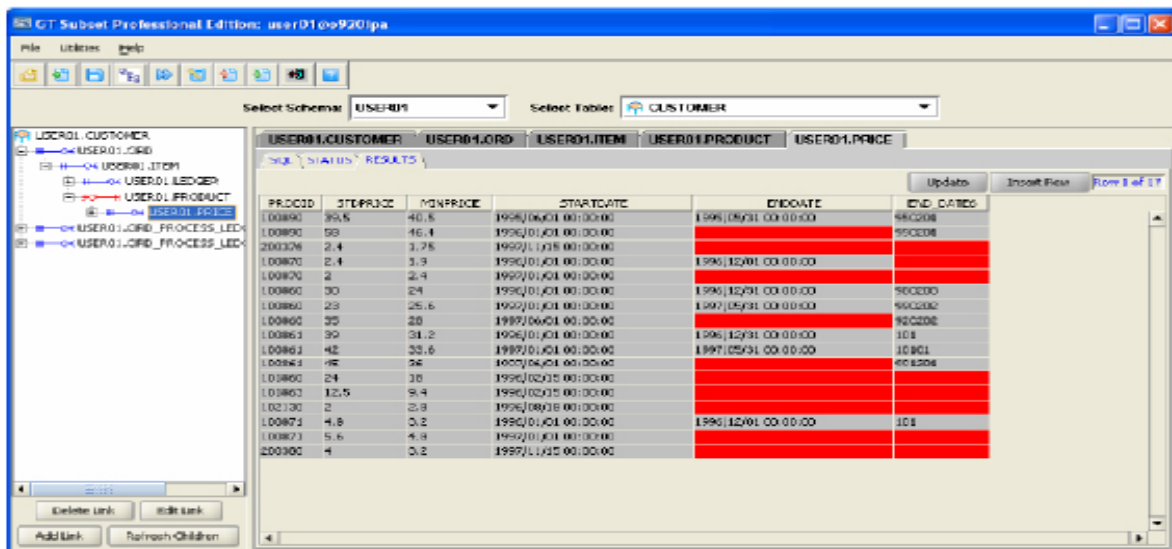
Files can be “actively archived” to the appropriate media or can remain transparent to the user using “stubbing” technology. HT FileStore manages the guaranteed storage of files using powerful policies, for example write the files to optical media offsite and retain a copy on local hard disk.



GT-Datastore interfaces with HT FileStore to ensure that extracted archive files have been “secured” before removing the data from production.

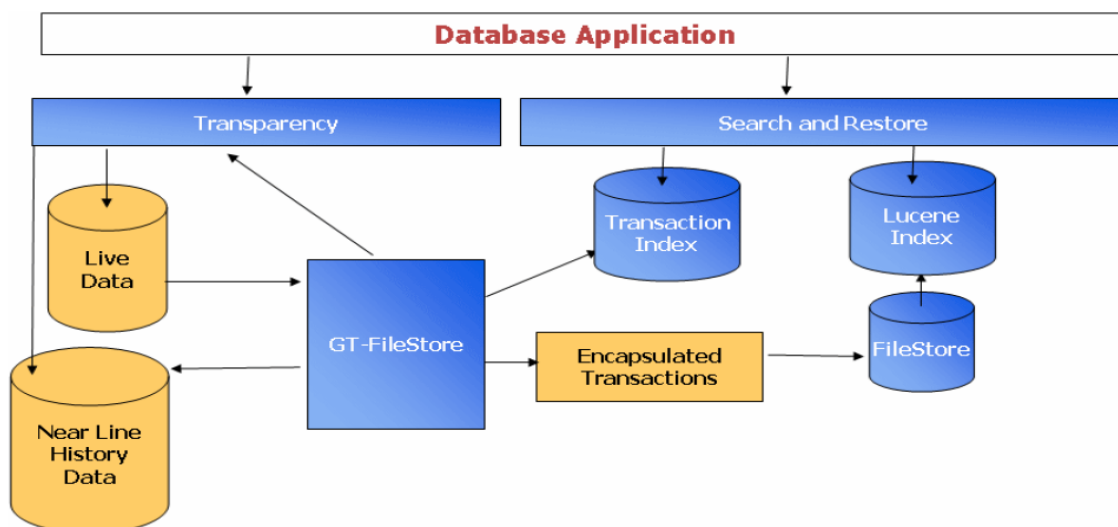
Standard Format

The database data is stored in a standard format that allows multiple RDBMS types to archive data to and from FileStore. Data can be archived from one technology and restored into another. The ability to archive data from legacy systems and then restore into newer open technology provides the IT manager with a powerful new toolset.



Transactions can be encapsulated – all appropriate references to a transaction are included at archive time.

GT Datastore allows you to manage your transaction data as it ages. The addition of new technology such as the Lucene searchable index will provide full search capability to the archived data.



Standard GT Datastore architecture.

© Grid-Tools Ltd 2010

www.grid-tools.com

info@grid-tools.com

UK: +44 (0) 1865 884 600

USA: 1 866 563 3120