

Claudio Bottari

Claudiobottari@hotmail.com

Basi di dati

Project work

Contenuti:

[1. Introduzione alle basi di dati](#)

[2. Basi di dati relazionali](#)

[3. Cenni di algebra e calcolo relazionale](#)

[4. SQL](#)

Introduzione alle basi di dati

Uno dei principali compiti dei sistemi informatici è l'attività di raccolta, organizzazione e conservazione dei dati. Tali sistemi garantiscono che questi dati siano conservati in modo permanente su dispositivi per la loro memorizzazione, permettendone l'aggiornamento e rendendone possibile l'accesso da parte degli utenti.

Questo project work ha come argomento la gestione dei dati tramite sistemi informatici; ha come obiettivo trattare in maniera semplice ma esaustiva i concetti che stanno dietro ad una tale gestione, concretizzati nel linguaggio SQL.

Di seguito verranno introdotti i concetti di sistema informativo e di base di dati, definendo poi quali sono i requisiti che deve avere un sistema informatico per gestire una base di dati.

Contenuti

[1.1 Sistemi informativi, informazione e dati](#)

[1.2 Basi di dati, la definizione](#)

[1.3 Sistemi di basi di dati](#)

[1.3.1 Modelli dei dati](#)

[1.3.2 I livelli di astrazione dei DBMS](#)

[1.3.3 Indipendenza dei dati](#)

1.1 Sistemi informativi, informazioni e dati

Per sistema informativo si intende quel sistema che permette la disponibilità e la gestione delle informazioni. L'esistenza di un sistema informativo è indipendente dalla sua automazione; lo dimostra il fatto che archivi e servizi anagrafici esistono da vari secoli. Per indicarne la porzione automatizzata viene utilizzato il termine sistema informatico. La diffusione dell'informatica ha fatto sì che la quasi totalità dei sistemi informativi siano anche sistemi informatici.

Le informazioni vengono rappresentate e scambiate in varie forme, quali la lingua, disegni, figure, numeri. In alcuni casi può anche non esistere una rappresentazione esplicita delle informazioni, come nel caso di informazioni trasmesse oralmente e ricordate a memoria. Col progredire delle attività umane, tuttavia, è nata l'esigenza di individuare opportune codifiche per la memorizzazione dei dati.

Nei sistemi informatici il concetto di rappresentazione e codifica viene portato all'estremo: le informazioni vengono rappresentate per mezzo di dati, che hanno bisogno di essere interpretati per fornire informazioni.

1.2 Basi di dati, la definizione

La più generale definizione di una base di dati è collezione di dati utilizzati per rappresentare le informazioni di interesse per un sistema informativo.

Tale definizione è molto semplicistica e troppo generale. Nel paragrafo seguente si cerca di definire il termine in maniera più precisa.

Occorre, tuttavia, trarre una prima considerazione sulle basi di dati. Se prendiamo come esempio i dati relativi alle applicazioni bancarie noteremo che essi hanno una struttura sostanzialmente invariata da decenni, mentre le procedure che agiscono su di essi variano con una certa frequenza. Inoltre, quando viene introdotta una nuova procedura occorre, prima di tutto, "ereditare" (=importare) i dati dalla vecchia, se pur con le necessarie trasformazioni.

Questa caratteristica di stabilità porta ad affermare che i dati costituiscono una "risorsa" per l'organizzazione che li gestisce, un patrimonio significativo da sfruttare e proteggere. Le normative attuali in fatto di privacy e tutela delle basi di dati lo dimostra.

1.3 Sistemi di gestione di basi di dati

Sebbene la gestione dei dati abbia catalizzato, fin dalle origini dell'informatica, l'attenzione delle applicazioni, solo negli anni settanta nascono linguaggi specificatamente dedicati alla gestione dei dati. Un esempio di tali linguaggi è il COBOL, nato in quegli anni e ormai superato, che è presente ancor oggi in un numero incredibile di applicazioni.

L'approccio "convenzionale" alla gestione dei dati sfrutta la presenza di archivi (o file) per memorizzare i dati in modo persistente sulla memoria di massa. Tale approccio presenta delle macroscopiche deficienze per quanto riguarda la ricerca e la condivisione dei dati, in pratica annullata; infatti con una simile metodologia di lavoro ogni utente lavora con la propria copia "locale", con i relativi problemi *ridondanza* e possibilità di *incoerenze*. Le basi di dati sono state concepite in buona parte per ovviare ad inconvenienti di questo tipo.

Un sistema di gestione di basi di dati, detto DBMS (Data Base Management System) è un sistema software in grado di gestire collezioni di dati che siano *grandi*, *condivise* e *persistenti* assicurando la loro *affidabilità* e *privatezza*. Inoltre, in quanto prodotto informatico, deve essere *efficiente* e *efficace*. Una base di dati è una collezione di dati gestita da un DBMS.

Riassumendo un DBMS si occupa di basi di dati con le seguenti caratteristiche:

Grandi: nel senso che possono avere anche dimensioni enormi (terabyte e oltre) e quindi oltre le capacità della memoria centrale di un elaboratore. Di conseguenza un DBMS deve essere in grado di gestire memorie secondarie.

Condivise: perché un DBMS deve permettere a più utenti di accedere contemporaneamente ai dati comuni. In tal modo viene anche ridotta la *ridondanza* e *inconsistenza* dei dati, dato che esiste una sola copia dei dati. Per controllare l'accesso condiviso di più utenti il DBMS dispone di un meccanismo apposito, *detto controllo di concorrenza* .

Affidabilità: dato che un DBMS deve garantire l'integrità dei dati anche in caso di malfunzionamento hardware e software, prevedendo per lo meno procedure

di recupero dei dati. I DBMS forniscono, per tali scopi, procedure di salvataggio e ripristino della base di dati (*backup* e *recovery*).

Privatezza: i DBMS gestiscono un sistema di *autorizzazioni* che definisce i *diritti* di ciascun utente (lettura, scrittura ecc.).

1.4 Modelli di dati

Un modello di dati è un insieme di concetti utilizzati per organizzare i dati di interesse e descrivere la struttura in modo che essa risulti comprensibile ad un elaboratore.

Ogni modello di dati fornisce meccanismi di strutturazione, analoghi ai costruttori di tipo dei linguaggi di programmazione, che permettono di definire nuovi tipi sulla base di tipi elementari predefiniti.

