APPUNTI
PER GLI STUDENTI DELLA CLASSE QUINTA

CHE DEVONO SOSTENERE

L'ESAME DI MATURITA'

Introduzione

Il Piano di lavoro per queste schede è suddiviso in due macro sezioni. La prima concettuale e di base: di che parliamo? Delle Basi di dati. La seconda operativa e sostanziale: come si lavora con le basi di dati? Con un linguaggio appositamente studiato per la loro gestione: l'SQL.

I due canali, fatte salve le necessarie integrazioni, implicano, nello schema di questi appunti, uno studio parallelo e coordinato volto alla acquisizione di uno schema mentale corretto ed efficiente adatto ad affrontare con sicurezza e gratificante produttività la prova scritta di Informatica dell'esame di maturità.

Lo scopo di questi appunti non è quello di sostituire il libro di testo, che resta l'unico strumento valido di riferimento per una corretta preparazione, ma quello di sintetizzare ed organizzare in un contesto organico gli argomenti del programma focalizzandone lo studio dal punto di vista di un obiettivo preciso: sostenere l'esame da persone che effettivamente stanno maturando e affrontano i problemi senza sovrappiù di paure inutili, ma con la coscienza tranquilla di chi ha predisposto quanto è nelle sue possibilità per tenere testa agli accidenti che gli capitano.

Ne segue che, se si vuole provare a cominciare a sentirsi persone cui non è negato rispetto e dignità, senza fanciulleschi atteggiamenti recriminatori e senza elemosinare umilianti richieste di suggerimenti a compagni indaffarati e professori imbarazzati, tutti i punti indicati in ogni scheda, per ognuna delle quali si esaurisce ogni argomento sempre in una pagina sola, devono essere conosciuti al meglio. Significa che, poiché questa è una sintesi estrema del programma di quinta, anche di una virgola qui riportata deve essere riconosciuto e compreso il senso. Senso che và confrontato con una precedente e successiva rilettura degli stessi argomenti trattati nel libro di testo.

Ad ogni scheda possono esserne accluse altre, con lo stesso codice, contenenti approfondimenti pratici, test e proposte di esercitazioni che vanno tutte eseguite.

PIANO DI LAVORO

REALIZZAZIONE DI UN SISTEMA INFORMATIVO

PERCORSO DI STUDIO
delle schede

Ciclo di vita

1. PIANIFICAZIONE

RICOGNIZIONE

RACCOLTA DELLA DOCUMENTAZIONE

2. ANALISI

PROGETTO CONCETTUALE

3. PROGETTAZIONE

SCHEMA STATICO

SCHEMA DINAMICO

Traduzione

4. REALIZZAZIONE

SCHEMA LOGICO

PROGRAMMAZIONE DEI SERVIZI E DELLE OPERAZIONI

5. TESTING

DATABASE

CANALE CONCETTUALE (A)

Sintesi
Integrazioni
Quadri riassuntivi – Mappe esplicative

Scheda N° **1** A1 SIN 01
Sistemi informativi

Scheda N° **2** A1 INT 01
Concetto di sistema

Scheda N° **3** A2 PLN 01
Progetto concettuale

Scheda N° **4** A3 SIN 01
Schema statico

Scheda N° **5** A3 INT 01
Modello Entità-Relazione

Scheda N° **6** A3 SIN 02
Schema dinamico

Scheda N° **7** A4 SIN 01
Realizzazione di un sistema informativo

Scheda N° **8** A4 SIN 02
Schemi logici per database

Scheda N° **9** A4 SIN 03
Il modello relazionale

Scheda N° **10** A4 SIN 04
L'algebra relazionale

Scheda N° **11** A4 INT 01
Teoria degli insiemi

Scheda N° **12** A4 PLN 01
Percorso ottimale

Scheda N° **13** A4 PLN 02
Realizzazione di un sistema informativo

Scheda N° **14** A4 SIN 05
I DBMS

Scheda N° **15** A4 SIN 06
I DBMS distribuiti

CANALE OPERATIVO (B)

Sintesi
Quadri riassuntivi – Mappe esplicative

Scheda N° **16** B4 SIN 01
SQL – Linguaggio per RDBMS

Scheda N° **17** B4 SIN 02
SQL – DDL – Defizione dei dati

Scheda N° **18** B4 SIN 03
SQL – DML – Le interrogazioni

Scheda N° **19** B4 SIN 04
SQL – DCL – Gestione della sicurezza

Scheda N° **20** B4 PLN 01
SQL – Tavola riassuntiva

Scheda N° **21** C1 PLN 01
Come impostare la prova scritta dell'esame di maturità

Un sistema informativo è l'oggetto per la realizzazione del quale, in prospettiva, avete scelto di interessarvi di Informatica e vi siete iscritti a questo tipo di scuola superiore. E a dare prova di saperlo fare, di essere in grado di sapervi inserire, con cognizione di causa, in un gruppo di lavoro a ciò dedicato, sarete chiamati, con ogni probabilità, al termine dei vostri studi con il compito scritto dell'esame di maturità. Sarà bene, quindi che fissiate con estrema precisione alcuni concetti fondamentali riguardo ad essi.

Naturalmente un Sistema Informativo (S.I.) è necessario e presente in qualsiasi organizzazione e prescinde dall'informatica. Ma è difficile, per come si sono messe oggi le cose, pensare ad un S.I. efficiente e all'altezza dei compiti che gli si richiedono che non utilizzi il supporto dell'elettronica e della tecnologia informatica. Né, d'altra parte, ciò sarebbe auspicabile dal punto di vista dei migliori esiti della vostra futura ricerca di un lavoro. Ecco perché di questo ci occupiamo e per questo motivo tutto il nostro studio è incentrato intorno all'analisi dei ***Sistemi Informativi aziendali automatizzati.*** *E per questo, infine, la prova d'esame non potrà prescindere da tale argomento che dal punto di vista informatico si sostanzia nell'automazione delle procedure di* raccolta, archiviazione, recupero ed elaborazione dei dati *(**Database**) per la fornitura di servizi (nella fattispecie: informazioni) agli utenti.*

Un **Sistema Informativo** di un'azienda è costituito dall'insieme delle risorse e delle procedure che, coordinate fra di loro, forniscono servizi tramite la diffusione di informazioni.

Si tratta di un **Sistema** in quanto costituito di elementi (dati, persone, procedure, tecnologie) che interagiscono fra di loro in vista di un obiettivo comune. Ed è **Informativo** perché il suo obiettivo primario è di produrre informazioni. Attraverso queste ultime sarà possibile offrire i **servizi** necessari agli utenti che a vario titolo, interni (*dipendenti, impiegati, dirigenti*) o esterni (*clienti, fornitori*), interagiscono con l'azienda.

La fornitura di servizi

può essere effettuata ogni qual volta siano richiesti, in *tempo reale* (**real-time**, come si dice), vedi per esempio la prenotazione di posti sui treni, oppure in periodi prefissati e in questo caso si parla di modalità *a lotti* (**batch**), come per l'elaborazione delle paghe mensili ai dipendenti.

Concetto di sistema

Un sistema in sintesi è una struttura, un apparato (che può, a sua volta, essere costituito da sotto-strutture e sotto-apparati, e così via), che riceve "qualcosa" dall'esterno la tratta, la elabora secondo le sue specifiche modalità e la restituisce di nuovo all'esterno come "qualcos'altra". Nel nostro caso Il S.I. riceve **dati grezzi**, numerosi, ridondanti e non organizzati e li restituisce come **informazioni** ovvero *conoscenza* che può essere utilizzata per i suoi scopi dall'utente che fruisce del servizio.

Il sotto-sistema informatico

In questo schema il sotto-sistema informatico, sistema **EDP** (*Electronic Data Processing*) interviene in tutte le fasi automatizzando le procedure che sarebbero eseguite in modo manuale, sostituisce gli archivi cartacei con archivi elettronici, agevola la comunicazione e la condivisione di risorse. Ed interviene, secondo i suoi due aspetti peculiari, sia dal punto di vista ***Hardware*** (naturalmente i computer, eventualmente collegati in reti locali o geografiche, *Server*, *workstation*, *microcomputer)*, sia dal punto di vista *software* con programmi adatti a controllare i componenti hardware e i dati stessi.

CICLO DI VITA DI UN SISTEMA INFORMATIVO

. La costruzione di un S.I. segue le metodologie proprie dell'**ingegneria del software** ed è quindi costituita dalle fasi di *Pianificazione, Analisi, Progettazione, Realizzazione, Testing, Manutenzione* che, come per il software, costituisce il **ciclo di vita** del sistema stesso. In sostanza si tratta:

- per la **pianificazione** di verificare le condizioni in cui il sistema verrà ad essere installato (le persone, gli addetti le risorse necessarie, ecc.);
- nella fase di **analisi** di descrivere per filo e per segno il S.I. da realizzare indicando le specifiche dei dati che saranno trattati e le operazioni previste su di essi, le specifiche tecnologiche sulla strumentazione hardware, le specifiche ambientali sul contesto in cui il sistema è inserito e, infine le specifiche per il testing;
- di mettere a punto, nella fase di **progettazione** un modello astratto dei dati secondo uno *schema statico* costituente la base informativa ed uno *schema dinamico* in grado di descrivere le operazioni necessarie che dalla base informativa fa scaturire le informazioni elaborate utilizzabili dagli utenti del sistema;
- nella fase di **realizzazione** si attuano nella pratica gli schemi statici progettati traducendoli in un database (insiemi di archivi elettronici correlati fra di loro) e gli schemi dinamici inserendo nel database stesso le procedure adatte per eseguire le operazioni previste sui dati;
- Il **testing** prevede il collaudo del S.I. prima della consegna al cliente;
- e infine con la **manutenzione**, dopo la consegna sono indicati gli interventi correttivi o adattativi sulla base dei dati o sulle operazioni su di essi o anche interventi perfettivi o preventivi.

Costruire un Sistema Informativo *non è cosa da poco né dal punto di vista delle risorse umane necessarie né da quello delle risorse economiche. Non è cosa da una persona sola. Occorre un* ***Gruppo di sviluppo*** *formato da figure professionali specifiche e specializzate (Oltre al Responsabile del progetto sono individuabili ruoli o figure professionali per* pianificatori-analisti, sistemisti, progettisti, programmatori, progettisti hardware, sistemisti ICT e di reti, installatori e manutentori di reti, *ecc.). Naturalmente occorre un* ***metodo*** *di lavoro che seppure non garantisce il risultato almeno permette di ripercorrere all'indietro i passi del lavoro svolto e agevola la possibilità di individuare gli errori commessi e porvi rimedio (o elimina i dubbi residui per decidere di ricominciare tutto da capo).*

Non sono più infrequenti aziende ICT che questo fanno per lavoro: producono S.I. e li manutengono presso le aziende loro clienti. Parallelamente sono sopraggiunte sul mercato informatico ambienti integrati di sviluppo chiamati ***sistemi per la gestione delle basi di dati*** *(DBMS Database Management System) con cui i realizzatori del sistema possono agevolmente trasformare uno schema statico in una base di dati e tradurre le operazioni previste sui dati in applicazioni software integrate per l'elaborazione e la raccolta dei risultati.*

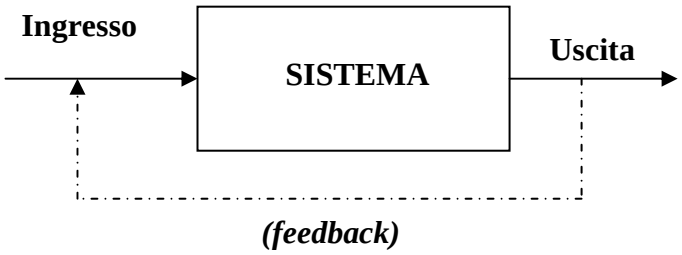
Non è affatto una cattiva idea puntare a studiare e specializzarsi in questo tipo di software applicativo e farne uno strumento per presentarsi in modo qualificato sul mercato del lavoro informatico.

La teoria dei sistemi è un'area di studi interdisciplinari che si occupa delle proprietà di un sistema nella sua interezza. Essa fu fondata negli anni 1950 da Ludwig von Bertalanffy, William Ross Ashby ed altri, basandola sui principi dell'ontologia, della filosofia della scienza, della fisica, della biologia e dell'ingegneria, trovando poi applicazioni in numerosi campi, tra cui geografia, sociologia, scienze politiche, teoria delle organizzazioni, management, psicoterapia ed economia. La Cibernetica è una disciplina strettamente correlata.

Generalità

Nato nell'ambito delle scienze biologiche per legare il comportamento di un organismo a quello dell'ambiente che lo circonda, il concetto di sistema si è rapidamente diffuso nell'approccio scientifico in generale dove certi strumenti interpretativi ad esso connessi possono ritenersi patrimonio consolidato.

Particolarmente efficace è la possibilità di ridurre, in sede di analisi, il funzionamento di fenomeni complessi all'interazione di sistemi più semplici e, viceversa, la possibilità di progettare sistemi in maniera strutturata componendo unità più semplici.



Scopo

Scopo della Teoria dei Sistemi (***TdS***) è introdurre ai principali metodi di studio dei sistemi dinamici orientati con particolare riferimento alla classe dei sistemi lineari e stazionari, a tempo continuo e a tempo discreto.

In Ingegneria la necessità di associare ai fenomeni una loro descrizione quantitativa ha poi dato luogo all'associazione sistema-modello, cuore della Teoria dei Sistemi: questa pertanto ha l'obiettivo di inquadrare in maniera unitaria le relazioni di causa-effetto e fornire degli strumenti di analisi matematica e sintesi.

Lo studio delle proprietà nel dominio del tempo e della frequenza fornisce elementi essenziali di interpretazione del comportamento di fenomeni e processi caratteristici dei diversi settori applicativi dell'Automatica e dell'Informatica.

Alcuni programmi di calcolo e simulazione attualmente disponibili costituiscono un formidabile ausilio all'utilizzo delle tecniche della Teoria dei Sistemi di cui hanno anche adottato il linguaggio grafico.

La teoria dei sistemi

Un sistema è una qualsiasi identità che è possibile analizzare e quindi scomporre. Ogni sistema ha delle caratteristiche che possono essere:

- Variabili / Condizionate
- Costanti

Possiamo rappresentare un sistema come una scatola con ingressi (u) ed uscite (y).

Lo stato del sistema è descritto da un insieme di variabili, dette appunto "di stato", solitamente indicate con la lettera x.

Gli ingressi agiscono sullo stato del sistema e ne modificano le caratteristiche: queste modifiche vengono registrate dalle variabili di stato.

I valori delle uscite del sistema, solitamente le uniche variabili misurabili (ingressi esclusi) dipendono dalle variabili di stato del sistema e dagli ingressi (in maniera più o meno diretta).

