

INTRODUZIONE

In un sistema informatico si individuano sempre due elementi:

- il **PROCESSO**
- l' **INFORMAZIONE**

L'Informazione è rappresentata mediante i dati.

In un sistema informatico (più in generale informativo) i dati di input vengono elaborati per produrre dei risultati.

Per la descrizione dei processi di elaborazione sono stati messi a punto i *Linguaggi di Programmazione*;

La comparsa di una moltitudine di tali strumenti, anche basati su differenti filosofie di concetto e la continua evoluzione degli stessi, è la riprova dell'importanza del procedimento logico rispetto ai dati.

I dati dunque sono solo un logico corollario delle applicazioni informatiche ?

Sì, nelle applicazioni in cui essi sono pochi e semplici.

In molti sistemi informativi (Azienda, Biblioteca, Scuola, ...) il flusso di dati è tale che l'Informazione riveste un ruolo determinante.

In queste realtà, la funzionalità difatti, dipende oltre che dalla validità dei metodi atti a processare l'informazione anche dall'organizzazione dell'informazione stessa.

In un'azienda l'esecuzione delle normali attività sia amministrative che operative, la definizione e la scelta delle politiche commerciali, finanziarie e di quelle relative al personale, sono strettamente legate all'elaborazione di insiemi di dati (nettamente separati fra loro ma correlati) chiamati **archivi**.

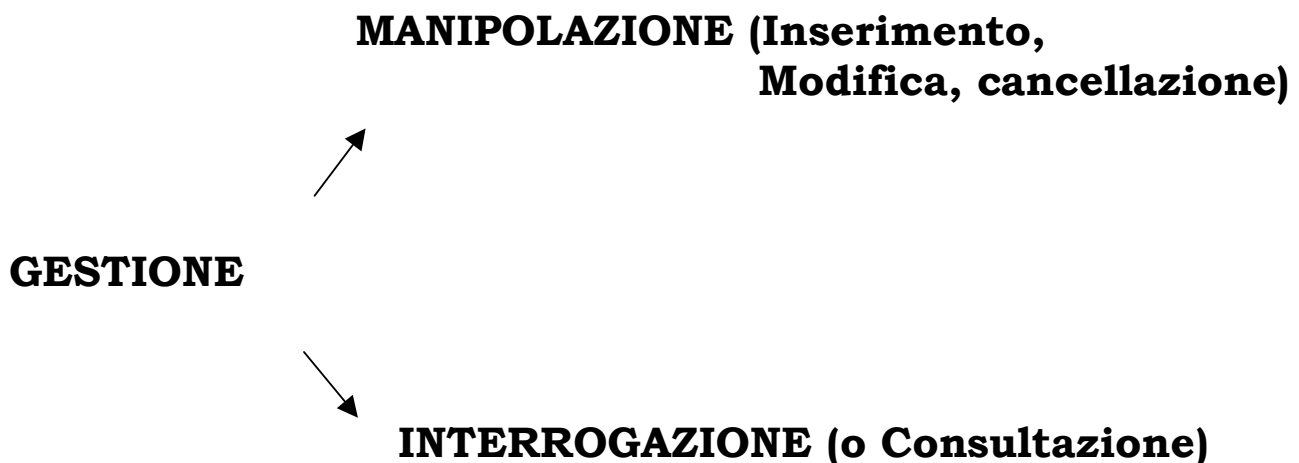
La conservazione dei dati, oltre a rispondere alle suddette esigenze dell'azienda, sempre più spesso rappresenta un'informazione che aggregata o rielaborata costituisce una preziosa fonte per prendere decisioni e aumentare la produttività.

DATA-BASE

Per questo sono nati Software specifici (i DBMS) con funzionalità, appunto, di



dove



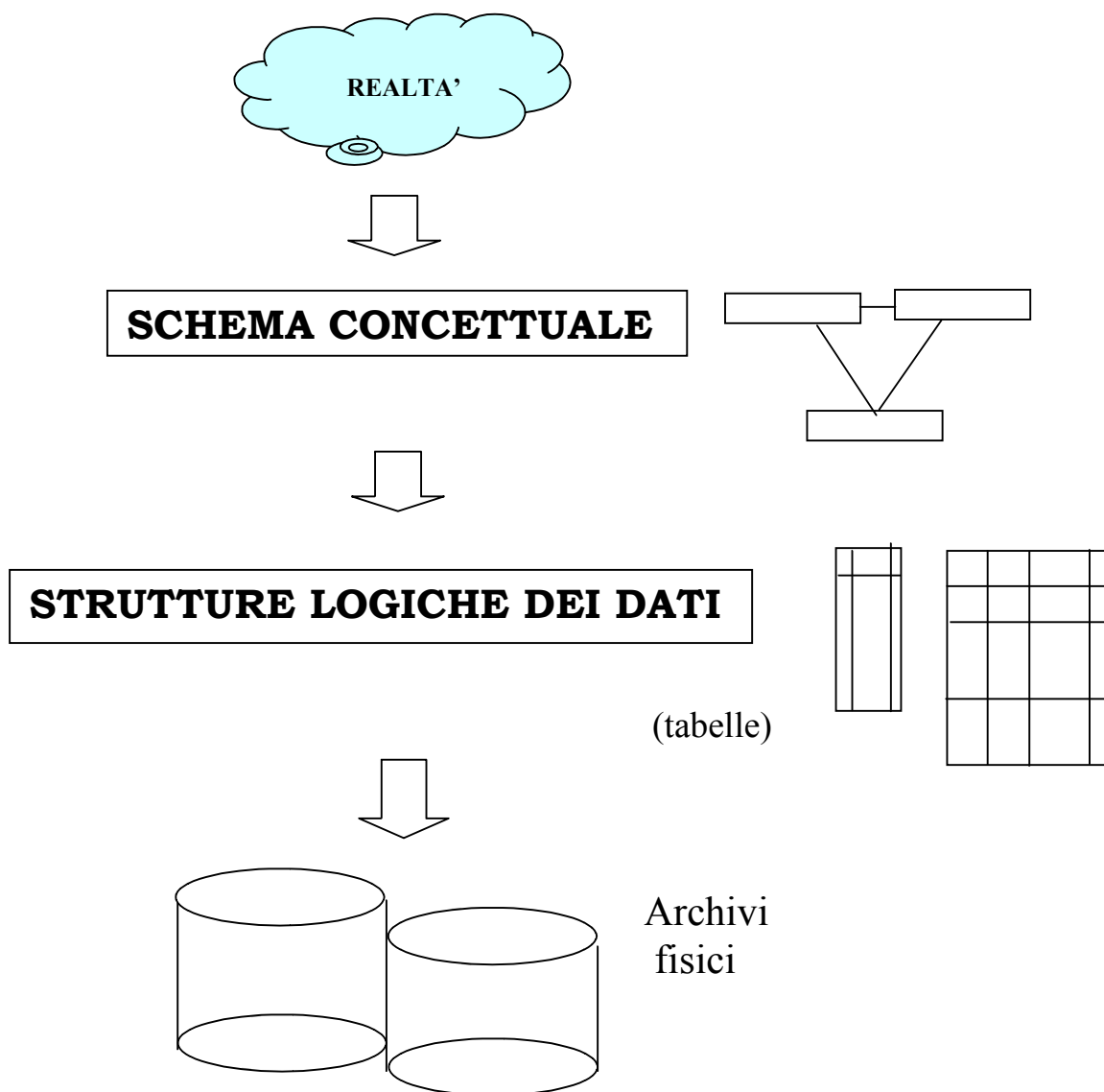
Una Base di Dati (DB) è un insieme di archivi organizzati in modo integrato, conservati nelle memorie di massa del computer e gestiti con appositi programmi.

Le caratteristiche fondamentali che un DB deve possedere sono: la **consistenza** e la **sicurezza** dei dati.

Un DBMS (come Access) è di enorme aiuto nell'introduzione, nel mantenimento e nella gestione di una Base di Dati ma non nella sua definizione, che è a carico dell'utente.

E' l'utente, infatti, che decide quali dati del sistema reale in esame sono significativi e come suddividerli. A supporto di questa fase alquanto delicata, poiché decisiva nella riuscita del prodotto, sono state studiate e realizzate tecniche di rappresentazione formale delle relazioni logiche esistenti tra i dati (es. il Modello E-R). L'utente a conclusione di tale fase produce uno SCHEMA CONCETTUALE.

Il processo di creazione di un DB è sintetizzato dai seguenti passi:



Il presente corso ha come scopo l'acquisizione delle abilità per operare in un DBMS e l'apprendimento del linguaggio in esso presente per la manipolazione e la consultazione dei dati, ma non può prescindere dall'esposizione dei fondamenti per la progettazione di un DB.

Tale breve esposizione non ha la pretesa di spiegare a fondo le tecniche di analisi dei dati, ma soltanto di fornire i concetti che vi sono alla base di una razionale organizzazione delle informazioni.