Il [modello relazionale](#) dei dati (modello su cui si concentra l'attenzione di questo project work) permette di definire tipi per mezzo del costruttore di *relazione*, che consente di organizzare i dati in insiemi di record a struttura fissa. Una relazione viene spesso rappresentata mediante una tabella in cui le righe rappresentano i specifici record e le colonne corrispondono ai campi dei record.

1.4.1 Schemi ed istanze

Esistono, oltre al modello relazionale, altri modelli di database quali il modello gerarchico, il modello reticolare, il modello ad oggetti. Tutti i modelli di basi di dati sono, però, accomunati da dalla presenza di una parte che rimane invariata nel tempo, detta *schema*, e da una parte, detta istanza o stato della base di dati, costituita dai valori effettivi.

Si dice anche che lo schema è la parte intensionale della base di dati mentre l'istanza è la parte estensionale.

1.4.2 Livelli di astrazione nei DBMS

Esiste una proposta di struttura standardizzata per i DBMS articolata su tre livelli, detti *esterno*, *logico* e *interno*; per ciascun livello esiste uno schema:

- ◆ Lo ***schema logico*** (o ***concettuale***), che costituisce la descrizione dell'intera base di dati per mezzo del modello logico adottato dal DBMS (cioè tramite uno dei modelli citati in precedenza, relazionale, gerarchico, reticolare o a oggetti).
- ◆ Lo ***schema interno*** costituisce la rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione.
- ◆ Uno ***schema esterno*** costituisce la descrizione di una porzione della base di dati di interesse, per mezzo del modello logico. Uno schema esterno può prevedere organizzazioni dei dati diverse rispetto a quelle utilizzate nello schema logico, che riflettono il punto di vista di un particolare utente o insieme di utenti. Pertanto, è possibile associare ad uno schema logico vari schemi esterni.

Nei sistemi moderni il livello esterno non è esplicitamente presente possibile definire relazioni derivate (o viste, dall'inglese *views*).

1.4.3 Indipendenza dei dati

L'architettura così definita garantisce l'indipendenza dei dati, ovvero la principale proprietà dei DBMS. Questa proprietà permette agli utenti ed ai programmi applicativi di utilizzare una base di dati ad un elevato livello di astrazione, che prescinde dai dettagli realizzativi utilizzati per la base di dati stessa. In particolare, l'indipendenza dei dati può essere caratterizzata ulteriormente come indipendenza fisica e logica:

- ◆ ***L'indipendenza fisica*** consente di interagire con il DBMS in modo indipendente dalla struttura fisica dei dati. In base a questa proprietà è possibile modificare le strutture fisiche senza influire sulle descrizioni dei dati ad alto livello e quindi sui programmi che utilizzano i dati stessi.
- ◆ ***L'indipendenza logica*** consente di interagire con il livello esterno della base di base in modo indipendente dal livello logico.

Basi di dati relazionali

Rappresenta il modello su cui si basa la maggior parte dei sistemi di basi di dati oggi sul mercato. Tale modello fu proposto in una pubblicazione scientifica nel 1970 al fine di superare le limitazioni logiche dei modelli allora utilizzati, che non permettevano di realizzare efficacemente la proprietà di indipendenza dei dati, già riconosciuta come fondamentale. Sebbene i primi prototipi di db basati sul modello relazionale risalgano ai primi anni settanta bisognerà aspettare la metà degli anni ottanta perché tale modello acquisisca una frazione significativa di mercato. La lentezza di affermazione del modello relazionale deriva principalmente dal suo alto livello di astrazione: non è stato immediato per gli operatori del settore imparare ad individuare relazioni efficienti.

Contenuti

[2 Il modello relazionale](#)

[2.1 Modelli logici nei sistemi di basi di dati](#)

[2.2 Relazioni](#)

[2.3 Informazione incompleta e valori null](#)

[2.4 Vincoli di integrità](#)

[2.5 Vincoli di tupla](#)

[2.6 Chiavi](#)

2. Il modello relazionale

Vengono qui illustrate le modalità secondo cui esso questo modello permette di organizzare i dati, come il concetto di relazione possa essere mutuato dalla teoria degli insiemi ed utilizzato, con le debite varianti, per rappresentare le informazioni di interesse in una base di dati. Vengono approfonditi *i concetti di corrispondenza fra dati in strutture diverse, informazione completa e vincoli di integrità.*

2.1 Modelli logici nei sistemi di basi di dati

Il [modello relazionale](#) si basa su due concetti fondamentali : *relazione* e *tabella*. Mentre il concetto di tabella è facilmente intuibile, quello di relazione proviene dalla matematica, ed in particolare dalla teoria degli insiemi. E' opinione diffusa che parte del successo del modello relazionale derivi dalla presenza contemporanea di questi due concetti, uno intuitivo ed uno formale. Infatti, mentre le tabelle risultano naturali e facilmente comprensibili le relazioni garantiscono una formalizzazione semplice e chiara che ha permesso uno sviluppo teorico del modello finalizzato al raggiungimento di risultati di interesse concreto.

Il modello relazionale risponde al requisito dell'indipendenza dei dati e, pertanto, prevede un livello fisico ed un livello logico. Utenti e programmatori interagiscono solo col livello logico e quindi non è necessario che essi conoscano le strutture fisiche della base di dati. Anche questo aspetto è responsabile del suo successo dato che i suoi principali concorrenti (reticolare e gerarchico) obbligavano gli utilizzatori a conoscerne, almeno a grandi linee, la struttura realizzativa.

2.2 Relazioni

Il concetto di relazione è legato al concetto puramente matematico *di prodotto cartesiano* tra due insiemi. Avendo due insiemi D_1 e D_2 il prodotto cartesiano ($D_1 \times D_2$) è l'insieme delle coppie ordinate (v_1 e v_2) tale che v_1 è un elemento di D_1 e v_2 è un elemento di D_2 . Quindi il prodotto cartesiano è l'insieme di tutti le combinazioni tra gli insiemi dati.

La relazione è un sottoinsieme della relazione matematica tra due insiemi, detti *domini della relazione*, rappresentato da un insieme di tuple omogenee, dove per tuple (traslitterazione dell'inglese "tuple") si intende un elemento definito tramite i suoi attributi. Le tuple sono in questo aspetto sono diverse dal concetto matematico di n-uple, elemento individuato tramite posizione, e tupla, in cui l'elemento è individuato tramite i suoi attributi.

Ad esempio dati due insiemi A e B dove $A = \{1,2,3\}$ e $B = \{h,k\}$ il prodotto cartesiano è uguale all'insieme $A \times B = \{(1,h),(2,h),(3,h),(1,k),(2,k),(3,k)\}$ mentre una relazione possibile è $\{(1,h),(1,k),(3,h)\}$.