Per lo studio del sistema si analizza e si fissa il lasso di tempo [T] nel quale sarà studiato. In questo lasso di tempo (insieme ordinato di istanti) si considerano una serie di istanti particolari. Ordinato significa che prendendo due elementi qualsiasi possiamo stabilire con certezza quale dei due precede l'altro.

Gli elementi necessari per studiare un sistema sono:

T = { t0, t1, ... ti } // Insieme ordinato del tempo
I = { i0, i1, ... ii } // Insieme delle variabili di ingresso
U = { u0, u1, ... ui } // Insieme delle variabili di uscita
X = { x0, x1, ... xi } // Insieme delle variabili di stato
f = f(T, I, U, X) // Equazione di stato
g = g(T, I, U, X) // Equazione di uscita

L'equazione di stato f serve per calcolare lo stato interno del sistema in un determinato istante
X(ti) = f (X(t0), I, [t0, ti))

quindi si tiene conto dello stato iniziale e di tutte le entrate fino a quel momento. Grazie a questa funzione possiamo studiare l'evoluzione dello stato interno di un sistema.

L'equazione di uscita g serve a calcolare l'uscita U(ti) nell'istante ti
U(ti) = g (X(ti), in(ti))

Tiene quindi conto dello stato interno del sistema e degli ingressi dell'istante ti.

Il sistema quindi dipende da questa sestupla di dati:

S = (T, I, U, X, f, g)

All'interno del ***ciclo di vita*** di un Sistema Informativo (S.I.), cioè della sua costruzione, si possono distinguere, in realtà, **due fasi essenziali** che costituiscono il cosiddetto **Progetto Concettuale**. La prima mira a definire un modello dei dati da trattare; tale modello viene indicato come **Schema Statico**. La seconda, realizzativa, in cui, sulla base della prima, si realizza un modello per le operazioni da effettuare sui dati; tale modello è indicato come **schema dinamico**. Il progetto concettuale è ***indipendente*** dalla tecnologia e dal software con cui verrà effettivamente realizzato. Esso è ***modulare***, ossia è il più semplice possibile in quanto costituito da tanti singoli moduli rivolti a descrivere singole realtà e solo quelle. Le realtà complesse vanno scomposte e bisogna individuare quelle più ***elementari*** che le costituiscono. Inoltre il Progetto Concettuale è ***generale*** in quanto non mira a risolvere un singolo caso specifico ma tutti quelli ad esso assimilabili perché facenti parte di una stessa categoria. Deve quindi essere anche ***dinamico*** perché dovrà tenere conto di tutte le possibili variazioni dei parametri del sistema. Infine, essenziale, il Progetto concettuale dovrà essere ottimamente ***documentato*** in modo semplice ma rigoroso seguendo un opportuno linguaggio di progetto (**PDL: *Program Design Language***).

IL PROGETTO CONCETTUALE

Di un sistema informativo è composto da:

Uno ***SCHEMA STATICO***

Con il modello dei dati da trattare, la loro struttura e collegamenti reciproci.

Uno ***SCHEMA DINAMICO***

Con il modello delle operazioni da svolgere sui dati per ottenere i servizi del sistema operativo.

LO SCHEMA STATICO

In questa fase di messa a punto dello schema logico dei dati non và tenuto nessun conto dei mezzi con cui verrà realizzato in pratica. Qui occorre enucleare i fatti elementari e salienti che concorrono a descrivere nel suo complesso il S.I. da costruire. Operazione che certamente non è immediata ma sicuramente importantissima e in grado di condizionare tutto lo sviluppo futuro del Sistema. Essa richiede tutto l'acume che solo la pratica e gli errori pregressi del progettista possono affinare per massimizzare i benefici e l'efficienza e abbattere costi e malfunzionamenti. Questa fase di "enucleazione", per così dire, non è semplice perché qualunque sistema, per quanto elementare, è per sua natura, in quanto tale, inserito in un complesso sistema di sistemi in cui e con cui interagisce e da cui è condizionato condizionandolo a sua volta. Eppure occorre individuare una ideale linea di separazione del Sistema stesso dagli altri sistemi con cui ha o avrà a che fare e dall'ambiente che li circonda e in cui tutti sono inseriti. **Questa linea di separazione costituirà l'interfaccia** del S.I. con cui esso comunica con l'esterno fornendo i servizi per i quali è stato costruito e attraverso il quale dall'esterno esso riceve informazioni. Occorrerà capire bene con chi il Sistema interagirà effettivamente. Quali sono i suoi utenti, uomini o altri Sistemi e così via. Che tipo di dati dovrà trattare e come perverranno ad esso (tramite immissione diretta, da archivi elettronici, da Internet, ecc.). **In questa fase lo scopo del progetto concettuale è quello di pervenire, dai fatti elementari e dati grezzi, numerosi e non organizzati, ad un *modello logico* dei dati stessi secondo una visione ordinata e strutturata in grado di fornire i servizi richiesti necessari per gli specifici utenti a cui è diretto.**

Scomporre la realtà in fatti elementari.

Una volta definiti gli obiettivi del problema si deve ***raccogliere la documentazione*** a propria disposizione. A partire da questa, secondo una tecnica dal generale al particolare, dall'alto verso il basso (top-Down), ***scomporre problematiche complesse in fatti più elementari*** che li costituiscono che descrivono la realtà che stiamo modellando.

Individuare le ENTITA' e il loro TIPO.

Il passo successivo è di evidenziare, all'interno dei **fatti elementari**, gli elementi caratteristici specifici del sistema e che indichiamo come **tipi di entità**: ossia un insieme di elementi con proprietà (attributi) comuni. *In sostanza, per esempio, in un sistema "scuola" tipi di entità sono: "professori", "allievi", "ausiliari", ecc. Attributi dei tipi sono: "Nome", "Cognome", "indirizzo", "codice", ecc.* Mentre una **Entità** è un elemento concreto nel sistema singolarmente contraddistinto dagli specifici valori assunti da tutti i suoi attributi. *In concreto: quello specifico professore, proprio quell'allievo lì, esattamente l'ausiliario che si chiama "Giovanni" (il cui attributo "Nome" ha assunto il valore "Giovanni"), ecc.*

Distinzione fra concetto di "entità" e "tipo di entità".

Il concetto di "Entità" non va quindi confuso con quello di "Tipo di Entità". *Fra quest'ultimo ed il primo passa la stessa differenza che passa fra nome di variabile e valore che essa assume nell'elaborazione di un programma oppure che fra concetto di "classe" nella programmazione ad oggetti ed "oggetto" stesso che è, come si dice, "istanza" della classe che lo definisce (il vostro proprio cagnolino "macchietta" che avete a casa vostra è un "oggetto", istanza specifica della classe "cani-barboncini a pelo bianco").*

Individuare le RELAZIONI fra i tipi di entità.

All'interno di un sistema le Entità non sono tra loro isolate ma comunicano e interagiscono. In un Sistema Informatico esse interagiscono attraverso **Relazioni**. Ossia collegamenti logici che se sono tra due tipi di entità sono detti binari.

Tre Tipi di relazioni.

Veniamo al punto: che **tipi di relazioni** possiamo definire tra i vari tipi di entità? Essenzialmente **tre**:

1. ***Uno a Uno:*** ad una entità di un certo tipo corrisponde una, ed una sola, entità di un altro tipo *(ad un dato professore corrisponde un certo scomparto nell'armadietto dei professori e solo quello, salvo indebiti accaparramenti).*
2. ***Uno a Molti*** (o ***molti a Uno*** a secondo del lato da cui consideriamo la relazione): ad una entità di un certo tipo corrispondono più entità di un altro tipo *(ad un certo allievo corrispondono un sacco di professori con cui deve combattere tutti i giorni in cui decide di venire a scuola, - ma la relazione sussiste anche quando non ci viene; oppure un sacco di professori corrispondono, non solo quando lo vedono, proprio a quell'allievo là).*
3. ***Molti a molti:*** ad una entità di un certo tipo corrispondono più entità di un altro tipo e, viceversa, ad una entità di quest'ultimo tipo corrispondono più entità del primo tipo *(ad ogni professore corrispondono un sacco di allievi e ad ogni allievo corrispondono un sacco di professori, e già).*

Ridefinizione del concetto di "Fatto elementare".

Alla luce di questa considerazione possiamo anche meglio definire il concetto di **Fatto elementare** nella descrizione di un S.I.: esso è un **insieme di due o più tipi di Entità in relazione fra di loro.**

Ma come si crea una relazione?

Semplicissimo:

facendo riferimento ad uno stesso valore assunto da un attributo comune delle entità dei due tipi collegati.

L'attributo "codice professore" comune nel tipo "Professore" e tipo "Allievo" assume uno specifico valore nel primo tipo e viene ripetuto, uguale, per ogni singola entità "allievo" che ha quel professore. Nel tipo entità "Professore" questo attributo è unico in quanto serve ad identificare ogni singolo professore e sono quindi tutti diversi fra di loro (è detto ***chiave primaria***). Nel tipo entità "Allievo" l'attributo "Codice professore", invece, può assumere più volte lo stesso valore, tutte le volte che si trova in corrispondenza degli allievi di quel determinato professore (e in questo caso è detto ***chiave esterna***).

VINCOLI per mantenere effettiva una relazione.

Una volta stabiliti i collegamenti, fare in modo che continuino a sussistere a seguito di variazioni di dati, cancellazioni e operazioni varie, è determinato dal rispetto dei **vincoli** definiti sui valori che possono assumere gli attributi. Quelli specifici definiti per gli attributi che entrano in gioco per effettuare i collegamenti relazionali sono detti **Regole di integrità referenziale** e impongono, essenzialmente, che in una relazione uno a molti non esistano entità del lato molti che non corrispondono a nessuna entità del lato uno *(non può esistere nessun allievo che sia senza neanche un professore, vi risulta?)*

MODELLO ENTITA' - RELAZIONE

Il modello Entità/Relazione (***Entity Relationship model E-R***) è un modello concettuale di dati, e fornisce una serie di strutture (**Costrutti**) per la descrizione della realtà. I costrutti sono utilizzati per definire degli schemi che descrivono l'organizzazione e la struttura delle occorrenze dei dati.

I costrutti principali sono:

Entità

rappresentano classi di oggetti (fatti, persone) che hanno proprietà comuni ed esistenza "autonoma" per una applicazione.
CITTA', DIPARTIMENTO, IMPIEGATO, ACQUISTO, VENDITA sono es. di entità di una applicazione aziendale. Una occorrenza di una entità è un oggetto della classe che l'entità rappresenta. na entità si rappresenta graficamente con una rettangolo, all'interno del quale si mette il nome del concetto che rappresenta.

Relazione

rappresentano legami logici, significativi per l'applicazione, tra due o più entità.
RESIDENZA è un esempio di relazione che può sussistere tra le entità CITTA' e IMPIEGATO, mentre ESAME può esserlo tra STUDENTE e CORSO.
Una occorrenza di una relazione è una n-upla (coppia nel caso di relazione binaria) costituita da occorrenze di entità, una per ciascuna delle entità coinvolte. Una relazione si rappresenta graficamente con una rombo, all'interno del quale si mette il nome del concetto che rappresenta.

Un esempio completo:

Attributi

descrivono proprietà elementari di entità o relazioni di interesse ai fini dell'applicazione.

Cardinalità delle relazioni:

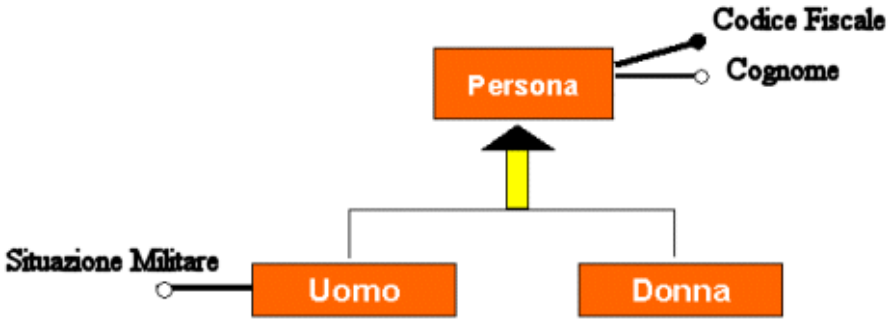
vengono specificate per ciascuna entità che partecipa a una relazione e dicono quante volte, in una relazione tra entità, un'occorrenza di una di queste entità può essere legata ad occorrenze delle altre entità coinvolte nella relazione. Indica il minimo e il massimo delle occorrenze. Nello schema si intende cha ad un impiegato possono essere assegnati da uno a cinque incarichi, e che ad un incarico possono essere assegnati da 0 a molti (N) impiegati.

Grazie alle cardinalità si possono definire tre tipi di relazioni:

1. **Uno a uno**: la cardinalità massima è pari a 1 per entrambe le entità.
   Ad es. Ordine (0,1) - Vendita - (1,1) Fattura: posso avere una vendita senza ordine (vado in negozio direttamente) ma non senza scontrino.
2. **Uno a molti**: la cardinalità massima è pari a 1 per una entità e cardinalità massima pari a N per l'altra.
   Ad es. Persona (0,1) - Impiego - (1,N) Azienda: una persona è impiegata in una sola azienda (o è disoccupata); l'azienda può impiegare una o molte persone.
3. **Molti a molti**: la cardinalità massima è pari a N (o più di uno) per entrambe le entità definisce una corrispondenza molti a molti (come nella figura precedente).

Anche per gli attributi è possibile definire la cardinalità. *es: Persona (0,1) - Numero Patente vuol dire che una persona può avere o non avere la patente*

Generalizzazioni



rappresentano legami logici tra una entità padre e una o più entità figlie, che sono comprese dal padre e che ne costituiscono un caso particolare. Il padre viene chiamato **generalizzazione** dei figli, che vengono chiamati, a loro volta, **specializzazioni** dell'entità padre.
Tra le entità coinvolte in una generalizzazione valgono le **proprietà**:

- ***Ogni occorrenza dell'entità figlia lo è anche dell'entità padre***
- ***Ogni proprietà dell'entità padre (attributi, identificatori, relazioni) è anche una proprietà delle entità figlie (ereditarietà).***

Per es. se l'entità Persona ha attributi Cognome ed Età, anche le entità uomo e Donna li possiedono. Inoltre l'identificatore di Persona è un identificatore valido anche per le entità Uomo e Donna.

LO SCHEMA DINAMICO

L'obiettivo della seconda parte del progetto concettuale è quello di fornire un **modello di tutte le operazioni** richieste nelle specifiche; fase detta **schema dinamico**, che unita alla messa a punto del modello dei dati, composto nel modello statico, completa l'intero progetto concettuale del sistema informativo.

