

|  |
| --- |
| EPAM Systems, RD Dep., RD Dep.  **introduction to dwh and etl** |
| Knowledge Base |

}+10

Contents

[1 **Source Systems** 3](#_Toc139554533)

[2 **DWH Architecture and Naming** 4](#_Toc139554534)

[3 **Natural Keys VS Surrogate Keys** 4](#_Toc139554535)

[3.1 Surrogate Key Pros and Cons 5](#_Toc139554536)

[3.2 Natural Key Pros and Cons 5](#_Toc139554537)

[4 **The Difference Between Full and Incremental Loading** 6](#_Toc139554538)

[4.1 Full load 6](#_Toc139554539)

[4.2 Incremental load 6](#_Toc139554540)

# **Source Systems**

A Data Warehouse works as a central repository where information arrives from one or more data sources. Data flows into a data warehouse from the transactional system and other relational databases.

You must have at least 2 different source systems (2 datasets – 2 CSV files) for the final project. Your 3NF and Star schemas in the DWH will be filled based on the data from both source systems.

The simplest way to have 2 systems is to find 1st dataset and then generate the data for the 2nd source system based on it.

The examples of different source systems:

* Based on customer: Retail or Business stack.
* Based on geographical data: different systems for different countries/regions.
* Based on brand: if the company sells products under different brands
* Based on payment type: Cash and Card

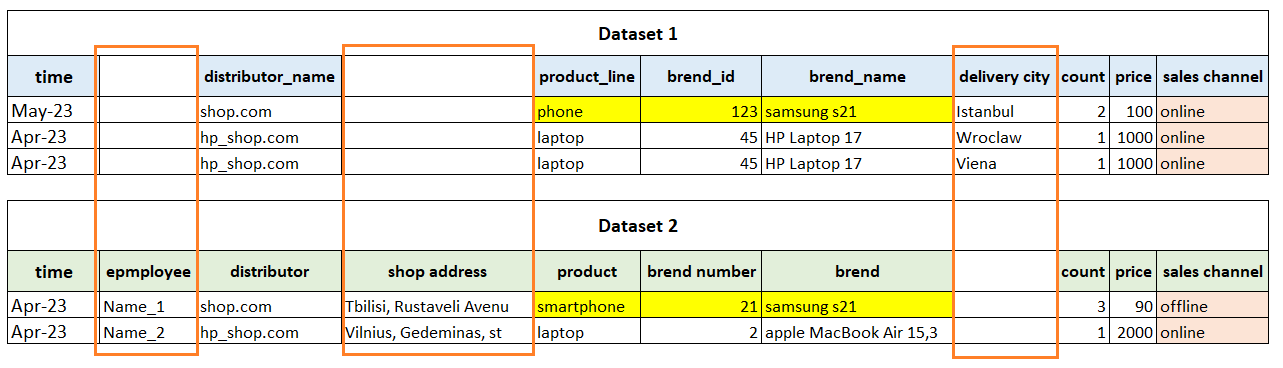
Pay attention:

1. 2 different source systems should have different attributes
2. The same rows in two sources (if any) must have different IDs

So, if you have only one source, you can duplicate it by making a copy, removing one column, and adding two new columns. In order to preserve identical rows, you can modify at least one attribute and the ID.

What should be in every source systems:

* Facts (sales transactions).
* Data for at least 1 dimension should be presented in both data sets, for example for Customer or Products -> they will be loaded in 1 table in the DWH.
* The rest dimensional data may be presented with various options:
  + Presented only in one source system: for example, full info about employees will be presented only in 1st system, the 2nd system will contain only employee’s natural identifiers.
  + Full data is presented in both systems.
  + The data is presented partially in both systems (part of the attributes in 1st one and another part in 2nd system). Then they will be combined in the DWH.

D

# **DWH Architecture and Naming**

We are going to build hybrid DWH with both layers 3NF - Inmon and Dimensional Model (DM) - Kimball.

Data Flow: csv files -> SA (Source Abstraction - it is staging area) layer -> 3NF layer -> DM layer

You should have 5 schemas:

* BL\_DM (Business Layer Dimensional Model). Star schema's (denormalized) tables will be stored there. Naming: FCT\_ and DIM\_ tables; PK - PRODUCT\_SURR\_ID; natural key - PRODUCT\_SRC\_ID or SOURCE\_ID (which was generated for 3NF schema); source\_system ('BL\_3NF') and source\_table ('CE\_PRODUCTS') columns are obligatory there with corresponding values.
* BL\_3NF (Business Layer 3NF). Normalized tables will be stored there. Naming: CE\_ (Core Entity); PK - PRODUCT\_ID; natural key PRODUCT\_SRC\_ID or SOURCE\_ID (value from source system (SA schemas); source\_system (‘SA\_SOURCE\_NAME’)/source\_table (‘SRC\_PRODUCTS’) are obligatory there with corresponding values.
* SA\_<SOURCE\_SYSTEM\_1\_NAME> and SA\_<SOURCE\_SYSTEM\_2\_NAME> (name of the source system). There will be external tables with prefix SRC\_<FILE\_NAME> for both source systems.
* BL\_CL (Business Layer Cleansing). There will be all objects which are used for ETL: packages, procedures, functions, work/cleansing tables (if needed).

