**What is a Data Warehouse?**

A data warehouse (DW or DWH) is a complex system that stores historical and cumulative data used for forecasting, reporting, and data analysis. It involves collecting, cleansing, and transforming data from different data streams and loading it into fact/dimensional tables.

A data warehouse represents a subject-oriented, integrated, time-variant, and non-volatile structure of data.

Focusing on the subject rather than on operations, the DWH integrates data from multiple sources giving the user a single source of information in a consistent format. Since it is non-volatile, it records all data changes as new entries without erasing its previous state. This feature is closely related to being time-variant, as it keeps a record of historical data, allowing you to examine changes over time.

## Data Warehouse Architecture

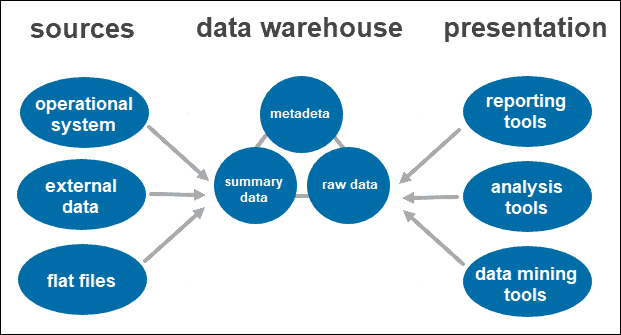
There are three ways you can construct a data warehouse system. These approaches are classified by the number of tiers in the architecture. Therefore, you can have a:

* Single-tier architecture
* Two-tier architecture
* Three-tier architecture

### Single-tier Data Warehouse Architecture

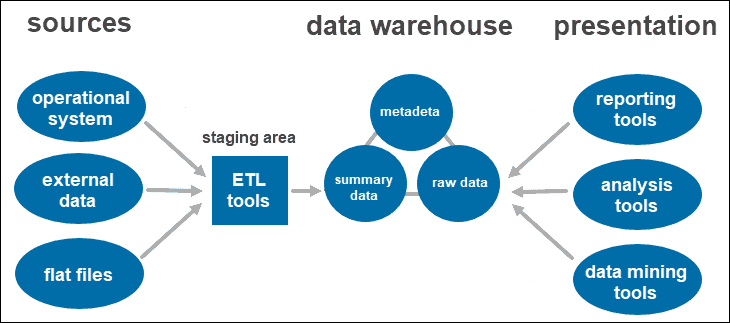
The single-tier architecture is not a frequently practiced approach. The main goal of having such an architecture is to remove redundancy by minimizing the amount of data stored.

Its primary disadvantage is that it doesn’t have a component that separates analytical and [transactional processing](https://phoenixnap.com/kb/oltp-database).



### Two-tier Data Warehouse Architecture

A two-tier architecture includes a staging area for all data sources, before the data warehouse layer. By adding a staging area between the sources and the storage repository, you ensure all data loaded into the warehouse is cleansed and in the appropriate format.



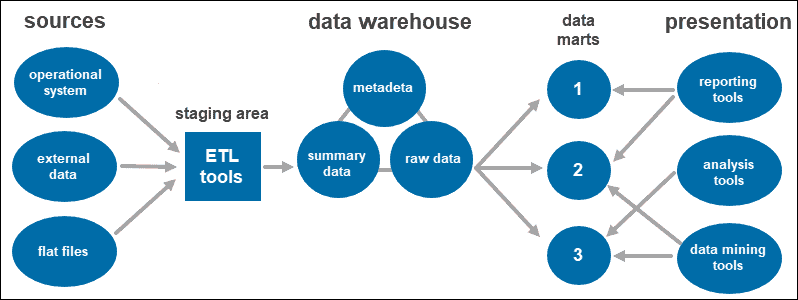
This approach has certain network limitations. Additionally, you cannot expand it to support a larger number of users.

### Three-tier Data Warehouse Architecture

The three-tier approach is the most widely used architecture for data warehouse systems.