*Nei fatti concreti le operazioni del S.I. sono realizzate in un programma mediante un opportuno piano di interrogazioni **(Query)** scritte con un linguaggio di programmazione all'interno di un ambiente di sviluppo per database (un DBMS).*

Il modello di un'operazione consiste nella descrizione di tutti gli elementi che la compongono, cioè:

1) una sua **descrizione** sintetica;
2) indicazione degli **utenti** a cui è rivolta;
3) il **tipo di operazione**;
4) gli **argomenti** o parametri di ingresso;
5) i **risultati** forniti (servizi o informazioni);
6) gli **errori** previsti e modalità di intercettazione.

A proposito dei tipi di operazioni indicati al punto 3) esse vengono generalmente classificate in **interne** (automatiche, gestite dal software che non coinvolgono direttamente gli utenti) e **esterne** direttamente invocate dagli utenti del sistema.

Queste ultime, a loro volta, vengono classificate in operazioni di:

a) *aggiornamento*;
b) *ordinamento*;
c) *ricerca*;
d) *elaborazione*;

il Cui significato, per studenti del quinto anno di una scuola di informatica che si accingono a sostenere un esame di maturità, dovrebbe essere di immediata comprensione.

Come si è accennato nelle schede precedenti, dopo la stesura delle specifiche per il progetto concettuale, che descrivono lo schema statico dei dati, ed un modello per le operazioni da effettuare su di esso, lo schema dinamico, la fase successiva è quella della **realizzazione**.

Mentre **prima** l'analisi è assolutamente ***sciolta*** dai riferimenti all'hardware e al software che interverranno nella realizzazione del Sistema Informativo, al fine di concentrarsi unicamente sul modello logico dei dati e quindi individuare soluzioni in grado di risolvere intere classi di problemi e non solo loro manifestazioni particolari; **adesso**, nella fase di realizzazione, la scelta del software e dell'hardware da impiegare, la situazione concreta in cui saranno utilizzati, il budget a disposizione, ecc. giocano un ruolo ***essenziale***.

Per la realizzazione del sistema statico occorrerà **valutare** il tipo di memoria di massa da utilizzare e il loro dimensionamento; per la realizzazione dello schema dinamico occorrerà decidere se ricorrere ad una gestione tradizionale degli archivi o ad un sistema integrato per le basi di dati.

Gli archivi tradizionali. Sono gestiti mediante programmi scritti ogni volta **daccapo**, con linguaggi procedurali specializzati per la loro gestione, tipo COBOL, CLIPPER, ecc.

Questa tecnica comporta notevoli **inconvenienti** tra cui la poco ***portabilità del software***, troppo legato alla gestione del particolare S.I. per cui è scritto, e la ***documentazione*** interna del sistema sempre ***insufficiente*** dato l'elevato numero di persone impiegate nei gruppi di lavoro.

Lo schema logico dei dati, suddivisi in più archivi, viene in realtà realizzato solo tramite le procedure operative appositamente scritte ogni volta. Aumentando, così, la ***complessità***, i ***tempi di sviluppo*** e l'onerosità della ***manutenzione*** del S.I. sia prima, nella fase di sviluppo, che dopo la consegna al committente.

LE BASI DI DATI. Per superare le difficoltà connesse alla programmazione tradizionale degli archivi negli anni '70 si sono imposti **sistemi integrati** per la loro gestione e il concetto di archivio si è evoluto in quello di Basi di dati (*Database*) dove non si parla più di singoli archivi collegati fra loro tramite il software scritto appositamente dai progettisti/programmatori ma di ***tabelle*** integrate in un database in grado sia di rispecchiare lo schema statico dei dati sia di gestire il modello delle operazioni previste su di essi. Tali sistemi integrati sono detti **DBMS** (*Data Base Management System*). Suddivisibili in sistemi **centralizzati** quando localizzati su un unico elaboratore e sistemi **distribuiti** quando gli archivi sono fisicamente raggiungibili in almeno due o più computer host collegati fra di loro in rete.

A loro volta i sistemi centralizzati possono essere o di tipo ***monoutente***, quando tutti i dati e i servizi sono richiedibili solo presso il computer centrale che li detiene e presso il quale bisogna fisicamente recarsi per ottenerli; o di tipo ***multiutente***, quando detti dati e servizi, comunque memorizzati su un unico sistema, sono ottenibili tramite una rete di calcolatori secondo una cosiddetta architettura *client-server* dove gli elaboratori *client* richiedono al computer centrale, il *server*, le informazioni e le elaborazioni desiderate.

PROPRIETA'DELLE BASI DI DATI Le basi di dati gestite da un DBMS sono ***memorizzate in modo permanente*** su supporti fisici adatti e ***organizzati secondo uno schema logico***; che risulta per l'utente, programmatore o operatore che sia, assolutamente ***indipendente*** dalla effettiva sottostante memorizzazione fisica. I dati stessi sono ***protetti dall'esterno e affidabili***, organizzati con la ***minima ridondanza*** e sono ***disponibili*** in modo controllato dal sistema per essere utilizzati da applicazioni diverse; anche le applicazioni, quindi, sono indipendenti dai dati diversamente che dalla gestione tradizionali degli archivi.

Le basi di dati hanno il compito di offrire un ***modello*** in grado di fornire una rappresentazione organizzata dei dati grezzi. In generale, lo **schema logico** costituisce la realizzazione, mediante un linguaggio di programmazione messo a disposizione dal gestore dei dati (il DBMS), dello schema statico del progetto concettuale. Con la terminologia "modello dei dati" si sottolinea il fatto importante che il programmatore di database è assolutamente svincolato, anche nella fase di realizzazione delle operazioni, da tutti i problemi relativi alla gestione degli archivi fisici posti sulle memorie di massa. Egli deve conoscere, appunto, al meglio possibile il modello dei dati implementato dal DBMS che usa. E utilizzare il/i linguaggio/i di programmazione che esso supporta per accedere e manipolare i dati secondo la visione strutturata e organizzata di essi che tale modello gli offre.

Fino ad oggi sono stati proposti ***vari schemi logici*** per database classificabili in base al ***tipo di struttura*** di dati impiegata per realizzarli: quello **gerarchico**, basato su strutture ad ***albero***; quello **reticolare**, fondato su ***grafi***; quello **relazionale**, che usa ***tabelle***; quello **object oriented** che si avvale di ***classi di oggetti***.

I primi ad essere sviluppati, negli anni '60, sono stati il modello gerarchico e quello reticolare ma hanno, via via, trovato sempre minore applicazione a causa della loro scarsa efficienza. A partire dagli anni '80 il paradigma della programmazione ad oggetti (OOP) ha trovato attuazione anche per la teoria dei database ma applicazioni commerciali diffuse ancora non sono affermate. ***Il modello consolidato dei dati su cui si basano tutti i DBMS in commercio o open source (****Access, SQL-Server, Oracle, MySQL, ecc.)* ***è quello relazionale, introdotto nei primi anni '70 da E. F. Codd. Ed esso è oggetto del nostro studio nelle prossime schede.***

MODELLO RETICOLARE

lo schema si fonda su un insieme di tipi di record in relazione fra loro tramite collegamenti (*link*).

Il modello è una **rete** formata da un ***grafo*** orientato in cui i singoli ***nodi*** rappresentano un tipo di record con tutti i suoi attributi e gli ***archi orientati*** rappresentano le relazioni fra i tipi.

Diversamente che nel modello relazionale è ammesso che due record dello stesso tipo abbiano gli stessi valori perché il sistema li riconosce a livello fisico mediante i diversi indirizzi dei puntatori.

Svantaggi: ricerche non sempre efficienti; possibile ridondanza e duplicazioni dei dati; spreco di spazio di memoria per conservare i *link* tra i tipi tramite i puntatori e Il programmatore deve tenere conto di tutto ciò.

MODELLO GERARCHICO

Come in quello reticolare lo schema è formato da un insieme di tipi di record i cui elementi sono record logici suddivisi in campi elementari.

Il modello è un **albero** formato dai ***nodi*** che costituiscono i tipi di record e gli ***archi*** che rappresentano relazioni uno a molti dai nodi padri a quelli figli.

La gerarchia è prevista dal modello per cui il programmatore, per piegarsi ad esso, è costretto a introdurre relazioni fittizie, magari anche quando queste non sussistono nello schema statico originale.

Svantaggi: lo schema del database è troppo dipendente dalla sua realizzazione a livello degli archivi fisici; le operazioni di ricerca sono scarsamente efficienti se non agiscono direttamente sulla struttura gerarchica del modello.

MODELLO OBJECT ORIENTED

Sono estesi alla teoria delle basi di dati i concetti OOP di:

- **classi di oggetti** per implementare i ***tipi di entità***;
- **istanza di classe** (***oggetto***) per rappresentare le singole ***entità***;
- **incapsulamento** per includere in una classe (tipo di entità) sia gli ***attributi*** che i ***metodi*** (le procedure e i vincoli per essi previste);
- **ereditarietà** per creare ***sottoclassi*** gerarchicamente figlie da tipi di entità padri con analoghi attributi e comportamenti.

Il meccanismo dell'ereditarietà agevola in particolar modo la possibilità di rappresentare tipi di entità (classi) complesse individuando, da esse, altri tipi più semplici.

Differenza importante tra questo modello e quello relazionale è che mentre in quest'ultimo una entità è distinta da un'altra dello stesso tipo mediante il valore dei suoi attributi (almeno per la chiave primaria) in quello Object Oriented lo è per il suo indirizzo all'interno del sistema.

Consiglio di rileggere attentamente questa scheda dopo aver bene assimilato quelle sul modello relazionale.

IL MODELLO RELAZIONALE

Il modello relazionale dei dati, introdotto in IBM da **Edgar F. "Ted" Codd,** è quello attualmente in uso nella stragrande maggioranza delle applicazioni. E la sua ottima conoscenza è sempre richiesta nel mercato informatico.
La struttura logica su cui si basa il modello è la **tabella**, ossia un insieme di dati, eventualmente di tipo diverso, ordinatamente organizzati in **righe** e **colonne**. Per ottenere lo schema logico del database, a partire dallo schema statico del progetto concettuale e con cui realizzare il Sistema informativo, sono utilizzabili ***due tecniche***, teoricamente indipendenti fra di loro, ma che è meglio pensare non in alternativa ma piuttosto come passaggi successivi di un **percorso ottimale** in grado di portare ad un sistema più efficiente ed efficacemente produttivo. Le due tecniche consistono, la prima, nella ***traduzione dello schema statico*** nella struttura tabellare del modello; la seconda nell'***applicazione*** su tali tabelle della ***teoria formale relazionale***.

TRADUZIONE DA SCHEMA STATICO A MODELLO RELAZIONALE

Non è difficile: occorre stabilire le seguenti ***corrispondenze*** tra schema statico del progetto concettuale e le strutture del modello relazionale:

SCHEMA STATICO		MODELLO RELAZIONALE
Tipo di Entità	--->	Tabella
Entità	--->	Righe della tabella
Attributi	--->	Colonne della tabella
Regole Attributi	--->	Vincoli sui valori delle colonne
Relazioni tra tipi	--->	Colonne in comune tra 2 tabelle

I **dati** contenuti nelle tabelle possono essere di tipo diverso per le varie colonne (*numeri, caratteri, booleani, stringhe, ecc.*) e la disponibilità di tali tipi può essere diversa a seconda del DBMS utilizzato.

Le relazioni tra le entità riscontrate nel modello entità-relazione dello schema statico tra due tipi di entità, che, come si sa, possono essere *uno a uno, uno a molti, molti a molti,* sono agevolmente tradotte, nel modello relazionale derivante, tramite l'utilizzo nelle due tabelle corrispondenti ai tipi in relazione, di ***una colonna in comune*** riportante lo stesso valore di attributo per le righe delle due tabelle che devono essere collegate fra di loro (*lo stesso* codice professore *nella tabella* professori *e nella tabella* allievi *per collegare ad un dato professore tutti i suoi allievi*). A tale proposito è importante notare che, per realizzare una relazione ***molti a molti*** è necessario, per evitare ridondanze e pernicioso spreco di memoria, introdurre una ***tabella ausiliaria*** che permetta un collegamento *uno a molti* sia da un lato che dall'altro. I*nfatti, se è vero che ad un professore corrispondono molti allievi è anche vero che ad un allievo corrispondono molti professori*: *tale tabella intermedia conterrà tutte le possibili coppie di* codice professore *e* codice allievo *in modo che, a partire da un dato professore nella tabella* professori, *si ritrovino in detta tabella intermedia le molte volte in cui esso è presente accoppiato con tutti i suoi allievi e, analogamente, a partire da un dato* codice allievo, *nella tabella* allievi, *si ritrovino, sempre in quella intermedia, tutte le volte che esso è presente accoppiato con i codici di tutti i suoi professori.*

Anche **le regole** riguardanti le limitazioni sui valori che possono assumere gli attributi dello schema statico devono essere riportate nel modello relazionale imponendo ***vincoli*** sui valori che si possono immettere nella colonna ad essi corrispondenti *(una colonna "sesso" ammetterà solo valore "M" o "F", una colonna "Età" sarà collegata ad una procedura che chiede chiarimenti per immissioni superiori a "130", e così via).*
Vincoli fondamentali del modello relazionale sono connessi alle colonne in comune delle tabelle in relazione: dal lato "uno" tale colonna prevede un unico possibile valore, non ripetibile, per individuare senza errori ogni singola riga e tale colonna assume la funzione di ***chiave primaria***. Dal lato "molti", invece, per permettere il collegamento con più righe, lo stesso valore deve, naturalmente, potersi ripetere e qui assume la funzione di ***chiave esterna***.

TEORIA RELAZIONALE DEI DATI

Secondo la teoria formale relazionale dei dati introdotta da Codd il modello finale ottimizzato dei dati, che rispetti pienamente tutte le proprietà contenute nella definizione di "DataBase", può essere ottenuto per affinamenti successivi tramite un processo, detto di normalizzazione, a partire da uno schema tabellare iniziale.

alcuni concetti di base: l'insieme di tutti i valori che può assumere un attributo si chiama **dominio**. L'insieme risultante dal **prodotto** fra due o più domini è dato da tutte le possibili *combinazioni* degli elementi dei singoli domini *(non si tratta di un "prodotto cartesiano" in quanto non ordinato, gli elementi dell'insieme si possono scambiare in quanto combinazioni, appunto, non disposizioni)*. Una **relazione** tra domini è un sottoinsieme del loro prodotto ossia un insieme in cui ogni elemento contiene *n* valori assunti dagli attributi e per questo motivo un elemento di questo insieme è detto ***ennupla*** (n-upla o, in inglese, ***tupla***). *Non vi spaventate troppo: in fondo stiamo parlando, ancora una volta, di tabelle! Secondo Codd una singola tabella è una relazione in quanto collega tra loro alcuni (n) attributi. E una riga di tabella è una ennupla. E le relazioni tra singole tabelle? Un'altra tabella! O, secondo la terminologia relazionale, un'altra relazione.* In una relazione (tabella) la **cardinalità** indica il numero delle ennuple presenti *(le righe)*; il **grado** rappresenta, invece, il numero dei domini *(gli attributi, il numero delle colonne)*. Per rappresentare una relazione si usa la seguente scrittura:

NomeRelazione/tabella (Dominio_1, Dominio_2,…, Dominio_N)
Es.: Allievi(CodiceAllievo, Nome, Cognome, telefono, ecc.)

