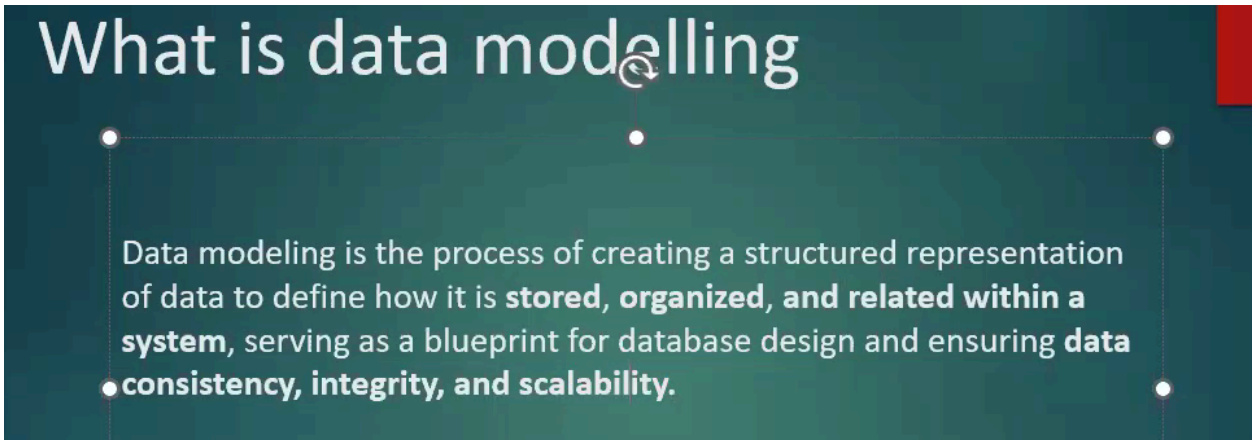
 # Data Modelling

Problems in storing data

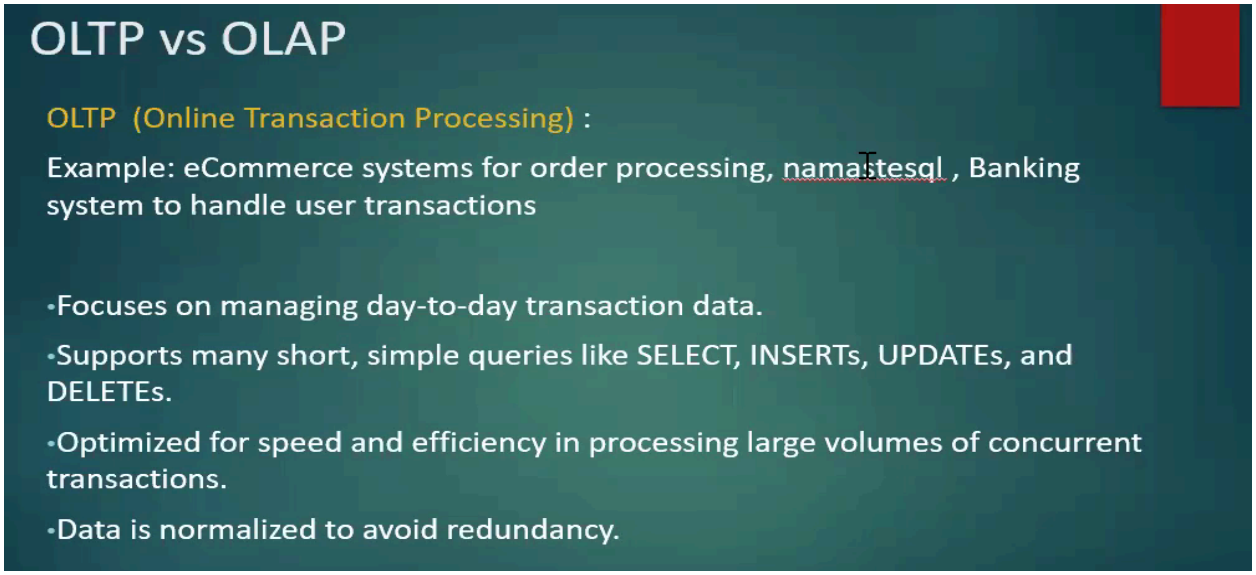
1) Data Redundancy - In fact table we store the Product name again and again
2) Data Consistency - some times product name change, city or address name change
3) Add new product - product id, product name, null, nul, null, null
4) Add new customer
5) Inserts will slower because have to insert same data again and again

Data modelling is a way to store data in organized way in a database so that
data is consistent,
there is no redundancy,
Inserts faster



It also depends on which kind of system you want to design

1) OLTP 2) OLAP



OLTP vs OLAP

OLAP (Online Analytical Processing) :

Example: Datawarehouse for business intelligence and reporting

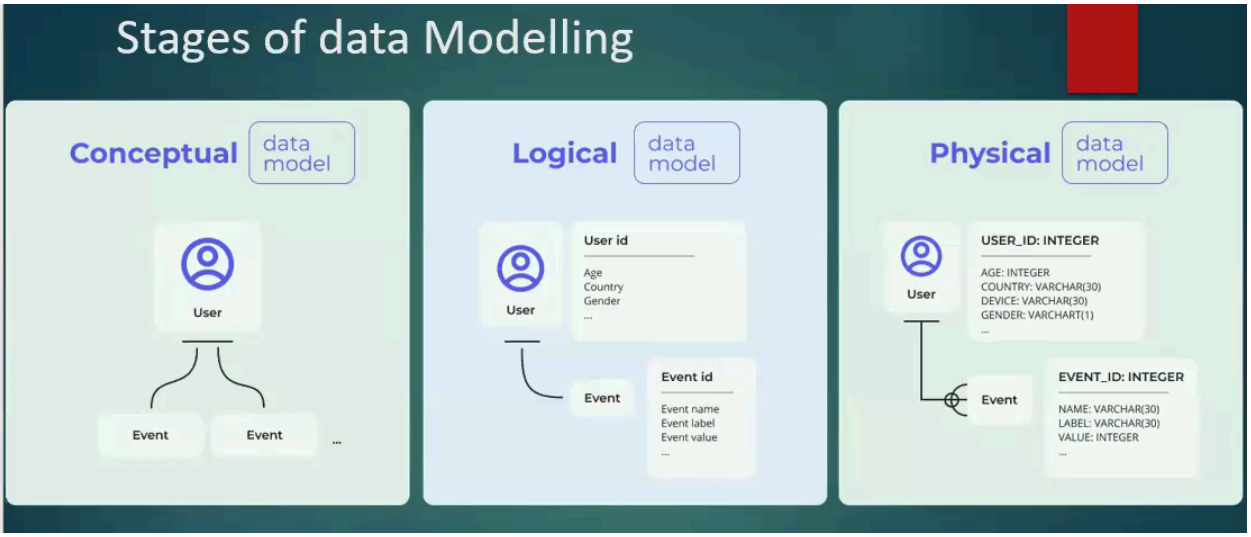
- Designed For Complex Analytical Queries And Reporting for historical data.
- Supports Fewer, Longer, And Complex Read-only Queries Like Aggregations And Trends.
- Optimized For Query Performance, Often Using Denormalized Data For Faster Retrieval.

KEY DIFFERENCES:

OLTP: REAL-TIME, TRANSACTIONAL, NORMALIZED, RELATIONAL DATA MODELLING

OLAP: HISTORICAL, ANALYTICAL, DENORMALIZED FOR FAST QUERYING, DIMENSIONAL MODELLING

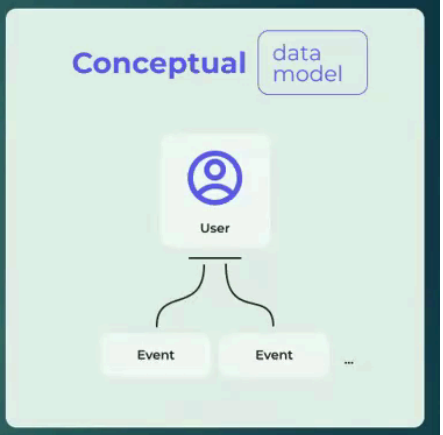
Entity - each table like customer, Location, Order



Conceptual data model

Conceptual data models are the most abstract form of a data model. They can be used to visualize the business or analytics operation that an application will support. Due to its simplicity, it is used by many stakeholders (especially business executives) to communicate ideas.

Conceptual data models help business executives to see how the application would work and ensure that it meets the business needs without going into the details, such as data types or technologies.

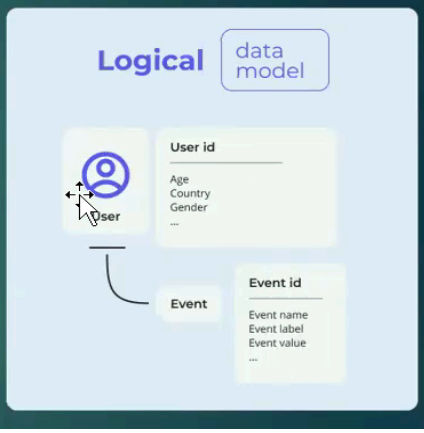


Logical data model

Logical data models take things a step further by adding more information. It works like this: logical data models show different datasets and describe their entities - including how they relate to each other.

As they contains information such as data structures, keys, data types, and other characteristics, technical teams use logical data models to translate business requirements into application and database design.

Similar to the conceptual model, they are not connected to any technology.