Per poter preparare uno schema concettuale è necessaria un'

ANALISI DATI

ossia lo studio di quella parte di realtà che è rilevante nel contesto che si sta analizzando.

La specifica natura dei dati, così complessa pur se all'apparenza semplice ed intuitiva, e le relazioni che li uniscono, a prescindere dai processi che li creano e li utilizzano, hanno rappresentato e rappresentano ancora un banco di prova particolarmente significativo.

Porre una considerevole attenzione all'Analisi Dati significa essere consapevoli degli innumerevoli riflessi di stabilizzazione che essa può arrecare all'informatica.

OBIETTIVO DELL' ANALISI DATI

L'Analisi Dati ha come obiettivo la realizzazione di una Base Dati che sia:

- indipendente dalle funzioni
- flessibile
- stabile nel tempo

Uno strumento di analisi e di schematizzazione concettuale dei dati che rispetti gli obiettivi imposti è stato proposto dal matematico *Chen* nel 1976 ed è denominato

MODELLO ENTITA'-RELAZIONI

Una volta individuata la porzione di realtà da analizzare, vi è la necessità di rappresentarla in modo formale.

Tale schematizzazione si basa su elementi abbastanza intuitivi, ma di non facile definizione:

Entità

Relazione

Attributo

ENTITA' - è la rappresentazione concettuale di un oggetto fisico o astratto (persona, luogo, documento, concetto...) che sia distintamente identificabile.

Ogni Entità rappresenta oggetti con caratteristiche concettuali omogenee.

Nel modello concettuale, le Entità sono identificate da una definizione che descrive, con chiarezza, le caratteristiche dell'Entità stessa.

Esempio: Dipendente, progetto, ecc...

RELAZIONE - è un legame (associazione) tra Entità.

Nel modello concettuale le Relazioni sono descritte sotto forma di azioni.

Esempio: Il DIPENDENTE **LAVORA** su un PROGETTO

LAVORA è una relazione tra le entità DIPENDENTE e PROGETTO.

Un'entità può anche avere una relazione con se stessa.

CARDINALITA' DI UNA RELAZIONE

Una Relazione tra due Entità può essere di tre tipi:

- uno a uno (1:1)** rara e legata ad un contesto ben preciso:
1 uomo sposa 1 donna e una donna
sposa 1 uomo
- uno a molti (1:n)** 1 persona nasce in 1 nazione e in 1
nazione nascono N persone
- molti a molti (m:n)** 1 persona visita N città e 1 città è visitata
da M persone

ATTRIBUTO - rappresenta una caratteristica di un Entità o di una Relazione.

Esempio: matricola, dati anagrafici, ..., sono attributi dell'Entità DIPENDENTE.

Ogni attributo assume in un certo istante un valore valido (vedi più avanti: Tipi di Dati).

NORMALIZZAZIONE DELLE INFORMAZIONI

Gli attributi di un modello relazionale devono verificare la seguente condizione (condizione di normalizzazione in 1NF):

ciascun valore di un attributo deve essere atomico, cioè non decomponibile in altri attributi.

Esempio:

l'attributo DATI ANAGRAFICI dell'Entità DIPENDENTE non è normalizzato, in quanto è scomponibile in nominativo, data e luogo nascita, ...

Fanno eccezione gli attributi temporali (date ed orari) e quelli composti destinati ad un uso integrale (codice fiscale).

Non risulta normalizzato neanche un dato ricavabile, come età che . Per da calcoli di qualsiasi tipo.

Vi sono altre condizioni di normalizzazione delle informazioni (2NF, 3NF, BCFN) che permettono di ottenere relazioni in una forma tale da prevenire la ridondanza e la non congruenza dei dati.

MODELLO CONCETTUALE COME SISTEMA APERTO

Un modello concettuale, visto come rappresentazione della parte di realtà in esame, è un sistema aperto, in quanto è sempre possibile “agganciare” ad esso altri modelli relazionali, che descrivono altre parti di realtà, con le quali abbia in comune almeno un elemento.

I PROGETTISTI DI BANCHE DATI

La progettazione di banche dati di una certa rilevanza è affidata a delle figure EDP altamente specializzate che costituiscono il cosiddetto **DBA** (*Data Base Administrator*) – si tratta di una struttura centralizzata in grado di fornire tutte le indicazioni necessarie agli specialisti dello sviluppo del SW, riguardo le informazioni di loro competenza.

Il DBA è responsabile dell'informazione “conservata” nel DB ed è colui che fornisce gli **account** a vari livelli per l'accesso al sistema.

Anche Access fornisce questa opportunità, che ha senso in ambiente multiutente.

STRUTTURE LOGICHE DI DATI – LE TABELLE

Un Data Base Relazionale, dal punto di vista dell'utente, è percepito come un'insieme di tabelle.

La TABELLA rappresenta la struttura fondamentale di un Data Base Relazionale.

La Tabella è una matrice bidimensionale nella quale le righe sono assimilabili ai tradizionali “**records**” e le colonne ai “**campi**”.

La Tabella è, in qualche modo, assimilabile al tradizionale “file”; Tuttavia in un file DB vengono registrate più tabelle. Le righe di una Tabella sono dette anche “*tuple*”.

Ogni Entità ed ogni Relazione del modello concettuale può essere ricondotta ad una Tabella dove ogni ricorrenza viene rappresentata da una riga ed ogni attributo da una colonna.

COLLEGAMENTO LOGICO FRA TABELLE

Teoricamente un modello concettuale rappresentato da N Entità e M Relazioni dovrebbe dar luogo ad un DB Relazionale composto da N+M Tabelle.

In realtà, spesso una Relazione di tipo 1:N viene inglobata nell'Entità che possiede le N ricorrenze associate dalla Relazione.

Esempio: 1 persona vive in 1 comune ed
In 1 comune vivono n persone (relazione 1:n)

Teoricamente dovrebbero aversi tre tabelle:

due per le Entità PERSONA e COMUNE

IDpersona	Cognome	Nome
1	ROSSI	MARIO
2	BIANCHI	PAOLO
3	VERDI	LUIGI

IDcomune	Comune	Prov
1	BRINDISI	BR
2	FASANO	BR
3	LECCE	LE

ed una per la relazione VIVE

IDpersona	IDcomune
1	3
2	2
3	1

In realtà, spesso, si hanno due sole tabelle:

la tabella PERSONA e la tabella COMUNE

IDpersona	Cognome	Nome	Idcomune
1	ROSSI	MARIO	3
2	BIANCHI	PAOLO	2
3	VERDI	LUIGI	1

IDcomune	Comune	Prov
1	BRINDISI	BR
2	FASANO	BR
3	LECCE	LE

La tabella relativa alla relazione VIVE è stata inglobata nella tabella relativa all'entità PERSONA.

CHIAVE PRIMARIA

Viene definita **Chiave Primaria** (*Primary Key*) di una tabella, l'attributo (colonna) o l'insieme di attributi (gruppo di colonne) che identificano univocamente le righe della tabella.

Nella struttura relazionale, qualunque attributo, chiave primaria o meno, può essere utilizzata come criterio di scelta nell'accesso ai dati.

CHIAVE ESTERNA

Viene definita **Chiave Esterna** (*Foreign Key*) quell'attributo (colonna) che è chiave primaria di un'altra tabella.

Una chiave esterna rappresenta il collegamento logico tra due tabelle.

Nell'esempio precedente "IDcomune" è la *foreign key* della tabella PERSONA e costituisce il legame logico con la tabella COMUNE.

NAVIGABILITA'

La capacità di recepire informazioni attraverso collegamenti tra tabelle viene definita "navigabilità".

INTEGRITA' REFERENZIALE

La presenza delle chiavi esterne evidenzia il problema dell'INTEGRITA' REFERENZIALE dei dati.

Operazioni di aggiornamento o cancellazione delle chiavi primarie di una tabella, che risultano essere anche chiavi esterne di altre, devono essere gestite con accortezza, in quanto si rischierebbe di avere delle chiavi esterne senza i corrispettivi collegamenti.

SCHEMA CONCETTUALE → SCHEMA LOGICO

Con riferimento ai DB Relazionali, vedremo ora i passi da seguire per passare dal Modello E/R (schema concettuale) alla definizione dell'insieme di tabelle costituenti la Base di Dati (schema logico):

1. Ogni **ENTITA'** diventa una **tabella**
2. Ogni **ATTRIBUTO** di un'Entità diventa un **campo** nella struttura record della relativa tabella
3. **L'attributo (o gruppo di attributi) che identificano univocamente un'entità** diventeranno la **Chiave Primaria** nel record della relativa tabella
4. La **relazione 1:1** diventa una **tabella unica** il cui tracciato record contiene i campi derivanti dagli attributi delle due entità in gioco
5. L'identificatore univoco dell'entità di partenza in una **associazione 1:n** diventa **chiave esterna** dell'entità di arrivo
6. Una **relazione m:n** diventa una **nuova tabella** (in aggiunta alle tabelle derivanti dalle due entità collegate) il cui record contiene gli identificatori univoci delle due entità e gli eventuali attributi dell'associazione.

