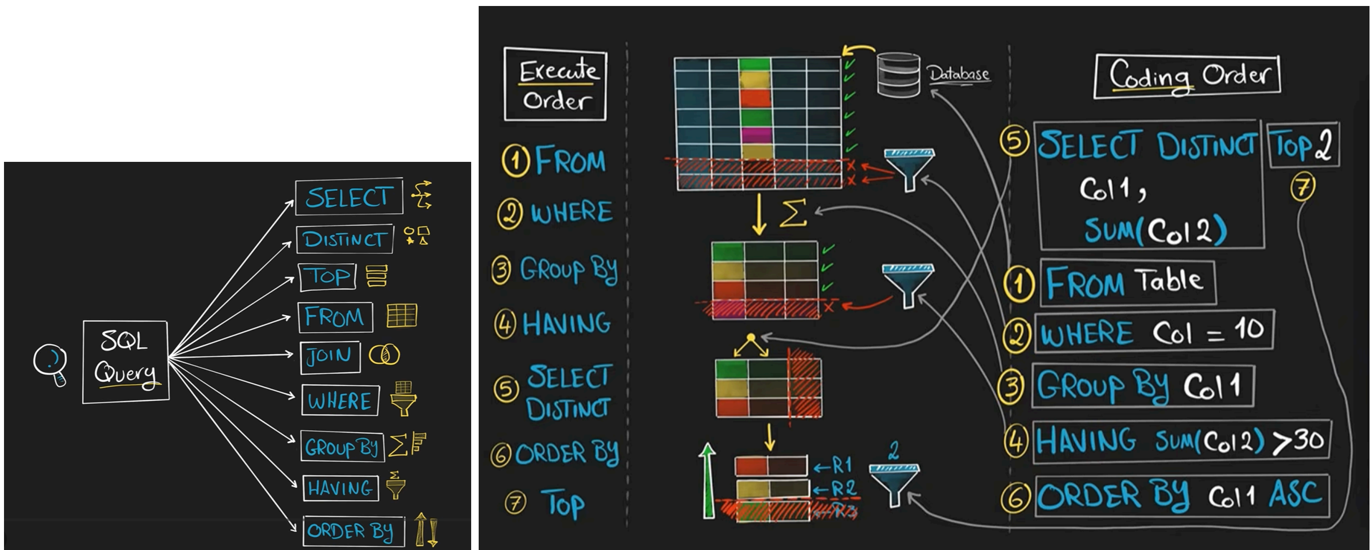
34:01 Query Data (SELECT)




```
-- Find the total score for each country

SELECT
    country,
    first_name,
    SUM(score) AS total_score
FROM customers
GROUP BY country
```


Aggregation function can be added as several ones.

Having

Filters Data After Aggregation

Can be used only with Group By

filter the result. like where

```
sort the results by the country and then b

SELECT *
FROM customers
ORDER BY country ASC, score DESC
```

IMPORTANT

Column order in ORDER BY is crucial, as sorting is sequential

	id	first_name	country	score
1	4	Martin	Germany	500
2	1	Maria	Germany	350
3	3	Georg	UK	750
4	2	John	USA	900
5	5	Peter	USA	0

select once

Bad Habit with DISTINCT

Don't use DISTINCT unless it's necessary; it can slow down your query

select distinct
country
from table

select 1 2 3 id of table. Row numbers retrieve only 2 customers

select top
3 *
from table

```
-- Retrieve the Top 3 Customers with the Highest Scores

SELECT TOP 3 *
FROM customers
ORDER BY score DESC
```

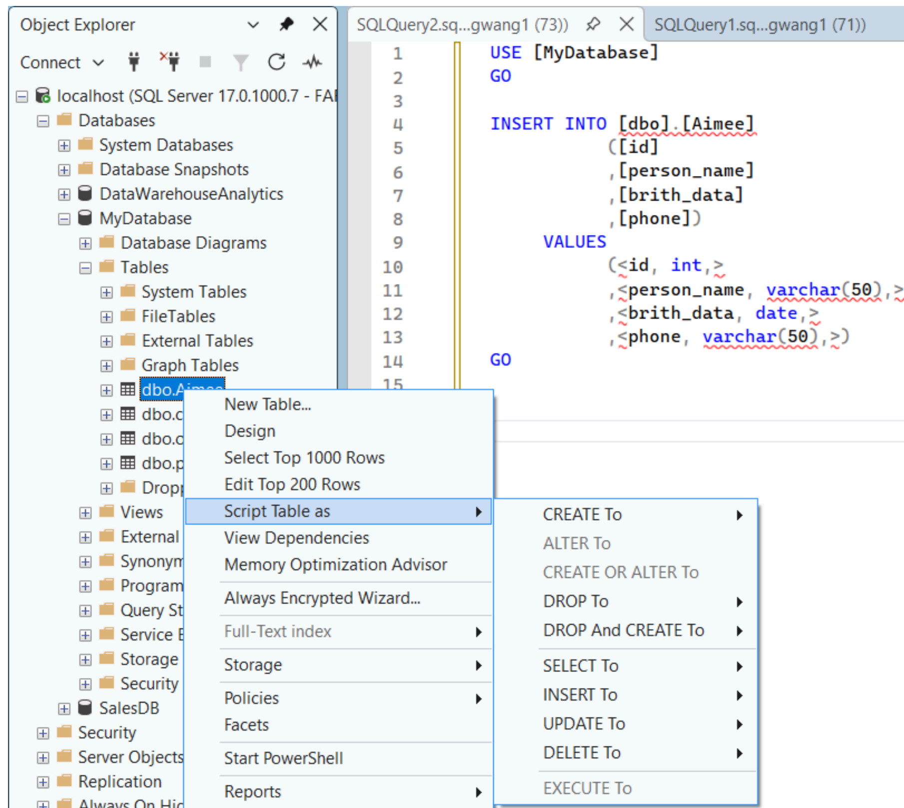
add info to database

select “new customer” , ... from database

01:32:31 Data Define Language Commands

create

Create table Aimee(
  id INT not null,
  person_name VARCHAR(50) NOT NULL,
  brith_data DATE,
  phone VARCHAR(50) NOT NULL,
CONSTRAINT primary_key Primary Key (id)
)



Alter

Alter Table Aimee ADD email VARCHAR(50) NOT NULL
        Alter table Aimee DROP column phone

Drop Table Aimee

01:43:44 Data Manipulate L Commands

Insert into aimee(
  id, person_name,brith_data DATE,
  phone
)
insert into Aimee (id,person_name,brith_data,email,phone)

Values (1,'aimee','20190807','','NULL')
error occurred
Cannot insert the value NULL into column 'id', table 'MyDatabase.dbo.Aimee'; column does not allow nulls. INSERT fails.
insert into Aimee (id, brith_data,email,phone)
Values (1,'20190807','','NULL')

```
INSERT INTO customers (id, first_name)
VALUES
    (10, 'Sahra')
```

NOTE
Columns not included in INSERT become NULL (unless a default or constraint exists)

change the column name

EXEC sp_rename 'dbo.Aimee.brith_data', 'birth_date', 'COLUMN';

Insert data from a table to another.

INSERT INTO persons (id, person_name, birth_date, phone)
SELECT id, first_name, NULL, 'unknown'
FROM customers
WHERE NOT EXISTS (SELECT 1 FROM persons p WHERE p.id = customers.id);

update

update table
set column1=value1, column2=value2
where

CAUTION
Without a WHERE, all rows will be updated!

BEST PRACTICE
Check with SELECT before running UPDATE to avoid updating the wrong data

Delete (slow)

delete from table
(where)

Truncate(fast)

truncate TABLE table_name

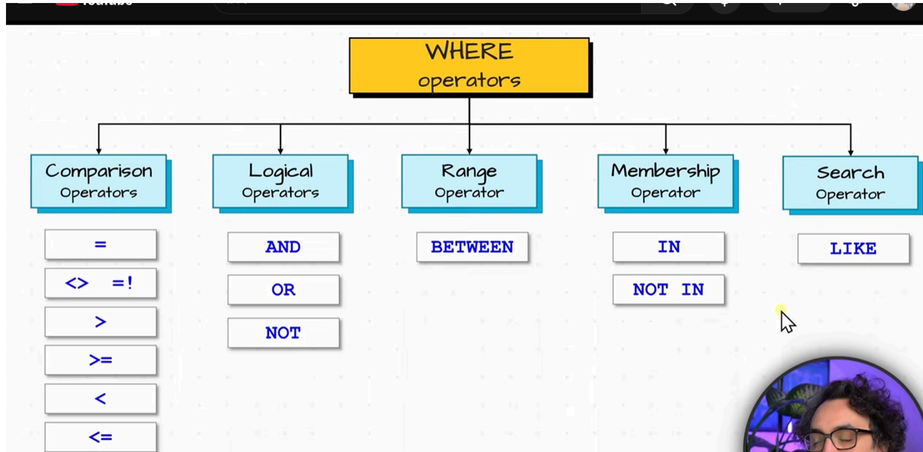
TRUNCATE

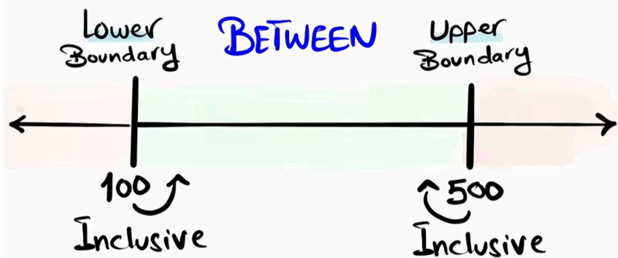
Clears the whole table at once without checking or logging

__🟡 *Intermediate Level*__

02:08:03 Filtering Data

Operators






```sql
SELECT
*
FROM customers
WHERE NOT score < 500
```

```sql
SELECT *
FROM customers
WHERE country = 'Germany' OR country = 'USA'

SELECT *
FROM customers
WHERE country IN ('Germany', 'USA')
```

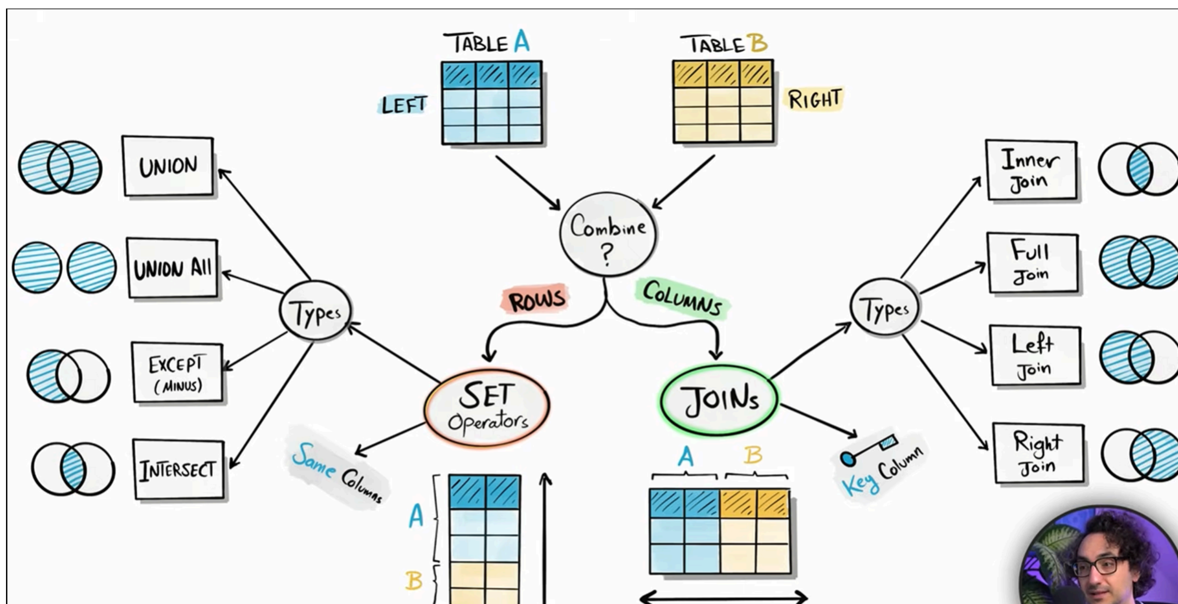
TIP

Use IN instead of OR for multiple values in the same column to simplify SQL

like “_ _ M%” 2 _ means 3rd position is M

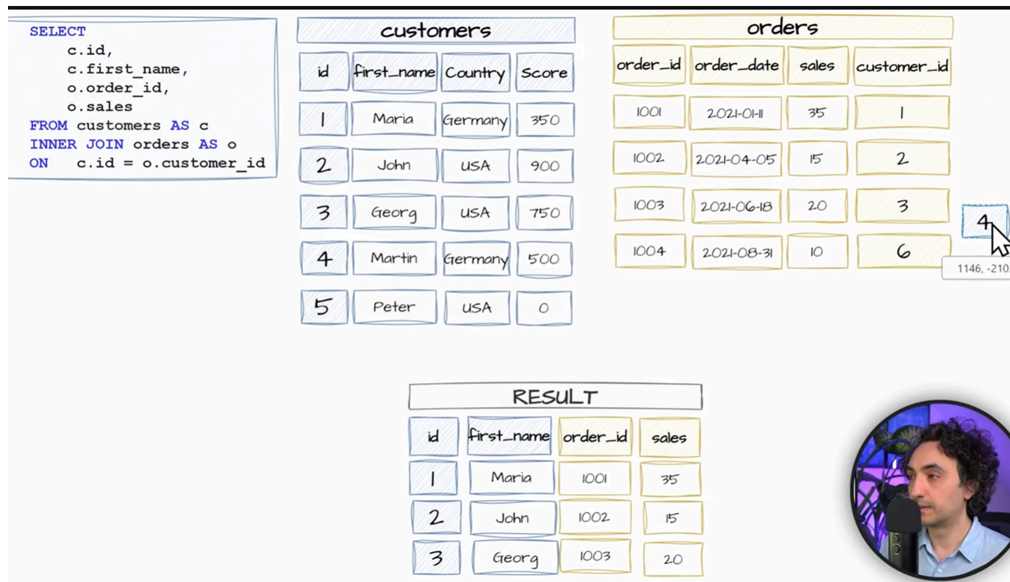
02:47:57 SQL Joins (Basics)

full picture. enrich. check existence

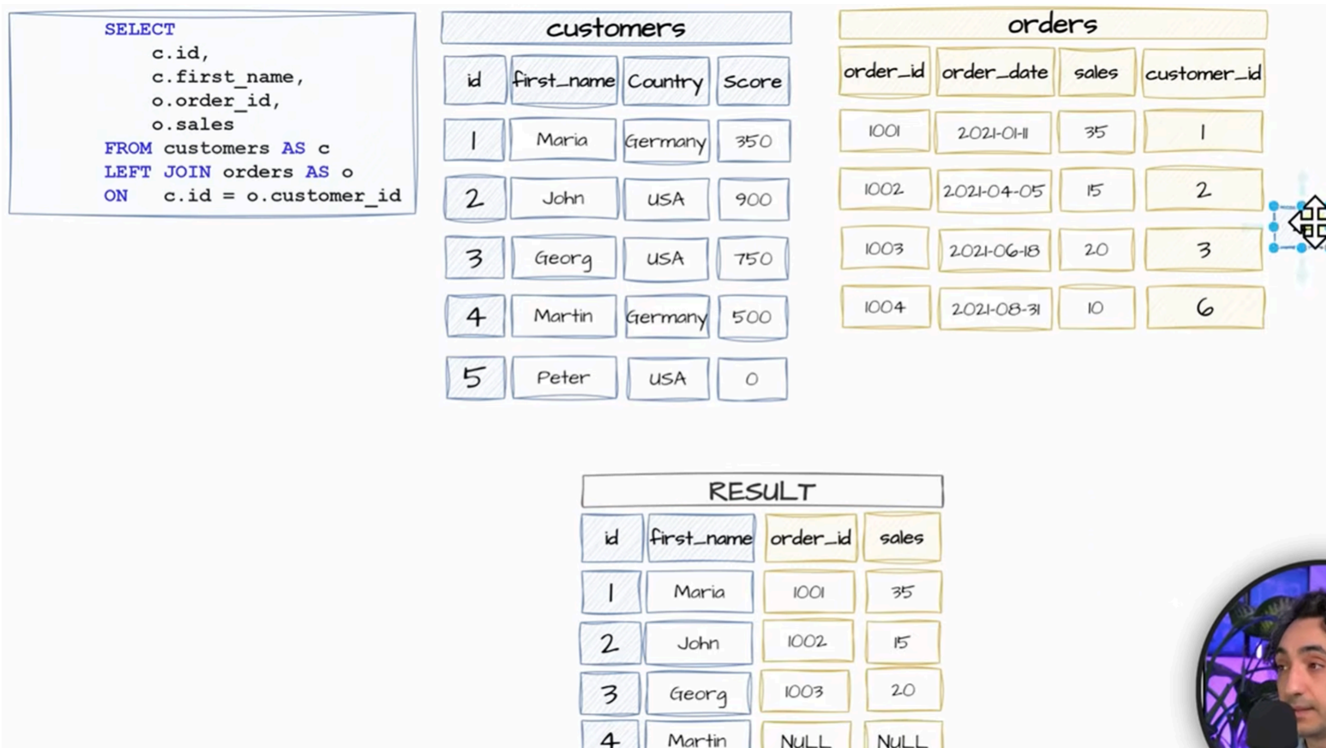


Inner join get more info / check info existence

customer who placed an order

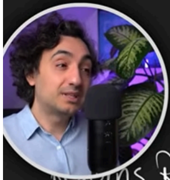


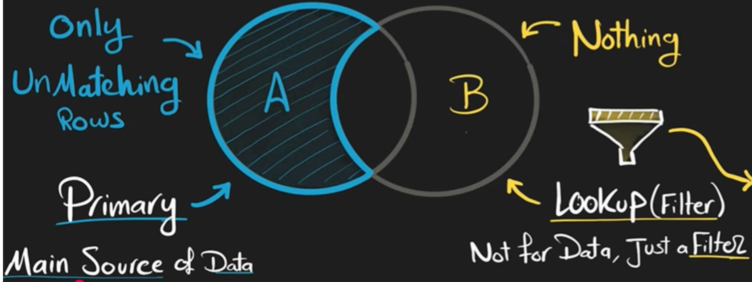
left join enrich data



fulljoin

left anti join

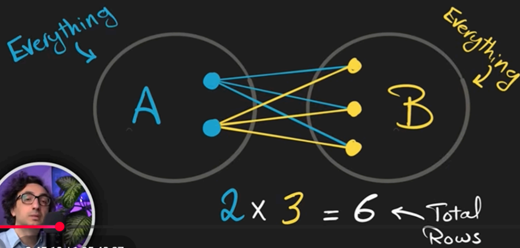




full antijoin

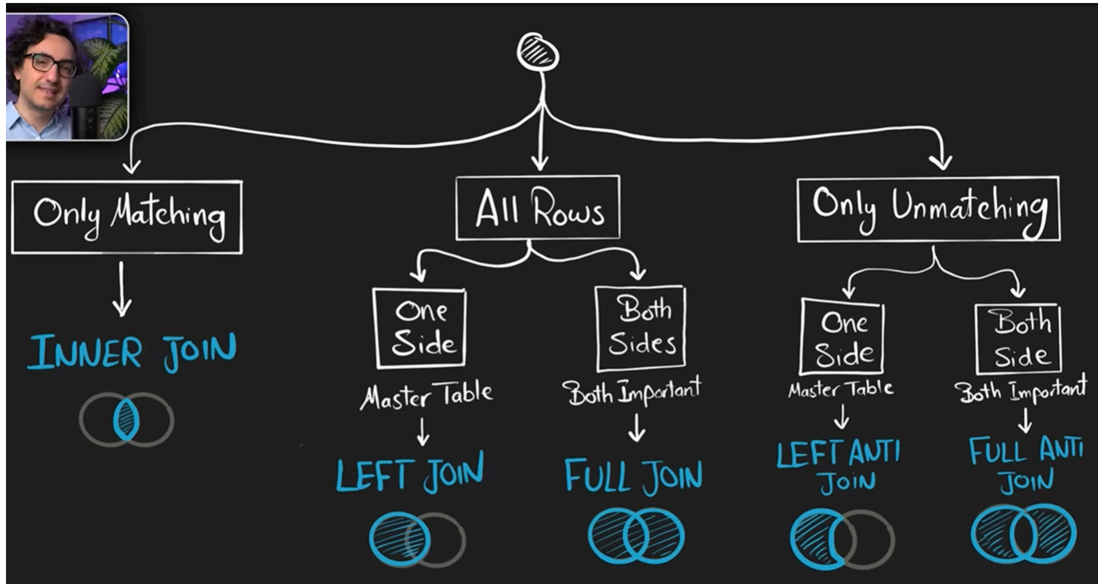
```sql
SELECT *
FROM orders AS o
FULL JOIN customers AS c
ON c.id = o.customer_id
WHERE c.id IS NULL OR o.customer_id IS NULL
```

cross join all possible combinations



03:27:29 SQL Joins (Advanced)

how to choose join type




```sql
-- Alternative to INNER JOIN using LEFT JOIN
/* Get all customers along with their orders,
   but only for customers who have placed an order */
SELECT *
FROM customers AS c
LEFT JOIN orders AS o
ON c.id = o.customer_id
WHERE o.customer_id IS NOT NULL
```

04:02:09 Set Operators

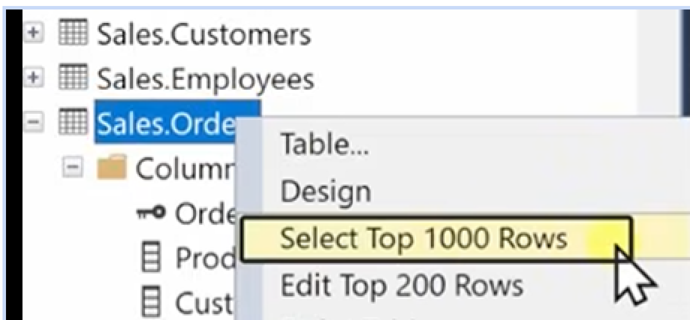
SET OPERATORS

Combine the results of multiple queries into a single result set

RULES
- Same Nr. of Columns, Data Types, order of columns.
- 1st Query Controls Column names.

USE CASES
- Combine Information (UNION + UNION All)
- Delta Detection (EXCEPT)
- Data Completeness Check (EXCEPT)

Union distinct automatically never use select *



helps you write the column names.

```sql
FROM Sales.Orders
UNION
SELECT
'OrdersArchive' AS SourceTable
,[OrderID]
,[ProductID]
,[CustomerID]
,[SalesPersonID]
,[OrderDate]
,[ShipDate]
,[OrderStatus]
,[ShipAddress]
,[BillAddress]
,[Quantity]
,[Sales]
,[CreationTime]
FROM Sales.OrdersArchive
ORDER BY OrderID
```

118 %

Results Messages

	SourceTable	OrderID	Proc
1	Orders	1	101
2	OrdersArchive	1	101
3	Orders	2	102
4	OrdersArchive	2	102
5	Orders	3	101
6	OrdersArchive	3	101
7	Orders	4	105
8	OrdersArchive	4	105
9	OrdersArchive	4	105
10	Orders	5	104
11	OrdersArchive	5	104

add labels

Union all including duplicates

RULES OF SET OPERATORS

#1 RULE | ORDER BY can be used only once

#2 RULE | Same Number of Columns

#3 RULE | Matching Data Types

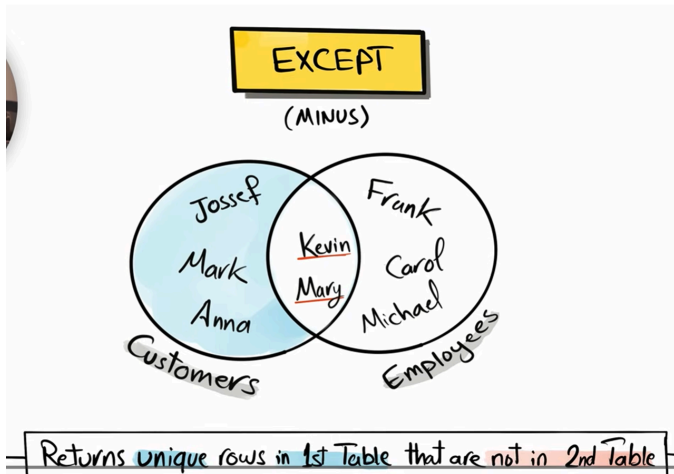
#4 RULE | Same Order of Columns

#5 RULE | First Query Controls Aliases

#6 RULE | Mapping Correct Columns

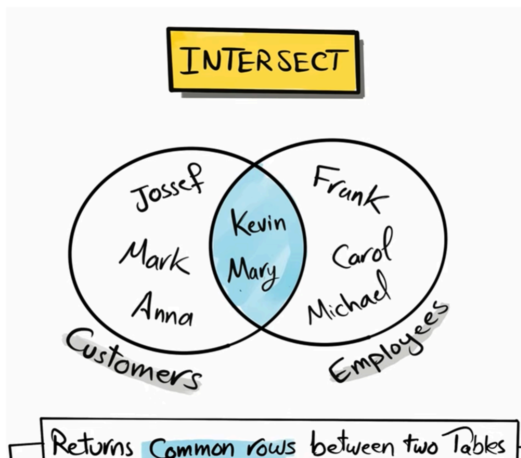
except order of table is very important!!!

check if data all is moved to another BD




```
*/
SELECT
    FirstName,
    LastName
FROM Sales.Employees
EXCEPT
SELECT
    FirstName,
    LastName
FROM Sales.Customers;
```

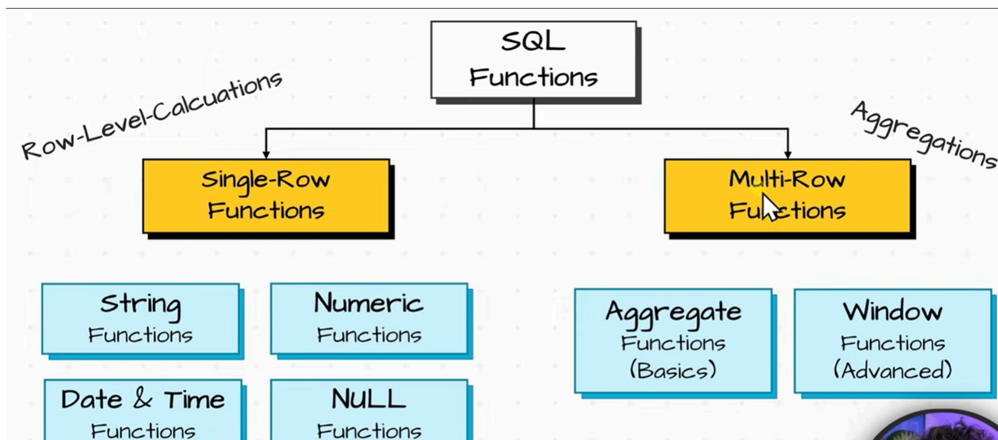
intersection

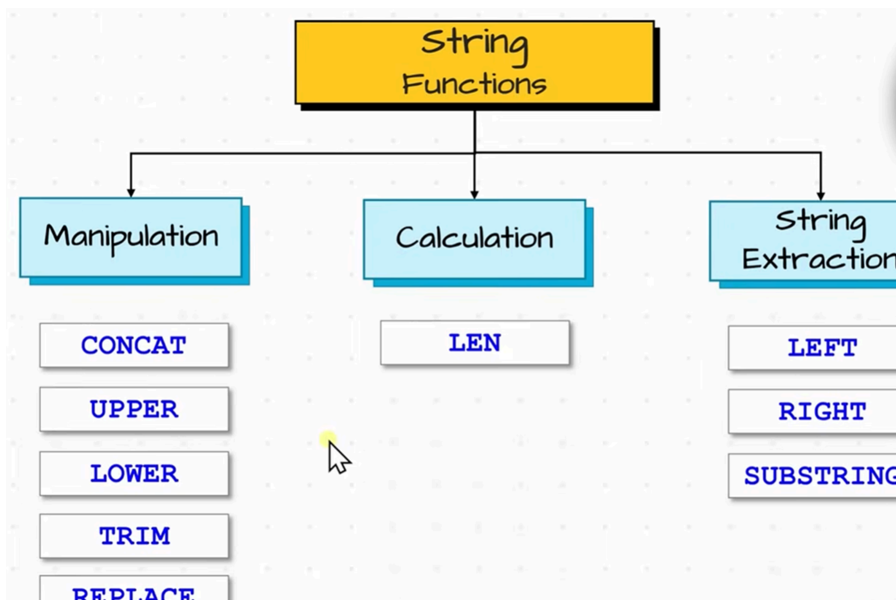



```
SELECT
    FirstName,
    LastName
FROM Sales.Employees
INTERSECT
SELECT
    FirstName,
    LastName
FROM Sales.Customers;
```