In questa descrizione la sottolineatura indica che quel dominio è **chiave primaria** nella relazione. Nel caso in cui più di un dominio entra a far parte della chiave primaria si parla di **superchiave**. Le relazioni tra tabelle *(associazioni tra relazioni)* si realizzano mediante domini in comune tra le tabelle *(relazioni). Questo spostamento di terminologia non deve generare confusione. Esso deriva dall'esigenza di formalizzare con rigore i presupposti della teoria.*

Lo schema di una base di dati relazionale consiste nella descrizione di tutte le sue relazioni e delle associazioni fra esse mediante l'indicazione dei domini in comune. Si definisce, infine **istanza** di una base di dati l'insieme di tutti i dati raccolti e memorizzati nelle sue tabelle.

Per ottenere, dunque, uno schema ottimizzato lo si sottopone al processo di **normalizzazione** che consiste nel trasformare uno schema iniziale fino a verificare che tutte le relazioni soddisfino i presupposti delle cosiddette **3 forme normali**, riconducibili al seguente enunciato complesso:

I valori degli attributi di ogni colonna in tutte le tabelle
1. **contengono solo attributi semplici.**
2. **dipendono solo dalla chiave primaria**
3. **non sono calcolabili tramite altre colonne che non siano chiave.**

A partire dal secondo punto si introduce il concetto di **dipendenza funzionale** il quale consiste nell'indicazione che il valore di un certo attributo A determina il valore di un altro attributo B e si indica con la scrittura A → B. volendo significare che B dipende funzionalmente da A. Ora, poiché tutti i punti in successione presuppongono che il precedente sia soddisfatto, la richiesta di ottimizzazione della base dati è riassumibile nell'unico enunciato seguente, *detto anche 4a forma normale o di Boyce e Codd*: **Una tabella è in forma normale se per ogni dipendenza funzionale A→B definita su di essa, A contiene una chiave della tabella.**

TEORIA RELAZIONALE DEI DATI: l' algebra relazionale

I linguaggi per database relazionali permettono di realizzare i servizi di un sistema informativo, ossia di tradurre il modello dinamico del progetto concettuale per ottenere le operazioni previste sui dati. Ciò avviene mediante **interrogazioni** effettuate sulla base dei dati con combinazioni di una o più operazioni elementari che qui andremo a conoscere. Tali interrogazioni (*query*) permettono di estrarre dai dati nuove tabelle, perciò dette ***derivate***, che contengono le informazioni richieste dagli utenti. E' importante sottolineare che, nel rispetto della teoria relazionale, le nuove tabelle ottenute costituiscono, a tutti gli effetti, relazioni nella base dei dati e quindi, a loro volta, riutilizzabili per altre interrogazioni. Una interrogazione può quindi diventare molto complessa. E spesse volte lo è. Le principali operazioni possibili sulla base dati sono quelle proprie dell'algebra relazionale: selezione, proiezione, ridenominazione e join; e quelle derivate dalla teoria degli insiemi: unione, intersezione e differenza. Le formalizzeremo con un linguaggio di tipo dichiarativo simile all'SQL più propriamente affrontato in schede apposite.

LA SELEZIONE. **Fornisce come risultato una nuova tabella derivata formata dall'insieme di tutte le righe di quella di partenza che soddisfano una data condizione.** *Si indica con il simbolo σ – la lettera greca sigma.*

Es.: data la relazione

Allievi(codAllievo, nome, cognome, indirizzo, classe, sezione)
Contenente tutti gli allievi di una scuola, ad esempio la nostra

L'operazione
σ(classe = "quinta") AND (sezione = "D") (Allievi)

```
sql:    SELECT *
        FROM Allievi
        WHERE (classe = "quinta") AND (sezione = "D")
```

realizza una tabella derivata contenente tutti gli allievi della quinta D.

LA PROIEZIONE. **Fornisce un sottoinsieme delle sue colonne.** *Si indica con il simbolo π – la lettera greca pi.*

Es.: data la relazione

Allievi(codAllievo, nome, cognome, indirizzo, classe, sezione)
Contenente tutti gli allievi di una scuola, ad esempio la nostra

L'operazione
π nome, cognome (**σ**(classe = "quinta") AND (sezione = "D") (Allievi))

```
sql:    SELECT nome, cognome
        FROM Allievi
        WHERE (classe = "quinta") AND (sezione = "D")
```

realizza una derivata contenente solo i nominativi degli gli allievi della quinta D.

LA RIDENOMINAZIONE. **Modifica uno o più nomi delle sue colonne.** *Si indica con il simbolo ρ – la lettera greca ro.*

Es.: L'operazione
ρ indirizzo ← recapito (Allievi)

sql: operatore AS

cambia il nome della colonna "*indirizzo*" in "*recapito*" e, naturalmente, ne lascia inalterato il contenuto.

IL JOIN. **Si applica ad almeno due tabelle in relazione fra loro e fornisce come risultato un sottoinsieme del loro prodotto in cui è soddisfatta la condizione posta sulle colonne in comune.** *La condizione posta sulle colonne in comune si indica con il simbolo **θ** – la lettera greca Teta*

.

La condizione posta sulle colonne in comune è, tipicamente, una operazione di confronto (>, <, =, ecc.). Se tale condizione è di uguaglianza si parla di **equi-join** (Teta-join per definizione). Se, oltre a ciò, le colonne interessate (non vi sono ulteriori condizioni su altre colonne) si parla di **join naturale**.

Es.: date le relazioni

Classi(classe, sezione, codProfessore)
e
Professori(codProfessore, nome, cognome, materia)

L'operazione sql

```
SELECT Professori.cognome, Professore.materia
FROM Professori INNER JOIN Classi ON
        Professori.codProfessore = Classi.codProfessore
WHERE (Classi.classe = "quinta") AND (Classi.sezione = "D")
```

realizza una tabella derivata contenente i cognomi di tutti i professori della quinta D con la materia che insegnano.

Il Join naturale può essere esteso a più di due tabelle. Al limite a tutte le tabelle della base dati.

UNIONE, INTERSEZIONE, DIFFERENZA Una tabella può essere considerata come un insieme i cui elementi sono le ennuple (le righe). E' quindi possibile definire su di essa le operazioni tipiche della teoria degli insieme:

Una **unione** di due tabelle realizza una tabella derivata contenente le righe della prima o della seconda tabella.
Una **intersezione** di due tabelle realizza una tabella derivata che contiene le righe della prima e simultaneamente della seconda tabella.
Una **differenza** tra due tabelle realizza una tabella derivata che contiene le righe della prima che non compaiono nella seconda tabella.

DEFINIZIONI

L'**insieme** è un concetto primitivo. Intuitivamente possiamo dire che esso è una collezione di oggetti (**Elementi**) che soddisfano tutti una stessa proprietà (**P(x) = Proprietà caratteristica dell'insieme**).

- **Insieme vuoto** = insieme che non contiene elementi ($\varnothing$ **= { }**).
- **Insieme finito** = insieme dove è possibile contare tutti i suoi elementi e tale conteggio a termine.
- **Insieme infinito** = insieme non finito.

SIMBOLOGIA

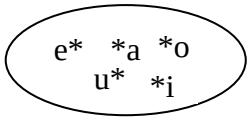
{ }	= insieme	$\in$	= appartiene a, elemento di
$\cap$	= intersezione	$\notin$	= non appartiene a, non elemento di
$\cup$	= unione	$\Rightarrow$	= implica che
$\subseteq$	= incluso in	$\bar{A}$	= non A
$\varnothing$	= insieme vuoto	$\forall$	= per ogni
Vel	= oppure	**et**	= e

Come si Rappresentano gli insieme

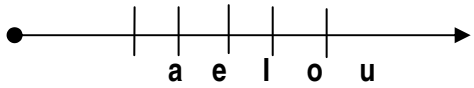
Per **elencazione** di tutti gli elementi che appartengono all'insieme: **A = { a,e,i,o,u}**

Per **proprietà caratteristica** enunciando la proprietà che gli elementi devono soddisfare per appartenere all'insieme: **A = { x| x è una vocale}**

Per **rappresentazione grafica** con un diagramma detto di ***Eulero-Venn***: **A =**



Mediante una **rappresentazione cartesiana**:

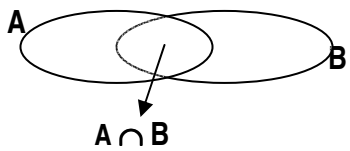


OPERAZIONI SUGLI INSIEMI

Intersezione

Dati 2 insiemi A e B, si dice intersezione **A $\cap$ B** l'insieme formato dagli elementi ***comuni di A e di B***:

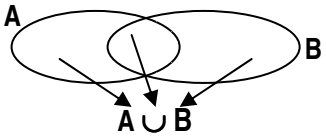
$A \cap B = \{x \mid x \in A \text{ et } x \in B\}$



Unione

Dati 2 insiemi A e B, si dice unione **A $\cup$ B** l'insieme formato dagli elementi ***di A o di B o di entrambi***:

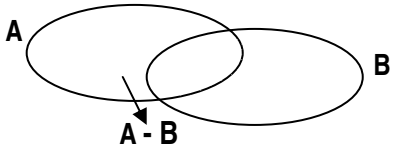
$A \cup B = \{x \mid x \in A \text{ vel } x \in B\}$



Differenza

Dati 2 insiemi A e B, si dice differenza **A - B** l'insieme formato dagli elementi ***di A non appartenenti a B***:

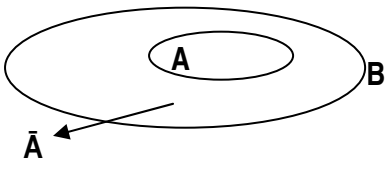
$A - B = \{x \mid x \in A \text{ e } x \notin B\}$



Complementazione

Dati 2 insiemi A e B, con **A $\subseteq$ B**, si dice complementazione di A rispetto a B l'insieme **$\bar{A}$** (*A soprassegnato*) formato dagli elementi ***di A non appartenenti a B***:

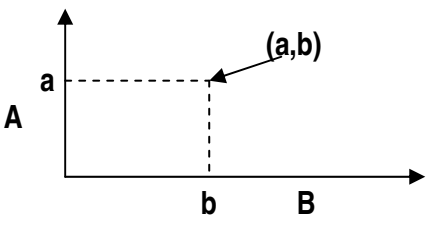
$\bar{A} = \{x \mid x \in A \text{ et } x \notin B\}$



Prodotto cartesiano

Dati 2 insiemi A e B, si dice prodotto cartesiano A x B l'insieme formato dalle **coppie ordinate** aventi per primo componente un elemento **di A** e per secondo componente un elemento **di *B***:

$A \times B = \{(a,b) \mid a \in A \text{ et } b \in B\}$



PERCORSO OTTIMALE

Per la realizzazione di uno schema logico relazionale a partire da un progetto concettuale.

PROGETTO CONCETTUALE

Composto da
SCHEMA STATICO
e
SCHEMA DINAMICO

TRADUZIONE DELLO SCHEMA STATICO DEL PROGETTO CONCETTUALE IN UNO SCHEMA LOGICO RELAZIONALE

SCHEMA STATICO		MODELLO RELAZIONALE
Tipo di Entità	--->	Tabella
Entità	--->	Righe della tabella
Attributi	--->	Colonne della tabella
Regole Attributi	--->	Vincoli sui valori delle colonne
Relazioni tra tipi	--->	Colonne in comune tra 2 tabelle

Ottenimento della versione di base dello schema logico relazionale dei dati

APPLICAZIONE DELLA TEORIA RELAZIONALE DEI DATI ALLA VERSIONE INIZIALE DELLO SCHEMA LOGICO

PROCESSO DI NORMALIZZAZIONE

Prima forma normale

Seconda forma normale

Terza forma normale

Verifica della congruità della base dati che deve rispettare la ***quarta forma normale***

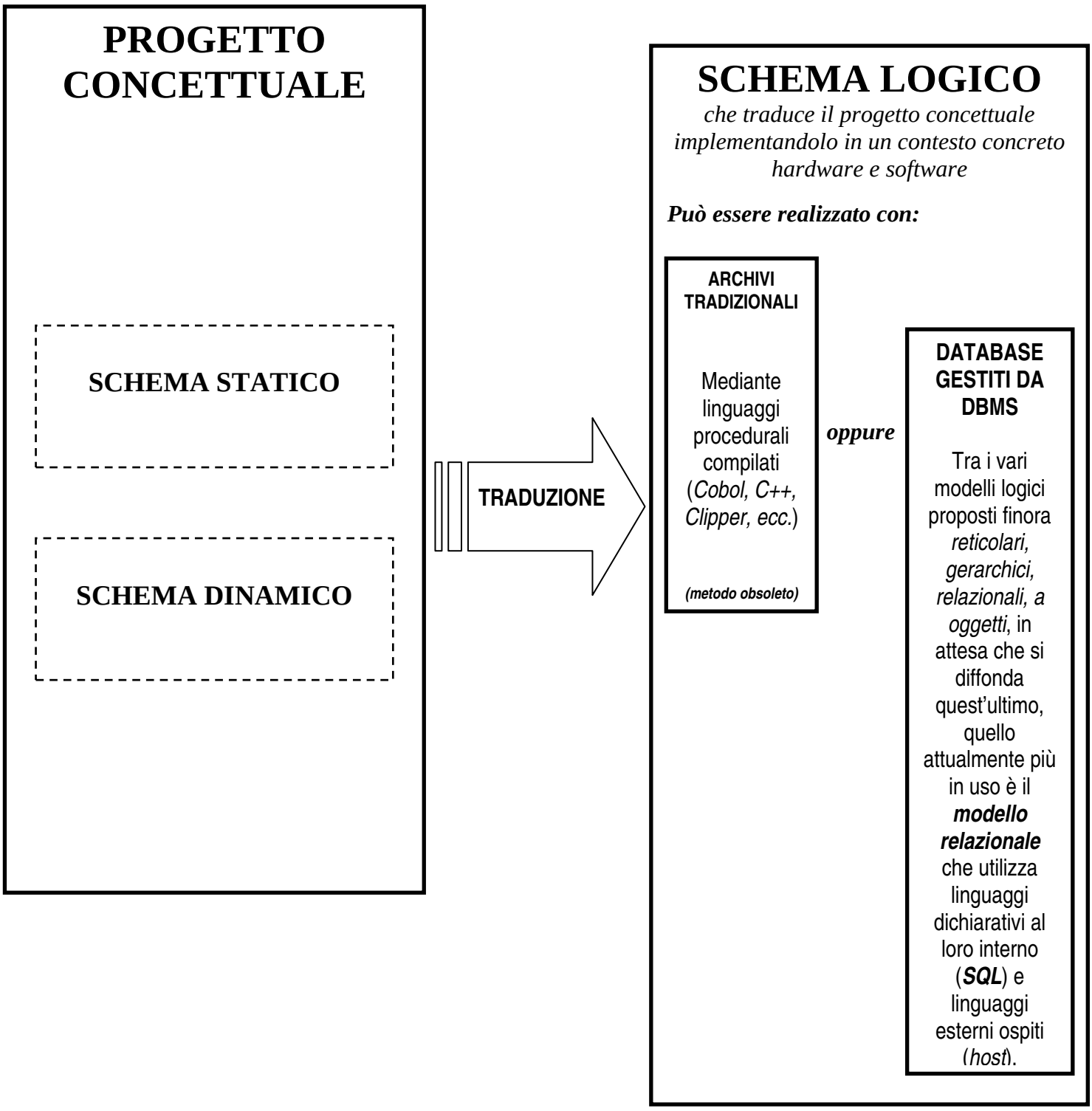
TRADUZIONE DELLO SCHEMA DINAMICO

Implementazione delle operazioni e dei servizi

REALIZZAZIONE DI UNO SCHEMA LOGICO RELAZIONALE

Organizzato, ordinato, efficiente, congruente, non ridondante, senza anomalie

Una volta messo a punto il progetto concettuale, composto nei suoi due aspetti dello schema statico e dello schema dinamico, occorre passare alla sua realizzazione pratica implementandolo in un contesto concreto ed ottenere uno **schema logico** in grado di offrire una visione ordinata e organizzata dei dati pronti per essere opportunamente manipolati e per eseguire su di essi le operazioni previste nello schema dinamico.