Microsoft Access

I prodotti di gestione dei Data Base, relazionali o no, sono detti **DBMS** (Data Base Management System).

Esistono sul mercato diversi DBMS: **DB2, DB3, Oracle, Informix, Access,...**

Il DB2 è molto diffuso in sistemi mainframe; *Oracle* su sistemi mini; DB3 ha avuto molto successo anni addietro tra gli utenti del Personal Computer.

Restando nell'ambiente dei Personal, la MicroSoft è presente sul mercato con **Access**, un DBMS potente, versatile, e semplice da utilizzare (con la veste grafica che caratterizza Windows). Esso fa parte di quei *package* integrati nell'applicazione *Office*.

E' un DB Relazionale ed è fondato sui concetti di **Algebra degli Insiemi**.

Il risultato di richieste fatte a tabelle Access, al fine di estrarre determinate informazioni, è ancora una tabella Access.

A livello fisico, i record sono allocati in **ordine casuale**.

L' OPTIMIZER

L' *Optimizer* è un sistema esperto che definisce i percorsi ed i metodi ottimali di accesso alle tabelle.

Mira al raggiungimento del massimo delle *performances*.

Non è quindi, teoricamente, compito dello specialista EDP la gestione di tali problematiche, anche se in realtà, l'utilizzo di alcuni parametri da esso manipolabili possono provocare un notevole miglioramento od un degrado delle prestazioni.

II CATALOGO

Tutte le operazioni in ambiente Access vengono gestite con l'ausilio del "Catalogo".

E' un dizionario interno autogestito ed inviolabile.

Costituisce il garante di tutte le informazioni relative a:

- strutture dati
- autorizzazioni di vario genere e livello
- programmi operanti
- utilizzazione dei supporti fisici

PRINCIPALI TIPI DI DATI GESTITI DA ACCESS

Il **tipo di dato** indica la natura del dato stesso e conseguentemente le operazioni possibili su di esso.

Le informazioni memorizzate in una tabella Access possono essere di vario tipo; i più comunemente usati sono:

- ◆ **testo** Un campo viene dichiarato di questo tipo quando deve contenere testo o combinazione di testo e numeri (\Leftrightarrow *string* del Pascal).
Utilizzare questo tipo per la trattazione di nomi e cognomi, indirizzi e tutti i numeri che non richiedono calcoli, come numeri di telefono, codici, ...
Dimensione predefinita=50 caratteri;
Dimensione massima=255 caratteri.
- ◆ **numerico** Un campo viene dichiarato di questo tipo quando deve contenere numeri che potranno costituire elementi di calcolo, come l'età, il numero di figli, le distanze, ...

Numerico, però, è un termine molto generico; in matematica sappiamo bene che esistono diversi insiemi di numeri: interi naturali, interi relativi, numeri con la virgola. Access permette di puntualizzare ciò attraverso le seguenti impostazioni:

byte	per numeri compresi tra 0 .. 255
intero	per numeri compresi tra -32768 .. +32767
intero lungo	per numeri compresi tra -2147483648 .. +2147483647
Precisione singola	per numeri reali compresi tra -3.4x10 ³⁸ .. -1.4x10 ⁻⁴⁵ (nell'intorno sinistro dello 0), 1.4x10 ⁻⁴⁵ .. 3.4x10 ³⁸ (nell'intorno dx dello 0), con una precisione di 7 cifre decimali
precisione doppia	per numeri reali molto grandi, nell'ordine di $\approx 10^{\pm 300}$, con una precisione decimale di 15 cifre

- ◆ **si/no** (↔ *boolean* del Pascal) – adatto alle informazioni del tipo vero/falso, come nelle situazioni di “conto pagato”, “porta gli occhiali”, “pratica evasa”,...

ALTRI TIPI DI DATI

Un particolare tipo di dato che Access mette a disposizione è il tipo

- ◆ **contatore** che consente di creare campi che immettono un numero automaticamente quando viene aggiunto un record alla tabella.
E' consigliabile utilizzarli per le chiavi primarie.
- ◆ **memo** Equivalente al tipo testo con la differenza di una

maggiore capacità: utilizzare un campo memo se occorre memorizzare più di 255 caratteri (ne gestisce fino a un max di 64000).

Sui campi memo, però, non è possibile né stabilire una indicizzazione né definire un criterio di ordinamento.

◆ **valuta**

di carattere numerico ma più indicato per la gestione di dati che corrispondono a stipendi, costi, ...

Tuttavia è possibile utilizzarlo anche in altri contesti (per dati contenenti da una a 4 cifre decimali): quando per un campo si prevede un alto tasso di utilizzo in calcoli matematici è addirittura conveniente dichiararlo VALUTA, in quanto tale tipo usa la notazione *fixed-point* e ciò implica maggiore rapidità di calcolo, al contrario dei tipi PRECISIONE SINGOLA e PRECISIONE DOPPIA che richiedono calcoli in *floating-point*.

ATTRIBUTI DI TIPO DATA/ORA

Gli attributi di tipo DATA e ORA consentono la gestione delle informazioni temporali costituite da date ed orari.

Queste particolari tipologie risultano di grande utilità (si pensi alla gestione delle stesse con i linguaggi procedurali), in quanto permettono di effettuare conversioni, operazioni aritmetiche e funzioni aggiuntive, tali da sopperire alla maggior parte delle esigenze applicative.

La loro rappresentazione fisica interna a *MicroSoft Access* è difforme rispetto ai formati presentati all'utente finale.

Access prevede diversi standard di formato, sia per le date che per gli orari:

data in cifre	es. 5/12/99
data estesa	es. domenica 5 dicembre 1999
data media	es. 5 dic 99
orario esteso	es. 20.35.23
orario media	es. 8.35 PM
orario breve	es. 20.35

E' possibile creare formati di data e ora personalizzati !

IL VALORE “NULL”

Un attributo viene definito NULL quando si vuole indicare che al suo interno vi è un' **assenza di informazione**.

Nei DBMS non sono ammessi, dal punto di vista logico, i valori “0” o “blank” come sostitutivi del NULL; essi rappresentano comunque un'informazione.

INDICI

Gli indici sono oggetti Access estranei alla teoria relazionale e creati unicamente per migliorare le prestazioni del sistema.

L'indice è sostanzialmente un puntatore fisico gestito totalmente da Access; dunque è trasparente all'utente, il quale deve soltanto sapere che:

- 2 conviene definire **indicizzati** i campi che costituiscono chiavi per la ricerca e l'ordinamento;
- 2 le chiavi primarie sono indicizzate automaticamente da Access.

Il linguaggio di INTERROGAZIONE delle BASI di dati

S Q L

(Structured Query Language)

L' **SQL (Structured Query Language)** è un linguaggio di IV generazione, rivolto all'utente finale:

E' stato sviluppato per rendere agevole sia la consultazione che la manipolazione dei dati rappresentati in un DB relazionale.

E' detto anche Linguaggio di interrogazione delle Basi di Dati.

L' SQL è logicamente suddiviso in due sottoinsiemi:

DDL che permette la definizione degli oggetti

DML che permette di manipolarli

Su DBMS di vecchia concezione e operanti in ambiente Mainframe (come DB2) la creazione delle tabelle avviene attraverso istruzioni in un linguaggio formale, il DDL appunto, ed è a carico del DBA.

Il DDL di Access è completamente *visuale* e permette la creazione delle tabelle in una maniera alquanto semplice

Per quanto riguarda il DML, per molti DBMS, la codifica di istruzioni secondo la sintassi di tale linguaggio, costituisce la sola interfaccia con i dati.

Access, invece, per manipolare ed interrogare i dati, prevede due modalità:

- mediante la scrittura di codice (descrizione di forme sintattiche opportune)
- quella gestita con veste grafica (orientata agli utenti finali).

QUERY

Manipolare i dati di un DB significa visualizzare, modificare ed analizzare i dati in modi diversi.

Tali operazioni vengono espresse con comandi dell' SQL che prendono il nome di **query** (=richiesta, interrogazione).

Gli oggetti sui quali agisce una query sono le tabelle, che vengono viste, a questo livello, come degli insiemi; le operazioni di query sono, difatti, le operazioni classiche dell'insiemistica: **selezione**, **intersezione**, **unione**, ...

Qualsiasi richiesta al DB relazionale restituirà all'utente tutti quegli elementi che soddisfano la richiesta stessa.

L'SQL è un linguaggio ricorsivo: Access si fa carico di iterare l'istruzione che esprime la richiesta su tutta la tabella, contrariamente alla tradizionale elaborazione "record per record".