[04:47:41](#) SQL Functions

1. [04:52:58](#) String Functions






```
SELECT
first_name,
country,
CONCAT(first_name, ' ', country) AS name_country
```

```
SELECT
    first_name
FROM customers
WHERE first_name != TRIM(first_name)
```

trim remove space

```
SELECT
'123-456-7890' AS phone,
REPLACE('123-456-7890', '-', '') AS clean_phone
```

```
SELECT
    first_name,
    LEFT(first_name, 2) first_2_char
FROM customers
```

SUBSTRING (Value , Start , Length)

After the 2nd Character extract 2 Characters

M a r i a

1 2 3

Start 3

```
SELECT
    first_name,
    SUBSTRING(TRIM(first_name), 2, LEN(first_name)) AS sub_name
FROM customers
```

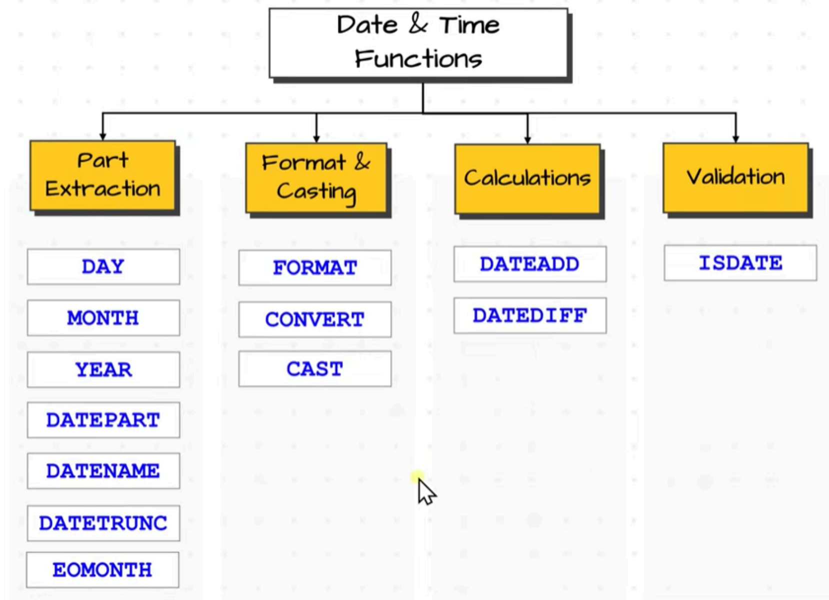
2. 05:18:44 Numeric Functions

ROUND
ABS

3. 05:22:48 Date and Time Functions

Table of Contents:

1. 1GETDATE | Date Values **GETDATE() AS TODAY**
2. Date Part Extractions (DATETRUNC, DATENAME, DATEPART, YEAR, MONTH, DAY)



2025-08-20
09:38:54.840

Date Parts

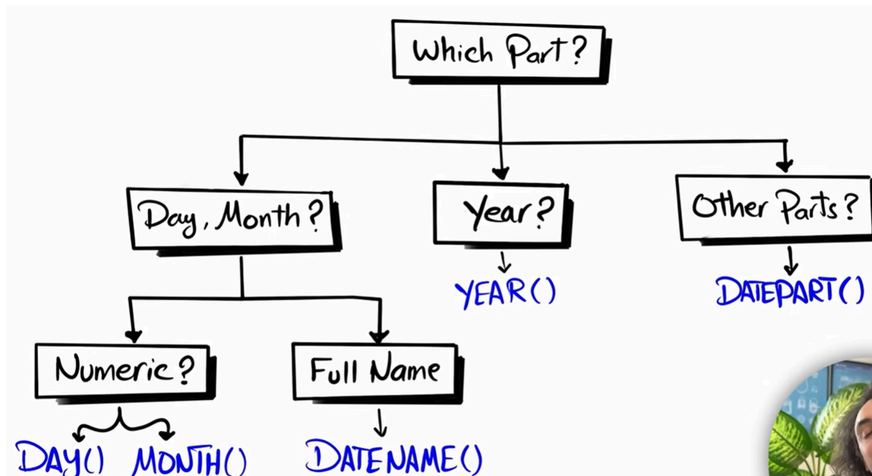
DATEPART

DATENAME

DATETUNC

Part	Abbre.	DATEPART (INT)	DATENMAME (String)	DATETRUNC (Datetime2)
year	yy, yyyy	2025	2025	2025-01-01 00:00:00
quarter	qq,q	3	3	2025-07-01 00:00:00
month	mm,m	8	August	2025-08-01 00:00:00
dayofyear	dy,y	232	232	2025-08-20 00:00:00
day	dd, d	20	20	2025-08-20 00:00:00
weekday	dw	4	Wednesday	Not supported
week	wk,ww	34	34	2025-08-17 00:00:00
iso_week	ns	34	34	2025-08-18 00:00:00
hour	hh	9	9	2025-08-20 09:00:00
minute	mi,n	45	45	2025-08-20 09:45:00
second	ss,s	21	21	2025-08-20 09:45:21
millisecond	ms	0	0	2025-08-20 09:45:21
microsecond	msc	0	0	2025-08-20 09:45:21
nanosecond	ns	0	0	Not supported
iso_week	isowk, isoww	0	+00:00	Not supported

How should we determine which function we should use.



2 YEAR(CreationTime) AS Year,

3 MONTH(CreationTime) AS Month,

4 DAY(CreationTime) AS Day

```
SELECT
OrderID,
CreationTime,
YEAR(CreationTime) Year,
MONTH(CreationTime) Month,
DAY(CreationTime) Day
FROM Sales.Orders
```

OrderID	CreationTime	Year	Month	Day
1	2025-01-01 12:34:56.0000000	2025	1	1
2	2025-01-05 23:22:04.0000000	2025	1	5
3	2025-01-10 18:24:08.0000000	2025	1	10
4	2025-01-20 05:50:33.0000000	2025	1	20

5EOMONTH

2025-01-31
2025-02-28

6 DATEPART Examples

DATEPART(year, CreationTime) AS Year_dp,
DATEPART(month, CreationTime) AS Month_dp,
DATEPART(day, CreationTime) AS Day_dp,
DATEPART(hour, CreationTime) AS Hour_dp,
DATEPART(quarter, CreationTime) AS Quarter_dp,

DATEPART(week, CreationTime) AS Week_dp，第几个星期

Year_dp	Month_dp	Day_dp	Hour_dp	Quarter_dp	Week_dp
2025	1	10	18	1	2
Click to select the whole column				1	4
2025	2	1	14	1	5
2025	2	6	15	1	6
2025	2	16	6	1	8
2025	2	18	10	1	8
2025	3	10	12	1	11

7DATENAME(month, CreationTime) AS Month_dn,

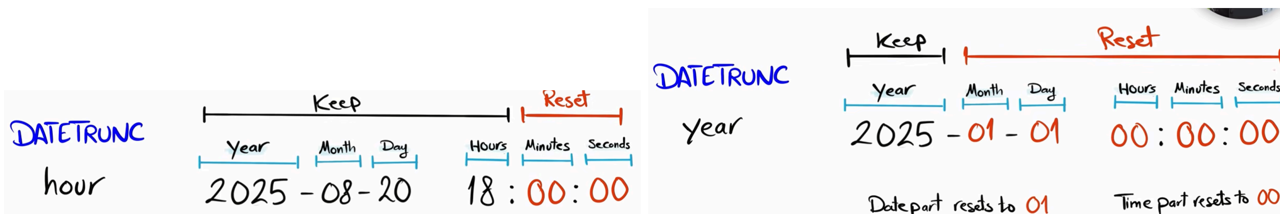
DATENAME(weekday, CreationTime) AS Weekday_dn,
DATENAME(day, CreationTime) AS Day_dn,
DATENAME(year, CreationTime) AS Year_dn

Month_dn	Weekday_dn	Day_dn	Year_dn
January	Wednesday	1	2025
January	Sunday	5	2025
January	Friday	10	2025
January	Monday	20	2025
February	Saturday	1	2025
February	Thursday	6	2025
February	Sunday	16	2025
February	Tuesday	18	2025

8 DATETRUNC get the start of the month

DATETRUNC(year, CreationTime) AS Year_dt,
DATETRUNC(day, CreationTime) AS Day_dt,
DATETRUNC(hour, CreationTime) AS Minute_dt,

Year_dt	Day_dt	Minute_dt
2025-01-01 00:00:00.0000000	2025-01-01 00:00:00.0000000	2025-01-01 12:00:00.0000000
2025-01-01 00:00:00.0000000	2025-01-05 00:00:00.0000000	2025-01-05 23:00:00.0000000
2025-01-01 00:00:00.0000000	2025-01-10 00:00:00.0000000	2025-01-10 18:00:00.0000000
2025-01-01 00:00:00.0000000	2025-01-20 00:00:00.0000000	2025-01-20 05:00:00.0000000



```sql
*/
SELECT
    DATETRUNC(month, CreationTime) AS Creation,
    COUNT(*) AS OrderCount
FROM Sales.Orders
GROUP BY DATETRUNC(month, CreationTime);
```

100 % No issues found

Results Messages

	Creation	OrderCount
1	2025-01-01 00:00:00.0000000	4
2	2025-02-01 00:00:00.0000000	4
3	2025-03-01 00:00:00.0000000	2

9 FORMAT

2025-08-20
18:55:45

Date & Time
Format Speicifiers

FORMAT

Format	Description	Result
D	Full day name	
d	Day of the month	8/20/2025
dd	Day of the month (two-digit)	20
ddd	Abbreviated day name	Wed
dddd	Full day name	Wednesday
M	Month number	44044
MM	Month number (two-digit)	8
MMM	Abbreviated month name	Aug
MMMM	Full month name	August
yy	Year (two-digit)	25
yyyy	Year (four-digit)	2025
hh	Hour (12-hour format, two-digit)	06
HH	Hour (24-hour format, two-digit)	18
m	Minutes	August 20
mm	Minutes (two-digit)	55
s	Seconds	2025-08-20T18:55:45
ss	Seconds (two-digit)	45
f	Fractional seconds (one digit)	Wednesday, August 20, 2025 6:55 PM
ff	Fractional seconds (two digits)	00
fff	Fractional seconds (three digits)	000
tt	AM/PM designator	PM

2025-08-20
18:55:45

Number Format Speicifiers

FORMAT

Format	Description	Query	Result
N	Numeric default	SELECT FORMAT(1234.56, 'N')	1,234.56
P	Percentage	SELECT FORMAT(1234.56, 'P')	123,456.00 %
C	Currency	SELECT FORMAT(1234.56, 'C')	$1,234.56
E	Scientific notation	SELECT FORMAT(1234.56, 'E')	1,23E+09
F	Fixed-point	SELECT FORMAT(1234.56, 'F')	1234.56
N0	Numeric no decimals	SELECT FORMAT(1234.56, 'N0')	1,235
N1	Numeric one decimal	SELECT FORMAT(1234.56, 'N1')	1,234.6
N2	Numeric two decimals	SELECT FORMAT(1234.56, 'N2')	1,234.56
N , de_DE	Numeric (German)	SELECT FORMAT(1234.56, 'N', 'de-DE')	1.234,56
N, en_US	Numeric (US)	SELECT FORMAT(1234.56, 'N', 'en-US')	1,234.56

```sql
SELECT
    OrderID,
    CreationTime,
    FORMAT(CreationTime, 'MM-dd-yyyy') AS USA_Format,
    FORMAT(CreationTime, 'dd-MM-yyyy') AS EURO_Format,
    FORMAT(CreationTime, 'dd') AS dd,
    FORMAT(CreationTime, 'ddd') AS ddd,
    FORMAT(CreationTime, 'dddd') AS dddd,
    FORMAT(CreationTime, 'MM') AS MM,
    FORMAT(CreationTime, 'MMM') AS MMM,
    FORMAT(CreationTime, 'MMMM') AS MMMM
FROM Sales.Orders;
```

No issues found

Results | Messages

OrderID	CreationTime	USA_Format	EURO_Format	dd	ddd	dddd	MM	MMM	MMMM
1	2025-01-01 12:34:56.0000000	01-01-2025	01-01-2025	01	Wed	Wednesday	01	Jan	January
2	2025-01-05 23:22:04.0000000	01-05-2025	05-01-2025	05	Sun	Sunday	01	Jan	January
3	2025-01-10 18:24:08.0000000	01-10-2025	10-01-2025	10	Fri	Friday	01	Jan	January
4	2025-01-20 05:50:33.0000000	01-20-2025	20-01-2025	20	Mon	Monday	01	Jan	January
5	2025-02-01 14:02:41.0000000	02-01-2025	01-02-2025	01	Sat	Saturday	02	Feb	February
6	2025-02-06 15:34:57.0000000	02-06-2025	06-02-2025	06	Thu	Thursday	02	Feb	February
7	2025-02-16 06:22:01.0000000	02-16-2025	16-02-2025	16	Sun	Sunday	02	Feb	February
8	2025-02-18 10:45:22.0000000	02-18-2025	18-02-2025	18	Tue	Tuesday	02	Feb	February

```sql
SELECT
OrderID,
CreationTime,
'Day ' + FORMAT(CreationTime, 'ddd MMM') +
' Q'+ DATENAME(quarter, CreationTime) + ' ' +
FORMAT(CreationTime, 'yyyy hh:mm:ss tt') AS CustomeFormat
FROM Sales.Orders
```

OrderID	CreationTime	CustomeFormat
1	2025-01-01 12:34:56.0000000	Day Wed Jan Q1 2025 12:34:56 PM
2	2025-01-05 23:22:04.0000000	Day Sun Jan Q1 2025 11:22:04 PM

10 CONVERT

```sql
SELECT
CreationTime,
CONVERT(DATE, CreationTime) AS [Datetime to Date CONVERT],
CONVERT(VARCHAR, CreationTime, 32) AS [USA Std. Style:32],
CONVERT(VARCHAR, CreationTime, 34) AS [EURO Std. Style:34]
FROM Sales.Orders
```

11 CAST

Syntax

```sql
CAST(value AS data_type)
```

```sql
SELECT
    CAST('123' AS INT) AS [String to Int],
    CAST(123 AS VARCHAR) AS [Int to String],
    CAST('2025-08-20' AS DATETIME) AS [String to DateTIME],
    CAST('2025-08-20' AS DATETIME2) AS [String to Datetime2],
    CreationTime,
    CAST(CreationTime AS DATE) AS [Datetime to Date]
```

No issues found

Results | Messages

String to Int	Int to String	String to DateTIME	String to Datetime2	CreationTime	Datetime to Date
123	123	2025-08-20 00:00:00.000	2025-08-20 00:00:00.0000000	2025-01-01 12:34:56.0000000	2025-01-01
123	123	2025-08-20 00:00:00.000	2025-08-20 00:00:00.0000000	2025-01-05 23:22:04.0000000	2025-01-05

	CASTING	FORMATING
CAST	Any Type to Any Type	X No Formating
CONVERT	Any Type to Any Type	Formates only Date & Time
FORMAT	Any Type to Only String	Formates < Date & Time, Numbers

- CAST/CONVERT = “转换数据类型”(CONVERT 额外支持日期/时间的 style)。
- FORMAT = “把值格式化成字符串(可控样式/本地化)”，但成本高。

12 DATEADD / DATEDIFF

```
*/
SELECT
    OrderID,
    OrderDate,
    DATEADD(day, -10, OrderDate) AS TenDaysBefore,
    DATEADD(month, 3, OrderDate) AS ThreeMonthsLater,
    DATEADD(year, 2, OrderDate) AS TwoYearsLater
FROM Sales.Orders;
```

```
-- Time Gap Analysis
-- Find the number of days between each order and the previous order
SELECT
OrderID,
OrderDate CurrentOrderDate,
LAG(OrderDate) OVER (ORDER BY OrderDate) PreviousOrderDate,
DATEDIFF(day, LAG(OrderDate) OVER (ORDER BY OrderDate), OrderDate)NrOfDays
FROM Sales.Orders
```

13 ISDATE

SQL Server 的 DATE 类型不接受 year = 0000，合法范围是 0001-01-01 到 9999-12-31。

```
select
orderdate,
isdate(orderdate),
case when isdate(orderdate) =1 then cast (orderdate as date)
     else '0000-01-01'
end newordereate

from
(
    SELECT '2025-08-20' AS OrderDate UNION
    SELECT '2025-08-21' UNION
    SELECT '2025-08-23' UNION
    SELECT '2025-08'
  )t
```

4. 06:59:06 NULL Functions before data aggregations.

WHAT is NULL?

NULL means nothing, unknown!

NULL is not equal to anything!

- NULL is not zero
- NULL is not empty string
- NULL is not blank space

ISNULL	COALESCE
Limited to two values	Unlimited
Fast	Slow
SQL Server → ISNULL Oracle → NVL MySQL → IFNULL	Available in All Databases

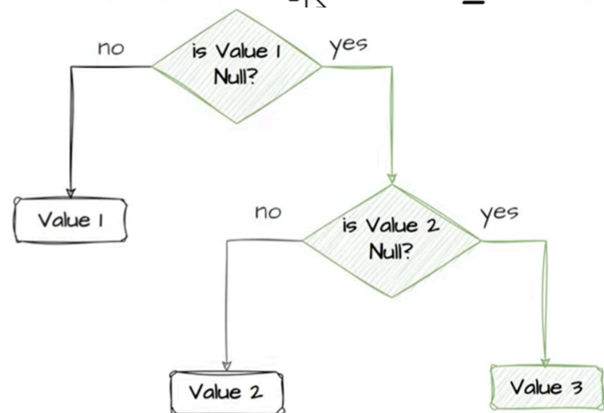
SYNTAX

```
COALESCE(value1, value2, value3)
```

```
COALESCE(ShippingAddress, BillingAddress, 'N/A')
```

OrderID	Shipment Address	Billing Address	COALESCE
1	A	B	A
2	NULL	C	C
3	NULL	NULL	N/A

```
ISNULL(value, replacement_value)
```



/* TASK 3:
  Sort the customers from lowest to highest scores,
  with NULL values appearing last. Null+ any is equal to Null 情况 A(你第一个写法)：
ORDER BY CASE WHEN Score IS NULL THEN 1 ELSE 0 END, Score

第一排序键是 CASE：对非 NULL 返回 0、对 NULL 返回 1 → 非 NULL (0) 会排在前，NULL (1) 在后。
在非 NULL 组内部再按 Score 排序（5, 10）。
结果顺序：C(5), A(10), B(NULL)。

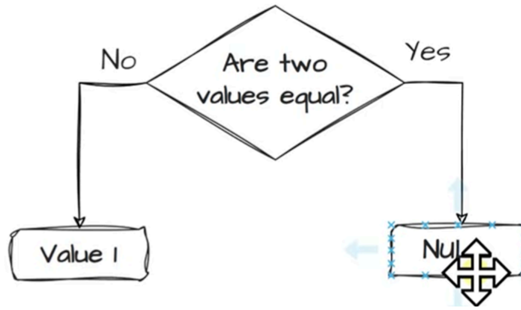
总结一句话：把 Score 放在第一个排序键会让 NULL 因为其自身排序特性（默认视为最小）出现在前面；
把一个把 NULL/非NULL 标识的 CASE 放在第一个排序键，则可以控制 NULL 是在前还是在后。
*/

```
SELECT
   CustomerID,
   Score,CASE WHEN Score IS NULL THEN 1 ELSE 0 END flag
FROM Sales.Customers
order by CASE WHEN Score IS NULL THEN 1 ELSE 0 END,Score
```

SYNTAX

```
NULLIF(value1, value2)
```

```
NULLIF(Price, -1)
```



NULLIF - DIVISION BY ZERO 相同的值 返回null

where

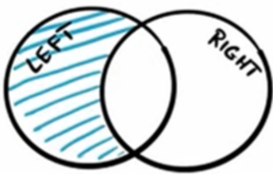
Syntax

```
Value IS NULL
```

```
Value IS NOT NULL
```

left antijoin -- is null

Left Anti Join



All Rows from Left without matching rows (Left Join + IS NULL)

	NULL	Empty String	Blank Space
Representation	NULL	''	' '
Meaning	Unknown	Known, Empty Value	Known, Space Value
Data Type	Special Marker	String (0)	String (1 or more)
Storage	Very minimal	occupies memory	occupies memory (each space)
Performance	Best	Fast	Slow
Comparison	IS NULL	= ''	= ' '

data policy --handling null

policy 2. before inserting into DB
policy 3. before presentation.

```
TRIM(Category) Policy1,
NULLIF(TRIM(Category), '') Policy2,
COALESCE(NULLIF(TRIM(Category), '') , 'unknown') Policy3
```

#3 DATA POLICY

Use the default value 'unknown' and avoid using nulls, empty strings, and blank spaces.

#2 DATA POLICY

Only use NULLS and avoid using empty strings and blank spaces

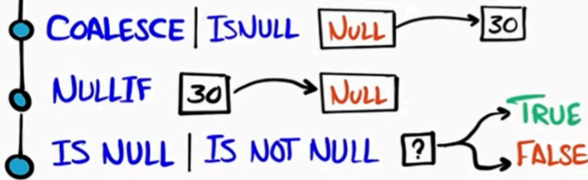
#1 DATA POLICY

Only use NULLs and empty strings, but avoid blank spaces.

NULL Functions

- Nulls special markers means missing value.
- Using Nulls can optimize storage and performance.

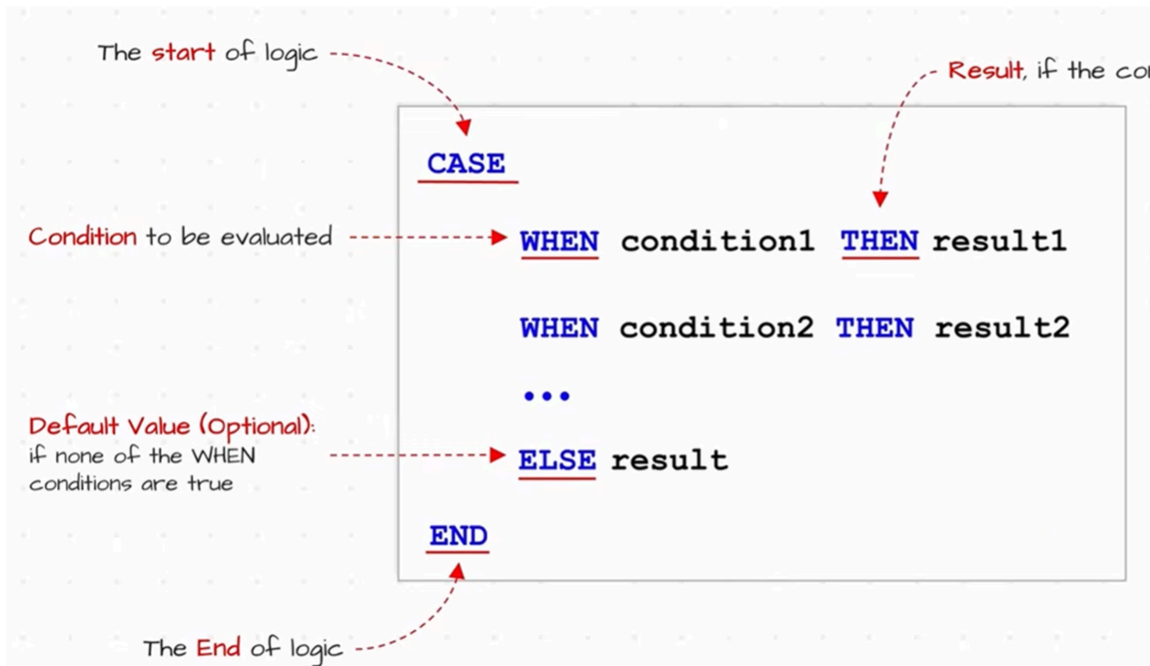
Functions



USE CASES

- Handle Nulls - Data Aggregation
- Handle Nulls - Mathematical operations
- Handle Nulls - Joining Tables
- Handle Nulls - Sorting Data
- Finding unmatched data - Left Anti Join
- Data Policies → Nulls / Default Value

5. 08:07:50 Case Statement

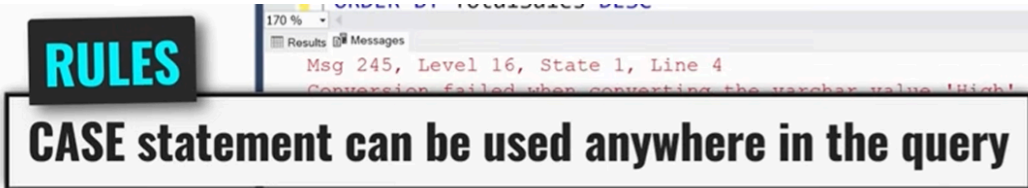


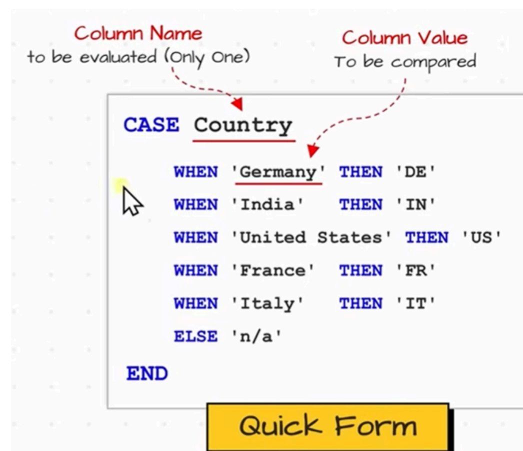
* NoteWindow aggregation and group by will generate error

```
select category, sum(sales) over(partition by category) as S
from
(SELECT
        OrderID,
        Sales,
        CASE
            WHEN Sales > 50 THEN 'High'
            WHEN Sales > 20 THEN 'Medium'
            ELSE 'Low'
        END AS Category
from sales.Orders)r
group by Category
```

1 0 Ln: 7, Ch

```
; 8120, Level 16, State 1, Line 1
.umn 'r.Sales' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

npletion time: 2026-01-08T11:00:06.3900355+08:00
```



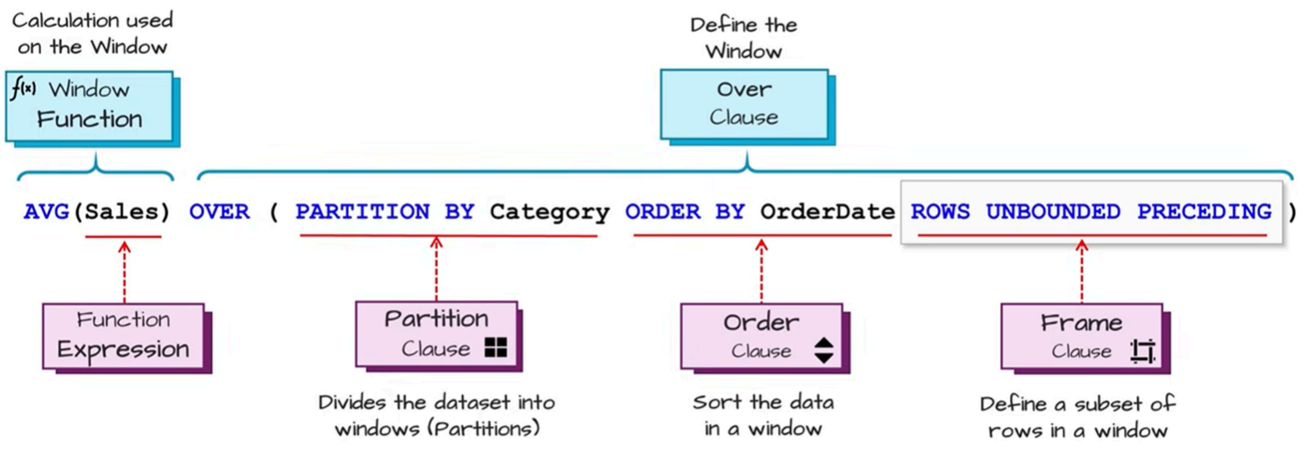



```
/* TASK 5:
   Count how many orders each customer made with sales greater than 30
*/
SELECT
    CustomerID,
    SUM(
        CASE
            WHEN Sales > 30 THEN 1
            ELSE 0
        END
    ) AS TotalOrdersHighSales,
    COUNT(*) AS TotalOrders
FROM Sales.Orders
GROUP BY CustomerID;
```

