**Data Ware House**

1. It is a warehouse with large set of data and there are rules that governs how we build our data, organize and store data, so it can be used for better analysis.

2. Dw not same as database, rather build on database

3. Source for data warehouse comes from different source (operational system, external source)

4. data are copied from source, which means data still exist in source

**Use case:**

1. Support making decisions in a data driven manner, rather than rely solely on experience intuition and hunches

2. Data is stored in one place rather than transitional and operational application where we get that data from.

**Data warehouse vs DataLake:**

1. DW build on top of RDBMS vs datalake build on top of big data

2. Datalake have 3 v's

- Volume (handle more volume than data warehouse)

- Velocity (handles data with high speed)

- Variety (Handles different data structure eg: Audio, video, un-structured whereas DW handles them to some extend using blob data type)

Data Lake consist of raw data, on top of which ETL is performed and analysis is done. It eliminate the need for Data Warehouse. Cost efficient.

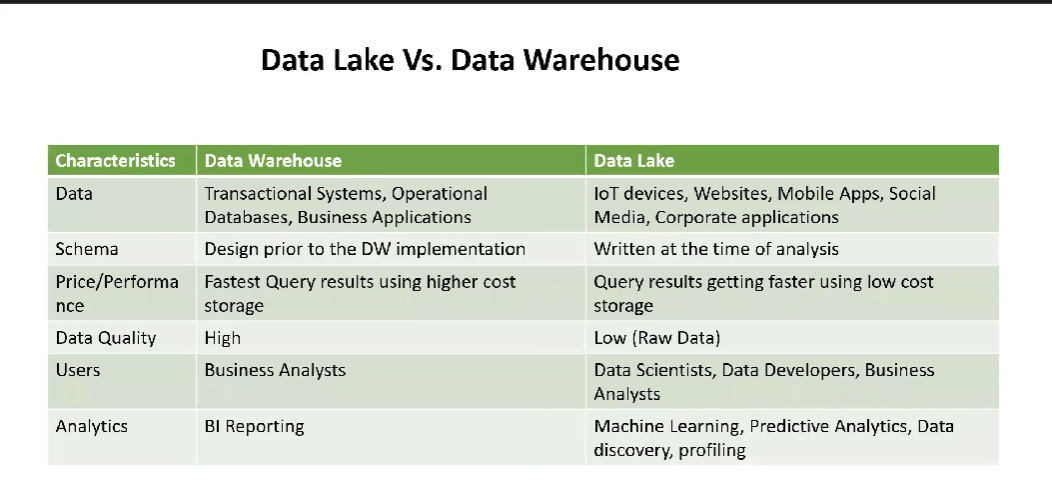
Data Lake can handle unstructured data.

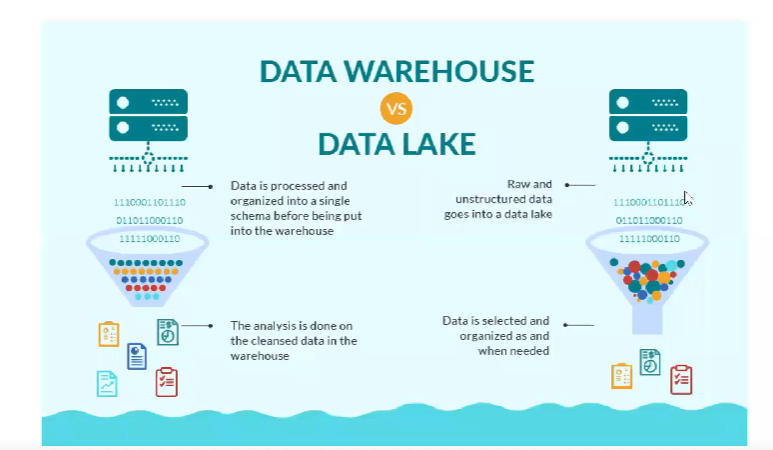
Source -> Data Lake -> ETL -> Reporting Tool

Source -> Data Lake -> ETL -> Data Warehouse -> DataMart -> Reporting Tool

**Data warehouse vs Data Virtualization:**

1. DV unlike DW, do not copy data, it directly modifies on original data source.





**Data Warehouse Architecture:**

***Data source (Suppliers) -> ETL -> Data warehouse (Wholsalers) -> ETL -> Data Mart(downstream)(Retailers)***

***Data Warehouse***

***Centralized Component-Based***

***EDW Data Lake Architected Non- Architected***

***Relational\_db Hadoop (AWS,S3…) Dws+DMs DMs Only***

***Speacialized\_db***

***Dependent DMs Front end DMs Federated-EDW***

***DW Bus (Independent DMs)***

***ODS:***

ODS – Operational data store

Integrates data from multiple sources. Emphasis on current operational data. No history of data.