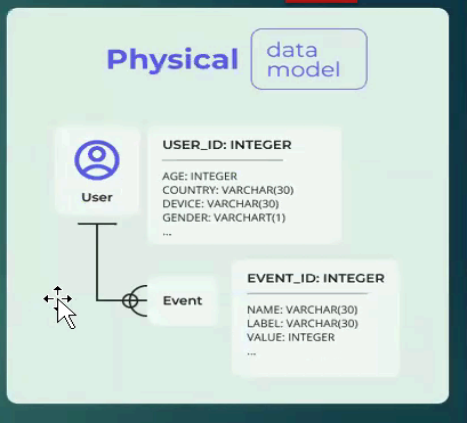


Physical data model

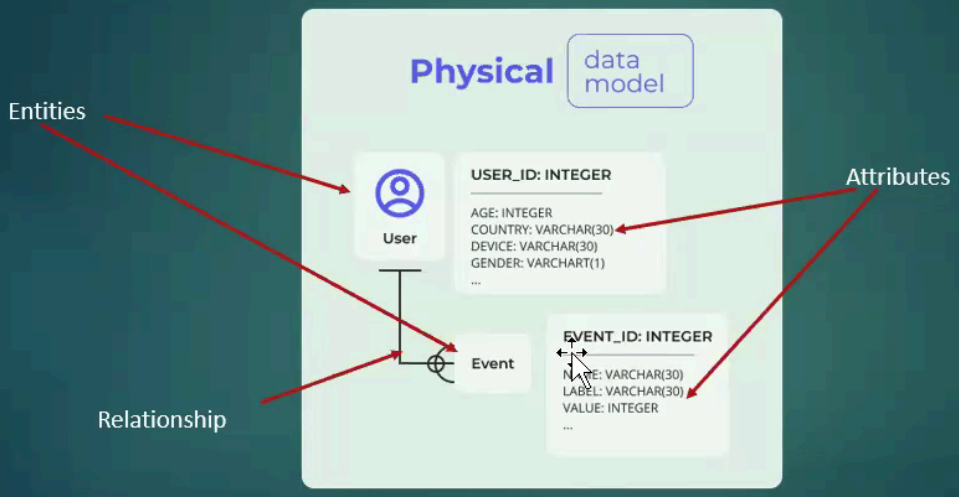
A logical data model is a building block for creating a physical data model that contains database-specific information about the data object, such as tables, columns, primary keys, and foreign keys.

These models are specific to implementing a database system or application software. Database designers use this to generate scripts to create the database.

The physical model identifies the data types, the primary keys, and connections.



Entities, Attributes and Relationship



What is Normalization

Database Normalization Is An Important Process Used To Organize And Structure Relational Databases. This Process Ensures That Data Is Stored In A Way That

1. **Minimizes Redundancy**: Normalization Improves Redundancy By Reducing The Repetition Of The Same Data Across Multiple Rows Or Tables. It Ensures That Each Piece Of Data Is Stored Only Once, In A Single Location, Which Eliminates Unnecessary Duplication

2. **Improves Data Integrity and consistency**: In A Non-normalized Database, The Same Piece Of Information Might Be Stored In Multiple Places. This Can Lead To Inconsistencies, Where One Version Of The Data Might Get Updated But The Others Don't. With Normalization, Data Is Stored In One Place, Reducing The Risk Of Inconsistency

3 Normal Forms

1st Normal Form (1NF): Each Column Contains Atomic Values, And There Are No Repeating Groups Or Arrays.

2nd Normal Form (2nf): The Table Is In 1nf, And All Non-key Attributes Are Fully Dependent On The Primary Key (No Partial Dependencies).

3rd Normal Form (3nf): The Table Is In 2nf, And All Non-key Attributes Are Directly Dependent On The Primary Key (No Transitive Dependencies).

1NF (First Normal Form):

Definition: A table is in 1NF if it meets the following criteria:

Each column contains atomic (indivisible) values and each column has unique values of a single type.

order_id	product_name	order_date	customer_name	customer_address	product_price	quantity
100	apple,samsumg	01-01-2024	Ankit	Mysore	15,25	1,2
200	apple	01-01-2024	Rahul	Bangalore	15	1

problem — The products column contains multiple values, which violates 1NF.

Solution (1NF) — Split the multi-valued column into separate rows for each product:

order_id	product_name	order_date	customer_name	customer_address	product_price	quantity
100	apple	01-01-2024	Ankit	Mysore	15	1
100	samsung	01-01-2024	Ankit	Mysore	25	2
200	apple	01-01-2024	Rahul	Bangalore	15	1

Definition: A table is in 2NF if:

It is in 1NF.

All non-key attributes (columns) are fully dependent on the **entire** primary key, not just a part of it (no partial dependency).

price depends on product name
here show some partial dependency

order_id	product_name	order_date	customer_name	customer_address	product_price	quantity
100	apple	01-01-2024	Ankit	Mysore	15	1
100	samsung	01-01-2024	Ankit	Mysore	25	2
200	apple	01-01-2024	Rahul	Bangalore	20	1

problem — the primary key in the above table is a composite key (order_id, product_name)

some columns like customer_name, customer_address, and order_date only depend on order_id

Solution (2NF) split the table into two: Orders and Order Items.

Step 1

orders

order_id	order_date	customer_name	customer_address
100	01-01-2024	Ankit	Mysore
200	01-01-2024	Rahul	Bangalore

order_items

order_id	product_name	product_price	quantity
100	apple	15	1
100	samsung	25	2
200	apple	15	1

step2

order_id	order_date	customer_name	customer_address
100	01-01-2024	Ankit	Mysore
200	01-01-2024	Rahul	Bangalore

order_items

order_id	product_id (fk)	quantity
100	1	1
100	2	2
200	1	1

products

product_id (k)	product_name	product_price
1	apple	15
2	samsumg	25

3NF (Third Normal Form):

Definition: A table is in 3NF if:

It is in 2NF.

There are no transitive dependencies (i.e., non-key attributes should not depend on other non-key attributes).

orders

order_id	order_date	customer_name	customer_address
100	01-01-2024	Ankit	Mysore
200	01-01-2024	Rahul	Bangalore

order_items

order_id	product_id (fk)	quantity
100	1	1
100	2	2
200	1	1

products

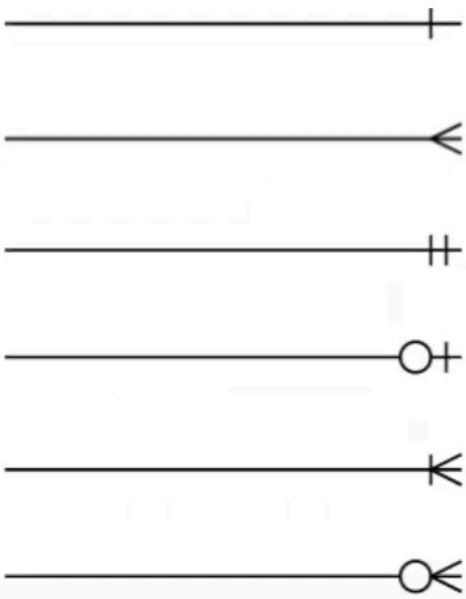
product_id (pk)	product_name	product_price
1	apple	15
2	samsumg	25

problem	customer address is dependent on customer
solution	create a seprate table for customer

customers

customerid	customer name	address
1	Ankit	Mysore
2	Rahul	Bangalore

Cardinality and ordinality are shown by the styling of a line and its endpoint, according to the chosen notation style.



One

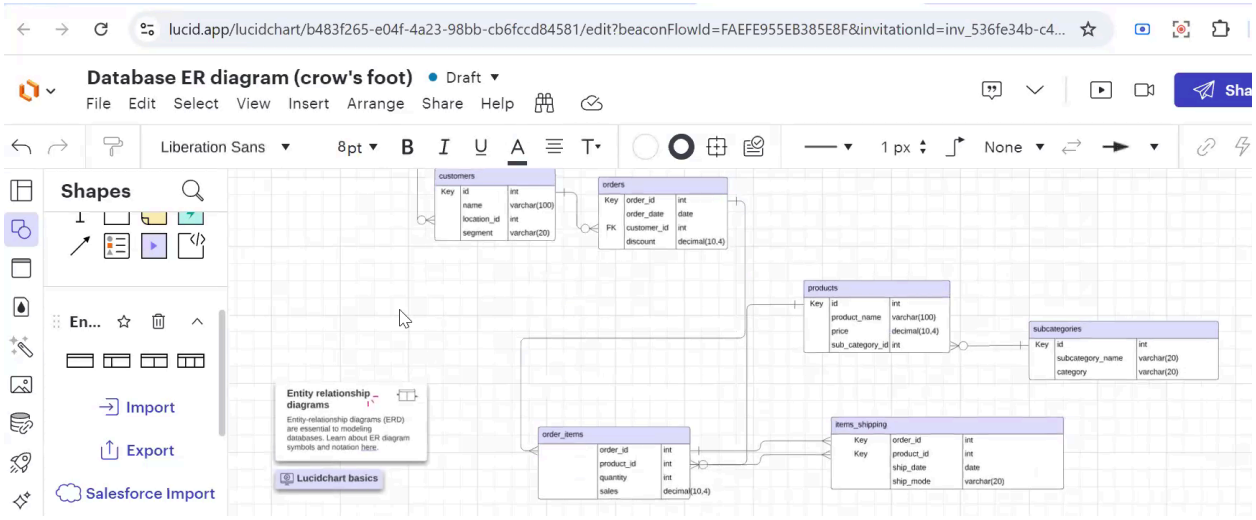
Many

One (and only one)

