Programma 21 ott2015

- Recap SQL
- Esercizio Vendite – Query Complesse
- Teoria Join
- Data Profiling
- Esercizio Data Profiling

RECAP QUERY SQL

Select

È l'istruzione più usata: serve ad interrogare i DB

Vesione **Base**:
SELECT : che colonne vuoi selezionare ('*' per tutte)
FROM : da che tabella
WHERE : quali righe vuoi prendere
(condizioni complesse con OR, AND, NOT, IN)

Molto simile all'inglese, è importante comprenderne la logica

RECAP QUERY SQL

Select

È l'istruzione più usata: serve ad interrogare i DB

Versione **Completa**:

SELECT :
FROM :
WHERE :
GROUP BY : raggruppa i valori univoci
HAVING : effettua filtri sulle righe raggruppate (es. sulle sum)
ORDER BY : ordina i valori

Ripristino DB Vendite

Creazione e riempimento DB:

https://www.dropbox.com/s/u1i2c2fic3lqpun/SIA_20151014.sql?dl=0

nb: in base al SQL Server installato, può essere necessario cambiare il formato di date da yyyy-mm-dd a dd/mm/yyyy

Query:

https://www.dropbox.com/s/sa66jygws8blrso/SIA_20151014_Queries.sql?dl=0

JOIN

Ipotizziamo di avere le informazioni in due tabelle diverse che dobbiamo unire.
Esempio:

- Legare le informazioni degli studenti con i gruppi a cui sono iscritti, cioè sapere in che gruppo è iscritto ogni studente
- Sono su due tabelle diverse, ma hanno alcuni campi in comune
- Bisogna scegliere il campo migliore che di solito è la chiave (email)

JOIN

Sintassi:
Select [elenco campi]
From
Tabella1 join Tabella2 on [condizioni di uguaglianza]
(in questo caso è l'email perché dovrebbe essere campo univoco)
in pratica stiamo dicendo: prendi i campi da entrambe le tabelle a patto che il valore indicato nella condizione sia uguale)

INNER JOIN

Mostra solo le righe per le quali la condizione è rispettata da entrambi i lati (ovvero il valore deve esistere in entrambe le tabelle

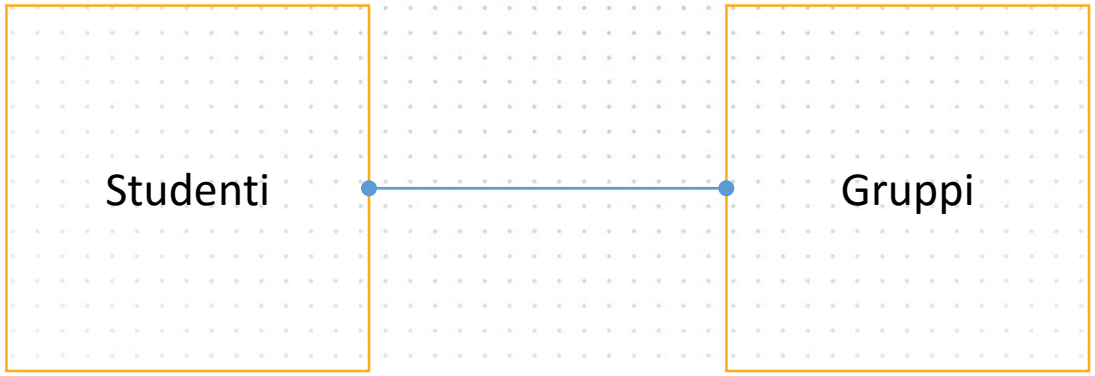
Sintassi:

Select [elenco campi]

From

Tabella1 **inner join** Tabella2 on [condizioni di uguaglianza]

Es: select nome, gruppo from studenti s inner join gruppi g on s.mail = g.mail



LEFT OUTER JOIN

Mostra tutte le righe della tabella da cui parte il join (left), se trova corrispondenza mostra i valori, altrimenti lascia NULL