L'SQL può essere ospitato da un linguaggio di elaborazione: ossia le informazioni che SQL restituisce a fronte di una query, possono essere elaborate dal linguaggio di programmazione. (previsto a tale scopo), detto

Per es. COBOL e PL/1 si possono interfacciare con DB2;
C (attraverso *Pro C*) ad ORACLE;

Access ha Visual Basic for Application (VBA) come linguaggio "ospite" per la sua personalizzazione ma può essere gestito da numerosi linguaggi di programmazione.

QUERY DI SELEZIONE

E' il tipo più comune di query, e serve ad estrarre certe informazioni da una o più tabelle in base a determinate condizioni.

In generale, ha la seguente struttura sintattica:

SELECT lista informazioni da estrarre
FROM lista delle tabelle interessate
WHERE criteri di estrazione (sotto forma di proposizioni booleane)
..... **opzioni** di ordinamento e/o raggruppamento dei dati ottenuti

Gli elementi contenuti in una lista devono essere separati da una virgola.

L'ordine con cui appaiono gli elementi nella lista che segue la *keyword* SELECT è lo stesso che si ha nell'esposizione del risultato.


Nell'espressione booleana che stabilisce i criteri di estrazione, si possono usare gli operatori di relazione

= < > <= >= <> (diverso)

e i connettivi logici

not and or

La parte relativa a WHERE e/o alle OPZIONI può essere omessa.

In Access la query può essere scritta nella forma appena vista e successivamente eseguita (modalità ) , oppure è possibile

definirla attraverso una finestra di “autocomposizione” indicando informazioni da estrarre, criteri di selezione ed opzioni su una griglia predisposta (modalità `Query`).

Si tenga presente il 2° metodo non implica un diverso comportamento del sistema; anzi una query definita in questo modo viene tradotta nella forma descrittiva del 1° metodo dal sistema ad opera di un generatore di codice.

SELEZIONE SEMPLICE

```
SELECT cognome, nome  
FROM dipendenti
```

fornisce l'elenco nominativo di tutti i dipendenti:

cognome	nome
Rossi	Mario
Verdi	Luigi
Bianchi	Antonio
Brambilla	Giovanna
Bruni	Angela
Galati	Nunzio

L'elenco viene presentato così com'è registrato.

Come Selezionare tutti i CAMPI

```
SELECT *  
FROM dipendenti
```

fornisce l'elenco completo delle informazioni dei dipendenti:

matricola	cognome	nome	data_nas	sex	data_ass	data_ces	codliv	Codbranc	Supmin
1	Rossi	Mario	01/03/70	M	15/06/95		1	1	1000000
2	Verdi	Luigi	20/02/68	m	01/05/88	01/07/97	1	1	1500000
3	Bianchi	Antonio	18/08/76	m	12/01/96		3	2	800000
4	Brambilla	Giovanna	03/03/79	f	25/05/98		4	3	650000
5	Bruni	Angela	14/08/77	f	25/05/98		4	3	0

6	Galati	Nunzio	01/12/73	m	05/10/97		1	2	1200000
---	--------	--------	----------	---	----------	--	---	---	---------

Si ha lo stesso effetto con:

```
SELECT matricola,cognome,nome,data_nasc,sexo,data_ass,  
data_cess,codLiv,codBranch,supmin  
FROM dipendenti
```

SELEZIONE CONDIZIONATA

```
SELECT nome  
FROM dipendenti  
WHERE sesso='f'
```

nome
Giovanna
Angela

fornisce i nomi dei dipendenti donne.

L'identico risultato si ottiene con la query:

```
SELECT nome  
FROM dipendenti  
WHERE sesso<>'m' (o ancora WHERE NOT sesso="M")
```

Osservazione__

La **SELECT** condiziona il numero di elementi da selezionare;

La **WHERE** condiziona il numero di righe da prendere in considerazione.

Gli attributi che compaiono nella **WHERE**, come elementi di condizione, possono non apparire nella lista degli elementi da estrarre.

Come trattare i valori NULL

```
SELECT nome, cognome  
FROM dipendenti  
WHERE data_cess IS NULL
```

fornisce l'elenco nominativo dei dipendenti in servizio

nome	cognome
Mario	Rossi
Antonio	Bianchi
Giovanna	Brambilla
Angela	Bruni
Nunzio	Galati

mentre

```
SELECT nome, cognome  
FROM dipendenti  
WHERE NOT data_cess IS NULL
```

dà i dipendenti non più in forza all'azienda.

nome	cognome
Luigi	Verdi

Altri esempi con OPERATORI LOGICI e COMPARATIVI

```
SELECT cognome, nome, data_ass  
FROM dipendenti  
WHERE data_ass >= #1/1/97#
```

fornisce informazioni dei dipendenti assunti dall' 1/1/97

cognome	nome	data_ass
Brambilla	Giovanna	25/05/98
Bruni	Angela	25/05/98
Galati	Nunzio	05/10/97

Le DATE e gli ORARI quando fanno parte di una condizione di query, vanno delimitati dal carattere “cancellotto” (#).

Per conoscere il nome e la data di assunzione dei dipendenti assunti prima della suddetta data ma ancora in servizio, bisogna eseguire

```
SELECT cognome, nome, data_ass  
FROM dipendenti  
WHERE data_ass < #1/1/97# AND data_cess IS NULL
```

Cognome	Nome	data_ass
Rossi	Mario	15/06/95
Bianchi	Antonio	12/01/96

mentre, per conoscere coloro che sono stati assunti prima del 1/1/95 e ancora in servizio oppure che sono cessati a partire dal 31/12/97 bisogna eseguire

```
SELECT cognome, nome, data_ass, data_cess  
FROM dipendenti  
WHERE data_ass < #1/1/95# AND  
      (data_cess IS NULL OR data_cess >=#31/12/97#)
```

con riferimento al DB Azienda, questa query non estrarrà alcuna riga.

Avvertenze sugli OPERATORI LOGICI

Porre attenzione all'uso combinato di AND, OR e NOT; la priorità seguita da Access è quella dell'Algebra di Boole. Tuttavia conviene fare uso il più possibile delle parentesi

Si evitano risultati indesiderati
Si rende più leggibile la query

Eliminazione delle RIGHE DUPLICATE (clausola DISTINCT)

```
SELECT codBranch  
FROM dipendenti
```

WHERE codLiv = 1

fornisce l'elenco dei branch con almeno un dipendente inquadrato al I° livello:

codBranch
1
1
2

SELECT DISTINCT codBranch
FROM dipendenti
WHERE codLiv = 1

codBranch
1
2

evita la ripetizione di righe identiche

ORDINAMENTO DEI DATI ESTRATTI (clausola ORDER BY)

La tabella risultato di una query ha un ordinamento casuale. Per ordinare i dati selezionati secondo una o più chiavi si utilizza l'opzione ORDER BY.

Nell' ORDER BY si specifica

I campi (della SELECT) da ordinare

Il tipo di ordinamento

I campi da ordinare si possono identificare

Con il loro nome

Con la posizione nella lista (della SELECT)

Il tipo di ordinamento è **ASC** (ascending) o **DESC** (descending)

Il formato generale è

ORDER BY elem ASC/DESC, elem ASC/DESC, ...

ASC è l'opzione di default.

Esempi:

```
SELECT codBranch, cognome, nome  
FROM dipendenti  
WHERE data_cess IS NULL  
ORDER BY codBranch, cognome (oppure ORDER BY 1,2)
```

Fornisce l'elenco dei dipendenti in servizio, ordinati per branch e, all'interno di questo, per cognome:

codBranch	Cognome	nome
1	Rossi	Mario
2	Bianchi	Antonio
2	Galati	Nunzio
3	Brambilla	Giovanna
3	Bruni	Angela

```
SELECT cognome, data_nasc  
FROM dipendenti  
ORDER BY 2 DESC
```

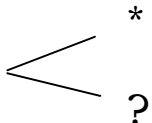
Visualizza tutti i dipendenti, dal più giovane al più anziano:

cognome	Data_nasc
Brambilla	03/03/79
Bruni	14/08/77
Bianchi	18/08/76
Galati	01/12/73
Rossi	01/03/70
Verdi	20/02/68

Selezione con parte di un valore: Opzione LIKE

L'opzione LIKE viene associata a WHERE, e seleziona un'informazione utilizzando solo una parte del suo valore.

Il suo uso è lecito per attributi di tipo "stringa"

Nella LIKE si usano due simboli 

Il simbolo "*" ha il significato di:
"stringa di zero o più caratteri con qualsiasi valore"

Il simbolo "?" ha il significato di:
"ogni singolo carattere in quella posizione"

In pratica, funzionano come i caratteri jolly nel DOS.

