EPAM Systems, RD Dep., RD Dep.

INTRODUCTION TO DWH AND ETL

ETL Overview

Legal Notice: This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of EPAM®.

CONTENTS

1 OVERVIEW OF ETL IN DATA WAREHOUSES

The challenge in data warehouse environments is to integrate, rearrange and consolidate large volumes of data over many systems, thereby providing a new unified information base for business intelligence.

The process of extracting data from source systems and bringing it into the data warehouse is commonly called ETL, which stands for extraction, transformation, and loading. Note that ETL refers to a broad process, and not three well-defined steps. The acronym ETL is perhaps too simplistic, because it omits the transportation phase and implies that each of the other phases of the process is distinct. Nevertheless, the entire process is known as ETL.

Experienced data warehouse architects realize that the process of understanding the data sources, designing transformations, testing the loading process, and debugging is often the most time-consuming part of deployment. Transformations are used to remove bogus data (including erroneous entries and duplicate entries), convert data items to an agreed-upon format, and filter data not considered necessary for the warehouse. These operations are often used to improve the quality of data loaded into the warehouse.

The frequency of data extraction from sources and loading into the data warehouse is largely determined by the required timeliness of the data in order to make business decisions. Most data extraction and loading takes place on a “batch” basis and data transformations cause a time delay. Early warehouses were often completely refreshed during the loading process, but as data volumes grew, this became impractical. Today, updates to tables are most common. When a need for near real-time data exists, warehouses can be loaded nearly continuously using a trickle feed if the source data is relatively clean, eliminating the need for complex transformations. Is Cleanliness Best?

Once the data in the warehouse is “clean,” is this version of the true nature of the data propagated back to the originating OLTP systems? This is an important issue for data warehouse implementation. In some cases, a “closed loop” process is implemented whereby updates are provided back to the originating systems. In addition to minimizing some of the cleansing that takes place during future extractions, operational reports become more accurate.

Another viable option is to avoid cleansing by improving the quality of the data at the time of its input into the operational system. This is critical if OLTP systems are to be directly accessed for business intelligence. Improving data quality at the source also enables high-speed loading techniques to be used in near real-time data warehouses (since transformations can be eliminated).

Improving data quality at the source can sometimes be accomplished by not allowing a “default” condition as allowable input into a data field. Presenting the data-entry person with an array of valid options, one of which must be selected, is often a way to ensure the most consistent and valid responses. Many companies also provide education to the data-entry people, showing them how the data they’re keying in will be used and what the significance of it is.

1.1 34 ETL SUBSYSTEMS

- **Data Profiling (subsystem 1)** – Explores a data source to determine its fit for inclusion as a source and the associated cleaning and conforming requirements.
- **Change Data Capture (subsystem 2)** – Isolates the changes that occurred in the source system to reduce the ETL processing burden.
- **Extract System (subsystem 3)** – Extracts and moves source data into the data warehouse environment for further processing.
- **Data Cleansing System (subsystem 4)** – Implements data quality processes to catch quality violations.
- **Error Event Tracking (subsystem 5)** – Captures all error events that are vital inputs to data quality improvement.
- **Audit Dimension Creation (subsystem 6)** – Attaches metadata to each fact table as a dimension. This metadata is available to BI applications for visibility into data quality.