Zero or one

One or many

Zero or many



City table - ERD diagram

```
with cte as (
select  city,state,region,country
, ROW_NUMBER() over(partition by city order by order_date desc) as rn
from orders
)
select  ROW_NUMBER() over(order by city ) as id ,city,state,region,country
into cities
from cte where rn=1
```

125 %

Results | Messages

	id	city	state	region	country
1	1	Aberdeen	South Dakota	Central	United States
2	2	Abilene	Texas	Central	United States
3	3	Akron	Ohio	East	United States
4	4	Albuquerque	New Mexico	West	United States
5	5	Alexandria	Virginia	South	United States
6	6	Allen	Texas	Central	United States
7	7	Allentown	Pennsylvania	East	United States
8	8	Altoona	Pennsylvania	East	United States
9	9	Amarillo	Texas	Central	United States
10	10	Anaheim	California	West	United States

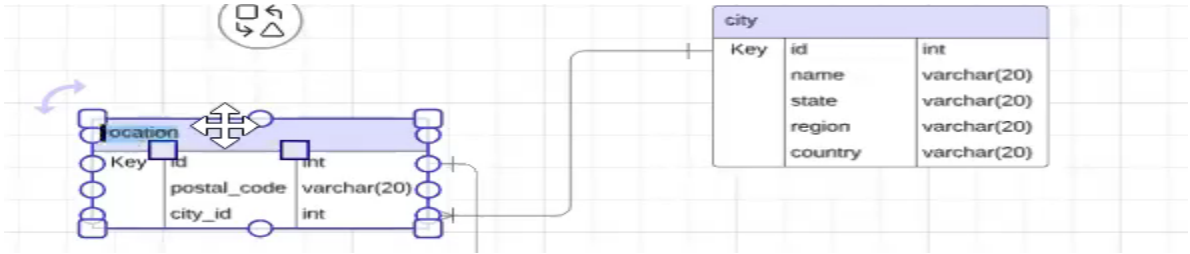
Location Table

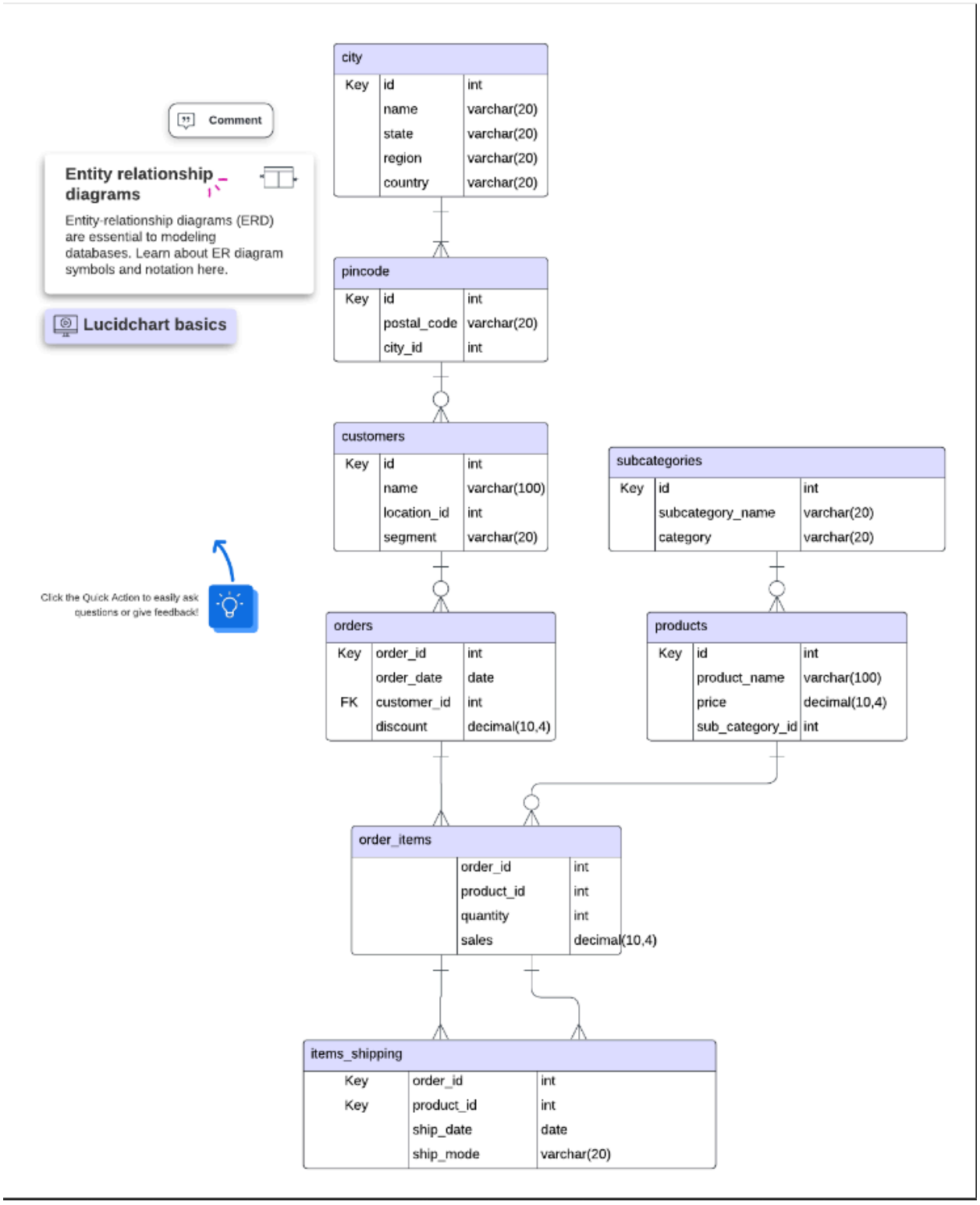
```
with cte as (
select postal_code,cities1.id as city_id
, ROW_NUMBER() over(partition by postal_code order by postal_code) as rn
from orders
inner join cities1 on orders.city=cities1.city
where postal_code is not null
)
select ROW_NUMBER() over(order by postal_code) as id,postal_code,city_id from cte
where rn=1
```

125 %

Results | Messages

	id	postal_code	city_id
1	1	1040	204
2	2	1453	254
3	3	1752	283
4	4	1810	11
5	5	1841	248
6	6	1852	270
7	7	1915	40
8	8	2038	157




Comment
Entity relationship diagrams
Entity-relationship diagrams (ERD) are essential to modeling databases. Learn about ER diagram symbols and notation here.
Lucidchart basics
Click the Quick Action to easily ask questions or give feedback!
city
Key id int
name varchar(20)
state varchar(20)
region varchar(20)
country varchar(20)
pincode
Key id int
postal_code varchar(20)
city_id int
customers
Key id int
name varchar(100)
location_id int
segment varchar(20)
subcategories
Key id int
subcategory_name varchar(20)
category varchar(20)
orders
Key order_id int
order_date date
FK customer_id int
discount decimal(10,4)
products
Key id int
product_name varchar(100)
price decimal(10,4)
sub_category_id int
order_items
order_id int
product_id int
quantity int
sales decimal(10,4)
items_shipping
Key order_id int
Key product_id int
ship_date date
ship_mode varchar(20)

- SQL + SELECT = Querying Data
- SQL + JOIN = Data Integration
- SQL + WHERE = Data Filtering
- SQL + GROUP BY = Data Aggregation
- SQL + ORDER BY = Data Sorting
- SQL + UNION = Combining Queries
- SQL + INSERT = Data Insertion
- SQL + UPDATE = Data Modification
- SQL + DELETE = Data Removal
- SQL + CREATE TABLE = Database Design
- SQL + ALTER TABLE = Schema Modification
- SQL + DROP TABLE = Table Removal
- SQL + INDEX = Query Optimization
- SQL + VIEW = Virtual Tables
- SQL + Subqueries = Nested Queries
- SQL + Stored Procedures = Task Automation
- SQL + Triggers = Automated Responses
- SQL + CTE = Recursive Queries
- SQL + Window Functions = Advanced Analytics
- SQL + Transactions = Data Integrity
- SQL + ACID Compliance = Reliable Operations
- SQL + Data Warehousing = Large Data Management
- SQL + ETL = Data Transformation
- SQL + Partitioning = Big Data Management
- SQL + Replication = High Availability
- SQL + Sharding = Database Scaling
- SQL + JSON = Semi-Structured Data
- SQL + XML = Structured Data
- SQL + Data Security = Data Protection
- SQL + Performance Tuning = Query Efficiency
- SQL + Data Governance = Data Quality

What is a Data Warehouse?

A data warehouse is A centralized repository that stores large volumes of structured data from various sources. It is designed specifically for query and analysis rather than transaction processing. Data is extracted from operational systems (like OLTP databases), transformed to ensure consistency, and loaded into the warehouse in A format optimized for reporting and analysis.