Generalizzando la relazione ad un numero di insiemi $n > 0$ avremmo D_1, D_2, \dots, D_n il prodotto $D_1 \times D_2 \times \dots \times D_n$ ed un sottoinsieme che descriverà la relazione. Il numero n delle componenti dell'insieme è detto *grado* del prodotto cartesiano e della relazione. Il numero degli elementi che della relazione è detta cardinalità della relazione.

Le tabelle nascono dall'esigenza di rappresentare graficamente le relazioni presentandole in una forma più facilmente comprensibile. In questo caso le righe della tabella saranno rappresentate dalle tuple mentre le colonne ne rappresentano i campi.

E' importante chiarire che in una relazione, in quanto insieme, non vi è alcun ordinamento fra le tuple che lo compongono; nelle tabelle che la rappresentano l'ordine c'è per necessità, ma è occasionale in quanto due tabelle con le stesse righe, ma in ordine diverso, rappresentano la stessa relazione. Inoltre le tuple di una relazione sono distinte l'una dall'altra, in quanto tra gli elementi di un insieme non possono essere presenti due elementi uguali; da cui si deduce che una tabella può rappresentare una relazione solo se le righe che la formano sono diverse l'una dall'altra.

2.3 Informazioni incompleta e valori nulli

Un'altra caratteristica importante dei sistemi relazionali è la presenza di un particolare valore che può assumere un'istanza di una tabella. Tale valore è detto *null*

e viene utilizzato per indicare una serie di situazioni che è possibile trovare in un campo di una tabella. Ad esempio ci si può trovare di fronte ad una tabella del tipo:

<i>Città</i>	<i>Indirizzo prefettura</i>
Roma	Via Quattro Novembre
Firenze	Null
Tivoli	Null
Prato	Null

In questo esempio il valore null indica tre diverse situazioni:

- **Valore sconosciuto** : nel primo caso dato che Firenze è un capoluogo di provincia ed avrà sicuramente una prefettura; significa che il DBMS non dispone del suo indirizzo.
- **Valore inesistente** : nel caso di Tivoli, dato che questa città non ha di Prefettura.
- **Senza informazione** : siccome la provincia di Prato è di recente istituzione significa che non sappiamo se la mancanza di un indirizzo dipenda dal fatto che essa ancora non esista, oppure da un deficit del DBMS.

2.4 Vincoli di integrità

I vincoli di integrità sono quei vincoli, caratteristica fondamentale delle basi di dati relazionali, che indicano la "bontà" delle informazioni. Ci possono essere, infatti, casi in cui i dati non rispettino, per una serie di motivi che andremo ad analizzare, l'integrità logica del database che, in quanto insieme formalizzato di informazioni, ha delle regole molto precise (e spesso rigide) che devono essere obbligatoriamente rispettate.

I vincoli di un DBMS si dividono in intrarelazionali, se riguardano l'interno nella relazioni, ed extrarelazionali, più pesanti perché riguardano i legami fra le relazioni e quindi la natura stessa di un DBMS relazionale.

Il più caratterizzante esempio di violazione di un vincolo extrarelazionale si ha qualora non vi sia corrispondenza tra le istanze di due tabelle che, per ragioni intrinseche allo schema della base di dati, sono legate tra loro da un vincolo detto di integrità referenziale, il quale impone che per ogni valore di una tabella vi sia un corrispondente nell'altra.

Un vincolo intrarelazionale, al contrario, trova il suo soddisfacimento rispetto alle singole relazioni del DBMS; esso può essere :

- **vincolo di tupla**; spiegato di seguito
- **vincolo su valori**, o **vincolo di dominio** : quando si impone a certi valori del database di rientrare in determinate caratteristiche (ad esempio il voto di un esame universitario deve obbligatoriamente rientrare tra il 18 ed il 30, ed un DBMS incaricato raccogliere tali dati deve prevederlo).

2.5 Vincoli di tupla

Il vincolo di tupla è un vincolo che esprime delle condizioni sui valori di ciascuna tupla, indipendentemente dalle altre tuple.

Ad esempio consideriamo l'esempio dei voti degli esami universitari e ipotizziamo di avere un DBMS che debba memorizzarli; poniamo, inoltre, di destinare un campo all'eventuale lode (campo di tipo booleano, vero/falso). In questo caso avremmo che l'attributo "lode" potrà essere vero solo se il voto di quella tupla è uguale a trenta. Cercando di dare una forma a tale vincolo potremmo scrivere che:

$$(\text{not} (\text{lode} = \text{vero})) \text{ or } (\text{voto} = 30)$$

mentre in questo caso il vincolo di dominio (citato nel paragrafo precedente) potrebbe essere espresso con la formula:

$$(\text{voto} \geq 18) \text{ and } (\text{voto} \leq 30)$$

2.6 Chiavi

Una chiave è un insieme di attributi (anche uno solo) utilizzato per identificare in maniera unicamente le tuple di una relazione. Questo per la natura stessa del modello relazionale che pone questo vincolo come suo assioma.

Ne consegue che la violazione del vincolo di chiave, inammissibile in un DBMS relazionale, è la presenza di due o più tuple aventi la stessa chiave.

Cenni di algebra e calcolo relazionale

L'algebra relazionale è un linguaggio procedurale, basato su concetti di tipo algebrico che deriva da quella parte della matematica che si occupa degli insiemi.

Siccome i concetti trattati sono molto astratti, poco legati al pratico utilizzo dei DBMS quanto a quello realizzativo, in questo capitolo ci limiteremo a descrivere i concetti generali senza entrare nel dettaglio delle formule e delle dimostrazioni. Di seguito descriveremo gli operatori fondamentali dell'algebra relazionale, lo stretto necessario per comprendere da cosa deriva l' SQL (capitolo successivo).

Contenuto:

[3. Algebra relazionale](#)

[3.1 Unione](#)

[3.2 Intersezione](#)

[3.3 Differenza](#)

[3.4 Ridenominazione](#)

[3.5 Selezione](#)

[3.6 Proiezione](#)

[3.7 Join](#)

[3.8 Interrogazioni](#)

3 . Algebra relazionale

L'algebra relazionale definisce una serie di operatori e regole per la creazione e la modifica delle relazioni. Di seguito vengono trattati i più comuni.

3.1 Unione

Appurato che le relazioni sono insiemi, ha senso intervenire sulle relazioni con gli operatori dell'algebra insiemistica, quali l'unione la differenza e l'intersezione.