Esempi:

```
SELECT cognome, nome  
FROM dipendenti  
WHERE cognome LIKE 'B*'
```

riporta i nominativi dei dipendenti il cui cognome inizia con B.

```
SELECT cognome, nome  
FROM dipendenti  
WHERE nome LIKE '*a*'
```

Da come risultato i dipendenti nel cui nome compare almeno una "a".

```
SELECT cognome  
FROM dipendenti
```

```
WHERE cognome LIKE '????i'  
ORDER BY 1
```

riporta [Bruni, Rossi e Verdi] in quanto il cognome è esattamente di 5 caratteri e il 5° è una “i”.

```
SELECT cognome  
FROM dipendenti  
WHERE cognome LIKE '??a*'
```

riporta [Bianchi e Brambilla] in quanto hanno nel loro cognome come 3° carattere una “a”.

L'opzione IN

A volte in presenza di condizioni multiple legate logicamente con delle OR, risulta più semplice utilizzare l'opzione IN, che presenta il seguente formato:

IN (elemento, elemento, ...)

Dove quanto è inserito tra parentesi è considerato un insieme dal punto di vista matematico e IN equivale ad “appartiene” (∈)

```
SELECT cognome, nome  
FROM dipendenti  
WHERE codLiv=1 OR codLiv=3 OR codLiv=4
```

può anche essere, più comodamente, scritta come

```
SELECT cognome, nome  
FROM dipendenti  
WHERE codLiv IN (1, 3, 4)
```

Per escludere un insieme di elementi:

```
SELECT cognome, nome  
FROM dipendenti  
WHERE NOT codLiv IN (1, 3, 4)
```

L'opzione BETWEEN

Quando si desidera condizionare la selezione al fatto che un attributo debba essere compreso tra due limiti, è possibile

utilizzare gli operatori di comparazione (<, >, <=, >=, =), ma solitamente, si ricorre all'opzione BETWEEN, che presenta il seguente formato:

BETWEEN valore1 AND valore2

dove valore1 e valore2 sono i limiti, rispettivamente, inferiore e superiore, entro i quali si svolge la selezione.

Così

```
SELECT cognome, nome  
FROM dipendenti  
WHERE data_ass >= #1-1-97# AND data_ass <=#25-5-98#
```

risulta perfettamente equivalente alla query

```
SELECT cognome, nome  
FROM dipendenti  
WHERE data_ass BETWEEN #1-1-97# AND #25-5-98#
```

e produrrà

cognome	nome
Brambilla	Giovanna
Bruni	Angela
Galati	Nunzio

che dimostra gli estremi definiti con la BETWEEN sono inclusi nell'intervallo di selezione.

La clausola COUNT

Permette di ottenere un conteggio delle righe che soddisfano la condizione di selezione.

Esempio:

```
SELECT COUNT (*)  
FROM dipendenti  
WHERE cognome LIKE 'B*'
```

fornirà il numero di dipendenti il cui cognome inizia per “B”, ossia:

Expr1000
3

La clausola COUNT esclude dal conteggio i valori NULL.

La clausola SUM

SUM somma i valori di una colonna; l'attributo relativo a quella colonna deve essere numerico e non NULL.

```
SELECT SUM(supMin)  
FROM dipendenti
```

restituisce il totale dei superminimi assegnati a tutti i dipendenti a tutti i dipendenti:

Expr1000
5150000

La seguente query, più articolata, restituisce quanti dipendenti attualmente in servizio percepiscono il superminimo e il totale dei superminimi assegnati

```
SELECT COUNT (matricola), SUM(supmin)
FROM dipendenti
WHERE (data_cess IS NULL) AND (supmin>0)
```

Expr1000	Expr1001
4	3650000

RINOMINARE LE COLONNE DELLA TABELLA RISULTATO

Abbiamo visto che l'effetto di una query di selezione non è altro che l'estrazione di una "porzione" della tabella originale in questione, presentata ancora in forma di tabellare (tabella risultato) e nella quale, alle colonne, viene assegnato dal sistema, lo stesso nome simbolico attribuito in fase di progettazione.

Quando i valori della tabella risultato non sono quelli estratti dalla tabella in questione ma bensì sono frutto di un calcolo su tali valori (come per COUNT, SUM e altri), il sistema assegna alle colonne un nome fittizio (expr1000, expr1001, expr1002, ...) come avrete sicuramente notato negli esempi precedenti.

In effetti è poco leggibile riepilogare con "supmin" un dato che rappresenta un totale, o con "matricola" il numero di record registrati nella tabella DIPENDENTI.

SQL di Access permette di assegnare dei nomi alle colonne della tabella risultato, attraverso la keyword **AS** .

Se ciò è indispensabile nel caso di SUM,COUNT, ... per sostituire il nome temporaneamente assegnato dal sistema, ritorna utile in ogni situazione in cui si voglia migliorare, dal punto di vista della chiarezza, la presentazione dei risultati.

Per esempio, riscrivendo la query di pag. 27, come segue:

```
SELECT cognome, data_nasc AS [Data di nascita]  
FROM dipendenti  
ORDER BY data_nasc DESC
```

Eseguendola, si ottiene:

cognome	Data di nascita
Brambilla	03/03/79
Bruni	14/08/77
Bianchi	18/08/76
Galati	01/12/73
Rossi	01/03/70
Verdi	20/02/68

dove il campo “data_nasc” è stato dettagliato.

Allo stesso modo si può rendere più chiaro il riepilogo prodotto dalla query descritta a pag. 32:

```
SELECT COUNT (matricola) AS [Dipendenti in servizio incentivati],  
SUM(supmin) AS [totale incentivazioni attribuite]  
FROM dipendenti  
WHERE (data_cess IS NULL) AND (supmin>0)
```

Dipendenti in servizio incentivati	totale incentivazioni attribuite
4	3650000

L' opzione **GROUP BY**

Serve per raggruppare le informazioni ottenute in base a un attributo o a un calcolo.

Per esempio, per avere un riepilogo col numero di dipendenti in servizio inquadrati in ogni livello previsto:

```
SELECT DISTINCT (codLiv) AS [Livello],  
COUNT(*) AS [Nro dipendenti inquadrati]  
FROM dipendenti  
WHERE data_cess IS NULL  
GROUP BY codLiv
```

Livello	Nro dipendenti inquadrati
1	2
3	1
4	2

OPERAZIONI su ATTRIBUTI e con le DATE

Come già visto per COUNT e SUM, nella lista di SELECT, oltre a nomi di campi, possono comparire funzioni di calcolo; a tal proposito Access permette di esprimere situazioni di calcolo anche complesse mediante operazioni e funzioni matematiche sui campi.

```
SELECT matricola, supmin * 10 / 100 AS [superminimo  
aumentato del 10%]  
FROM dipendenti  
WHERE (data_cess IS null) AND (supmin > 0)
```

visualizza il superminimo aumentato del 10% per chi già lo possiede:

matricola	Superminimo aumentato del 10 %
1	1100000
3	880000
4	715000
6	1320000

```
SELECT count(*) AS [Totale dipendenti interessati]  
FROM dipendenti  
WHERE (data_cess IS null) AND (supmin > 0) AND  
((supmin + supmin * 10 / 100) < 1000000)
```

fornisce il numero di dipendenti in servizio aventi già un superminimo, per i quali lo stesso resterebbe al di sotto del milione anche a fronte di un aumento del 10%.

Nro dipendenti interessati
2

Anche sulle DATE sono definite delle operazioni.

Per esempio, è possibile, semplicemente sottraendo due campi

definiti DATE, conoscere il numero di giorni che intercorrono. Sempre con riferimento al DB_Azienda, può essere utile sapere il periodo lavorativo (in giorni) dei dipendenti che hanno lasciato l'azienda:

```
SELECT cognome & " " & nome AS [Nominativo],  
          data_cess – data_nasc AS [Periodo lavorativo (in giorni)]  
FROM dipendenti  
WHERE NOT (data_cess IS null)
```

Nominativo	Periodo lavorativo (in giorni)
Verdi Luigi	3348

dove si è usato per la prima volta l'operatore di **concatenazione (&)** fra campi TESTO.

Oltre all'operatore di sottrazione (-), tra date ha significato anche l'operatore di addizione (+).

Ad ogni modo l'utilizzo di tali operatori aritmetici in espressioni contenenti date determina il risultato in giorni.

Esistono delle funzioni più potenti per la manipolazione di date, e sono:

✓ **DateDiff (tipocalcolo, data1, data2)**

Fornisce la differenza in giorni, mesi o anni in base a *tipocalcolo*, tra *data1* e *data2*.

Es. DateDiff(“d”, data_consegna, data_pagamento)

Ritorna i giorni (d ⇔ day) trascorsi tra la data di consegna e la data di pagamento.

Tipocalcolo può essere d | m | yyyy

✓ **DateAdd (*tipocalcolo*, *valore*, *data*)**

Fornisce una data che è il risultato tra *data* e *valore* il cui significato è indicato da *tipocalcolo*.

Es. DateAdd(“m”, 7, data_nascita)

Addiziona 7 mesi alla data di nascita, ottenendo la data corrispondente.

✓ **DatePart (*tipocalcolo*, *data*)**

Permette di considerare l'informazione giorno, mese, o anno (dipendentemente da come è specificato il 1° parametro) di cui è composta data, come dato disgregato.

