Appunti di basi di dati

Appunti di basi di dati

Prime lezioni

Ecco alcuni concetti necessari per un buon primo approccio verso la materia:

- **SI** (Sistem informativo) è l'insieme di persone, risorse, strumenti e infrastrutture impiegate nella lavorazione di informazioni.
- **DBMS** (Data base management system) è un sistema software in grado di gestire efficacemente le inforamzioni di un SI, rappresentandone i dati in forma integrata, secondo un modello logico, e garantendone la persistenza, la condivisione, l'affidabilità e la privatezza.
- **RDBMS** (Relational data base management system) è un tipo di DBMS che adotta il modello relazionale per rappresentare i dati.
- **DB** (Data base) rappresenta generalmente una collezione di dati per un'applicazione, o meglio una raccolta di dati gestiti da un DBMS. Questi dati sono memorizzati su dispositivi logici e rispettano un modello fisico dettato dal tipo di DBMS. Inoltre gli utenti si possono interfacciare tramite un **query language** come SQL.
- **Database Engine** è una componente software che un DBMS usa per gestire le operazioni principali, ovvero *Create, Read, Update e Delete*. Queste operazioni sono racchiuse nell'acronimo CRUD.
- **Modello E/R** è un modello *concettuale* dei dati che permette di descrivere la realtà, indipendentemente dai dati raprresentati.
- Il **Dato** è l'unità di misura dell'informazione che, in ambito informatico può essere un numero, una lettere, immagini o suoni e che ha la necessità di avere un significato in un contesto ben specifico per rappresentare la realtà.
- L'**Informazione** è un qualsiasi tipo di evento che porta conoscenza. Questa è prodotta dall'elaborazione di più dati.
  Una volta date queste piccole definizioni possiamo dire che la raccolta dei dati e la loro elaborazione in un ambiente come quello lavorati è da sempre stato necessario. Prima di parlare dell'avvento della tecnologia moderna possiamo vedere che il tutto era molto difficile, soprattutto perchè non c'era uno *standard* che potesse essere rispettato. Questo in un ambiente crea problemi nella trasmissione di informazioni verso l'esterno. Per questo vediamo che con la tecnologia sono stati creati dei programmi ad hoc per la gestione dei dati. I primi si basavano su file system e quindi non portavano molta novità ai già esistenti file di testo. Questo metodo ha fatto emergere i punti più critici della raccolta dei dati quali l*a ridondanza dei dati, i vincoli di integrità dei dati (range accettabile di un dato) e diversi problemi di rappresentazione* oltre a problematiche di accesso. Ecco perchè i **sistemi informatici**, ovvero tutti quei sistemi che permettono di digitalizzare i sistemi

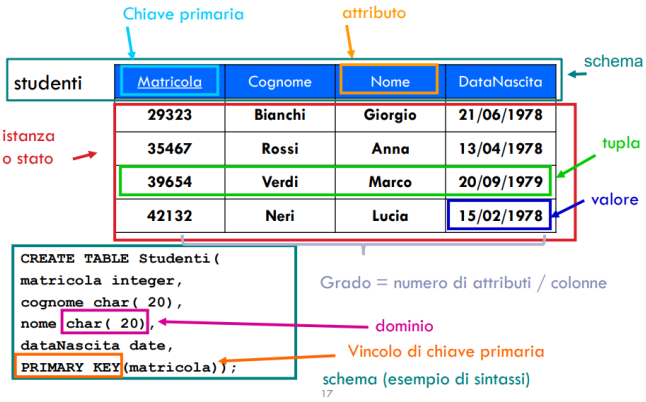
informativi, hanno portato una svolta epocale sotto questo punto di vista. I *DBMS* offorno un sistema hardware capace di gestire grandi quantità di dati persistenti e condivisi offrendo un livello di astrazione agli utenti. Inoltre garantisce efficienza ed efficacia nella gestione dei dati, oltre a garantire l'affidabilità dei dati. Tutto questo e molto altro è fattibile grazie ai DBMS che gestiscono basi di dati.