Tuttavia, dato che una relazione è un insieme di tuple omogenee (ovvero definite dagli stessi attributi) questi operatori potranno essere usati solo con relazioni aventi gli stessi attributi, o almeno degli attributi comuni.

Detto questo è possibile dire che l'unione tra due relazioni r_1 ed r_2 , definite sullo stesso insieme di attributi X (si scrive $r_1(X)$ ed $r_2(X)$), è indicata con $r_1 \cup r_2$ ed è una relazione ancora su X contenente le tuple che appartengono ad r_1 oppure ad r_2 , oppure ad entrambe.

Nella fattispecie delle basi di dati l'unione di due tabelle, che come abbiamo visto sono le corrispondenti delle relazione, è l'insieme delle righe che appartengono alle tabelle (alla prima, alla seconda o ad entrambe).

3.2 Intersezione

L'intersezione di $r_1(X)$ ed $r_2(X)$ è indicata con $r_1 \cap r_2$ ed è una relazione su X contenente le tuple che appartengono sia ad r_1 che ad r_2 .

Riferito alle tabelle l'intersezione di due di queste è l'insieme delle righe che appartengono ad entrambe.

3.3 Differenza

La differenza di $r_1(X)$ ed $r_2(x)$ è indicata con $r_1 - r_2$ ed è una relazione su X contenente le tuple che appartengono ad r_1 e non appartengono ad r_2 .

Quindi la differenza tra due tabelle è l'insieme delle righe che appartengono alla prima tabella e non alla seconda.

3.4 Ridenominazione

Per risolvere il problema derivante dal fatto che gli operatori sopra citati possono essere usati solo con attributi uguali, l'algebra relazionale ci mette a disposizione l'operatore di ridenominazione.

Questo operatore ci permette di ridefinire il nome dell'attributo in modo da operare operazioni di unione, differenza ed intersezioni anche su relazioni con diversi attributi (sempre ammesso che ciò possa avere un senso).

3.5 Selezione

La selezione è quell'operatore dell'algebra relazionale che produce un sottoinsieme di tuple, producendo una relazione che ha gli stessi attributi dell'operando ma un numero minore (o uguale in casi particolari) di istanze.

La selezione introduce un concetto fondamentale dell'algebra relazionale e, di conseguenza, delle basi di dati : il concetto di condizione (trattato in maniera più esauriente nel capitolo successivo). Infatti la discriminante che permette di stabilire quali tuple andranno a formare il risultato della selezione è il fatto che essa rispondano meno ad una determinata condizione.

3.6 Proiezione

La proiezione è un concetto simile a quello della selezione. Essa produce, alla pari della selezione, un sottoinsieme della relazione genitrice; a differenza della selezione, però, il sottoinsieme prodotto contiene tutte le tuple originali ma con un numero inferiore di attributi.

3.7 Join

L'operatore di join è il più caratteristico dell'algebra relazionale.

Il join naturale è un operatore che correla i dati di due relazioni sulla base di valori uguali in attributi con lo stesso nome.

Esistono diversi tipi di join, a seconda delle diverse situazione di relazioni, che sono tuttavia riconducibili al modello del join naturale. Un'analisi più approfondita esula dai compiti di questo lavoro, volto all'aspetto pratico delle basi di dati. Ulteriori informazioni circa il join si possono trovare nel capitolo successivo, che mostra l'implementazione di tale operatore nel linguaggio SQL.

3.8 Viste

La vista non è un operatore dell'algebra relazionale; più semplicemente è una tecnica dell'algebra che permette di mettere a disposizione dell'utente rappresentazioni diverse degli stessi dati.

La tecnica che ci permette di raggiungere questo scopo è quella delle relazioni derivate, relazioni il cui contenuto è funzione del contenuto di altre relazioni. Queste relazioni si contrappongono alle relazioni di base (le relazioni vere e proprie), il cui contenuto è autonomo.

A livello di implementazioni si possono dividere le viste in due tipi:

- **viste materializzate** : relazioni derivate effettivamente memorizzate sulla base di dati;
- **viste virtuali** (o semplicemente) : relazioni definite per mezzo di funzioni, non memorizzate sulla base di dati, ma utilizzabili nelle interrogazioni come se effettivamente lo fossero.

SQL

SQL è l'acronimo di Structured Query Language, ed è un linguaggio di interrogazione delle basi di dati sviluppato dall'IBM nella seconda metà degli anni Settanta. L'SQL è largamente diffuso e riveste una notevole importanza nel mondo delle applicazioni per basi di dati. Tale diffusione nasce principalmente dalla sua standardizzazione che lo pone come il linguaggio "universale" di accesso ai database. Infatti quasi tutti i MDBS, nonostante abbiano delle caratteristiche proprietarie, riconoscono i comandi SQL in modo che ci si possa interfacciare con essi in modo univoco.

Contenuti

- [4.1 Definizione dei dati in SQL](#)
 - [4.1.1 Domini elementari e schema](#)
 - [4.1.2 Definizione delle tabelle](#)
 - [4.1.3 Vincoli intrarelazionali](#)
 - [4.1.4 Vincoli interrelazionali](#)
 - [4.1.5 Modifica degli schemi](#)
- [4.2 Interrogazioni in SQL](#)
 - [4.2.1 Interrogazioni semplici](#)
 - [4.2.2 Uso del join](#)
 - [4.2.3 Operatori aggregati](#)
 - [4.2.4 Interrogazioni con raggruppamento](#)
 - [4.2.5 Predicati sui gruppi](#)
- [4.3 Manipolazione dei dati in SQL](#)
 - [4.3.1 Inserimento di righe](#)
 - [4.3.2 Cancellazione delle righe](#)
 - [4.3.3 Modifica delle righe](#)

4.1 Definizione dei dati in SQL

4.1.1 Domini elementari e schema

Innanzitutto illustriamo come SQL si occupi della definizione degli schemi delle basi dati.

SQL mette a disposizione sei famiglie di domini elementari, dove come domini elementari si intendono i domini dai quali si può partire per costruire i domini da associare agli attributi dello schema.

- **Carattere** : singoli caratteri oppure stringhe (lunghezza fissa o variabile)
- **Bit** : per valori 0/1 (può essere utilizzato anche per stringhe di bit in cui memorizzare più dati 0/1). Introdotto con SQL-2.

- **Tipi numerici esatti** : per i valori interi o per valori decimali a virgola fissa.
 - Numeric :
 - Decimal :
 - Integer :
 - SmallInt :
- **Tipi numerici approssimativi** :
 - Float
 - Double precision (precisione doppia)
 - Real
- **Data e ora**
- **Intervalli temporali**. Introdotto con SQL-2.