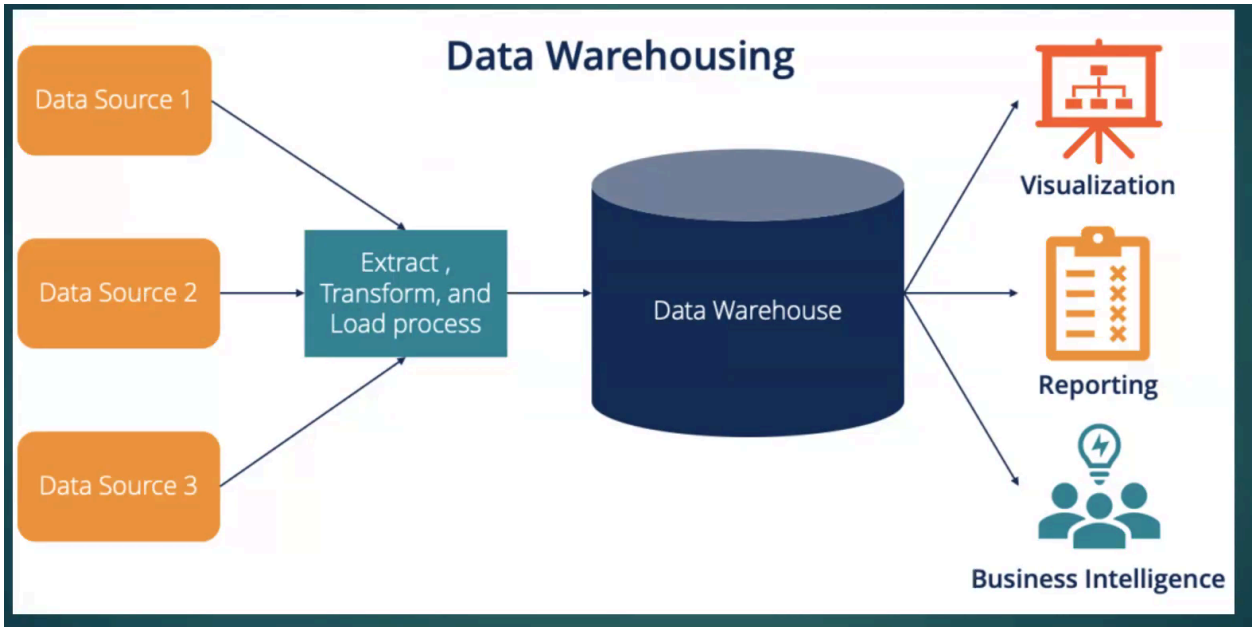
Key features:

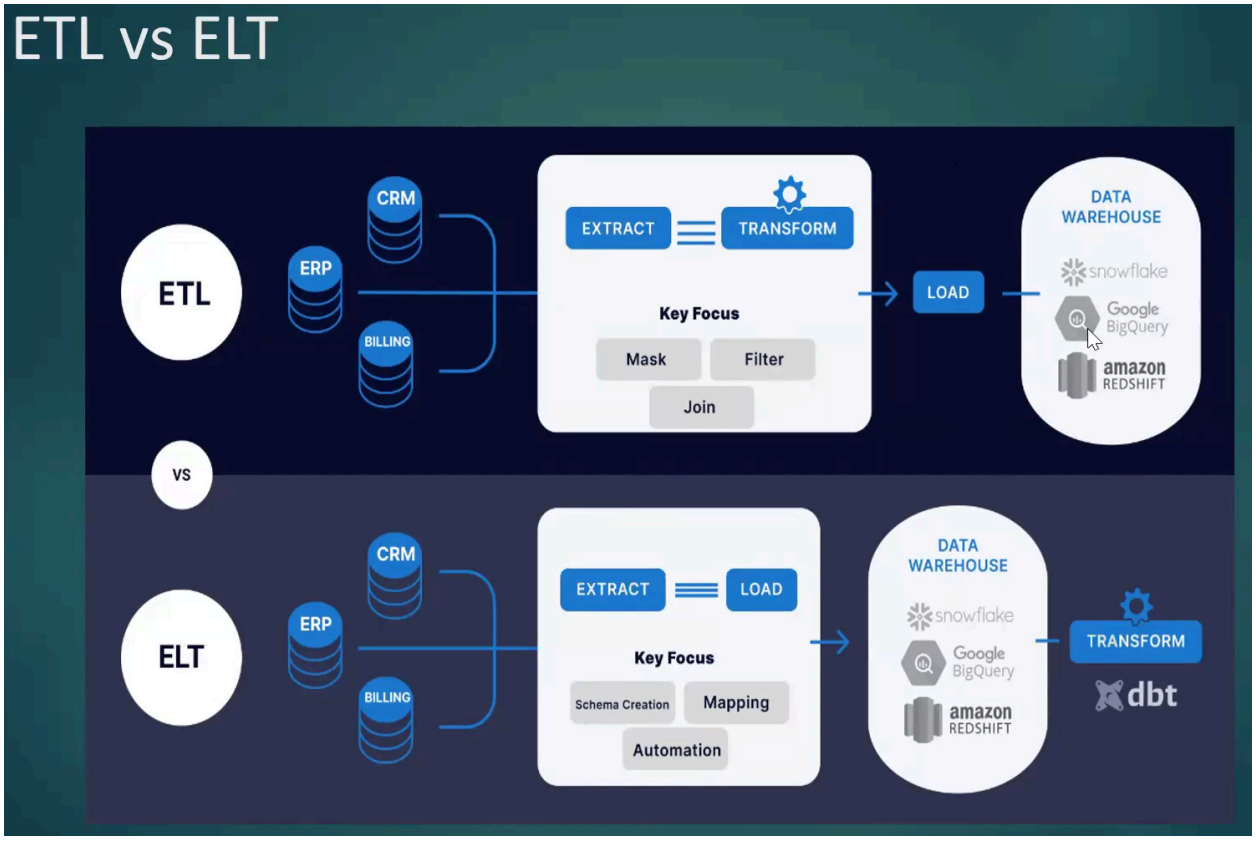
Integrated data: data from different sources (E.G., Sales, marketing, finance) is combined into A single, unified format, enabling A holistic view of the organization.

Historical data: it stores historical data, allowing analysis over time to identify trends, patterns, and insights.

Optimized for queries: data is organized and structured in ways that support complex queries, aggregations, and reporting, making it faster and easier to analyze.

Subject-oriented: data is organized around specific subjects or business areas like customer behavior, product sales, etc.





	E	T	L			
	oltp	python,datastage, Alteryx	datawarehouse			
			dw tables			
	E	L	T			
	extract	load to datawarehhouse				
			raw	staging	dw table	

Need for a data warehouse ?
Why we cannot do analytics on OLTP system?

1. Purpose and Optimization:

•OLTP systems (e.g., relational databases like MySQL, PostgreSQL) are designed to handle day-to-day transactions efficiently. They are optimized for fast inserts, updates, and deletes, and their focus is on transactional integrity, concurrency, and high availability.

•Data Warehouses (e.g., Snowflake, Amazon Redshift, Google Bigquery) are optimized for analytical processing. They are designed to handle complex queries, aggregations, and report generation across large datasets, where data is usually read more than written.

2. Performance Concerns:

•OLTP systems are optimized for quick transactions (e.g., placing an order, updating inventory). Running complex queries (like joins across multiple tables, aggregations, or reports) on an OLTP system can severely slow down these transactional processes.

•Data Warehouses are optimized for reading and querying large amounts of data. They use column store, data compression, data partitioning and parallel processing to handle heavy analytical workloads without impacting performance.

•3. Data Structure:

•OLTP systems have a highly normalized schema (e.g., multiple small tables with relationships) to minimize data redundancy and optimize transactional efficiency.

•Data Warehouses typically use denormalized structures (e.g., star schema, snowflake schema) to optimize query performance. This makes it easier to perform aggregations and complex joins, which are common in analytics.

•4. Historical Data and Large Volumes:

•OLTP systems are built to store current, real-time transactional data and are not designed to store large volumes of historical data.

•Data Warehouses are designed to handle large volumes of historical data. They store data over long periods, allowing you to analyze trends, patterns, and insights over time.

5. Separation of Concerns:

•OLTP systems need to be available 24/7 for transactional operations. Running heavy analytical queries on an OLTP system increases the risk of slowing down or crashing the system, affecting live operations.