Es. DatePart(“yyyy”, data_laurea)

Dall'attributo data_laurea definito di tipo DATE ricava la parte corrispondente all'anno.

✓ **Date()**

La funzione date senza parametri ritorna la data odierna

(data di sistema).

Esempi di query:

E' evidente l'identico risultato della query di pag. 36, si ottiene con la seguente query, che sfrutta DateDiff in luogo dell'operatore (-):

```
SELECT cognome & " " & nome AS [Nominativo],  
        DateDiff("d",data_nasc, data_cess) AS [Periodo  
        lavorativo (in giorni)]  
FROM dipendenti  
WHERE NOT (data_cess IS null)
```

Mentre

```
SELECT cognome, DateDiff("yyyy",data_ass, data_cess) AS  
        [anni di servizio]  
FROM dipendenti  
WHERE cognome="verdi"
```

cognome	anni di servizio
Verdi	9

fornisce quanti anni ha prestato servizio l'impiegato Verdi.

Ora, si suppone di dover calcolare la data da quando ha avuto inizio il pagamento dello stipendio sindacale per i dipendenti assunti nel 98, ipotizzando che nei primi 120 giorni dalla data di assunzione (periodo di formazione) vigevo un 'altra forma di pagamento deciso dall'azienda; la query che risponde a tale esigenza è la seguente:

```
SELECT matricola, cognome,  
        DateAdd("d",120, data_ass) AS [Decorrenza giuridica]  
FROM dipendenti  
WHERE DatePart("yyyy", data_ass)=1998
```

e il risultato che ne consegue è il seguente

matricola	Cognome	Decorrenza giuridica
4	Brambilla	22/09/98
5	Bruni	22/09/98

La seguente query

```
SELECT cognome, nome,  
        Datediff("yyyy",data_nasc,date()) AS [ETA']  
FROM dipendenti  
ORDER BY 3,1
```

elenca i nominativi di tutti i dipendenti con le relative età, dal più giovane al più anziano:

cognome	nome	ETA'
Brambilla	Giovanna	20
Bruni	Angela	22
Bianchi	Antonio	23
Galati	Nunzio	26
Rossi	Mario	29
Verdi	Luigi	31

FUNZIONI DI AGGREGAZIONE

Sono funzioni che consentono di calcolare vari indici statistici su gruppi di valori.

Funzioni di questo tipo, già esaminate, sono **COUNT** e **SUM**.

Altre funzioni previste, sono:

MIN, MAX determinano rispettivamente il valore minimo e il valore massimo sui dati selezionati.

AVG Calcola la media di una serie di valori.

STDEV Calcola la deviazione standard.

Esempio

```
SELECT Min(supmin) AS MIN, Max(supmin) AS MAX,  
Avg(supmin) AS MEDIO  
FROM dipendenti
```

produce il seguente riepilogo statistico sui valori del superminimo:

MIN	MAX	MEDIO
0	1500000	858333,333333333

L'opzione HAVING

E' usata con le funzioni di aggregazione per definire una ulteriore selezione sui gruppi di dati ottenuti tramite la GROUP BY.

La seguente query restituisce il totale dei dipendenti per ogni livello insieme al totale dei superminimi se tutti i dipendenti di quel livello posseggono un superminimo:

```
SELECT DISTINCT (codLiv) AS [Livello], COUNT(*) AS [totale  
dipendenti], SUM(supmin) AS [totale superminimi]  
FROM dipendenti  
GROUP BY codLiv  
HAVING MIN(supmin)>0
```

Livello	Totale dipendenti	totale superminimi
1	3	3700000

3	1	800000
---	---	--------

Predicati di selezione: TOP e IIF

I predicati di selezione sono quelle parole chiavi che possono essere combinate con un campo della SELECT allo scopo di condizionare la scelta dei valori di quel campo. DISTINCT ne è un esempio. Un'altra possibilità interessante è data dal predicato **TOP**, il cui funzionamento è chiarificato dal seguente esempio:

```
SELECT TOP 2 stipendio AS [stipendi più alti]
FROM livelli
ORDER BY stipendio DESC
```

Visualizza i 2 maggior stipendi:

Stipendi più alti
L. 50.000.000
L. 40.000.000

Per ottenere i due più bassi basta cambiare l'opzione di ordinamento in ASC.

Il predicato **IIF** ha il seguente formato

IIF (*condizione*, *stringa1*, *stringa2*)

IIF valuta la *condizione*: se risulta vera visualizza *stringa1*, altrimenti visualizza *stringa2*.

Esempio

La seguente SELECT fa uso di IIF per listare i 4 dipendenti in servizio con maggior numero di anni di anzianità, riportando, oltre al cognome, se trattasi di dipendente uomo o dipendente donna:

```
SELECT TOP 4 datepart("yyyy",date()) - datepart("yyyy",data_ass)
      AS [anzianità di servizio], cognome,
      IIF(sezzo="M", "UOMO","DONNA") AS [UOMO/DONNA]
FROM dipendenti
WHERE data_cess is null
ORDER BY 1 desc
```

Anzianità di servizio	cognome	UOMO/DONNA
4	Rossi	UOMO
3	Bianchi	UOMO
2	Galati	UOMO
1	Bruni	DONNA
1	Brambilla	DONNA

OSS. In questo caso specifico, i record che il predicato TOP ha selezionato dalla tabella risultato sono 5 anziché 4 come specificato. Questo, perché, gli ultimi due dipendenti hanno esattamente lo stesso valore e, in questi casi, TOP li considera entrambi.

OPERAZIONI DI JOIN

Gli esempi finora riportati hanno riguardato interrogazioni basate su un'unica tabella.

A questo punto non siamo ancora in grado di definire, per esempio, una query che riporti “lo stipendio di un certo dipendente”, in quanto una tale operazione riguarderebbe due tabelle: DIPENDENTI e LIVELLI.

Le operazioni di JOIN (congiunzione) consentono di collegare logicamente i dati appartenenti a più tabelle, appunto, sulla base di una condizione specifica.

La congiunzione è un'operazione fatta dal DBMS a livello fisico producendo una tabella temporanea costituita da tutte le combinazioni di record delle tabelle coinvolte (effettua cioè un prodotto cartesiano).

Se T1 e T2 rappresentano due tabelle coinvolte in una operazione di JOIN, il prodotto cartesiano si denota con

$$T1 * T2$$

e

$$cardinalità(T1*T2) = cardinalità(T1) \times cardinalità(T2)$$

dove per *cardinalità* si intende il numero di righe di cui è composta la tabella.

Dalla tabella temporanea T1*T2, il sistema determina la tabella risultato, filtrando determinate righe, in base alla

Tipologia di JOIN

- **EQUI-JOIN (o JOIN NATURALE)**

Vengono prese in considerazione le righe nelle quali, i valori dei campi in comune (specificati nella condizione), corrispondono;

- **NOT EQUI-JOIN**

Vengono prese in considerazione le righe nelle quali, i valori dei campi in comune (specificati nella condizione), non corrispondono;

Access prevede altri due tipi di Join poco utilizzati:

- **LEFT-JOIN**

Nella tabella risultato vengono inclusi, comunque, tutti i record della prima tabella (di sinistra), anche se ad alcuni dei campi in comune non corrisponde nessun valore nell'altra tabella;

- **RIGHT-JOIN**

Nella tabella risultato vengono inclusi, comunque, tutti i record della seconda tabella (di destra), anche se ad alcuni dei campi in comune non corrisponde nessun valore nell'altra tabella.

Esempi:

La seguente query interessa due tabelle e serve ad estrarre il profilo professionale di ogni dipendente:

SELECT cognome, nome, des AS ruolo

FROM dipendenti, livelli
WHERE dipendenti.codliv = livelli.codliv
ORDER BY 1

Cognome	Nome	Ruolo
Bianchi	Antonio	Programmatore Sr
Brambilla	Giovanna	Programmatore Jr
Bruni	Angela	Programmatore Jr
Galati	Nunzio	Progettista
Rossi	Mario	Progettista
Verdi	Luigi	Progettista

Si può notare che la distinzione tra l'attributo "codLiv" della tabella DIPENDENTI e l'attributo "codLiv" della tabella LIVELLI si è ottenuta antepo-
nendo al nome dell'attributo il nome della tabella che lo contiene, separati dal punto (.) (**riferimento puntuale**). Il riferimento puntuale a un campo, dunque, si denota specificando, accanto ad esso, la sua tabella di origine.

Questa è una regola generale di Access, che è necessario applicare tutte le volte nelle quali la query riguarda campi omonimi di differenti tabelle.

Questo problema non esiste

- per le query che interessano una sola tabella;
- per le query che, nonostante coinvolgano più tabelle, referenziano campi identificati tutti con un nome diverso.

Tuttavia, nel 2° caso, è consigliabile specificare la tabella di origine dei campi, per ragioni di chiarezza.