Essentially, it consists of three tiers:

1. **The bottom tier** is the database of the warehouse, where the cleansed and transformed data is loaded.
2. **The middle tier** is the application layer giving an abstracted view of the database. It arranges the data to make it more suitable for analysis. This is done with an OLAP server, implemented using the ROLAP or MOLAP model.
3. **The top-tier** is where the user accesses and interacts with the data. It represents the front-end client layer. You can use reporting tools, query, analysis or data mining tools.



2. What are the stages of Datawarehousing?

There are 4 stages of a data warehouse that help in finding out and understanding how the data changes in the warehouse.

### **4 Stages of Data Warehousing**

* **Offline Operational Database:** This is the initial stage where data is simply copied to a server from an operating system. It is done so that data loading, processing, and reporting do not affect the performance of the operational system.
* **Offline Data Warehouse:** In this stage, all the data warehouses are updated on a regular time cycle from the operational database to get actionable business insights.
* **Real-time Data Warehouse:** In this stage, data warehouses are updated based on transaction or event basis. Whenever a transaction takes place in an operational database, it is updated in the data warehouse.
* **Integrated Data Warehouse:** This is the final stage where all the transactions which are used daily by the organization are passed back into the operational system. Each transaction that takes place in the operational database is updated in the warehouse simultaneously. These transactions are then forwarded to the operational database.
  1. What is surrogate key?

Data warehouse surrogate keys are sequentially generated meaningless numbers associated with each and every record in the data warehouse. These surrogate keys are used to join [dimension](https://dwgeek.com/types-of-dimension-tables-data-warehouse.html/) and [fact tables](https://dwgeek.com/types-of-fact-tables-data-warehouse.html/).

* Usually, [database sequences](https://dwgeek.com/netezza-sequence-create-use.html/) are used to generate surrogate key so it is always **unique number**
* Surrogate keys **cannot be NULLs**. Surrogate key are never populated with NULL values.
* It does not hold any meaning in data warehouse, often called meaningless numbers. It is just **sequentially generated INTEGER** number for better lookup and faster joins.

#### Why surrogate keys are used in Data warehouse?

Basically, surrogate key is an artificial key that is used as a substitute for natural key (NK) defined in data warehouse tables. We can use natural key or business keys as a primary key for tables. However, it is not recommended because of following reasons:

* **Natural keys (NK)** or **Business keys** are generally alphanumeric values that is not suitable for index as traversing become slower. For example, prod123, prod231 etc
* Business keys are often reused after sometime. It will cause the problem as in data warehouse we maintain historic data as well as current data.

For example, product codes can be revised and reused after few years. It will become difficult to differentiate current products and historic products. To avoid such a situation, surrogate keys are used.

**Data Warehouse Surrogate Key examples**

Surrogate Keys are integers that are assigned sequentially in the dimension table which can be used as primary key. The surrogate key column could be identity column or database sequences are used.

Below is the sample example of surrogate key:

|  |  |  |  |
| --- | --- | --- | --- |
| **PATIENT\_SK** | **PATIENT\_ID** | **PATIENT\_NAME** | **PATIENT\_AGE** |
| 1 | P001 | ABC | 20 |
| 2 | P002 | BCD | 25 |
| 3 | P003 | CDE | 19 |
| 4 | P004 | DEF | 45 |

4. Types of Dimension Tables based on frequency of data change

## ****Types of Dimensions****

**Slowly Changing Dimensions**– Dimension attributes that change slowly over a period of time rather than changing regularly is grouped as SCDs.  Attributes like name, address can change but not too often.

These attributes can change over a period of time and that will get combined as a slowly changing dimension. Consider an example where a person is changing from one city to another. Now there are 3 ways to change the address;

Type 1  is to over write the old value, Type 2 is to add a new row and Type 3 is to create a new column.

**Type 1**

The advantage of type 1 is that it is very easy to follow and it results  in huge space savings and hence cost savings. The disadvantage is that no history is maintained.