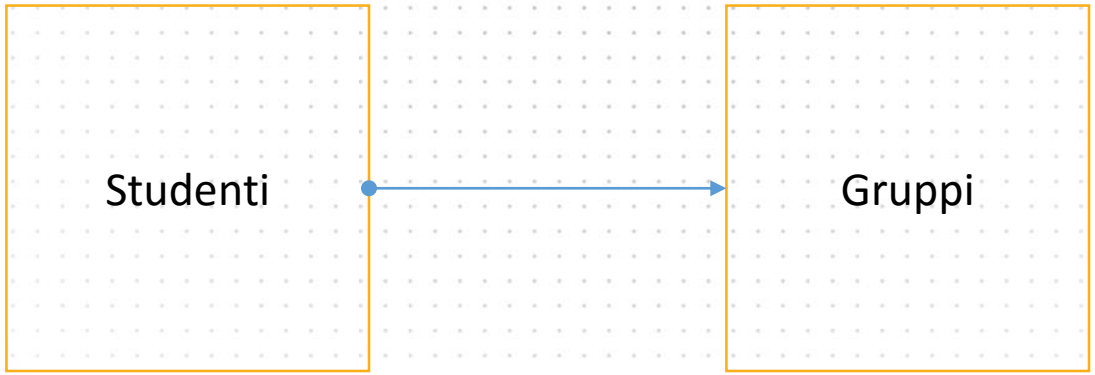
Sintassi:

Select [elenco campi]

From

Tabella1 **left outer join** Tabella2 on [condizioni di uguaglianza]

Es: select nome, gruppo from studenti s left outer join gruppi g on s.mail = g.mail
(in questo caso vediamo tutti gli studenti, e se sono associati ad un gruppo)



Right Outer: uguale ma parte dalla seconda tabella indicata
Full Outer: right+left outer contemporaneamente, estrae tutte le righe di entrambe le tabelle.

Data Profiling

Definizione:

Data profiling *is the process of examining the data available in an existing data source (e.g. a database or a file) and collecting statistics and information about that data. The purpose of these statistics may be to:*

- *Find out whether existing data can easily be used for other purposes*
- *Improve the ability to search the data by tagging it with keywords, descriptions, or assigning it to a category*
- *Give metrics on data quality including whether the data conforms to particular standards or patterns*
- *Assess the risk involved in integrating data for new applications, including the challenges of joins*
- *Discover metadata of the source database, including value patterns and distributions, key candidates, foreign-key candidates, and functional dependencies*
- *Assess whether known metadata accurately describes the actual values in the source database*
- *Understanding data challenges early in any data intensive project, so that late project surprises are avoided. Finding data problems late in the project can lead to delays and cost overruns.*
- *Have an enterprise view of all data, for uses such as master data management where key data is needed, or data governance for improving data quality.*

Data Profiling

Data profiling è l'esame dei dati disponibili in un db con lo scopo di recuperare informazioni sullo stesso.

Nel nostro caso:

a. Capire quali sono i dati e cosa rappresentano

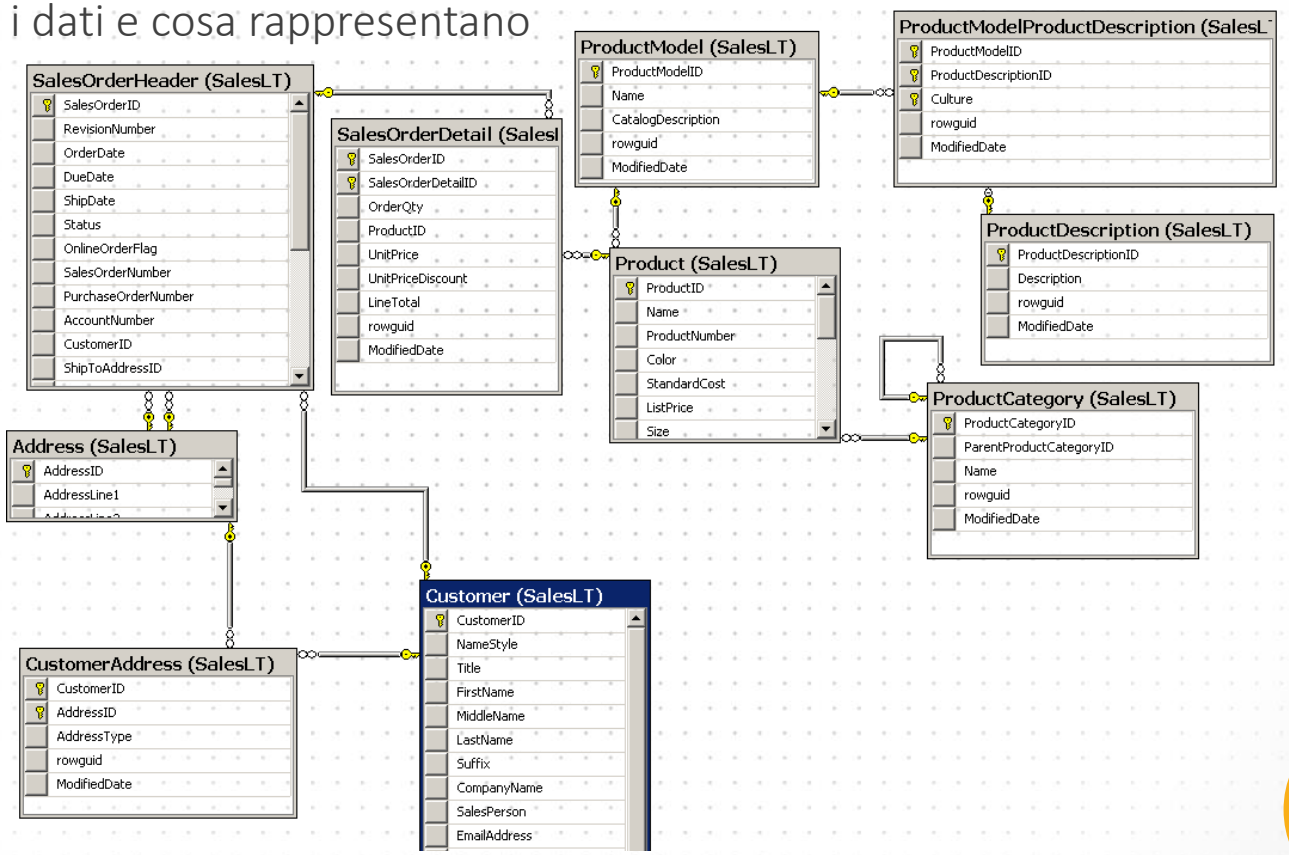
b. Capire come sono i dati in termini numerici per individuare criticità di performance

