Informatica Documentale
Modulo III

Marino Miculan
miculan@dimi.uniud.it

Introduzione alle basi di dati

- La *raccolta*, l'*archiviazione* e la *manipolazione* di *dati* sono operazioni ricorrenti in molte attività (e.g., conti bancari, elenchi telefonici, elenchi degli iscritti ad un corso di laurea ecc.).
- Tali attività possono prescindere dall'uso del computer; tuttavia questi ultimi garantiscono una memorizzazione ed un trattamento dei dati stabili ed efficienti.
  - Esempio eclatante: cataloghi bibliotecari, dove si usavano (e si usano ancora) le schede in cartoncino
- Sempre più necessario: collegamento tra informazioni diverse (I così detti "controlli incrociati")
- I DataBase Management System (DBMS) sono le applicazioni per questo trattamento dei dati

Dati e Informazioni

- Un *dato* in sé non costituisce un'informazione in quanto consiste semplicemente di un insieme di simboli; ad es., la sequenza di caratteri *Mario Rossi* e le cifre *06 658976* non hanno un significato intrinseco.
- Quando un dato viene interpretato come risultato di un'interrogazione (e.g., "chi è il direttore della biblioteca e qual è il suo numero telefonico?") diventa *informazione*.
- Quindi il significato (il contenuto) di un dato è solo in relazione ad una domanda

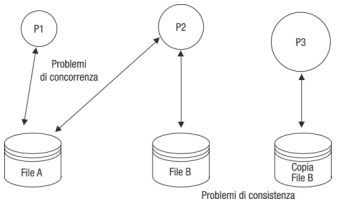
Basi di dati (database)

- Ogni organizzazione necessita di un *sistema informativo* per il trattamento delle informazioni inerenti alla sua attività.
- Solitamente solo una parte del sistema informativo di un'organizzazione sfrutta la tecnologia legata ai computer.
- I dati (rappresentanti le informazioni) sono *stabili*, i.e., difficilmente cambiano la propria struttura nel corso del tempo.
- Le procedure del sistema informativo che operano sui dati invece sono facilmente soggette a modifiche.
- Un *database* (nella sua accezione più generale) è un insieme di dati che *codificano* dell'informazione utile in un sistema informativo.
- In base a quanto detto a proposito della stabilità dei dati, un database rappresenta una *risorsa* fondamentale da sfruttare e proteggere.

L'accesso tradizionale ai dati

Fino agli anni 60: non esistevano software specifici per la gestione dei dati, che venivano memorizzati in file e manipolati per mezzo di linguaggi di programmazione tradizionali.

- L'organizzazione dei dati per mezzo di file comporta dei problemi di **condivisione** quando più utenti devono lavorare sugli stessi dati.
- Inoltre l'autonomia delle singole procedure operanti sui file, comporta la duplicazione di questi ultimi con la conseguente problematica di garantire la **consistenza** dei dati (e.g., date due copie di uno stesso file, quale è da ritenersi valida?).



I DBMS mirano a risolvere questi problemi, in modo efficiente e generale

Caratteristiche dei DataBase e dei DataBase Management Systems

- Un database può contenere una *grande quantità di dati*; per questo utilizza principalmente la *memoria secondaria* di un computer.
- I database sono condivisi, i.e., garantiscono l'accesso a più applicazioni ed utenti *contemporaneamente*.
- La memorizzazione dei dati in un database è permanente, i.e., i database sono *persistenti*.
- I DBMS sono *affidabili*, i.e., assicurano che i dati non andranno persi oppure che potranno essere recuperati, in caso di problemi hardware o software, tramite meccanismi di *backup* e/o *data recovery*.
- Tramite un meccanismo di autenticazione i DBMS assicurano la *privacy* dei dati, i.e., ogni utente è in grado di accedere soltanto ai dati che gli competono.
- I DBMS sono *efficienti*, i.e., svolgono il loro compito utilizzando al meglio le risorse del sistema.

Caratteristiche dei DBMS

- I DBMS seguono la strategia opposta a quella della memoria virtuale nei sistemi operativi: non si cerca di "simulare" una grande RAM, ma piuttosto si forniscono specifici linguaggi e funzionalità per accedere efficientemente ai dati in memoria secondaria
- I DBMS sono prodotti costosi e complessi la cui introduzione in una realtà esistente può comportare notevoli investimenti in termini di hardware, software ed addestramento di personale.
- Però i DBMS permettono di incrementare la produttività degli utenti che li utilizzano, a lungo termine.
- Spesso i DBMS forniscono un numero di servizi maggiore di quelli richiesti; la difficoltà di isolare soltanto quelli necessari comporta molte volte costi e perdita di efficienza dell'attività.

Modelli dei dati

- Un modello di dati fornisce un insieme di costrutti per l'organizzazione dei dati.
- Esistono vari modelli dei dati:
  - Il modello gerarchico (hierarchical data model)
  - Il modello a grafo (network data model)
  - **Il modello relazionale (relational data model)**
  - Il modello a oggetti (object data model)
- I modelli dei dati sono chiamati modelli logici in quanto i loro costrutti, pur essendo astratti, riflettono una particolare struttura (albero, grafo, relazione ecc.).