Un po' di storia

Lo sviluppo di data bases inizia negli anni 60 circa con *IDS* (Integrated data store) sviluppato da General Electric creando il primo tipo di DBMS con il modello a network.
In contemporanea per la missione lunare APollo IBM ha sviluppato l'*IMS* (Information management System) con modello gerarchico.
Negli anni 70 vediamo la prima implementazione di DBMS relazionali che negli anni 80 vince su tutti i suoi predecessori, rendendo standard.
Fino agli anni 2000 i cambiamenti sono stati pochi e abbastanza trascurabili per essere citati. Ma a un certo punto sono nati i **NoSQL** concentrati su alta disponibilità e alta scalabilità che si contrapponevano ai DBMS relazionari.

Il modello relazionale

Per rappresentare i dati in un DB è necessario ridursi a un modello di dati e, più precisamente, a un **modello logico** che permette di descrivere i dati con le relative associzioni e vincoli. Il concetto di relazione viene espresso tramite le tabelle collegate grazie a dei **valori** che verranno meglio espressi in seguito. In generale la peculiarità di questo modello è l'astrazione data dal livello logico che rende disponibili in modo efficace i dati all'utente. Inoltre lo **schema fisico** descrive come è rappresentato il DB a livello fisico e quindi in quale file è memorizzata una relazione.
In un preciso istante di tempo i dati nel database sono chiamati **snapshot** del database, comprendnendo le occorrenze e istanze presenti sul DB.
Le **viste o view** sono una rappresentazione personalizzata del DB, permettendo di creare tabella aggiuntive con campi foramti da unioni tra più tabelle. Il concetto di view verrà spiegato in seguito a livello pratico.
Ogni tabella ha delle caratteristiche che devono essere rispettate per mantenere all'interno dati in un modo non ambigue. Il primo ocncetto principale dei DB e delle tabelle è quello della **priamry key**, ovvero un attributo (colonna) che identifica ogni record o tupla in modo univoco. Ogni primary key ha la necessità di avere come attributo supplementare il **NOT NULL**, perchè una chiava NULL crea problemi nel dominio. Oltre alla chiave primaria possiamo avere la **superchiave**, ovvero un insieme insieme di attributi, chiave priamria compresa, che formano una chiave più *forte*. Questo permette di mettere dei vincoli ai dati inseriti, potendoli distinguere con facilità. Inoltre possiamo vedere che oltre ai classici tipi di dichiarazioni dei dati, come l'int il char e il varchar (stringa), abbiamo degli attributi che ne definiscono caratteristiche molto specifiche come il **NULL**, l'**AUTOINCREMENT** e le **FOREIGN KEY**.



Per **Foreign key** si definisce una chiave che punta alla chiave primaria di un'altra tabella e che quindi è vincolata a questa tabella. Questo porta ad avere due tabelle con la stessa chiave, ma molto probabilmente con attributi diversi. Questo è necessario quando si uniscono(join) due o più tabelle.
Con **business rule** definiamo un vincolo molto forte e molto spesso personalizzato per fare controlli su determinati attributi.
Quando parliamo di **dominio** definiamo un range di valori possibili per un attributo in particolare; questo range è molto importante perchè limita le caratteristiche del dato, prevenendo possibili errori.
Quando parliamo della **Prima forma normale 1F** quando ogni tabella del DB è foramta da soli attributi atomici, dove atomico vuol dire che non è scomponibile.