- **Deduplication (subsystem 7)** – Eliminates redundant members of core dimensions, such as customers or products. May require integration across multiple sources and application of survivorship rules to identify the most appropriate version of a duplicate row.
- **Data Conformance (subsystem 8)** – Enforces common dimension attributes across conformed master dimensions and common metrics across related fact tables (see related article, “Kimball University: Data Integration for Real People”).
- **Slowly Changing Dimension (SCD) Manager (subsystem 9)** – Implements logic for slowly changing dimension attributes.
- **Surrogate Key Generator (subsystem 10)** – Produces surrogate keys independently for every dimension.
- **Hierarchy Manager (subsystem 11)** – Delivers multiple, simultaneous, embedded hierarchical structures in a dimension.
- **Special Dimensions Manager (subsystem 12)** – Creates placeholders in the ETL architecture for repeatable processes supporting an organization’s specific dimensional design characteristics, including standard dimensional design constructs such as junk dimensions, mini-dimensions and behavior tags.
- **Fact Table Builders (subsystem 13)** – Construct the three primary types of fact tables: transaction grain, periodic snapshot and accumulating snapshot.
- **Surrogate Key Pipeline (subsystem 14)** – Replaces operational natural keys in the incoming fact table record with the appropriate dimension surrogate keys.
- **Multi-Valued Bridge Table Builder (subsystem 15)** – Builds and maintains bridge tables to support multi-valued relationships.
- **Late Arriving Data Handler (subsystem 16)** – Applies special modifications to the standard processing procedures to deal with late-arriving fact and dimension data.
- **Dimension Manager (subsystem 17)** – Centralized authority who prepares and publishes conformed dimensions to the data warehouse community.
- **Fact Table Provider (subsystem 18)** – Owns the administration of one or more fact tables and is responsible for their creation, maintenance and use.
- **Aggregate Builder (subsystem 19)** – Builds and maintains aggregates to be used seamlessly with aggregate navigation technologies for enhanced query performance.
- **OLAP Cube Builder (subsystem 20)** – Feeds data from the relational dimensional schema to populate OLAP cubes.
- **Data Propagation Manager (subsystem 21)** – Prepares conformed, integrated data from the data warehouse presentation server for delivery to other environments for special purposes.
- **Job Scheduler (subsystem 22)** – Reliably manages the ETL execution strategy, including the relationships and dependencies between ETL jobs.
- **Backup System (subsystem 23)** – Backs up the ETL environment for recovery, restart and archival purposes.
- **Recovery and Restart (subsystem 24)** – Processes for recovering the ETL environment or restarting a process in the event of failure.
- **Version Control (subsystem 25)** – Takes snapshots for archiving and recovering all the logic and metadata of the ETL pipeline.
- **Version Migration (subsystem 26)** – Migrates a complete version of the ETL pipeline from development into test and finally into production.
- **Workflow Monitor (subsystem 27)** – Ensures that the ETL processes are operating efficiently and that the warehouse is being loaded on a consistently timely basis.
- **Sorting (subsystem 28)** – Serves the fundamental, high-performance ETL processing role.

- **Lineage and Dependency (subsystem 29)** – Identifies the source of a data element and all intermediate locations and transformations for that data element or, conversely, start with a specific data element in a source table and reveals all activities performed on that data element.
- **Problem Escalation (subsystem 30)** – Support structure that elevates ETL problems to appropriate levels for resolution.
- **Paralleling and Pipelining (subsystem 31)** – Enables the ETL system to automatically leverage multiple processors or grid computing resources to deliver within time constraints.
- **Security (subsystem 32)** – Ensures authorized access to (and a historical record of access to) all ETL data and metadata by individual and role.
- **Compliance Manager (subsystem 33)** – Supports the organization's compliance requirements typically through maintaining the data's chain of custody and tracking who had authorized access to the data.
- **Metadata Repository (subsystem 34)** – Captures ETL metadata including the process metadata, technical metadata and business metadata, which make up much of the metadata of the total DW/BI environment.

1.2 ETL TOOLS FOR DATA WAREHOUSES

Designing and maintaining the ETL process is often considered one of the most difficult and resource-intensive portions of a data warehouse project. Many data warehousing projects use ETL tools to manage this process.

Besides the support of extraction, transformation, and loading, there are some other tasks that are important for a successful ETL implementation as part of the daily operations of the data warehouse and its support for further enhancements. Besides the support for designing a data warehouse and the data flow, ETL tools such as OWB typically address these tasks.

The successive loads and transformations must be scheduled and processed in a specific order. Depending on the success or failure of the operation or parts of it, the result must be tracked and subsequent, alternative processes might be started. The control of the progress as well as the definition of a business workflow of the operations are typically addressed by ETL tools.

As the data warehouse is a living IT system, sources and targets might change. Those changes must be maintained and tracked through the lifespan of the system without overwriting or deleting the old ETL process flow information. To build and keep a level of trust about the information in the warehouse, the process flow of each individual record in the warehouse can be reconstructed at any point in time in the future in an ideal case.

Key products and database features that often help facilitate process include:

- **Pentaho Data Integration.** Pentaho is a very powerful but simple tool that is used to extract, transform, and load the data to any database. Pentaho is a software company that has developed an ETL tool known as Pentaho data integration. Pentaho is also known as kettle.
  Pentaho has its headquarters in Florida, USA and it provides services like data mining, data integration, and data warehousing. Pentaho is used by many organizations to copy or move the data from SaaS applications and also from the databases into their data warehousing which helps users to view the data into the dashboards and also in reports.
  In Pentaho, developers can perform data replication which can help them schedule jobs and can get the data in minutes to upto 24 hours. In Pentaho, we can do data analysis of data without writing SQL or ETL scripts.

  Important Features of Pentaho:
  - **Total Self-Service tool**

  Pentaho is totally a Self-Service tool. We do not have to contact account managers and customer representatives as it is very simple and easy to understand.
  - **Documentation**

The documentation provided by Pentaho is very accurate. If you study the documentation, you can easily set up the ETL integration process.