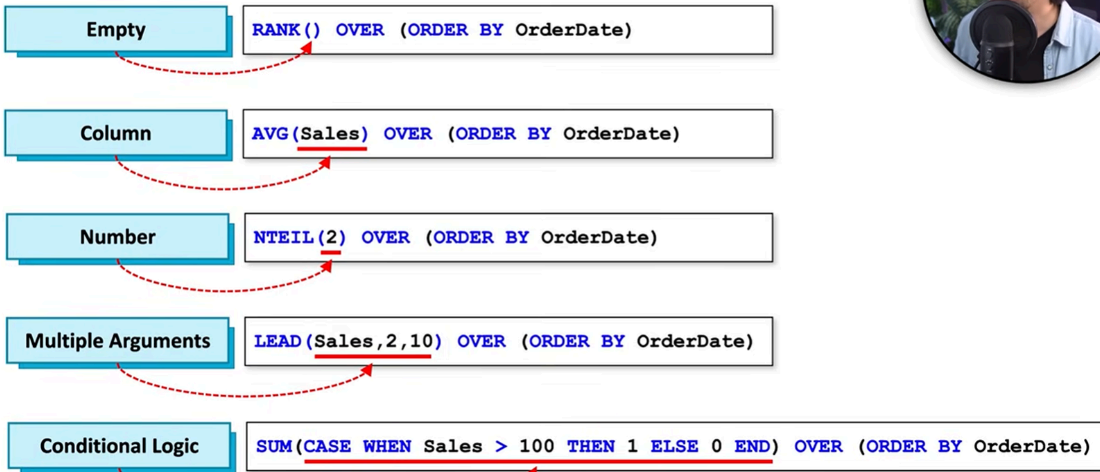
08:43:36 Aggregate Functions

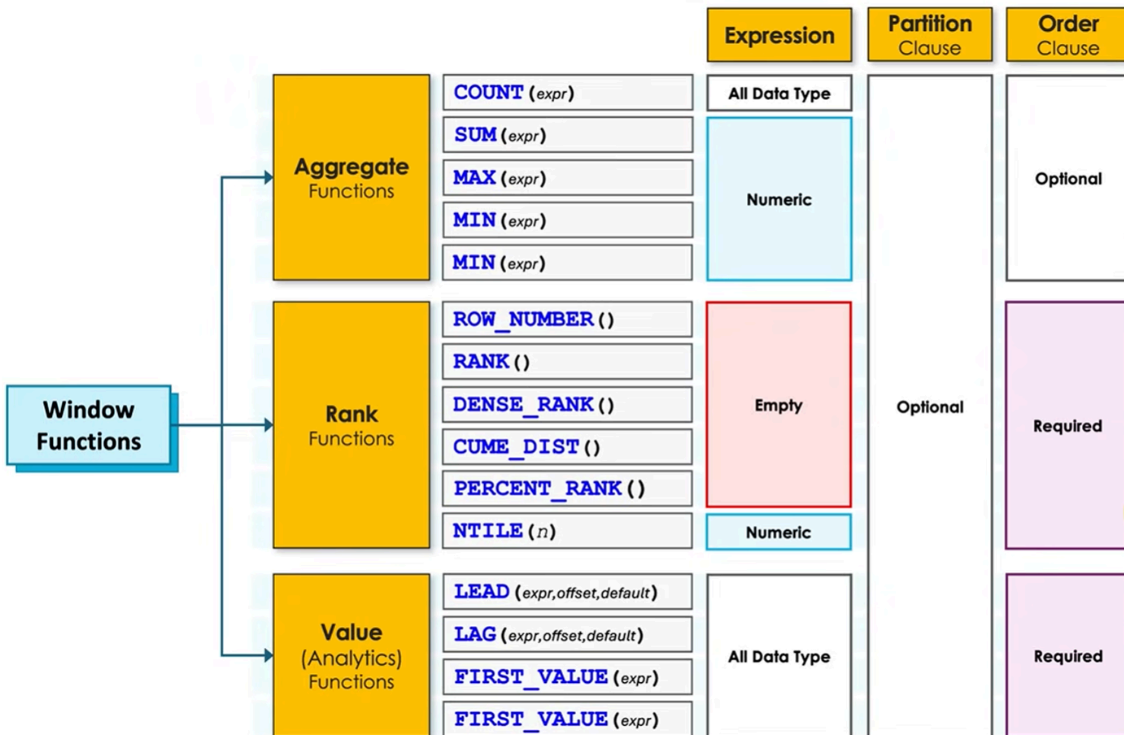
```
1. Basic Aggregate Functions
   - COUNT
   - SUM
   - AVG
   - MAX
   - MIN
2. Grouped Aggregations
   - GROUP BY
```

6. 08:50:11 Window Functions Basics



7.





OVER CLAUSE

Tells SQL that the function used is a window function

It defines a window or subset of data

SUM(Sales) over(partition by productid,orderstatus) Total_Sales

Noted error

```
0          SUM(Sales) OVER (PARTITION BY OrderStatus) AS Total_Sales
1      FROM Sales.Orders
2      WHERE SUM(Sales) OVER (PARTITION BY OrderStatus) > 100;  -- Invalid: window function in WHERE clause
```

No issues found   Ln: 202,

Messages

```
Msg 4108, Level 15, State 1, Line 202
Windowed functions can only appear in the SELECT or ORDER BY clauses.
```

```
    sum(Sales) over() totalsales,
    SUM(Sales) over(partition by productid,orderstatus order by orderid) Total_Sales,
    rank() over(order by SUM(Sales) over(partition by productid,orderstatus order by orderid) DESC) ranksales
FROM Sales.Orders
```

No issues found

Messages

```
Msg 4109, Level 15, State 1, Line 5
Windowed functions cannot be used in the context of another windowed function or aggregate.
```

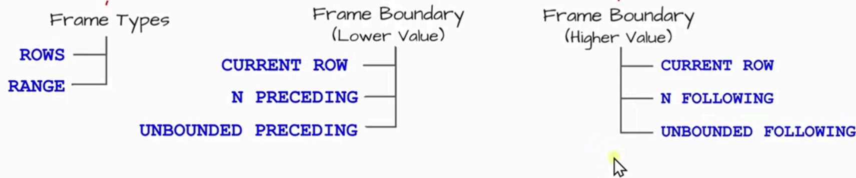
granularity 粒度 aggregation 聚合

GROUP BY LIMITS

Can't do aggregations and provide details at same time

Frame Clause | Syntax

```
AVG(Sales) OVER (PARTITION BY Category ORDER BY OrderDate
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
```



Rules

- Frame Clause can only be used together with order by clause.
- Lower Value must be BEFORE the higher Value.

Rules cannot use in where

Components Window Functions + Window Definition OVER

Define Subset FRAME

Rules

- Nesting is not allowed!
- Window Can be used only in SELECT and ORDER BY
- SQL executes window AFTER filtering data using WHERE

8. 09:47:00 Window Aggregate

WINDOW AGGREGATE FUNCTIONS

Aggregate set of Values and return a single aggregated Value

Rules

- Expressions
  - Numbers (All Functions)
  - Any Data Type – COUNT()
- All Clouses are Optional

Use Cases

- Overall Analysis
- Total Per Groups Analysis
- Part-to-Whole Analysis
- Comparison Analysis
  - Average
  - Extreme: Highest/Lowest
- Identify Duplicates
- Outlier Detection
- Running Total
- Rolling Total
- Moving Average

56 / 1:05:48:27

9.

		Expression	Partition Clause	Order Clause	Frame Clause
Aggregate Functions	COUNT (expr)	All Data Type	Optional	Optional	Optional
	SUM (expr)	NumericValues			
	AVG (expr)	NumericValues			
	MIN (expr)	NumericValues			
	MAX (expr)	NumericValues			

Aggregate Functions			
	COUNT (expr)	Returns the number of Rows in a window	COUNT(*) OVER (PARTITION BY Porduct)
	SUM (expr)	Returns the sum of values in a window	SUM(Sales) OVER (PARTITION BY Porduct)
	AVG (expr)	Returns the average of values in a window	SUM(Sales) OVER (PARTITION BY Porduct)
	MIN (expr)	Returns the minimum value in a window	SUM(Sales) OVER (PARTITION BY Porduct)
	MAX (expr)	Returns the maximum value in a window	SUM(Sales) OVER (PARTITION BY Porduct)

count (1) same as count(*)

在 `COUNT` 函数中，`NULL` 值会被自动忽略（不会计入统计），这是它的默认行为。如果你希望在 `COUNT` 时避免忽略 `NULL`，可以通过将 `NULL` 替换为一个非空的占位字符串（例如 `'Unknown'`），然后再统计。

可以使用 `COALESCE` 或 `ISNULL` 函数来替换 `NULL`，以下是修改后的代码：

```
SELECT
    *,
    COUNT(1) OVER () AS totalorders,
    COUNT(COALESCE(customerid, 'Unknown')) OVER (PARTITION BY COALESCE(customerid, 'Unknown'))
AS orderbycx
FROM Sales.Customers;
```

check Primary Key

```
-- Check whether the table 'orders' contains any duplicate rows
SELECT
    OrderID,
    COUNT(*) OVER (PARTITION BY OrderID) CheckPK
FROM Sales.Orders
```

EXPECTATION

Maximum number of rows for each window (ID) = 1

COUNT | USE CASES

#1 Overall Analysis

#2 Category Analysis

#3 Quality Checks: Identify NULLs

#4 Quality Checks: Identify Duplicates

Running Total

```
SUM(Sales) OVER( ORDER BY Month)
```

Default

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Rolling Total

```
SUM(Sales) OVER( ORDER BY MONTH
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
```

10. 10:53:09 Window Ranking

Rank Functions	Expression	Partition Clause	Order Clause	Frame Clause
ROW_NUMBER()	Empty	Optional	Required	Not allowed
RANK()	Empty	Optional	Required	Not allowed
DENSE_RANK()	Empty	Optional	Required	Not allowed
CUME_DIST()	Empty	Optional	Required	Not allowed
PERCENT_RANK()	Empty	Optional	Required	Not allowed
NTILE(n)	Number	Optional	Required	Not allowed

```
25    SELECT
26        OrderID,
27        ProductID,
28        Sales,
29        ROW_NUMBER() OVER (ORDER BY Sales DESC) AS SalesRank_Row,
30        RANK() OVER (ORDER BY Sales DESC) AS SalesRank_Rank,
31        DENSE_RANK() OVER (ORDER BY Sales DESC) AS SalesRank_Dense
32    FROM Sales.Orders;
```

0 % No issues found

Results Messages

OrderID	ProductID	Sales	SalesRank_Row	SalesRank_Rank	SalesRank_Dense
8	101	90	1	1	1
4	105	60	2	2	2
10	102	60	3	2	2
6	104	50	4	4	3
7	102	30	5	5	4
5	104	25	6	6	5
9	101	20	7	7	6
3	101	20	8	7	6

Assign Unique IDs to the Rows of the 'Order Archive'

Use Case | Identify Duplicates:

Identify Duplicate Rows in 'Order Archive' and return a clean result without any duplicates

***/**

```
SELECT *
FROM (
   SELECT
      ROW_NUMBER() OVER (PARTITION BY OrderID ORDER BY CreationTime DESC) AS rn,
      *
   FROM Sales.OrdersArchive
) AS UniqueOrdersArchive
WHERE rn = 1;
```

```sql
NTILE(2) OVER (ORDER BY Sales DESC)
```

Sales	NTILE
100	

$$\text{Bucket Size} = \frac{\text{Number of Rows}}{\text{Number of Buckets}}$$

larger group comes first

```sql
CUME_DIST() OVER (ORDER BY Sales DESC)
```

Sales	DIST
100	0,2
80	0,6
80	0,6
50	0,8

$$\text{CUME_DIST} = \frac{\text{Position Nr}}{\text{Number of Rows}}$$

$$\text{CUME_DIST} = \frac{4}{5}$$

CUME_DIST

$$\frac{\text{Position Nr}}{\text{Number of Rows}}$$

Inclusive
(The current row is included)

PERCENT_RANK

$$\frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

Exclusive
(The current row is excluded)

distributiuon

```sql
PERCENT_RANK() OVER (ORDER BY Sales DESC)
```

Sales	Per
100	0
80	0,25
80	0,25
50	0,75

$$\text{Percent_Rank} = \frac{\text{Position Nr} - 1}{\text{Number of Rows} - 1}$$

$$\text{Percent_Rank} = \frac{3}{4}$$

relevant position of each rows

特性	CUME_DIST()	PERCENT_RANK()
公式	$\frac{\text{当前行及之前的行数}}{\text{总行数}}$	$\frac{\text{当前行排名} - 1}{\text{总行数} - 1}$
是否包含当前行	是（累积分布，包括当前行）	否（基于排名，当前行不计入）
值范围	$(0, 1]$	$[0, 1]$
应用场景	用于累积分布分析，反映当前行的分布范围	用于排名分析，反映当前行的相对排名位置

选择哪一个?

- **选择 CUME_DIST()**：如果你想知道某个值的 **累积分布比例**（例如，判断一个销售额是否处于前 80%）。
- **选择 PERCENT_RANK()**：如果你想知道某个值的 **相对排名位置**（例如，计算前 25% 的数据）。

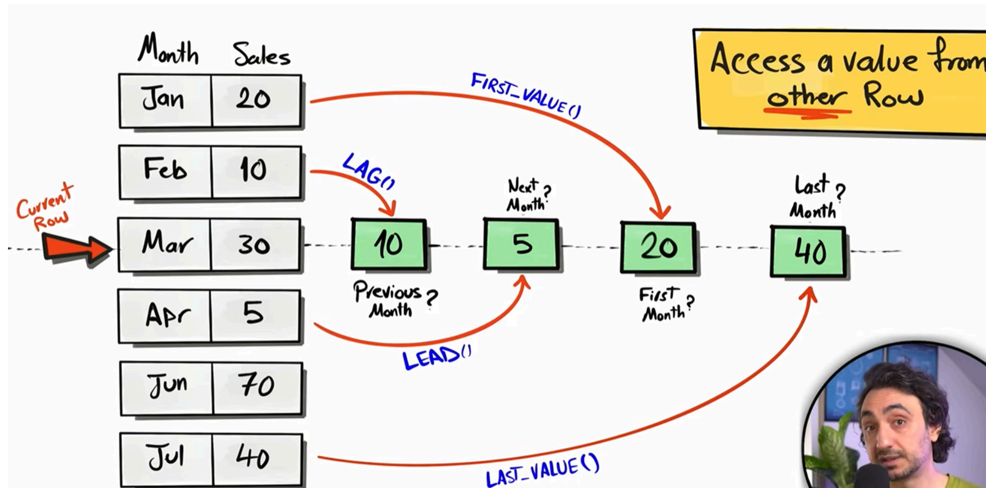
WINDOW RANK FUNCTIONS

Assign a RANK for each row within a window

Rules
- Expression → Empty
- ORDER BY → Required
- FRAME → Not Allowed

Use Cases
- Top N Analysis
- Bottom N Analysis
- Identify & Remove Duplicates
- Assign Unique IDs & Pagination
- Data Segmentation
- Data Distribution Analysis
- Equalizing Load Processing

11. 11:56:05 Window Value



Value (Analytics) Functions	Expression	Partition Clause	Order Clause	Frame Clause
LEAD (expr,offset,default)	All Data Type	Optional	Required	Not allowed
LAG (expr,offset,default)				
FIRST_VALUE (expr)				Optional
LAST_VALUE (expr)				Should be used

Value (Analytics) Functions		
LEAD (expr,offset,default)	Returns the value from a previews row	`LEAD(Sales,2,0) OVER (ORDER BY OrderDate)`
LAG (expr,offset,default)	Returns the value from a subsequent row	`LAG(Sales,2,0) OVER (ORDER BY OrderDate)`
FIRST_VALUE (expr)	Returns the first value in a window	`FIRST_VALUE(Sales) OVER (ORDER BY OrderDate)`
LAST_VALUE (expr)	Returns the last value in a window	`FIRST_VALUE(Sales) OVER (ORDER BY OrderDate)`

```
LEAD(Sales, 2 , 10 ) OVER(PARTITION BY ProductID ORDER BY OrderDate)
```

Expression is required (Any Data Type)

Offset (Optional)
Number of rows forward or backward from current row
default = 1

Default Value (Optional)
Returns default value if next/previous row is not available!
Default = NULL

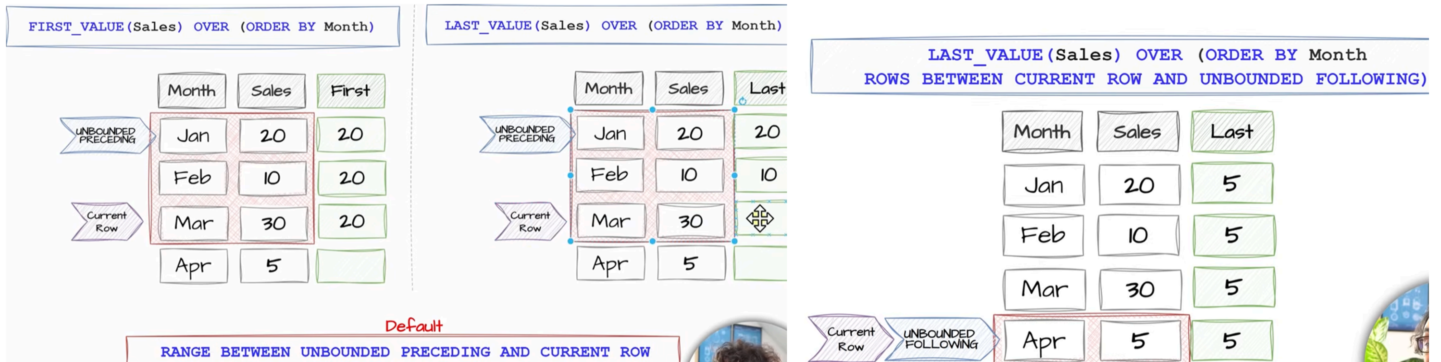
```
    Analyze the Month-over-Month Performance by Finding the Percentage Change in Sales
    Between the Current and Previous Months
*/

SELECT
    *,
    CurrentMonthSales - PreviousMonthSales AS MoM_Change,
    ROUND(
        CAST((CurrentMonthSales - PreviousMonthSales) AS FLOAT)
        / PreviousMonthSales * 100, 1
    ) AS MoM_Perc
FROM (
    SELECT
        MONTH(OrderDate) AS OrderMonth,
        SUM(Sales) AS CurrentMonthSales,
        LAG(SUM(Sales)) OVER (ORDER BY MONTH(OrderDate)) AS PreviousMonthSales
    FROM Sales.Orders
    GROUP BY MONTH(OrderDate)
) AS MonthlySales;
```

No issues found

Results | Messages

OrderMonth	CurrentMonthSales	PreviousMonthSales	MoM_Change	MoM_Perc
1	105	NULL	NULL	NULL
2	195	105	90	85.7
3	80	195	-115	-59

```
/* TASK 2:
    Customer Loyalty Analysis - Rank Customers Based on the Average Days Between Their Or
*/
SELECT
    CustomerID,
    AVG(DaysUntilNextOrder) AS AvgDays,
    RANK() OVER (ORDER BY COALESCE(AVG(DaysUntilNextOrder), 999999)) AS RankAvg
FROM (
    SELECT
        OrderID,
        CustomerID,
        OrderDate AS CurrentOrder,
        LEAD(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS NextOrder,
        DATEDIFF(
            day,
            OrderDate,
            LEAD(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate)
        ) AS DaysUntilNextOrder
    FROM Sales.Orders
) AS CustomerOrdersWithNext
GROUP BY CustomerID;
```




```sql
/* TASK 3:
   Find the Lowest and Highest Sales for Each Product,
   and determine the difference between the current Sales and the lowest Sales for each Product.
*/
SELECT
    OrderID,
    ProductID,
    Sales,
    FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY Sales) AS LowestSales,
    LAST_VALUE(Sales) OVER (
        PARTITION BY ProductID
        ORDER BY Sales
        ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
    ) AS HighestSales,
    Sales - FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY Sales) AS SalesDifference
FROM Sales.Orders;
```

```sql
-- Find the lowest and highest sales for each product
SELECT
    OrderID,
    ProductID,
    Sales,
    FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY S
    LAST_VALUE(Sales)  OVER (PARTITION BY ProductID ORDER BY S
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) HighestS
    FIRST_VALUE(Sales) OVER (PARTITION BY ProductID ORDER BY S
    MIN(Sales) OVER (PARTITION BY ProductID) LowestSales2,
    MAX(Sales) OVER (PARTITION BY ProductID) HighestSales3
FROM Sales.Orders
```

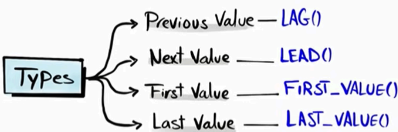
USE CASE

Compare to Extermes

How well a value is performing relative to the extremes

WINDOW VALUE (ANALYTICAL) FUNCTIONS

Allow Access specific Value from another Row

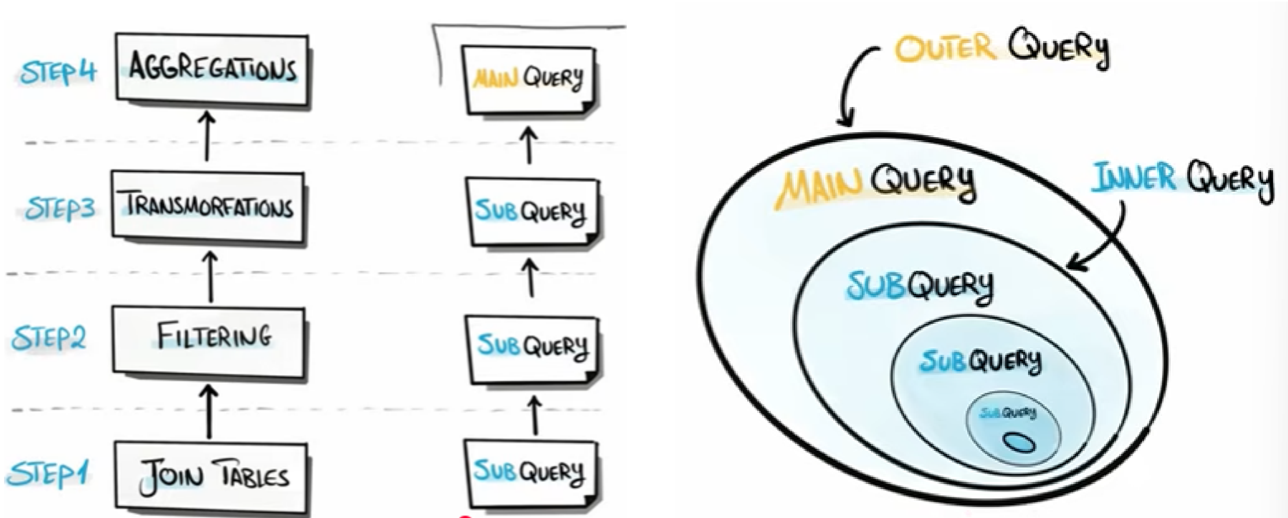


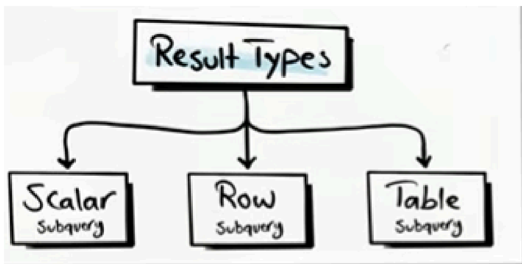
Rules
- Expression — Any Data Type
- ORDER BY — Required
- FRAME — Optional

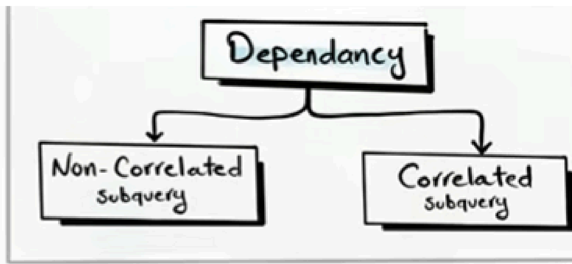
Use Cases
- Time Series Analysis: MoM & YoY
- Time Gaps Analysis: Customer Retention
- Comparision Analysis: Extreme (Highest / Lowest)

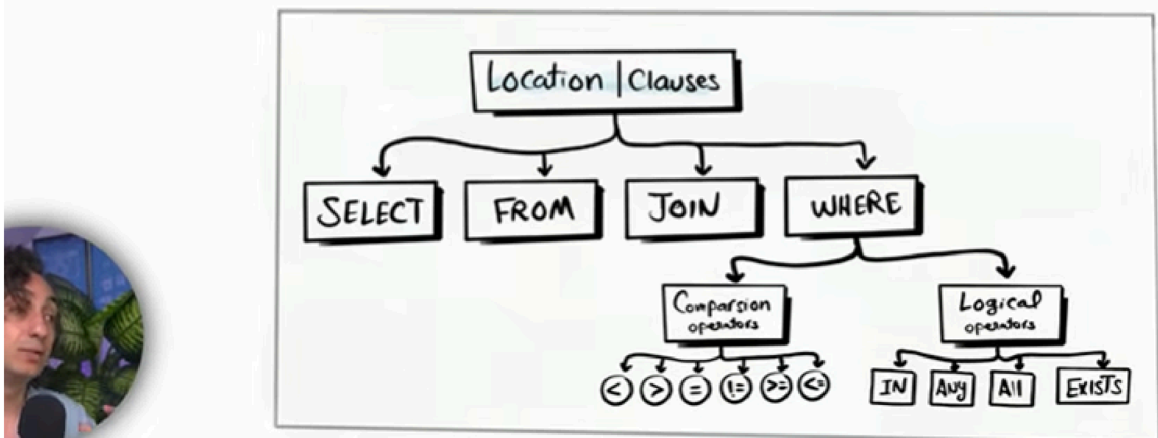
__🔴Advanced Level__

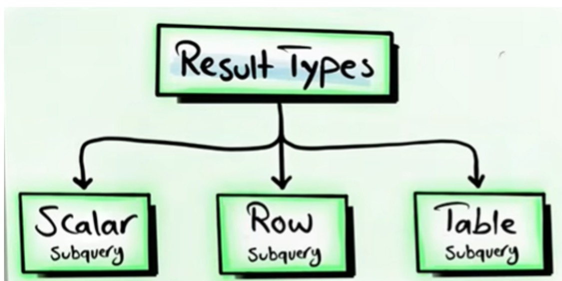
12. 12:58:04 Subqueries can only be used from the main query



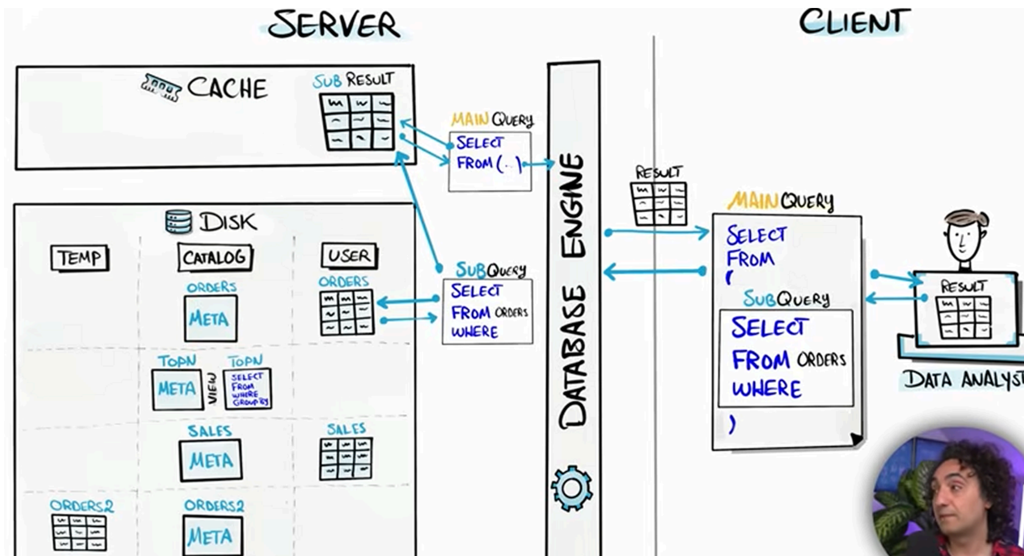


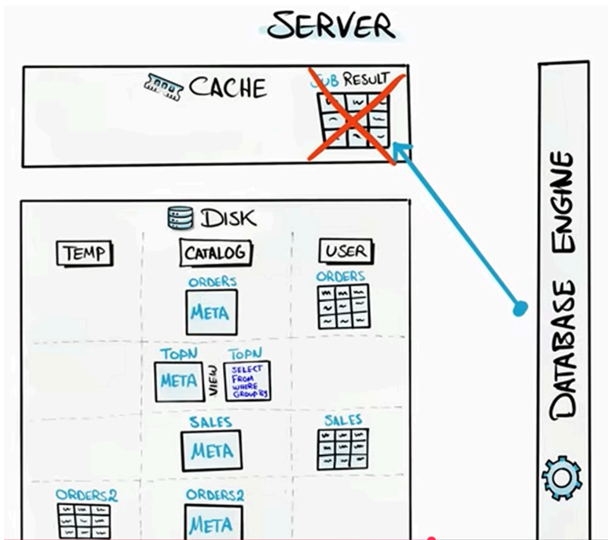






scalar has only one row and one value





clean the cache

Subquery in SELECT Clause

Main Query

```
SELECT
  Column1,
  ( SELECT column FROM table1 WHERE condition ) AS alias
FROM table1
```

Subquery

Rules

Only Scalar Subqueries are allowed to be used

```sql
SELECT c.*, COALESCE(R.total, 0) AS total_orders
FROM sales.Customers c
LEFT JOIN (
    SELECT COUNT(*) AS total, CustomerID
    FROM sales.Orders
    GROUP BY CustomerID
) R
ON c.CustomerID = R.CustomerID;
```

select avg(price) from sales.Products
不可用用**over()** 否则返回多个数字

Subquery in WHERE Clause Comparison Operators

Main Query

```
SELECT column1, column2,...
FROM table1
WHERE column = ( SELECT column FROM table2 WHERE condition )
```

Subquery

Rules

Only Scalar Subqueries are allowed to be used

```sql
SELECT column1, column2,...
FROM   table1
WHERE  column < ALL( SELECT column FROM table1 WHERE condition )
```

```sql
SELECT column1, column2,...
FROM   table1
WHERE  column < ANY( SELECT column FROM table1 WHERE condition )
```

NON-CORRELATED SUBQUERY

A Subquery that can run independtly from the Main Query

CORRELATED SUBQUERY

A Subquery that relays on values from the Main Query

	Non-Correlated Subquery	Correlated Subquery
Definition	Subquery is independent of the main query	Subquery is dependent of the main query
Execution	Executed once and its result is used by the main query Can be executed on its Own	Executed for each row processed by the main query Can't be executed on its Own.
Easy to use	Easier to read	Harder to read and more complex
Performance	Executed only once leads to better Performance	Executed multiple times leads to bad Performance
Usage	Static Comparisons, Filtering with Constants	Row-by-Row Comparisons, Dynamic Filtering

correlated subquery example.