Caratteristiche e Linguaggio SQL

Quando parliamo di SQL parliamo di un linguaggio standard disponibile in tante versione con dei propri *dialetti* che bene o male permettono di fare sempre le stesse cose. Questo linguaggio è formato da una suddivisione delle classi dei comandi che permettono di capire meglio il funzionamento di questo **linguaggio di interrogazione ai database**. Le catergorie sono tre, ovvero:

- **DDL** Data definition language che comprende i comandi di *Create, Alter e Drop* e che quindi permette di lavorare sulle tabelle, creandole e modificandole;
- **DML** Data manipulation language che comprende i comandi di *Insert, Update e Delete* che permette di avere una visione solo interna delle tabelle, senza poter creare legami tra le relazioni;
- **DCL** Data control language che comprende i comandi di *Grant e Revoke* e che quindi gestisce i permessi utente come visualizzazione o modifica delle tabelle.
  Ora che bene o male sappiamo cos'è SQL e come funziona, è arrivato il momento di fare degli esercizi per capire realmente come funziona.

```
create tabel sedi(
    Sede varchar(30) primary key,
    Responsabile varchar(30) default 'Sviluppatore',
    Città varchar(20)
);
```

Questo piccolo esempio permette di creare una tabella in un database pre-esistente e di creare tre campi, uno tra cui è primary key. Piccola noda, ovvero che il tag **default** definisce il valore predefinito in

caso sia NULL.

Nome	Descrizione	Posizione
Limit	Permette di limitare il numero di tuole prodotte da una query	Viene messo alla fine di una query
Not Null	Obbliga un attributo a non essere nullo	Viene messo nella dichiarazione di un attributo
Is Null	Controlla se un attributo è nullo	
Is Not Null	Controlla che un attributo non è nullo	Viene utilizzato nella where
Distinct	Evita ripetizioni nell'output di una query	VIene messo subito dopo il select
Check	Permette di aggiungere un controllo arbitrario su uno o più attributi nella creazione di tabelle	Può essere messo nella dichiarazione di un attributo o alla fine della tabella
Unique	È molto simile alla primary key, permettendo di avere attributi con valori non ripetibili nella stessa colonna	Viene messo nella dichiarazione dell'attributo
Order by	Permette di ordinare l'output in base a una colonna in modo cresente o decrescente	Viene messo alla fine di una query
Between	Operatore che permette di creare un range di appartenenza per la query	Viene utilizzato nel where
In	Operatore che permette di esprimere un gruppo di valori accettabili	Viene utilizzato nel where
Like	Operatore simile all' = che permette di fare un confronto con le stringhe e anche di utilizzare caratteri jolly come l'underscore o il percentuale che permettono di esprimere pattern specifici	Viene utilizzato nel where

Join

La **Join** è un comando che permette di unire due tabelle tramite un vincolo di chiave esterna, a livello algebrico quello che facciamo è un prodotto cartesiano, ovvero andiamo a fare un matching tra tupla a tupla tra le tabelle. La join può essere di diversi tipi per esigenze diverse, possiamo avere:

- **Natural join** è la join più comune che unisce più tabelle ed elimina le colonne simili, lasciandone solo una nella tabella prodotta. Per scrivere una natural join usiamo `SELECT * FROM Impiegati I NATURAL JOIN Sedi S` e, tramite alias andiamo a cambiare i nomi delle tabelle per poterli utilizzare meglio.
- **Inner Join** permette di unire due o più tabelle tramite un vincolo di chiave esterna a noi noto. Per scriverla utilizziamo lo stesso modo della natural ma andiamo a specificare il vincolo `SELECT * FROM Impiegati I JOIN Sedi S ON (I.Sede = S.Sede)`.

- **Outer join** sono un gruppo di join che permettono l'unione tra due tabelle non omogenee, dove con omogenee definisco tabelle con tuple mal formate o non munite di chiave. Queste join sono di diversi tipi e si basano sulla posizione della tabella nel momento della scrittura vera e propria della query. Questa join `SELECT * FROM Impiegati I LEFT OUTER JOIN Sedi S ON (I.Sede = S.Sede)` è di tipo left outer perchè andiamo a prendere **tutti** i valori della tabella di sinistra, anche quelli con valore nullo.