# **Natural Keys VS Surrogate Keys**

When you design DWH, tables typically have a column or a number of columns that are known as the primary key. The primary key is a unique value that identifies each record.  Sometimes the primary key is made up of real data and these are normally referred to as natural keys, while other times the key is generated when a new record is inserted into a table.  When a primary key is generated at runtime, it is called a surrogate key.   A surrogate key is typically a numeric value.

A natural key is a single column or set of columns that uniquely identifies a single record in a table, where the key columns are made up of real data.  When we say “Real data” we mean data that has meaning and occurs naturally in the world of data.  A natural key is a column value that has a relationship with the rest of the column values in a given data record.   Here are some examples of natural keys values: Passport Number, Social Security Number, ISBN, and TaxId.

A surrogate key like a natural key is a column that uniquely identifies a single record in a table.  But this is where the similarity stops. They are keys that don’t have a natural relationship with the rest of the columns in a table.  The surrogate key is just a value that is generated and then stored with the rest of the columns in a record.  The key value is typically generated at run time right before the record is inserted into a table.   It is sometimes also referred to as a dumb key, because there is no meaning associated with the value.  Surrogate keys are commonly a numeric number.

## Surrogate Key Pros and Cons

**Pros:**

* The primary key has no business intelligence built into it. Meaning you cannot derive any meaning, or relationship between the surrogate key and the rest of the data columns in a row.
* If your business rules change, which would require you to update your natural key, this can be done easily without causing a cascade effect across all foreign key relationships.   By using a surrogate key instead of a natural key, the surrogate key is used in all foreign key relationships.  Surrogate keys will not be updated over time.
* Surrogate keys are typically integers, which only require 4 bytes to store, so the primary key index structure will be smaller in size than their natural key counter parts.  Having a small index structure means better performance for JOIN operations.

**Cons:**

* If foreign key tables use surrogate keys then you will be required to have a join to retrieve the real foreign key value.  Whereas if the foreign key table used a natural key then the natural key would be already be included in your table and no join would be required.  Of course, this I only true if you only needed the natural key column returned in your query
* Surrogate keys are typically not useful when searching for data since they have no meaning.

## Natural Key Pros and Cons

**Pros:**

* Will require less joins when you only need to return the key value of a foreign key table.   This is because the natural key will already be imbedded in your table.
* Easier to search because natural keys have meaning and will be stored in your table.  Without the natural key in your table, a search for records based on a natural key would require a join to the foreign key table to get the natural key.

**Cons:**

* Requires much more work to change a natural key, especially when foreign relationships have been built off the natural key.
* Your primary key index will be larger because natural keys are typically larger in size than surrogate keys.
* Since natural keys are typically larger in size then surrogate keys and are strings instead of integers joins between two tables on a natural key will take more time.

**In our DWH** we will use the concept of surrogate keys. As you know, we will have two main layers - 3NF layer and Star layer. The source key will be composite (source triplet) – for example, **employee\_src\_id (or source\_id), source\_system and source\_table.** On each layer, we will generate surrogate keys using sequences.

As a result, on 3NF, we will have two fields in each table - ***\_src\_id*** and ***\_id*** (for example, employee\_id and employee\_src\_id), where ***\_src\_id*** will be a natural key (to be more precise - part of the natural key), and ***\_id*** - a surrogate.

In the same way, on the Star (DM) layer, we will also have two fields in each table - ***source\_id (or\_src\_id)*** and ***\_surr\_id***, where **\_src\_id** is the surrogate key generated in the previous step (on BL\_3NF), and ***\_surr\_id*** is the surrogate key generated at the stage of loading into the Star. The primary key on 3NF will be ***\_id*** field (for SCD2 table – ***\_id + start\_dt***), and on the Star layer - ***\_surr\_id*** field. As a result, each layer will use its own non-overlapping surrogate keys. Also, using the chain ***\_surr\_id*** -> ***\_id*** -> ***\_ src\_id***, we can always go to the natural business key, knowing the surrogate key.

# **The Difference Between Full and Incremental Loading**

There are two primary methods to load data into a warehouse:

## 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.

**In our DWH** we will be using the Batch incremental load. At the end of Module 3, you will need to demonstrate how your incremental logic works and that it works correctly.