c. Capire le relazioni tra I dati:
   - Chiavi
   - Valori Null
   - Gerarchie

Data Profiling

Esercizio:

a. Capire quali sono i dati e cosa rappresentano



Per problemi su diagrammi: ALTER AUTHORIZATION ON DATABASE::AdventureWorks_Lite TO [sa]

Data Profiling

Esercizio:

a. Capire quali sono i dati e cosa rappresentano

1. SalesOrderHeader: sono le informazioni comuni in ogni ordine (sono sempre uguali per ogni riga)
2. SalesOrderDetails: sono le righe degli ordini, in particolare vengono descritte le condizioni di vendita per ogni prodotto venduto
3. Product: è l'anagrafica dei prodotti
4. Customers: è l'anagrafica dei clienti
5. …

Quando non si ha la certezza queste possono essere supposizioni che devono essere confermate dalle relazioni tra le tabelle (FK/join ammissibili). Tecnicamente siamo nell'ambito del reverse engineering.

Data Profiling

Esercizio:

b. Capire come sono i dati in termini numerici per individuare criticità di performance

Select count(*)
From SalesOrderDetail

Di solito le tabelle più popolate sono quelle delle transazioni (transaction data) non le anagrafiche (reference data).
Quando la numerosità dei dati eccede le capacità dei sistemi che abbiamo a disposizione si deve analizzare quale è il set minimo di informazioni che servono.
Es.: per vedere un analisi delle vendite è necessario avere tutte le single righe fattura? Oppure è sufficiente vedere le vendite per cliente/prodotto/mese?

Data Profiling

c. Capire le relazioni tra I dati:

- Chiavi
- Valori Null
- Gerarchie

--Partiamo dagli ordini: Cerchiamo intanto di capire cosa c'è dentro gli indirizzi
*select * from SalesLT.SalesOrderHeader where ShipToAddressID<>BillToAddressID --- NESSUNO: l'indirizzo di spedizione è sempre uguale a quello di fatturazione*