Esercizi pagina 24 slide SQL database Northwindit

```sql
/*1) Selezionare i primi 10 Prodotti (nomeProdotto) con prezzo > 30*/
select *
from prodotti
where PrezzoUnitario > 30
limit 10

/*2) Selezionare i clienti italiani (NomeSocietà, Città, Nazione)*/
select NomeSocietà, Città, Posizione
from clienti
where Paese like 'Italia'

/*3) Selezionare gli impiegati con posizione rappresentante (*)*/
select *
from impiegati
where Posizione like 'rappresentante    '

/*4)Selezionare tutte le città dei clienti (Città, Nazione)*/
Select Città, Nazione
from clienti

/*5) Eliminare i valori ripetuti dalla query precedente*/
Select Distinct Città, Nazione
from clienti

/*6) Selezionare i prodotti sospesi con scorte maggiori di 0*/
select *
from prodotti
where Sospeso = 0 and Scorte > 0

/*7) Selezionare i prodotti con scorte inferiori al livello di riordino
o quantità ordinata maggiore di 0*/
select *
from prodotti
where Scorte < LivelloDiRiordino or QuantitàOrdinata > 0
```

```sql
/*8) Selezionare gli ordini relativi all'anno 1997*/
select *
from ordini
where year(DataOrdine) = 1997

/*9) Selezionare gli ordini la cui destinazione è Francia o Germania*/
select *
from ordini
where PaeseDestinatario in ('Francia', 'Germania')

/*10) Selezionare i clienti che iniziano con la B*/
select *
from clienti
where Contatto like 'B%'

/*11) Per tutti i prodotti relativi all'ordine 10260 stampare prezzo unitario, quantità, sconto e totale
*/
select PrezzoUnitario, Quantità, Sconto, IDProdotto, (Quantità * PrezzoUnitario) * (1 - Sconto) as Totale
from dettagliordini
where IDOrdine = 10260

/*12) Stampare i fornitori per i quali la zona non è definita*/
select *
from fornitori
where Zona is null

/*13) Stampare i fornitori per i quali la zona è presente*/
select *
from fornitori
where Zona is not null

/*14)Visualizzare i clienti tedeschi in ordine alfabetico*/
select *
from clienti
where Paese like "Germania"
order by Contatto asc

/*15) Visualizzare i 10 prodotti più costosi (in ordine decrescente di prezzo)*/
select *
from northwindit.prodotti
```

```
group by PrezzoUnitario
order by PrezzoUnitario desc
limit 10

/*16) Visualizzare i nomi dei prodotti e le relative categorie*/
select p.NomeProdotto as Nome,  c.IDCategoria as Categoria
from categorie c natural join prodotti p

/*17) Stampare il nome dei fornitori dei prodotti di categoria bevande*/
select f.IDFornitore
from fornitori as f natural join categorie as c natural join  prodotti as p
where c.NomeCategoria like "Bevande" and p.IDCategoria like c.IDCategoria

/*18) Visualizzare gli ordini che contengono almeno un prodotto della
categoria Bevande*/
select distinct p.*
from prodotti p natural join categorie c
where c.NomeCategoria like "Bevande"

/*19) Selezionare le coppie di clienti della stessa città*/
select  c1.IDCliente, c2.IDCliente, c1.Città, c2.Città
from clienti as c1 join clienti as c2 on (c1.Città = c2.Città)
where  c1.IDCliente < c2.IDCliente
```

```
/*20) Stampare per ogni impiegato (Nome, Cognome) i dati del suo superiore
(Nome, Cognome, Posizione) NON FUNZIONA*/
SELECT i.Nome, i.Cognome, d.Nome, d.Cognome, d.Posizione
from impiegati i join impiegati d on (i.IDImpiegato = d.IDImpiegato)
where i.Superiore < d.Superiore

/*21) Elencare gli impiegati che hanno gestito ordini relativi a clienti
tedeschi*/
select distinct i.Nome, i.Cognome
from impiegati as i join ordini as o on (i.IDImpiegato = o.IDImpiegato) join
clienti as c on (c.IDCliente = o.IDCliente)
where c.Paese like "Germania"
```

da spiegare il linguaggio sql pag 46