Per creare un dominio occorre usare la sintassi:

```

Create domain NomeDominio as TipoDiDato
      [valore di default]
      [vincolo]

```

in questo caso l'operazione è concettualmente simile alla definizioni dei tipi di dato in linguaggi evoluti tipo il C (o Pascal, VB ecc.). Al contrario i vincoli sui domini in SQL non hanno un corrispondente nei linguaggi evoluti.

Quindi SQL permette di definire lo schema di una base di dati come una collezione di oggetti; ogni schema è costituito da un insieme di famiglie di caratteri, domini, tabelle (=relazione), indici, asserzioni, viste e privilegi.

La sintassi è :

```

Create schema nomeschema [autorizzazione]

```

dove autorizzazione rappresenta il nome dell'utente proprietario dello schema.

Una volta creato lo schema occorre definire i suoi componenti, anche se non è necessario che venga fatto contemporaneamente alla creazione e può avvenire in fasi successive.

4.1.2 Definizione delle tabelle

Una tabella SQL è costituita da una collezione di attributi a da un insieme (eventualmente vuoto) di vincoli.

La sintassi è :

```

create table NomeTabella
(
      NomeAttributo Dominio [valore di default] [vincoli]
      { ecc. }
)

```

quindi una tabella viene definita associandole un nome ed elencandone gli attributi che ne compongono lo schema.

Ecco un esempio ...

```
Create table persona
(
    Nome      char(20)  primary key,
    Indirizzo char(50),
    Città     char(20)
)
```

I valori di default sono i valori che deve assumere l'attributo quando viene inserita una riga nella tabella senza che venga specificato il valore dell'attributo stesso.

Sia nella definizione dei vincoli che in quella delle tabelle è possibile stabilire dei vincoli; per vincoli si intendono delle proprietà che devono essere verificate da ogni istanza della base di dati.

Esistono *vincoli intrarelazionali*, ovvero interni alla tabella, e *vincoli interrelazionali*, che rappresentano i vincoli di integrità referenziale (detti anche vincoli di riferimento).

4.1.3 Vincoli intrarelazionali

In SQL i vincoli intrarelazionali che possono essere definiti sugli attributi di una tabella sono i vincoli di *not null*, *unique* e *primary key*.

- **Not null** : significa che l'attributo deve essere sempre specificato.
- **Unique** : si applica ad uno o più attributi imponendo che righe differenti abbiano valori differenti. Viene fatta eccezione solo per il valore null.
- **Primary Key** : specifica quali attributi rappresentano la chiave primaria.

Mentre i vincoli not null e unique possono essere definiti n volte, la primary key può essere definita una volta sola (sia che sia specificata a livello di attributo che in calce per indicare una serie di attributi).

Esempio:

```
Create table persona
(
    Nome      char(20),
    Cognome   char(20),
    Indirizzo char(50),
    Città     char(20),
    Primary key (Nome, Cognome)
)
```

...crea una tabella in cui la chiave primaria è rappresentata dall'insieme del nome e del cognome.

```
Create table persona
(
    Matricola char(6) primary key,
    Nome char(20) not null,
    Cognome char(20) not null,
    Indirizzo char(50),
    Città char(20),
    Unique (Nome, Cognome)
)
```

...crea una tabella in cui la chiave primaria è il codice di matricola, in cui gli attributi nome e cognome devono essere obbligatoriamente specificati (per via del not null) ed in cui non possono comparire due righe aventi stesso nome e cognome.

4.1.4 Vincoli interrelazionali

In SQL i vincoli tali vincoli si definiscono tramite l'apposito costrutto della *chiave esterna* (vincolo di *foreign key*).

Questo vincolo impone che ogni valore dell'attributo di una tabella (se diverso da null) sia presente tra i valori di un attributo delle righe appartenenti ad un'altra tabella (detta tabella *esterna*).

Questo vincolo può essere definito in due modi : con l'uso del costrutto sintattico *references* ,con il quale si specificano la tabella esterna con il quale l'attributo deve essere legato, oppure con l'uso del costrutto *foreign key*.

Di seguito compare il nostro esempio con la definizione di un vincolo di integrità referenziale con un'altra tabella ("Dipartimento"). Il vincolo è specificato tramite l'uso di *references* per fare in modo che nella tabella persona l'attributo Dipart fosse obbligatoriamente un valore contenuto nella tabella esterna nell'attributo NomeDipart.

```
Create table persona
(
    Matricola char(6) primary key,
    Nome char(20) not null,
    Cognome char(20) not null,
    Indirizzo char(50),
    Dipart char(15)
        References Dipartimento(NomeDip),
    Città char(20),
    Unique (Nome, Cognome)
)
```

Invece il costrutto *foreign key* (posto non a livello di attributo ma in calce) offre delle possibilità maggiori. Innanzitutto permette di definire dei vincoli per un'insieme di attributi; inoltre permette di definire la reazione del DBMS ad eventuali forzature dei vincoli di *integrità referenziale*.

Infatti, trovandosi di fronte ad una relazione tra tabelle, occorre prendere in considerazione quale sarà la reazione della tabella interna (o *master*) all'atto della modifica di una riga della tabella esterna (detta *slave*). Le operazioni che possono mettere in crisi il nostro sistema di vincoli sono, ovviamente, le operazioni di *cancellazione* e *modifica* sulla tabella esterna, operazioni alle quali la tabella interna dovrà adeguarsi.

SQL ci permette di definire, come opzioni del costrutto *foreign key*, quattro reazioni del DBMS del caso di modifica (update) e cancellazione (delete):

	<i>On update</i>	<i>On delete</i>
<i>Cascade</i>	Il nuovo valore viene riportato su tutte le relative righe della tabella master	Tutte le relative righe della tabella master vengono cancellate
<i>Set null</i>	Nella tabella master viene inserito null al posto dei valori relativi all'attributo modificato	Nella tabella master viene inserito null al posto dei valori relativi all'attributo modificato
<i>Set default</i>	Nella tabella master viene inserito il valore di default al posto dei valori relativi all'attributo modificato	Nella tabella master viene inserito il valore di default al posto dei valori relativi all'attributo modificato
<i>No action</i>	Non viene eseguita alcuna operazione. Il DBMS respinge la richiesta di update.	Non viene eseguita alcuna operazione. Il DBMS respinge la richiesta di cancellazione.