Often real time data. Data will be transmitted once transaction happen, will not wait for data refresh as in DW.

***Source -> ODS -> DWs***

***DW:***

***Staging Layer:***

* Landing zone of source data
* ‘E’ in ETL (extract)
* 2 varieties of staging

1. Non-persistence: source -> load to staging layer -> User access layer -> Empty staging layer
2. Persistence: source -> load to staging layer -> User access layer (Does not empty staging layer. Data will be added on top of existing staging data. This will increase storage space)

**The scenario:**

A large state university system has four different campuses located all across the state. Each of these campuses operates as, essentially, its own self-contained university, which means:

* Each campus has its own IT organization that operates and maintains its own applications for that campus only
* Each campus' applications include the following key software systems for:
  + Faculty management
  + Student admissions, grading, and administration
  + Facilities management (e.g., buildings and classrooms)
* The state university's "main campus" is one of the four individual campuses, and is located in a small town in the center of the state
* The main campus IT organization has responsibilities for not only that campus, but also support for the entire state university system as a whole
* The main campus IT organization is just starting on a data warehousing initiative that is intended to serve the needs of 1) individual campuses as well as 2) the state university system as a whole.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Solution 1:**

Source -> DWs -> 4 diff DMs (campus1, campus2, campus3, campus4…)

**Solution 2:**

Source -> DWs -> 3 diff DMs (Faculty, Student, Facilities)

**ETL:**

**Extract:**

* Pull data from source. Traditionally done in batches
* Raw data…error and all. Land in DW staging layer

**Transform:**

* Uniform the data. Can be complex

**Load:**

* Store uniform data in user layer.

**ELT: Extract Load and then Transform**

* “Blast” data into big data environment
* Raw form in Hadoop, AWS S3, etc.
* Use big data environment computing power to transform when needed

**Two models of ETL:**

* Initial (One time load – History data)
* Incremental (Refresh data)

1. Append (Add new data to existing)
2. In-place update (Maintain same row number, just update the modified row)
3. Complete replacement (Even for small change it will replace complete data)
4. Rolling append (remove old equivalent data and append new data)

**Transformation:**

* Uniformity (Whatever the raw data, there should be uniformity in the data student, customer…)
* Re-structuring (very well engineered set of data structure)

Common Transformation Model:

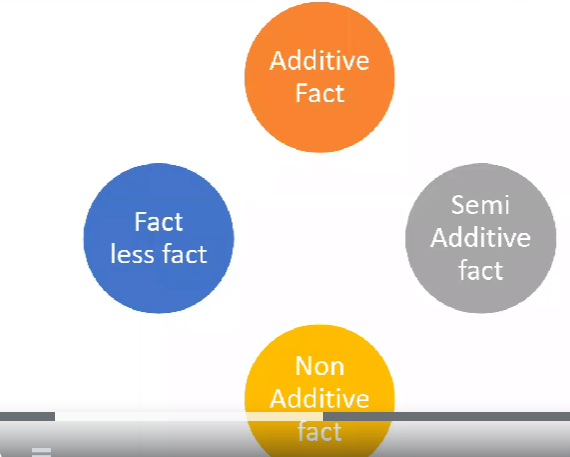
1. Data value unification (Assistant Prof , AP)
2. Data type and size unification (char(30), char(3))
3. De-Duplication (remove duplicate entry for same person)
4. Dropping column (vertical slicing)
5. Value-based row filtering (horizontal slicing)
6. Correcting known values

**Design Engineering**

**Principle of dimensionality:**

1. Fact - One or more measurement
2. Dimension - Dimensional Context of description

**Fact Table:**

****

**Fact is not same as Fact table**

3 types of measurements can be stored in fact table,

* **Additive:** Measurements in a fact table that can be summed up across all dimensions (salary, credit score etc which can be added across and provide valuable result)
* **Semi-additive:** Measurements in a fact table that can be summed up across only a few dimensions. (eg: we cannot sum up current balance across Acct Id. If we ask balance for Id 21653 we will say that 22000, not 22000+80000 )
* **non-additive:** Facts that cannot be summed up across any dimension key.  % and ratio columns are non-addictive facts
* **Factless:** A fact table without any measures is called the factless fact table. It acts as a bridge between dimension keys. Record occurrence of transaction that has no measurements. Record cover or eligibility relationship

Types of fact table:

1. Transactional: Measurements from transaction
2. Periodic snapshot: Track a given measurements at regular interval daily, weekly, monthly
3. Accumulation snapshot: Track progress of business process through formally defined stages

**Rules of fact in fact table:**

Two facts(measurements) can be stored in same table when it satisfies below rules,

Rule 1: two facts should have same level of granularity

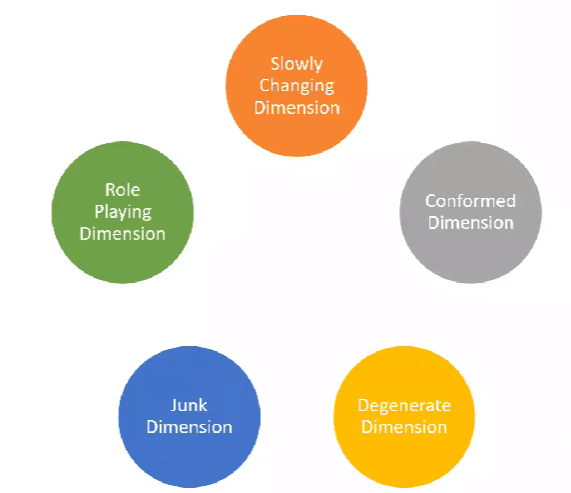
Rule 2 : two facts should occur at same instance

Eg: Tuition Payment & Tuition Bill amount – same granularity, occur at different time.

Tuition Bill amount & Activities bill amount – same granular, occur at same time

Primary key in fact table is combination of all foreign keys relating back to dimension table.

**Dimension Table:**

****

* **Slowly changing Dimension:** Dimensions that are susceptible to change are called slowly changing dimensions (SCDs).
* **Confirmed Dimension:** A conformed dimension can be associated with different fact tables, maintaining the same meaning with all of them. In constellation-type data warehouse designs with multiple fact tables, conformed dimensions make cross-domain queries possible.
* **Degenerate Dimension:** data that is dimensional in nature but stored in a fact table. Duplicate columns from different table into one table
* **Junk Dimension:** facts often have indicator attributes like flags, Boolean values, or some other set of values that do not make sense as a dimension because of their low cardinalities. To avoid creating small dimensions for each of these attributes and increasing the number and sizes of the fact tables unnecessarily, a junk dimension is often created to gather all these attributes into a single table. Store 0/1 for transaction success or failed.
* **Role Playing Dimension**: Role-playing dimensions are used by different fact tables, just like conformed dimensions. But unlike conformed dimensions, they have different meanings depending on the fact table or even the field within the fact table. A dimension that can play different roles in a fact table depending on the context.

**Hierarchical vs flat dimension**

For example, if we have a product hierarchy as shown below,

**Product Category**

**Product Family**

**Product**

* **Start schema:**

One dimension table will be created and all 3 dimensions from same hierarchy is present in one table

* Star Schema:

Three dimensions table created for 3 dimensions

**A screen shot of a computer

Description automatically generated with low confidence**

**A screen shot of a computer

Description automatically generated with low confidence**

**RDBMS Keys:**

* Primary and Foreign key
* Natural and surrogate key

Primary Key: Uniquely identify a row in a table. Can be single or multiple columns in one db. (eg: Faculty\_ID)

Foreign key: Some other table’s PK. Used to integrate logical relationship

Natural key: Cryptic (random 6 digit num) or understandable (FACULTY\_ID). Travel from source system along with rest of the data.

Surrogate Key: Generated by Database itself or supplemental “key management” system. (eg: Faculty\_Key)

**Best Practices:**

1. Use Surrogate key (create key within DW).
2. Keep natural key as secondary key in dimension table, discard in fact table.

**Slowly Changing Dimension**

**SCD Type1:** “In-place Update” ETL pattern. No history maintained. Same row & column maintained in table. Use case: Correcting errors. Disadvantage: Reports vary before and after change.

**SCD Type2:** Maintain version of row history for each new row. Complex architecture, but robust representation. To maintain version we can use version\_flag or eff\_date & expire\_date. In version\_flag we cannot maintain order.

**SCD Type3:** Add new column instead of new row to reflect changes. Prev\_div (NORTH,SOUTH) current\_div (EAST,WEST). Mostly used when we are sure about the number of occurrences it will change. Not suited for student address change as we are not sure abt the occurrence.

**ETL Best Practices:**

1. Limit incoming data from source (new and modified data). This is done by transactional data timestamp in source (GG\_TS).
2. Process dimension table before Fact table
3. Opportunities for parallel processing

**Top-Down (Inmon) Approach:**

Source -> ETL -> DWH -> ETL -> DM

**Bottom-Up (Kimball)**

Source -> ETL -> DM -> ETL -> DWH

**Conformed Dimension :** Dimension table shared by different fact table. In Bottom-Up approach the dimensions from different Datamart(DM) is combined by conformed dimension table.