- **User-Friendly GUI**

The graphical interface provided in Pentaho is very user-friendly as it has simple drag-drop features.

- **Metadata Approach**

Pentaho data integration follows the metadata approach which is basically the data inside the data approach.

- **Oracle GoldenGate.** Acquired by Oracle in 2009, GoldenGate has replaced Oracle Streams as the primary software recommended for log-based replication. Often used for zero downtime software upgrades, during software migrations, and for low latency transaction replication and recovery, GoldenGate supports a wide variety of data sources and targets. It is often used to load Oracle-based data warehouses where the need for data transformations is minimal and near real-time updates of the data in the data warehouse are desired. Read on to find out if this is the tool for your PostgreSQL ETL needs.
  Oracle GoldenGate tool is built to manage data integration and this tool is suitable for very large companies which have frequent data migration requirements. This tool is designed for CDC (change data capture), real time delivery and routing purposes. This is a very comprehensive tool and is used for low impact, high speed, real time data replication and integration in different IT environments.
  Oracle GoldenGate allows us to filter and transform data from one database to another. Oracle GoldenGate tool can be replicated with flat files, Java messaging queue and with big data. Oracle GoldenGate is used to move the data very quickly with zero downtime.

  Important Features of Oracle GoldenGate

  - **Maintains Logs**

  Oracle GoldenGate tool maintains logs for any CDC (change data capture), replication, transformation, distribution, and delivery.

  - **Real-Time Data Movement**

  In Oracle GoldenGate tool, data moves in real-time and it avoids latency or delay while moving the data.

  - **Easy Problem Analysis**

  Oracle GoldenGate tool helps the developers to analyse the problems or issues very easily because of its very simple design structure.

  - **Supports Popular Databases**

  Oracle GoldenGate supports very popular databases like Teradata, Sybase, IBM db2, Exadata, etc.

  - **Data Delivery and Fast Recovery**

  In Oracle GoldenGate, data can be delivered reliably and also data recovery is very fast after some interruptions or network traffic.

- **Talend Open Studio.** Talend was launched in the year 2005 and it is a US-based software company headquartered in California, USA. It employs about 600 people. In the year 2006, Talend introduced its first data integration tool known as “Talend Open Studio” which was the first product introduced by it.
  Talend Open Studio supports data migration, data warehousing and profiling. It is one of the most powerful and innovative tools introduced in the market and it is open source. Talend Open Studio meets all the data integration needs of both small and big sized companies. It has a very interactive GUI which allows dragging and dropping components, connecting them together to create and then run the ETL pipelines.
  Talend Open Studio performs ETL and can be deployed on-premise or on any of the SaaS applications. In Talend Open Studio, there is no need to write any code as Java code is automatically generated. Talend Open Studio can be connected to various data warehouses like SaaS applications, Google sheets, RDBMS, IBM db2, Oracle etc.
  Talend Open Studio is an open-source software and it does not have any licensing cost as it is freely available.

Important Features of Talend Open Studio

- **No Licensing Cost**

Talend Open Studio is an open source software and it does not have any licensing cost.

- **Interactive GUI**

In Talend Open Studio, GUI is very interactive and we can drag and drop the components to create and run the ETL pipelines.

- **In-Built Components**

Talend Open Studio has nearly about 900+ built-in components that are used to connect to the data sources.

- **Deployment Methods**

In Talend Open Studio, code can be deployed on-premise or it can be deployed on SaaS applications.

- **Improves Productivity**

Talend Open Studio has inbuilt components therefore it can improve productivity and can also save time.

- **Informatica PowerCenter** (cloud based PowerCenter is Informatica Cloud Data Integration). The next on our list of GUI tools for PostgreSQL ETL is Informatica PowerCenter. Informatica is a software company which is headquartered in California, USA. It was founded in the year 1993 and it has a total revenue of about 1.03 billion and a total employee count of about 4000. PowerCenter is a product of Informatica and it is developed for data integration. Let us talk about why it could be a suitable tool for PostgreSQL ETL.
  Informatica PowerCenter is an ETL tool that is used to build enterprise data warehouses.
  Informatica has a variety of tools that help the developer to do a variety of tasks like manage repositories, define mapping and the properties of the fields, report metadata, etc.
  Informatica PowerCenter repository, which is the central repository, stores all the source and destination data along with the mapping of the fields. The Informatica PowerCenter server connects the sources and the destinations, allowing data transfer and transformations.
  Informatica PowerCenter has different areas like data migration, data governance, data warehousing, data replication, SOA (service oriented architectures) etc.

  Important Features of Informatica PowerCenter

  - **Supports Agile Process**

  Informatica PowerCenter supports Agile processes for the SDLC (software development life cycle), therefore activities like requirement analysis, designing, coding and testing are done in parallel, in the form of sprints.