Esempio di utilizzo del *foreign key*:

Create table persona

```
(
    Matricola char(6),
    Nome char(20) not null,
    Cognome char(20) not null,
    Indirizzo char(50),
    Dipart char(15)
    References Dipartimento(NomeDip),
    Città char(20),
    Primary key(Matricola),
    Foreignkey(Dipart) references Dipartimento(NomeDip)
    On delete set null,
```

On update cascade,
Unique (Nome, Cognome)
)

4.1.5 Modifica degli schemi

Il linguaggio implementa anche delle istruzioni dedicate alla modifica delle strutture esistenti. Tali comando sono *alter*, che permette di modificare domini e schemi di tabelle, e *drop*, che permette di rimuovere dei componenti siano essi schemi, domini, tabelle viste o asserzioni (non trattate).

Ecco un esempio che aggiunge un attributo alla tabella Dipartimenti.

```
Alter table Dipartimenti add column NroUffici numeric(4)
```

in questo modo la tabella ha una colonna in più che rappresenta, ad esempio, il numero degli uffici.

Invece il comando :

```
Drop table Dipartimenti
```

cancella la colonna appena creata.

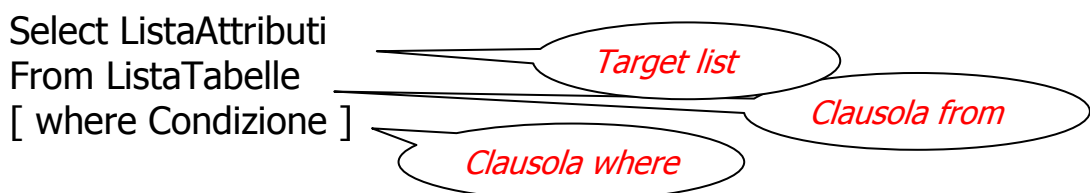
4.2 Interrogazioni in SQL

La parte di SQL dedicata all'interrogazione del DBMS è probabilmente la parte più importante ed utilizzata del linguaggio.

Innanzitutto bisogna precisare che SQL esprime le dichiarazioni in modo *dichiarativo*, ovvero si specifica l'obiettivo dell'interrogazione e non il modo con cui ottenerlo; in questo si contrappone ai linguaggi di interrogazione procedurali, come l'algebra relazionale, in cui si specifica il modo in cui l'interrogazione deve essere eseguita. Per essere eseguita l'interrogazione deve essere analizzata dall'interprete SQL (che è un componente del DBMS) per essere tradotta in un'interrogazione equivalente in linguaggio procedurale.

4.2.1 Interrogazioni semplici

Le interrogazioni in SQL sono specificate tramite l'uso dell'istruzione *select* la cui struttura essenziale è la seguente :



L'interrogazione SQL seleziona, tra le righe che appartengono al prodotto cartesiano delle tabelle elencate nella clausola `from`, quelle che soddisfano la clausola

where (se è assente vengono selezionate tutte le righe). Su ciascuna riga vengono considerate quelle che compaiono nella target list (tutte se è specificato il carattere jolly *).

Nella target list possono essere definiti degli *alias*, ovvero dei nomi che possono essere usati, successivamente, per richiamare il campo.

Esempio:

```
Select Stipendio as Salario
from Impiegato
where Cognome = "Rossi"
```

Prende il valore dello stipendio di "Rossi" dalla tabella Impiegato ridefinendolo come Salario.

Nella target list possono essere inserite anche espressioni generiche (addizione, sottrazione, moltiplicazione ecc.). Ad esempio :

```
Select Stipendio as Salario, Stipendio/12 as SalarioMensile
from Impiegato
where Cognome = "Rossi"
```

Per quanto riguarda la *clausola from* essa rappresenta l'insieme delle tabelle a cui si vuole accedere, e delle quali verrà eseguito il prodotto cartesiano. Nel caso di più tabelle il join (unione) viene eseguito specificando nella clausola where il legame tra le tabelle. Considerando l'esempio del paragrafo precedente (tabelle Impiegato e Dipartimento) e volendo prendere valori da entrambe le tabelle:

```
Select Impiegato.Nome, Impiegato.Cognome, Dipartimento.Citta
From Impiegato, Dipartimento
Where Impiegato.Dipart = Dipartimento.Nome
```

Rispetto agli esempi precedenti si nota l'uso del punto (in piena sintassi OOP) per indicare a quali a quale tabella si riferisce l'attributo.

La *clausola where* ammette una serie di espressioni booleane (=logiche); è possibile, all'interno della clausola, combinare predicati semplici con gli operatori logici *and*, *or* e *not*. Ciascun predicato semplice confronta (con gli operatori =, <>, <, >, <= e >=) una espressione costruita a partire dai valori degli attributi per la riga o con un valore costante o con il risultato della valutazione di un'altra espressione.

Esempio:

```
Select Nome
From Impiegato
Where Cognome = 'Rossi' and
      (Dipart = 'Amministrazione' or Dipart = 'Produzione')
```

che seleziona l'impiegato di nome "Rossi" appartenente al dipartimento 'Amministrazione' o 'Produzione'.

Oltre agli operatori standard già citati SQL mette a disposizione l'operatore like per il confronto di stringhe. Tale operatore si comporta come l'operatore di uguaglianza ma, a differenza di quest'ultimo, supporta anche i caratteri "%" (percentuale) e "_" (trattino sottolineato). Il primo rappresenta un carattere arbitrario mentre il secondo una stringa arbitraria (anche vuota).

Ad esempio:

```
Select *
From Impiegato
Where Cognome like '_o%i'
```

Seleziona tutti gli impiegati il cui cognome termina con la lettera "i" e la cui seconda lettera è una "o".

Di particolare importanza la gestione dei valori null nella valutazione dei predicati. Infatti nell'algebra booleana convenzionale si prendono in considerazione unicamente due valori (vero o falso) mentre abbiamo visto che avendo a che fare con delle basi di dati, possiamo avere anche il valore *null* (=sconosciuto). Le ultime versioni di SQL gestiscono i campi null considerandoli nella valutazione di relazioni complesse, estendendo a tre valori, *vero* | *falso* | *null* appunto, il possibile risultato.

Quindi avremmo invece delle classe tabelle di verità (noto anche ad i principianti di algebra booleana):

Not	
Falso	Vero
Vero	Falso

And	Vero	Falso
Vero	Vero	Falso
Falso	Falso	Falso

Or	Vero	Falso
Vero	Vero	Vero
Falso	Vero	Falso

...le nuove tabelle comprensive del valore *null*:

Not	
Falso	Vero
Null	Null
Vero	Falso

And	Vero	Null	Falso
Vero	Vero	Null	Falso

Null	Null	Null	Falso
Falso	Falso	Falso	Falso

Or	Vero	Null	Falso
Vero	Vero	Vero	Vero
Null	Vero	Null	Null
Falso	Vero	Null	Falso

Ultima precisazione riguarda la selezione dei campi *null* che si effettua tramite la sintassi:

Attributo is [not] null

Tale predicato risulta vero solo se il valore è *null*. Il predicato *not null* è il suo contrario.