**Enterprise Relationship Model:**

A picture containing text, screenshot, indoor, design

Description automatically generated

**Normalization :** Normalization is the process which divided larger table into smaller tables to minimize data and avoid data redundancy.

* Insertion Anomaly: If a table contain student and course details, course data will be repeated for millions of students thus occupying more disk space.
* Deletion Anomaly: If a particular student detail is deleted it deletes corresponding course details also. If all student details deleted, then all course data also removed.
* Update Anomaly: If the course name changes from Coding to Java, it need to be updated in all the millions of record. If it didn’t update properly, it contain both Coding and Java.

**1NF (First Normal Form):** Column cell cannot hold multiple values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student\_ID (PK) | Name | State | Region | Course |
| 100 | Amit | LA | US | Accounting, Coding |

The above example violates first Normal Form.

**2 NF (Second Normal Form):** The table should have achieved first normal form. Redundant data across multiple rows of a table must be moved to separate table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_ID (PK)** | **Name** | **State** | **Region** | **Course** |
| 100 | Amit | LA | US | Accounting |
| 100 | Amit | LA | US | Coding |

In the above example, student\_ID can no more act as a unique key as a single student have multiple course. So we might need to use composite key, combination of two or more PK(student ID & Course ID) to make it unique. Second form should avoid composite key. To solve this, redundant data should be kept in separate table and create relationship using PK & FK.

**Composite key indicate many-to-many relationship.**

**Student table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_ID (PK)** | **Name** | **State** | **Region** |
| 100 | Amit | LA | US |

**Course table:**

|  |  |
| --- | --- |
| **Course ID** | **Course** |
| AC01 | Accounting |
| COD1 | Coding |

**Student-Course table:**

|  |  |
| --- | --- |
| **Student ID** | **Course ID** |
| 100 | AC01 |
| 100 | COD1 |

**3 NF (Second Normal Form):** For this table should already have attained second normal form. Eliminate fields that do not depend on the Primary Key. We need to eliminate transitive dependency.

If a non-key column does not depend on another non-key column that is called transitive dependency.

The columns which are not directly dependent on PK column can be separated out to another table.

**Student table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_ID (PK)** | **Name** | **State** | **Region** |
| 100 | Amit | LA | US |

In above example, the state column is not directly dependent on student id, but it is dependent on Region column.

**Student table:**

|  |  |  |
| --- | --- | --- |
| **Student\_ID (PK)** | **Name** | **State\_ID** |
| 100 | Amit | 1 |

**State table:**

|  |  |  |
| --- | --- | --- |
| **State\_ID (PK)** | **State** | **Region** |
| 1 | LA | US |

**Denormalization:** Denormalization is the process where data from multiple tables are combined into one table, so the data retrieval will be faster.

**OLTP:** In OLTP, we use normalized form, as the insert, update & delete will happen faster. Eg: In ATM mini-statement taken after transaction.

**OLAP:** Suppose we need a data where electronic purchase happened in US, we need to combine multiple tables into one table and get result. But if we use denormalized form, data retrieval will be faster. Eg: Last 6 months transactions

**Challenges in Datawarehouse model:**

DW solutions take long time to deliver

Changes in business rule or architecture involve huge construction effort

Parent-child complexities, any changes in parent table will take time to reflect in child table.

If there is change in granularity of fact table, we might need to construct new fact table or modify existing fact table which might impact other reports which are not necessary.

With digitalization, we need to capture structured and unstructured data, huge volumes of data & real time data.

**Why Data Vault?**

To address modern problem Data Vault is an innovative data modeling methodology for large scale DW platforms. It was developed to address agility, flexibility and scalability issues. Also address granularity. Non-volatile for enterprise data.

**Hub:** Unique list of business keys

**Link:** Unique list on n to n relationship between business keys

**Satellite:** Contains descriptive historical data association with Hubs and Links.

**Hub Table attributes :** Similar to dimension table in dimension model

HK\_Studentcode(PK), student\_code, LOAD\_DTS, REC\_SRC

HK\_Facultycode(PK), Faculty\_code, LOAD\_DTS, REC\_SRC

**Link Table attributes:** Similar to Fact table in dimension model

HK\_Stu\_Fac(PK), HK\_Studentcode(FK), HK\_Facultycode(FK), LOAD\_DTS, REC\_SRC

**Satellite Table attributes :** Similar to type-2 table.Satellite provides descriptive information in hubs and links. Primary purpose of satellite is to track history in the system by capturing all the changes.

A hub can have one or multiple Satellite, but a satellite cannot be a parent to any other table.

Parent table of satellite can be HUB or LINK table.

HK\_Student\_Code(PK,FK) ), LOAD\_DTS, REC\_SRC, name, gender, email…