Il modello relazionale

- Il modello relazionale prevede un costrutto di *relazione* che permette di organizzare i dati in insiemi di *record* (tuple) a struttura *fissa*; la rappresentazione più usata di tale costrutto è quella di tabella:
  - le righe corrispondono ai record;
  - le colonne corrispondono agli attributi o campi dei record.
- Esempio: Programma

  Insegnamenti

Corso	Docente
Algebra	Mario Rossi
Geometria	Paolo Bianchi
Fisica	Guido Verdi

Laurea	Materia	Anno
Matematica	Fisica	I
Informatica	Algebra	I
Ingegneria civile	Fisica	II
Matematica	Geometria	I

Schemi e istanze di un Database

- La parte di un database che è *invariante* rispetto al tempo è detta *schema* e descrive le caratteristiche fondamentali dei dati.
- La parte di un database che varia nel tempo è detta *istanza* o *stato* ed è costituita dai dati veri e propri.
- Nell'esempio precedente, gli schemi delle due relazioni si indicano come segue:

  `Insegnamenti(Corso,Docente)`

  `Programma(Laurea,Materia,Anno)`

  Le istanze delle due relazioni sono costituite dall'insieme delle righe delle rispettive tabelle.
- Lo schema di un database è anche detto *componente intensionale*, mentre l'istanza è detta anche *componente estensionale*.

Esempio: orario ferroviario

NUMERO	PROVENIENZA	ORA_DI_ARRIVO	RITARDO	BINARIO
83	ROMA	7,40		3
84	MILANO	9,20		8
109	ROMA	10,45	15'	6
213	NAPOLI	12,20		6
38	FIRENZE	9,15		5
214	ROMA	11,30		4

Schema dell'orario

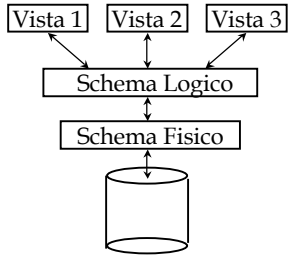
Nome campo	Tipo dati	Descrizione
NUMERO	Testo	
PROVENIENZA	Testo	
ORA_DI_ARRIVO	Testo	
RITARDO	Testo	
BINARIO	Testo	

Livelli di astrazione nei DBMS

- L'architettura di un DBMS può essere suddivisa in tre strati:
  - Schema *logico* (**relazionale**, gerarchico, a grafo, a oggetti).
  - Schema *interno*: descrizione dell'implementazione del database in termini delle strutture fisiche di memorizzazione.
  - Schema *esterno*: descrizione per mezzo dello schema logico di una parte del database, in modo da riflettere il punto di vista di un particolare utente o gruppo di utenti (per questo si chiama anche *view*). Ad esempio ad uno studente di CBC interesseranno soltanto le righe della tabella Programma in cui l'attributo Laurea assume il valore "Conservazione dei Beni Culturali".
- Note:
  - Per ogni database, possono esistere diverse view (schemi esterni).
  - Un meccanismo di autorizzazioni può essere usato per regolare l'accesso alle view appropriate per ogni categoria di utenti.
  - L'interazione con i dati avviene solamente per mezzo dello schema esterno (che può coincidere con quello logico).

Livelli di astrazione

- Molte viste, un singolo schema logico e uno schema fisico
  - Le viste descrivono i dati come vengono visti dagli utenti
  - Lo schema logico definisce la struttura logica
  - Lo schema fisico descrive i file e gli indici usati



Indipendenza dei dati

- I vari livelli di astrazione dei DBMS permettono a programmi ed utenti di interagire con i dati senza preoccuparsi dei dettagli dell'implementazione.
- Vi sono due livelli di indipendenza:
  - L'indipendenza fisica permette di modificare le strutture (e.g., file su disco) che implementano le relazioni senza influenzare la descrizione ad alto livello dei dati ed il loro utilizzo da parte di programmi ed utenti.
  - L'indipendenza logica permette di modificare le view (schemi esterni) senza influenzare lo schema logico e, di conseguenze, quello interno.
- L'indipendenza dei dati è raggiunta per mezzo dell'interazione tramite lo schema esterno, in quanto il DBMS si preoccupa di tradurre le varie operazioni in termini dei livelli sottostanti.

Linguaggi orientati ai database

- In corrispondenza alla distinzione fra schemi ed istanze di database esistono due tipi di linguaggi:
  - Data Definition Language (DDL): linguaggi che consentono di definire gli schemi logici, interni ed esterni di un database con i relativi meccanismi di autorizzazione degli accessi (a volte quest'ultima funzionalità viene espressa come DCL).
  - Data Manipulation Language (DML): linguaggi che consentono di interrogare e modificare le istanze dei database.
- Esistono linguaggi che offrono le funzionalità proprie sia dei DDL, sia dei DML. Il più noto e diffuso è SQL (Structured Query Language).
- Nota: questi NON sono linguaggi di programmazione!