  - **Automatic Data Validation**

  In Informatica PowerCenter, all the result validations, development and testing can be automated in the production environments which saves lots of time and effort.

  - **Cost Reduction**

  Informatica PowerCenter tool has very easy training modules in such a way that even a non technical person can run jobs and can also monitor them which in turn helps in reduction of costs.

  - **Expose Primary Functionality**

  Informatica PowerCenter has a service gateway which exposes all the primary and important functionality of the product to the clients and makes a very presentable UI.

  - **Tight Integration and Scalability**

  Informatica PowerCenter is tightly coupled and integrated with messaging systems and supports concurrent data processing.

  - **Central Repository system**

  Informatica PowerCenter has a central repository service which provides all the information to extract, transform and load the data from MS SQL server targets.

The core PostgreSQL Database engine features embedded ETL functions that ETL tools support to varying degrees. Examples of these features include support for table functions, merge (i.e., insert or update depending on whether a data item exists), multitable inserts, change data capture, and resumable statements.

2 OVERVIEW OF EXTRACTION IN DATA WAREHOUSES

Extraction is the operation of extracting data from a source system for further use in a data warehouse environment. This is the first step of the ETL process. After the extraction, this data can be transformed and loaded into the data warehouse.

The source systems for a data warehouse are typically transaction-processing applications. For example, one of the source systems for a sales analysis data warehouse might be an order entry system that records all of the current order activities.

Designing and creating the extraction process is often one of the most time-consuming tasks in the ETL process and, indeed, in the entire data warehousing process. The source systems might be very complex and poorly documented, and thus determining which data needs to be extracted can be difficult. The data has to be extracted normally not only once, but several times in a periodic manner to supply all changed data to the data warehouse and keep it up-to-date. Moreover, the source system typically cannot be modified, nor can its performance or availability be adjusted, to accommodate the needs of the data warehouse extraction process.

These are important considerations for extraction and ETL in general. This chapter, however, focuses on the technical considerations of having different kinds of sources and extraction methods. It assumes that the data warehouse team has already identified the data that will be extracted, and discusses common techniques used for extracting data from source databases.

Designing this process means making decisions about the following two main aspects:

- *Which extraction method do I choose?* This influences the source system, the transportation process, and the time needed for refreshing the warehouse.
- *How do I provide the extracted data for further processing?* This influences the transportation method, and the need for cleaning and transforming the data.

3 EXTRACTION METHODS IN DATA WAREHOUSES

The extraction method you should choose is highly dependent on the source system and also from the business needs in the target data warehouse environment. Very often, there is no possibility to add additional logic to the source systems to enhance an incremental extraction of data due to the performance or the increased workload of these systems. Sometimes even the customer is not allowed to add anything to an out-of-the-box application system.

3.1 LOGICAL EXTRACTION METHODS

There are two types of logical extraction:

- Full Extraction
- Incremental Extraction

3.1.1 Full Extraction

The data is extracted completely from the source system. Because this extraction reflects all the data currently available on the source system, there is no need to keep track of changes to the data source since the last successful extraction. The source data will be provided as-is and no additional logical information (for example, timestamps) is necessary on the source site. An example for a full extraction may be an export file of a distinct table or a remote SQL statement scanning the complete source table.

3.1.2 Incremental Extraction

At a specific point in time, only the data that has changed since a well-defined event back in history is extracted. This event may be the last time of extraction or a more complex business event like the last booking day of a fiscal period. To identify this delta change there must be a possibility to identify all the changed information since this specific time event. This information can be either provided by the source data itself such as an application column, reflecting the last-changed timestamp or a change table where an appropriate additional mechanism keeps track of the changes besides the originating transactions. In most cases, using the latter method means adding extraction logic to the source system.

Many data warehouses do not use any change-capture techniques as part of the extraction process. Instead, entire tables from the source systems are extracted to the data warehouse or staging area, and these tables are compared with a previous extract from the source system to identify the changed data. This approach may not have significant impact on the source systems, but it clearly can place a considerable burden on the data warehouse processes, particularly if the data volumes are large.

Change Data Capture (CDC) mechanism can extract and maintain such delta information. See Chapter 17, "Change Data Capture" for further details about the Change Data Capture framework.

3.2 PHYSICAL EXTRACTION METHODS

Depending on the chosen logical extraction method and the capabilities and restrictions on the source side, the extracted data can be physically extracted by two mechanisms. The data can be extracted either online from the source system or from an offline structure. Such an offline structure might already exist or it might be generated by an extraction routine.