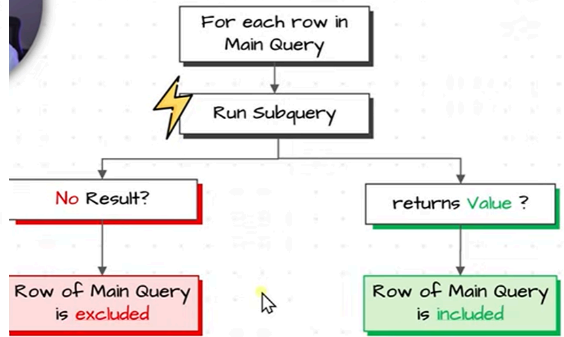
```sql
-- Show all customer details and find the total orders of each customer

-- Main Query
SELECT
*,
(SELECT COUNT(*) FROM Sales.Orders o WHERE o.CustomerID = c.CustomerID) TotalSales
FROM Sales.Customers c
```

Correlated Subquery in WHERE Clause EXISTS Operator

Main Query

```sql
SELECT column1, column2,...
FROM   Table2
WHERE  EXISTS  ( SELECT 1
                 FROM Table1
                 WHERE Table1.ID = Table2.ID )
```

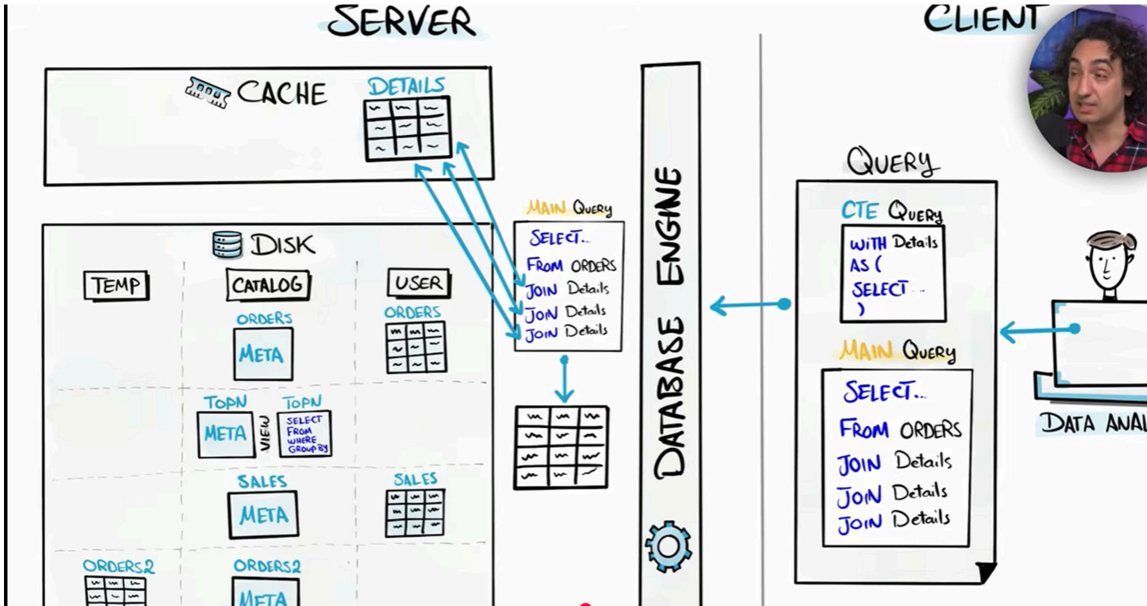
Subquery

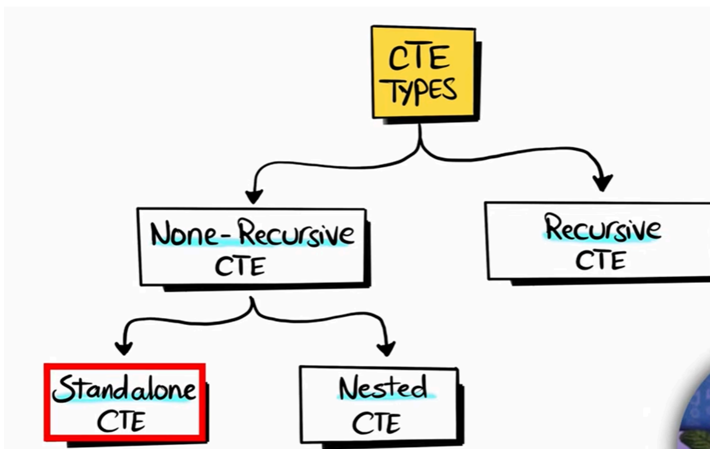


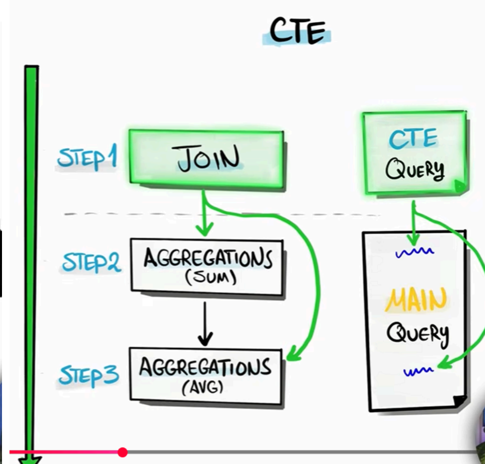
```
select *
from sales.orders o
where exists(
select* from sales.Customers c where Country='Germany'
and o.CustomerID= c.CustomerID
)
```

```
select *
from sales.orders o
where exists(
select 3 from sales.Customers c where Country='Germany'
and o.CustomerID= c.CustomerID
)
```

13. 14:18:08 Common Table Expressions (CTE) same memory cache. speed up



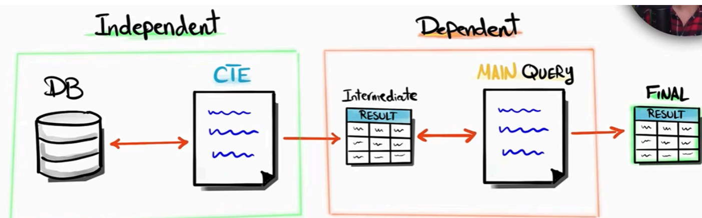




Standalone CTE

Defined and Used independently.

Runs independently as it's self-contained and doesn't rely on other CTEs or queries.



CTE
syntax

```
WITH CTE-Name AS
(
  SELECT ...
  FROM ...
  WHERE ...
)
```

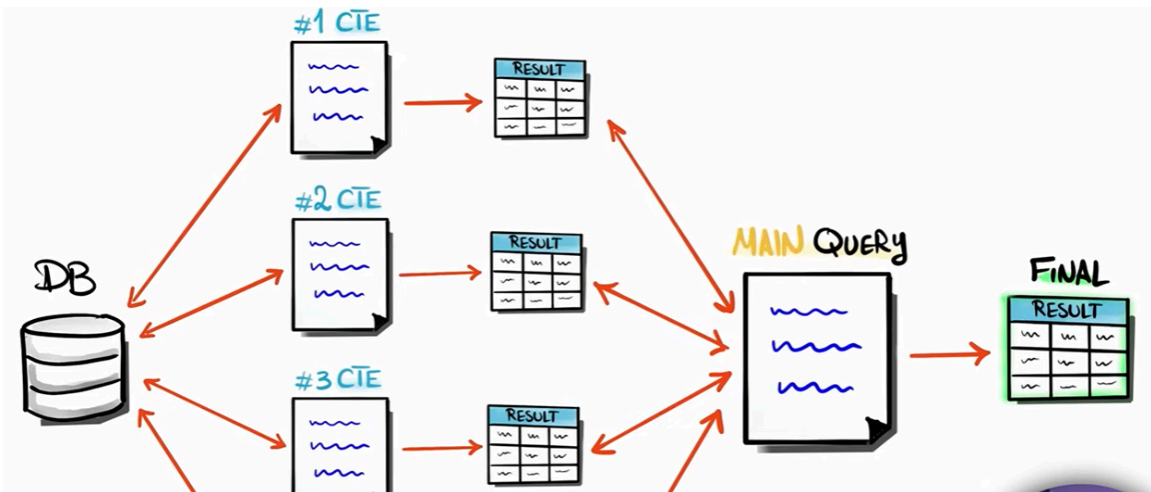
CTE Query
- CTE Definition -

```
SELECT ...
FROM CTE-Name
WHERE ...
```

Main Query
- CTE Usage -

CTE-RULE

You cannot use ORDER BY directly within the CTE



Multiple Standalone CTEs

CTE Query
- CTE Definition -

```
WITH CTE-Name1 AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

CTE Query
- CTE Definition -

```
, CTE-Name2 AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

Main Query
- CTE Usage -

```
SELECT ...
FROM CTE-Name1
 JOIN CTE-Name2
WHERE ...
```

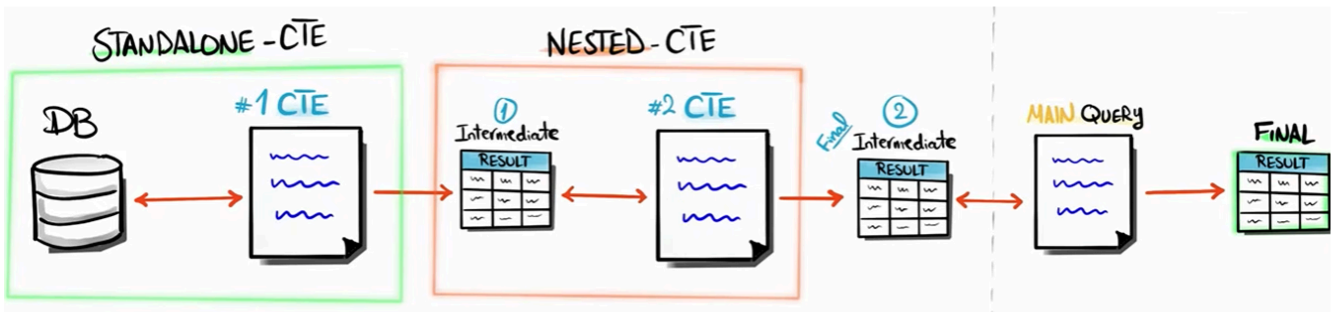
Nested CTE

CTE inside another CTE

A nested CTE uses the result of another CTE, so it can't run independently.

Rethink and refactor your CTEs before starting a new one.

Don't use more than 5 CTEs in one query; otherwise, your code will be hard to understand and maintain.



Non-Recursive CTE

is executed only once without any repetition.

Recursive CTE

Self-referencing query that repeatedly processes data until a specific condition is met.

anchor is executed only once,then recursive add where. break condition

```
Msg 530, Level 16, State 1, Line 2
The statement terminated. The maximum recursion 100 has been exhausted before
```

Recursive CTE syntax

CTE Query - CTE Definition -

```
WITH CTE-Name AS
(
    SELECT ...
    FROM ...
    WHERE ...
    UNION ALL
    SELECT ...
    FROM CTE-Name
    WHERE [Break Condition]
)
```

Anchor Query

Recursive Query

Main Query - CTE Usage -

```
SELECT ...
FROM CTE-Name
WHERE ...
```

CTE

Common Table Expression (CTE) is Temporary, named result set that can be used multiple times within the Query.

Advantages

- Readability: Breaks down Complex Queries into Smaller Pieces.
- Modularity: Pieces are easy to manage, develop, and self-contained.
- Reusability: Reduce Redundancy in Query.
- Recursive: Iterations & Looping in SQL.

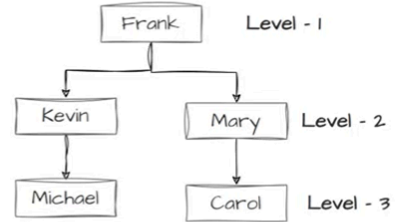
Result of CTE is like Table but can't be used from multiple Queries.

Tip: Don't create more than 5 CTEs in One Query.

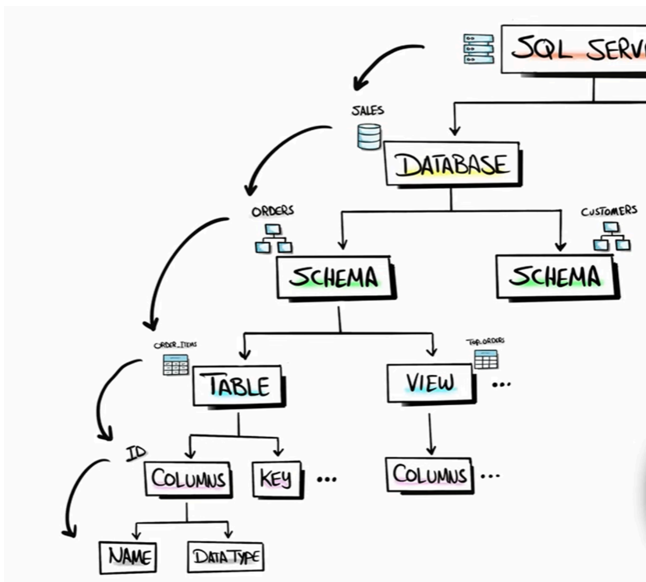
```sql
WITH CTE_Emp_Hierarchy AS
(
    SELECT
        EmployeeID,
        FirstName,
        ManagerID,
        1 AS Level
    FROM Sales.Employees
    WHERE ManagerID IS NULL

    UNION ALL

    SELECT
        e.EmployeeID,
        e.FirstName,
        e.ManagerID,
        Level +1
    FROM Sales.Employees AS e
    INNER JOIN CTE_Emp_Hierarchy ceh
    ON e.ManagerID = ceh.EmployeeID
)

SELECT *
FROM CTE_Emp_Hierarchy
```

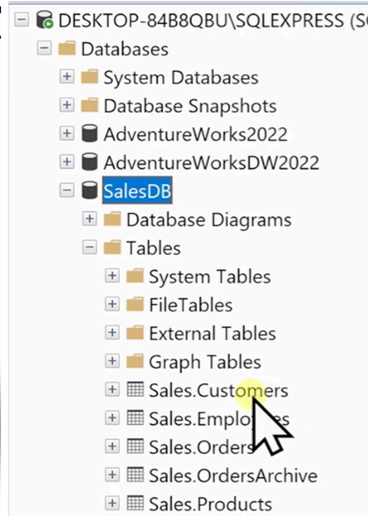
CTE RESULT

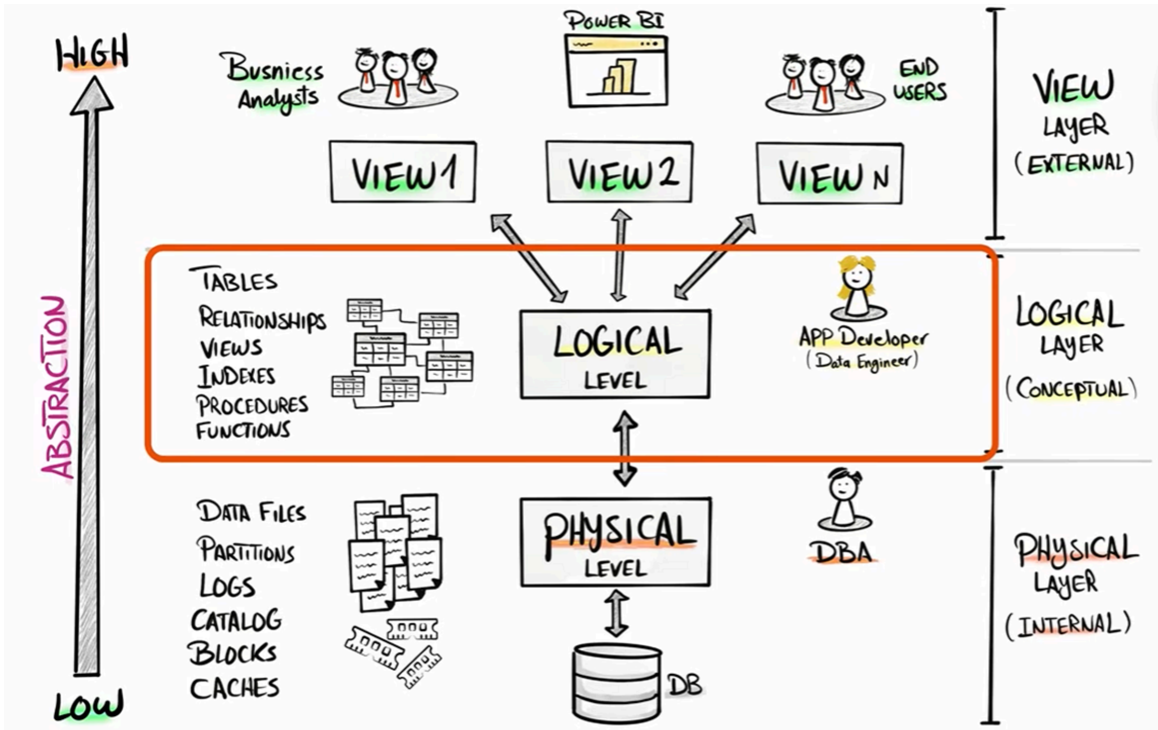
EmployeeID	FirstName	ManagerID	Level
1	Frank	NULL	1
2	Kevin	1	2
3	Mary	1	2
4	Michael	2	3
5	Carol	3	3



14. [15:35:02](#) Views







VIEW	TABLE
No Persistance	Persisted Data
Easy to Maintain	Hard to Maintain
Slow Response	Fast Response
Read	Read / Write

Central Query Logic

Store central, complex query logic in the database

for access by muiltiple queries, reducing project complexity.

VIEWS	CTE
Reduce Redundancy in Multi-Queries	Reduce Redundancy in 1 Query
Improve Reusability in Multi-Queries	Improve Reusability in 1 Query
Persisted logic	Temporary Logic - On the Fly -
Need to Maintain - CREATE/DROP -	No Maintenance - Auto cleanup -

DDL Command

```
CREATE VIEW VIEW-NAME AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

Query

VIEWS

- Virtual Tabel based on result of Query without storing data.
- We use Views to Persist Complex SQL Query in Database.
- Views are better than CTE - improves reusability in multiple Queries.
- Views are better than Tables - Flexible & ease to maintaine.

USE CASES

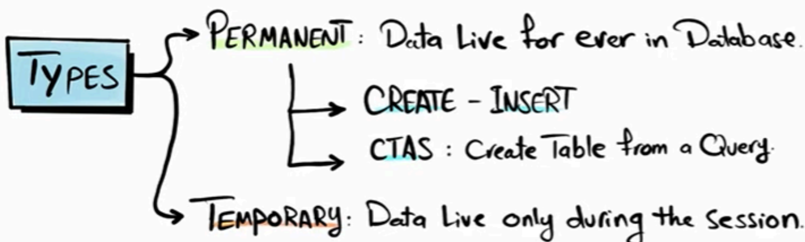
- Store Central Complex Business logic to be reused.
- Hide Complexitity by offering friendly views to users.
- Data Security by hiding sensitive rows & Columns.
- Flexibility & Dynamic
- offer your objects in Multiple Languages.
- Virtual layer (Data Marts) in Data Warehouses.

15. 16:36:40 **CTAS** and Temp Tables

CREATE TABLE AS SELECT

TABLES

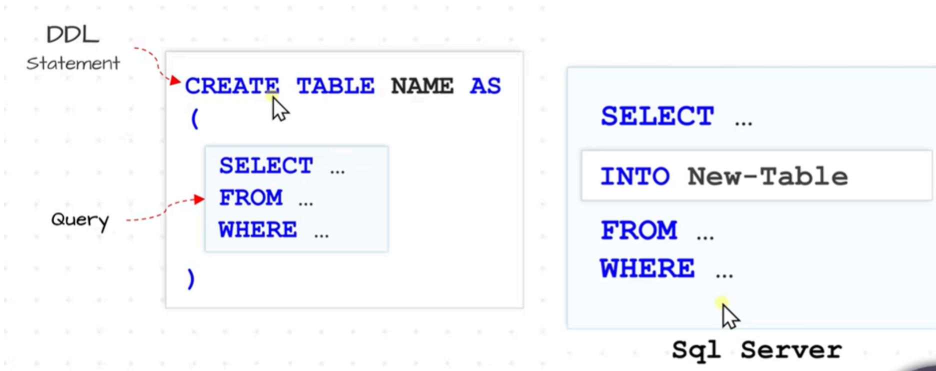
Structured Collection of Data like spreadsheet (Columns & Rows)

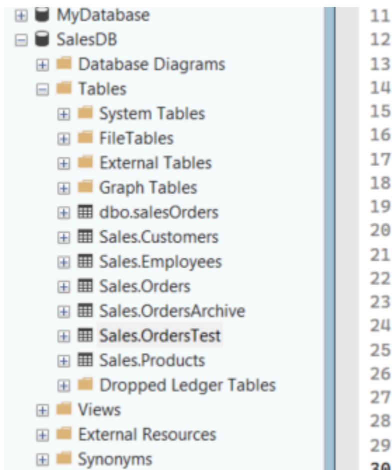


CTAS USE CASES

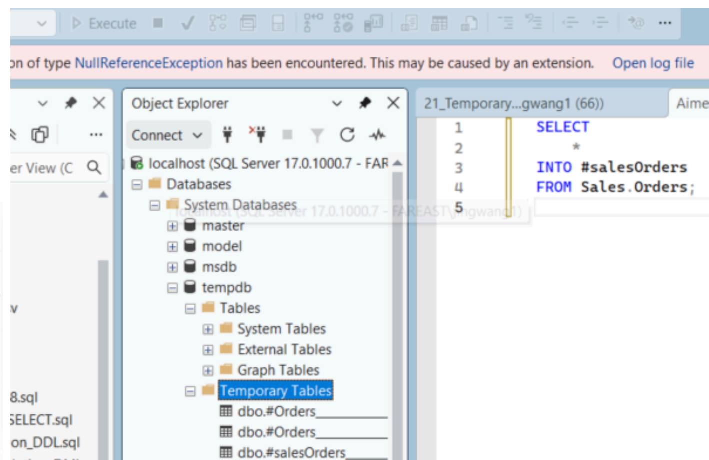
- Optimize Performance : Persist Complex SQL Logic in Table.
- Creating Snapshot : to analyse Bugs and data issues.

Advantage of TEMP Tables Automatic cleanup of data after session ends.






```
11  ================================================
12  SELECT
13      *
14  INTO #Orders
15  FROM Sales.Orders;
16
17  /* ============================================
18     Step 2: Clean Data in Temporary Table
19  ================================================
20  DELETE FROM #Orders
21  WHERE OrderStatus = 'Delivered';
22
23  /* ============================================
24     Step 3: Load Cleaned Data into Permanent T
25  ================================================
26  SELECT
27      *
28  INTO Sales.OrdersTest
29  FROM #Orders;
```

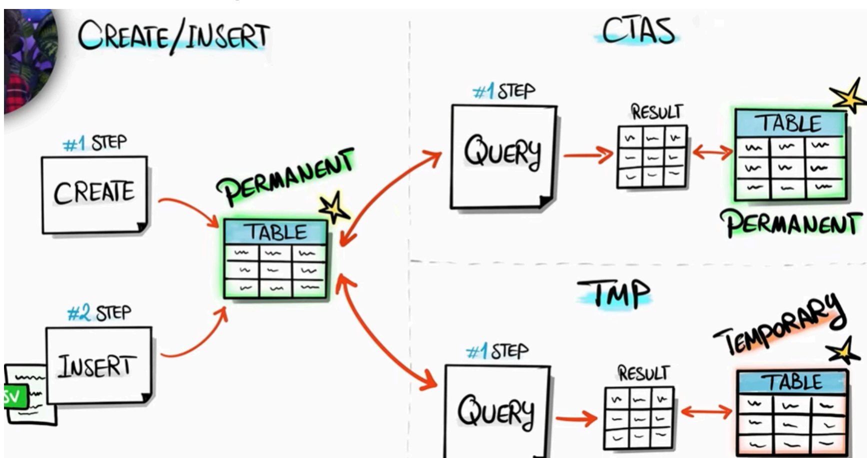


TEMPORARY TABLES

Stores intermediate results in temporary storage within the database during the session.

The database will drop all temporary tables once the session ends.

session then all gone.



16. [17:17:31] Compare Advanced Techniques

	SUBQUERY	CTE	TMP	CTAS	VIEW
STORAGE	MEMORY		DISK		✗ NO STORAGE
LIFE TIME	TEMPORARY			PERMANENT	
WHEN DELETED	END OF QUERY		END OF SESSION	DDL - DROP	
SCOPE	SINGLE - QUERY		MULTI - QUERIES		
REUSABILITY	LIMITED 1 PLACE - 1 QUERY	LIMITED Multi PLACES - 1 QUERY	MEDIUM Multi QUERIES During Session	HIGH MULTI QUERIES	
UP2DATE	✓	✓	✗	✗	✓

复用性：如果想被多次/多用户复用，选 CTAS 或 VIEW；只在一次查询里用就子查询/CTE。

17. [17:27:04] Stored Procedures

PROCEDURE Syntax

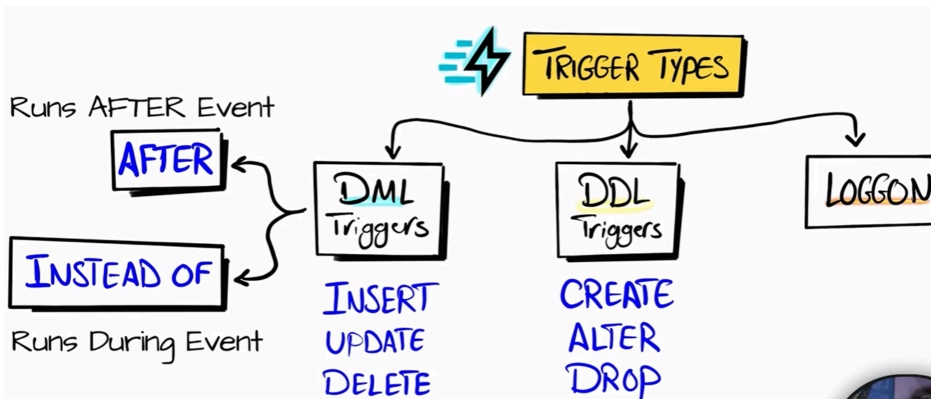
Stored Procedure Definition ----->

```
CREATE PROCEDURE ProcedureName AS
BEGIN


END
```

Stored Procedure Execution (Call) ----->

```
EXEC ProcedureName
```

18. 18:12:58 Triggers




```
        CREATE TRIGGER TriggerName ON TableName
WHEN ----> AFTER INSERT, UPDATE, DELETE
        BEGIN
WHAT ---->     -- SQL STATEMENTS GO HERE
        END
```

Performance

`sys` 架构下的主要内容和作用：

1. **系统视图（System Views）**：
    - 这是最常用的部分。它们提供数据库中各种对象的信息。
    - 例如：
        - `sys.tables`：查询数据库中的所有表。
        - `sys.columns`：查询所有表的列信息。
        - `sys.indexes`：查询所有索引信息。
        - `sys.databases`：查询服务器上的所有数据库。
        - `sys.partitions`：查询分区信息。
        - `sys.objects`：查询数据库中的所有对象（表、视图、存储过程等）。
        - `sys.partition_schemes`：查询分区方案的信息（你正在使用的）。
        - `sys.partition_functions`：查询分区函数的信息（你正在使用的）。
        - `sys.filegroups`：查询文件组的信息（你正在使用的）。
        - `sys.dm_db_stats_properties`：动态管理函数，用于获取统计信息属性（我们之前讨论的）。
    - 这些视图是只读的，你不能直接修改它们。它们是SQL Server底层系统表的一个友好、结构化的展示。

19. 18:23:42 Indexes

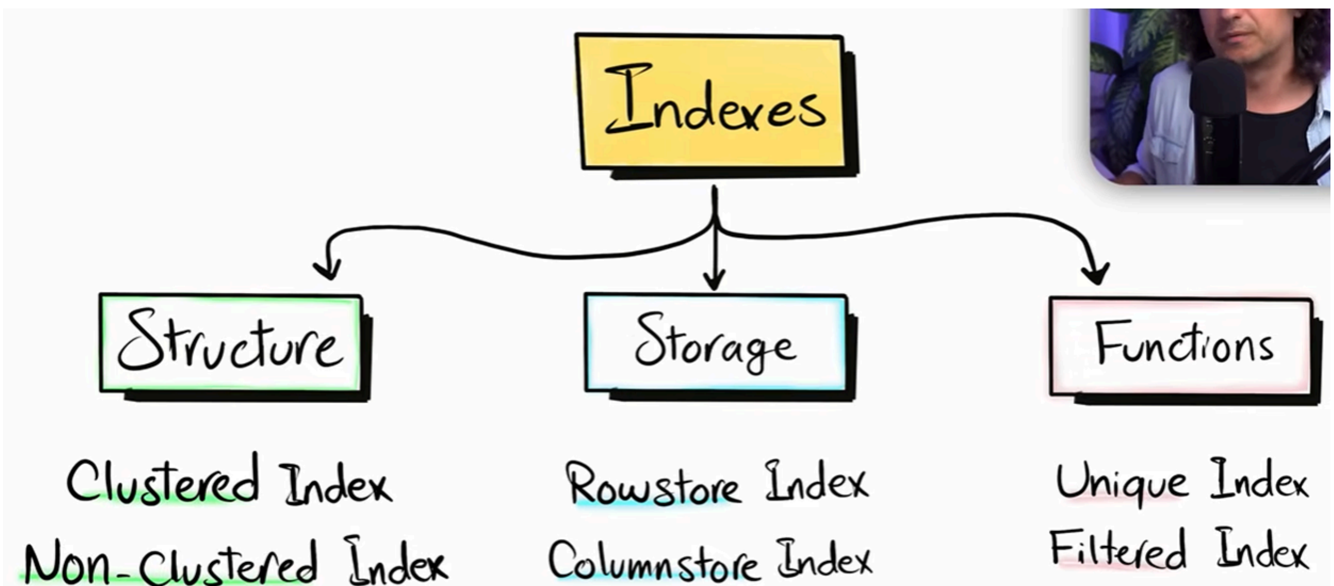
btree

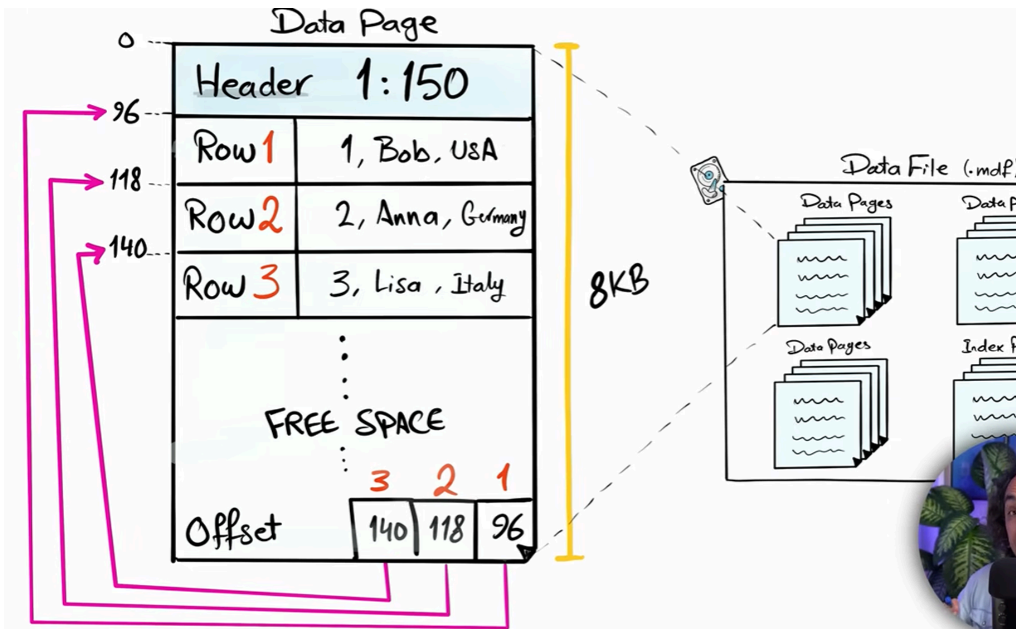
使用 B-Tree 索引加速查询

```sql
-- 单值查询：通过 B-Tree 快速查找特定值
SELECT * FROM Products WHERE Price = 1.5;

-- 范围查询：通过 B-Tree 快速查找范围内的数据
SELECT * FROM Products WHERE Price BETWEEN 0.5 AND 1.5;

-- 排序查询：B-Tree 索引能加速 ORDER BY
SELECT * FROM Products ORDER BY Price;

-- 前缀匹配：使用索引加速 LIKE 查询
SELECT * FROM Products WHERE ProductName LIKE 'A%';
```

SQL Server：执行计划

```sql
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
-- 查询时右键查看实际执行计划
SELECT * FROM Products WHERE Price = 1.5;
```





slow read fast write

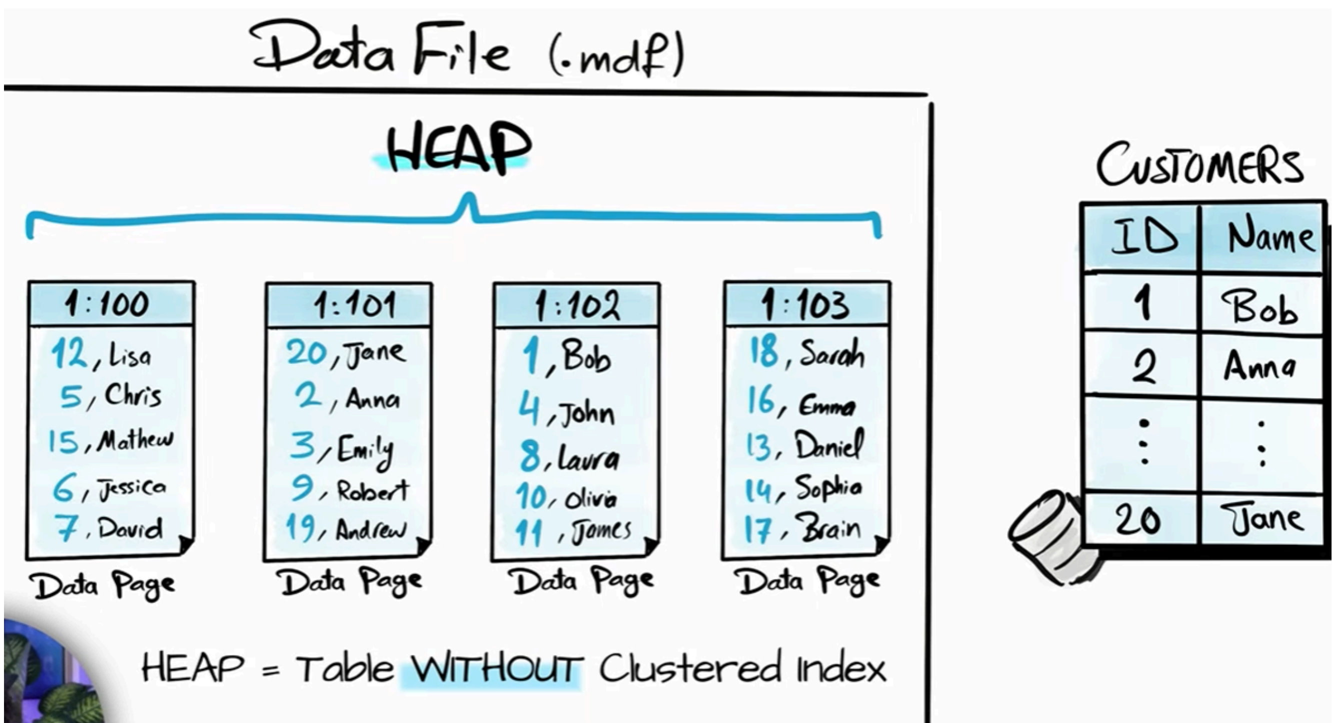


Table Full Scan

Scans the entire table page by page and row by row, searching for data.

INDEX PAGE

It stores key values (Pointers) to another page.

It doesn't store the actual rows.

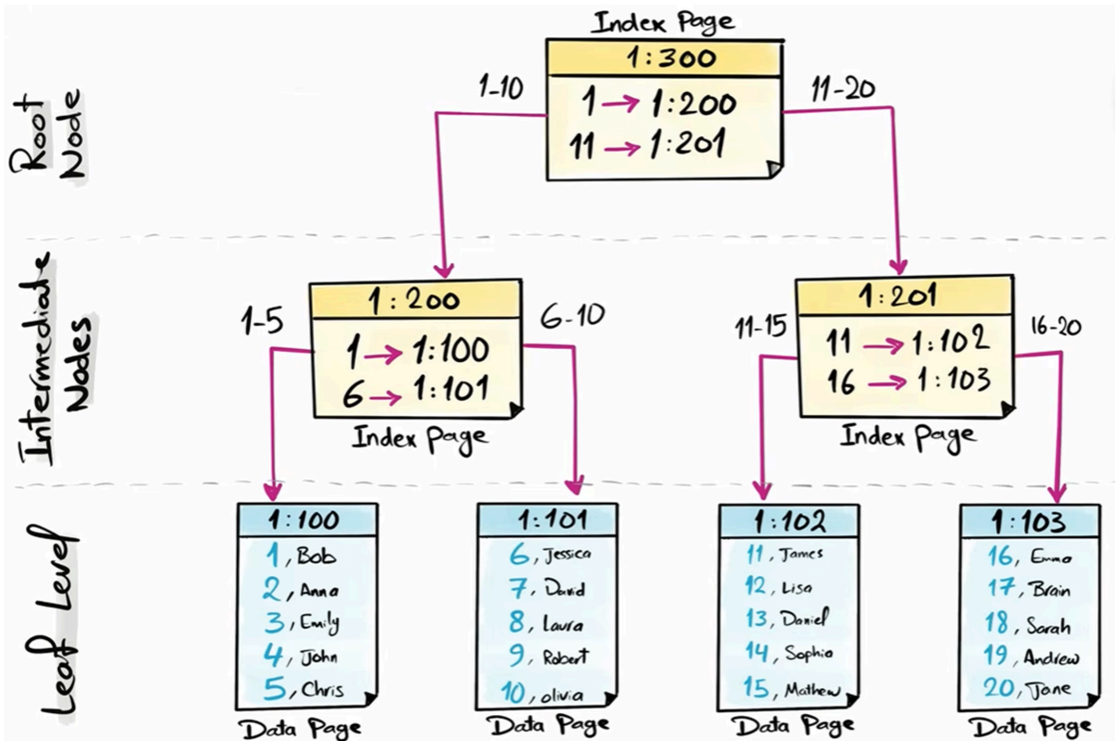
B-TREE

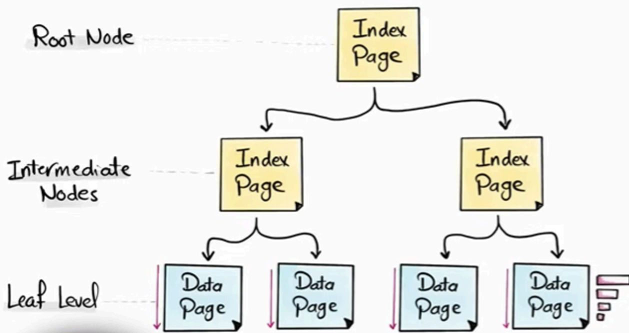
(BALANCE TREE)

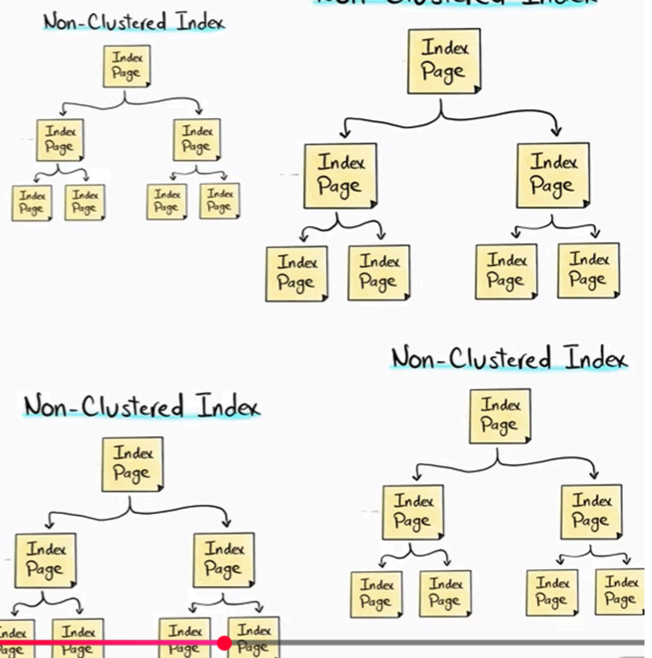
Hierarchical structure storing data at leaves,

to help quickly locate data.

Cluster







	Clustered Index	Non-Clustered Index
Definition	Physically sorts and stores rows	Separate structure with pointers to the data
Numebr of Indexes	One Index per Table	Multiple indexes are allowed
Read Performance	Faster	Slower
Write Performance	Slower, due to potential data row reordering	Faster, since physical data order is unaffected
Storage Efficiency	More storage-efficient	Requires additional storage space
Use Case	• Unique Column • Not frequently modified Column • Improve range query performance	• Columns frequently used in search conditions and joins • Exact match queries

Index Syntax

Default is NONCLUSTERED

```
CREATE [CLUSTERED | NONCLUSTERED] INDEX index_name ON table_name (column1, c
```

```
CREATE CLUSTERED INDEX IX_Customers_ID ON Customers (ID)

CREATE NONCLUSTERED INDEX IX_Customers_City ON Customers (City)

CREATE INDEX IX_Customers_Name ON Customers (LastName ASC, FirstName DESC)
```

NONCLUSTERED

INFO

a Primary Key (PK) automatically creates a clustered index by default.

RULE

Only ONE clustered index can be created per table

```
A,B,C,D
```

Could not find stored procedure 'A'.

```
-- Index will be used
A
A,B

-- Index won't be used
B
A,C
```

Leftmost Prefix Rule

Index works only if your query filters start from the first column in the index and follow its order.

3. 区别总结

特性	聚集索引（Clustered Index）	非聚集索引（Non-Clustered Index）
数据存储	数据按索引排序存储，数据页是索引的叶子节点	数据独立存储，索引叶子节点存储指针
数量限制	每张表只能有一个	每张表可以有多个
速度优势	范围查询、排序、聚合性能高	单值查找性能高
维护成本	数据插入/更新可能引发重排	插入/更新只需维护索引结构
空间需求	不需要额外空间存储索引	需要额外空间存储索引

---

4. 图中细节

聚集索引

- 聚集索引的叶子节点是 **数据页（Data Page）**，存储了表中的实际数据。
- 访问聚集索引时，只需沿着 B+树结构从根节点到叶子节点，就可以直接访问目标数据。
- 优势在于，数据存储与索引一致，因此适合排序、范围查询等。

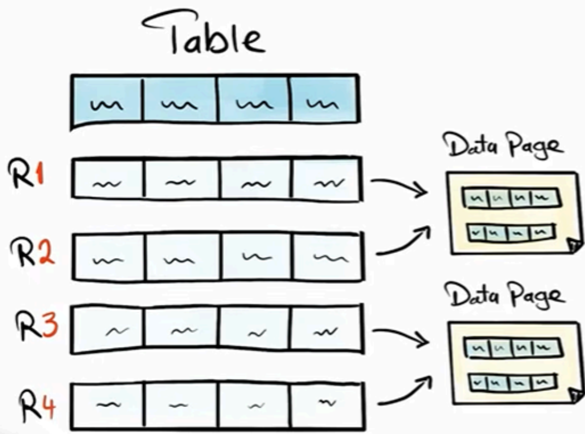
非聚集索引

- 非聚集索引的叶子节点是 **索引页（Index Page）**，存储指向实际数据行的指针。
- 访问非聚集索引时，需要先查找索引页，然后通过指针访问实际的数据行。
- 图中展示了多个非聚集索引（可以有多个索引树结构），每个索引都服务于不同的查询需求。

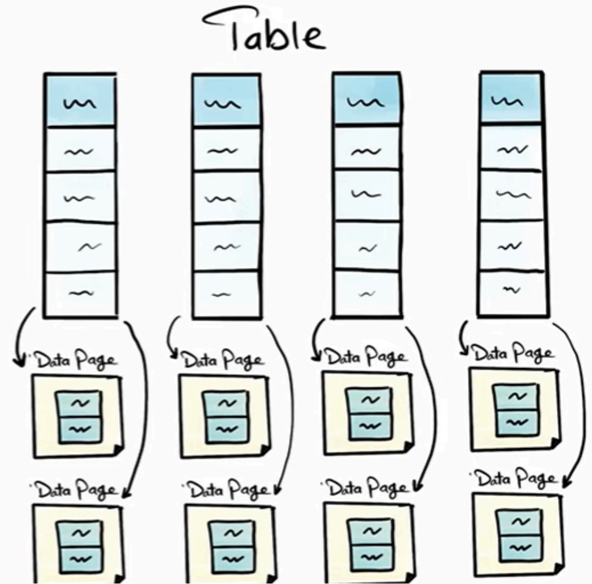
查询性能对比

```SQL
-- 使用非聚集索引加速查询
SELECT * FROM Employees WHERE LastName = 'Smith';

-- 使用聚集索引加速范围查询
SELECT * FROM Employees WHERE HireDate BETWEEN '2022-01-01' AND '2023-01-01';
```

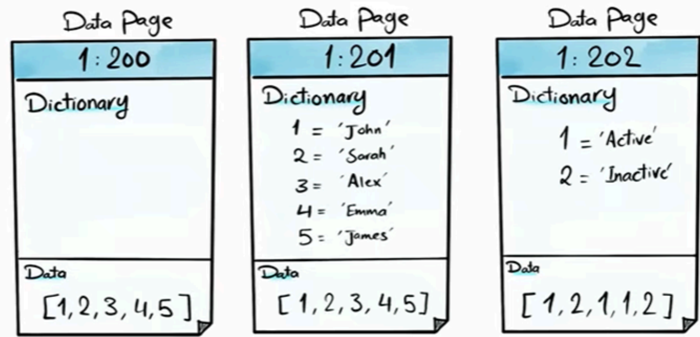
Rowstore Index



Columnstore Index



Columnstore



ID	Name	Status
1	John	Active
2	Sarah	Inactive
3	Alex	Active
4	Emma	Active
5	James	Inactive

Rowstore

[1, John, Active]
[2, Sarah, Inactive]
[3, Alex, Active]
[4, Emma, Active]
[5, James, Inactive]

Query

```
SELECT COUNT(*) FROM Customers
WHERE Status = 'Active'
```

Feature	Rowstore	Columnstore
Storage organization	Data stored row by row	Data stored column by column
Access pattern	Reads entire rows, even if only one column is needed	Reads only relevant columns
Query performance	Optimized for single-row lookups	Optimized for column-based aggregations
Space efficiency	Stores data as-is; less compression	High compression (e.g., dictionaries)
Use case	OLTP (transactional systems, frequent updates)	OLAP (analytical queries, aggregations)
Write performance	Faster writes (row-level operations)	Slower writes (column-based organization)
Examples	Banking systems, e-commerce applications	Data warehouses, reporting systems

3. Practical Example: When to Use Each

Rowstore Example (OLTP)

- Use when performing operations like:
    - `INSERT INTO Customers (ID, Name, Status) VALUES (...)`
    - `UPDATE Customers SET Status = 'Active' WHERE ID = 1`
    - `DELETE FROM Customers WHERE ID = 2`
- These operations are efficient in rowstore because the database can quickly find, update, or delete entire rows.

Columnstore Example (OLAP)

- Use when performing operations like:
    - `SELECT COUNT(*) FROM Sales WHERE Region = 'North America'`
    - `SELECT AVG(SalesAmount) FROM Sales WHERE ProductCategory = 'Electronics'`
- These operations are efficient in columnstore because only the necessary columns are scanned, saving time and reducing I/O.

	Rowstore Index	Columnstore Index
Definition	Organizes and stores data **row by row**	Organizes and stores data **column by column**
Storage Efficiency	**Less efficient in storage**	**Highly efficient with Compression**
Read/Write Optimization	**Fair speed for read & write operations**	**Fast read performance** **Slow write performance**
I/O Efficiency	**Lower (retrieves all columns)**	**Higher (retrieves specific columns)**
Best for ..	**OLTP (Transactional)** commerce, banking, Financial systems, order processing	**OLAP (Analytical)** Data Warehouse, Business intelligence, Reporting, Analytics
Use Case	**- High-frequency transaction applications** **- Quick access to complete records**	**- Big Data Analytics** **- Scanning of large datasets** **- Fast aggregation**

Columnstore Index Syntax

Default is ROWSTORE

```
CREATE [CLUSTERED | NONCLUSTERED] [COLUMNSTORE] INDEX index_name
ON table_name (column1, column2, ...)
```

Rowstore

```
CREATE NONCLUSTERED INDEX IX_Customers_Country ON Customers (Country)
CREATE CLUSTERED INDEX IX_Customers_ID ON Customers (ID)
```

Columnstore

```
CREATE NONCLUSTERED COLUMNSTORE INDEX IX_Customers_Country ON Customers (Country)
CREATE CLUSTERED COLUMNSTORE INDEX IX_Customers ON Customers
```

NOT ALLOWED TO USE COLUMNS

Rules

- You can't specific columns in Clustered Index Columnstore

HEAP = 9,641 MB

Rowstore Clustered = 9,688 MB

Columnstore Clustered = 1,461 MB

Unique Index

Ensures no duplicate values exist in specific column.

Benefits

1. Enforce uniqueness
2. Slightly increase query perfromance

Performance

Writing to an unique index is slower than non-unique.

Reading from an unique index is faster than non-unique.

Index Syntax

Default is NOT Unique

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] [COLUMNSTORE] INDEX index_name
ON table_name (column1, column2, ...)
```

Index Allows Duplicates

```
CREATE INDEX IX_Customers_Email ON Customers (Email)
```

Duplicates are not allowed

```
CREATE UNIQUE INDEX IX_Customers_Email ON Customers (Email)
```

RULE

Duplicates in the columns will prevent creating a unique index.

Filtered Index

An index that includes only rows meeting the specified conditions

Benefits

- Targeted Optimization
- Reduce storage: Less data in the index

Filtered Index Syntax

```
CREATE [UNIQUE] [NONCLUSTERED] INDEX index_name
ON table_name (column1, column2, ...)
WHERE [Condition]
```

Rules

- You cannot create a filtered index on a clustered index.
- You cannot create a filtered index on a columnstore index.

When To Use

HEAP
Fast **Inserts**
(For Staging Tables)

Clustered Index (OLTP)
For **Primary keys**
If not, then for date columns

Columnstore Index (OLAP)
For **Analytical** Queries
Reduce Size of Large Table

Non-Clustered Index
For **non-PK** columns
(Foreign keys, Joins, and Filters)

Filetered Index
Target **Subset** of Data
Reduce Size of Index

Unique Index
Enforce **Uniqueness**
Improve Query Speed

Index 对比

- Heap（堆表）
  - 说明：没有聚集索引的表，数据以插入顺序存放。
  - 何时用：需要非常快速的大量插入（例如临时的 staging 表或批量加载 ）时。
  - 注意：查询性能通常较差，尤其是范围查询或需要排序时；可能需要额外的非聚集索引来提升读取性能。
- Clustered Index（聚集索引）
  - 说明：按索引键的顺序物理存放数据行（每张表只有一个聚集索引）。
  - 何时用：通常用于主键；如果主键不合适，优先选择用于频繁按范围/日期查询的列（例如时间序列的日期列）。
  - 优点：范围扫描、按序检索、排序和分区友好。
  - 注意：在写入密集（频繁插入/更新）且插入顺序与聚集键不一致时会产生页分裂和碎片。
- Columnstore Index（列存储索引）
  - 说明：按列压缩与存储，针对列扫描优化。
  - 何时用：分析型查询/OLAP 场景（大表的聚合、报表、扫描），能显著减少存储并加速大规模扫描。
  - 注意：对 OLTP 小事务或频繁点更新不适合。可用作非聚集列存储索引实现混合场景。
- Non-Clustered Index（非聚集索引）
  - 说明：独立于表数据的索引结构，包含键值与对数据行的指针（聚集键或 RID）。
  - 何时用：用于非主键列，尤其是外键、用于连接（JOIN）的列或常作为 WHERE/过滤条件的列。
  - 优点：加速特定查询；可以通过包含列（INCLUDE）创建覆盖索引减少回表。
  - 注意：每新增索引会增加写入开销和维护成本，避免在高写表上过度索引。
- Filtered Index（带过滤的索引）
  - 说明：只索引满足某一 WHERE 条件的行，索引更小、更针对性。
  - 何时用：目标是数据的一个子集（例如状态列仅为 'Active' 的行），能减少索引大小并提高该子集查询性能。
  - 注意：适用于稀疏值或热点子集；查询必须能利用该过滤条件。
- Unique Index（唯一索引）
  - 说明：在索引级别强制唯一性（也可作为约束实现 ）。
  - 何时用：需要保证列值唯一（例如业务唯一键、 复合唯一约束），并能提升基于该列的查找速度。
  - 注意：唯一性检查会在写入时增加开销。

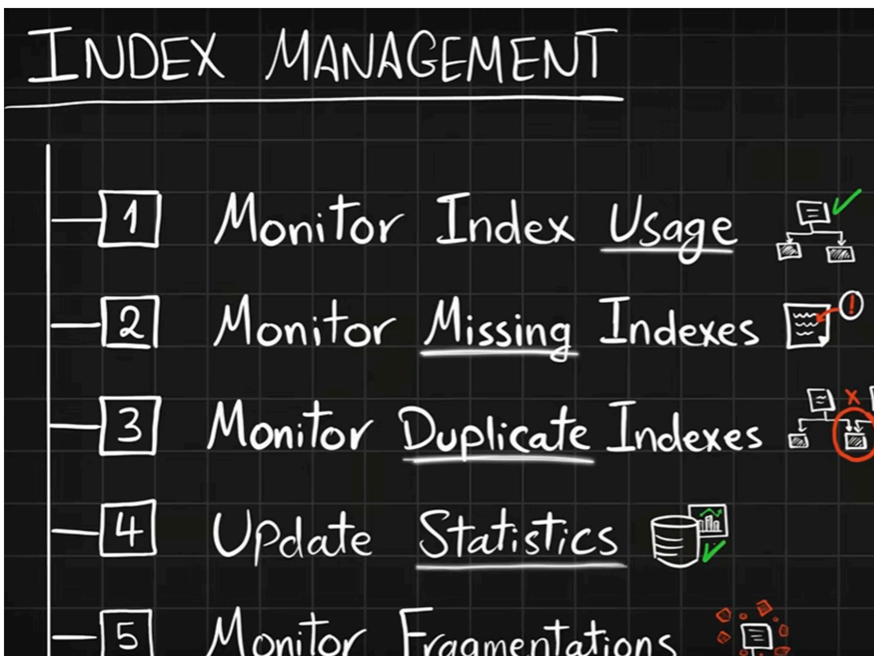
实用建议与权衡

- OLTP vs OLAP：OLTP 多为事务型，优先聚集索引（主键 ）+几个非聚集索引以加速点查与连接；OLAP 偏向列存储以优化大范围扫描与聚合。
- 索引选择原则：优先高选择性列（区别度高），尽量保持索引窄（少列），必要时使用 INCLUDE 添加非键列以覆盖查询。
- 写/读平衡：写密集表要限制索引数量并合理设置填充因子（fill factor）；读密集表可多建索引以提高查询速度。
- 维护：定期重建/重组索引与更新统计信息，监控碎片与缺失索引建议。

- 其他技巧:对于需要排序或范围查询的场景,把聚集索引键设计为时间类或增长键可以减少页分裂;使用带过滤的索引加速典型子集查询;在需要强制唯一性的地方使用唯一索引代替额外逻辑检查。

简短定义

- OLTP(在线事务处理):面向日常业务事务的系统,处理大量并发的短小读写操作(例如:下单、付款、库存变更)。
- OLAP(在线分析处理):面向分析和报表的系统,处理复杂的聚合、历史分析和大规模扫描(例如:销售报表、数据仓库查询)。
- OLTP 示例:电商网站处理订单创建与库存扣减(高并发、低延迟)。
- OLAP 示例:分析整个季度的销售趋势并按地区/产品聚合(大表扫描、复杂聚合)。



'Sys' System Schema

contains metadata about database tables, views, indexes..etc

Dynamic Management View (DMV)

provides real-time insights into Database performance and system health

```sql
FROM sys.dm_db_missing_index_details
```

TIP: Evaluate the recommendations before creating any index

Fragmentation

- Unused spaces in data pages
- Data pages are out of order

Fragmentation Methods

Reorganize

- Defragments leaf nodes to keep them sorted
- "Light" Operation

Rebuild

- Recreates Index from Scratch
- "Heavy" Operation

```sql
-- Retrieve index fragmentation statistics for the current database
SELECT
    tbl.name AS TableName,
    idx.name AS IndexName,
    s.avg_fragmentation_in_percent,
    s.page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS s
INNER JOIN sys.tables tbl
    ON s.object_id = tbl.object_id
INNER JOIN sys.indexes AS idx
    ON idx.object_id = s.object_id
    AND idx.index_id = s.index_id
ORDER BY s.avg_fragmentation_in_percent DESC;

-- Reorganize the index (lightweight defragmentation)
ALTER INDEX idx_Customers_CS_Country
ON Sales.Customers REORGANIZE;
GO

-- Rebuild the index (full rebuild, more resource-intensive)
ALTER INDEX idx_Customers_Country
ON Sales.Customers REBUILD;
GO
```

1. ALTER INDEX ... REORGANIZE

- 做什么：在叶级页上做“在线整理”（defragment/compact），按逻辑顺序重排页、合并空闲空间、压缩页内碎片。是增量、轻量的碎片整理操作。
- 是否在线：是在线操作（不会把表/索引长期置于独占锁），对用户并发影响小。
- 日志/资源：耗资源较少，日志写入也比较分散（相对比 REBUILD 小）。
- 统计信息：通常不会像重建那样做完全重建的统计直方图（不会做 FULLSCAN 的统计重建）。
- 何时用：索引碎片率较低到中等（常见经验规则：碎片在 ~5%–30% 之间）或不能承受重建带来的开销/锁定时用。适合在线维护与渐进整理。

2. ALTER INDEX ... REBUILD

- 做什么：彻底重建索引（在内部会创建新的索引结构然后替换旧的），清除所有碎片、重建页分配、可以改变 fillfactor、回收空间。相当于全量重建索引。
- 是否在线：可以是离线也可以是在线（取决于 SQL Server 版本/许可和你是否加 WITH (ONLINE = ON)），离线重建会阻塞访问，在线重建仍会在某些时刻短暂切换。
- 日志/资源：资源和事务日志开销大（重建期间会大量写日志），需要更多临时磁盘和 I/O。
- 统计信息：重建会更新索引相关的统计信息（默认是全表/全索引的统计重建，通常比整理更准确）。
- 何时用：碎片高（常见经验：> 30%）或需要回收大量空间、需要更新统计、或者做了大量批量加载/删除后使用。对性能改进通常更明显但代价更高。

实用建议（快速决策）

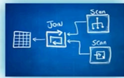
- 碎片 < ~5%：不操作。
- 碎片 ~5%–30%：REORGANIZE（如果系统不能承受重建开销，或想在线逐步整理）。
- 碎片 > ~30%：REBUILD（若可安排维护窗口或可做在线重建）。
- 大批量导入/删除后：优先 REBUILD（确保索引和统计都是新鲜且最优）。
- 如果担心事务日志空间或不能阻塞用户，优先 REORGANIZE；如果需要最彻底的清理并同时更新统计，优先 REBUILD。

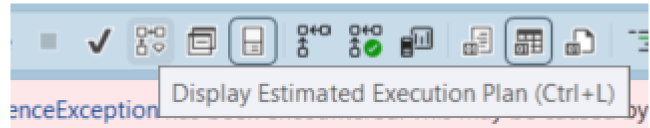
注意点

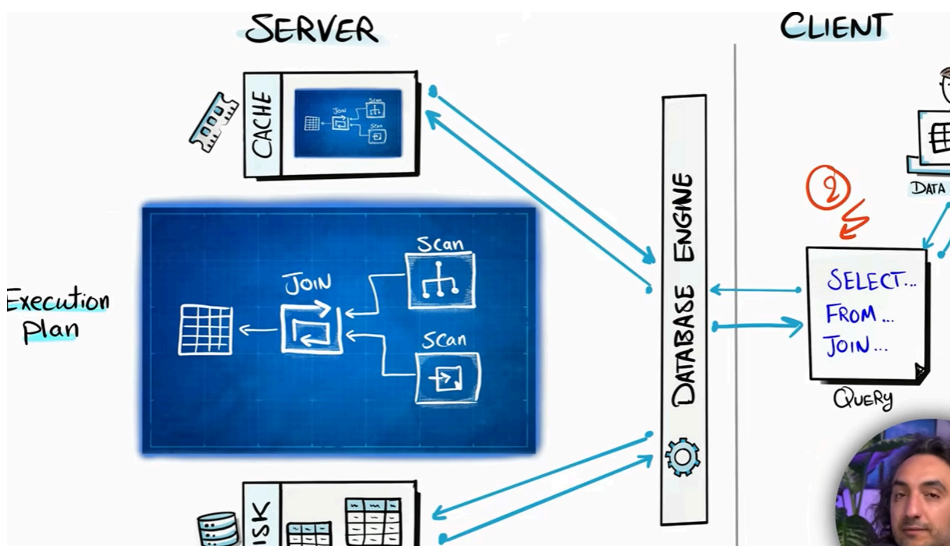
- REBUILD 会默认更新统计（有助于生成更好执行计划）。
- REORGANIZE 不会大幅更改 fillfactor，也不能指定填充因子；REBUILD 可以设置 FILLFACTOR、SORT_IN_TEMPDB 等选项。
- 对列存储索引（columnstore）也适用，但内部语义有差别（例如 REORGANIZE 可以合并 rowgroups，REBUILD 则完全重建列存储结构），所以对 CCI 的维护要参考具体场景。
- 在生产环境做 REBUILD 前评估日志/IO/时间成本，必要时在维护窗口或使用 ONLINE=ON（若支持）执行。

如果你把某个索引的碎片率（或 sys.dm_db_index_physical_stats 的结果）贴出来，我可以帮你判断是用 REORGANIZE 还是 REBUILD，并给出带选项的具体命令示例。

20. 20:20:31 Execution Plan







SQLQuery2.sql - D...8QBU\Youtube (52))*

```sql
SELECT *
FROM FactResellerSales
ORDER BY SalesOrderNumber
```

Results | Messages | Execution plan

Query 1: Query cost (relative to t…
SELECT * FROM FactResellerSales OR…

SELECT Cost: 0 % — Clustered… [FactRese… Cost: 100… 0.048s 60855 of 60855 (10…

Misc	
Actual Execution Mode	Row
Actual I/O Statistics	
Actual Number of Batches	0
Actual Number of Rows for All Executions	60855
Actual Number of Rows Read	60855
Actual Rebinds	0
Actual Rewinds	0
Actual Time Statistics	
Defined Values	[AdventureWorksDW2022].[dbo].[FactResellerSales].Pr
Description	Scanning a clustered index, entirely or only a range.
Estimated CPU Cost	0,0670975
Estimated Execution Mode	Row
Estimated I/O Cost	1,24164
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	60855
Estimated Number of Rows Per Execution	60855
Estimated Number of Rows to be Read	60855
Estimated Operator Cost	1,30874 (100%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	224 B
Estimated Subtree Cost	1,30874
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Clustered Index Scan
Node ID	0
NoExpandHint	False
Number of Executions	1
Object	[AdventureWorksDW2022].[dbo].[FactResellerSales].[P
Database	[AdventureWorksDW2022]
Index	[PK_FactResellerSales_SalesOrderNumber_SalesOrderL
Index Kind	Clustered
Schema	[dbo]

Index
Index name for the referenced object

TIP

After creating a new index, check the execution plan to see if your query uses the index

Index Seek

A targeted search within an index, retrieving only specific rows.

Types of Scan

Table Scan: Reads every row in a table.

Index Scan: Reads all entries in an index to find results.

Index Seek: Quickly locates specific rows in an index.

Join Algorithms

Nested Loops: Compares tables row by row; best for small tables.

Hash Match: Matches rows using a hash table; best for large tables.

Merge Join: Merge two sorted tables; efficient when both are sorted.

```sql
SELECT
    p.EnglishProductName AS ProductName,
    SUM(s.SalesAmount) AS TotalSales
FROM FactResellerSales_HP s
JOIN DimProduct p
ON p.ProductKey = s.ProductKey
GROUP BY p.EnglishProductName

CREATE CLUSTERED COLUMNSTORE INDEX idx_FactReseller
ON FactResellerSales_HP
```

Query 1: Query cost (relative to the batch): 100%
SELECT p.EnglishProductName AS ProductName, SUM(s.Sale…

Fact Columnstore Costs = 0,012

Fact Rowstore Costs = 1,308

Execution Plan

Understand how SQL executes your query

How many resources your query consumes?

Check if your new Indexes are used

Testing & Experimenting Indexes

Bad Execution Plan!

1. Outdated Statistics
2. Too Many Indexes

SQL HINTS

Commands you add to a query to force the database to run it in a specific way for better performance

TIPS | SQL HINTS

1. Test hints in all project environments (DEV, PROD) as performance may vary.
2. Hints are quick fixes (Workaround not Solution) You still have to find the cause and fix it.

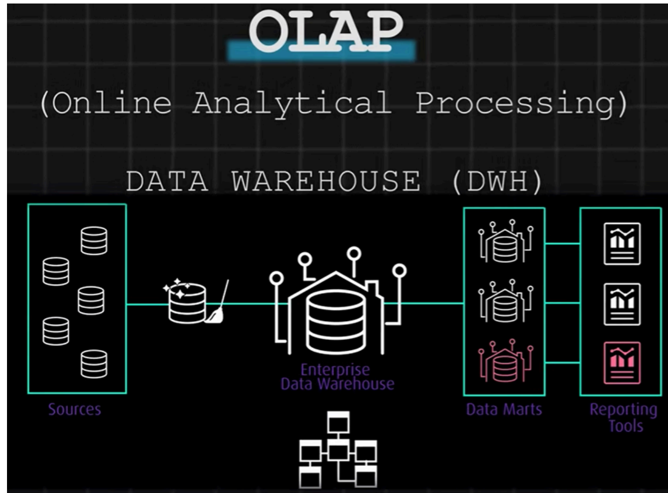
TOO MANY INDEXES CAN CONFUSE EXECUTION PLAN

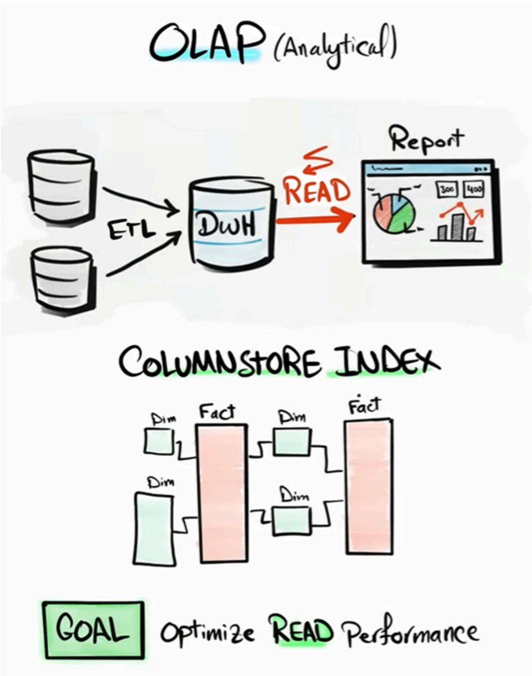
Increase Planing Time

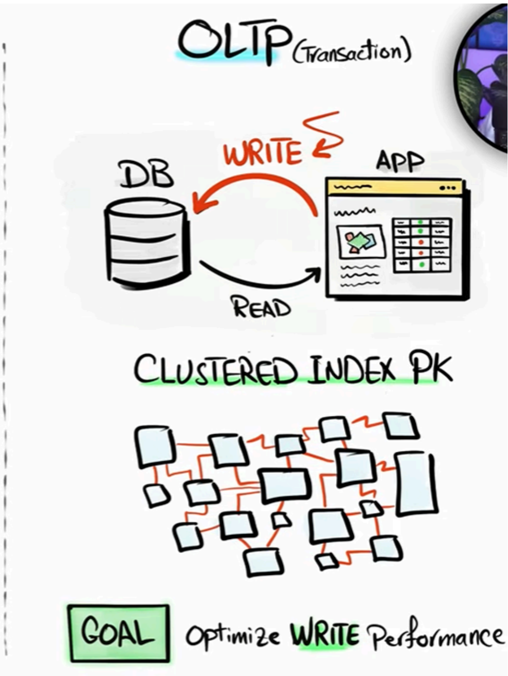
Choose a suboptimal plan

AVOID OVER INDEXING

what is your goal???








```
CROSS APPLY sys.dm_db_stats_properties(s.object_id, s.stats_id) AS sp
```

When to Defragment?

< 10%	No Action needed
10 – 30%	Reorganize
> 30%	Rebuild

Indexing Strategy

#1 Initial Indexing Strategy

OLAP
Optimize
Read
Performance

Switch **Large** frequently used tables into **ColumnStore**

OLTP
Optimize
Write
Performance

Clustered Index
Primary Keys

#2 Usage Patterns Indexing

1. Identify **frequently** used **Tables** & **Columns**
2. Choose **Right** Index
3. **Test** Index

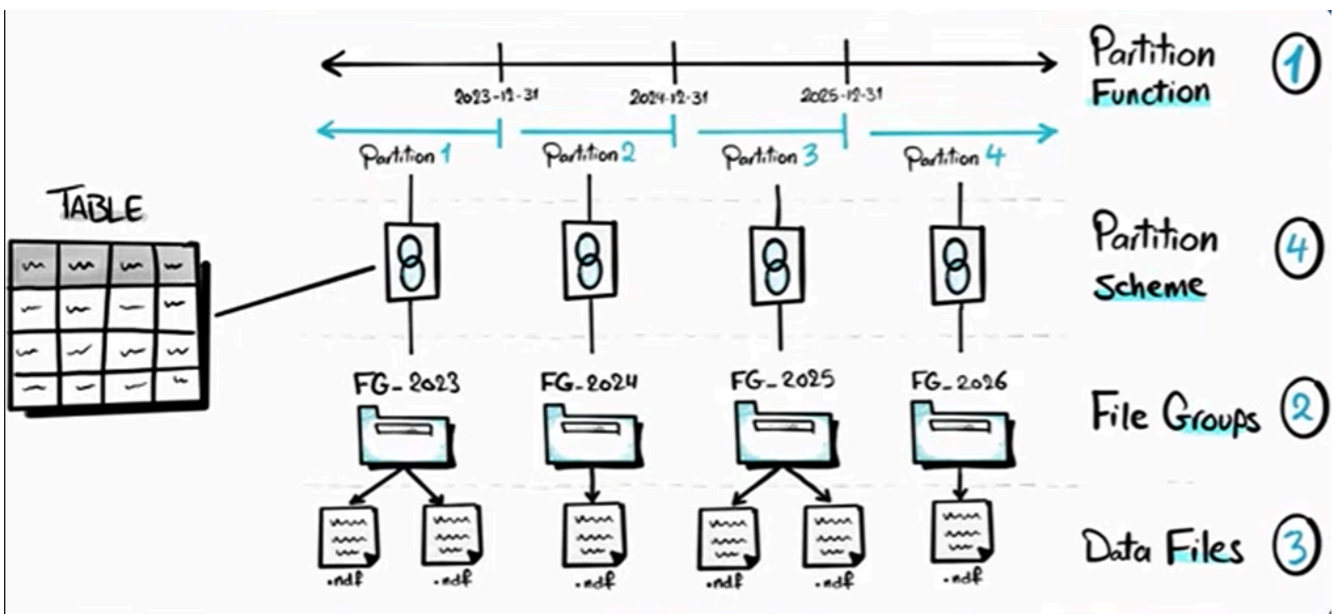
#3 Scenario-Based Indexing

1. Identify **Slow** Queries
2. Check **Execution Plan**
3. Choose **Right** Index
4. (Test) **Compare** Execution Plans

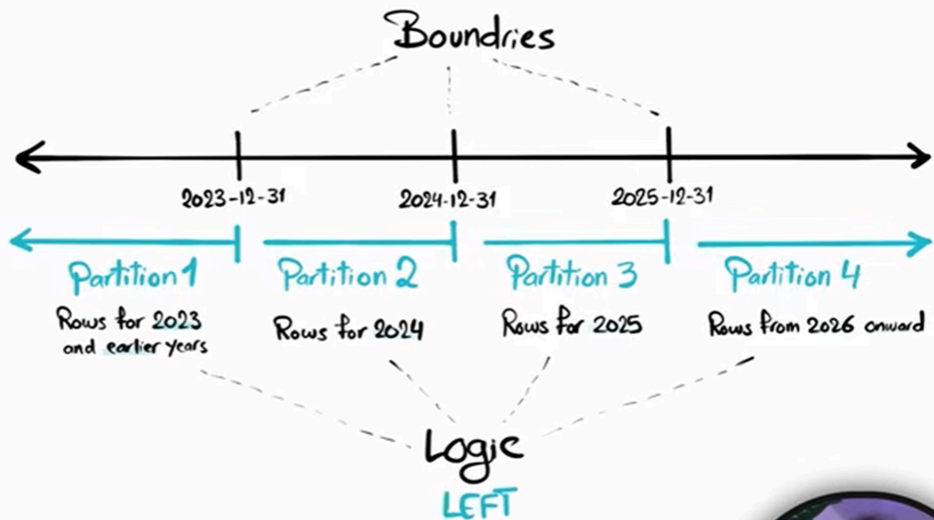
#4 Monitoring & Maintenance

1. Monitor Index **Usage**
2. Monitor **Missing** Indexes
3. Monitor **Duplicate** Indexes
4. Update **Statistics**
5. Monitor **Fragmentations**

21. 21:11:03 Partitions

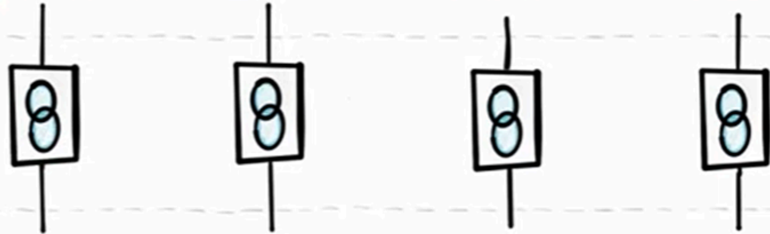


① Partition Function



① Partition Function

④ Partition Scheme



② File Groups

③ Data Files

GOLDEN RULE

Always check the execution plan to confirm performance improvements when optimizing your query.

If there's no improvement, then just focus on readability.

FILEGROUPS

Logical container of one or more data files to help organize partitions.

NOTE

Functions on columns can block index usage

```sql
-- Good Practice
SELECT *
FROM Sales.Customers
WHERE LastName LIKE 'Gold%'
```

```sql
SELECT *
FROM Sales.Customers
WHERE LastName LIKE '%Gold%'
```

#6 TIP

Avoid using leading wildcards, as they prevent index usage

#7 TIP

Use IN instead of multiple OR conditions

22. 21:43:39 30x Performance Tips

1. 数据获取 (FETCHING DATA)
* **只选择需要的列：** 避免使用 `SELECT *`，只选取查询必需的列，减少数据传输和处理量。
* **避免不必要的 `DISTINCT` 和 `ORDER BY`：** 这些操作开销较大，仅在确实需要唯一结果或特定排序时使用。
* **探索数据时限制行数：** 对于数据探索，使用 `TOP N` (或 `LIMIT N`) 限制返回的行数，以提高响应速度和避免处理大量数据。

2. 过滤 (FILTERING)
* **在 `WHERE` 子句中频繁使用的列上创建非聚集索引：** 这能显著加快数据检索速度。
* **避免在 `WHERE` 子句中对列应用函数：** 对列应用函数（如 `LOWER()`、`YEAR()`）会阻止索引的使用。应转换比较值或使用范围查询（如 `BETWEEN`）。
* **避免使用前导通配符 (`%`)：** `LIKE '%keyword'` 无法利用索引。尽量使用 `LIKE 'keyword%'`。
* **使用 `IN` 替代多个 `OR`：** 当对同一列进行多个 `OR` 条件过滤时，`IN` 操作符更清晰且通常性能更好。

3. 连接 (JOINS)
* **理解连接类型并优先使用 `INNER JOIN`：** `INNER JOIN` 通常性能最佳，`LEFT/RIGHT JOIN` 稍慢，`FULL OUTER JOIN` 最慢。根据数据关系选择最合适的连接类型。
* **使用显式 `JOIN` 语法（ANSI Join）：** 显式 `JOIN`（如 `INNER JOIN ... ON`）比隐式 `JOIN`（`FROM Table1, Table2 WHERE`）更清晰、更易于维护和调试。
* **在 `ON` 子句中使用的列上创建索引：** 为连接条件中的列创建索引能大幅提升连接性能。
* **在大表连接时，先过滤再连接：** 使用子查询或CTE在连接前减少大表的行数，降低连接的复杂性。
* **在大表连接时，先聚合再连接：** 对大表进行聚合操作后，再将聚合结果与另一个表连接，可减少连接时的行数。避免使用相关子查询进行聚合。
* **在 `JOIN` 的 `OR` 条件中考虑使用 `UNION`：** 当 `ON` 子句包含 `OR` 且连接的列不同时，可以考虑将其拆分为两个 `INNER JOIN` 查询，然后使用 `UNION` 合并结果，可能获得更好的性能。
* **了解不同连接算法（如 `Nested Loops`）并在必要时使用 SQL 提示：** 审慎使用 `OPTION (HASH JOIN)` 等提示，强制优化器选择特定连接算法，尤其适用于大小表连接，但需充分测试。

4. UNION
* **如果允许重复，使用 `UNION ALL` 替代 `UNION`：** `UNION ALL` 不会检查并移除重复项，因此比 `UNION`（会进行 `DISTINCT` 操作）更快。
* **如果需要唯一结果，使用 `UNION ALL` + `DISTINCT` 替代 `UNION`：** 将 `UNION ALL` 的结果包裹在一个外部 `SELECT DISTINCT` 中，有时能比直接使用 `UNION` 更高效。

5. 聚合 (AGGREGATIONS)
* **在大表聚合时使用列存储索引：** 对于主要用于分析和聚合的大型表，列存储索引能带来显著的性能提升。
* **预聚合数据并存储在新表中用于报表：** 对于频繁运行的复杂报表，可以将预聚合的结果存储到汇总表或物化视图中，从而加快报表生成速度。

6. 子查询, CTE (SUBQUERIES, CTE)
*   **在检查存在性时，优先使用 `JOIN` 或 `EXISTS`，避免使用 `IN`：** `EXISTS` 对于大表通常比 `IN` 更高效，因为它在找到第一个匹配项后就会停止扫描。`JOIN` 也通常与 `EXISTS` 性能相当。
*   **避免查询中的冗余逻辑：** 简化复杂查询，使用窗口函数（如 `AVG() OVER ()`）等方法一次性计算值并应用于多行，而不是重复计算或使用多个 `UNION ALL` 分支。

7. DDL (Data Definition Language)
*   **尽量避免使用 `VARCHAR(MAX)` 数据类型：** 使用更精确的数据类型和合适的长度（如 `VARCHAR(50)`），避免过大的长度。
*   **尽可能使用 `NOT NULL`：** 声明列为 `NOT NULL` 可提高数据完整性、帮助优化器并节省存储空间。
*   **确保所有表都有聚集主键：** 聚集主键会物理地组织表数据，对基于主键的查询和范围查询性能提升巨大。
*   **在频繁使用的外键上创建非聚集索引：** 外键通常用于 `JOIN` 操作，为其创建索引能加快连接速度。

8. 索引 (INDEXING)
*   **避免过度索引：** 过多的索引会减慢 `INSERT`、`UPDATE` 和 `DELETE` 操作，因为每次数据修改时都需要更新所有相关索引。
*   **定期审查并删除未使用的索引：** 未使用的索引浪费存储空间并增加写入操作的开销。
*   **每周更新表统计信息：** 确保查询优化器拥有最新、最准确的数据分布信息，以生成最优的执行计划。
*   **每周重组和重建碎片化的索引：** 定期维护（重组或重建）碎片化索引可以减少I/O操作并提高查询性能。
*   **对于大表（如事实表），分区数据并应用列存储索引：** 结合数据分区和列存储索引，能为大型分析表提供最佳性能。

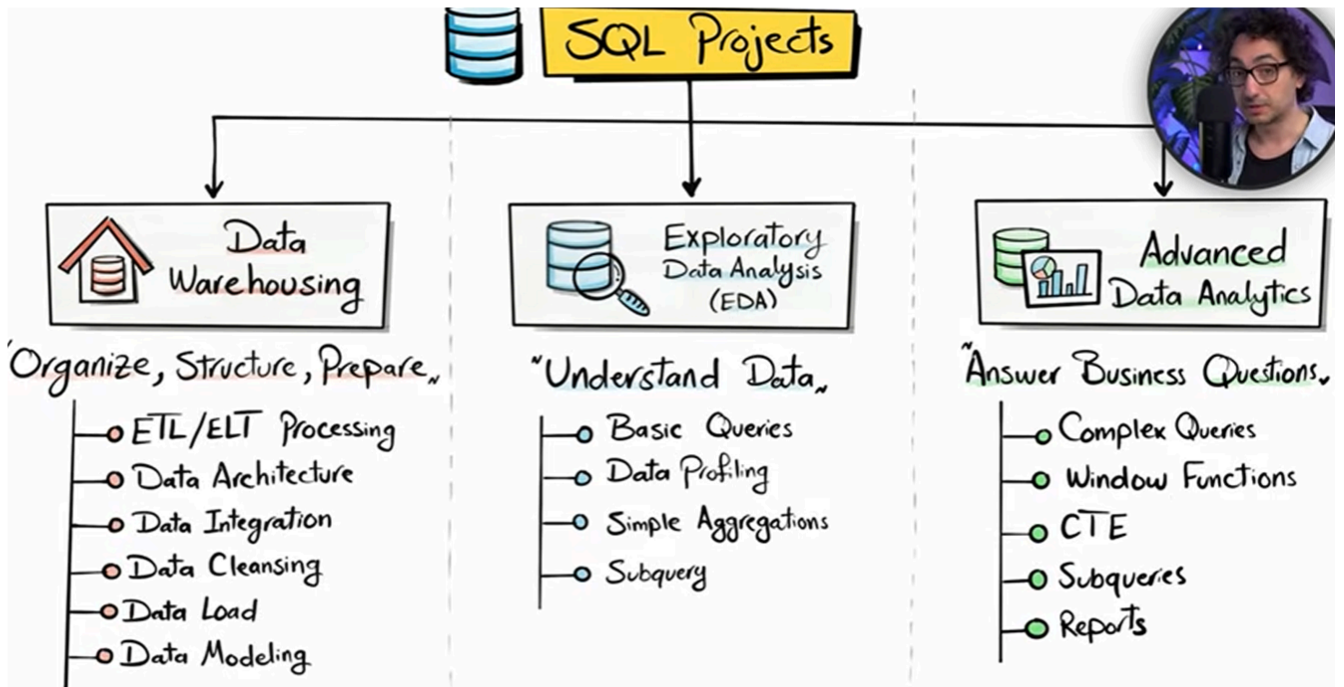
23. 22:24:25 AI and SQL

What can I help with?

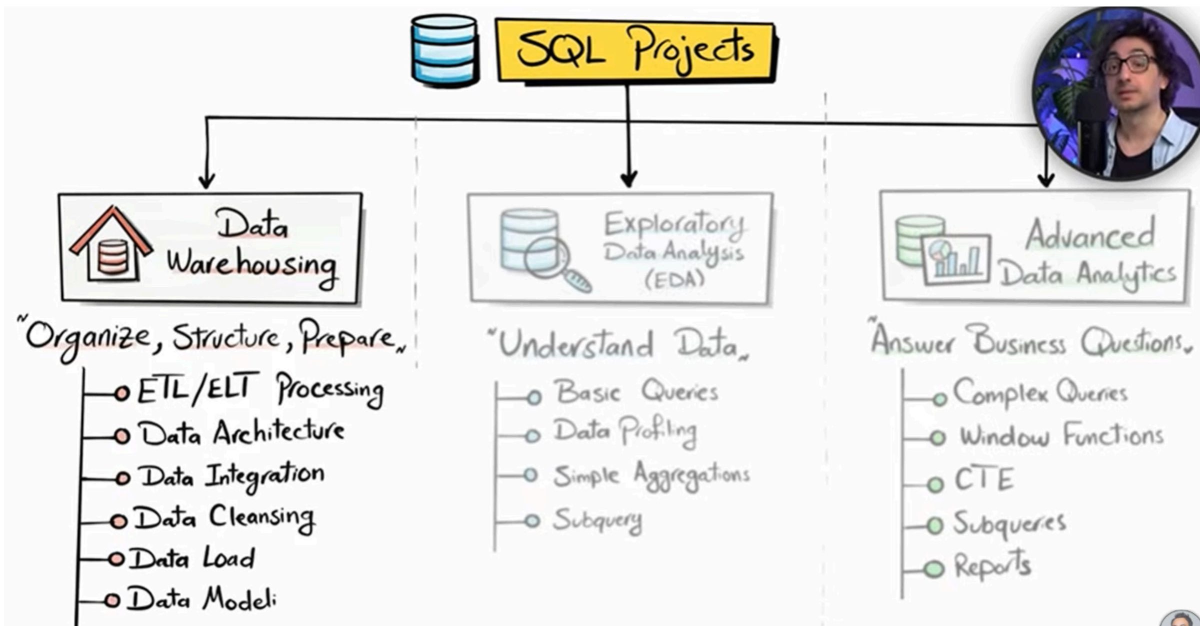
Analyze the following SQL queries and generate a report on table and column usage statistics. For each table, provide:

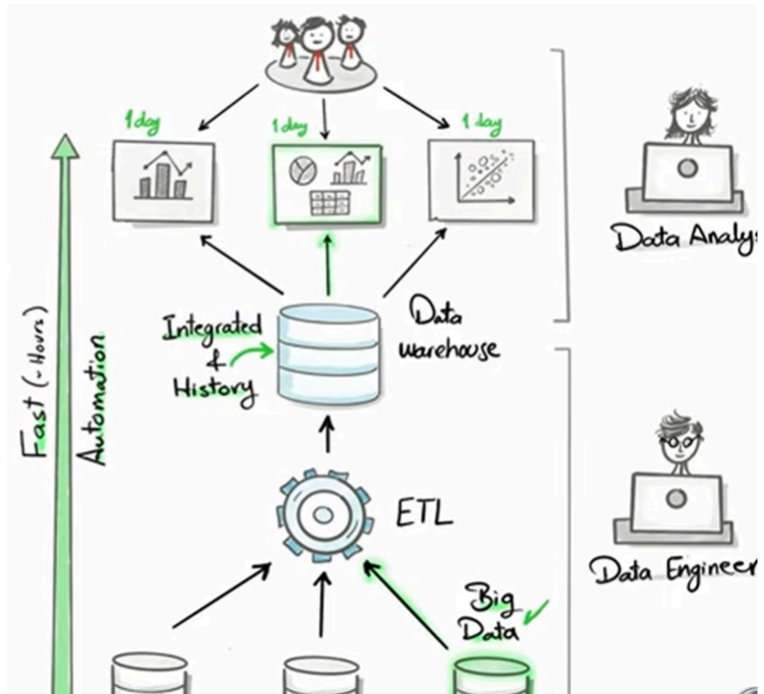
- The total number of times the table is used across all queries.
- A breakdown of each column in the table, showing:
- The number of times each column appears.
  The primary purpose of the column's usage (e.g., filtering, joining, grouping, aggregating).
  Sort the tables in descending order based on their total usage.

__🧪 Projects__



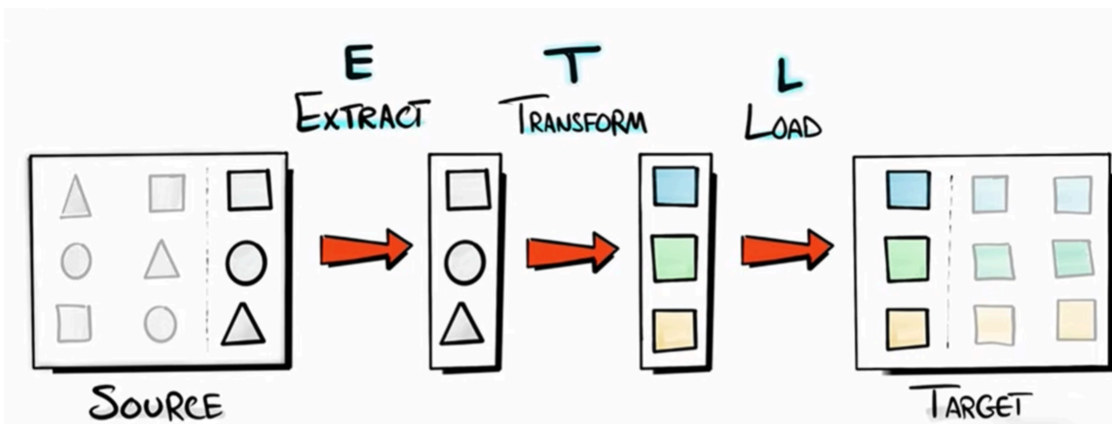
24. [23:21:04] Project: SQL Data Warehouse

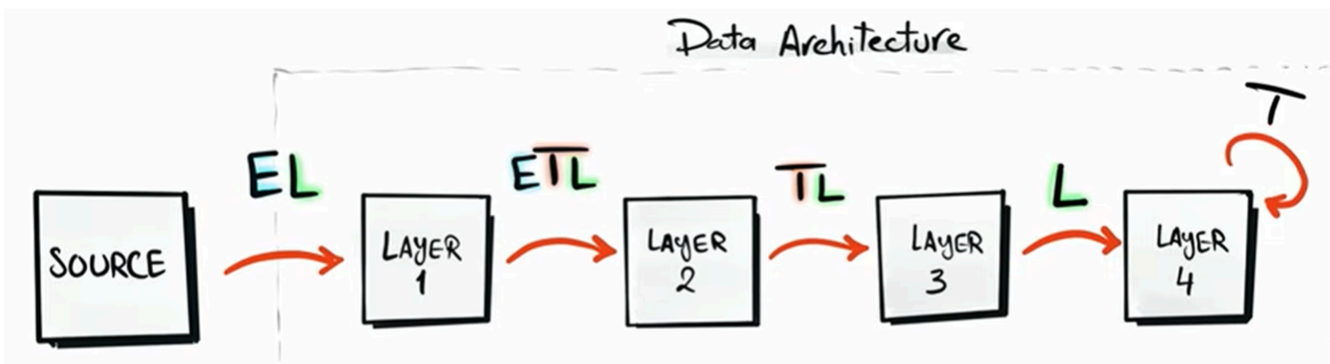


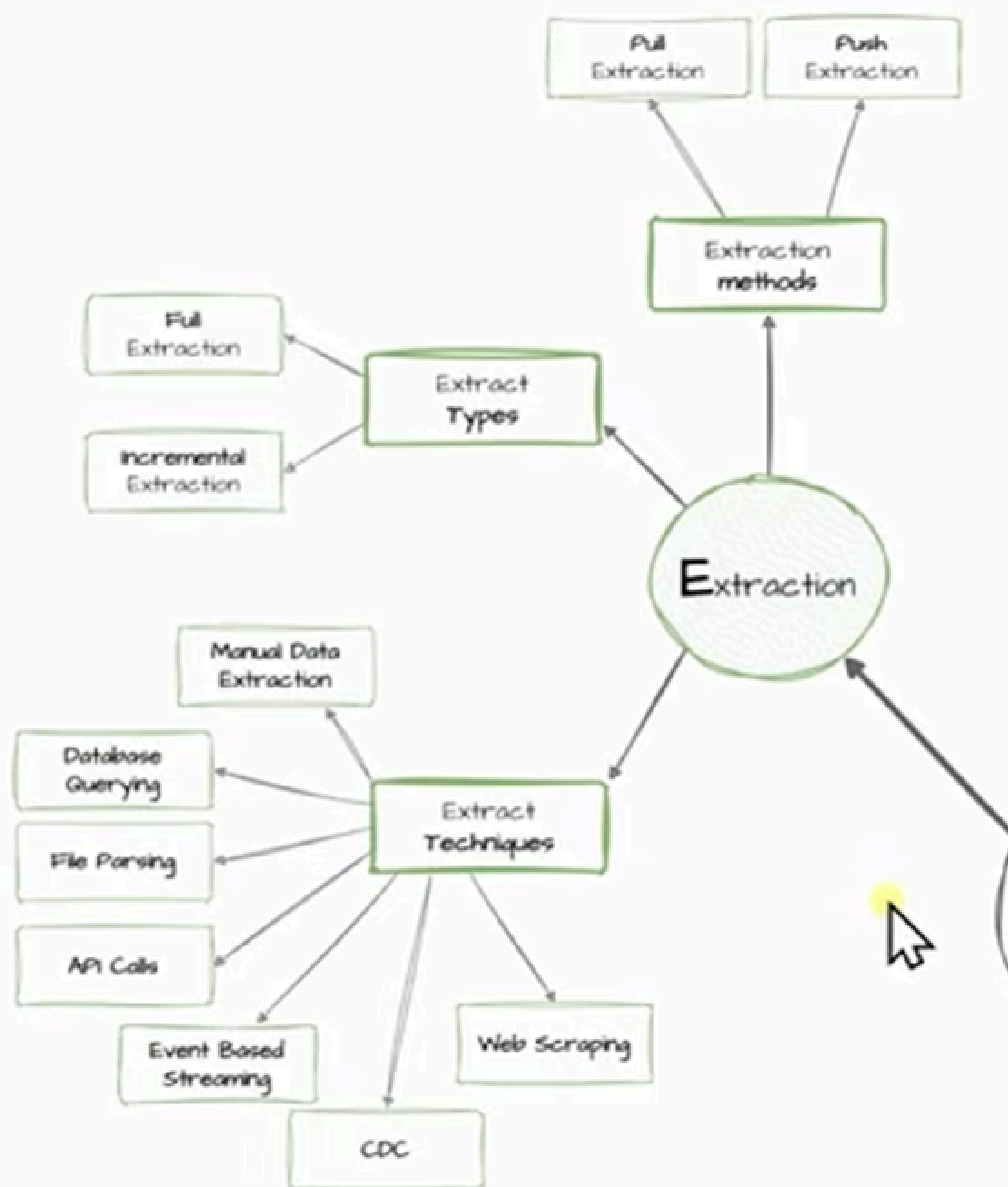


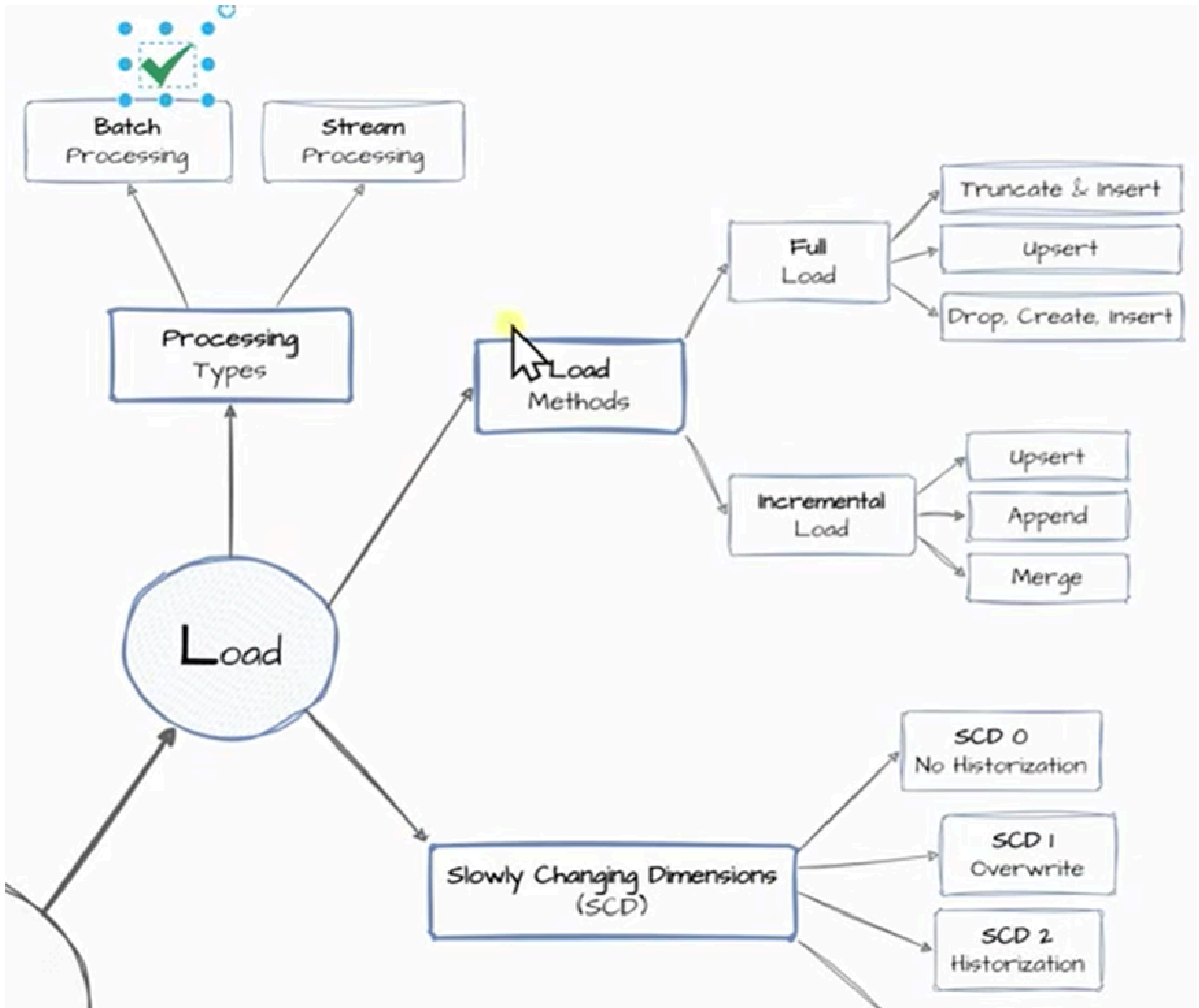
DATA WAREHOUSE

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

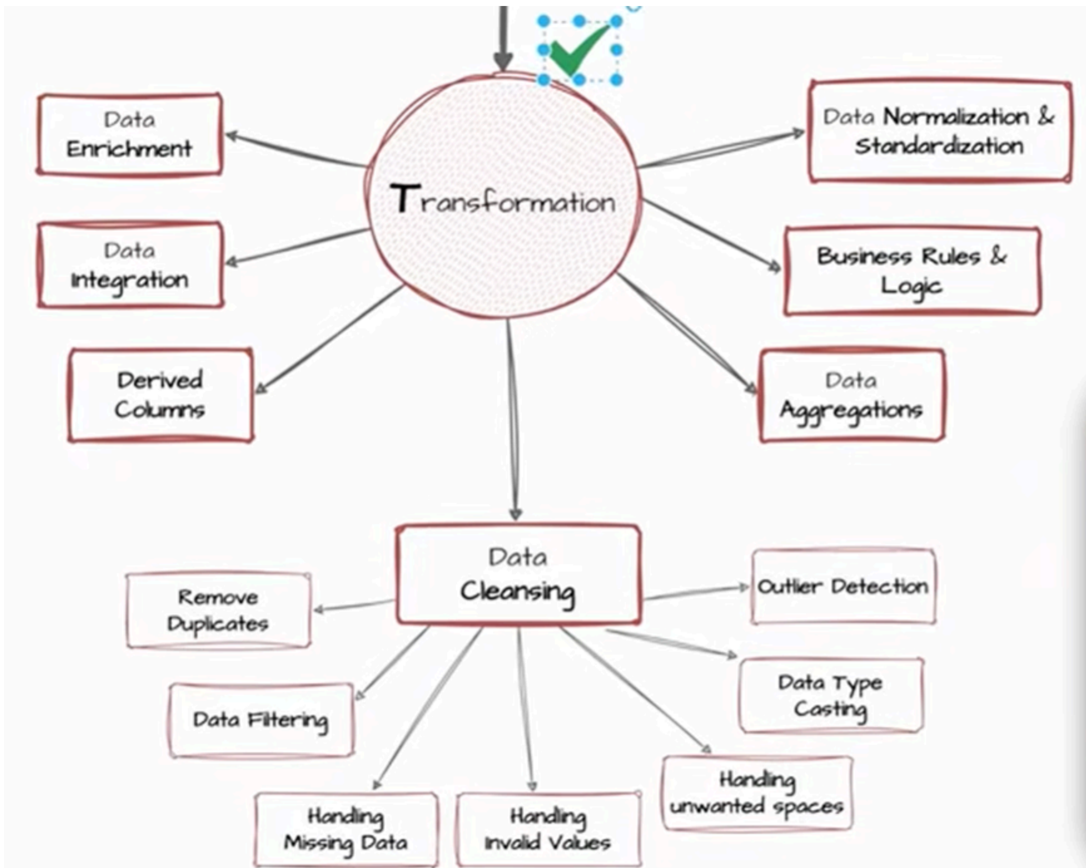


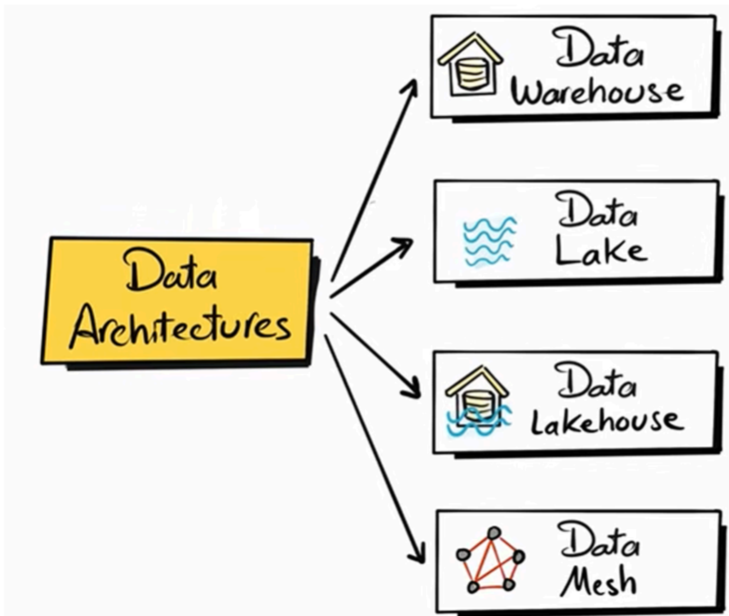


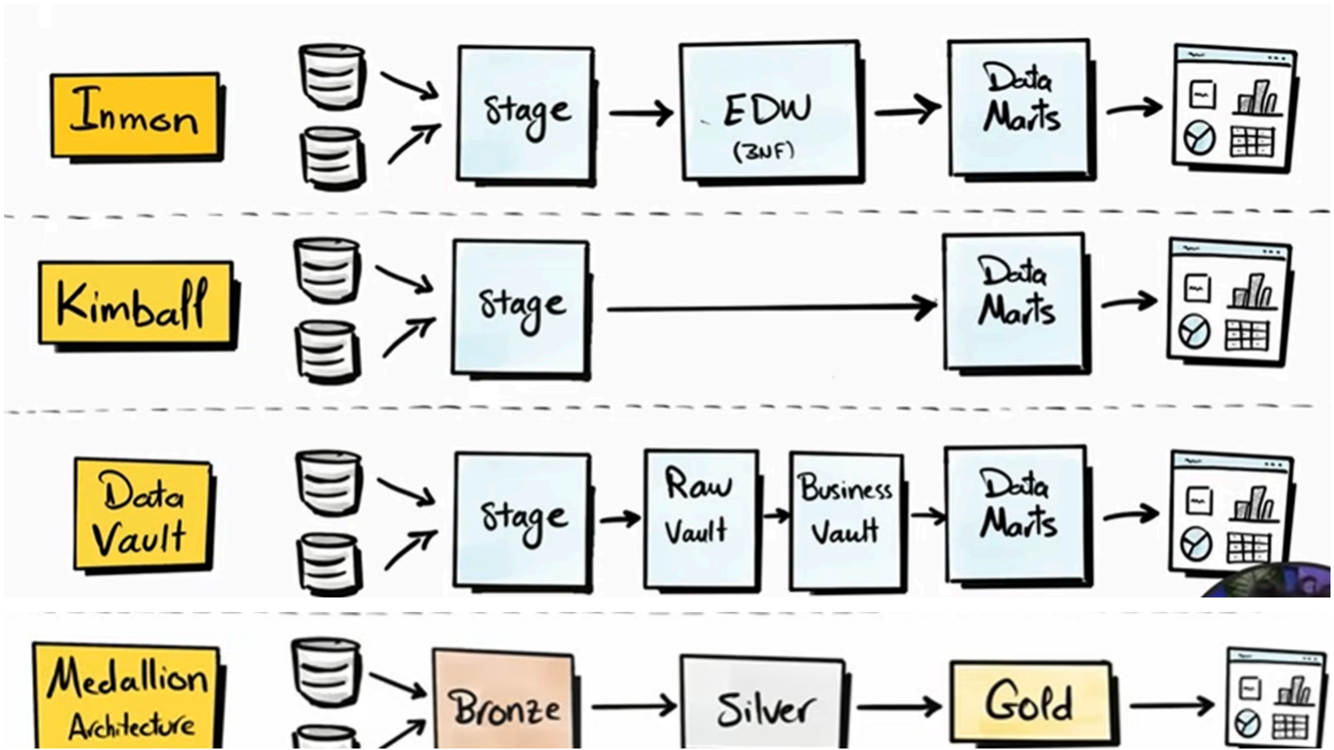

Pull
Extraction
Push
Extraction
Extraction
methods
Full
Extraction
Extract
Types
Incremental
Extraction
Extraction
Manual Data
Extraction
Database
Querying
Extract
Techniques
File Parsing
API Calls
Event Based
Streaming
Web Scraping
CDC



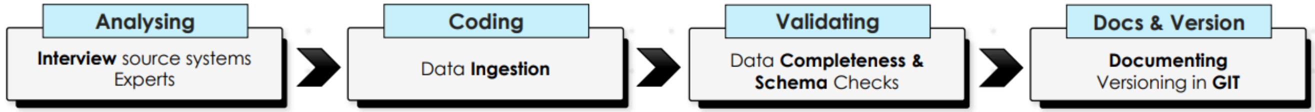
Batch
Processing
Stream
Processing
Processing
Types
Load
Methods
Full
Load
Truncate & Insert
Upsert
Drop, Create, Insert
Incremental
Load
Upsert
Append
Merge
Load
SCD 0
No Historization
SCD 1
Overwrite
SCD 2
Historization
Slowly Changing Dimensions
(SCD)

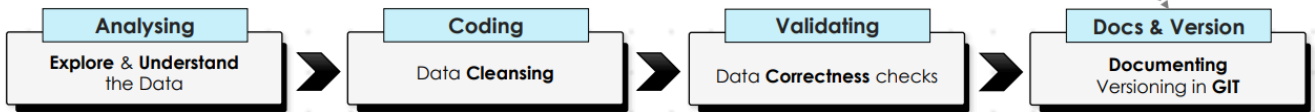

Transformation
Data Enrichment
Data Integration
Derived Columns
Data Normalization & Standardization
Business Rules & Logic
Data Aggregations
Data Cleansing
Remove Duplicates
Data Filtering
Handling Missing Data
Handling Invalid Values
Handling unwanted spaces
Data Type Casting
Outlier Detection


Data Architectures
Data Warehouse
Data Lake
Data Lakehouse
Data Mesh

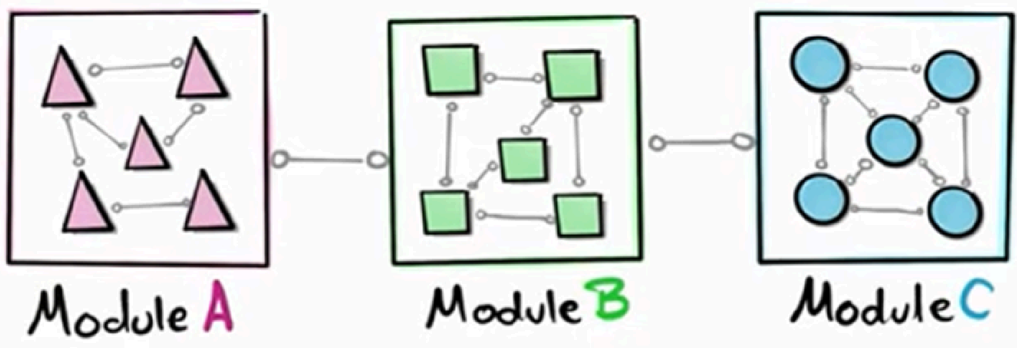


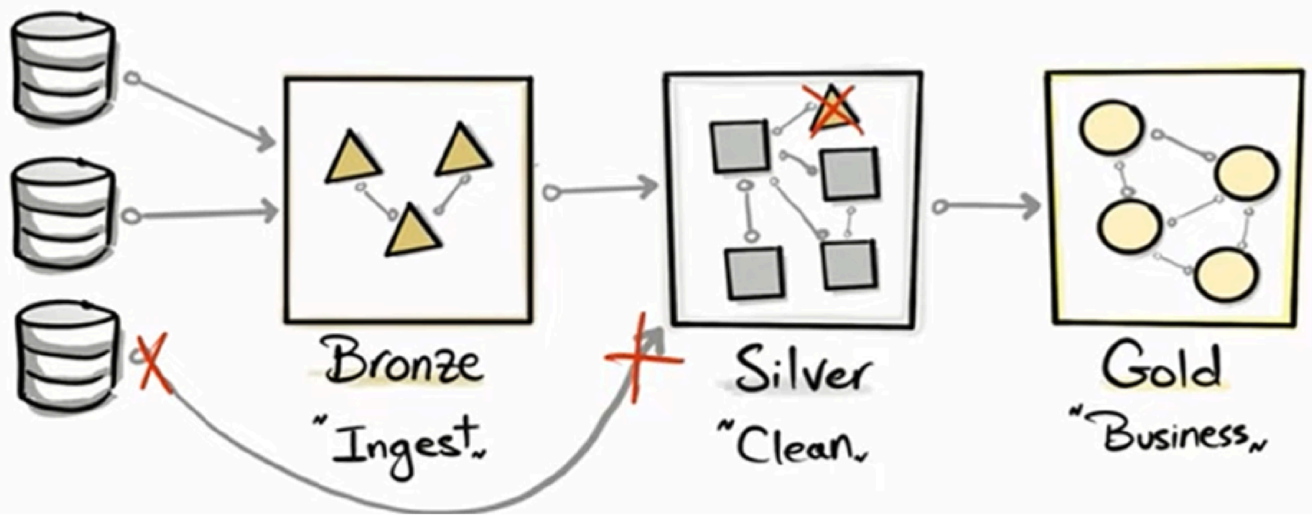
	Bronze Layer	Silver Layer	Gold Layer
Definition	Raw, unprocessed data as-is from sources	Clean & standardized data	Business-Ready data
Objective	Traceability & Debugging	(Intermediate Layer) Prepare Data for Analysis	Provide data to be consumed for reporting & Analytics
Object Type	Tables	Tables	Views
Load Method	Full Load (Truncate & Insert)	Full Load (Truncate & Insert)	None
Data Transformation	None (as-is)	- Data Cleaning - Data Standardization - Data Normalization - Derived Columns - Data Enrichment	- Data Integration - Data Aggregation - Business Logic & Rules
Data Modeling	None (as-is)	None (as-is)	- Start Schema - Aggregated Objects - Flat Tables
Target Audience	- Data Engineers	- Data Analysts - Data Engineers	- Data Analysts - Business Users

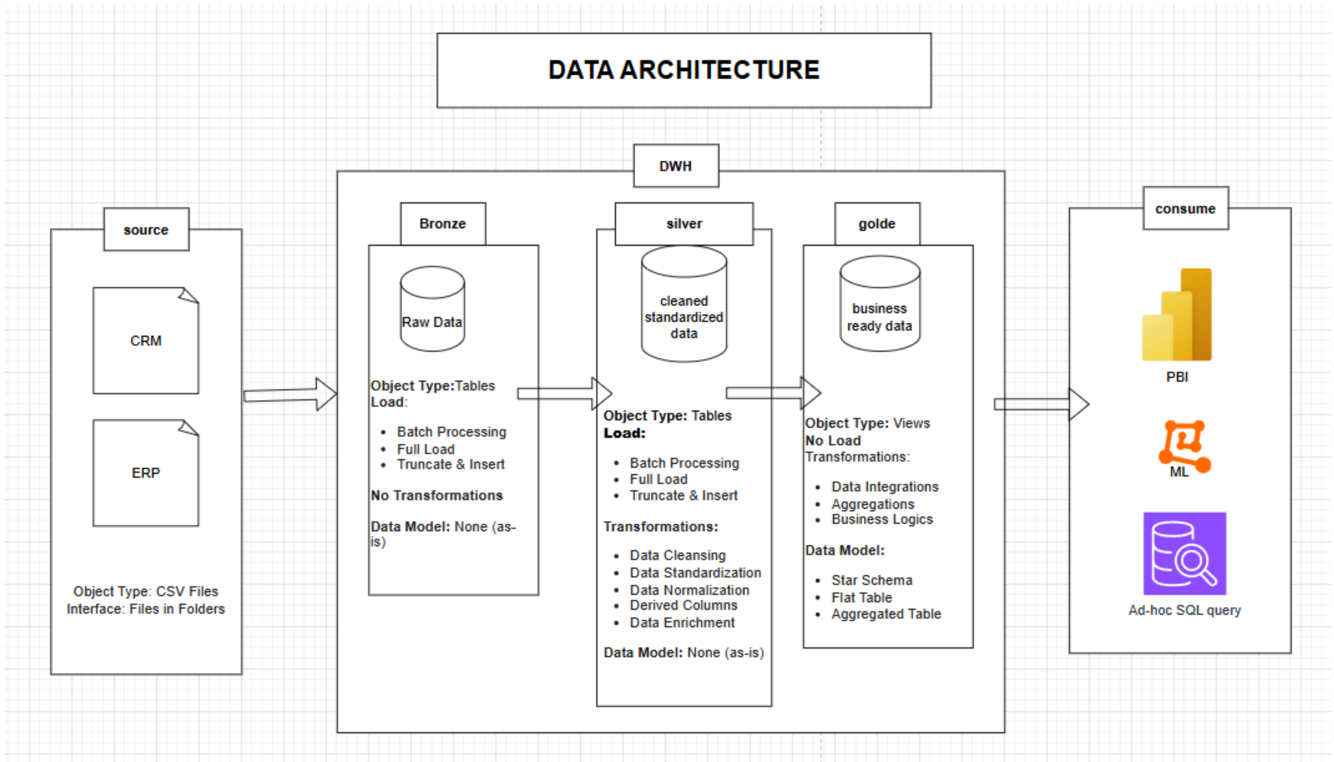

Bronze Layer
Analysing
Interview source systems Experts
Coding
Data Ingestion
Validating
Data Completeness & Schema Checks
Docs & Version
Documenting Versioning in GIT


Silver Layer
Data Flow
Data Integration
Analysing
Explore & Understand the Data
Coding
Data Cleansing
Validating
Data Correctness checks
Docs & Version
Documenting Versioning in GIT

Gold Layer
Data Model
Data Catalog
Data Flow
Analysing
Explore & Understand the Business Objects
Coding
Data Integration
Validating
Data Integration Checks
Docs & Version
Documenting Versioning in GIT


with
SoC
Module A
Module B
Module C


Bronze
"Ingest"
Silver
"Clean"
Gold
"Business"



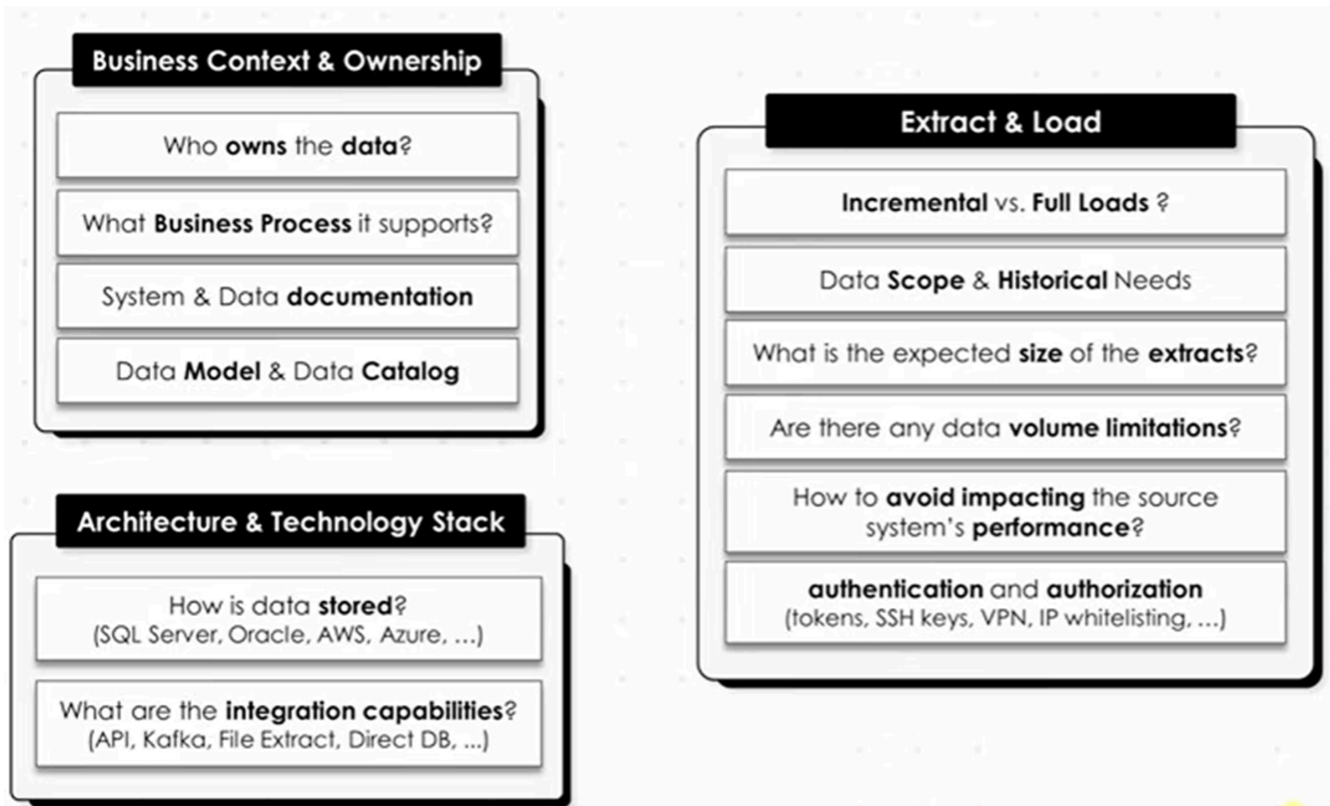
https://github.com/DataWithBaraa/sql-data-warehouse-project?tab=readme-ov-file
https://github.com/DataWithBaraa/sql-data-warehouse-project.git
https://github.com/DataWithBaraa/sql-data-warehouse-project/blob/main/docs/naming_conventions.md

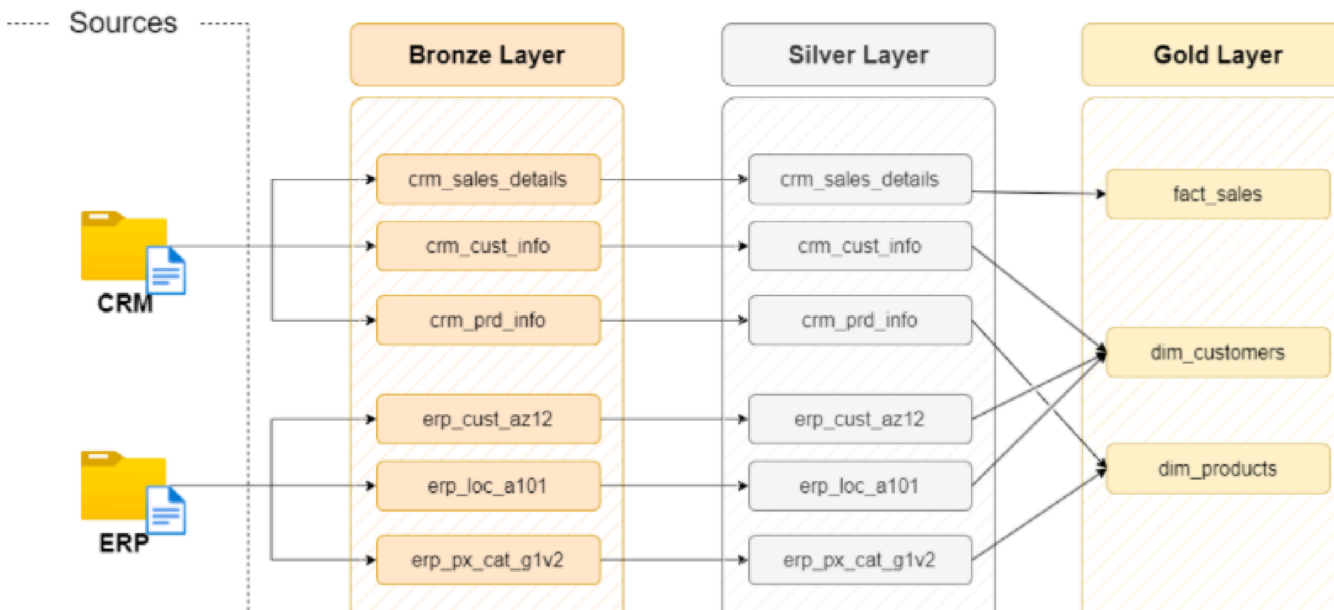
https://www.markdownguide.org/cheat-sheet/

Service Broker
Storage
Security
Users
Roles
Schemas
bronze
db_accessadmin
db_backupoperator
db_datareader

```sql
CREATE SCHEMA bronze;
GO
CREATE SCHEMA silver;
GO
CREATE SCHEMA gold;
GO
```

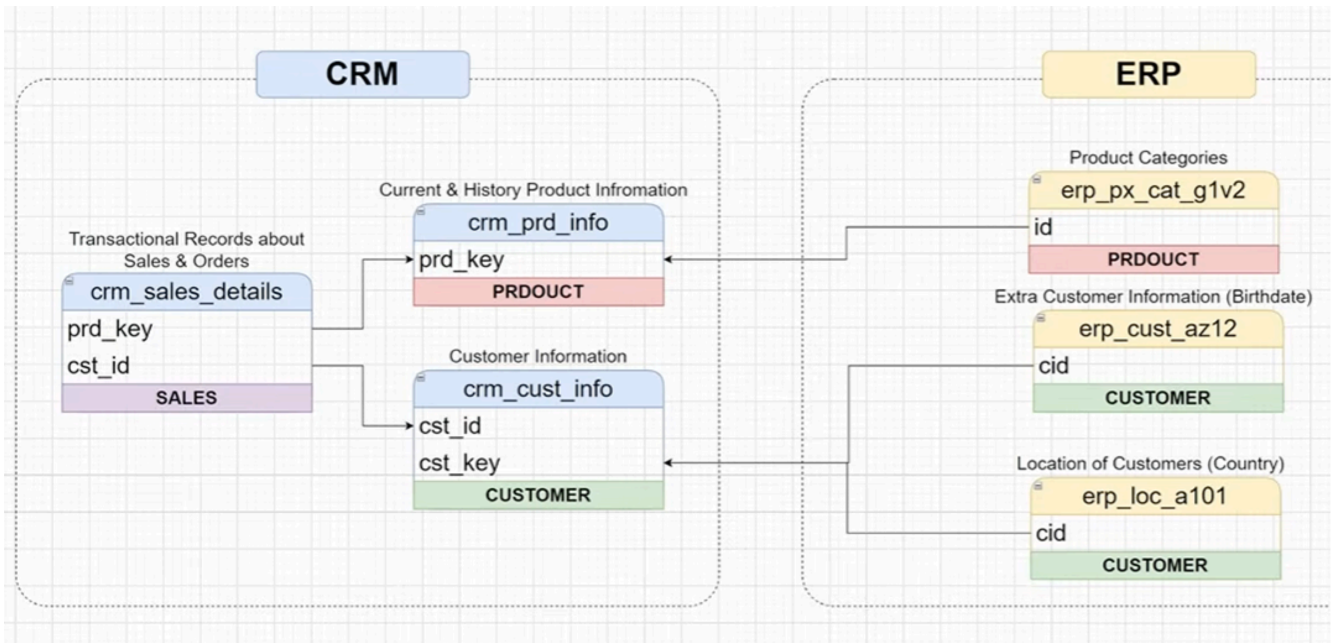
GO separate batches when working with multiple SQL statements



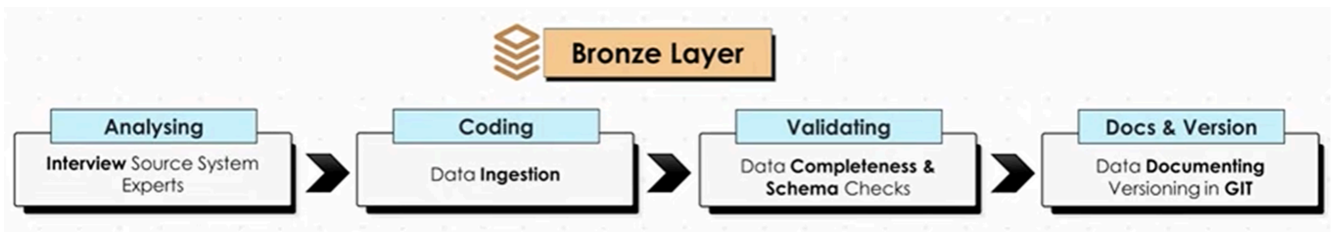


CRM：Customer Relationship Management（客户关系管理），指管理客户信息和互动的系统。

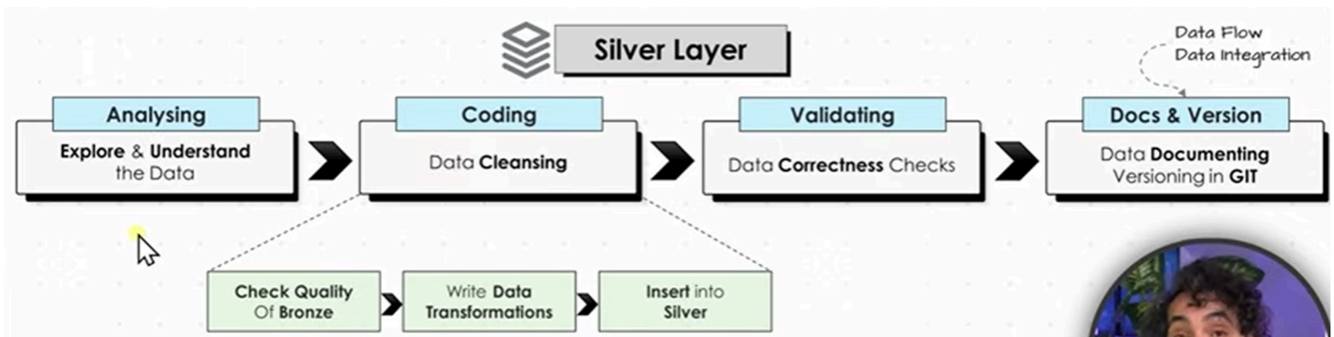
ERP：Enterprise Resource Planning（企业资源计划），指用于管理企业内部资源和业务流程的系统。



25. [24:32:54] Project DWH | Bronze



26. [25:10:09] Project DWH | Silver



METADATA COLUMNS

Extra columns added by data engineers that do not originate from the source data.

create_date : The record's load timestamp.

update_date: The record's last update timestamp.

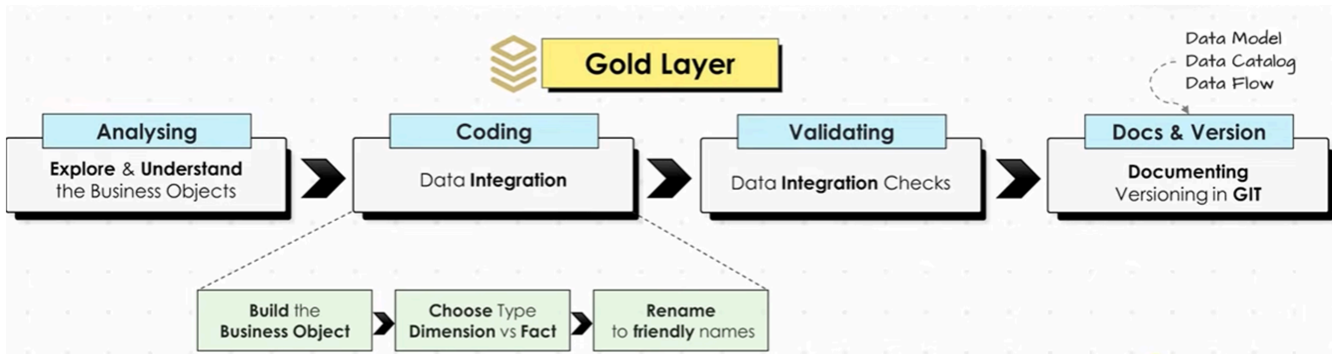
source_system: The origin system of the record.

file_location: The file source of the record.

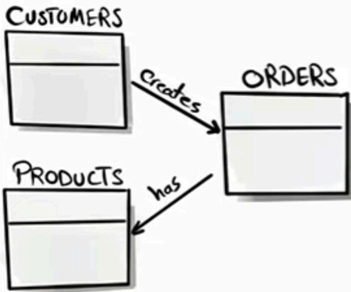
Quality Check

A Primary Key must be unique and not null

27. 26:47:46 Project DWH | Gold

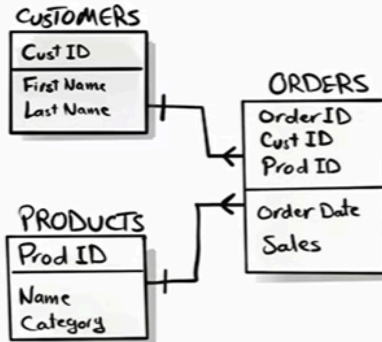


Conceptual
Data Model



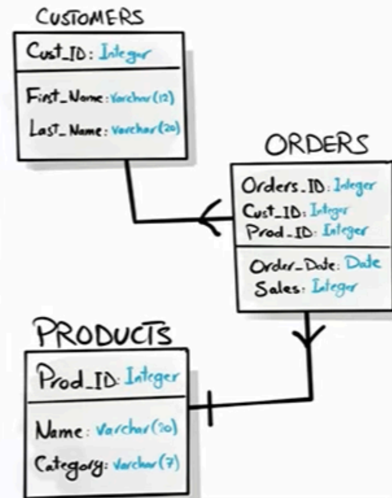
BIG PICTURE

Logical
Data Model



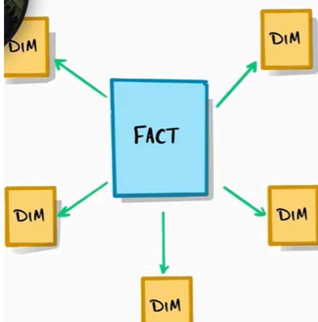
BLUE PRINT

Physical
Data Model



IMPLEMENTATION

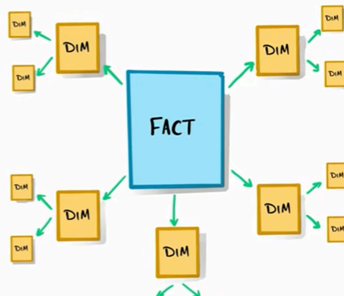
STAR SCHEMA



Simple & Easy

Big Dimensions

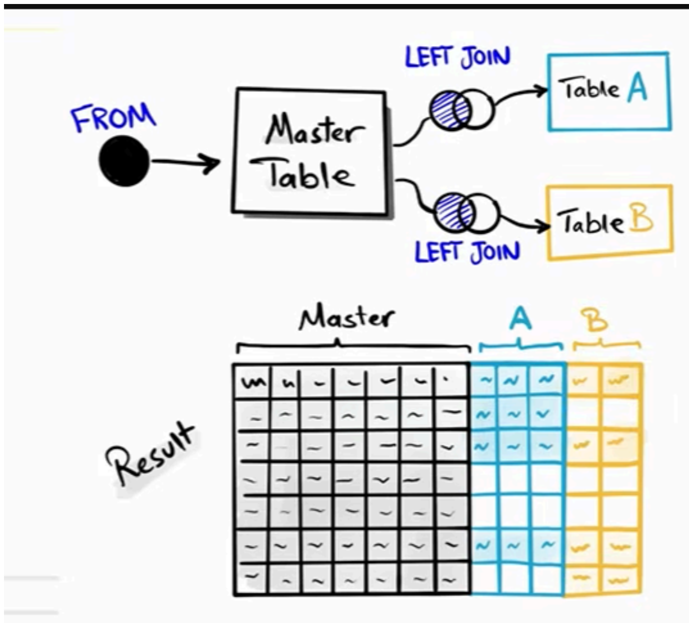
SNOWFLAKE SCHEMA



More Complex

Large Datasets

DIMENSION	FACT
Descriptive information that give context to your data.	Quantitative information that represents events
Who? What? Where?	How much? How many?



	gold.dim_customers
PK	customer_key
	customer_id
	customer_number
	first_name
	last_name
	country
	marital_status
	gender
	birthdate
	country

	gold.fact_sales
	order_number
FK1	product_key
FK2	customer_key
	order_date
	shipping_date
	due_date
	sales_amount
	quantity
	price

	gold.dim_products
PK	product_key
	product_id
	product_number
	product_name
	category_id
	category
	subcategory
	maintenance
	cost
	product_line
	start_date

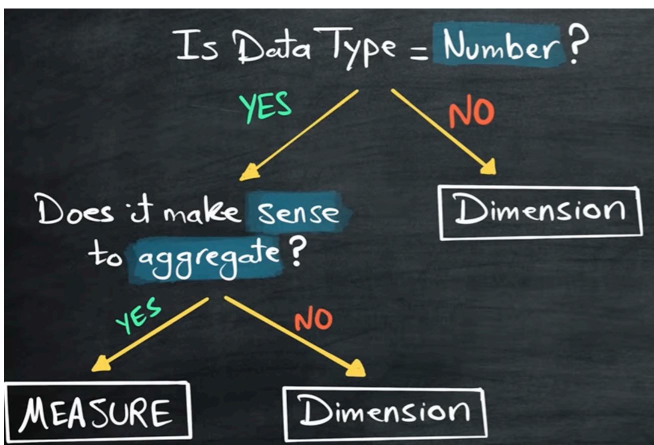
Sales Calculation

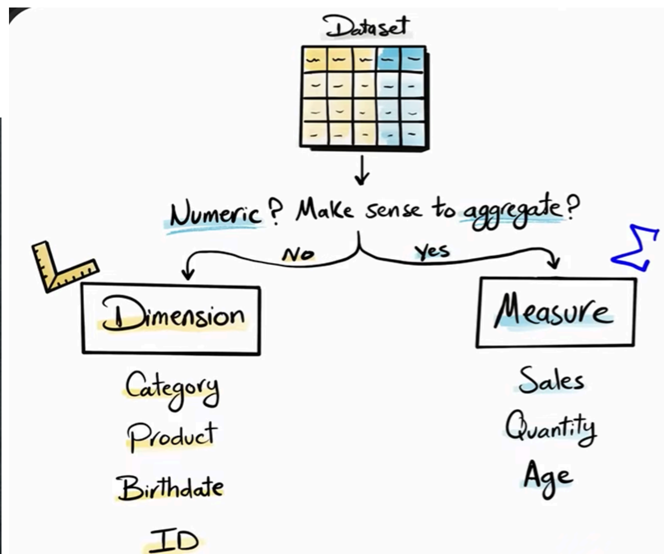
28. 27:41:51 Project: Exploratory Data Analysis (EDA)

#1 OPTION
Create Project Database By Running a Script

#2 OPTION
Improt Flat Files to Database

#3 OPTION
Restore The Project Database

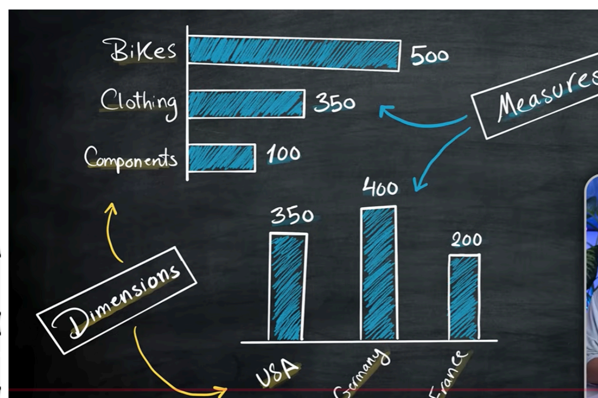


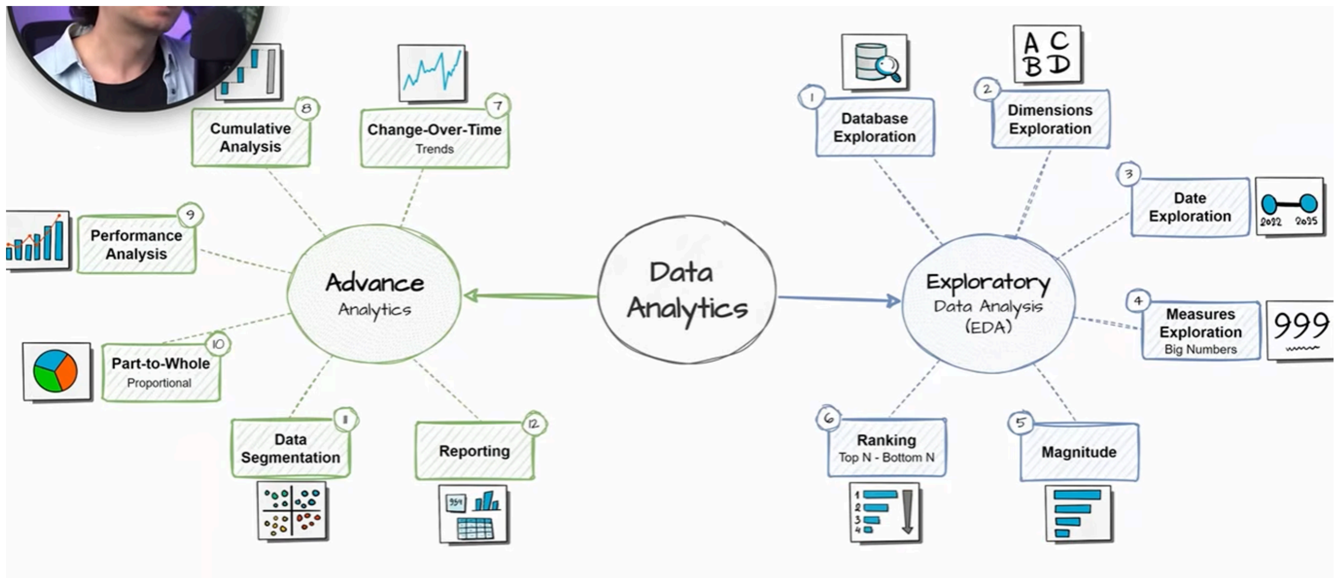


#1 Is it Numeric?

#2 Does it make sense to Aggregate?

ID





Data
Analytics
Advance
Analytics
Exploratory
Data Analysis
(EDA)
Database
Exploration
Dimensions
Exploration
Date
Exploration
Measures
Exploration
Big Numbers
Magnitude
Ranking
Top N - Bottom N
Change-Over-Time
Trends
Cumulative
Analysis
Performance
Analysis
Part-to-Whole
Proportional
Data
Segmentation
Reporting