4.2.2 Uso del join

L'ultima versione di SQL (SQL-2) ha introdotto una sintassi alternativa per l'espressione di join che permette di distinguere tra le condizioni che rappresentano condizioni di join e quelle che rappresentano condizioni di selezioni di riga. Inoltre il linguaggio è stato arricchito di nuove espressioni di *join*.

Innanzitutto la sintassi è la seguente:

```
Select AttrExpr [ [as] Alias ] {, AttrExpr [ [as] Alias ] }
From Tabella [ [as] Alias ]
    {, TipoJoin Tabella [ [as] Alias ] on CondizioneDiJoin }
[ where AltraCondizione ]
```

Mediante questa sintassi la condizione di join non compare come argomento della clausola *where*, ma viene invece spostata nell'ambito della clausola *from*, associata alle tabella che vengono coinvolte nel join.

Il parametro *TipoJoin* specifica quale è il tipo di join da usare, ed ad esso si possono sostituire i termini *inner*, *right outer*, *left outer* o *full outer* (in cui il qualificatore *outer* è opzionale). L'operatore *inner join* è il più comune e produce una selezione delle righe del prodotto cartesiano per cui la condizione è vera.

Ad esempio l'interrogazione realizzata precedentemente con l'utilizzo del *where* ora può essere riscritta:

```
Select I.Nome, Cognome, Citta
From Impiegato I inner join Dipartimento D on Dipart = D.Nome
```

Questo è il caso di un *join interno*, perché le righe che vengono coinvolte nel join sono in generale un sottoinsieme delle righe di ciascuna colonna. Può, infatti, capitare che alcune righe non vengano considerate in quanto non esiste una corrispondente riga nell'altra non vengano considerate in quanto non esiste una

corrispondente riga nell'altra per cui la condizione sia soddisfatta. Questo comportamento molto spesso non rispetta le esigenze delle applicazioni, le quali, alla eliminazione delle righe operata dal join, possono preferire di mantenere le righe, ponendo eventualmente opportuni valori *null*. Il *join esterno* (outer join) ha proprio questo compito.

Esistono tre diversi tipi di *join esterno* : left, right e full. Con il full join vengono presi tutti i valori delle tabelle, con il left tutti quelli della prima tabella specificata mentre con il right tutti quella della seconda.

Mentre una relazione è costituita da un insieme non ordinato di tuple, nell'uso reale delle basi di dati sorge spesso il bisogno di costruire un ordine sulle righe delle tabelle. SQL permette di costituire un ordine con l'istruzione order by posta dopo la clausola where. La sintassi è :

```
Order by AttrDiOrdinamento [ ASC | DESC ]
      { , AttrDiOrdinamento [ ASC | DESC ] }
```

In questo modo si specificano gli attributi che devono essere presi in considerazione per realizzare l'ordinamento (crescente o decrescente).

4.2.3 Operatori aggregati

Tali operatori sono una serie di parole chiave del linguaggio SQL dedicate all'aggregazione dei dati, grazie ai quali è possibile definire interrogazioni di notevoli interesse applicativo. E' necessario ricorrere a questi operatori ogni qual volta occorra valutare delle proprietà che dipendono da insiemi di righe.

Lo standard SQL prevede cinque operatori aggregati, divisibili in due gruppi: count da una parte e sum, max, min, avg dall'altra.

L'operatore count permette di determinare il numero di righe di una interrogazione; la sintassi è la seguente:

```
Count ( * | [ distinct ] | [ all ] ListaAttributi )
```

L'opzione * restituisce il numero delle righe; l'opzione *distinct*, invece, restituisce il numero dei diversi valori degli attributi mentre l'opzione *all* restituisce il numero dei valori diversi da *null*.

Ad esempio:

```
Select count(distinct Stipendio)
From Impiegato
```

Restituisce il numero dei diversi valori dell'attributo Stipendio fra tutte le righe della tabella Impiegato.

Oppure:

```
Select count(all Nome, Cognome)
From Impiegato
```

Restituisce il numero delle righe che possiedono un valore diverso da *null* sia per l'attributo Nome che per l'attributo Cognome.

Gli altri operatori di aggregazione richiedono, invece, solo un attributo o un'espressione, eventualmente preceduta dalle parole chiave *distinct* o *all*. Le funzioni *sum* ed *avg* ammettono come argomento solo espressione numeriche mentre *max* e *min* accettano anche intervalli di tempo e stringhe.

Gli operatori hanno il seguente significato:

- **Sum** : restituisce la somma dei valori posseduti dall'attributo su tutte le righe
- **Max** e **min** : restituiscono il massimo ed il minimo valore tra quelli di ciascuna riga (su questi operatori *distinct* o *all* non hanno alcun effetto)
- **Avg** : restituisce il valore medio tra quelli dell'attributo

4.2.4 Interrogazioni con raggruppamento

Qualora occorra produrre delle aggregazioni a dei sottoinsiemi di righe occorre fare ricorso alla clausola *group by*. Questa istruzione permette di creare dei raggruppamenti parziali, mostrando una sola riga per ogni insieme che ha uno stesso valore nell'attributo passato come parametro. Ad esempio:

```
Select Dipart, avg(Stipendio)
From Impiegato
Group by Dipart
```

Questa interrogazione ha come risultato un'insieme di righe, una per ogni valore dell'attributo Dipart, contenente ognuna il nome del dipartimento e la media tra i valori dell'attributo Stipendio relativi a ciascun dipartimento.

4.2.5 Predicati sui gruppi

Una volta eseguito un raggruppamento in sottoinsiemi con la clausola *group by* è possibile selezionare solo alcuni gruppi usando la clausola *Having*.

La clausola *having* è il corrispondente per i gruppi della clausola *where* per gli attributi. Essa descrive le condizioni che si devono applicare al termine dell'esecuzione di una interrogazione che fa uso della clausola *group by*.