There are the following methods of physical extraction:

- Online Extraction
- Offline Extraction

3.2.1 Online Extraction

The data is extracted directly from the source system itself. The extraction process can connect directly to the source system to access the source tables themselves or to an intermediate system that stores the data in a preconfigured manner (for example, snapshot logs or change tables). Note that the intermediate system is not necessarily physically different from the source system.

With online extractions, you must consider whether the distributed transactions are using original source objects or prepared source objects.

3.2.2 Offline Extraction

The data is not extracted directly from the source system but is staged explicitly outside the original source system. The data already has an existing structure (for example, redo logs, archive logs or transportable tablespaces) or was created by an extraction routine.

You should consider the following structures:

- *Flat files*. Data in a defined, generic format. Additional information about the source object is necessary for further processing.
- *Dump files*. PostgreSQL-specific format. Information about the containing objects may or may not be included, depending on the chosen utility.
- *Redo and archive logs*. Information is in a special, additional dump file.

3.3 CHANGE DATA CAPTURE

An important consideration for extraction is incremental extraction, also called Change Data Capture. If a data warehouse extracts data from an operational system on a nightly basis, then the data warehouse requires only the data that has changed since the last extraction (that is, the data that has been modified in the past 24 hours). Change Data Capture is also the key-enabling technology for providing near real-time, or on time, data warehousing.

When it is possible to efficiently identify and extract only the most recently changed data, the extraction process (and all downstream operations in the ETL process) can be much more efficient, because it must extract a much smaller volume of data. Unfortunately, for many source systems, identifying the recently modified data may be difficult or intrusive to the operation of the system. Change Data Capture is typically the most challenging technical issue in data extraction.

Because change data capture is often desirable as part of the extraction process and it might not be possible to use the Change Data Capture mechanism, this section describes several techniques for implementing a self-developed change capture on Database source systems: Timestamps, Partitioning, Triggers.

These techniques are based upon the characteristics of the source systems, or may require modifications to the source systems. Thus, the owners of the source system must carefully evaluate each of these techniques prior to implementation.

Each of these techniques can work in conjunction with the data extraction technique discussed previously. For example, timestamps can be used whether the data is being unloaded to a file or accessed through a distributed query.

3.3.1 Timestamps

The tables in some operational systems have timestamp columns. The timestamp specifies the time and date that a given row was last modified. If the tables in an operational system have columns containing timestamps, then the latest data can easily be identified using the timestamp columns. For example, the following query might be useful for extracting today's data from an orders table:

```
SELECT * FROM orders
WHERE date_trunc('day',order_date) = current_timestamp::date;
```

If the timestamp information is not available in an operational source system, you are not always able to modify the system to include timestamps. Such modification would require, first, modifying the operational system's tables to include a new timestamp column and then creating a trigger to update the timestamp column following every operation that modifies a given row.

3.3.2 Partitioning

Some source systems might use range partitioning, such that the source tables are partitioned along a date key, which allows for easy identification of new data. For example, if you are extracting from an orders table, and the orders table is partitioned by week, then it is easy to identify the current week's data.

3.3.3 Triggers

Triggers can be created in operational systems to keep track of recently updated records. They can then be used in conjunction with timestamp columns to identify the exact time and date when a given row was last modified. You do this by creating a trigger on each source table that requires change data capture. Following each DML statement that is executed on the source table, this trigger updates the timestamp column with the current time. Thus, the timestamp column provides the exact time and date when a given row was last modified.

A similar internalized trigger-based technique is used for materialized view logs. These logs are used by materialized views to identify changed data, and these logs are accessible to end users. However, the format of the materialized view logs is not documented and might change over time.

If you want to use a trigger-based mechanism, use synchronous change data capture. It is recommended that you use synchronous Change Data Capture for trigger based change capture, because CDC provides an externalized interface for accessing the change information and provides a framework for maintaining the distribution of this information to various clients.

Materialized view logs rely on triggers, but they provide an advantage in that the creation and maintenance of this change-data system is largely managed by the database.

However, some databases recommend the usage of synchronous Change Data Capture for trigger-based change capture, since CDC provides an externalized interface for accessing the change information and provides a framework for maintaining the distribution of this information to various clients

Trigger-based techniques might affect performance on the source systems, and this impact should be carefully considered prior to implementation on a production source system.

4 OVERVIEW OF TRANSPORTATION IN DATA WAREHOUSES

Transportation is the operation of moving data from one system to another system. In a data warehouse environment, the most common requirements for transportation are in moving data from:

- A source system to a staging database or a data warehouse database
- A staging database to a data warehouse
- A data warehouse to a data mart

Transportation is often one of the simpler portions of the ETL process and can be integrated with other portions of the process. For example, distributed query technology provides a mechanism for both extracting and transporting data.