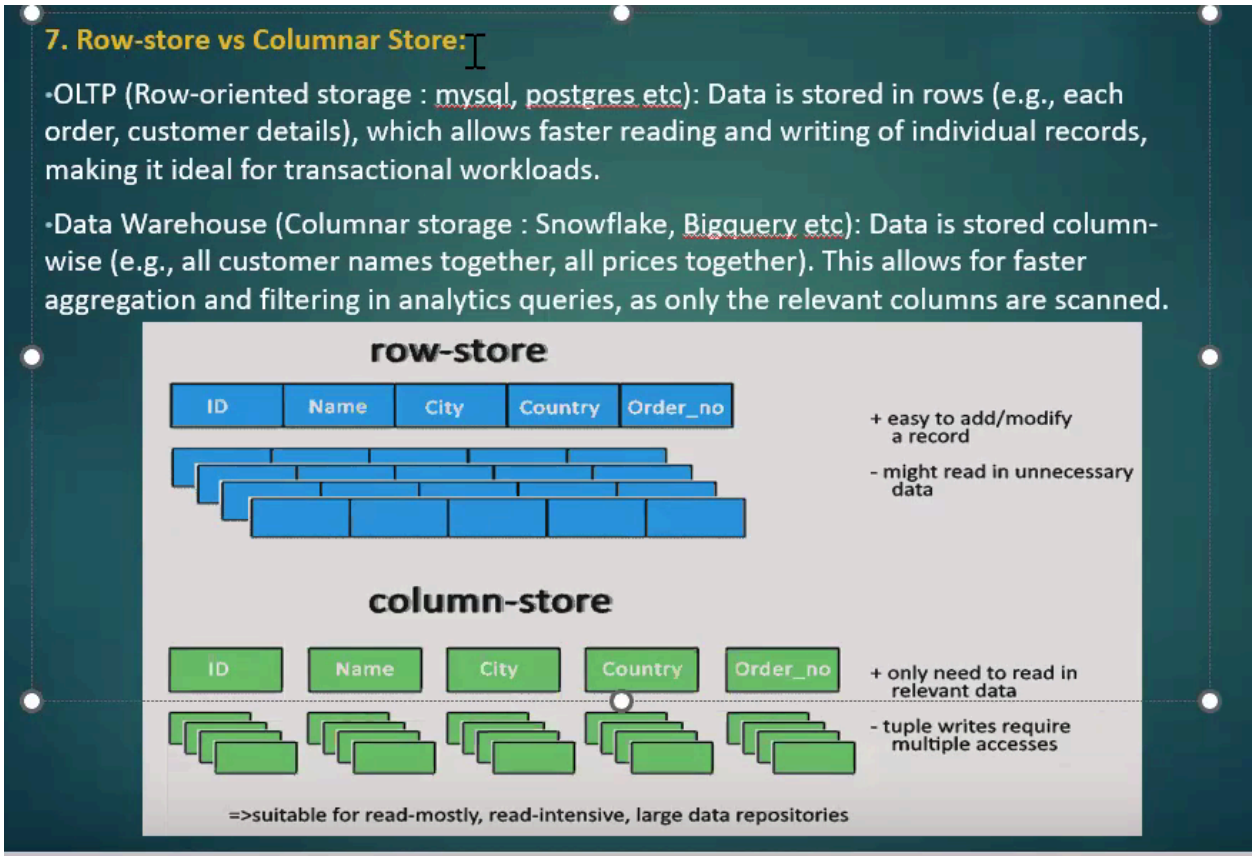
•Data Warehouses are dedicated for analytics, keeping analytical workloads separate from day-to-day operations, which reduces the risk of disrupting business-critical services.

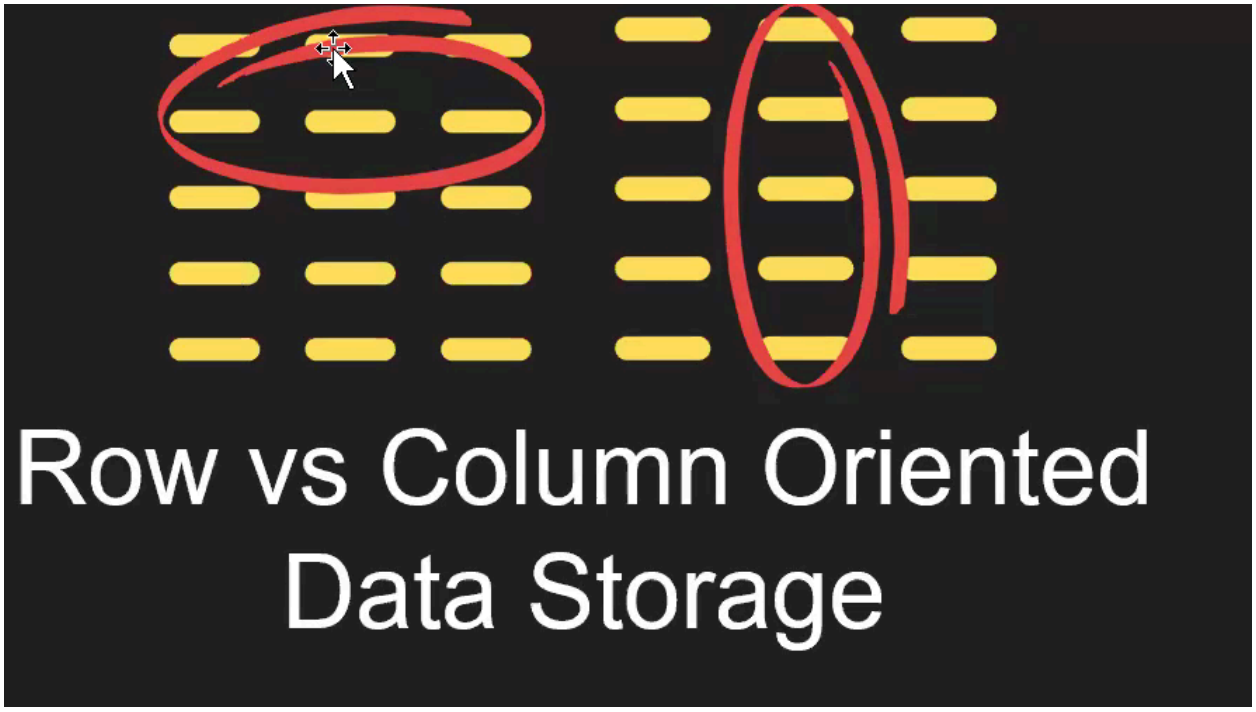
6. Concurrency:

•OLTP systems are designed for many small, quick transactions happening simultaneously (e.g., hundreds of users updating their shopping cart). They struggle with large-scale analytical queries that might lock tables or affect transaction performance.

•Data Warehouses are designed to handle multiple, concurrent analytical queries without affecting performance.

OTLP		OLAP				OLAP
3 nf		1 ow data is stored in the system				star schema





Table

rowid	id	first_name	last_name	ssn	salary	dob	title	joined
1001	1	John	Smith	111	101,000	1/1/1991	eng	1/1/2011
1002	2	Kary	White	222	102,000	2/2/1992	mgr	2/1/2012
1003	3	Norman	Freeman	333	103,000	3/3/1993	mkt	3/1/2013
1004	4	Nole	Smith	444	104,000	4/4/1994	adm	4/1/2014
1005	5	Dar	Sol	555	105,000	5/5/1995	adm	5/1/2015
1006	6	Yan	Thee	666	106,000	6/6/1996	mkt	6/1/2016
1007	7	Hasan	Ali	777	107,000	7/7/1997	acc	7/1/2017
1008	8	Ali	Bilal	888	108,000	8/8/1998	acc	8/1/2018

Queries

- Select first_name from emp where ssn = 666
- Select * from emp where id = 1
- Select sum(salary) from emp

Row-Oriented Database

- Tables are stored as rows in disk
- A single block io read to the table fetches multiple rows with all their columns.
- More IOs are required to find a particular row in a table scan but once you find the row you get all columns for that row.

Row-Oriented Database

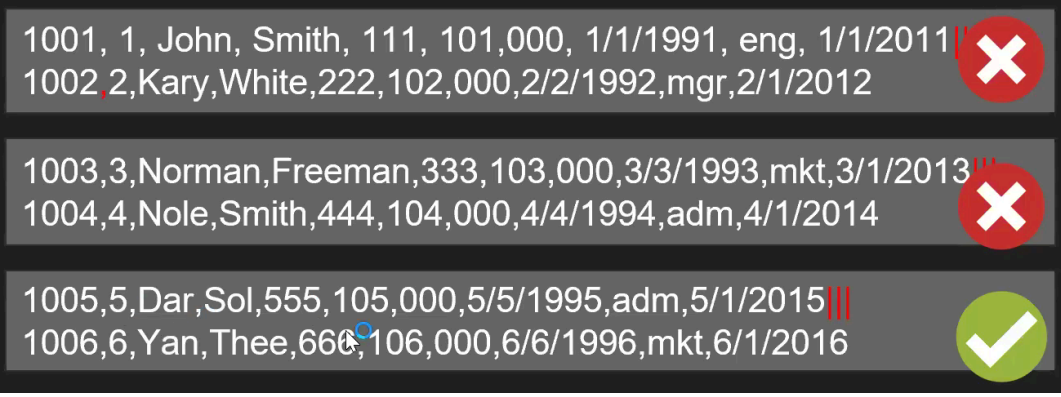
1001, 1, John, Smith, 111, 101,000, 1/1/1991, eng, 1/1/2011
1002,2,Kary,White,222,102,000,2/2/1992,mgr,2/1/2012

1003,3,Norman,Freeman,333,103,000,3/3/1993,mkt,3/1/2013
1004,4,Nole,Smith,444,104,000,4/4/1994,adm,4/1/2014

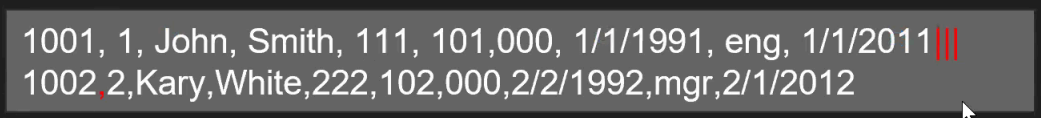
1005,5,Dar,Sol,555,105,000,5/5/1995,adm,5/1/2015
1006,6,Yan,Thee,666,106,000,6/6/1996,mkt,6/1/2016

1007,7,Hasan,Ali,777,107,000,7/7/1997,acc,7/1/2017
1008,8,Ali,Bilal,888,108,000,8/8/1998,acc,8/1/2018

Select first_name from emp where ssn=666



Select * from Emp where id = 1



Select sum(salary) from emp

1001, 1, John, Smith, 111, 101,000, 1/1/1991, eng, 1/1/2011|||
1002,2,Kary,White,222,102,000,2/2/1992,mgr,2/1/2012

1003,3,Norman,Freeman,333,103,000,3/3/1993,mkt,3/1/2013|||
1004,4,Nole,Smith,444,104,000,4/4/1994,adm,4/1/2014

1005,5,Dar,Sol,555,105,000,5/5/1995,adm,5/1/2015|||
1006,6,Yan,Thee,666,106,000,6/6/1996,mkt,6/1/2016

1007,7,Hasan,Ali,777,107,000,7/7/1997,acc,7/1/2017|||
1008,8,Ali,Bilal,888,108,000,8/8/1998,acc,8/1/2018

Column-Oriented Warehouses

- Tables are stored as columns first in disk
- A single block io read to the table fetches multiple columns with all matching rows
- Less IOs are required to get more values of a given column. But working with multiple columns require more IOs.