Per superare le difficoltà poste dalle metodologie tradizionali ad una corretta e più agevole gestione degli archivi di una base di dati, a cominciare dalla prima metà degli anni '70 sono comparsi i primi sistemi applicativi integrati specializzati per la loro creazione e manutenzione. I **Data Base Management System (DBMS)** sono un sistema software volto a risolvere tutti i problemi realizzativi di un progetto concettuale sia per la traduzione del sistema statico che per la realizzazione delle operazioni di quello dinamico. Col passare degli anni si sono evoluti fino a sviluppare interfacce grafiche che permettono una costruzione interattiva del sistema informativo. Ma si tratta, in realtà, di sistemi complessi il cui corretto utilizzo richiede comunque buone conoscenze generali della teoria e altrettante precise conoscenze specifiche dell'ambiente con cui si lavora. Con la padronanza di un DBMS si ha a disposizione un potente strumento per attuare tutte le **proprietà** di un database: ***la protezione, la non ridondanza, l'indipendenza fisica e logica e la gestione della sicurezza e dell'integrità dei dati***. I DBMS realizzano, tipicamente, un sistema informativo **centralizzato**, localizzato su un computer centrale (*server*), e **multiutente**, a cui, cioè, più utenti autorizzati possono accedere tramite computer collegati in rete (*client*). I DBMS relazionali, quelli attualmente più diffusi, sono anche detti RDBMS (*Relational DBMS*).

ARCHITETTURA *Client-server* I DBMS sono progettati per essere installati su un computer centrale (***server***) da cui forniscono i servizi richiesti ai computer collegati in rete ***(client***). Secondo una **architettura Client-Server** aperta, tramite *hub,* ai client nella LAN locale o anche, tramite *router*, a client remoti su internet. Questa architettura condiziona anche le modalità di scrittura del software a seconda se progettato per gestire i dati direttamente sul server *(lato server*) o per richiederne i servizi dal client *(lato client*). Si tratta, infatti, di un sistema complesso riguardo al quale possiamo individuare diversi **livelli** di percezione:

a) ***fisico-hardware*** dove và previsto il dimensionamento dei file e delle memorie di massa per la loro gestione e il backup;
b) ***del server*** il cui hardware deve comprendere CPU, RAM e dischi adatti e che deve essere configurato per la rete;
c) ***logico dei dati*** dove si realizzano lo schema statico e dinamico;
d) ***esterno delle applicazioni*** per l'accesso ai servizi del sistema.

ORGANIZZAZIONE FISICA DEI DATABASE **Ogni database si compone essenzialmente di un file principale (con tutte le tabelle, indici, stored procedure, ecc) e del registro delle transazioni *(transaction log*).**

Il file principale, che può essere affiancato da file secondari, memorizza, oltre alla base dati, anche **i database di sistema** con i dati relativi alla sicurezza *(user name, pasword, privilegi e autorizzazioni per ogni utente registrato)*; e alla configurazione dei dati associata ad ogni utente.

I dati, all'interno del file principale (o secondari), sono organizzati per **pagine,** insieme di righe di una o più tabelle, e **domini**, un numero intero di pagine coincidente con il blocco fisico di byte trasferito in una singola operazione di lettura/scrittura tra RAM e memoria esterna. Per ottimizzare questa operazione i DBMS impostano la grandezza di un dominio uguale a quella di un settore (*cluster*) del hard disk. Ed ogni operazione di lettura/scrittura sposta un intero dominio alla volta. Per velocizzare le operazioni sui dati (aggiornamento, ricerca, ecc.) esse vengono effettuate in RAM, in una zona a ciò espressamente dedicata dal DBMS, denominata ***buffer cache***.

Il registro delle transazioni, invece, ha il compito di memorizzare tutte le operazioni eseguite sui dati e il loro esito. Solo se un blocco di operazioni, che può essere anche molto complesso e comportare svariate modifiche in più punti del database, viene effettivamente completato, allora viene marcato come eseguito inserendo nel registro un ***checkpoint***. Periodicamente, o in momenti di bassa intensità di lavorio, il DBMS scorre il registro delle transazioni e trasferisce effettivamente sul file fisico le operazioni memorizzate nel registro in corrispondenza dei *checkpoint* che le danno per eseguite.

L'accesso fisico ai dati contenuti nelle tabelle può avvenire o attraverso una loro **scansione** (*table scan*) dalla prima all'ultima riga, con evidente dispendio di tempo e risorse, o percorrendo un **albero binario** (***B-Tree***) di nodi ordinati in un file a parte, detto **indice**, contenenti solo le chiavi di ordinamento e un puntatore alla pagina in cui si trova la riga da recuperare e solo questa pagina viene caricata ed effettivamente scorsa in RAM per completare l'accesso.

LINGUAGGI PER DATABASE Per realizzare il sistema informativo i DBMS, oltre ad evolute interfacce grafiche interattive, rendono disponibili:

1. Un **linguaggio interno** con sottoinsiemi di istruzioni specifiche per
   a) la definizione dello schema dei dati: ***Data Definition Language*** (**DDL**);
   b) la realizzazione delle operazioni e dei servizi: ***Data Manipulation Language*** (**DML**):
   c) l'impostazione dei criteri di sicurezza: ***Data Control Language*** (**DCL**);
2. Uno o più **linguaggi esterni** (*host*) per la codifica di algoritmi complessi.

I linguaggi interni possono, a loro volta, essere classificati come **procedurali**, quando il programmatore deve definire "***come"*** e in quale ordine devono essere eseguite le operazioni sui dati per ottenere i servizi e le informazioni richieste; oppure come **dichiarativi**, quando il programmatore deve solo dichiarare ***"quali"*** sono i risultati che vuole ottenere.

Il linguaggio interno più comune, nei moderni RDBMS, è l'**SQL** (*Structured Query Language*). Ed è un linguaggio dichiarativo.

CREAZIONE DELLO SCHEMA LOGICO ***Si utilizza il sottoinsieme di istruzioni raccolte nel DDL per tradurre lo schema statico del progetto concettuale.*** Utilizzando le istruzioni dichiarative del linguaggio il programmatore non deve preoccuparsi di come la loro esecuzione produrrà effettivamente i file sul disco in quanto ciò è compito specifico dell'ambiente in cui lavora. Egli deve solo concentrarsi sullo schema logico che intende ottenere in rispondenza dello schema statico che sta realizzando. Questa visione dei dati, indipendente dalla loro organizzazione fisica sulle memorie di massa, si denomina, appunto, **indipendenza fisico-logica dei dati**.
A sua volta, tale visione dei dati per i programmatori, può essere proposta, per ogni tipo di utente, in modo diversificato in virtù dei privilegi o delle autorizzazioni che gli derivano dal suo *account* (conto dei permessi con cui è registrato). Può non vedere talune tabelle e per talune altre può non essere autorizzato a modificarne i dati, e così via. Cosicché è possibile individuare tre livelli di percezione di un database nella sua fase di realizzazione:

1. **livello fisico** corrispondente all'organizzazione fisica sulle memorie di massa:
2. **livello logico** corrispondente allo schema logico delle relazioni impostate dai programmatori sulle tabelle;
3. **livello esterno** corrispondente alla viste (*view*) che dei dati hanno i singoli utenti.

REALIZZAZIONE DEI SERVIZI DEL SISTEMA INFORMATIVO ***Si utilizza il sottoinsieme di istruzioni raccolte nel DML per tradurre lo schema dinamico del progetto concettuale.*** Con esse è possibile aggiornare, ordinare, cercare dati ed effettuare elaborazioni su di essi per ottenere servizi e informazioni. L'insieme di queste istruzioni è orientato unicamente alla gestione dei dati; per cui, per ottenere prestazioni particolari o per codificare algoritmi complessi, si deve ricorrere a linguaggi di programmazione esterni.

GESTIONE E CONTROLLO DELLE PROPRIETA ***Si utilizza il sottoinsieme di istruzioni raccolte nel DCL e nel DML per realizzare il controllo e l'integrità dei dati***. Un'operazione fondamentale di gestione è costituita dal ***backup*** e ***restore*** dei dati in seguito a perdita degli stessi. Mentre, per il controllo, altrettanto fondamentale è la **gestione della sicurezza** e dei ***livelli di accesso*** ai dati regolamentati da precisi permessi concessi agli utenti nei loro *account* dall'amministratore del database. Con questo set di istruzioni si controlla, altresì, l'**integrità dei dati** stessi sia a livello delle tabelle e dei vincoli (*constraint*) per i suoi attributi, sia a livello dell'***integrità referenziale*** per l'intero database per la quale tutti i dati raccolti nelle tabelle devono rispettare le relazioni stabilite fra esse.

Abbiamo definito un sistema informativo come un insieme di risorse e procedure organizzate e coordinate fra di loro per fornire servizi tramite la produzione e diffusione di informazioni. I DBMS relazionali realizzano tali servizi mediante interrogazioni (*query*). In tali sistemi, quando sono centralizzati, la risorsa "rete", è utilizzata unicamente per accedere, da stazioni locali (*client*) ai dati posti su un unico elaboratore centrale (*server*) che provvede a fornire i servizi. Anche con l'avvento e la capillare diffusione di Internet si è posta la possibilità e la necessità di memorizzare la base dati di un S.I. non più solamente su un unico elaboratore ma su più stazioni server, collocate geograficamente in luoghi diversi, anche a grandi distanze fra di loro. In questo caso si parla di **sistemi distribuiti** e anche le interrogazioni, realizzate attraverso più computer della rete, sono dette interrogazioni distribuite (*distributed query*). La rete stessa, quindi, non più solo un collegamento, entra a far parte costitutiva fondamentale delle risorse del sistema. Questi sistemi vengono detti **DDBMS** (*Distributed Database management System*).

CARATTERISTICHE E TIPOLOGIA DEI DDBMS I condizionamenti posti dal fattore "Rete" introducono nuovi elementi per la valutazione e le scelte atte a definire i livelli ottimali di efficienza e di prestazioni (***performance***) dei S.I. distribuiti:

1. **Throughput**. E' il numero di interrogazioni eseguite in un dato tempo.
2. **Response time**. Il tempo di risposta che intercorre tra la richiesta di una query da un client e il primo risultato ricevuto.
3. **Availability**. Ossia i tempi operativi in cui un sistema è disponibile in linea.
4. **Scalability**. Cioè il numero massimo di utenti che possono contemporaneamente essere collegati al sistema (scalabilità).

I sistemi distribuiti possono essere: **in linea** (**OLTP**: *On-Line Transaction Processing databases*) o **fuori linea** (**OLBP**: *Off-line Batch Processing databases*).

Il primo è il caso tipico di Internet in cui gli utenti in rete, mediante pagine web possono, in tempo reale, fare acquisti o effettuare prenotazioni su treni o aerei. Nel secondo caso l'elaborazione è effettuata a lotti (*batch*) in cui le operazioni memorizzate nel registro delle transazioni periodicamente (ogni ora, alla fine della giornata, una volta al mese, ecc.) vengono scorse per aggiornare i file di database. Naturalmente i sistemi OLTP devono avere alti i valori di troughput, disponibilità e scalabilità e bassi tempi di risposta. Mentre in un sistema OLBP per la gestione dei dipendenti in un'azienda a scala nazionale non è necessario che la configurazione di *performance* sia così elevata.

ARCHITETTURA DEI SISTEMI DISTRIBUITI Le architetture decentrate possono essere così classificabili:

1. **Propriamente detti** in quanto **distribuiti sia nei dati che nelle operazioni**. Le tabelle del modello relazionale sono frammentate sui vari server della rete quindi per garantire la disponibilità integrale del sistema tutti i server devono essere sempre collegati e funzionanti. Per ovviare a possibili guasti di un server, che farebbe mancare una parte del modello, si ricorre a duplicazioni parziali di dati su più server. Sono, ovviamente del tipo OLTP con elevati valori di troughput e scalabilità e basso tempo di risposta.
2. **Multidatabase**. **Insiemi di S.I. locali, autonomi fra di loro**. Decentrati e alla pari (*peer-to-peer*). Essendo perfettamente autonomi possono essere ***eterogenei*** tra loro, ossia avere, ognuno di essi, un modello dei dati differente il che comporta la necessità di risolvere via software i problemi di compatibilità tra DBMS diversi per realizzare le operazioni che li coinvolgono. troughput e scalabilità devono essere elevati.
3. **Centrale-periferico**. **Master-Slave**. Esistono un server centrale (*master*), con ruoli privilegiati di coordinamento per dati e operazioni, e server decentrati (*Slave*) con compiti di raccolta di informazioni e operazioni da inviare periodicamente al server centrale per l'aggiornamento dei dati. Il modello dei dati è comune. La tipologia è OLBP. Un esempio può essere quello di una banca a scala nazionale con le sue filiali locali. Per offrire una buona disponibilità il database della sede centrale può essere duplicato in più server locali.

PROPRIETA' DI UNA BASE DATI DISTRIBUITA Con gli opportuni adattamenti sono del tutto analoghe a quelle d una centralizzata.