Esiste un'altra forma per stabilire un riferimento puntuale ad un campo di una determinata tabella, che consiste nell'associare un'etichetta (una stringa simbolica) ad ogni tabella, come illustrato nell'esempio seguente:

```

SELECT A.cognome, A.nome, B.des AS [livello], C.des AS
[area di lavoro], B.stipendio AS [stipendio base],
IIF(A.data_cess IS NULL, "SI", "NO") AS [in servizio]

FROM dipendenti A, livelli B, branch C
WHERE A.codLiv = B.codLiv
        AND
        A.codBranch = C.codBranch
    
```

Che estrae le informazioni riassunte dalla seguente tabella risultato:

cognome	Nome	livello	area di lavoro	stipendio	in servizio
Rossi	Mario	progettista	trasporti	L. 50.000.000	SI
Verdi	Luigi	progettista	trasporti	L. 50.000.000	NO
Bianchi	Antonio	programmatore	enti locali	L. 35.000.000	SI
Brambilla	Giovanna	programmatore Jr	banche	L. 30.000.000	SI
Bruni	Angela	programmatore Jr	banche	L. 30.000.000	SI
Galati	Nunzio	progettista	enti locali	L. 50.000.000	SI

Un'applicazione dell'operazione di NOT EQUI-JOIN è quella di ricavare, dalle informazioni contenute nel DB, i probabili branch sui quali ogni dipendente potrà essere spostato nell'ipotesi di una mobilità del personale:

```

SELECT cognome, nome, B.des AS trasferibilità
FROM dipendenti A, branch B
WHERE A.codbranch <> B.codbranch
    
```

Cognome	nome	trasferibilità
Rossi	Mario	enti locali
Rossi	Mario	banche
Verdi	Luigi	enti locali
Verdi	Luigi	banche
Bianchi	Antonio	trasporti
Bianchi	Antonio	banche
Brambilla	Giovanna	trasporti
Brambilla	Giovanna	enti locali
Bruni	Angela	trasporti
Bruni	Angela	enti locali
Galati	Nunzio	trasporti
Galati	Nunzio	banche

OPERAZIONI DI UNION

L'operazione di UNION combina i risultati di due o più query.

In accordo con l'operazione di UNIONE così com'è intesa nell'algebra degli insiemi, UNION non restituisce i record duplicati.

Qualora si desideri ottenere tutte le righe risultanti dall'unione, si deve ricorrere al predicato **ALL**.

Nel definire query che utilizzano la UNION bisogna rispettare certi vincoli:

- Tutte le select devono richiedere lo stesso numero di campi; questi, tuttavia, non devono essere dello stesso tipo o dimensioni.
- Gli ALIAS (AS) vanno utilizzati solo nella prima SELECT; eventuali alias inseriti nelle select sottostanti, vengono ignorate.
- Un eventuale ORDER BY va inserita dopo l'ultima SELECT e si deve riferire ai nomi di campo specificati nella prima select (o, preferibilmente, alla loro posizione nella lista).

Esempi:

```
SELECT cognome, stipendio+supmin AS [incentivo / totale in busta  
pag]
```

```
FROM dipendenti, livelli
```

```
WHERE dipendenti.codliv = livelli.codliv
```

```
UNION
```

```
SELECT cognome, supmin
```

```
FROM dipendenti, livelli
```

```
WHERE dipendenti.codliv = livelli.codliv
```

```
ORDER BY 1
```

dà come risultato, per ogni dipendente, una riga che riporta solo il superminimo ed una riga con lo stipendio totale:

cognome	Incentivo / totale in busta paga
Bianchi	L. 800.000
Bianchi	L. 35.800.000
Brambilla	L. 650.000
Brambilla	L. 30.650.000
Bruni	L. 0
Bruni	L. 30.000.000
Galati	L. 1.200.000
Galati	L. 51.200.000
Rossi	L. 1.000.000
Rossi	L. 51.000.000
Verdi	L. 1.500.000
Verdi	L. 51.500.000

Ora, se si modifica la query precedente, per visualizzare lo stipendio base e lo stipendio comprensivo del superminimo, dei dipendenti in servizio, si ha:

```
SELECT cognome, (stipendio+supmin) / 1000 AS  
           [incentivo / totale in busta paga (x 1000)]  
FROM    dipendenti, livelli  
WHERE   dipendenti.codliv = livelli.codliv  
           AND  
           data_cess IS null  
UNION  
SELECT cognome, stipendio / 1000  
FROM    dipendenti, livelli  
WHERE   dipendenti.codliv = livelli.codliv  
           AND  
           data_cess IS null  
ORDER BY cognome
```

e come risultato:

cognome	incentivo / totale in busta paga (x 1000)
Bianchi	35000
Bianchi	35800
Brambilla	30000
Brambilla	30650
Bruni	30000

Galati	50000
Galati	51200
Rossi	50000
Rossi	51000

Da cui si può notare che Bruni compare una volta sola; di fatto tale dipendente, non avendo superminimo, dava origine a due record identici, dei quali, il sistema ha provveduto ad eliminarne uno.

In questo caso particolare, può sembrare che ci sia un'incompletezza nell'informazione, nel senso che, all'utente finale, può non esser chiaro il fatto che Bruni percepisca o meno il superminimo. Far apparire due righe con lo stesso valore di stipendio, infatti, non genera alcun dubbio di interpretazione. Ciò si ottiene inserendo **ALL** dopo UNION, nella query precedente.

Si noti, inoltre, che il criterio su "data_cess" è stato inserito in entrambe le select; se viene omessa dalla seconda select, per es., apparirebbe, per il dipendente Verdi, soltanto l'informazione relativa allo stipendio base.

LE SUBQUERY

Le SUBQUERY sono delle SELECT richiamate all'interno di altre SELECT.

L'SQL, difatti, permette ad una query di considerare un valore od un set di valori ricavati da un'altra query.

Ciò, di solito, si fa quando nella WHERE una condizione è data dal confronto di un attributo con valori non noti a priori, ma che si ricavano tramite una SELECT ad un'altra tabella.

La SUBQUERY non può essere utilizzata comparandola con l'opzione BETWEEN.

Ovviamente, il risultato della SUBQUERY deve essere dello stesso tipo del dato di comparazione.

Predicati utilizzabili con le SUBQUERY

Il campo di comparazione può essere connesso con i risultati prodotti da una subquery attraverso le seguenti keyword:

- **ANY**
- **ALL**
- **IN**
- **EXISTS**

Per cui il forma generale di una SUBQUERY è il seguente:

SELECT
FROM
WHERE espressione operatore [ALL], [ANY]

(SELECT)

dove *operatore* è un operatore di comparazione, e ciò che è descritto tra parentesi quadre, è opzionale,

oppure

```
SELECT .....  
FROM .....  
WHERE espressione [NOT] IN / EXISTS  
      (SELECT .....)
```

SUBQUERY CON SINGOLO VALORE

SELECT

.....

WHERE espressione > (subquery)

significa che la SUBQUERY deve restituire un singolo valore (altrimenti vi è una condizione di errore).

Il predicato è vero se il risultato della SUBQUERY è minore dell'espressione.

In questo caso risulta più conveniente utilizzare una JOIN.

OSS Per ogni interrogazione che utilizza una subquery esiste sempre

una interrogazione equivalente realizzata con una JOIN

SUBQUERY CON ALL O ANY

SELECT

.....

WHERE espressione > ALL (subquery)

significa che la SUBQUERY può dare un set di zero, uno o più valori. Il predicato è vero se l'espressione è maggiore di ogni valore tra quelli dati dalla SUBQUERY o se non è dato nessun valore.

SELECT

.....

WHERE espressione > ANY (subquery)

significa che la SUBQUERY può dare un set di zero, uno o più valori. Il predicato è vero se l'espressione è maggiore di almeno uno tra i valori dati dalla SUBQUERY. Se non è dato nessun valore il predicato è falso.

Esempi con ALL ed ANY:

```
SELECT DISTINCT codliv  
FROM dipendenti  
WHERE codliv = ALL (SELECT codliv  
FROM livelli)
```

Non da alcun risultato, in quanto non tutti i livelli di inquadramento risultano presenti nella tabella DIPENDENTI.

```
SELECT DISTINCT codliv  
FROM dipendenti  
WHERE codliv = ANY (SELECT cod_liv  
FROM livelli)
```

Da l'elenco dei livelli di inquadramento presenti nella tabella DIPENDENTI:

codliv
1
3
4

SUBQUERY CON IN O EXISTS

Sul set di valori prodotti da una SUBQUERY si possono applicare gli operatori IN ed EXISTS (con le loro negazioni).

SELECT

.....

WHERE espressione IN (subquery)

Il set di valori dato dalla SUBQUERY viene utilizzato dalla query principale attraverso l'opzione IN.

L'operatore IN è equivalente a '= ANY'.

SELECT

.....

