Youtube: Pyspark Telugu

Data Warehousing Concepts

1

Advantages of Warehousing

- High query performance
  - But not necessarily most current information
- Doesn't interfere with local processing at sources
  - Complex queries at warehouse
  - OLTP at information sources
- Information copied at warehouse
  - Can modify, annotate, summarize, restructure, etc.
  - Can store historical information
  - Security, no auditing

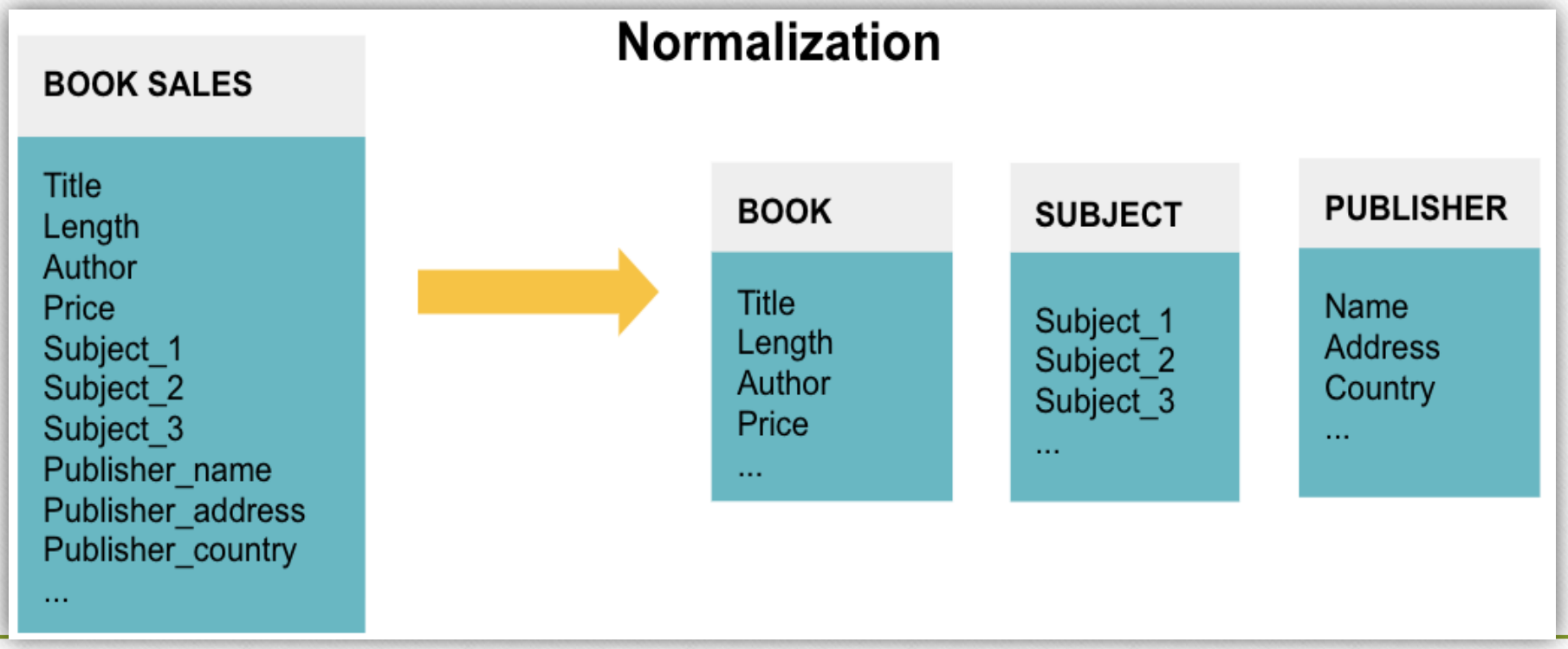
2

Normalization

Youtube: Pyspark Telugu

Database normalization is the process of structuring a database, usually a relational database, in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. It was first proposed by **E.F Codd** as part of his relational model.. We call a relation normalized if:

1. it does not contain any redundancy
2. it does not cause maintenance problems
3. it is an accurate representation of the data



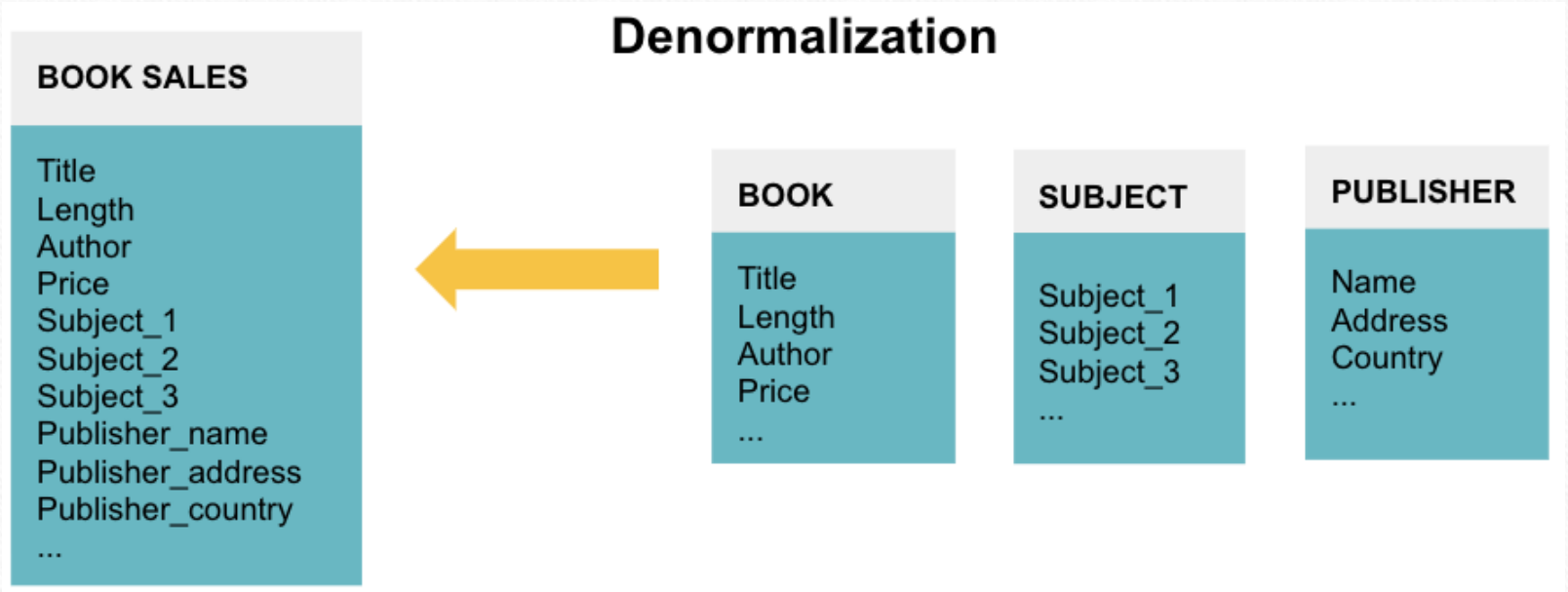
3

Denormalization

Youtube: Pyspark Telugu

Denormalization is a technique used by database administrators to optimize the efficiency of their database infrastructure. This method allows us to add redundant data into a normalized database to alleviate issues with database queries that merge data from several tables into a single table. The denormalization concept is based on the definition of normalization that is defined as arranging a database into tables correctly for a particular purpose.

When we normalize tables, we break them into multiple smaller tables. So when we want to retrieve data from multiple tables, we need to perform some kind of join operation on them. In that case, we use the denormalization technique that eliminates the drawback of normalization.



4

Types of Normalization

Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints. It is accomplished by applying some formal rules either by a process of synthesis (creating a new database design) or decomposition (improving an existing database design).

Here are the most commonly used normal forms:

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)

5

First normal form (1NF)

First normal form:

A relation is in first normal form if every attribute in every row can contain only one single (atomic) value. A university uses the following relation:

Students(Name,Class)

Databases in first normal form must follow certain rules . Firstly comma the data must be atomic. Atomic data is data that has been divided down into its smallest meaningful parts. In the example below, our database which is a normalised or in 0NF holds the name of our students in one field. Putting this into first normal form means breaking down the data into 3 separate fields..

Non-Atomic

Students	
Name	**Class**
Timothy Smith	Computer Science
Jessica Green	Computer Science
Jessica Green	Maths
Mark Lynch	Maths

Atomic

Students		
FName	**SName**	**Class**
Timothy	Smith	Computer Science
Jessica	Green	Computer Science
Jessica	Green	Maths
Mark	Lynch	Maths

6

Second normal form (2NF)

Second normal form:

A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.

A university uses the following relation:

1. It is in first normal form
2. All non-key attributes are fully functional dependent on the primary key

1NF

Students		
FName	**SName**	**Class**
Timothy	Smith	Computer Science
Jessica	Green	Computer Science
Jessica	Green	Maths
Mark	Lynch	Maths

2NF

Students			
ID	**FName**	**SName**	**Class**
1	Timothy	Smith	1
2	Jessica	Green	1
3	Jessica	Green	2
4	Mark	Lynch	2

Class	
ID	**Class**
1	Computer Science
2	Maths

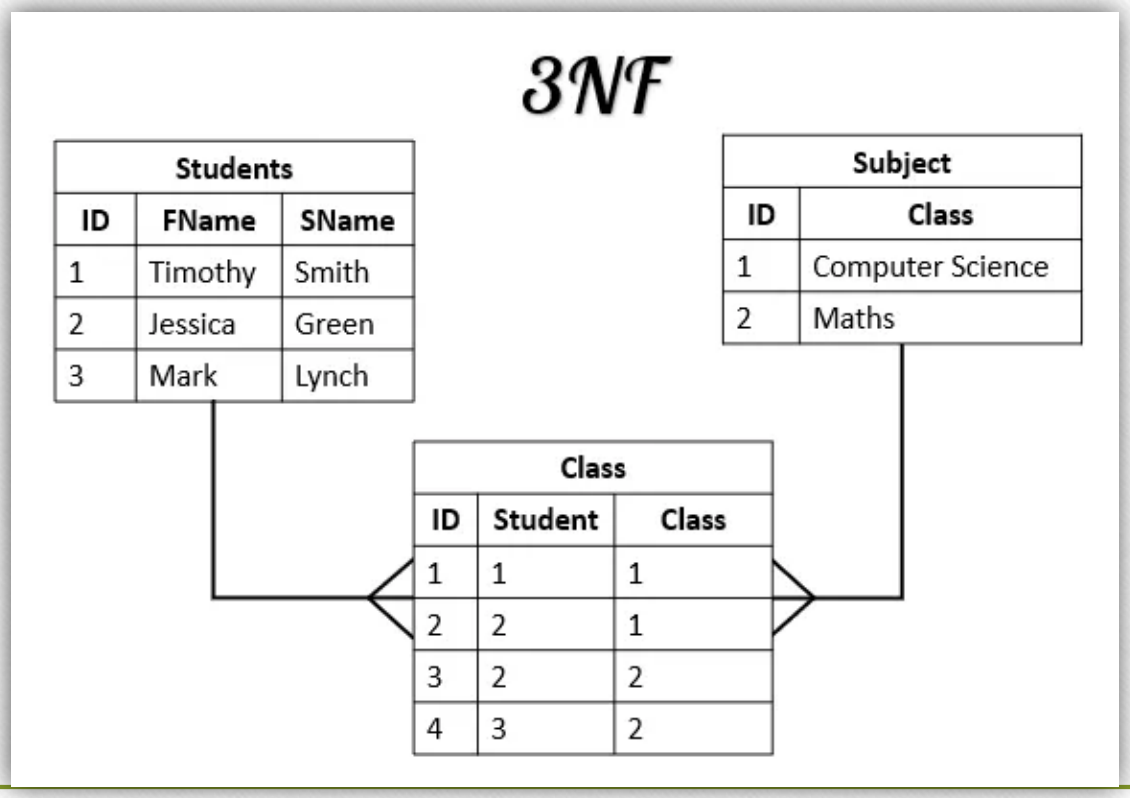
7

Third normal form (3NF)

A database is in third normal form if it satisfies the following conditions:

1. It is in second normal form
2. There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B



8

What is OLTP System?

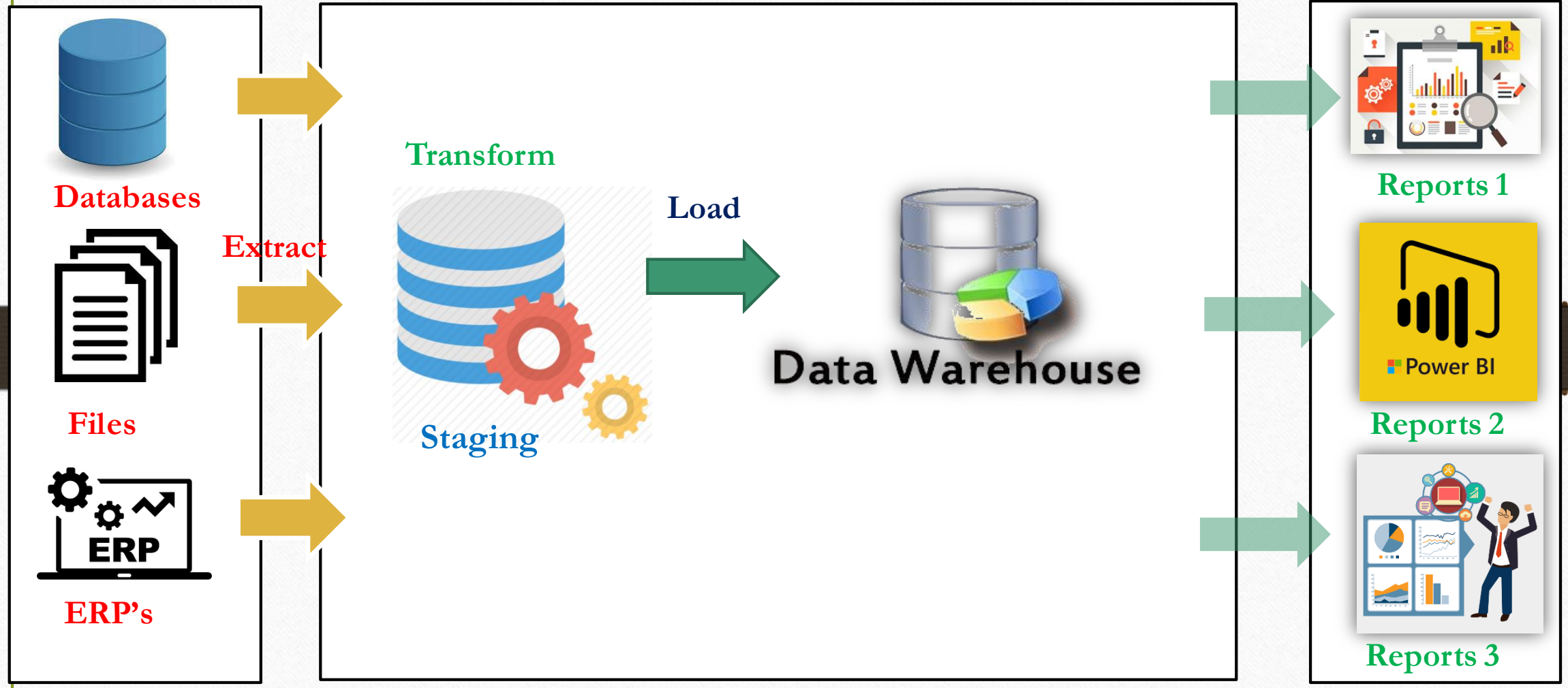
- Database Systems have been used traditionally for OLTP
  - clerical data processing tasks
  - detailed, up to date data
  - structured repetitive tasks
  - read/update a few records
  - isolation, recovery and integrity are critical

What is a OLAP?