--vediamo se l'anagrafica degli indirizzi è consistente
*select * from SalesLT.SalesOrderHeader Z*
where ShipToAddressID not in (select [AddressID] from [SalesLT].[CustomerAddress] X where x.CustomerID=z.CustomerID)
--- NESSUNO : quindi ShipToAddressID/BillToAddressID sono tra quelli specificati in [CustomerAddress]

```
-- MA ALLORA A COSA SERVE, COME POSSIAMO USARE la TABELLA [CustomerAddress] ?
 SELECT [CustomerID]
 FROM  [SalesLT].[CustomerAddress]
 GROUP BY [CustomerID]
 HAVING COUNT(DISTINCT [AddressID])>1
 -- SI CI SONO ALCUNI CUSTOMER CON PIù INDIRIZZI, QUINDI *** NON *** VALE fd customerID --> ADDRESSid

SELECT [AddressID]
 FROM SalesLT].[CustomerAddress]
 GROUP BY [AddressID]
 HAVING COUNT(DISTINCT [CustomerID])>1
 -- VUOTA, QUINDI VALE fd ADDRESSid --> customerID
 -- cioè la chiave specificata nello schema in realtà è una superchiave,
 -- la chiave è solo address
```

```
--cerchiamo di capire se ci sono clienti con più di un indirizzo per tipo

SELECT
[CustomerID]      ,[AddressType]
  FROM [SalesLT].[CustomerAddress]
GROUP BY [CustomerID], [AddressType]
HAVING COUNT(*)>1
-- vuota, ok [CustomerID]      ,[AddressType] : un customer ha - per un tipo di indirizzo solo un address ...
-- Ho un'altra chiave (diciamo alternativa): [CustomerID] ,[AddressType]

-- che tipi di indirizzo ci sono?
SELECT  DISTINCT [AddressType]  FROM [SalesLT].[CustomerAddress] -- SOLO 2
-- 2 tipi: Sede centrale e sede periferica

-- riassumiamo
  select * FROM SalesLT.Customer  -- ho 847 clienti
  select distinct [CustomerID] FROM [SalesLT].[CustomerAddress]
-- 407 di questi hanno almeno un indirizzo
```

```
-- Per ottenere quelli che hanno 2 indirizzi
   SELECT [CustomerID]
  FROM [SalesLT].[CustomerAddress]
  GROUP BY [CustomerID]
  HAVING COUNT(DISTINCT [AddressID]) = 2

  --- sono 10

  -- Non ci possono essere customer con più di due indirizzi perchè abbiamo solo due tipi [AddressType]
  --- e la coppia [CustomerID]      ,[AddressType] è chiave ...
  -- verifichiamo

-- Per ottenere quelli che hanno >2 indirizzi
  SELECT [CustomerID]
  FROM  [SalesLT].[CustomerAddress]
  GROUP BY [CustomerID]
  HAVING COUNT(DISTINCT [AddressID]) >2
---- >>>>>>>>>>>>> NESSUNO
```

```
-- Per ottenere quelli che hanno 2 indirizzi
   SELECT [CustomerID]
  FROM [SalesLT].[CustomerAddress]
  GROUP BY [CustomerID]
  HAVING COUNT(DISTINCT [AddressID]) = 2

  --- sono 10

  -- Non ci possono essere customer con più di due indirizzi perchè abbiamo solo due tipi [AddressType]
  --- e la coppia [CustomerID]      ,[AddressType] è chiave ...
  -- verifichiamo

-- Per ottenere quelli che hanno >2 indirizzi
  SELECT [CustomerID]
  FROM  [SalesLT].[CustomerAddress]
  GROUP BY [CustomerID]
  HAVING COUNT(DISTINCT [AddressID]) >2
---- >>>>>>>>>>>>> NESSUNO
```

-- IN QUESTO DB CHE FARE CON la TABELLA [CustomerAddress] ?

-- NON SI USA, visto che abbiamo ShipToAddressID e BillToAddressID nell'ordine

Oppure potrebbe essere considerata come una SLOW CHANGING DIMENSION:
(Si vedrà successivamente)

-- IN QUESTO DB CHE FARE CON la TABELLA [CustomerAddress] ?

-- NON SI USA, visto che abbiamo ShipToAddressID e BillToAddressID nell'ordine

Oppure potrebbe essere considerata come una SLOW CHANGING DIMENSION:
(Si vedrà successivamente)

```
-- oppure cerco di ricavare per ogni customer il suo (eventuale) IMO (Indirizzo Main Office e il suo eventuale ISH (Indirizzo Shipping)


SELECT     c.CustomerID,
                                MIN(case when ca.AddressType = 'Main Office' then AddressID
                                else '99999' end) as MainOfficeAddress,

                                MIN(case when ca.AddressType = 'Shipping' then AddressID
                                else '99999' end) as ShippingAddress,

                                count(*)
FROM       SalesLT.Customer AS c INNER JOIN
             SalesLT.CustomerAddress AS ca ON c.CustomerID = ca.CustomerID

GROUP BY   c.CustomerID

ORDER BY   COUNT(*) desc

-- in pratica prendo ogni customerid e isolo in due colonne i due tipi di indirizzo. Nel caso non venga trovato mette un numero molto alto
```