Memorizzazione permanente. In questo caso effettuata su più server.

Organizzazione sulla base di un modello. Per una visione ordinata e semplificata di tutti i dati.

Protezione dei dati, affidabilità e sicurezza. Oltre ai guasti hardware e software la cui probabilità per un singolo elaboratore è moltiplicata per tutti i server della rete, bisogna anche tenere conto degli errori nella rete stessa e nelle telecomunicazioni nonché della aumentata esposizione agli accessi non autorizzati (*hacker*).

Disponibilità all'utilizzo da applicazioni e utenti diversi. Per i quali non è dato sapere dove si trovano fisicamente i dati che utilizza realizzando così la cosiddetta ***trasparenza dei dati*** (*transparent property*).

Ma una ***importante differenza*** con le proprietà di una base dati centralizzata esiste e consiste nel fatto che in una base dati distribuita ***è ammessa la ridondanza*** che infatti non compare nelle proprietà indicate sopra. La duplicazione dei dati è spesso necessaria in tutti i casi in cui, come per i sistemi OLTP, bisogna implementare operazioni in tempo reale e assicurare la massima disponibilità e scalabilità.

MODELLO DEI DATI DISTRIBUITI Anche qui distinguiamo i tre classici schemi per i dati: 1) quello **interno**, fisico-hardware al livello delle memorie di massa in questo caso decentrate nei vari server; 2) quello **logico** al livello del gruppo dei programmatori che hanno una visione completa dei dati e della loro organizzazione e collocazione fisica; 3) quello **esterno** per gli utenti con *viste* parziali di dati dei quali non devono conoscere la posizione fisica.

Nella progettazione di un modello di dati distribuiti la fase specifica e caratterizzante è quella della **frammentazione** delle tabelle in più parti componenti che vanno memorizzate nei diversi server. Essa può essere effettuata secondo tre modalità: 1) in **verticale**, ***per colonne***, e le diverse sottotabelle ottenute, memorizzate in posti diversi, sono collegate fra di loro mediante una relazione *uno a uno* su una colonna chiave primaria in comune che viene quindi duplicata nei vari frammenti; 2) in **orizzontale,** ***per righe***, ripartendo la tabella sui server lasciandone integra la struttura comune degli attributi; 3) con una **tecnica mista** tra le prime due effettuando prima una frammentazione orizzontale e in seguito una verticale.
Per verificare la correttezza dell'operazione eseguita, ricostruendo la tabella dai vari frammenti distribuiti, si deve ottenere di nuovo lo schema logico di partenza.

La distribuzione dei dati tra i vari server (**siti**) della rete che compone l'intero S.I. può teoricamente avvenire in modo "***puro***" ripartendo i singoli frammenti oppure con ***repliche*** (duplicazioni) delle tabelle stesse presso ciascun server. Nei fatti essa avviene secondo ***tecniche miste*** tra queste due modalità.

OPERAZIONI DISTRIBUITE Il linguaggio SQL facilita l'esecuzione di query distribuite secondo una sintassi del tipo:

SELECT *elenco_colonne* FROM *nome_server.tabella* WHERE *condizione*

In una operazione distribuita, che deve recuperare dati in tabelle frammentate localizzate su più server, occorrerà prima individuare i ***siti*** che le contengono e quindi effettuare su ciascuno di essi la *query*. Dopodiché i dati parziali raccolti dovranno essere ricomposti e inviati al *client* che li ha richiesti. Questo tipo di operazione può essere progettata interamente dai programmatori o essere gestita in automatico dal DDBMS che, con sistemi operativi di rete avanzati, è in grado di fornire servizi di ***clustering*** (raggruppamento) i quali permettono a due o più server di comportarsi, per gli utenti, come un unico computer. L'interrogazione è ***decomposta*** in query locali e fatta eseguire ***in parallelo*** su ciascuno dei computer con i dati interessati.

GESTIONE DELLE TRANSAZIONI Un problema comune sia ai sistemi centralizzati che a quelli distribuiti è costituito dall'**accesso simultaneo e concorrente** di due o più utenti che vogliono eseguire operazioni sugli stessi dati. L'attività sui dati viene gestita nei DBMS tramite le ***transazioni*** che sono un gruppo di operazioni da eseguire considerato come un unico blocco di istruzioni. O viene eseguito tutto per intero oppure interamente annullato (*rollback*), anche le modifiche parziali già effettuate. Quando inizia una transazione, per garantire l'integrità del database, i dati interessati vengono **bloccati**, non sono cioè disponibili per altre operazioni di altri utenti. Quando avviene che una transazione debba effettuare operazioni su dati bloccati da un'altra si pone in attesa del suo sblocco e si parla di ***collisioni***. Nel caso in cui la risorsa di cui si attende l'uso sia bloccata da una transizione che a sua volta attende lo sblocco di una risorsa bloccata dalla prima si parla di ***stallo*** (*deadlock*). Le collisioni e gli stalli non sono prevedibili però sono riconoscibili dai DBMS che, secondo algoritmi preimpostati, attivano procedure automatiche per interrompere una delle transazioni concorrenti. Transazioni speciali collegate a una tabella che "scattano" automaticamente in determinate condizioni per proteggere i dati sono dette ***trigger***.

SQL – Linguaggio per RDBMS

Il modello relazionale, formalizzato da E. F. Codd, applicato alle basi di dati per costruire i sistemi informativi si è imposto, dopo la sua introduzione nei primi anni '70, come quello più semplice e "naturale" per la descrizione e il trattamento dei dati, ossia per tradurre e realizzare lo schema statico e lo schema dinamico del progetto concettuale. Lo strumento software per realizzare la gestione dei database secondo tale modello è l'***SQL*** (*Structured Query Language*), si legge "siquel", che è un linguaggio di interrogazioni messo a punto negli anni 1976-77 presso i laboratori IBM di San Josè, in California.

Fin dalla sua prima apparizione ha trovato notevoli applicazioni specialmente nell'ambito dei grandi calcolatori (*mainframe*). In seguito è stato ulteriormente sviluppato presso la ***ORACLE*** *Corporation*, società di software che con la sua versione di tale linguaggio si è specializzata nella gestione dei database. Altre importanti applicazioni di SQL si sono avute presso la Microsoft con ***SQL server*** e ad oggi ne è particolarmente diffusa anche una versione *open source*, disponibile in rete, denominata ***MYsql.*** Per altro, già a partire dagli anni '80, con la diffusione di Personal Computer adatti a realizzare reti locali, la stessa Microsoft aveva incluso nel suo pacchetto "Office" , con ***Access***, una versione di SQL in grado di gestirle.

Questa varietà di versioni ha spinto alcuni enti internazionali come l'ISO (*International Organization for Standardition*) in collaborazione con i principali produttori di DBMS, a proporre versioni standard del linguaggio non tanto per imporne una versione unica, che non sarebbe possibile in una società dove vige il libero mercato, ma piuttosto per promuoverne una base comune di riferimento per i programmatori e gli utilizzatori che agevolasse la portabilità del software da loro scritto o per lo meno non li lasciasse disorientati di fronte ad un ambiente di sviluppo diverso da quello da loro utilizzato abitualmente.

CARATTERISTICHE DEL LINGUAGGIO L'SQL:

1. **gestisce i database relazionali**
   In tutti gli aspetti: creazione, modifica, aggiornamento, sicurezza, ecc.;
2. **non è un linguaggio procedurale**
   non possiede strutture per controllare il flusso di un algoritmo o per definire strutture dati complesse; non è suo compito. Benché alcune versioni posseggano strutture di controllo algoritmico di base, ciò viene realizzato, in genere, o dall'ambiente del DBMS in cui è inserito oppure da linguaggi di programmazione esterni (*host*) supportati dall'ambiente stesso;
3. **è un linguaggio dichiarativo**
   Rivolto agli insiemi, nel caso specifico le tabelle. Il programmatore non si deve preoccupare di ***come*** farlo deve, invece, dichiarare ***quali*** sono le proprietà dei risultati che vuole ottenere riferendosi alle operazioni dell'algebra relazionale che l'SQL realizza;
4. **non può essere utilizzato da solo ma unicamente all'interno di un DBMS**
   o in modo **interattivo** per mezzo di un'interfaccia offerta dal DBMS come quella, ad esempio, di *Access* per formare le *query;* oppure in **modalità programma**, cioè inserito (*embedded*) nel sorgente di un programma procedurale, interno o esterno (in tal caso si dice *host*) al DBMS, con cui si realizzano taluni servizi complessi dello schema dinamico del sistema informativo.
5. **è uno standard**
   ossia esiste una base comune di riferimento a cui si rifanno tutte le versioni del linguaggio che implementano il set di istruzioni ANSI-SQL ISO; salvo le caratteristiche speciali, specifiche e aggiuntive, che ogni produttore di DBMS, anche per esigenze commerciali oltre che di efficienza, introduce nel suo prodotto. Ad esempio la versione del linguaggio in SQL Server denominata ***Transact-SQL.***

ARCHITETTURA SOFTWARE DEI RDBMS L'SQL è inserito in librerie software che funzionano nei DBMS i quali sono concepiti per essere installati in sistemi ***Client-Server***, sistemi, cioè, dove dei computer sono dedicati alla conservazione, cura e gestione della base dei dati (***server***) e altri, collegati in rete, sono in grado di richiederne i servizi dello schema dinamico (***client***). Ne segue che, come l'intero sistema, anche l'aspetto software si presenta con **due lati applicativi**:

) uno dedicato alla gestione dei dati sul server** dove risiede sia il **motore relazionale** (relational engine***), che realizza le funzioni definite nell'algebra relazionale, traduce le *query* SQL che riceve e ne ottimizza il codice; sia il **motore di accesso alle memorie di massa** (***storage engine***) dove risiedono fisicamente i dati;

***) e uno dedicato alle richieste di elaborazione** per l'ottenimento dei servizi, che viene installato **sui client** e spesse volte contiene anche moduli di elaborazione locale i quali, con procedure che non compromettono l'integrità della base dati salvaguardata sul server, ne alleggeriscono però il lavoro e minimizzano il traffico sulla rete.

Si tratta di gestire un flusso bidirezionale:

dal client verso il server (*lato client*) dove le librerie software hanno il compito di elaborare la richiesta del servizio e inviarla al server tramite un messaggio sulla rete;

dal server verso il client (*lato server*) dove il motore relazionale traduce e ottimizza la richiesta SQL che riceve, recupera i risultati, tramite il motore di accesso ai dati, e li reinvia al *client* che ne ha fatto richiesta.

Spesse volte il completamento di un servizio può implicare più di uno di questi "colloqui" tra il server ed uno dei client per cui, dove possibile, talune elaborazioni, ad esempio il controllo e la congruità formale delle richieste, vengono effettuate preliminarmente sul *client* prima dell'invio, invece che sul server, limitando così anche il traffico sulla rete.

ASPETTI E CLASSIFICAZIONE DELLE ISTRUZIONI DEL LINGUAGGIO seconda **le funzionalità** e gli aspetti del **progetto concettuale che traducono** e realizzano:

TIPO DI FUNZIONALITA'	Obbiettivo	PROGETTO CONCETTUALE	Esempi di istruzioni
DDL *Data Definition Language*	Definizione dei dati, tabelle e relazioni	MODELLO STATICO	***CREATE ALTER DROP***
DML *Data Manipulation Language*	Aggiornamento, manutenzione e interrogazioni	MODELLO DINAMICO	***SELECT INSERT UPDATE DELETE***
DCL *Data Control Language*	Sicurezza dei dati in ambiente multiutente	"" "" "" Congruenza e integrità	***GRANT DENY REVOKE***

COSTRUTTO DI UNA ESPRESSIONE SQL E' composto, tipicamente, da una **chiave principale**, con l'indicazione dell'insieme su cui operare, eventualmente seguita da **clausole** che descrivono o circoscrivono specifiche azioni:

SELECT *<Colonne>* FROM *<Tabelle>* [WHERE *<Condizione>*]

In un ambiente multiutente distribuito, che può gestire più di un database, l'indicazione dell'***insieme su cui operare*** necessita di tutte le specifiche necessarie affinché possa essere individuato, secondo un formato denominato ***fully qualified name***:

<NomeServer>.<NomeDatabase>.<NomeUtente>.<Nometabella>

REGOLE LESSICALI L'SQL non distingue fra maiuscolo e minuscolo ma è buona norma scrivere in maiuscolo le parole chiave e le clausole. Per i nomi degli attributi e delle tabelle, gli identificatori in genere, è meglio evitare caratteri speciali (lettere accentate, segni di interpunzione, ecc.) e il carattere spazio che costringerebbe a racchiudere l'intero nome tra parentesi quadre (piuttosto che *[codice cliente]* è meglio scrivere *CodiceCliente*, con le iniziali delle parti componenti in maiuscolo). Commentare va sempre benissimo. Lo si fa racchiudendo il commento tra /* *…linee di commento …* */.

DDL – *Definizione dei dati*

Con la parte delle istruzioni SQL classificate come DDL, Data Definition Language, è possibile realizzare, in un RDBMS, lo schema statico del progetto concettuale di un sistema informativo traducendolo in uno schema logico di un database relazionale.

Secondo lo standard SQL un sistema informativo è composto da 1) una base dati; 2) l'insieme delle procedure applicative che ne realizzano le operazioni e i servizi previste nello schema dinamico; 3) le persone che lo creano (i programmatori), lo gestiscono (gli amministratori) oppure lo usano (gli utenti). In questa sezione ci occupiamo del primo punto: la definizione della base dati.

Un database SQL contiene i seguenti elementi fondamentali: **tabelle**, strutture informative di base; **domini**, l'insieme dei valori che legittimamente può assumere un attributo; **viste**, tabelle virtuali derivate i cui dati visualizzati scaturiscono dai risultati di interrogazioni effettuate sulle tabelle relazionate di base, o altre viste, e ne costituiscono quindi un loro sottoinsieme; **indici**, archivi complementari alle tabelle a cui sono associati contenenti, in modo ordinato, i valori degli attributi utilizzati come chiavi di accesso ai dati della tabella stessa. Sono essenziali per effettuare ricerche efficienti e prevenire la duplicazione di dati; **catalogo**, file associato al database che contiene informazioni sulla sua struttura e dati sulla sicurezza.

Vediamo ora quali sono le istruzioni fondamentali per creare un database, le sue tabelle di base, i vincoli sui valori degli attributi di queste ultime e i loro reciproci collegamenti relazionali atti a costituire lo schema logico di base eventualmente da sottoporre a normalizzazione per ottenere uno schema logico relazionale finale ottimizzato (solo per completezza sono riportate qui alcune istruzioni più propriamente DML). Tutte le istruzioni SQL possono essere eseguite in ***modo interattivo*** con un *prompt* gestito dall'ambiente DBMS che si utilizza oppure in ***modo programma*** inserendole in uno *script*, un file sorgente scritto con un editor qualsiasi, che verrà eseguito da un interprete/compilatore del DBMS al momento della sua lettura. *Gli esempi, il più generali possibile, seguono la versione Transact-SQL(T-SQL) del linguaggio, implementata in SQL Server. Tra parentesi quadre sono indicate le parti opzionali, non obbligatorie.*

CREAZIONE DI UN DATABASE

```
CREATE DATABASE NomeDatabase
ON
PRIMARY ( NAME = NomelogicoFile,
          FILENAME = percorsoFile,
          SIZE = DimensioneIniziale,
          MAXSIZE = MassimaDimensione,
          FILEGROWTH = IncrementoFile)
LOG ON (  NAME = NomelogicoFile,
          FILENAME = percorsoFile,
          SIZE = DimensioneIniziale,
          MAXSIZE = MassimaDimensione,
          FILEGROWTH = IncrementoFile)
```

La clausola PRIMARY si riferisce al file principale (.mdf). La clausola LOG ON si riferisce al file del registro delle transazioni (.ldf).

CREAZIONE DI UN DOMINIO

```
CREATE DOMAIN NomeDominio AS TipoDato
  DEFAULT ValoreDiDefault
  <Vincoli Integrità della Colonna>
```

E' utile per la definizione, una volta per tutte, di una colonna che può essere impiegata in più di una tabella.

CREAZIONE DI UN INDICE

```
CREATE [UNIQUE] INDEX NomeIndice
  ON NomeTabella (Cln1[, Cln2] …)
```

La clausola UNIQUE, automatica per le chiavi primarie, impone che ogni valore sia unico nella colonna su cui si costruisce l'indice. Possono intervenire più colonne e per ognuna di esse si può stabilire se l'ordinamento debba essere ascendente (ASC) o discendente (DESC).

MODIFICA DELLO SCHEMA

```
ALTER TABLE NomeTabella
ADD NuovaColonna
DROP NomeColonna
```

Con ADD si aggiunge una nuova colonna ad una tabella. Con DROP la si elimina.

CREAZIONE DI UNA TABELLA

```
CREATE TABLE NomeTabella
( NomeColonna TipoDiDato [ValoreDefault]
            [IDENTITY(ValoreIniziale,
                      Incremento)] , … … … )
```

Per ogni colonna bisogna indicare il tipo di dato che dovrà contenere. Oltre a quelli standard, versioni di SQL diverse possono offrire ulteriori tipologie.

chiave	Tipo	Byte	Versione
SMALLINT	Intero	2	Standard
INTEGER	Intero	4	Standard
DECIMAL	Reale	2 – 17	Standard
FLOAT	Reale	4	Standard
DOUBLE	Reale	8	Standard
CHAR(n)	Stringa	n	Standard
DATE	Data	8	Standard
TIME	Ora	8	Standard
BIT	Logico	1	Standard
MONEY	Reale	8	T-SQL
NTEXT	Testo	0 – 2 Gb	T-SQL
IMAGE	Immagine	0 – 2 Gb	T-SQL

La clausola IDENTITY definisce una colonna con valori inseriti automaticamente ad ogni nuova riga. In genere viene utilizzata per le chiavi primarie.

INSERIMENTO DATI *(DML)*

```
INSERT INTO NomeTabella (colonna1, colonna2,…)
VALUES (Valore1, Valore2, …)
```

MODIFICA DATI *(DML)*

```
UPDATE NomeTabella
  SET Colonna1 = Valore, Colonna2 = Valore, …
  [WHERE <Condizione>]
```

L'istruzione UPDATE si applica a tutta la tabella. Per modificare una singola riga occorre utilizzare opportunamente la clausola WHERE imponendo una condizione di ricerca sulla chiave primaria.

CANCELLAZIONE DATI *(DML)*

```
DELETE FROM NomeTabella
  [WHERE <Condizione>]
```

Stessa avvertenza che per l'istruzione UPDATE.

DEFINIZIONE DEI VINCOLI

```
CREATE TABLE NomeTabella
( [CONSTRAINT NomeVincolo]
  NomeColonna <TipoColonna> <VincoloIntegrità> ..
FOREING KEY ColonnaTabellaEsterna, …
  REFERENCES
    NomeTabellaInterna (ColonnaTabellaInterna) … )
```

Per salvaguardare l'integrità dei dati è possibile imporre delle **restrizioni** per ogni valore che si può immettere in una determinata colonna e in tal caso il vincolo si può inserire a seguire la sua definizione. Se il vincolo riguarda l'intera tabella si inserisce al termine delle definizioni di tutte le colonne. Per imporre un vincolo di ***integrità referenziale*** su una tabella lato *molti* (detta ***esterna***) collegata ad una tabella lato *uno* (detta ***interna***) si usa la clausola FOREING KEY (associata alla clausola REFERENCES) ad indicare la colonna/e in comune con la tabella interna dove ha funzione di chiave primaria. Abbiamo i seguenti **tipi di integrità**:

Tipo	Restrizioni su…	Clausole SQL
Su Domini	Colonna	DEFAULT CHECK NOT NULL
Su Entità	Tabella	PRIMARY KEY UNIQUE
Referenziale	Relazioni	FOREIGN KEY

Quando si cerca di inserire un valore non ammesso per una colonna o si viola in qualche modo l'integrità referenziale che potrebbe compromettere il collegamento tra tabelle in relazione il DBMS segnala all'utente il vincolo che ha violato e non permette il completamento dell'operazione.

Esempio creazione database per la gestione delle assenze.

```
CREATE DATABASE Assenze

CREATE TABLE Allievi (Classe CHAR(6),NumeroOrdine
INTEGER, Cognome CHAR(15) NOT NULL, Nome CHAR(15)
NOT NULL, PRIMARY KEY (Classe, NumeroOrdine) )

CREATE TABLE Classe (Classe CHAR(6) PRIMARY KEY)

CREATE TABLE Anni (Anno INTEGER PRIMARY KEY)

CREATE TABLE Mesi (Mese INTEGER PRIMARY KEY)

CREATE TABLE Assenze (NumeroOrdine INTEGER, Classe
CHAR(6), Giorno INTEGER, Mese INTEGER, Anno INTEGER,
Tipo CHAR2),
CONSTRAINT TipoAssenzaRitardo
CHECK (Tipo IN ("AA","RR","AG","RG")
PRIMARY KEY (NumeroOrdine, Classe, Giorno,Mese,Anno)
FOREIGN KEY (Classe, NumeroOrdine)
  REFERENCES Allievi (Classe, NumeroOrdine)
FOREIGN KEY (Classe) REFERENCES Classi (Classe)
FOREIGN KEY (Mese) REFERENCES Mesi (Mese)
FOREIGN KEY (Anno) REFERENCES Anni (Anno)
```

DML – *Le interrogazioni*

Con la parte delle istruzioni SQL classificate come DML, Data Manipulation Language, è possibile realizzare, in un RDBMS, lo schema dinamico, con le operazioni e i servizi, previsti nel progetto concettuale di un sistema informativo tramite interrogazioni (*query*) dello schema logico di un database relazionale.

Le operazioni e i servizi da realizzare vengono effettuate sulla base di **interrogazioni** che definiscono un sottoinsieme dei dati di base riorganizzati secondo i criteri di selezione, ricerca e ordinamento impostati. I risultati delle interrogazioni costituiscono, a loro volta, benché virtuali, nuove tabelle, dette **derivate**, a tutti gli effetti riutilizzabili per ulteriori interrogazioni. Le tabelle derivate possono essere memorizzate nella base dati con un nome che le identifichi e in tale caso vengono indicate come **viste** (***view***).

L'esecuzione di una *query* viene effettuata dopo una sua traduzione in linguaggio macchina, tramite l'interprete o il compilatore. Questa fase viene gestita dal motore relazionale del RDBMS il quale produce, attraverso un processo di ottimizzazione, il codice binario necessario a realizzare la sequenza di operazioni dell'algebra relazionale che sono state richieste. L'attivazione di questo codice fa intervenire i moduli del sistema dedicati al recupero fisico dei dati memorizzati sulle memorie di massa esterne (***storage engine***). Una volta raccolti, i dati vengono opportunamente assemblati e prodotti. Tra i compiti dell'**ottimizzatore delle interrogazioni** (***query optimizer***) c'è quello di scegliere il modo migliore per raggiungere i dati in una tabella: percorrerla tutta, riga per riga, oppure sfruttare gli indici e, in quest'ultimo caso, individuarne quello più efficiente con il minor numero di ripetizioni nelle chiavi di ricerca.

L'istruzione SELECT realizza le operazioni dell'algebra relazionale

```
SELECT [DISTINCT] Cln1,…clnN [AS clnRidenominata] …
  FROM ElencoTabelle
  [WHERE Condizione]
  [GROUP BY ElencoColonneDaRaggruppare]
  [HAVING CondizioneDiRaggruppamento]
  [ORDER BY ElencoColonnePerOrdinamento]
```

L'istruzione **SELECT** costituisce la direttiva di base per effettuare una *query*. Con le sue varie clausole è possibile realizzare le operazioni dell'algebra relazionale:

AS realizza l'operazione di **ridenominazione** di una colonna

FROM realizza l'operazione **prodotto** tra le tabelle.

WHERE realizza l'operazione di **selezione** nell'insieme prodotto da FROM. La condizione di ricerca che supporta può essere molto ricca ed offre varie opportunità: sono ammessi i classici ***operatori di confronto***: <,>,>=,<=,<>; gli ***operatori logici***: AND, OR, NOT e ***operatori speciali SQL*** quali:

1) **BETWEEN *Valore1* AND *Valore2*** per selezionare valori tra i due estremi indicati;
2) **LIKE *stringa*** per confrontare dati di tipo carattere. Può contenere i cosiddetti caratteri *jolly* **%** per indicare un qualsiasi insieme di caratteri e _ per indicare qualsiasi carattere unico (in Access corrispondono, rispettivamente a ***** e a **?**).
3) **IN (*Valore1*, *Valore2*,…)** per verificare se il valore di una colonna è contenuto nell'insieme indicato;
4) **IS NULL; IS NOT NULL** per verificare se il contenuto di una colonna è o non è nullo.

La clausola **DISTINCT** impone di eliminare dal prodotto ottenuto da FROM tutte le righe duplicate.

La clausola **GROUP BY** raggruppa, invece, le righe ottenute a seconda del valore contenuto nelle colonne indicate; mentre la clausola **HAVING**, ad essa collegata, impone una condizione di selezione sul contenuto delle colonne.

La clausola **ORDER BY [ASC|DESC]**, infine, permette di ottenere i risultati richiesti secondo un determinato ordinamento, ascendente o discendente, sulle colonne indicate.

Elaborazione dati nelle interrogazioni

L'SQL mette a disposizione, nelle interrogazioni, varie **funzioni aggregate** che agiscono su tutti i dati contenuti in una colonna: **COUNT**() per contare numero di righe di una tabella, numero di valori in una colonna, ecc.; **AVG**(), **MAX**(), **MIN**(), **SUM**() per calcolare medie, valori massimi, minimi, sommatorie; e altre ancora.

E' possibile, inoltre, includere nelle colonne da visualizzare anche **colonne calcolate** che contengono il risultato di una elaborazione effettuata riga per riga sulla selezione della base dati richiesta.

Operazioni sugli insiemi

Sono disponibili gli operatori per realizzare le tipiche operazioni sugli insiemi:

<query1> **UNION** *<query2>* per l'***unione*** di due tabelle derivate
<query1> **INTERSECT** *<query2>* per l'***intersezione*** di due tabelle derivate
<query1> **EXCEPT** *<query2>* per la ***differenza*** tra due tabelle derivate

L'operatore JOIN traduce le relazioni tra le tabelle

```
SELECT <ElencoColonne>
  FROM Tabella1 <Tipo di Join> JOIN Tabella2 ON <Condizione di join>
  WHERE Condizione
```

Come si sa due tabelle in relazione possiedono una **colonna in comune**, con funzione di chiave primaria nella tabella interna e di chiave esterna in quella connessa. Per ottenere una tabella derivata che visualizzi tutti i dati fra loro collegati possiamo opportunamente utilizzare la clausola WHERE di una istruzione SELECT:

```
SELECT Tab1.Cln1, Tab1.cln2, , … Tab2.Cln1, Tab2.cln2,…
FROM Tab1, Tab2
WHERE Tab1.ClnComune = Tab2.ClnComune
```

Questa stessa operazione è ottenibile con un operatore apposito che l'SQL mette a disposizione: il **JOIN**:

```
SELECT Tab1.Cln1, Tab1.cln2, , … Tab2.Cln1, Tab2.cln2,…
FROM Tab1 INNER JOIN Tab2 ON Tab1.ClnComune = Tab2.ClnComune
```

e I risultati ottenuti sono del tutto identici.
Oltre a quello appena visto, **INNER JOIN**, detto anche JOIN naturale o Theta JOIN, l'SQL ne prevede anche altri tipi:

- **LEFT JOIN** o **RIGHT JOIN**, join sinistro o join destro, in cui nella tabella risultato, oltre alle righe che derivano dal theta-Join si aggiungono le righe della tabella posta, rispettivamente, a sinistra o a destra dell'operatore JOIN;
- **CROSS JOIN** che realizza il prodotto tra le due tabelle;

E' possibile effettuare, inoltre, anche una operazione detta **SELF JOIN** che si applica tra due copie della stessa tabella.

Subquery

Una *subquery* è una interrogazione contenuta all'interno di un'altra (è detta, infatti, **query interna**) ed è molto usata nella produzione dei servizi dello schema dinamico. Si tenga presente che una *subquery* può contenerne, a sua volta, un'altra e così via. Ciò fa immaginare come, nel concreto, possa diventare complessa la formulazione di una interrogazione dove si utilizza, per esempio, una o più *subquery* siffatte in colonne calcolate e/o in condizioni di ricerca elaborate.

Query con parametri

I **parametri** sono variabili di ingresso che vengono valorizzati solo al momento in cui l'interrogazione viene avviata. Hanno la funzione di "***segnaposto***" per indicare al sistema che prima di cominciare l'operazione deve chiedere cosa effettivamente utilizzare per effettuarla. Un parametro si indica facendolo precedere dal segno **%**

Es.: SELECT nominativo FROM studenti WHERE classe = %classe

Le Viste

CREATE VIEW NomeVista [cln1Vista, cln2Vista,…] AS <Interrogazione SQL>

Si crea una nuova **tabella virtuale**, derivante dall'interrogazione SQL richiamata, che entra a far parte della base dati e può essere utilizzata per offrire una visione semplificata dei dati, per nuove interrogazioni, per aggiornamenti, e così via.

DCL – *Gestione della sicurezza*

Con la parte delle istruzioni SQL classificate come DCL, *Data Control Language*, è possibile impostare e gestire la protezione dello schema logico di un database e realizzare i servizi di un sistema informativo distribuito.

L'accesso ai dati in un sistema informativo, centralizzato o distribuito che sia, costituisce, naturalmente, lo scopo del suo esistere ma comporta anche la messa a punto dei necessari accorgimenti per la protezione della sua integrità. Sono individuabili **tre livelli di accesso** che, in successione, devono essere superati per raggiungere le informazioni e ottenere i servizi: 1) **al server**; 2) **allo schema logico** della base di dati; 3) **ai file fisici** che la compongono. Per accedere ai dati bisogna essere riconosciuti dal sistema affinché esso possa associare a tale accesso l'opportuno livello di autorizzazioni per le attività effettivamente realizzabili. Per cui ogni utente, con un ID account personale di riconoscimento e una password, deve essere inserito in un **gruppo** al quale siano collegabili specifici permessi e privilegi con cui può interagire con la base dati e accedere ai servizi che può ottenere. Tale raggruppamento avviene **per funzione** svolta nel sistema informativo (programmatore, amministratore, impiegato, dirigente, utente generico, ecc.), oppure, negli ambienti distribuiti, anche **per area geografica**.

Il primo passo consiste nell'accedere al server dove sono memorizzati i dati e il sistema di gestione. Questo controllo, oltre che dal RDBMS, può anche essere preceduto da quello effettuato dai servizi di sicurezza del sistema operativo di rete in cui lo stesso server è inserito. Mentre, se per il terzo sbarramento, quello riguardante il livello fisico dei dati, gli amministratori possono ricorrere al loro criptaggio (*encription*), per il secondo livello, lo schema logico dei dati, l'accesso ai servizi viene pianificato, per singoli o gruppi di utenti, da opportune istruzioni SQL.

Permessi, dinieghi e revoche per l'utilizzo di specifiche istruzioni

GRANT | DENY ALL | istruzione1, istruzione2 …
{ ON {NomeTabella|NomeVista} [Colonna1, colonna2,…] }
TO AccountUtente1|AccountGruppo1, AccountUtente2|AccountGruppo2, …

L'istruzione **GRANT** viene utilizzata per concedere permessi e privilegi sia all'utilizzo di istruzioni DDL quali: ROLLBACK TRANSACTION, BACKUP DATABASE, BACKUP LOG, ecc. sia all'utilizzo di istruzioni DML che modificano i dati quali: SELECT, UPDATE, INSERT, DELETE e possono riguardare singole colonne, come si vede, o intere specifiche tabelle. Per negare i permessi si usa l'istruzione **DENY**.

Es.: GRANT ALL PRIVILEGES
ON Studenti ON Classi ON Docenti
TO gruppoDocenti

DENY INSERT, UPDATE, DELETE
ON Studenti, ON Classi, ON Docenti
TO gruppoStudenti

Speculare rispetto all'istruzione di concessione/negazione dei permessi è quella di revoca degli stessi che si effettua con **REVOKE**:

REVOKE ALL | istruzione1, istruzione2 …
{ ON {NomeTabella|NomeVista} [Colonna1, colonna2,…] }
FROM AccountUtente1|AccountGruppo1, AccountUtente2|AccountGruppo2, …

Gestione delle transazioni

Una transazione è un insieme di istruzioni trattate dal sistema come singola unità di elaborazione: o viene completata tutta o interamente annullata. Essa può essere **implicita** se viene gestita automaticamente dal sistema (per esempio quando incontra istruzioni quali INSERT, UPDATE, DELETE, DROP, CREATE, ecc.); oppure **esplicita** se viene impostata manualmente dal programmatore il quale, per realizzare una complessa operazione che deve essere interamente compiuta per integrarla nei dati altrimenti potrebbe comprometterne lo schema logico, deve segnalare l'inizio e la fine della transazione stessa e intercettare, nei punti critici, la possibilità che l'elaborazione non possa proseguire e quindi annullarla completamente.

Per iniziare una transazione: **BEGIN TRANSACTION** [Nome Transazione]
IstruzioneSQL1, istruzioneSQL2, ….

Per interrompere/annullare: **ROLLBACK TRANSACTION**

Per terminare una transazione: **COMMIT TRANSACTION**

I TRIGGER

I ***trigger*** ("scatto") sono transazioni speciali che i programmatori associano alle modifiche di dati in una certa colonna di una tabella. Se un'altra transazione tenta la modifica dei dati di una tabella protetta da un *trigger*, secondo modalità che non ne rispettano le specifiche programmate, esso interviene, *scatta*, e ne impedisce il completamento.
Per creare un trigger:

CREATE TRIGGER NomeTrigger
ON NomeTabellaProprietariaDelTrigger
{**FOR** INSERT|UPDATE|DELETE} |
{**INSTEAD OF** INSERT|UPDATE|DELETE}
AS <Transazione associate alla tabella>

Come si vede il *trigger* può essere associato ad ogni specifica operazione di modifica (**FOR**) e per ognuna di esse si può programmare la condizione di intercettamento e quindi di annullamento dell'operazione che potrebbe compromettere l'integrità dello schema. Da notare il tipo di trigger **INSTEAD OF** ("invece di") il quale, alla sua attivazione, fa eseguire un'operazione alternativa a quella che blocca.

TAVOLA RIASSUNTIVA DELLE PRINCIPALI ISTRUZIONI SQL

Realizzazione del progetto concettuale

SCHEMA LOGICO RELAZIONALE

Schema statico — *Data Definition Language*

CREAZIONE DELLA BASE DATI

Database	*Domini*	*Tabelle*	*Indici*	*Vincoli*	*Modifica*
CREATE DATABASE .. **ON PRIMARY (..) LOG ON (..)**	**CREATE DOMAIN** .. ***AS*** .. ***DEFAULT*** ..	**CREATE TABLE** .. (*colonne, ..* ***IDENTITY*** ..)	**CREATE INDEX** .. **ON** *tabella (Colonna/e)*	CREATE TABLE .. **CONSTRAINT** ***FOREING KEY REFERENCES***	**ALTER TABLE** .. ***ADD*** .. ***DROP*** ..

Schema dinamico — *Data Manipulation Language*

AGGIORNAMENTO DELLA BASE DATI

Inserimento	*Modifica*	*Cancellazione*
INSERT INTO *Tab (cln1, cln2,..)* ***VALUES*** *(Val1, Val2, ..)*	**UPDATE** *Tab* ***SET*** *Cln1 = Val1, Cln2 = Val2,* [***WHERE*** *<Cdz>]*	**DELETE FROM** *Tab* [***WHERE*** *<Cdz>*]

INTERROGAZIONE DELLA BASE DATI

Selezione dei dati	*Funzioni aggregate*	*Join*	*Operazioni su insiemi*	*Viste*
SELECT *colonne* **FROM** *tabelle* ***WHERE*** *cdz* ***GROUP BY*** *cln* ***HAVING*** *cdz* ***ORDER BY*** *cln*	**COUNT**() **AVG**() **MAX**() **MIN**() **SUM**() **…**	**SELECT … FROM *tab1*** **INNER JOIN *tab2* ON *Tab1.clnCom = Tab2.clnCom***	**UNION** **INTERSECT** **EXCEPT**	**CREATE VIEW …** **AS *<query>***

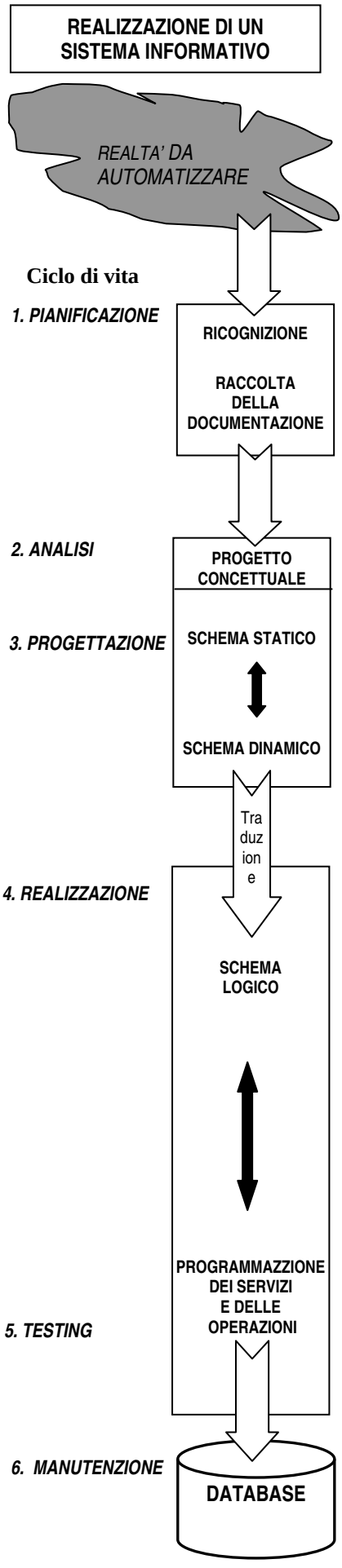
TRANSAZIONI

Inizio	*Annullamento*	*Completamento*	*Trigger*
BEGIN TRANSACTION	**ROLLBACK TRANSACTION**	**COMMIT TRANSACTION**	**CREATE TRIGGER .. ON** *tabella* **FOR** INSERT|UPDATE|DELETE **INSTEAD OF** INSERT|UPDATE|DELETE **AS** *transazione*

Data Control Language

CONTROLLO E GESTIONE DELLA SICUREZZA

Permessi	*Dinieghi*	*Revoche*
GRANT ***ALL*** *istr1, istr2 …* **ON** *Tabella* **TO** *utenti/gruppi*	**DENY** ***ALL*** *istr1, istr2 …* **ON** *Tabella* **TO** *utenti/gruppi*	**REVOKE** ***ALL*** *istr1, istr2 …* **ON** *Tabella* **FROM** *utenti/gruppi*



LA PROVA SCRITTA DI INFORMATICA quasi sempre si articola in **due fasi**:

- **la prima** incentrata su un tema generale, che deve essere affrontato con tono discorsivo, ma comunque tecnico, dando prova di competenza e conoscenza;
- **la seconda** basata sull'impostazione, per grandi linee, di procedure operative collegata al tema proposto al punto precedente.

I temi proposti sono sempre riconducibili alla realizzazione di un sistema informativo, alle strutture dei dati che in esso sono utilizzati, alle applicazioni software che ne realizzano i servizi e le operazioni e alla relativa documentazione. Attenersi al processo di sviluppo, ciclo di vita di un sistema operativo, senz'altro orienta e vi darà uno schema di sviluppo per impostare il lavoro.

Qualche suggerimento:

- sviluppate i passi che individuate con ordine, magari numerandoli e lasciando un rigo tra i punti;
- non lesinate riferimenti alla documentazione e alla sua necessarietà;
- le spiegazioni devono essere sintetiche ma chiare ed esaurienti;
- nella codifica di qualche operazione utilizzate nomi esplicativi, anche lunghi, che contengano evidente il rimando all'oggetto cui si riferiscono *(una tabella dei clienti invece che CLIENTI è meglio nominarla TabClienti, una query sulla stessa tabella è meglio nominarla qryClienti, e così via);*
- commentate il più possibile.

Prima parte: Costruire il progetto concettuale.

1. raccogliere e riordinare la ***documentazione*** sul problema tra i dati proposti nella traccia e con eventuali *proposte aggiuntive;*
2. progettare lo ***schema statico*** in un apposito paragrafo evidenziato nel tema che state svolgendo:
   a) individuare i *fatti elementari* nella documentazione e nelle specifiche;
   b) individuare *i tipi di entità*;
   c) elencare gli *attributi* (entità) di tutti i tipi di entità;
   d) individuare le *relazioni* tra i tipi di entità;
   e) evidenziare i *vincoli* che regoleranno gli inserimenti negli attributi critici dei tipi di entità.
3. progettare lo ***schema dinamico*** in un apposito paragrafo:
   a) elencare ogni operazione evidenziando i parametri di ingresso e quelli in uscita
   b) documentare ogni operazione commentandone le specifiche di utilizzo e elencando i servizi che risolve.

Seconda parte: realizzare il sistema informativo con un DBMS.

1. analizzare le ***specifiche tecnologiche***:
   a) **hardware:** motivare la scelta del:
      - tipo di sistema centralizzato (mono o multiutente) oppure distribuito;
      - tipo di memorie di massa (dischi, nastri, ecc) sulla base delle loro prestazioni, capacità, costi;
   b) **software:** scelta del sistema di sviluppo (DBMS);
2. scrivere i moduli di programma per la **creazione della base dati**;
   a) tradurre lo **schema statico** del progetto concettuale nello schema logico utilizzato (gerarchico, reticolare, relazionale o a oggetti) fare cenno ai vari tipi di schema logico.
   b) Evidenziare che nello schema logico relazionale le tabelle sono ottenute mediante un **processo di normalizzazione** e fare cenno alle tre **forme normali**;
   c) Descrivere che cos'è la **dipendenza funzionale** e sottolineare che l'obiettivo è quello di ottenere una base dati le cui tabelle verifichino tutte quante la rispondenza alla **quarta forma normale**;
   d) Indicare che nella creazione della base dati si usa la parte **DDL** del linguaggio interno del DBMS.
   e) Impostare e descrivere opportunamente le regole di **integrità referenziale** che salvaguardano l'efficienza del sistema.
3. scrivere i moduli di programma per la **realizzazione dei servizi**;
   a) tradurre le operazioni descritte nello **schema dinamico** del progetto concettuale
   b) indicare che si utilizza la parte **DML** del linguaggio interno del DBMS;
   c) indicare che per operazioni complesse è possibile utilizzare un linguaggio procedurale ospite **(host)**
   d) i indicare che le operazioni e i servizi si realizzano mediante **transazioni** e ***trigger*** e spiegare cosa sono;
4. impostare la **sicurezza** e i **livelli di accesso** al sistema
   a) fare cenno alla necessità della sicurezza dei dati e alla conservazione della loro **integrità**;
   b) ricordare l'estrema importanza del **backup** periodico, parziale o totale, dei dati:
   c) indicare che per impostare la sicurezza si ricorre alla parte **DCL** del linguaggio interno del DBMS;
   d) fare cenno all'uso dei **gruppi**; alle varie modalità di accesso ai dati, alle **viste**;
   e) Spiegare che un database, oltre che un insieme di record e file fisici sulle memorie di massa ha, soprattutto, un **livello interno** per i programmatori e amministratori e un **livello esterno** per gli utenti.

Organizzate ogni cosa secondo un filo logico che le leghi. Non producete elenchi di frasi sconnesse. Questa scaletta va bene come riferimento generale, è un promemoria non è il vostro compito, ma vi tornerà utile. Molto utile se vi sarete adeguatamente preparati.