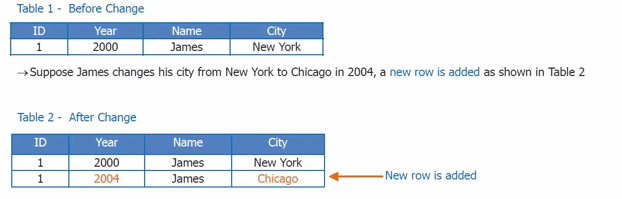
[[](https://www.edureka.co/datawarehousing-self-paced)](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)

### [Data Warehousing Certification Training](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)

* *[Course Type](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)*
* *[Real-life Case Studies](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)*
* *[Assignments](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)*
* *[Lifetime Access](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)*

[Explore Curriculum](https://www.edureka.co/datawarehousing-self-paced" \t "_blank)

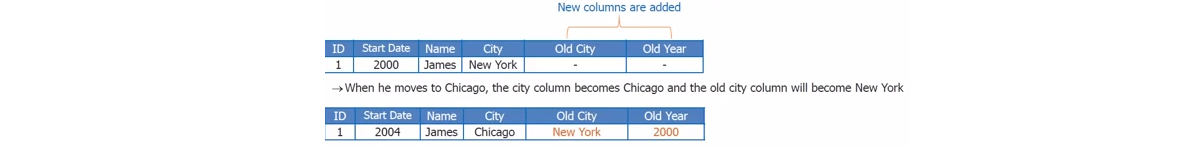
**Type 2**



The advantage of type 2 is that the complete history is maintained. The only disadvantage lies in the huge space allocation because the entire history right from the start has to be maintained.

**Type 3**

The best approach could be to add a new column where you add two new columns. In this case keeping a tracking of the history becomes very easy.

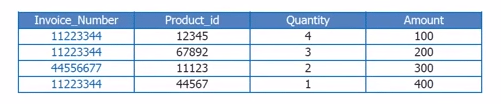
**Conformed Dimension-**This is used in multiple locations**.**It helps in creating consistency so that the same can be maintained across the fact tables. Different tables can use the table across the fact table and it can help in creating different reports.

Next

For example, there are two fact tables. Fact table 1 is to determine the number of products sold by geography. This table will calculate just the number of products by geography and fact table 2 will determine the revenue generated by customer. Both are dependent on the product which contains product Id, name and source.

There is the geography dimension and customer dimension which are being shared by two fact tables. The revenue fact gives the revenue generated by both the geography and the customer, while the product units fact gives number of units sold in the geography to a customer.

**Degenerate Dimension**– A degenerate dimension is when the dimension attribute is stored as part of the fact table and not in a separate table. Product id comes from product dimension table. Invoice number is a standalone attribute and has no other attributes associated with it. An invoice number can be crucial since the business would want to know the quantity of the products.



**Junk Dimension**– It is a single table with a combination of different and unrelated attributes to avoid having a large number of foreign keys in the fact table. They are often created to manage the foreign keys created by rapidly changing dimensions.

**Role play dimension**– It is a table that has multiple valid relationships with a fact table. For example, a fact table may include foreign keys for both ship date and delivery date. But the same attributes apply to each foreign key so the same tables can be joined to the foreign keys.

**What are slowly changing dimensions?**

When organising a datawarehouse into Kimball-style star schemas, you relate fact records to a specific dimension record with its related attributes. But what if the information in the dimension changes? Do you now associate all fact records with the new value? Do you ignore the change to keep historical accuracy? Or do you treat facts before the dimension change differently to those after?

It is this decision that determines whether to make your dimension a slowly changing one. There are several different types of SCD depending on how you treat incoming change.