WHERE EXISTS (subquery)

In questo caso si testa l'esistenza di una riga che soddisfi la condizione della SUBQUERY.

```
SELECT codliv
FROM livelli
WHERE codliv IN (SELECT codliv FROM dipendenti)
```

e

```
SELECT codliv
FROM livelli T1
WHERE EXISTS (SELECT *
              FROM dipendenti T2
              WHERE T1.codliv = T2.codliv)
```

Danno come risultato i livelli di inquadramento con almeno un dipendente, ossia 1, 3 e 4.

Nella seconda query, il campo *codliv*, costituisce il legame logico tra la query principale e la subquery, ed è chiamato “**CORRELATION VARIABLE**”.

QUERY PARAMETRICHE

E' possibile realizzare delle query, nelle quali, alcuni dati necessari alla sua esecuzione vengono immessi dall'utente (fonte esterna).

Questo rende le query generalizzate.

Per esempio, se si interroga spesso il DB per ottenere le generalità di un dipendente (cognome, nome, sesso, data di nascita) è ripetitivo (=> controproducente) riscrivere la query

```
SELECT cognome, nome, sesso, data_nasc  
FROM dipendenti  
WHERE matricola = 3
```

cambiando il valore della matricola ad ogni occorrenza.

Il problema si risolve scrivendo la query parametrica:

```
PARAMETERS [inserisci matricola] Byte;  
SELECT dipendenti.cognome, dipendenti.nome  
FROM dipendenti  
WHERE [inserisci matricola]=matricola
```

La sua esecuzione, genera una finestra di dialogo con il messaggio di richiesta “inserisci matricola” (quello specificato tra parentesi quadre); l'utente immette un valore (compatibile con il tipo specificato sulla riga **PARAMETER**, in questo Byte), quindi la selezione ha luogo.

Se l'utente inserisce il valore 3, la presente query diviene del tutto equivalente alla precedente, con il vantaggio che questa stessa query potrà essere richiamata in futuro con matricole differenti.

OPERAZIONI DI AGGIORNAMENTO DEL DB

Un Data Base è un sistema informatico per la gestione di dati; i dati con la loro organizzazione, rispecchiano una certa realtà, nel nostro caso un'azienda. Ora, essendo la realtà mutevole nel tempo, deve esserlo necessariamente il relativo DB (altrimenti risulterebbe "disallineato").

Per esempio, lo stato di un'azienda evolve perché ci sono nuove assunzioni, perché aumentano gli stipendi, perché cambiano i progetti, ...; lo stato di una biblioteca evolve perché si acquistano nuovi libri, perché dei libri vengono presi in prestito, poi riportati, ... Per una gestione completa di un DB, dunque, non basta soltanto avere a disposizione strumenti per la lettura dei dati registrati, ma è necessario poter modificare, cancellare, inserire nuove informazioni al DB.

Modifica di informazioni: comando UPDATE

L'UPDATE è l'operazione di aggiornamento del DB Relazionale. Nella forma più generale, si presenta:

```
UPDATE nome-tabella  
    SET attributo 1 = valore 1,  
        attributo 2 = valore 2,  
        ..... = .....,  
        attributo n = valore n,  
WHERE lista condizioni
```

Nella SET si elencano gli attributi da aggiornare con i relativi nuovi valori.

Il nuovo valore dell'attributo può essere una costante, una variabile, il valore NULL (se l'attributo è di tipo NULL) o un'espressione aritmetica.

Vengono aggiornate tutte le righe che soddisfano le condizioni.

Se la clausola WHERE è omessa vengono aggiornate tutte le righe della tabella.

Nella WHERE possono essere utilizzate le subqueries.

Esempi:

Con la seguente query di aggiornamento

```
UPDATE dipendenti  
  SET supmin = 350000  
  WHERE supmin = 0
```

Viene assegnato un superminimo di £ 350000 a tutti i dipendenti che ne erano sprovvisti.

Supponiamo, per assurdo, che il dipendente Bianchi Antonio abbia cambiato sesso (ipotesi, oggi, non così remota) e che l'azienda abbia deciso di raddoppiarli il superminimo; l'operazione da eseguire per memorizzare questo nuovo stato del dipendente la cui matricola è 3, è la seguente:

```
UPDATE dipendenti  
  SET sesso = "F", supmin = supmin*2  
  WHERE matricola=3
```

Quando viene eseguita una query di aggiornamento, Access avverte che l'operazione avrà effetto sul DB, informa sul numero di record che riguarderà l'aggiornamento, e chiede conferma!

A conferma dell'avvenuta operazione di UPDATE, il risultato della SELECT:

matricola	cognome	nome	data_nasc	sex	data_ass	data_cess	Codliv	codbranch	supmin
3	Bianchi	Antonio	18/08/76	F	12/01/96		3	2	1600000

Eliminare informazione: comando DELETE

La DELETE è l'operazione di cancellazione del DB Relazionale.
Il formato della DELETE è:

**DELETE FROM nome-tabella
WHERE lista condizioni**

Vengono cancellate tutte le righe che soddisfano le condizioni.

Se la clausola WHERE è omessa vengono cancellate tutte le righe della tabella.

Nella WHERE possono essere utilizzate le subqueries.

Per eliminare tutte le informazioni, dalla tabella dipendenti, di coloro che non sono più in forza all'azienda, basta digitare:

**DELETE FROM dipendenti
WHERE data_cess IS null**

Aggiunta di informazione: comando INSERT

Il formato della INSERT è:

**INSERT INTO nome-tabella
[(colonna1, colonna2,, colonna n)]
VALUES (valore1, valore2,, valore n)**

I valori da inserire vanno specificati nell'ordine previsto al momento della definizione della tabella, oppure nell'ordine previsto dalla lista

delle colonne della INTO, se specificata (si ha in questo caso l'indipendenza fisica dei dati).

La seguente query serve per aggiungere al DB un neoassunto:

```
INSERT INTO dipendenti  
    (matricola, cognome, nome, data_nasc, sesso, data_ass,  
     data_cess, codliv, codbranch, supmin)  
VALUES ( 7, 'Belli', 'Daria', '5-12-75', 'F',  
          '20-7-98', NULL, 2, 1, 170000 )
```

Considerazioni

Occorre sapere che eventuali aggiornamenti sulle tabelle, in Access, possono essere fatte direttamente intervenendo sulle stesse con le stesse modalità operative adoperate durante il data-entry, quindi come visto, in modo estremamente semplice:

Per inserire un nuovo record basta posizionarsi sull'ultima riga contrassegnata da un asterisco, ed immettere i dati;

Per cancellare una o più righe basta selezionarle e scegliere "Taglia" dalla *tool bar*;

Per modificare un valore, basta posizionarsi sulla colonna e sovrascrivere quello esistente.

Tuttavia, è utile conoscere la possibilità di effettuare tali operazioni mediante istruzioni SQL, poiché, questo è l'unico modo per fare aggiornamenti sui dati da un linguaggio di programmazione.

Esercizio:

Dalle informazioni registrate nel DB_Azienda, ricavare le figure professionali previste in ogni progetto.

Anche se apparentemente semplice tale interrogazione coinvolge quasi tutte le tabelle (comprese quelle di relazione):

```

SELECT A.codprj, D.des AS [progetto], A.codliv, E.des AS
    [profilo],cognome,nome
FROM liv_prj A, dip_prj B, dipendenti C, progetti D, livelli E
WHERE A.codprj=B.codprj      AND
       B.matricola=C.matricola AND
       A.codliv=C.codliv     AND
       A.codprj=D.codprj     AND
       A.codliv=E.codliv

UNION
SELECT A.codprj, D.des, A.codliv, E.des, "*****", "*****"
FROM liv_prj A, progetti D, livelli E
WHERE A.codprj=D.codprj     AND
       A.codliv=E.codliv
AND NOT EXISTS
    ( SELECT codprj, codliv
      FROM dip_prj B, dipendenti C
        WHERE B.matricola=C.matricola AND
              A.codprj= B.codprj     AND
              A.codliv=C.codliv )

ORDER BY 1, 2, 3
    
```

codprj	progetto	codliv	Profilo	cognome	nome
1	prenotazioni	3	programmatore Sr	Bianchi	Antonio
2	turni del personale	1	progettista	Galati	Nunzio
2	turni del personale	1	progettista	Rossi	Mario
3	anagrafe comunale	1	progettista	Galati	Nunzio
3	anagrafe comunale	1	progettista	Rossi	Mario
5	conti correnti	1	progettista	Galati	Nunzio
5	conti correnti	4	programmatore Jr	Brambilla	Giovanna
6	gestione mutui	1	progettista	*****	*****
6	gestione mutui	4	programmatore Jr	Bruni	Angela
7	gestione catasto	1	progettista	*****	*****
7	gestione catasto	3	programmatore Sr	Bianchi	Antonio

E' possibile esportare in Excel una tabella derivante dall'esecuzione di una query per ottenere facilmente la composizione di un grafico, come quello in figura:

