EPICODE

# Built-in function
# Le view

## AGENDA

- ❑ BUILT IN function
- ❑ VIEW

la fine di questo modulo sarai in grado di:

- ✔ Utilizzare efficacemente le funzioni BUILT IN più comuni
- ✔ Implementare view a supporto delle interrogazioni e della analisi per tool client a valle (Excel, Power BI,....)

## Le BUILT-IN function

Le BUILT-IN function assumono un ruolo fondamentale nella manipolazione dei dati.

Le BUILT-IN function maggiormente utilizzate sono classificate nel seguente modo:

- Date and Time Function
- Aggregate Function (che abbiamo già trattato nel modulo precedente: MIN, MAX, AVG, COUNT...)

Il focus è sulle funzioni di tipo **Date** and **Time**

## Le Built-in function: Date and Time

Le funzioni **Date and Time** consentono di manipolare campo di tipo **DATE, DATETIME E TIMESTAMP**

- **DATE**

Il formato DATE è utilizzato per campi o valori caratterizzati da una sola parte DATE (e non anche TIME...)
Il formato **DATE** ha un struttura **YYYY-MM-DD**
Il range di valori ammesso è compreso tra **1000-01-01** e **9999-12-31**

- **DATETIME**

Il formato DATETIME è utilizzato per campi o valori caratterizzati sia dalla parte DATE che dalla parte TIME
Il formato **DATETIME** ha un struttura **YYYY-MM-DD hh:mm:ss**
Il range di valori ammesso è compreso tra **1000-01-01 00:00:00** e **9999-12-31 23:59:59**

- **TIMESTAMP**

Il formato TIMESTAMP è utilizzato per campi o valori caratterizzati sia dalla parte DATE che dalla parte TIME
Il formato **TIMESTAMP** ha un struttura **YYYY-MM-DD hh:mm:ss**
Il range di valori ammesso è compreso tra **1970-01-01 00:00:01 UTC** e **2038-01-19 03:14:07 UTC**

## Date and Time: YEAR function

Le funzione YEAR restituisce un intero che rappresenta la componente temporale YEAR

```
SELECT YEAR('2024-03-30') AS Year
```

| Year |
|---|
| 2024 |

```
SELECT
OrderDate
, YEAR(OrderDate) AS YearSales
, Sales
FROM Sales
```

| OrderDate | YearSales | Sales |
|---|---|---|
| 2020-05-31 | 2020 | 6959.95 |
| 2020-06-01 | 2020 | 89.97 |

```
SELECT
YEAR(OrderDate) AS YearSales
, SUM(SalesAmount) AS
YearTotalSales
FROM Sales
GROUP BY YEAR(OrderDate)
ORDER BY YearSales DESC
```

| YearSales | Sales |
|---|---|
| 2020 | 12650022.79 |
| 2019 | 32500005.58 |
| 2018 | 24328786.25 |

# Date and Time: MONTH function

Le funzione MONTH restituisce un intero che rappresenta la componente temporale MONTH

```
SELECT YEAR('2024-03-30') AS Month
```

| Month |
|---|
| 3 |

```
SELECT
YEAR(OrderDate)     AS YearSales
, MONTH(OrderDate) AS MonthSales
, SUM(SalesAmount) AS YearTotalSales
FROM Sales
WHERE YEAR(OrderDate) in (2019, 2018)
GROUP BY
YEAR(OrderDate)
, MONTH(OrderDate)
ORDER BY
YearSales DESC
, MonthSales ASC
```

| YearSales | MonthSales | Sales |
|---|---|---|
| 2019 | 1 | 1318592.01 |
| 2019 | 2 | 2386073.19 |
| ... | ... | .. |
| 2018 | 1 | 713229.54 |

1) **FROM**: consente di indicare la tabella oggetto di interrogazione
2) **WHERE**: consente di filtrare le righe della tabella Sales (quelle per le quali la data dell'ordine è nell'anno 2019 e 2018
3) **GROUP BY**: consente di raggruppare le righe della tabella Sales per le quali la data ordine è nell'anno 2019 e 2018, per anno e mese
4) **SELECT**: indica I campi da esporre nel result set
5) **ORDER BY**: consente di imporre un ordinamento per le righe in output

# Date and Time: MONTH function

Le funzione MONTH restituisce un intero che rappresenta la componente temporale MONTH

```
SELECT YEAR('2024-03-30') AS Month
```

| Month |
|---|
| 3 |

```
SELECT
YEAR(OrderDate)     AS YearSales
, MONTH(OrderDate) AS MonthSales
, SUM(SalesAmount) AS YearTotalSales
FROM Sales
WHERE YEAR(OrderDate) in (2019, 2018)
GROUP BY
YEAR(OrderDate)
, MONTH(OrderDate)
ORDER BY
YearSales DESC
, MonthSales ASC
```

| YearSales | MonthSales | Sales |
|---|---|---|
| 2019 | 1 | 1318592.01 |
| 2019 | 2 | 2386073.19 |
| ... | ... | .. |
| 2018 | 1 | 713229.54 |

1) **FROM**: consente di indicare la tabella oggetto di interrogazione
2) **WHERE**: consente di filtrare le righe della tabella Sales (quelle per le quali la data dell'ordine è nell'anno 2019 e 2018
3) **GROUP BY**: consente di raggruppare le righe della tabella Sales per le quali la data ordine è nell'anno 2019 e 2018, per anno e mese
4) **SELECT**: indica I campi da esporre nel result set
5) **ORDER BY**: consente di imporre un ordinamento per le righe in output

## Date and Time: DATEDIFF function

Le funzione DATEDIFF restituisce il numero di giorni compresi tra due date

*Esempio: quanti giorni intercorrono dal momento in cui l'ordine è registrato al momento in cui avviene l'effettiva spedizione? È possibile ottimizzare il processo di evasione dell'ordine?*

```
SELECT
SalesOrderNumber
, SalesOrderLineNumber
, OrderDate
, ShipDate
, DATEDIFF(OrderDate, ShipDate) AS DeliveryTime
FROM Sales
```

| SalesOrderNumber | SalesOrderLine Number | OrderDate | ShipDate | DeliveryTime |
|---|---|---|---|---|
| SO43659 | 1 | 2017-07-01 | 2017-07-08 | 7 |
| SO43659 | 2 | 2017-07-01 | 2017-07-09 | 8 |
| ... | ... | .. | | |
| SO43660 | 1 | 2017-07-01 | 2017-07-10 | 9 |

## … altre BUILT-IN function: COALESCE

In alcuni scenari, è utile comparare più colonne al fine di determinare il primo valore non-NULL.
Le funzione COALESCE restituisce il primo valore non-NULL della lista passata come argomento (della funzione stessa).

*Esempio: è necessario monitorare la disponibilità dei prodotti. Un prodotto è disponibile se ha una data di inizio validità (StartDate) e data di fine validità NULL (EndDate). Un prodotto non è più disponibile (non vendibile) se la data di fine validità (EndDate) è valorizzata (.. è indicato appunto fino a quando è stato disponibile il prodotto..). IN questo caso, è necessario esporre la data in cui ciascun prodotto ha subito l'ultimo aggiornamento..*

```
SELECT
ProductKey
, Product
, StartDate
, EndDate
, COALESCE(StartDate, EndDate) LastUpdated
FROM Product
```

| ProductKey | Product | StartDate | EndDate | LastUpdated |
|---|---|---|---|---|
| 248 | HL Road Frame | 2012-07-01 | 2008-12-27 | 2012-07-01 |
| 249 | HL Road Frame | 2013-07-01 | NULL | 2013-07-01 |

## … altre BUILT-IN function: IFNULL

La funzione IFNULL restituisce il valore specificato (come secondo argomento) se l'espressione (passata come primo argomento della funzione) restituisce NULL

```
SELECT
ProductID
, ProductNumber
, Color                 AS Color_old
, IFNULL(Color, 'NA')   AS Color
FROM Product
```

| ProductID | ProductNumber | Color_old | Color |
|---|---|---|---|
| 800 | BK-R64Y-44 | Yellow | Yellow |
| 802 | FK-1639 | NULL | NA |
| 823 | RW-M423 | Black | Black |

## … altre BUILT-IN function: IFNULL

La funzione IFNULL restituisce il valore specificato (come secondo argomento) se l'espressione (passata come primo argomento della funzione) restituisce NULL.

```
SELECT
ProductID
, Product
, Category AS Category_old
, IFNULL(Category, 'NO CAT') AS Category
FROM Product AS P
LEFT JOIN Category AS C
ON P.CategoryKey = C.CategoryKey
```

| ProductID | Product | Category_old | Category |
|---|---|---|---|
| 209 | Rear Derailleur Cage | NULL | NO CAT |
| 210 | HL Road Frame - Black, 58 | Road Frames | Road Frames |

## VIEW

Una VIEW in SQL è una 'tabella virtuale' che espone il result set di uno statement SQL.

È possibile definire una VIEW come un 'modo' per memorizzare una certa interrogazione in modo tale che sia sempre disponibile (...senza la necessità di doverla riscrivere).

Una VIEW si crea con lo statement CREATE VIEW.

Una VIEW puo' essere utilizzata come sorgente dati per un tool di BI (in alternativa alle tabelle del db) al quale è possibile quindi restituire un dataset 'pulito'.
Il vantaggio consiste nel delegare le operazioni di data preparation alla sorgente dati.

Trade-off: utilizziamo tool di ETL con Power Query (Excel, Power BI) per la preparazione dei dati (propedeutica alla fase di data viz) o  prepariamo le strutture dati nella sorgente (view SQL per esempio...)?

```sql
CREATE VIEW YearCategorySales
AS (
SELECT
YEAR(S.OrderDate) AS YearSales
, C.Category
, SUM(SalesAmount) AS Sales
, SUM(OrderQuantity) AS Quantity
, COUNT(SalesOrderLineNumber) AS Orders
FROM Sales AS S
INNER JOIN Product AS P
ON S.ProductKey = P.ProductKey
INNER JOIN Category AS C
ON P.CategoryKey = C.CategoryKey
GROUP BY
YEAR(S.OrderDate)
, C.CategoryName
)
```

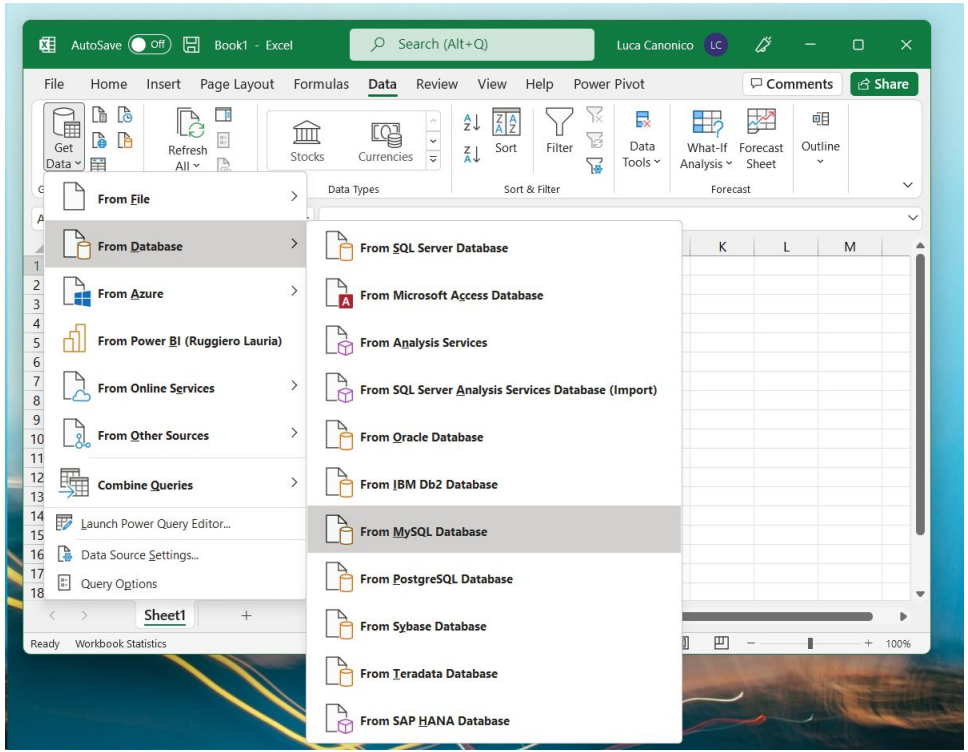
## VIEW

```
CREATE VIEW YearCategorySales
AS (
SELECT
YEAR(S.OrderDate) AS YearSales
, C.Category
, SUM(SalesAmount) AS Sales
, SUM(OrderQuantity) AS Quantity
, COUNT(SalesOrderLineNumber) AS Orders
FROM Sales AS S
INNER JOIN Product AS P
ON S.ProductKey = P.ProductKey
INNER JOIN Category AS C
ON P.CategoryKey = C.CategoryKey
GROUP BY
YEAR(S.OrderDate)
, C.CategoryName
)
```

```
SELECT *
FROM YearCategorySales
ORDER BY YearSales DESC, Sales DESC
```

| | YearSales | Category | Sales | Quantity | Orders |
|---|---|---|---|---|---|
| 1 | 2020 | Touring Bikes | 3859822.30 | 4316 | 1644 |
| 2 | 2020 | Road Bikes | 3709859.31 | 4405 | 1538 |
| 3 | 2020 | Mountain Bikes | 3137860.01 | 3492 | 1237 |
| 4 | 2020 | Mountain Frames | 652911.29 | 1938 | 810 |
| 5 | 2020 | Touring Frames | 428545.40 | 1010 | 423 |
| 6 | 2020 | Road Frames | 252535.89 | 703 | 309 |
| 7 | 2020 | Jerseys | 117919.03 | 3785 | 730 |
| 8 | 2020 | Shorts | 91999.07 | 2271 | 401 |
| 9 | 2020 | Vests | 72294.19 | 1933 | 317 |
| 10 | 2020 | Bike Racks | 61644.00 | 864 | 158 |
| 11 | 2020 | Cranksets | 58754.37 | 303 | 121 |
| 12 | 2020 | Pedals | 39702.13 | 1055 | 439 |
| 13 | 2020 | Helmets | 38923.00 | 1870 | 354 |

## Get Data in Excel

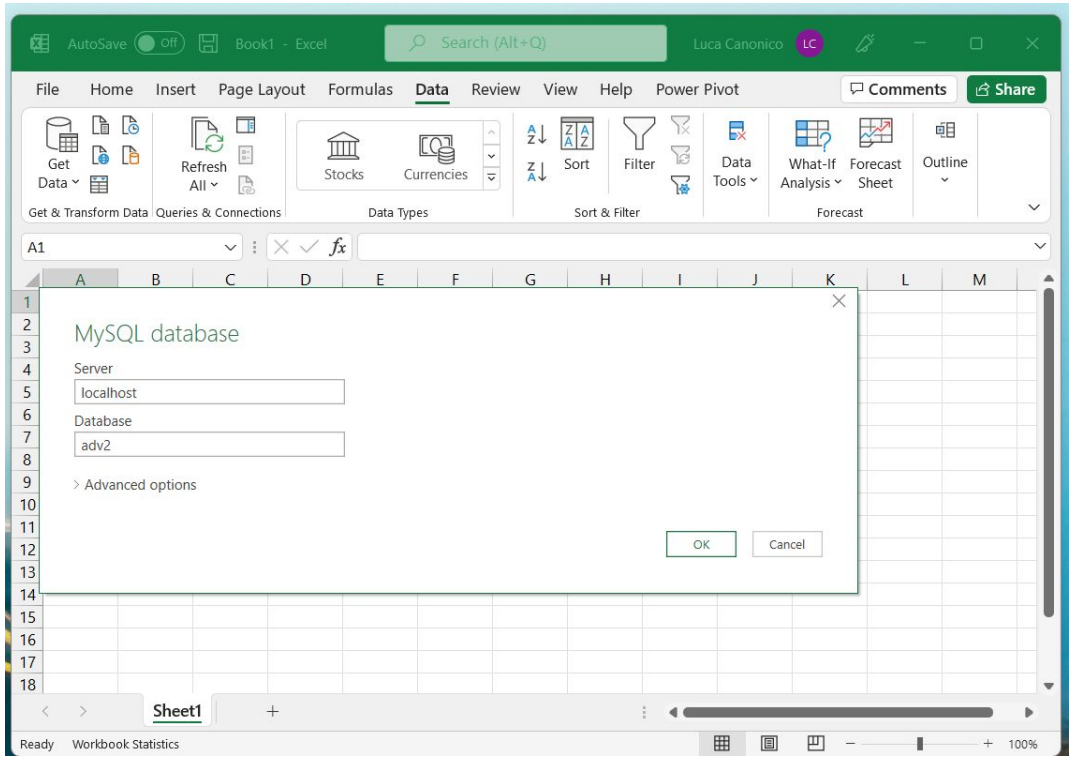



Per utilizzare MySQL come sorgente dati per Excel e Power BI occorre installare il seguente connettore:

https://dev.mysql.com/downloads/file/?id=526410

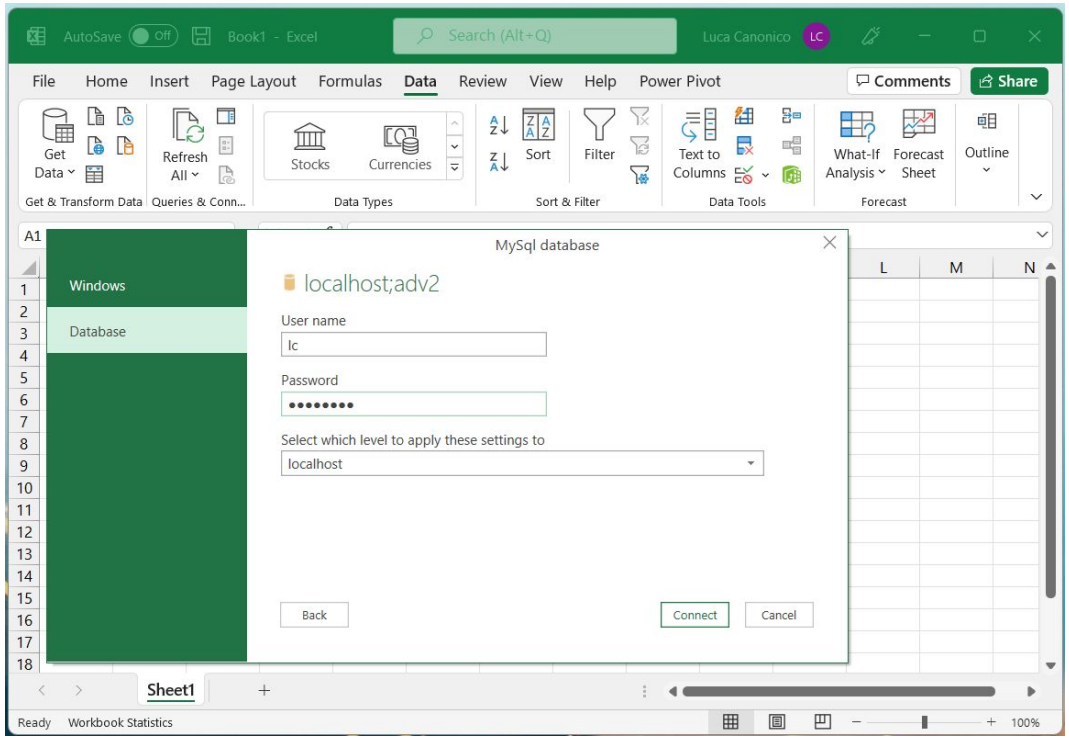
Se si utilizza SQL Server come DBMS non è necessario..!

## Get Data in Excel




Indicare indirizzo server (host) o l'alias del server oppure **localhost** e il nome del Database

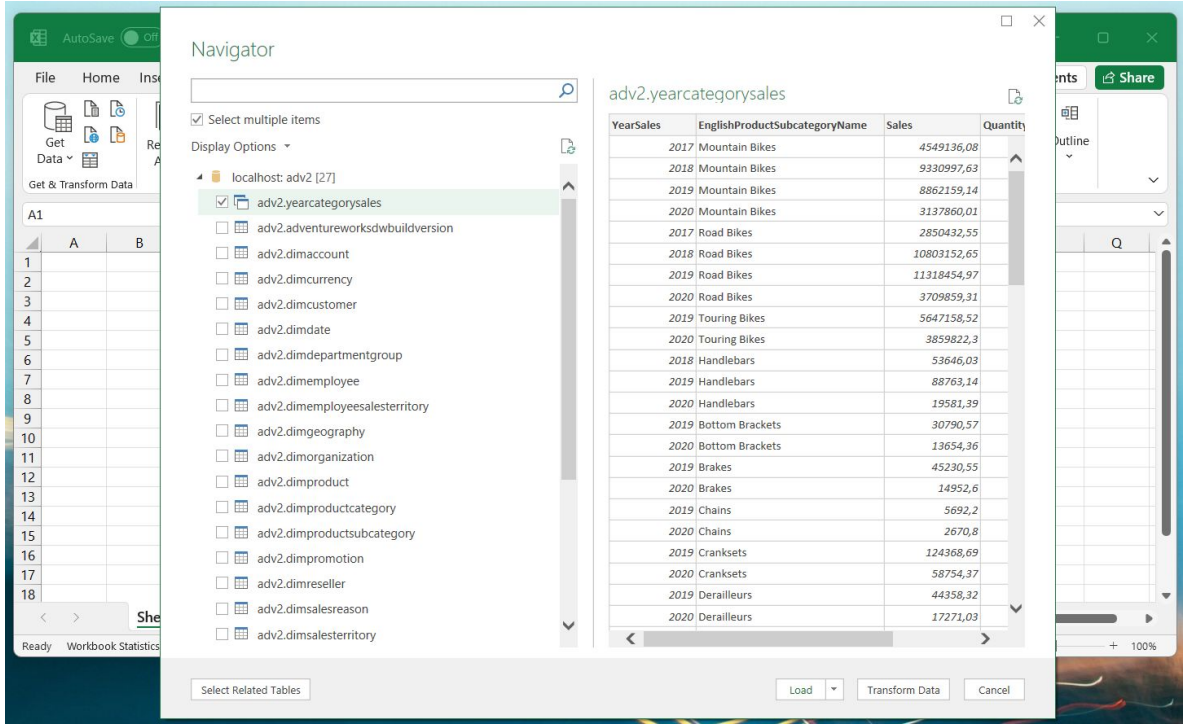
## Get Data in Excel




Scegliere come tipologia di utenza **Database** e indicare **User name** e **Password.**.

Infine, **Connect**

# Get Data in Excel




A questo punto, utilizzando **Load**, è possibile caricare i dati in un foglio Excel o in un modello logico di Excel (Power Pivot) opportunamente progettato per supportare la costruzioni di report in Excel.

È possibile utilizzare il Power Query Editor, cliccando su **Transform Data,** per ulteriori trasformazioni o per attività di profilazione.

## Get Data in Excel

In generale, un processo di sviluppo di una soluzione di Data Analysis prevede le seguenti fasi:

1) **Analisi dei requisiti, comprensione dello scenario, individuazione delle sorgenti dati.** Questa è una fase delicata che prevede la raccolta dei requisiti funzionali che devono essere chiari, concisi e non ambigui.
2) **Acquisizione e preparazione dei dati** utilizzando tool low-code come il Power Query. Le strutture dati, se si ha la possibilità di farlo (è molto probabile avere utenti con permessi limitati o in sola lettiura), possono essere costruite e preparate anche nella sorgente dati (è la best practice). Se non è possibile farlo diventa estremamente utile completare questa fase nel tool di BI a valle.
3) **Modellazione logica** prevede lo sviluppo di un modello semantico a supporto dell'analisi e visualizzazione dei dati
4) **Analisi, esplorazione e visualizzazione** in report e/o dashboard

*Le slide a questo punto del corso mostrano l'acquisizione dei dati in Excel.*

*Le stesse fasi di sviluppo verranno implementate in maniera molto più approfondita nei successivi moduli dedicati agli strumenti di BI (Looker e soprattutto Power BI).*

# SQL

LET'S TAKE A LOOK!

EPICODE

# GRAZIE

Epicode