**What are the types of SCD?**

Very simply, there are 6 types of Slowly Changing Dimension that are commonly used, they are as follows:

* Type 0 – Fixed Dimension
  + No changes allowed, dimension never changes
* Type 1 – No History
  + Update record directly, there is no record of historical values, only current state
* Type 2 – Row Versioning
  + Track changes as version records with current flag & active dates and other metadata
* Type 3 – Previous Value column
  + Track change to a specific attribute, add a column to show the previous value, which is updated as further changes occur
* Type 4 – History Table
  + Show current value in dimension table but track all changes in separate table
* Type 6 – Hybrid SCD
  + Utilise techniques from SCD Types 1, 2 and 3 to track change

In reality, only types 0, 1 and 2 are widely used, with the others reserved for very specific requirements. Confusingly, there is no SCD type 5 in commonly agreed definitions.

After you have implemented your chosen dimension type, you can then point your fact records at the relevant business or surrogate key. Surrogate keys in these examples relate to a specific historical version of the record, removing join complexity from later data structures.

**Practical Examples**

We have a very simple ‘customer’ dimension, with just 2 attributes – Customer Name and Country:

[simple customer dimension](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_07E50CFA.png)

However, Bob has just informed us that he has now moved to the US and we want to update our dimension record to reflect this. We can see how the different SCD types will handle this change and the pro/cons of each method.

**Type 0**

Our table remains the same. This means our existing reports will continue to show the same figures, maybe it is a business requirement that each customer is always allocated to the country they signed up from.

All future transactions associated to Bob will also be allocated to the ‘United Kingdom’ country.

**Type 1**

The table is updated to reflect Bob’s new country:

[customer dimension table](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_6A25BE8D.png)

All fact records associated with Bob will now be associated with the ‘United States’ country, regardless of when they occurred.

We often just want to see the current value of a dimension attribute – it could be that the only dimension changes that occur are corrections to mistakes, maybe there is no requirement for historical reporting.

**Type 2**

In order to support type 2 changes, we need to add four columns to our table:

· Surrogate Key – the original ID will no longer be sufficient to identify the specific record we require, we therefore need to create a new ID that the fact records can join to specifically.

· Current Flag – A quick method of returning only the current version of each record

· Start Date – The date from which the specific historical version is active

· End Date – The date to which the specific historical version record is active

With these elements in place, our table will now look like:

[](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_49324BE6.png)

This method is very powerful – you maintain the history for the entire record and can easily perform change-over-time analysis. However, it also comes with more maintenance overhead, increased storage requirement and potential performance impacts if used on very large dimensions.

Type 2 is the most common method of tracking change in data warehouses.

**Type 3**

Here, we add a new column called “Previous Country” to track what the last value for our attribute was.

[add new columns to customer dimension](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_3B40D056.png)

Note how this will only provide a single historical value for Country. If the customer changes his name, we will not be able to track it without adding a new column. Likewise, if Bob moved country again, we would either need to add further “Previous Previous Country” columns or lose the fact that he once lived in the United Kingdom.

**Type 4**

There is no change to our existing table here, we simply update the record as if a Type 1 change had occurred. However, we simultaneously maintain a history table to keep track of these changes:

Our Dimension table reads:

[tracking dimension table](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_7ABDB97B.png)

Whilst our Type 4 historical table is created as:

[](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_7A518686.png)

Depending on your requirements, you may place both ID and Surrogate Key onto the fact record so that you can optimise performance whilst maintaining functionality.

Separating the historical data makes your dimensions smaller and therefore reduces complexity and improves performance if the majority of uses only need the current value.

However, if you do require historical values, this structure adds complexity and data redundancy overheads. It is generally assumed that the system will use Type 1 or Type 2 rather than Type 4.

**Type 6**

The ‘Hybrid’ method simply takes SCD types 1, 2 and 3 and applies all techniques. We would maintain a history of all changes whilst simultaneously updating a “current value” column on all records.

[](https://adatis.co.uk/wp-content/uploads/historic/simonwhiteley_image_60E9834C.png)

This gives you the ability to provide an element of change comparison without additional calculation, whilst still maintaining a full, detailed history of all changes in the system.

Personally, if this requirement came up, I would avoid the data redundancy of this extra column and simply calculate the current value using the “LAST\_VALUE()” window function at run-time. Although this depends on your priorities between data storage and direct querying performance.