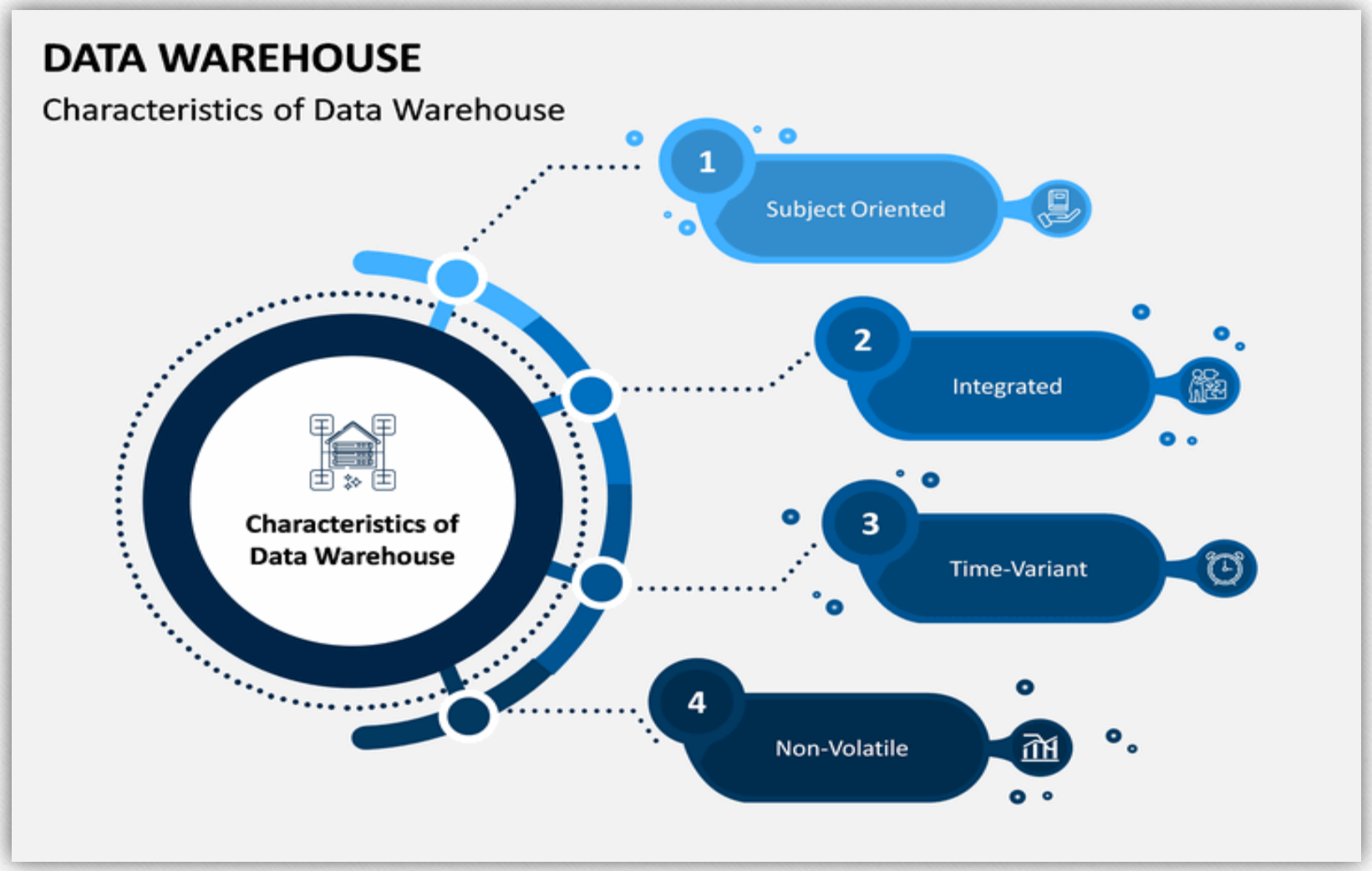
- Defined in many different ways, but not rigorously.
  - A decision support database that is maintained separately from the organization's operational database
  - Support information processing by providing a solid platform of consolidated, historical data for analysis.
- "**A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process**."—W. H. Inmon

Data Warehousing

Youtube: Pyspark Telugu

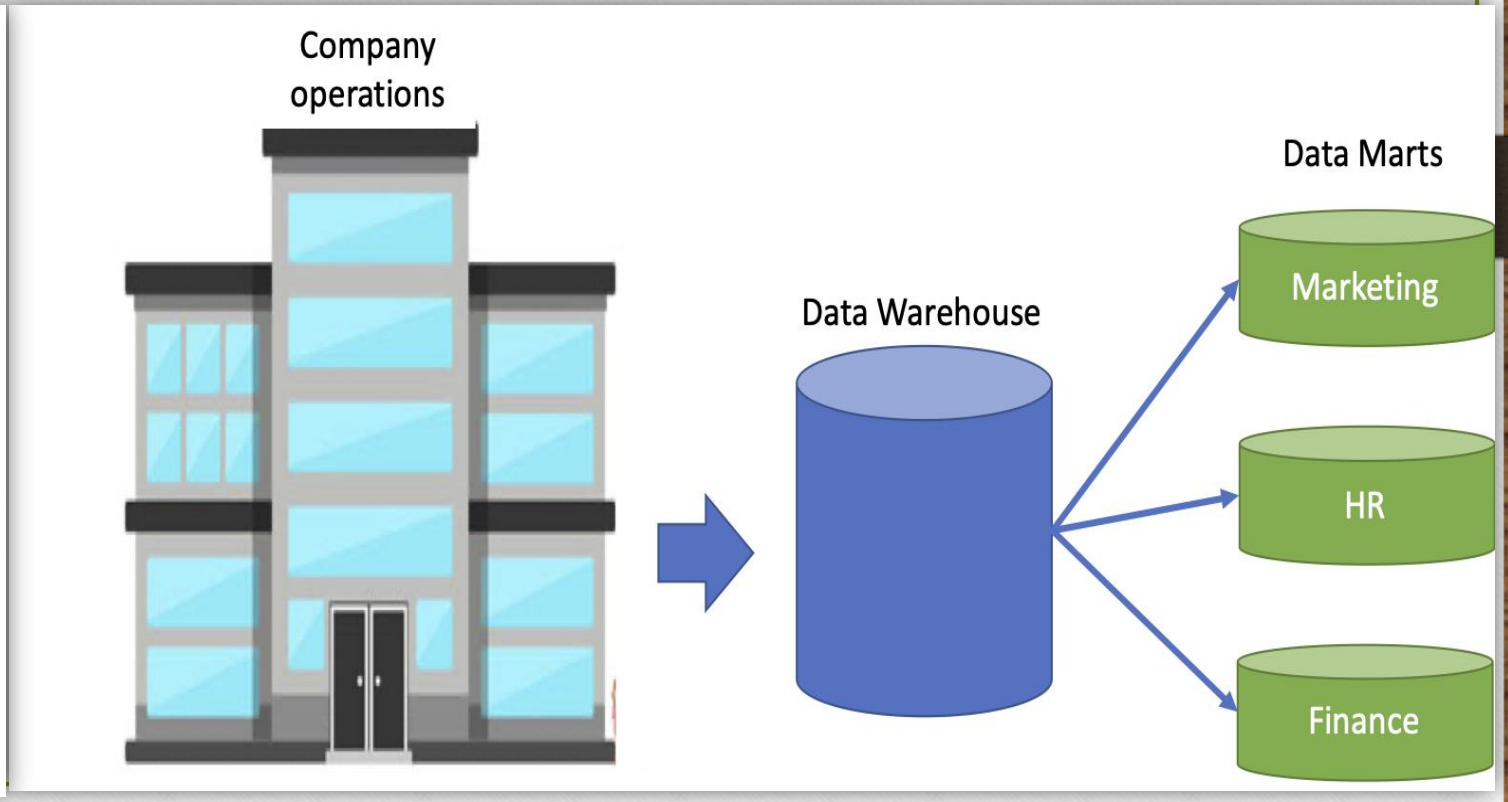


A data warehouse is a **subject-oriented**, **integrated**, **time-variant** and **non-volatile** collection of data in support of management's decision making process.



Subject-Oriented

A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.



Integrated

A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.

Time-Variant

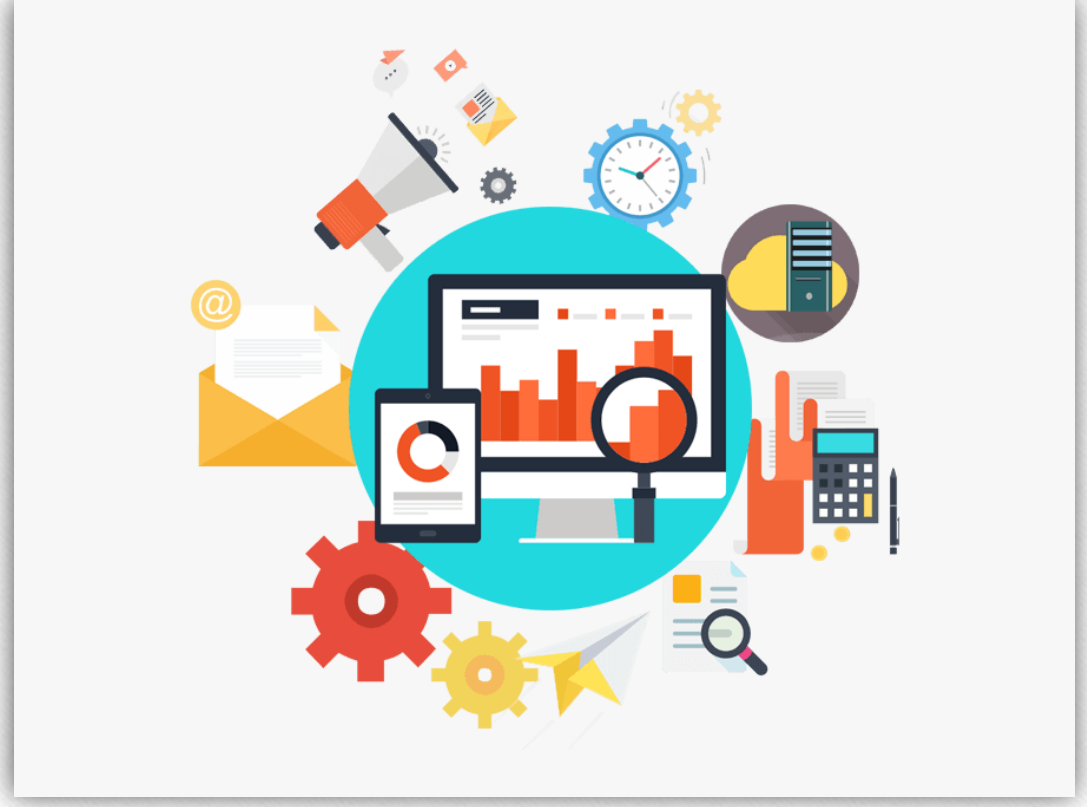
Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse. This contrasts with a transactions system, where often only the most recent data is kept. For example, a transaction system may hold the most recent address of a customer, where a data warehouse can hold all addresses associated with a customer.

Non-volatile

Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered. Ralph Kimball provided a more concise definition of a data warehouse:

No- Update

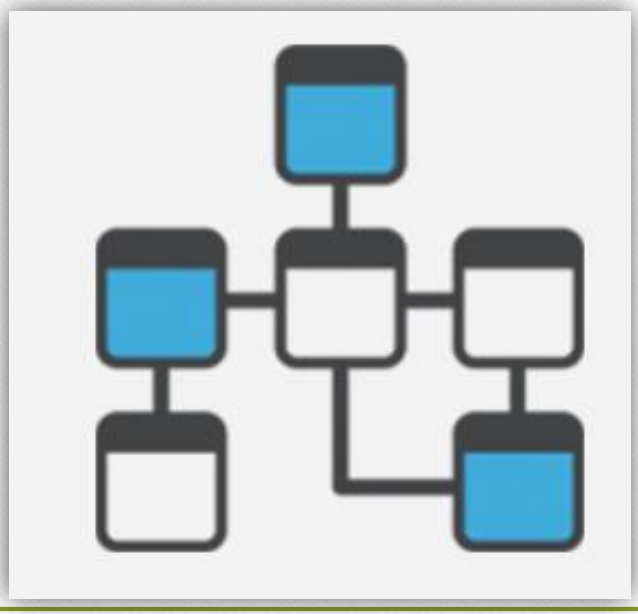
Only- Analysis



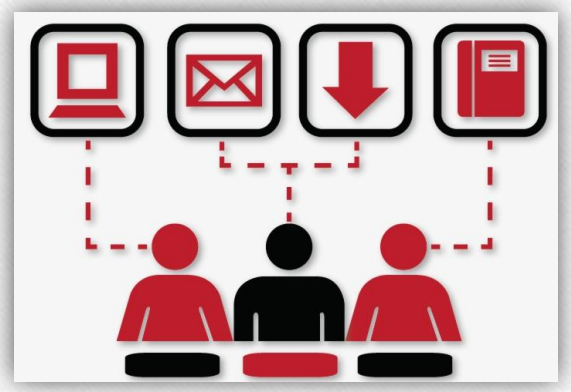
Data Models

It would be wildly inefficient to store all your data in one massive table. So, your data warehouse contains many tables that you can join together to get specific information. The main tables is called a **fact table**, and **dimension tables** surround it.

The first step in designing a data warehouse is to build a conceptual data model that defines the data you want and the high-level relationships between them.



Dimension table



1) A dimension table contains dimensions of a fact.
2) They are joined to fact table via a foreign key.
3) Dimension tables are de-normalized tables.
4) The Dimension Attributes are the various columns in a dimension table
5) No set limit set for given for number of dimensions
6) The dimension can also contain one or more hierarchical relationships
7) Dimensions offers descriptive characteristics of the facts with the help of their attributes

What is Fact Table?

1. Fact table contains measurements, metrics, and facts about a business process while the Dimension table is a companion to the fact table which contains descriptive attributes to be used as query constraining.
2. Fact table is located at the center of a star or snowflake schema, whereas the Dimension table is located at the edges of the star or snowflake schema.
3. Fact table is defined by their grain or its most atomic level whereas Dimension table should be wordy, descriptive, complete, and quality assured.
4. Fact table helps to store report labels whereas Dimension table contains detailed data.
5. Fact table does not contain a hierarchy whereas the Dimension table contains hierarchies.

Types of Dimensions and Fact tables

Types of Dimensions

1. Junk Dimensions
2. Conformed Dimensions
3. Degenerate Dimensions
4. Rapidly Changing Dimensions
5. Role Playing Dimensions
6. Inferred Dimensions
7. Shrunken Dimensions
8. Static Dimensions
9. Slowly Changing Dimensions

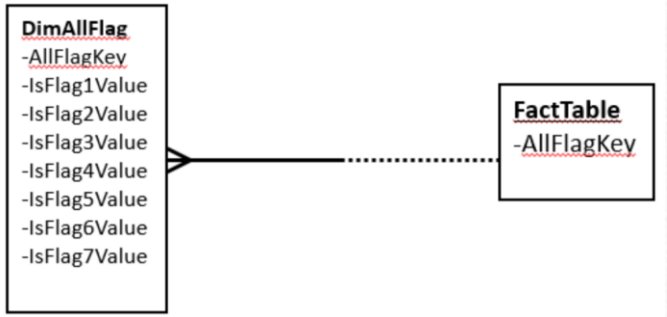
Types of Facts

1. Transaction Fact
2. Snapshot Fact
3. Accumulated Fact
4. Fact-less Fact

Junk Dimensions

A single table that includes a compilation of unrelated and different attributes is referred as the junk dimension table. This table is meant to avoid including a huge amount of foreign keys in the fact table. This table is often generated to handle foreign keys generated by the rapidly changing dimensions.

AllFlagKey	IsFlag1Value	IsFlag2Value	IsFlag3Value	IsFlag4Value	IsFlag5Value	IsFlag6Value	IsFlag7Value
1	Y	Y	Y	Y	Y	Y	Y
2	N	Y	Y	Y	Y	Y	Y
3	Y	N	Y	Y	Y	Y	Y
4	Y	Y	N	Y	Y	Y	Y

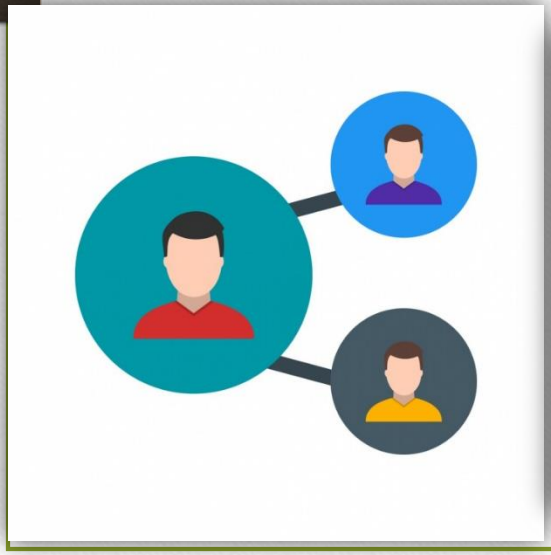


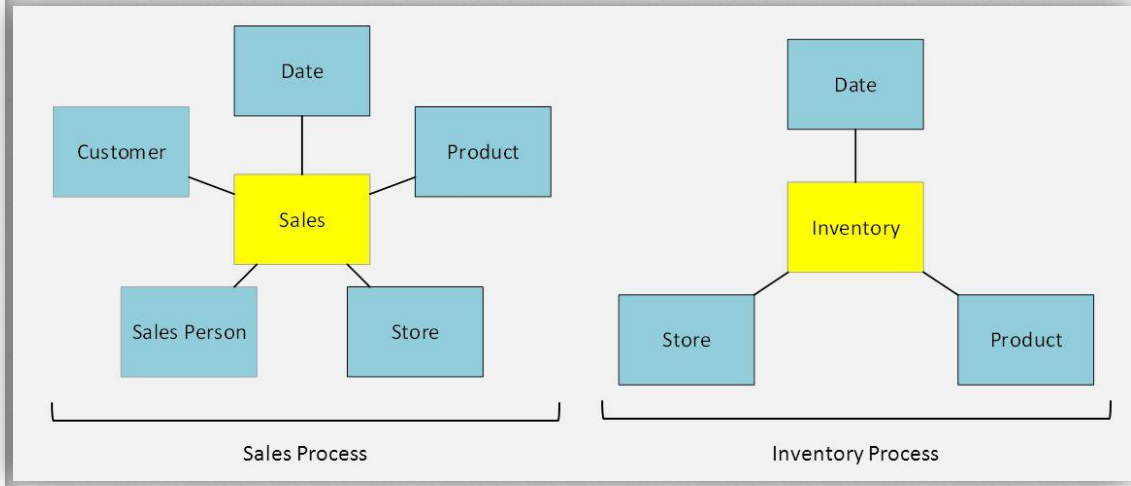
0 → 1

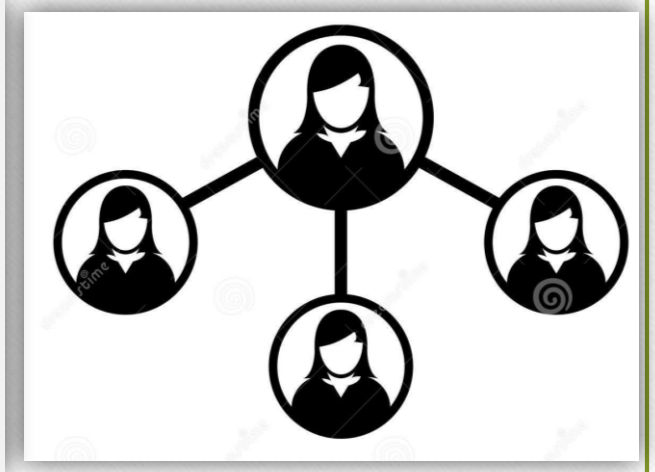
Conformed Dimensions

A conformed dimension connects to multiple fact tables

A conformed dimension allow a consistent analysis of different business process metrics used across multiple dimensional models. A product dimension may join to the inventory, sales, procurement, and other fact tables. It is important to closely collaborate with data governance team to have a consistent definition for this dimension.

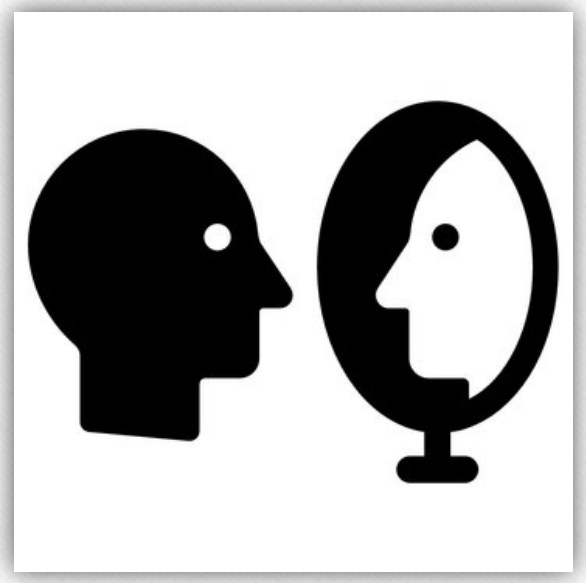


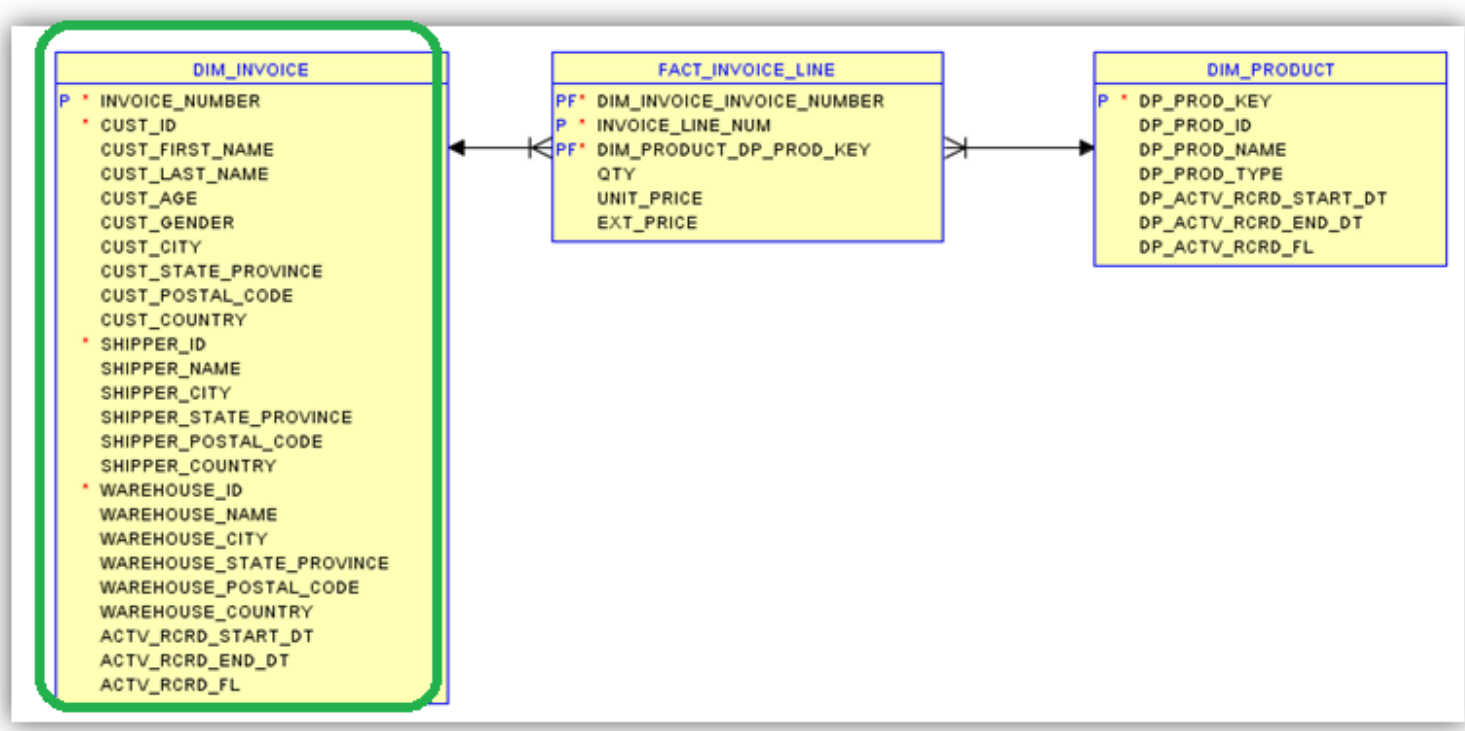




Degenerate Dimensions

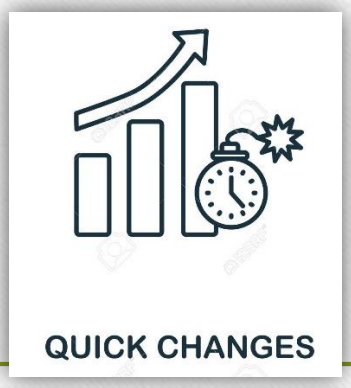
A degenerate dimension is a special dimension like invoice number, check number that is an identifier for a transaction. However, it does not have attributes linked to it as all the important attributes are already part of their respective dimensions. So an invoice may have a customer name attribute but it's already a part of the customer dimension. A degenerate dimension is a part of the fact table but it's not a measure, it is still a dimension.





Rapidly Changing Dimensions

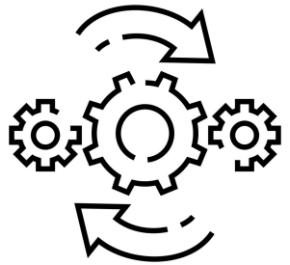
An attribute in the dimension, which is changing frequently, is referred as the rapidly changing attribute. Handling rapidly changing dimension in data warehouse is very difficult because of many performance implications. As we know **slowly changing dimension type 2** is used to maintain the history for the changes. But the problem with type 2 is, with each and every change in the dimension attribute, it adds new row to the table. If in case there are dimensions that are changing a lot, table become larger and may cause serious performance issues. Hence, use of the type 2 may not be the wise decision to implement the rapidly changing dimensions.



DIM_CUSTOMER	
P * CUST_KEY	NUMBER
CUST_NAME	VARCHAR2
CUST_CITY	VARCHAR2
CUST_STATE	VARCHAR2
CUST_AGE	NUMBER
CUST_INCOME	NUMBER
CUST_LIFETIME_PURCHASES	NUMBER
CUST_RATING	VARCHAR2
CUST_ACCOUNT_STATUS	VARCHAR2
CUST_CREDIT_SCORE	NUMBER
CUST_GENDER	VARCHAR2
CUST_ACTV_RCRD_FL	NUMBER
CUST_ACTV_RCRD_START_DT	DATE
CUST_ACTV_RCRD_END_DT	DATE
DIM_CUSTOMER_PK (CUST_KEY)	

DIM_CUSTOMER	
P * CUST_KEY	NUMBER
CUST_NAME	VARCHAR2
CUST_CITY	VARCHAR2
CUST_STATE	VARCHAR2
CUST_ATTRIBUTE_KEY	NUMBER
CUST_GENDER	VARCHAR2
CUST_ACTV_RCRD_FL	NUMBER
CUST_ACTV_RCRD_START_DT	DATE
CUST_ACTV_RCRD_END_DT	DATE
DIM_CUSTOMER_PK (CUST_KEY)	

DIM_CUSTOMER_ATTRIBUTE	
P * CUST_ATTR_KEY	NUMBER
CUST_ATTR_AGE	VARCHAR2
CUST_ATTR_INCOME	VARCHAR2
CUST_ATTR_LIFETIME_PURCHASES	VARCHAR2
CUST_ATTR_RATING	VARCHAR2
CUST_ATTR_ACCOUNT_STATUS	VARCHAR2
CUST_ATTR_CREDIT_SCORE	VARCHAR2
CUST_ATTR_ACTV_RCRD_FL	NUMBER
CUST_ATTR_ACTV_RCRD_START_DT	DATE
CUST_ATTR_ACTV_RCRD_END_DT	DATE
DIM_CUSTOMER_ATTRIBUTE_PK (CUST_ATTR_KEY)	



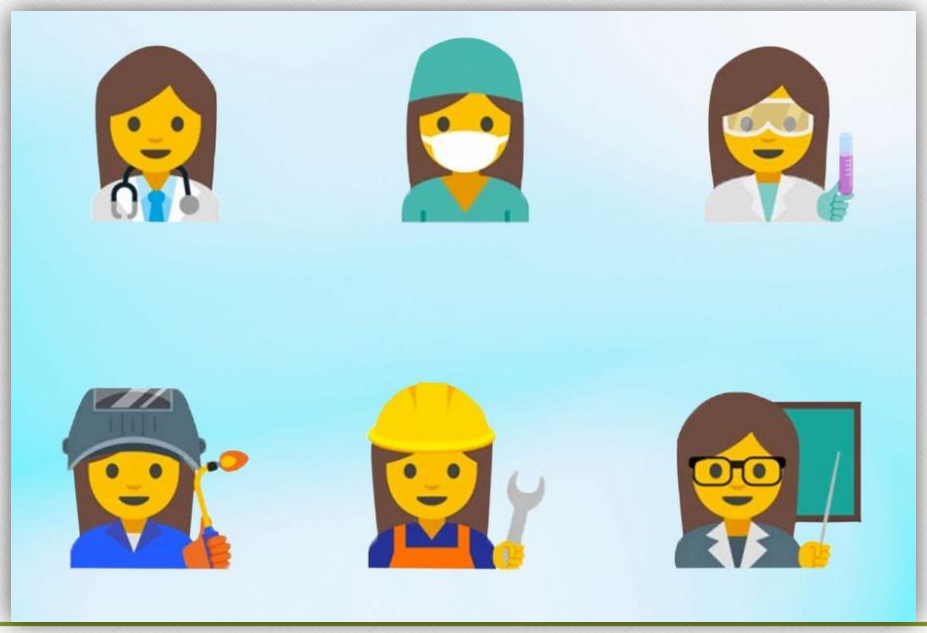
Role Playing Dimensions

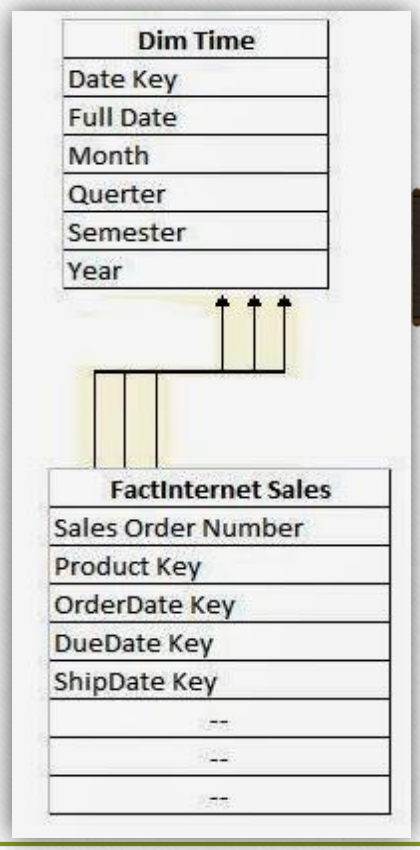
Role Playing Dimension

A database Dimension that acts as multiple dimensions with in a cube is called as Role Playing Dimension. From the same table if we have multiple foreign keys in fact tables then the table acts differently for each key attribute.

Example:

Time Dimension is one of the best Example of Role playing Dimension, you can have one Time Dimension called Date and then you can add **ShipDate**, **DueDate** and **OrderDate** as Cube Dimensions.





Inferred Dimensions

Inferred Dimension, also referred to as **Early coming Fact** or **Late Arrival dimension**.

Sometimes, dimension records might not be ready while loading the fact records. A solution to handle this situation is creating a surrogate key with a null value for the entire other attributes. This process is technically referred as the inferred member or inferred dimension.

Dim_Promo_Code

Promo_Code_PKEY	Promo_Code	Promotion Type	Discount Type	Inferred
1	P1	Marketing	Cashback	0
2	P2	Employee	Instant	0
3	PSpecialOrg	Unknown	Unknown	1

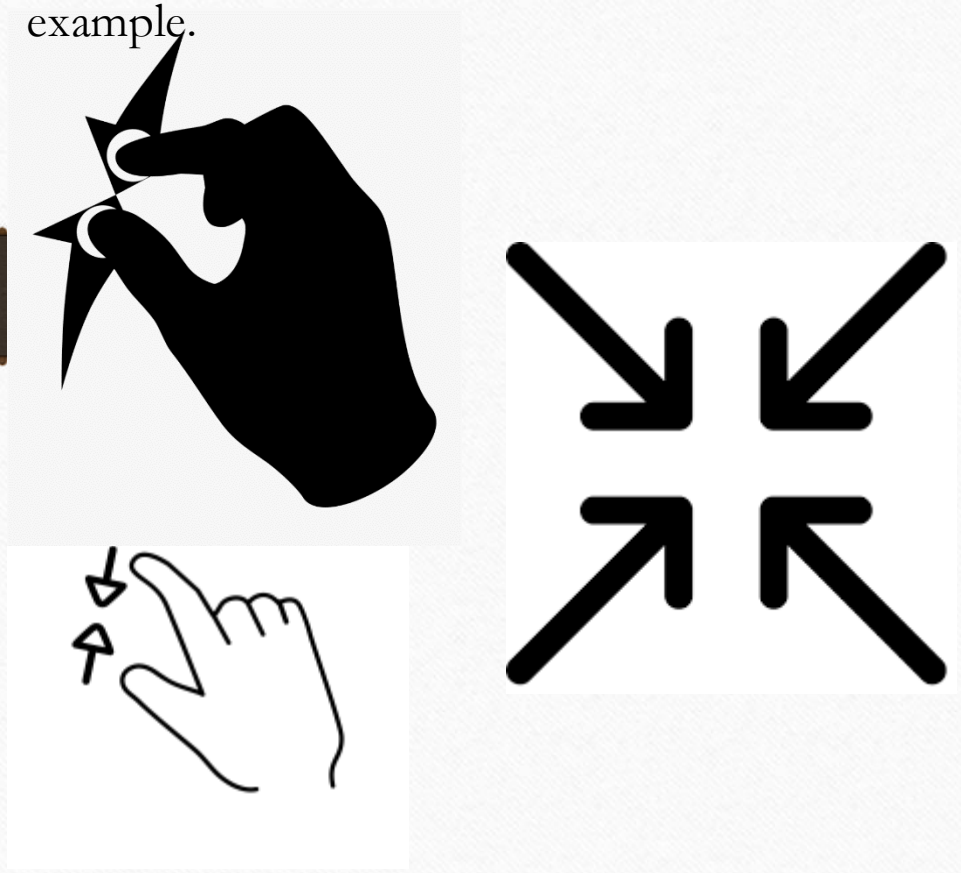
Fact _Tickets

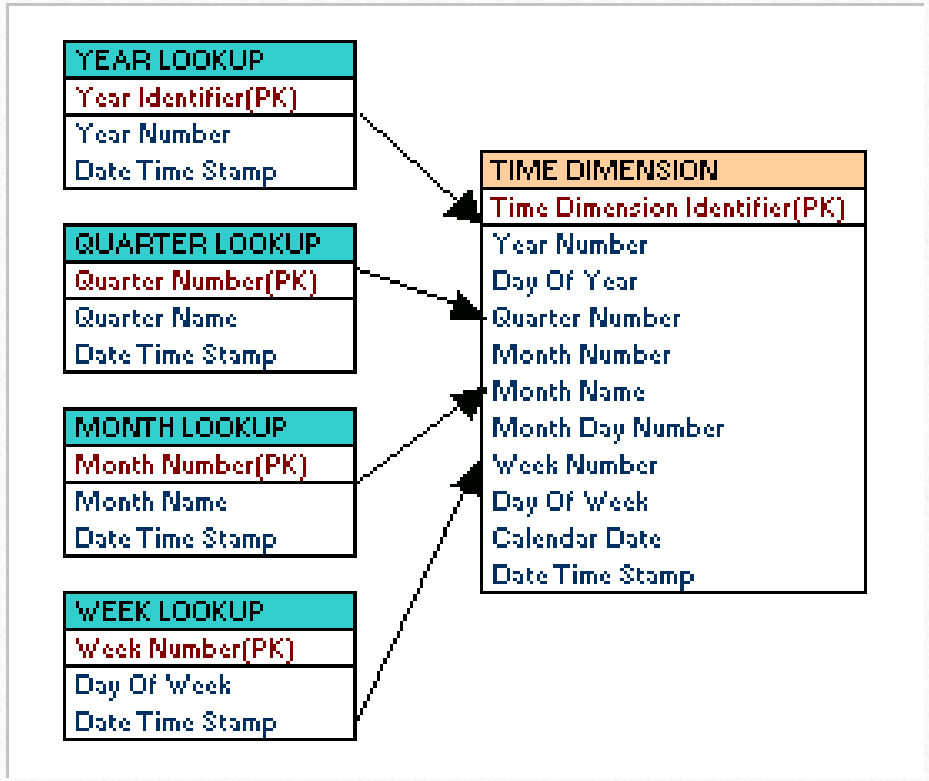
Ticket_ID	TransactionDate_FKey	Promo_Code	Promo_Code_FKEY
T1	20180819	P1	1
T2	20180819	P2	2
T3	20180819	PSpecialOrg	-1

Shrunken Dimensions

The shrunken dimension table is referred as the subset of alternative dimension.

Month could be a shrunken dimension of date(time or calendar) table, which could be connected to a fact which has grain of month like monthly sales. But this is a very classic example, let us deep dive a little with another example.





Static Dimensions

Static Dimensions : (no-source)

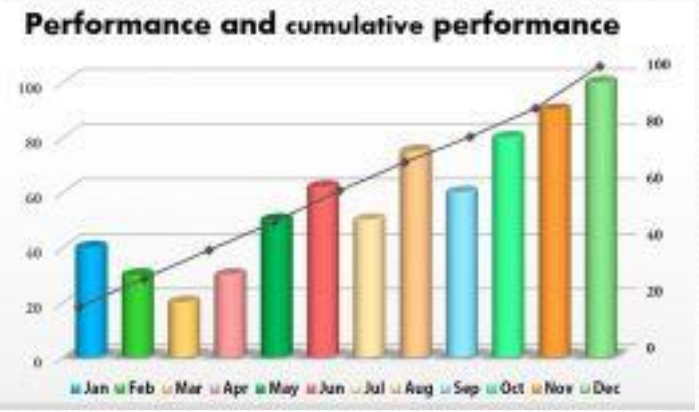
A static dimension is a dimension that is not loaded from the source system.
The static dimensions are generated using a SQL script or a stored procedure and are manually loaded. The time dimension is a classic example of a static dimension. Also, status codes dimension is mostly a static dimension.

A few common examples of the static dimensions would be
"DIM_SOURCE_SYSTEMS",

id	source_name	type	status
1	SAP	ERP	Y
2	Oracle	Database	Y
3	DB2	Database	Y
4	SalesForce	ERP	Y
5	systemlogs	file	Y

Types of Facts

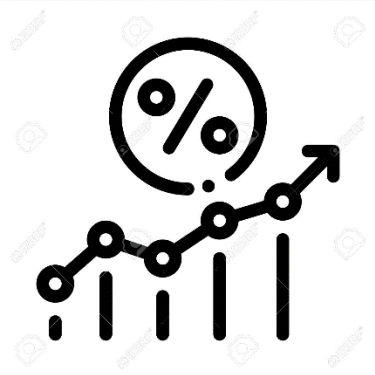
1. **Additive**: Additive facts can be used with any aggregation function like Sum(), Avg() etc. Example is Quantity, sales amount etc
2. **Semi-Additive**: Semi-additive facts are facts that can be summed up for some of the dimensions in the fact table, but not the others. For example, Consider bank account details. You cannot apply the Sum() on the bank balance that does not give useful results but min() and max() function may return useful information
3. **Non-Additive:** Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table. For example of non-additive fact is any kind of ratio or percentage. Non numeric facts can also be a non-additive facts

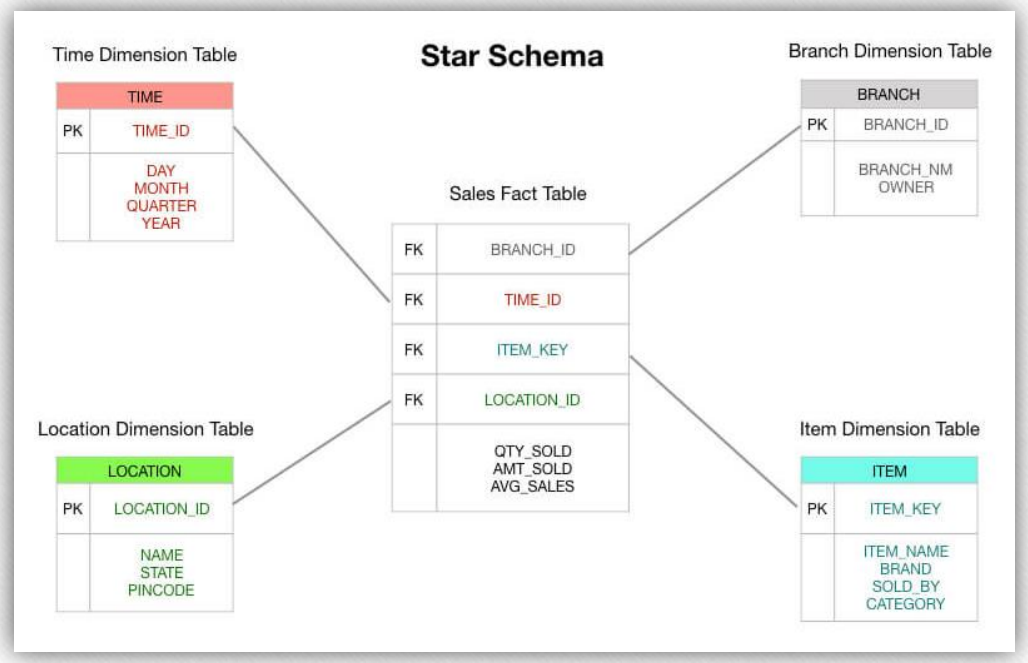


Transaction Fact

Transaction Fact

Transaction fact tables are easy to understand: a customer or business process does some thing; you want to capture the occurrence of that thing, and so you record a transaction in your data warehouse and you're good to go.

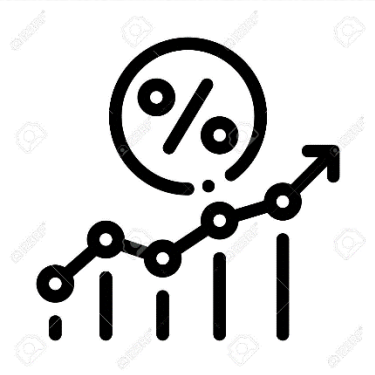




Cumulative Fact

Cumulative

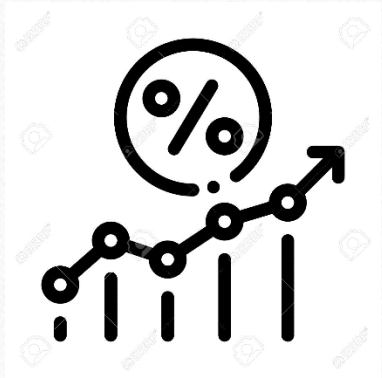
This type of fact table describes what has happened over a period of time. For example, this fact table may describe the total sales by product by store by day. The facts for this type of fact tables are mostly additive facts. The first example presented here is a cumulative fact table.



FACT_CLAIM_PROCESSING		
P *	CLAIM_KEY	NUMBER
P *	CUSTOMER_KEY	NUMBER
P *	POLICY_KEY	NUMBER
*	CLAIM_DATE	DATE
	INVESTIGATION_DATE	DATE
	DAYS_TO_INVESTIGATION	NUMBER
	REVIEW_DATE	DATE
	DAYS_TO_REVIEW	NUMBER
	DESCISION_DATE	DATE
	DAYS_TO_DESCISION	NUMBER
	PAYMENT_DATE	DATE
	DAYS_TO_PAYMENT	NUMBER

Snapshot Fact

Snapshot: (Periodic) Snapshot fact tables capture the state of the measures based on the occurrence of a status event or at a specified point-in-time or over specified time intervals (week, month, quarter, year, etc.).

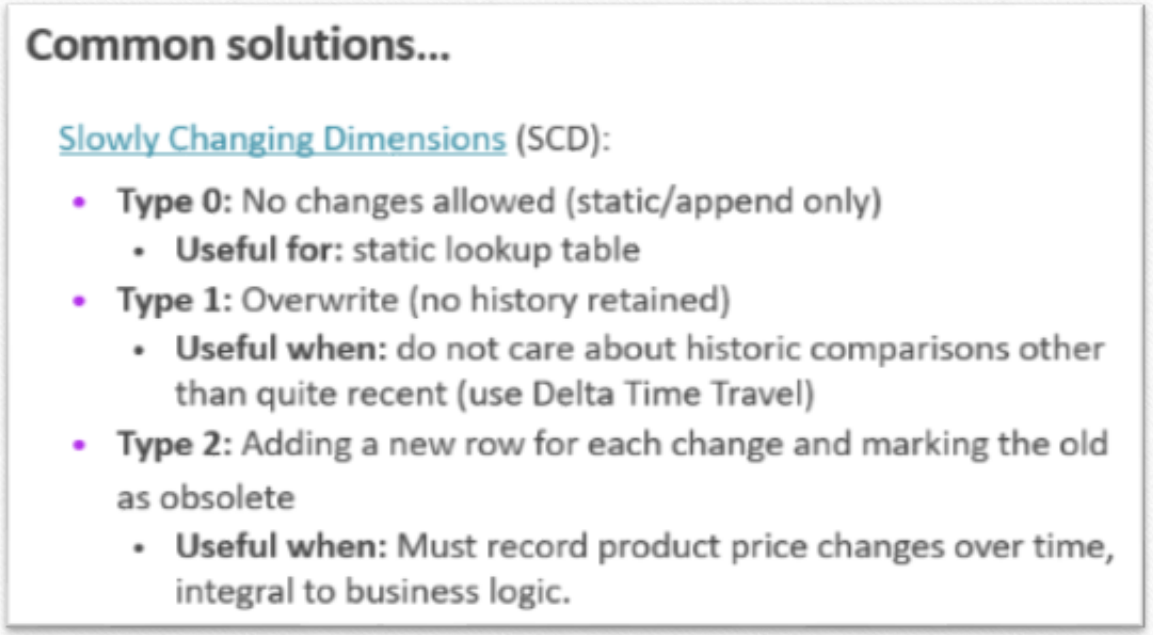


FACT_MONTHLY_SALES		
P *	SALE_MONTH_DATE_KEY	NUMBER
P *	SALE_CUSTOMER_KEY	NUMBER
P *	SALE_PRODUCT_KEY	NUMBER
P *	SALE_AUTHOR_GROUP_KEY	NUMBER
P *	SALE_STORE_KEY	NUMBER
	SALE_QTY	NUMBER
	SALE_AMT	NUMBER

Star Schemas and Dynamic Partition Pruning

The **star schema** is very effective for Analytics queries, as long as all the dimensions stay the same. But what happens when something changes in a dimension table? For example, here is a dimension table that represents our company's Suppliers. Suppose we are keeping 5 years of history in our warehouse, and at some point, say year 3, this Supplier moves its facilities to a new State? If we are building reports that report on Suppliers grouped by State, how do we keep our report accurate?

Remember, if you have a reporting over a 5-year history, so the problem is that we want results that are accurate over that whole time period.



There are several design choices available to solve the SCD problem. There are actually more choices than we're showing here, but these are the most common.
These solutions are known by Type x, where x is a number from 1 to 6 (although there is actually no Type 5).

Type 0 is easy (but not very effective). We simply refuse to allow changes. In Type 1 we simply overwrite the old information with new information. In Type 2, we keep a historical trail of the information. Above type 2, we simply implement more sophisticated ways of keeping history. Let's look deeper into Types 1 and 2…

SCD Type 1

Example of a supplier table:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

If the supplier relocates the headquarters to Illinois the record would be overwritten:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

Here is a Type 1 example. When the Supplier moves from CA to IL, we simply overwrite the information in the record. That means that the older historical parts of our report will now be incorrect, because it will appear that this Supplier was *always* in IL.

SCD Type 2

Supplier_Key	Supplier_Code	Supplier_Name	Supplier	Version
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

Another method is to add 'effective date' columns.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	2000-01-01T00:00:00	2004-12-22T00:00:00
124	ABC	Acme Supply Co	IL	2004-12-22T00:00:00	NULL

And a third method uses an effective date and a current flag.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Effective_Date	Current_Flag
123	ABC	Acme Supply Co	CA	2000-01-01T00:00:00	N
124	ABC	Acme Supply Co	IL	2004-12-22T00:00:00	Y

```sql
%sql
MERGE INTO delta.emp_scd as tgt
USING (
    -- These rows will either UPDATE the current addresses of existing employee or
    --INSERT the new addresses of new employee
  SELECT emp_csv.empid as emp_key,emp_csv.*
  FROM emp_csv
  UNION ALL
  -- These rows will INSERT new addresses of existing employee
  -- Setting the mergeKey to NULL forces these rows to NOT MATCH and be INSERTed.
  SELECT NULL as emp_key,emp_csv.*
  FROM emp_csv JOIN delta.emp_scd as delta
  ON emp_csv.empid = delta.empid
  WHERE delta.status = true AND emp_csv.loc <> delta.loc
) get_updates
ON tgt.empid = emp_key
WHEN MATCHED AND tgt.status = true AND tgt.loc <> get_updates.loc THEN
  UPDATE SET status = false, end_date = current_date()
-- Set current to false and endDate to source's effective date or current date if its on same date.
WHEN NOT MATCHED THEN
  INSERT(empid,name,loc,start_date,end_date,status)
  VALUES(get_updates.empid,get_updates.name, get_updates.loc, current_date(), '9999-12-12',True)
-- Set current to true along with the new address and its effective date.
```

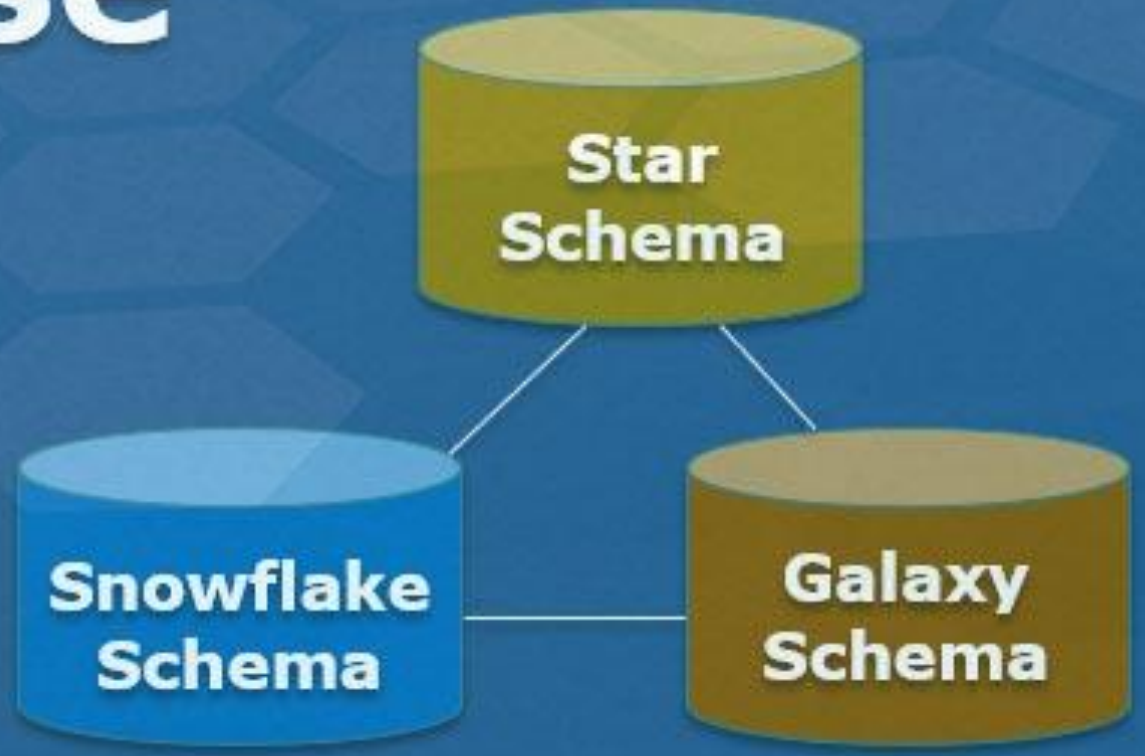
Here are three different ways we might implement a Type 2 solution. Our goal here is to make sure our historical reports are accurate throughout the entire time period.

In the top example, we add a new row to the dimension table whenever data changes, and we keep a version number to show the order of the changes. Why do you think this might be an ineffective solution? Is it important to know ***when*** in time each version changed?

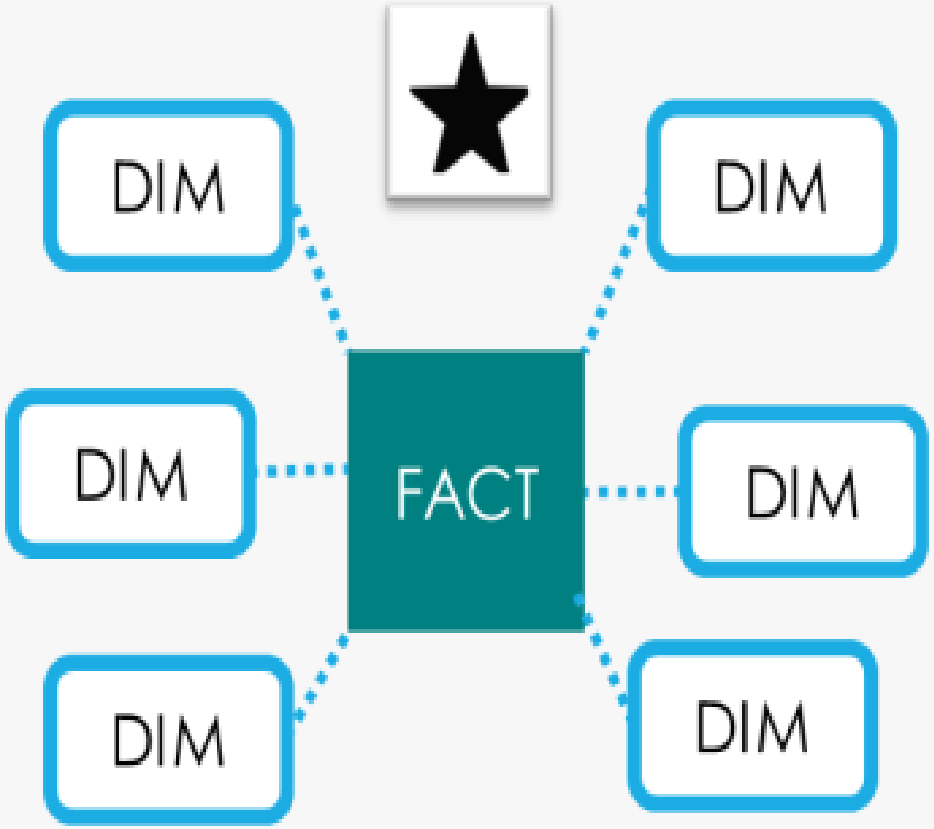
In the middle example, we see a more effective solution. Each row for the supplier has a start and end date. If we design our queries well, we can make sure that each time period in our report uses the corresponding Supplier row. If there is no data in the End_Date column, we know we have the current information.

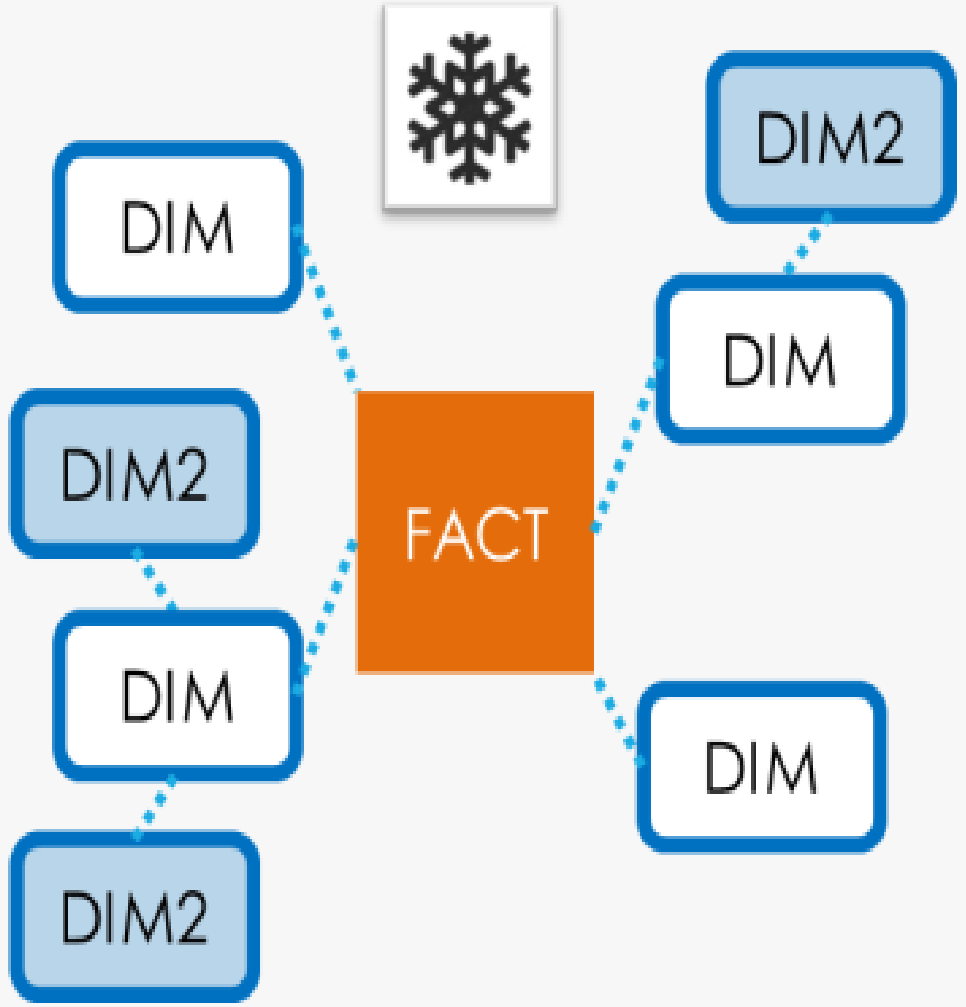
The bottom example is similar. Effective_Date is the same as Start_Date in the middle example. Instead of End_Date, we have a flag that says whether or not this row is current. Our queries may get a bit more complex here, because we must read a row to get the start date, then read the next row to determine the end date (assuming we are using non-current rows).

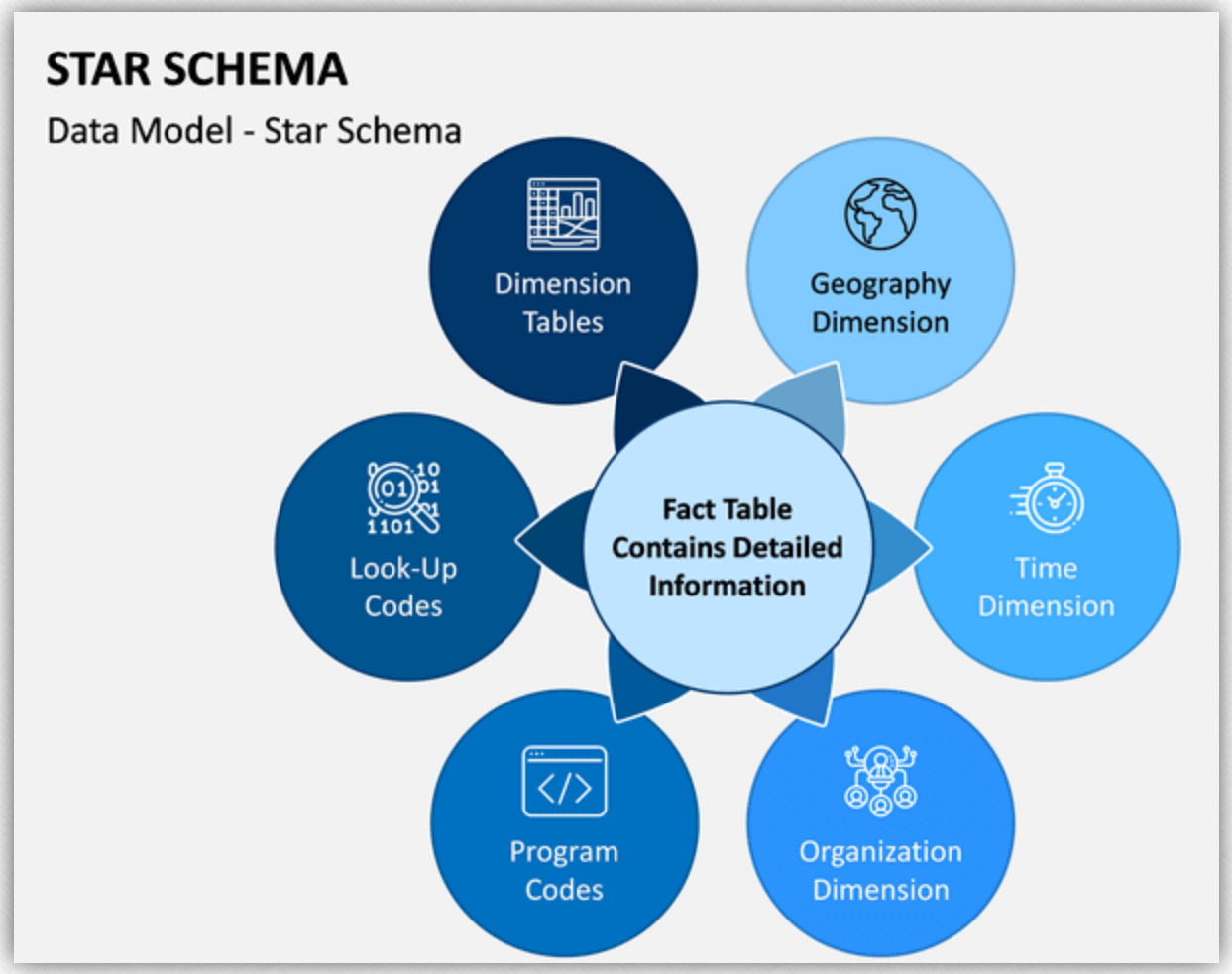
Data Warehouse Schema



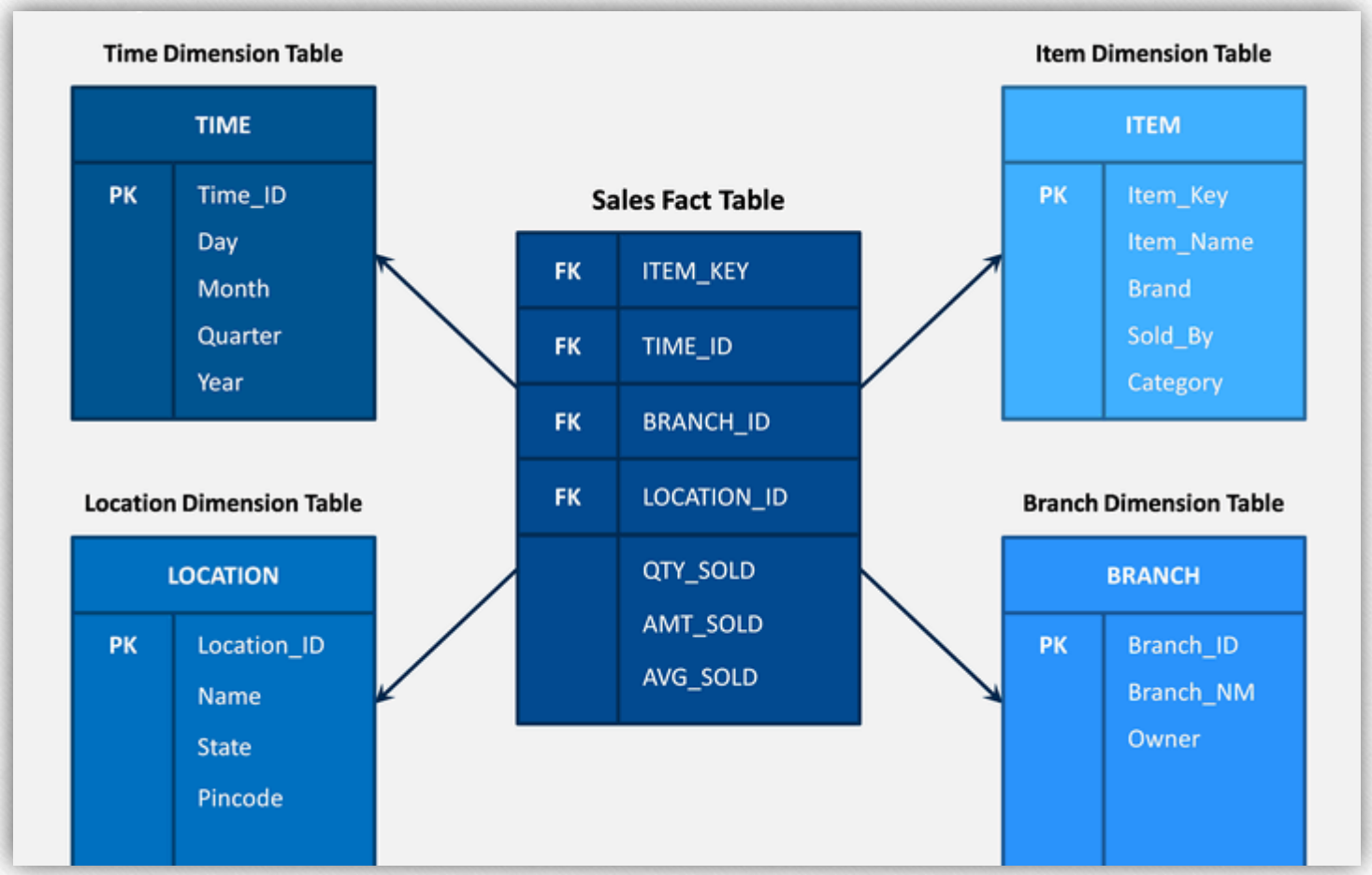
Star Schema And Snowflake Schema



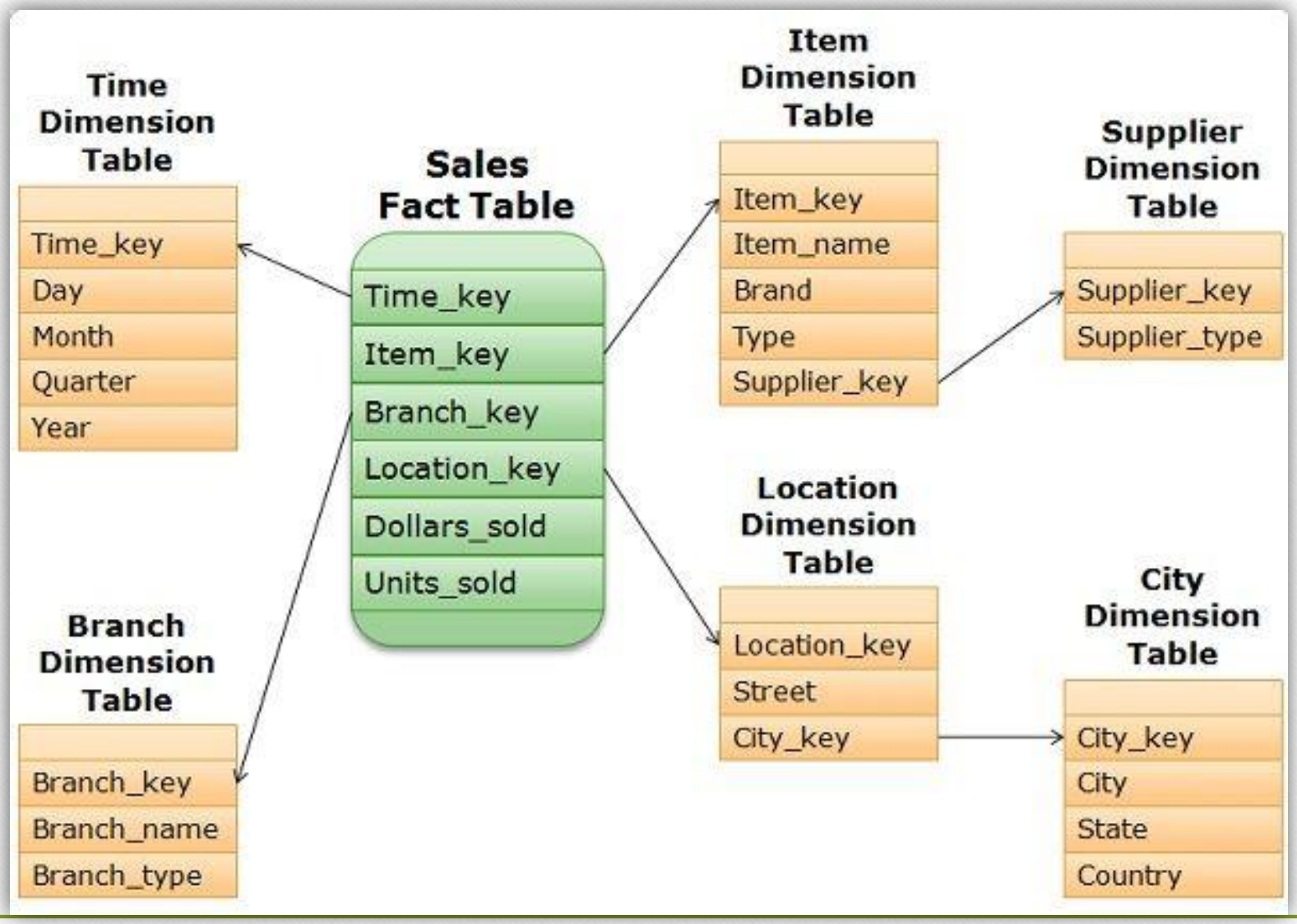



STAR SCHEMA
Data Model - Star Schema
Dimension Tables
Geography Dimension
Look-Up Codes
Fact Table Contains Detailed Information
Time Dimension
Program Codes
Organization Dimension

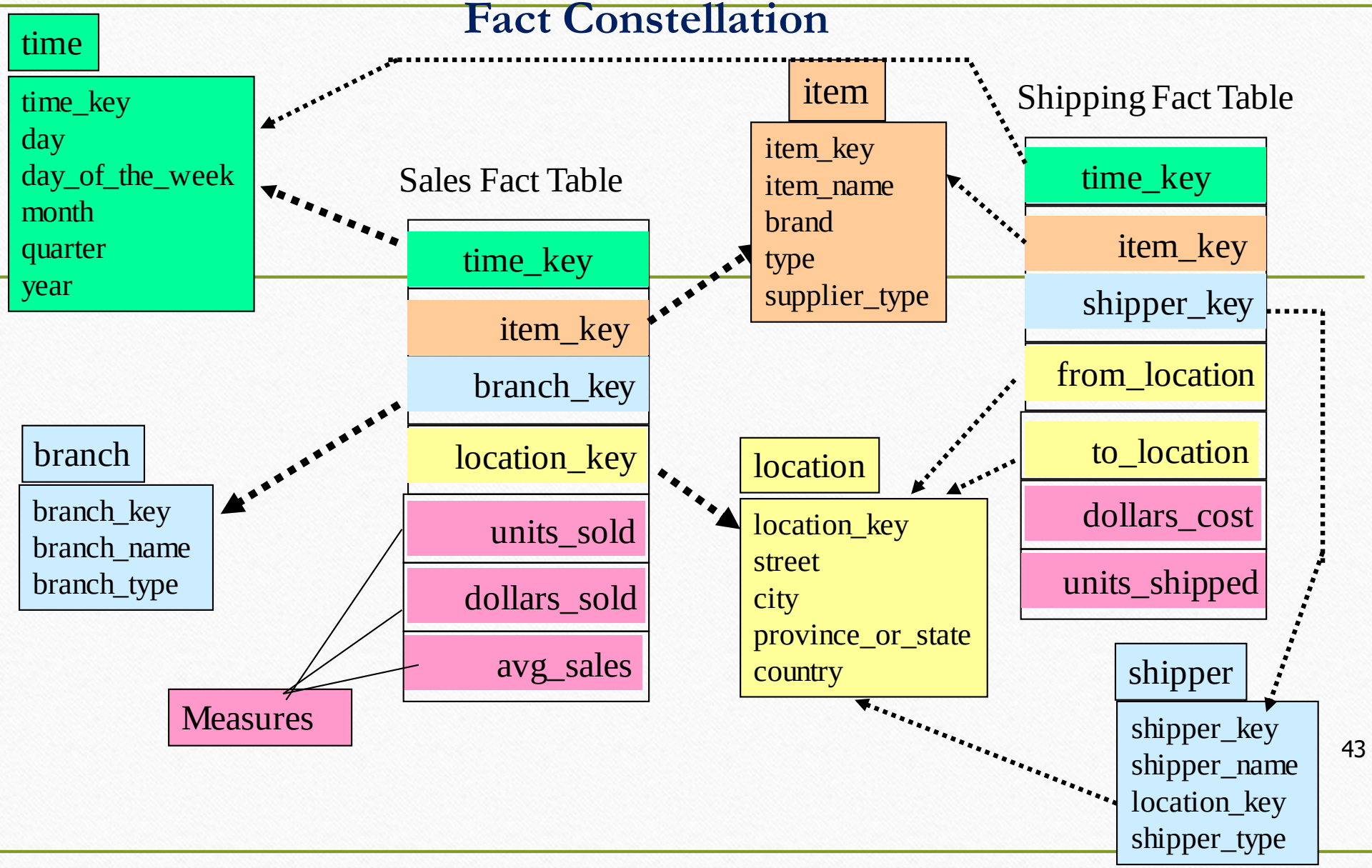
Star Schema


Time Dimension Table
TIME
PK
Time_ID
Day
Month
Quarter
Year
Item Dimension Table
ITEM
PK
Item_Key
Item_Name
Brand
Sold_By
Category
Sales Fact Table
FK
ITEM_KEY
FK
TIME_ID
FK
BRANCH_ID
FK
LOCATION_ID
QTY_SOLD
AMT_SOLD
AVG_SOLD
Location Dimension Table
LOCATION
PK
Location_ID
Name
State
Pincode
Branch Dimension Table
BRANCH
PK
Branch_ID
Branch_NM
Owner

Snowflake Schema

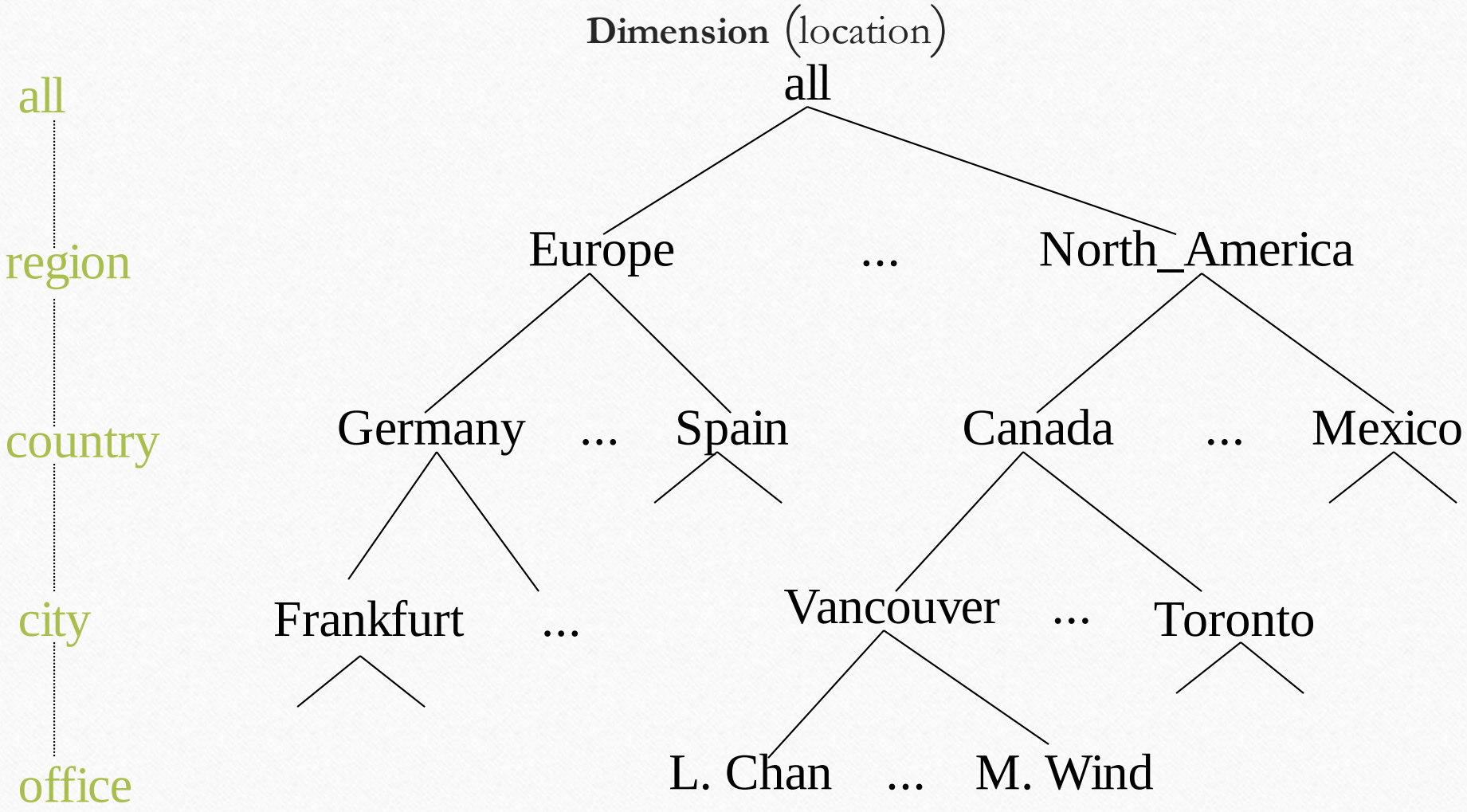


Fact Constellation

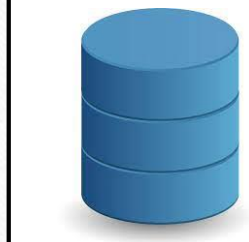


OLTP	OLAP
♦ Current data	♦ Current data as well as history.
♦ Used to support transaction processing	♦ Used to support the business interests
♦ Clerical data processing tasks	♦ Decision support tasks
♦ Simple and known queries	♦ Ad hoc, complex, and iterative queries which access million records and perform a lot of joins and aggregates
♦ A few tables involved and unlikely to be scanned	♦ Multiple tables involved and likely to be scanned
♦ Small foundset	♦ Large foundset
♦ Short transactions	♦ Long transactions
♦ Update/Select	♦ Select (Read only)
♦ Real time update	♦ Batch update
♦ Unique index	♦ Multiple index
♦ Known access path	♦ Do not know access path until users start asking queries
♦ Detail row retrieval	♦ Aggregation and group by
♦ High selectivity queries	♦ Low selectivity queries
♦ Low I/O and processing	♦ High I/O and processing
♦ Response time does not depend on database size	♦ Response time depends on database size
♦ Data model: entity relational	♦ Data model: multidimensional

A Concept Hierarchy



Top-down Approach



Databases

Files

ERP's

Extract

Transform

Staging

Load

Data Warehouse

DataMart-Sales

DataMart- Inventory

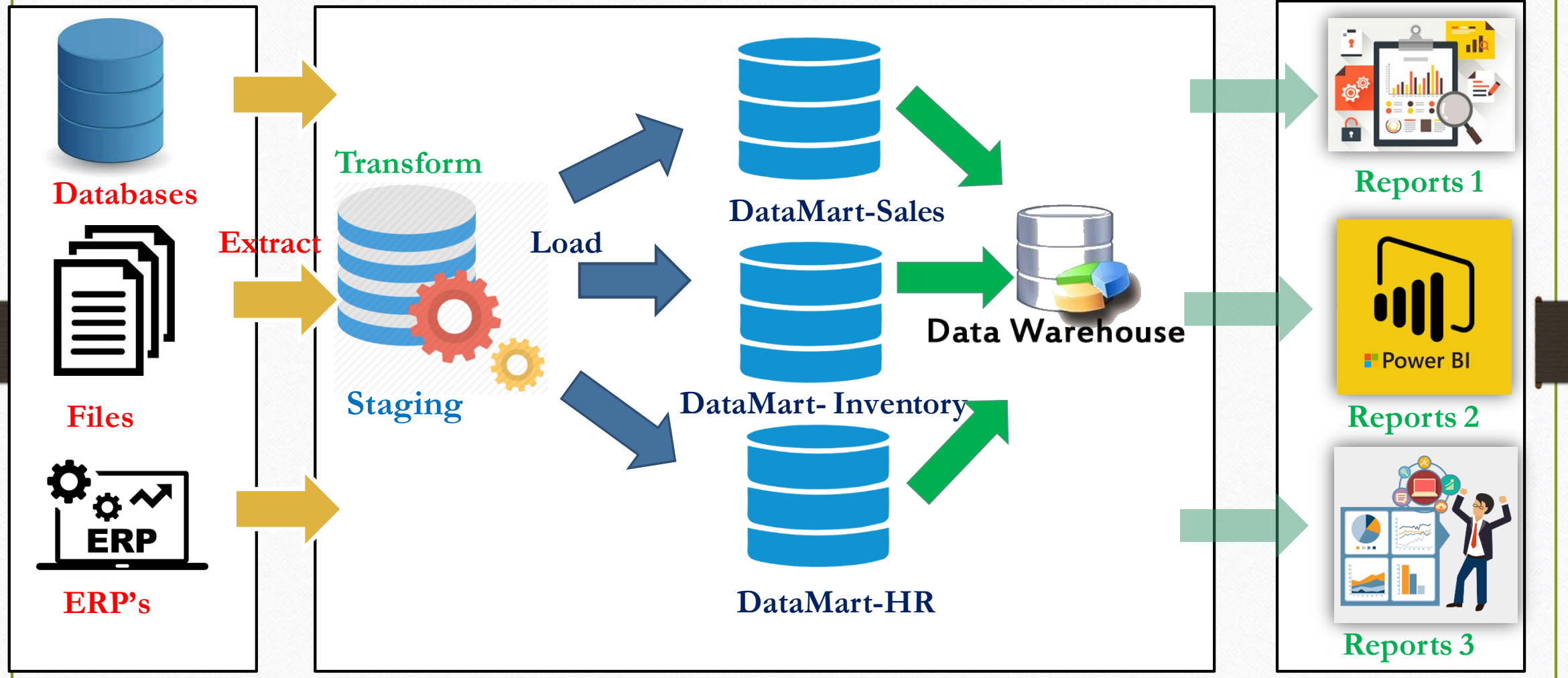
DataMart-HR

Reports 1

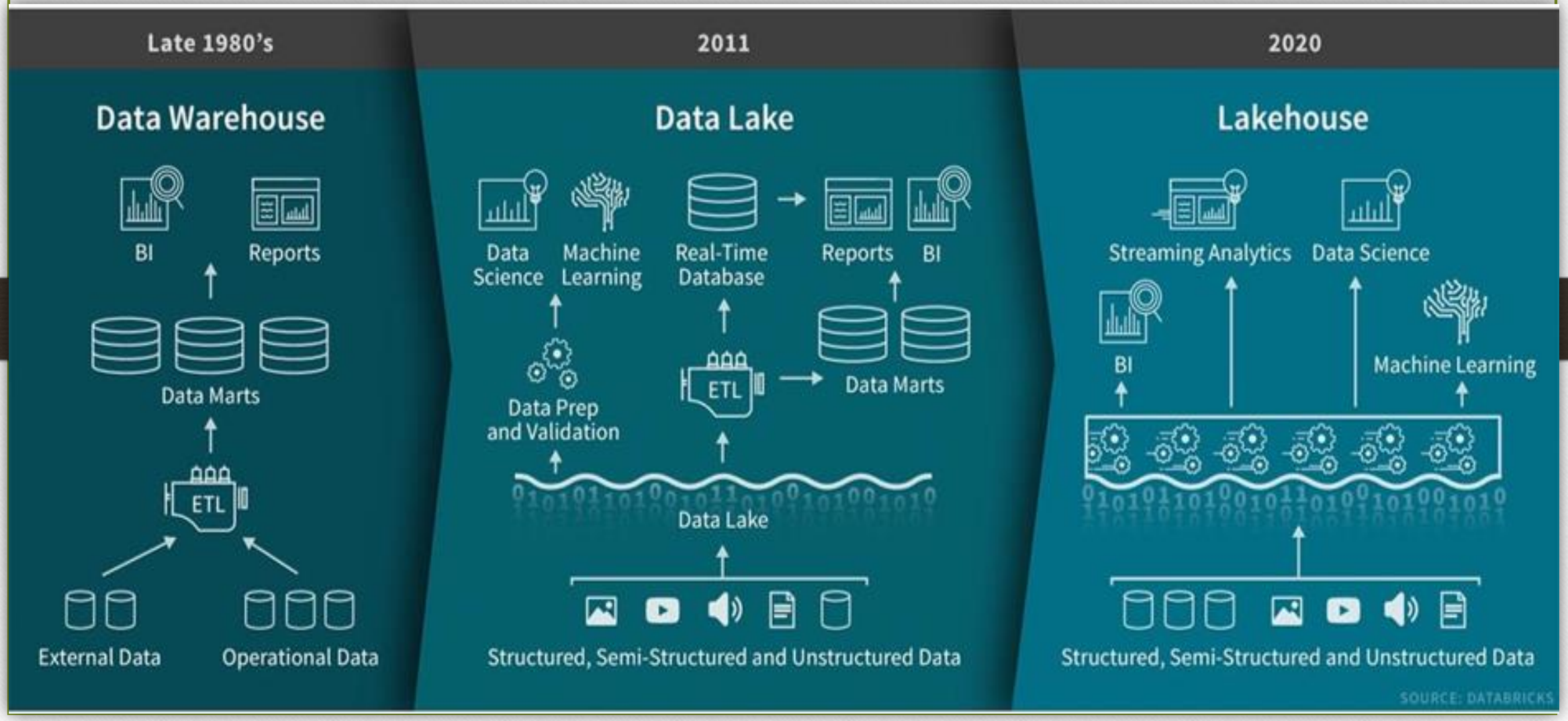
Reports 2

Reports 3

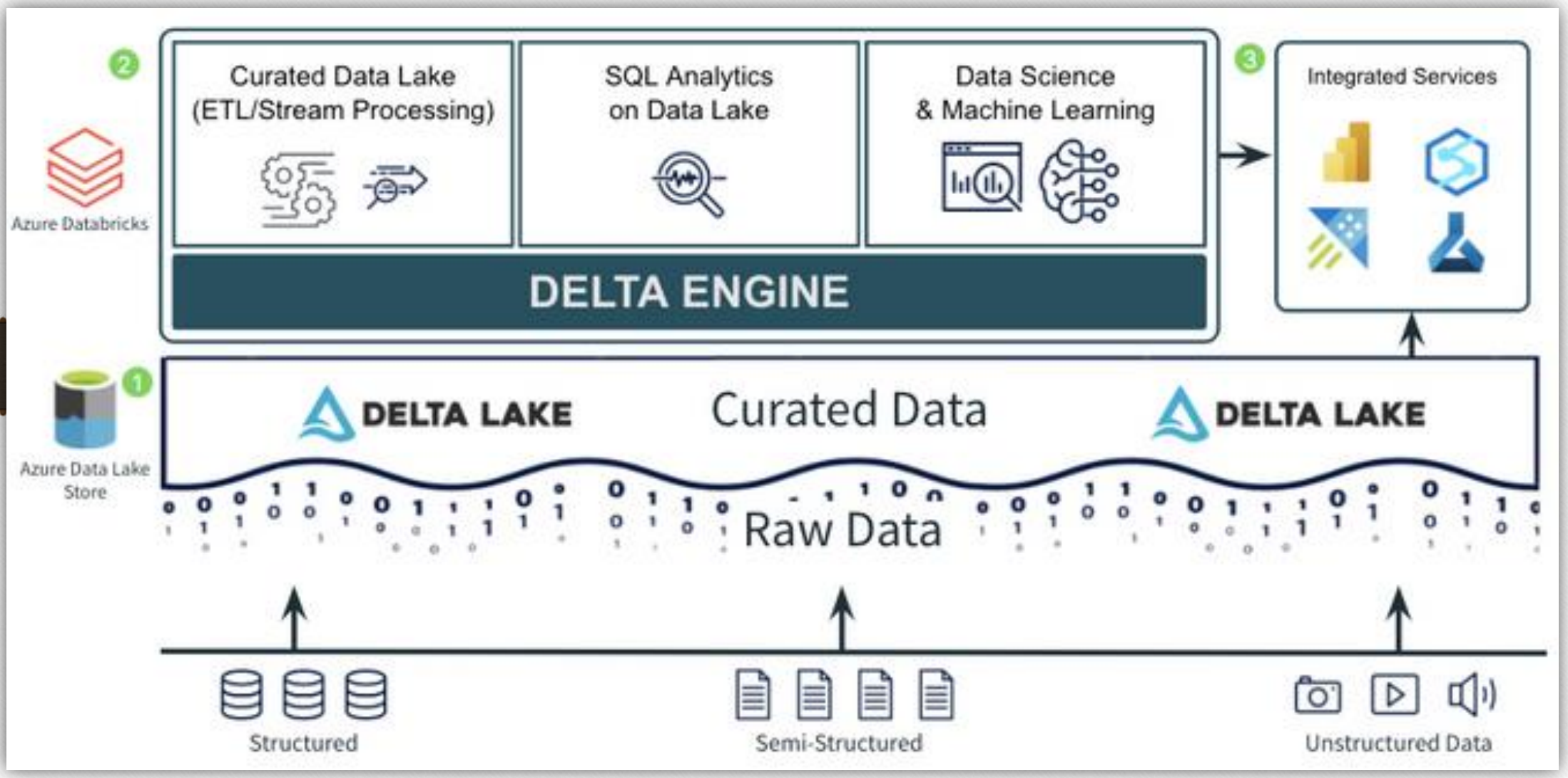
Bottom-up approach



Future Lake House Architecture

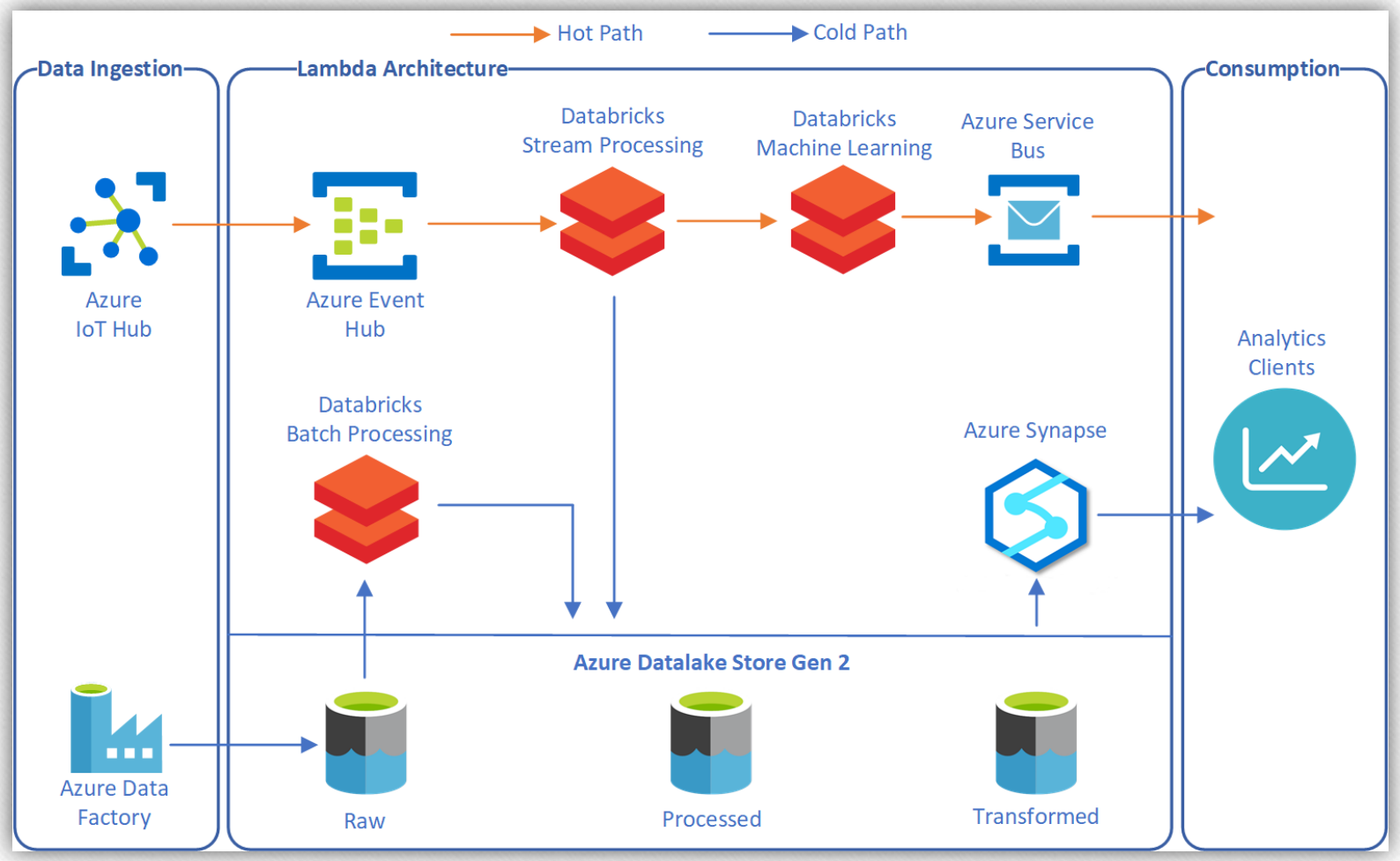


Azure Cloud Data Lake Architecture



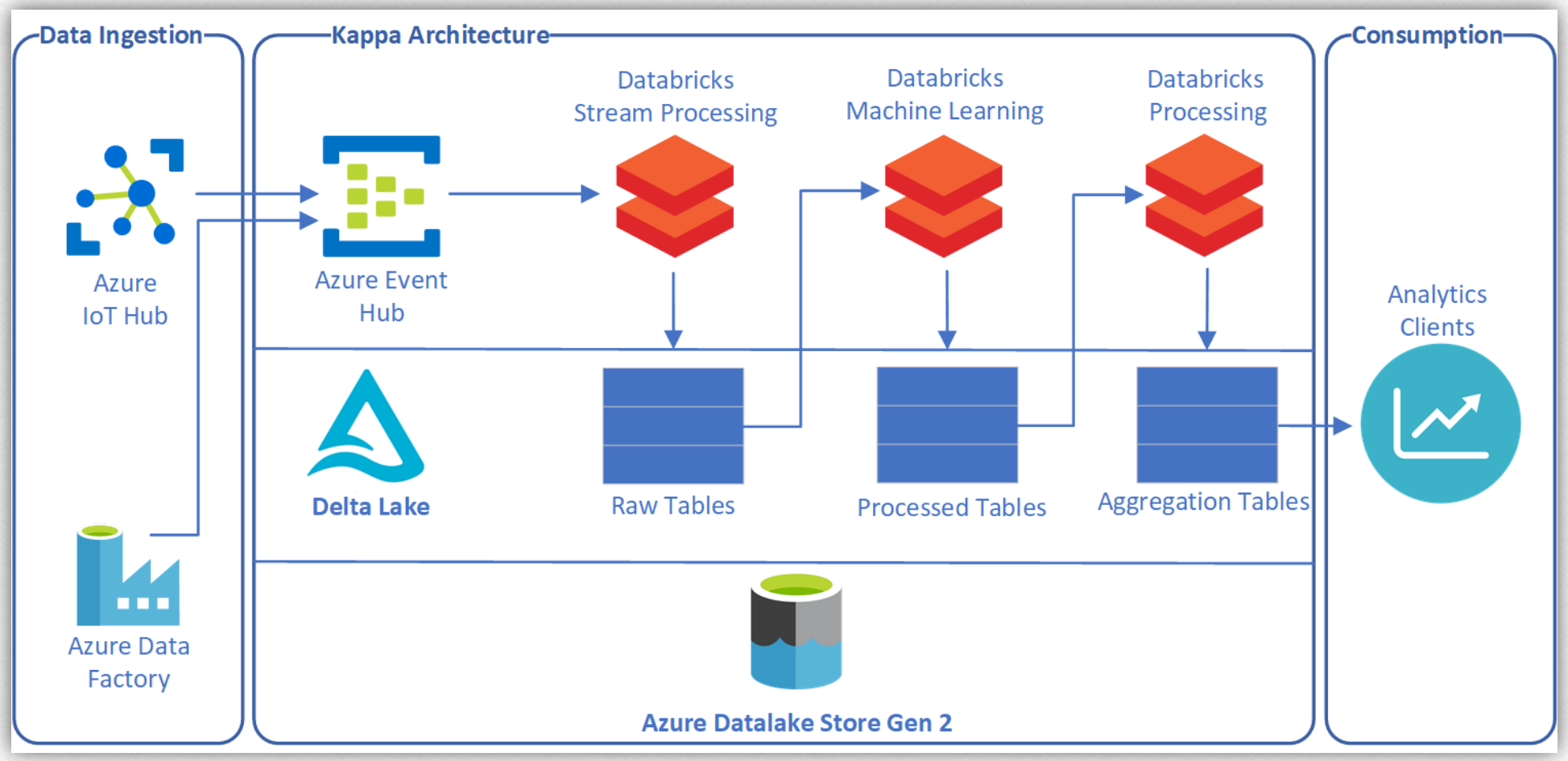
Lambda Architecture with Azure Databricks

In proposed Lambda Architecture implementation, the Databricks is a main component as shown in the below diagram.



Kappa Architecture with Databricks

The Kappa Architecture suggests to remove the cold path from the Lambda Architecture and allow processing in near real-time.



Differences between Data Lake , Cloud Warehouses and On-Premises Databases

	Data Lake	Cloud Data Warehouse	On-Premises Databases
Unstructured data (schema-less data)	Yes	No	No
Semi-Structured data (Self-describing schema)	Yes	Yes	No
Structured Data (Relational)	Yes	Better	Better
Independently scale storage and compute	Yes	Yes	No
Schema-on-Read	Yes	Yes (semi-structured)	No
Schema-on-Write	No	Yes	Yes

All The Best ☺