How in this select * (selecting all columns is costly)- going to read each block and then each column- in general analytics we will do aggregation

- Row-Based
- Optimal for read/writes
- OLTP
- Compression isn’t efficient
- Aggregation isn’t efficient
- Efficient queries w/multi-columns

- Column-Based
- Writes are slower
- OLAP
- Compress greatly
- Amazing for aggregation
- Inefficient queries w/multi-columns

Data Compression (In Columnar Storage)

Table

orderId	region	Sales
1	North	100
2	North	150
3	South	100
4	North	200
5	South	250
6	South	100
7	South	200
8	South	500

Step 1: Columnar Storage

In columnar storage, data is stored column by column rather than row by row.

For our example, the columns are stored as:

- **Transaction_ID**: 1, 2, 3, 4, 5, 6, 7, 8
- **Region**: North, North, South, North, South, South, South, South
- **Sales_Amount**: 100, 150, 200, 100, 200, 100, 200, 500

1. Dictionary Encoding (for "Region" col

This technique replaces repeated values with small, unique numeric codes, reducing storage space.

Original Data:
- North, North, South, North, South, South, South, South

Dictionary:
North = 1
South = 2

Encoded Data:
1, 1, 2, 1, 2, 2, 2, 2

2. Run-Length Encoding (for "Region" col

This technique compresses consecutive repeated values by storing the value and its frequency. Only stores changes and counts, significantly reducing size for repeated values.

Original Data:
1, 1, 2, 1, 2, 2, 2, 2 (after dictionary encoding)

Encoded Data:
(1, 2), (2, 1), (1, 1), (2, 4)

Meaning: "1 appears 2 times, 2 appears 1 time, 1 appears 1 time, 2 appears 4 time"

Compression Ratios

The compression ratio depends on the redundancy and patterns in the data:

- Highly repetitive data (e.g., "Region" column) achieves better compression.
- Unique or less predictable data (e.g., “Order_ID" column) achieves lower compression.

Decompression During Query Execution

When you query the data:

1- The compressed data is read from storage.

2- It is decompressed on the fly to reconstruct the original data.

3- The query processes the decompressed data in memory.

Key Benefits of Compression

Reduced Storage Cost: Smaller storage footprint saves costs in cloud storage systems.

Faster Query Execution: Less data to read from disk, reducing I/O overhead.

Transparent Handling: Compression and decompression are automatic and invisible to users.

8. Massive Parallel Processing (MPP):

OLAP (Online Analytical Processing) systems require Massive Parallel Processing (MPP) to efficiently handle complex, analytical queries involving large datasets. MPP allows these queries to be processed simultaneously across multiple nodes, significantly speeding up the execution of operations like aggregations and multi-dimensional analyses. OLAP deals with historical data and extensive analytical tasks, making MPP crucial for rapid data retrieval and analysis.

In contrast, OLTP (Online Transaction Processing) systems focus on managing simple, high-volume transactions that prioritize speed and data integrity. Since OLTP queries are typically straightforward and involve fewer data manipulations, they do not benefit from the parallel processing capabilities of MPP. Instead, OLTP systems aim for quick read and write operations without the overhead associated with MPP, ensuring efficient transaction handling.

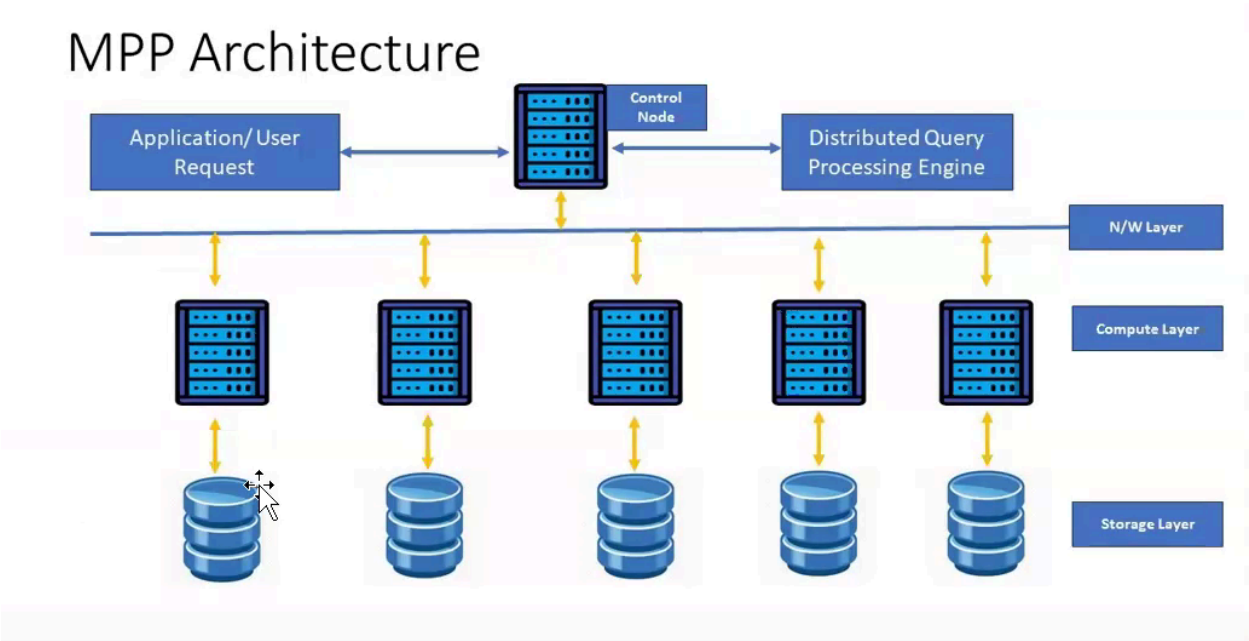
Example:

OLTP: select * from orders where order_id=100

OLAP: select category , sum(sales) as sales

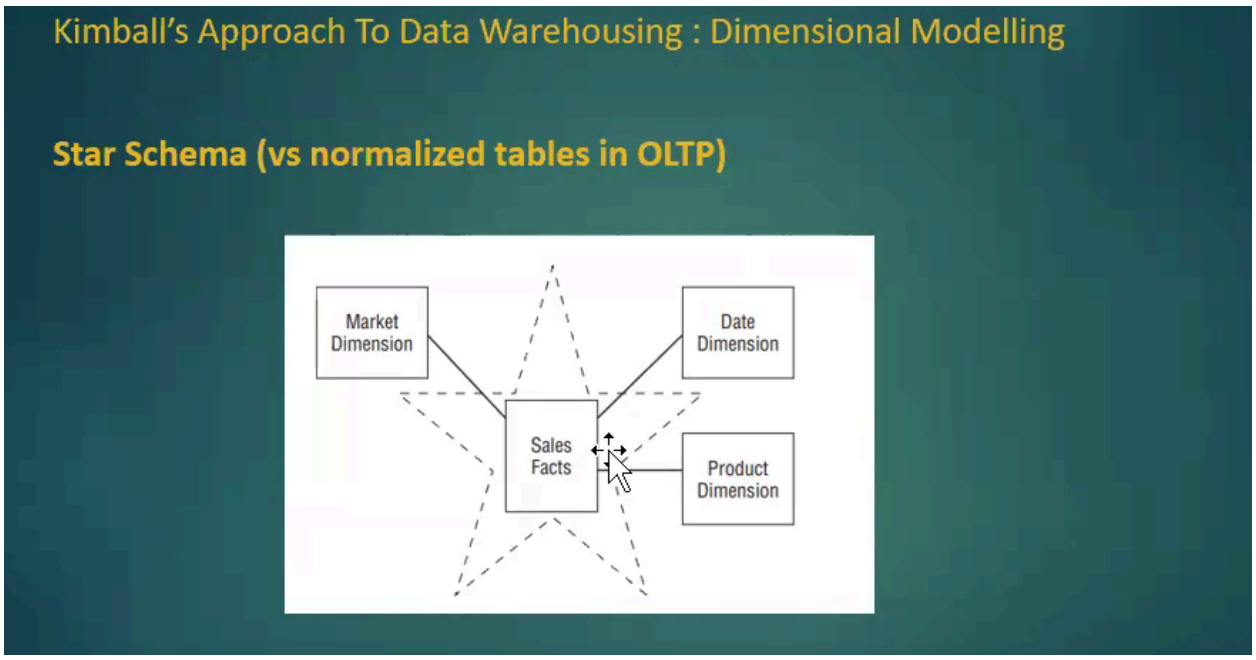
from orders

group by category



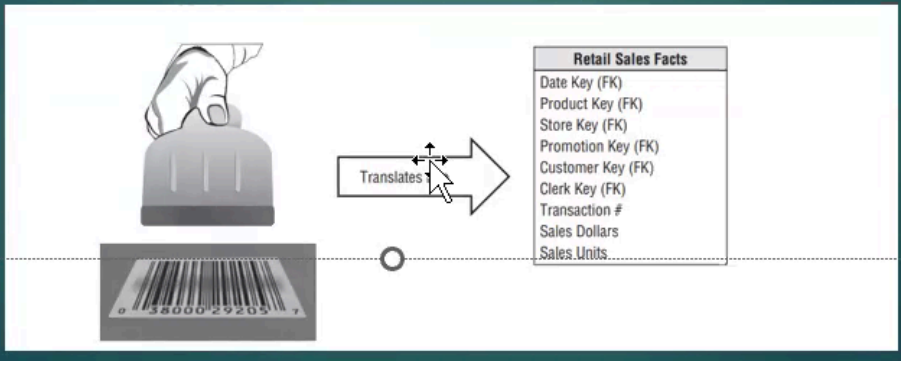
Let's assume we are going to run - select category, sum(sales) from orders

	categor a				node 1	category a	
	category b				node 2	category b	output
	category c				node 3	category c	
	category d				node 4	category d	



Fact Table

The term fact represents a business measure. Imagine standing in the marketplace watching products being sold and writing down the unit quantity and dollar sales amount for each product in each sales transaction. These measurements are captured as products are scanned at the register. Each row in a fact table corresponds to a measurement event. The data on each row is at a specific level of detail, referred to as the grain, such as one row per product sold on a sales transaction. One of the core tenets of dimensional modeling is that all the measurement rows in a fact table must be at the same grain. The most useful facts are numeric and additive, such as dollar sales amount.



Dimension Table

Dimension tables are integral companions to a fact table. The dimension tables contain the textual context associated with a business process measurement event. They describe the "who, what, where, when, how, and why" associated with the event.

Dimension tables often have many columns or attributes. It is not uncommon for a dimension table to have 50 to 100 attributes. They tend to have fewer rows than fact tables, but can be wide with many large text columns.

Each dimension is defined by a single primary key (refer to the PK notation in Figure 1-3), which serves as the basis for referential integrity with any given fact table to which it is joined.

Product Dimension
Product Key (PK)
SKU Number (Natural Key)
Product Description
Brand Name
Category Name
Department Name
Package Type
Package Size
Abrasive Indicator
Weight
Weight Unit of Measure
Storage Type
Shelf Life Type
Shelf Width
Shelf Height
Shelf Depth
...

Dimensions are denormalized for simplicity and accessibility

Dimension tables often represent hierarchical relationships. For example, products roll up into brands and then into categories. For each row in the product dimension, you should store the associated brand and category description. The hierarchical descriptive information is stored redundantly in the spirit of ease of use and query performance. You should resist the perhaps habitual urge to normalize data by storing only the brand code in the product dimension and creating a separate brand lookup table, and likewise for the category description in a separate category lookup table. This normalization is called snowflaking. Instead of third normal form, dimension tables typically are highly denormalized with flattened many-to-one relationships within a single dimension table.

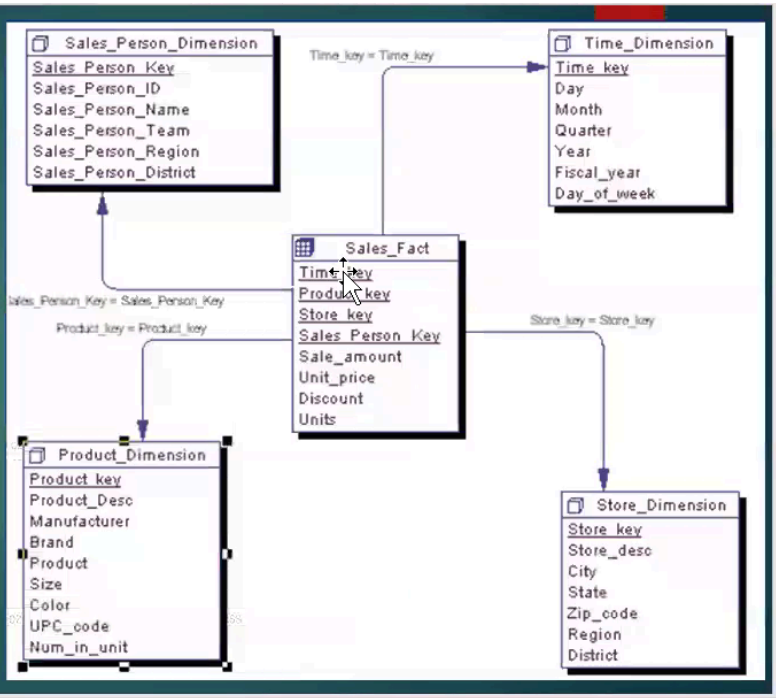
Product Key	Product Description	Brand Name	Category Name
1	PowerAll 20 oz	PowerClean	All Purpose Cleaner
2	PowerAll 32 oz	PowerClean	All Purpose Cleaner
3	PowerAll 48 oz	PowerClean	All Purpose Cleaner
4	PowerAll 64 oz	PowerClean	All Purpose Cleaner
5	ZipAll 20 oz	Zippy	All Purpose Cleaner
6	ZipAll 32 oz	Zippy	All Purpose Cleaner
7	ZipAll 48 oz	Zippy	All Purpose Cleaner
8	Shiny 20 oz	Clean Fast	Glass Cleaner
9	Shiny 32 oz	Clean Fast	Glass Cleaner
10	ZipGlass 20 oz	Zippy	Glass Cleaner
11	ZipGlass 32 oz	Zippy	Glass Cleaner

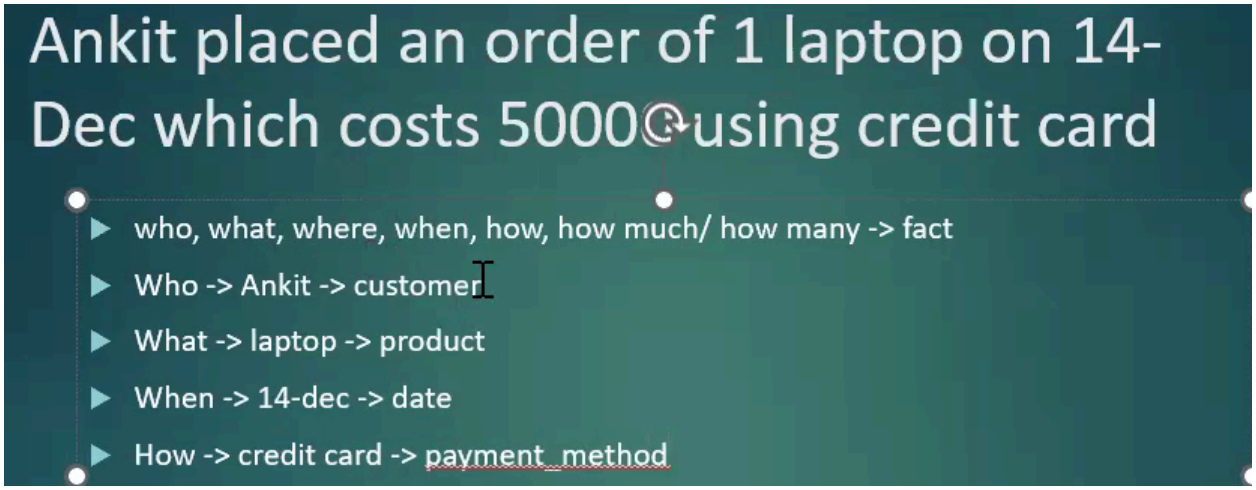
Star Schema - All the dimensions are directly connected with the fact table and dimensions tables are not further divided into multiple tables.

Facts and Dimensions in a star schema

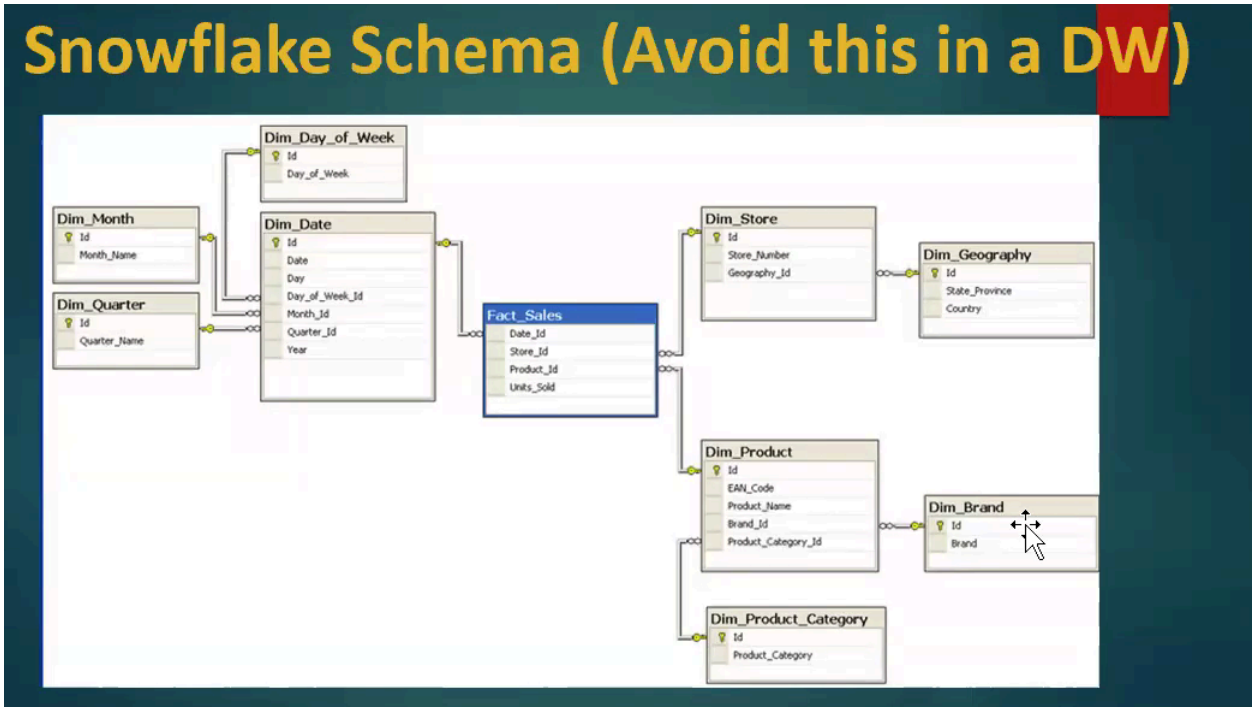
Now that you understand fact and dimension tables, it's time to bring the building blocks together in a dimensional model, as shown in Figure.

Each business process is represented by a dimensional model that consists of a fact table containing the event's numeric measurements surrounded by a halo of dimension tables that contain the textual context that was true at the moment the event occurred. This characteristic star-like structure is often called a star join or star schema.





Snowflake - Dimensions are further divided into multiple tables (similar 3rd Normalization)



Steps of dimensional modelling:

1- Choose The business processes

Identify the processes or operations the business wants to analyze. Examples: Sales transactions, inventory management, order fulfillment, customer support.

2- Declare the grain (or granularity)

Granularity determines what a single row in the fact table represents. Declaring the grain ensures clarity about what data the fact table will store and prevents ambiguity during data loading. Examples: Per transaction (e.g., one row for each sale) , Daily summary (e.g., one row per store per day).

3- Identify the dimensions

Define the descriptive attributes (dimensions) that provide context for the facts.

Dimensions answer the "who," "what," "where," "when," "why," and "how" of the business process.

Examples: Product, Customer, Time, Store, Region.

4- Identify the facts

Identify the numeric, measurable values that are stored in the fact table.Facts are metrics that are analyzed or aggregated. Examples: Sales revenue, units sold, profit, cost.

Fact Tables

Types of fact tables :

Transaction Fact Table: Detailed records of individual transactions (e.g., sales).

Snapshot Fact Table: Periodic snapshots of a process (e.g., inventory levels).

Accumulating Fact Table: Tracks the progress of processes that evolve over time (e.g., order fulfillment).

Fact less Fact Table: Tracks the occurrence of events without numeric data (e.g., attendance).

Examples on next slides ->>>

Transaction Fact Table:

This fact table records individual sales transactions. Each row represents one transaction, detailing the customer who made the purchase (customer_id), the product purchased (product_id), the quantity, and the total amount. The granularity is very high, as each row reflects a specific sale that occurred at a specific time.

Example: Sales Transactions

transaction_id	customer_id	product_id	quantity	total_amount	transaction_date
101	1	201	2	500	2024-10-15
102	2	202	1	250	2024-10-16
103	1	203	3	750	2024-10-17

Use Case: Useful for analyzing daily sales by product, customer, or region to track revenue, profit, or sales trends.

Snapshot Fact Table: This table captures a snapshot of inventory levels at different points in time. Each row represents the stock of a product (product_id) at a specific warehouse (warehouse_id) as recorded on a specific date (snapshot_date). Stock levels are periodically recorded, making this table useful for time-based comparisons of inventory.

Example: Inventory Snapshot

product_id	warehouse_id	stock_level	snapshot_date
201	1	150	2024-10-01
202	2	200	2024-10-01
201	1	130	2024-10-15
202	2	180	2024-10-15

Use Case: Used for analyzing inventory trends over time, such as understanding seasonal fluctuations or ensuring stock levels meet demand.

Accumulating Fact Table: This accumulating fact table tracks the progress of an order through different stages of its lifecycle (ordering, payment, shipping, delivery). As each stage is completed, the relevant date columns (payment_date, shipping_date, delivery_date) are filled in. Rows are updated as orders progress, so it tracks the completion status of each order.

Example: Order Fulfillment Lifecycle

order_id	order_date	payment_date	shipping_date	delivery_date	total_amount
101	2024-10-10	2024-10-11	2024-10-12	2024-10-15	500
102	2024-10-12	2024-10-13	2024-10-14	2024-10-18	250
103	2024-10-15	2024-10-16	NULL	NULL	750

Use Case: This is ideal for analyzing the time taken for each stage of order processing, such as the average shipping time or identifying delays in deliveries.

Factless Fact Table:

A factless fact table captures events or occurrences without actual numeric data. In this case, it records student attendance. Each row represents a student attending a specific class on a specific date, but no measurements (e.g., grades, hours) are tracked—just the event of attendance.

Example: Student Attendance

student_id	class_id	attendance_date
1	101	2024-10-15
2	102	2024-10-15
1	101	2024-10-16
2	102	2024-10-16

Use Case: Useful for tracking attendance, participation, or event logging, where the occurrence of an event is the focus rather than any quantitative data.

Slowly changing dimensions:

Definition: A dimension where data changes slowly over time, requiring historical tracking. eg:

1- Customer details: A customer's name, address, and phone number can change over time.

2- Product descriptions: If a product's ingredients change, the description in the product dimension table needs to be updated.

3- Geographical locations: A historical value might show that a location was in Illinois, but the current value shows it's in New York.

4- Employee names: An employee's name might change after marriage.

5- Products Cost Price: The cost price of the products can change over time. How would you measure the profit.

Types of (Slowly changing dimensions) tables :

SCD Type 1: Overwrites the old data with new data (no history).

SCD Type 2: Creates a new record for every change, keeping the history.

SCD Type 3: Stores only limited history (previous and current values).

SCD1 -> overwrite the changes (no history)

products			100	Jan-01				
product_id	product_name	cost_price	90	Jan-30		Jan-15	120	20
1	iphone 5	90		scd - type 1		Jan-31	120	30

here we need to store history for profit calculation

customer			
customer_id	first name	last name	scd1
1	ankit	gupta	

here we can change last name without any history

scd type 2

products				
product_id	product_name	cost_price	effective_date	end_date
1	iphone 5	100	2024-01-01	2024-01-29
1	iphone 5	90	2024-01-30	2024-03-14
1	iphone 5	75	2024-03-15	9999-12-31

scd type 3

product_id	product_name	cost_price	prev_cost_price
1	iphone 5	75	90

You can see in scd2 that there is no primary key, so we have created a product key
-now it uniquely identifies each row and joins are fast

surrogate key

scd type 2
products

product_key	product_id	product_name	cost_price	effective_date	end_date
1	1	iphone 5	100	2024-01-01	2024-01-29
2	1	iphone 5	90	2024-01-30	2024-03-14
3	1	iphone 5	75	2024-03-15	9999-12-31

order

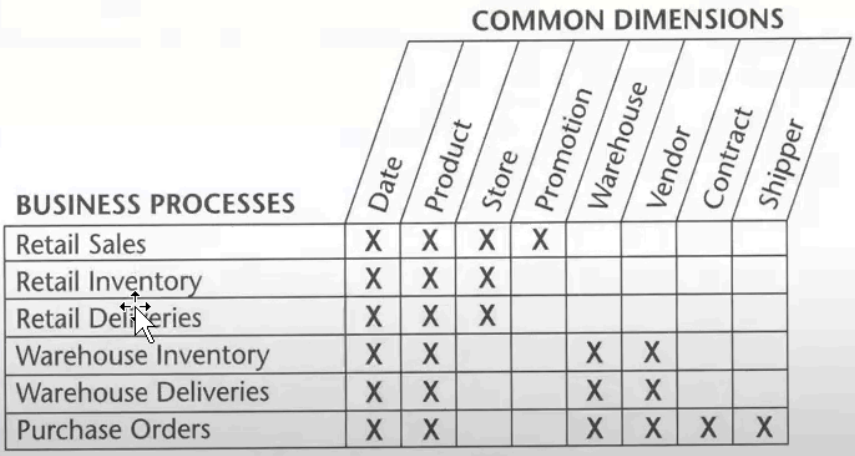
order_id	order_date	product_ke	sales
1	2024-01-01	1	120
2	30-01-2024	2	120

Conformed dimension: A dimension that is shared across multiple fact tables or data marts, maintaining consistent values across the entire data warehouse.

Example: A Date dimension used in both the sales and inventory fact tables to provide a unified view of time-related data.

Use Case: If multiple fact tables (e.g., Sales and Orders) use the same dimension (e.g., Date), this dimension is conformed to ensure consistency.

Bus matrix : Finding the common dimensions



BUSINESS PROCESSES	COMMON DIMENSIONS: Date	Product	Store	Promotion	Warehouse	Vendor	Contract	Shipper
Retail Sales	X	X	X	X				
Retail Inventory	X	X	X					
Retail Deliveries	X	X	X					
Warehouse Inventory	X	X			X	X		
Warehouse Deliveries	X	X			X	X		
Purchase Orders	X	X			X	X	X	X

Figure 3.8 Sample data warehouse bus matrix.