4.1 INTRODUCTION TO TRANSPORTATION MECHANISMS IN DATA WAREHOUSES

You have three basic choices for transporting data in warehouses:

1. Transportation Using Flat Files
2. Transportation Through Distributed Operations
3. Transportation Using Transportable Tablespaces

4.1.1 Transportation Using Flat Files

The most common method for transporting data is by the transfer of flat files, using mechanisms such as FTP or other remote file system access protocols. Data is unloaded or exported from the source system into flat files using techniques discussed in Chapter 13, "Extraction in Data Warehouses", and is then transported to the target platform using FTP or similar mechanisms.

Because source systems and data warehouses often use different operating systems and database systems, using flat files is often the simplest way to exchange data between heterogeneous systems with minimal transformations. However, even when transporting data between homogeneous systems, flat files are often the most efficient and most easy-to-manage mechanism for data transfer.

4.1.2 Transportation through Distributed Operations

Distributed queries, either with or without gateways, can be an effective mechanism for extracting data. These mechanisms also transport the data directly to the target systems, thus providing both extraction and transformation in a single step. Depending on the tolerable impact on time and system resources, these mechanisms can be well suited for both extraction and transformation.

As opposed to flat file transportation, the success or failure of the transportation is recognized immediately with the result of the distributed query or transaction. See Chapter 13, "Extraction in Data Warehouses" for further information.

5 OVERVIEW OF LOADING IN DATA WAREHOUSES

The load process in ETL refers to the phase where data that has been transformed and possibly cleansed in the previous stages is loaded into the target destination.

The entire process of transferring data to a data warehouse repository is referred to in the following ways:

- Full Load: With a full load, the entire dataset is dumped, or loaded, and is then completely replaced (i.e. deleted and replaced) with the new, updated dataset. No additional information, such as timestamps, is required.

  For example, take a store that uploads all of its sales through the ETL process in data warehouse at the end of each day. Let's say 5 sales were made on a Monday, so that on Monday night a table of 5 records would be uploaded. Then, on Tuesday, another 3 sales were made which need to be added. So on Tuesday night, assuming a full load, Monday's 5 records as well as Tuesday's 3 records are uploaded – an inefficient system, although relatively easy to set up and maintain. While this example is overly simplified, the principle is the same.

- Incremental Load:

  Only the *difference* between the target and source data is loaded through the ETL process in data warehouse. There are 2 types of incremental loads, depending on the volume of data you're loading; streaming incremental load and batch incremental load.

  - Streaming incremental load – better for loading small data volumes
  - Batch incremental load – better for loading large data volumes

  Following the previous example, the store that made 3 sales on Tuesday will load only the *additional* 3 records to the sales table, instead of reloading *all* records. This has the advantage of saving time and resources but increases complexity. Incremental loading is of course **much faster** than a full load. The main drawback to this type of loading is maintainability. Unlike a full load, with an incremental load you can't re-run the entire load if there's an error. In addition to this, files need to be loaded in order, so errors will compound the issue as other data queues up.

- Initial Load: For the very first time loading all the data warehouse tables.
- Full Refresh: Deleting the contents of a table and reloading it with fresh data.

6 POSTGRESQL DATA LOAD

6.1 COPY COMAND

PostgreSQL provides a way to load data into tables quickly, many rows at a time, instead of executing INSERT commands row by row. The COPY command loads data into a table. The data can be taken from a file that is located on the database server in a place accessible by the PostgreSQL server process or from a stream.

The data by default should look like a tab-separated list of fields; with records separated by a new line. However, this is customizable. The command is not included in the SQL standard. Here is a simplified syntax diagram for this command:

```
COPY >table name> [(column [, ...])]
FROM { >file name> | STDIN | PROGRAM >command> } [[WITH] (>options>)]
```

Here, options is used to specify data format, delimiters, quotations, escape characters, and some other parameters. STDIN can be specified instead of the filename to copy the data from standard input. PROGRAM is used to take the data from the output of a shell command. The column list can be provided when the file does not have the data for all the columns or they are given in a different order.

The full description of all the options and parameters that the COPY command can take you can find in the documentation, here: https://www.postgresql.org/docs/current/sql-copy.html

To load the data into the database, you should copy the file to the server and then execute the following command:

```
COPY dwh.access_log FROM '/tmp/access.log';
```

This would load the contents of the file located at /tmp/access.log into the dwh.access_log table. However, this is not a good idea for several reasons:

- Only a superuser can execute the COPY command to read or write from files.
- As the PostgreSQL server process needs to access the file, it would be necessary to either place the file in the location where other database files are stored, or allow the PostgreSQL process to access other locations. Both may be considered issues from a security point of view.
- Copying the file to the server may be the issue by itself; it would be necessary to allow the user (or the application) to access the file system on the server or mount shared network locations to the server.