Esempio:

```
Select Dipart, sum(Stipendio) as SommaStipendi
From Impiegato
Group by Dipart
Having SommaStipendio > 100
```

Volendo riassumere la sintassi SQL dell'istruzione *select* con tutte le clausole analizzate avremmo che:

```
Select ListaAttributiOEspressioni
From ListaTabelle
[ where CondizioniSemplici ]
[ group by ListaAttributiDiRaggruppamento ]
[ having CondizioniAggregate ]
[ order by ListaAttributiDiOrdinamento ]
```

4.3 Manipolazione dei dati in SQL

La parte di manipolazione dei dati si occupa delle operazioni di inserimento, modifica e cancellazione delle righe.

4.3.1 Inserimento di righe

Il comando di inserimento di righe nella base di dati presenta due sintassi alternative:

```
Insert into NomeTabella [ ListaAttributi ]
    < values ( ListaDiValori ) | SelectSQL >
```

Questa sintassi permette di inserire una riga specificandone i valori dei suoi attributi. Una seconda sintassi, meno usata ma più consona all'impostazione SQL che ha come oggetto le tabelle e non le singole righe, permette di aggiungere degli insiemi di righe. La sua sintassi è:

```
Insert into NomeTabellaDestinazione
    ( Select ListaAttributi
      from NomeTabellaOrigine
      where Condizione )
```

In questo modo vengono inseriti nella tabella destinazione le righe risultanti dalla selezione sulla tabella origine.

4.3.2 Cancellazione delle righe

Il comando SQL *delete* è il comando che permette di eliminare delle righe dalle tabelle di una base di dati.

La sintassi è:

```
Delete from NomeTabella [ where Condizione ]
```

...e attua la cancellazione delle righe che rispondono alla condizione specificata.

La condizione rispetta la sintassi SQL, per cui potremmo avere al suo interno delle interrogazioni nidificate che fanno riferimento ad altre tabelle. Un semplice esempio che elimina i dipartimenti senza impiegati:

```
Delete from Dipartimento
      Where Nome not in (select Dipart
                        From Impiegato)
```

E' da notare la differenza tra il comando delete ed il comando drop. Ad esempio il comando:

```
Delete from Dipartimento
```

...elimina tutte le righe della tabella dipartimento, ma lo schema rimane immutato; il comando, infatti, cancellerà solo le istanze della tabella. Mentre il comando:

```
Drop table Dipartimento cascade
```

...elimina tutte le istanze della tabella, nonché lo schema.

4.3.3 Modifica delle righe

In SQL la modifica delle righe avviene mediante l'utilizzo del comando update, seguendo la sintassi:

```
Update NomeTabella
      Set Attributo = < Espressione | SelectSQL | Null | default >
      { , Set Attributo = < Espressione | SelectSQL ... > }
[ where Condizione ]
```

Il comando update permette di aggiornare uno o più attributi delle righe di *NomeTabella* che soddisfano l'eventuale condizione. Se la condizione non compare ovviamente si suppone di default il valore vero e si esegue la modifica su tutte le righe della tabella.

Ecco un esempio che aumenta del 20% lo stipendio degli impiegati che si chiamano "Claudio":

```
Update Impiegato Set Stipendio = Stipendio * 1.2
      Where Nome = 'Claudio'
```

Lascio al lettore il compito di interpretare la seguente istruzione di update:

```
Update Impiegato
      Set Stipendio = ( Select Max(Stipendio) from Dirigenti )
```

Where Nome like '_l*di_' and Residenza = 'Sestri Levante'
and Domicilio = 'Milano' and DataNascita > 8-23-1975 and DataNascita
< 8-25-1975

Conclusion

Questo project work nasce dalla volontà di approfondire un mondo, quello delle basi di dati, che reputo tanto affascinante quanto sottovalutato.

E' la natura stessa dei database, difficilmente inquadrabile negli stilemi del mondo informatico, dominato da linguaggi e applicazioni, che li pone in una posizione di sudditanza rispetto a programmi più "affascinati" e meno "noiosi".

Spesso si tende a considerare, per ignoranza, le basi di dati come un'informe contenitore in cui le informazioni vengono buttate e miracolosamente restituite, senza preoccuparsi di cosa si trova dietro delle interfacce spesso poco accessibili ai non addetti ai lavori.

Spesso si sottovalutano i grandi strumenti che i moderni DBMS mettono a disposizione. Anche un sistema desktop come Access, che vanta decine di milioni di installazioni a livello mondiale, è snobbato dagli operatori del mondo informatico; mi riferisco a tutti coloro che hanno un PC con il pacchetto Office installato, e che si affidano ad un foglio elettronico per memorizzare i propri dati, ignari del fatto che gli basterebbe una piccola apertura mentale per trovare in un database un sistema più potente, veloce e facile(!!!) per ottenere risultati senz'altro migliori.

Tuttavia la cultura dominante è non è ancora pronta ad accettare i database come l'unico sistema in grado di custodire e rendere accessibili i dati. La maggior parte delle persone ha ancora bisogno di vedere i dati sullo schermo per essere sicura della loro esistenza; quindi è facile intuire perché la maggior parte dei dati dell'intero pianeta sia memorizzata su piattaforma Excel (notizia di fonte Microsoft) che non è nato per memorizzare dei dati, quanto per fare calcoli. Ma Excel mostra, obbligatoriamente, tutti i dati immessi, ed è per questo che una società (la nostra), non ancora pronta ad una informatizzazione più consapevole, lo preferisce a sistemi di memorizzazione più consoni.

Ma questa situazione è destinata, inevitabilmente, a mutare.

Viviamo in un periodo di estremo fermento informatico : c'è Internet, il PC è un bene di consumo, l'archiviazione elettronica è una realtà, la sicurezza è garantita maggiormente dai sistemi informatici che non da altri mezzi, il livello medio di informatizzazione si sta alzando. E' lecito supporre che il numero informazioni gestite dai sistemi informatici aumenterà a livelli esponenziali, ed è altrettanto lecito supporre che i database saranno l'unica risposta in grado di gestire questa mole di dati.

C'è chi vuole vedere nell'ultima frontiera dell'informatica, Internet, una realtà che risponde a principi completamente diversi a quelli esposti sinora (parlo di caos, mancanza di schemi, "libertà"). A costoro vorrei far notare che Internet sarebbe niente senza strumenti come i motori di ricerca ed i server di posta elettronica, tipici esempi di uso dei DBMS.

Bibliografia:

- Atzeni, Ceri, Paraboschi, Torlone - **Basi di dati** - McGraw Hill
- **MSDN Library** - Microsoft
- **Visual Basic 6** Tutto e oltre - McGraw Hill
- **Visual Basic 6.0** Manuale del programmatore – Microsoft Press