To avoid these issues, you should use the COPY command to load the data from a stream, namely standard input. To do so, it may be necessary to write an application that would connect to the database, execute the COPY, command, and then stream the data to the server. Luckily, the psql console can do this.

Let's try to load the access.log sample file that is provided in the attached media.

Suppose the file is located in the current directory. The database is running on localhost and the car_portal_app database user is allowed to connect. The following command will load the data from the file to the dwh.access_log table:

```
user@host:~$ psql -h localhost -U car_portal_app -d car_portal \
-c "COPY dwh.access_log FROM STDIN WITH (format csv, delimiter ';')" \
> access.log
COPY 15456
```

The access.log file is redirected to the input of the psql client. It executes the COPY command that loads the data from standard input. The output of the command is the word COPY followed by the number of rows copied.

The command will be the same on both Linux and Windows.

It's also possible to execute this operation interactively from the psql console. There's a \copy command provided by psql, which has a similar syntax to the SQL COPY command: \copy dwh.access_log FROM 'access.log' WITH csv delimiter ';'. In the background, it's still the same: on the server, the COPY ... FROM STDIN command is executed, and then psql reads the file and sends its contents to the server.

Using this approach, it's very easy to organize a simple ETL process. It may be executed once per day and would just execute the preceding command. The prerequisite for this would be setting up log rotation for the HTTP server to make sure that the same data is not loaded twice.

The COPY command can be used not only to load data into tables, but can also copy the data the other way around: from tables to files or to standard output. This feature can be used again in ETL processes that can be organized in the form of simple bash scripts, instead of complex software that would use highlevel access to the database using JDBC or other APIs.

Here is an example of how to copy a data table from one server to another (supposing the structure of the table is identical):

```
$ psql -h server1 -d database1 -c "COPY table1 TO STDOUT" | psql -h server2 -d database2 -c "COPY table2 FROM STDIN"
```

Here, the first psql command will read the table and output it to standard output. This stream is piped to the second psql command that writes the data to the other table, taking it from standard input.

ETL can also include enriching the input raw data with additional attributes or preprocessing it to make it easier to query. For example, the entries could be mapped to the records in the car_portal database. Let's say API calls to /api/cars/30 could be mapped to the record in the car table with car_id = 30. Such processing can be done in the database, as follows:

```
car_portal=> ALTER TABLE dwh.access_log ADD car_id int;
ALTER TABLE
car_portal=> UPDATE dwh.access_log
SET car_id = (SELECT regexp_matches(url, '/api/cars/(d+)W'))[1]::int
WHERE url like '%/api/cars/%';
UPDATE 2297
```

In a similar way, the access_log table can be enriched by another attribute and can be joined to other tables.

In real life, ETL is usually more complex. It would normally include checking whether the data being loaded exists already in the destination table and would clean it before loading. This will make the process idempotent. Additionally, the process could identify which data to load and locate the input data. For example, consider an ETL job that loads a dataset every day. If it has failed for some reason, when started the next day, it should detect the previous failure and load two sets of data instead of only one. Notifying the right systems or people in case of failure can also be done by the ETL process.

There are a lot of different ETL tools available on the market, and as open source software.

6.2 DATABASE LINKS

In the real world, applications are not isolated, and they often have to send data between each other. Such interaction canbe implemented at the application level, for example, with the help of web services or file exchange, or you can use the database functionality for this purpose.

PostgreSQL supports the ISO/IEC 9075-9 standard (SQL/MED, Management of External Data), which defines work with external data sources from SQL via a special mechanism of foreign data wrappers.

The idea is to access external (foreign) data as if it were located in regular PostgreSQL tables. It requires creating foreign tables, which do not contain any data themselves and only redirect all queries to an external data source. This approach facilitates application development since it allows to abstract from specifics of a particular external source.

Creating a foreign table involves several sequential steps.

1. The CREATE FOREIGN DATA WRAPPER command plugs in a library for working with a particular data source.

2. The CREATE SERVER command defines a foreign server. You should usually specify such connection parameters as host name, port number, and database name.

3. The CREATE USER MAPPING command provides username mapping since different PostgreSQL users can connect to one and the same foreign source on behalf of different external users.

4. The CREATE FOREIGN TABLE command creates foreign tables for the specified external tables and views, while IMPORT FOREIGN SCHEMA allows to import descriptions of some or all tables from the external schema.

6.2.1 dblink – considering phasing out

The desire to use database links has been around for many years. However, around the turn of the century, PostgreSQL foreign data wrappers weren't even on the horizon, and a traditional database link implementation was definitely not in sight either. Around this time, a PostgreSQL developer from California (Joe Conway) pioneered work on database connectivity by introducing the concept of dblink to PostgreSQL. While dblink has served people well over the years, it is no longer state of the art.

Therefore, it is recommended we move away from dblink to the more modern SQL/MED implementation (which is a specification that defines the way external data can be integrated with a relational database).

The postgres_fdw extension has been built on top of SQL/MED and offers more than just database connectivity as it allows you to connect to basically any data source.

6.3 FETCHING FILES WITH FILE_FDW

In some cases, it can make sense to read a file from disk and expose it to PostgreSQL as a table. This is exactly what you can achieve with the file_fdw extension. The idea is to have a module that allows you to read data from a disk and query it using SQL.

Installing the module works as expected:

CREATE EXTENSION file_fdw;

Now, we need to create a virtual server:

CREATE SERVER file_server FOREIGN DATA WRAPPER file_fdw;

file_server is based on the file_fdw extension foreign data wrapper, which tells PostgreSQL how to access the file.

To expose a file as a table, use the following command:

```
CREATE FOREIGN TABLE t_passwd
(
username text,
passwd text,
uid int,
gid int,
gecos text,
dir text,
shell text
) SERVER file_server
OPTIONS (format 'text', filename '/etc/passwd', header 'false', delimiter ':');
```

In this example, the /etc/passwd file will be exposed. All the fields have to be listed and the data types have to be mapped accordingly. All of the additional important information is passed to the module using OPTIONS. In this example, PostgreSQL has to know the type of the file (text), the name and path of the file, as well as the delimiter.

It is also possible to tell PostgreSQL whether there is a header. If the setting is true, the first line will be skipped and deemed unimportant. Skipping headers is especially important if you happen to load a CSV file.

Once the table has been created, it is possible to read data:

SELECT * FROM t_passwd;

Unsurprisingly, PostgreSQL returns the content of /etc/passwd:

```
test=# \x
Expanded display is on. test=# SELECT * FROM t_passwd LIMIT 1;
-[ RECORD 1 ]-------
username | root
passwd | x
uid | 0
gid | 0
```

```
gecos | root
dir | /root
shell | /bin/bash
```

When looking at the execution plan, you will see that PostgreSQL uses what is known as a foreign scan to fetch the data from the file:

```
test=# explain (verbose true, analyze true) SELECT * FROM t_passwd;
QUERY PLAN
------------------------------------------------------------------
Foreign Scan on public.t_passwd (cost=0.00..2.80 rows=18 width=168)
(actual time=0.022..0.072 rows=61 loops=1)
Output: username, passwd, uid, gid, gecos, dir, shell
Foreign File: /etc/passwd
Foreign File Size: 3484
Planning time: 0.058 ms
Execution time: 0.138 ms
(6 rows)
```

The execution plan also tells us about the file's size and so on. Since we're talking about the planner, there is a side note that is worth mentioning: PostgreSQL will even fetch statistics for the file. The planner checks the file size and assigns the same costs to the file, just like it would to a normal PostgreSQL table of the same size.

The PostgreSQL distribution includes two foreign data wrappers: postgres_fdw and file_fdw. The first one is designed for working with external PostgreSQL databases, while the second one works with files on a server. Besides, the community develops and supports various libraries that provide access to many popular databases.

6.4 CONNECTING TO REMOTE SERVERS USING POSTGRES_FDW

postgres_fdw is designed for working with external PostgreSQL databases. Besides, the community develops and supports various libraries that provide access to many popular databases. Check out https://pgxn.org/tag/fdw for the full list.

Foreign data wrappers for Oracle, MySQL, and SQL Server are available as extensions:

1. Oracle: https://github.com/laurenz/oracle_fdw
2. MySQL: https://github.com/EnterpriseDB/mysql_fdw
3. SQL Server: https://github.com/tds-fdw/tds_fdw

Follow the instructions on these web pages to build and install these extensions, and this process will run smoothly.

7 SOURCE BOOKS AND ARTICLES

1. Kimball R., & Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition. Indianapolis: John Wiley & Sons, 2013.
2. URL link: https://hevodata.com/learn/best-postgresql-etl-tools/
3. URL link:https://severalnines.com/database-blog/best-etl-tools-migrating-postgresql
4. Salahaldin Juba, Andrey Volkov. Learning PostgreSQL 11 Third Edition, 2019.
5. Pavel Luzanov, Egor Rogov, Igor Levshin. Postgres. The First Experience, Edition 6, 2020.
6. Hans-Jürgen Schönig. Mastering PostgreSQL 13 - Fourth Edition, 2020.