Accesso ai DBMS

- L'accesso ai dati può avvenire tramite:
  - Linguaggi interattivi testuali (e.g., SQL).
  - Comandi appositi richiamabili dai tradizionali linguaggi di programmazione (C, C++, Java ecc.)
  - Comandi di linguaggi ad hoc realizzati per svolgere compiti specifici (e.g. stampa di resoconti, generazione di grafi ecc.).
  - Strumenti interattivi visuali.

Linguaggi per basi di dati

- Un altro contributo all'efficacia: disponibilità di vari linguaggi e interfacce
  - ⇨ linguaggi testuali interattivi (**SQL**)
  - ⇨ comandi (SQL) immersi in un linguaggio ospite (Pascal, Java, C ...)
  - ⇨ comandi (SQL) immersi in un linguaggio ad hoc, con anche altre funzionalità (p.es. per grafici o stampe strutturate)
  - ⇨ con interfacce amichevoli (senza linguaggio testuale)

SQL, un linguaggio interattivo

- "Trovare i corsi tenuti in aule a piano terra"

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	Pincherle	Primo

SQL, un linguaggio interattivo

```
SELECT Corso, Aula, Piano
FROM Aule, Corsi
WHERE Nome = Aula
AND Piano = "Terra"
```

Corso	Aula	Piano
Sistemi	N3	Terra
Reti	N3	Terra

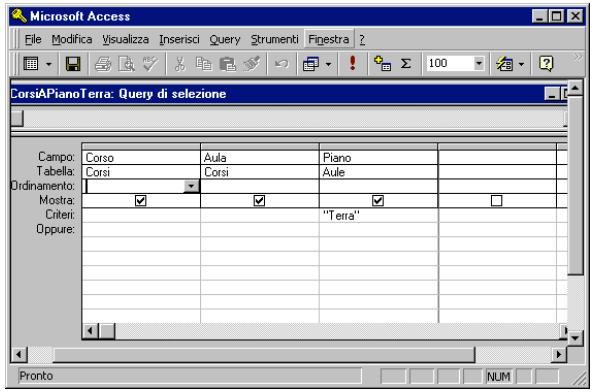
SQL immerso in linguaggio ospite

```
write('nome della citta"?'); readln(citta);
EXEC SQL DECLARE P CURSOR FOR
      SELECT NOME, REDDITO
      FROM PERSONE
      WHERE CITTA = :citta ;
EXEC SQL OPEN P ;
EXEC SQL FETCH P INTO :nome, :reddito ;
while SQLCODE = 0 do begin
    write('nome della persona:', nome, 'aumento?');
    readln(aumento);
    EXEC SQL UPDATE PERSONE
              SET REDDITO = REDDITO + :aumento
                WHERE CURRENT OF P
    EXEC SQL FETCH P INTO :nome, :reddito
  end;
EXEC SQL CLOSE CURSOR P
```


SQL in linguaggio ad hoc (Oracle PL/SQL)

```
declare  Stip number;
begin
    select Stipendio into Stip from Impiegato
    where Matricola = '575488' for update of Stipendio;
    if Stip > 30 then
       update Impiegato set Stipendio = Stipendio * 1.1
              where Matricola = '575488';
    else
       update Impiegato set Stipendio = Stipendio * 1.15
              where Matricola = '575488';
    end if;
    commit;
  exception
    when no_data_found then
      insert into Errori values('Matricola inesistente',sysdate);
  end;
```


Interazione non testuale (Access)



Una distinzione terminologica (separazione fra dati e programmi)

data manipulation language (DML)

per l'interrogazione e l'aggiornamento di (istanze di) basi di dati

data definition language (DDL)

per la definizione di schemi (logici, esterni, fisici) e altre operazioni generali

Un'operazione DDL (sullo schema)

```
CREATE TABLE orario (
  insegnamento  CHAR(20) ,
  docente       CHAR(20) ,
  aula          CHAR(4)  ,
  ora           CHAR(5)  )
```


Vantaggi e svantaggi dei DBMS, 1

Pro

- dati come risorsa comune, base di dati come modello della realtà
- gestione centralizzata con possibilità di standardizzazione ed “economia di scala”
- disponibilità di servizi integrati
- riduzione di ridondanze e inconsistenze
- indipendenza dei dati (favorisce lo sviluppo e la manutenzione delle applicazioni)

Vantaggi e svantaggi dei DBMS, 2

Contro

- costo dei prodotti e della transizione verso di essi
- Necessità di acquisire le competenze di progettazione e amministrazione di un sistema complesso
- non scorporabilità delle funzionalità (con riduzione di efficienza)

Sommario

- Un DBMS è usato per mantenere e interrogare grandi insiemi di dati
- Tra i benefici, il ripristino dai crash del sistema, l’accesso concorrente, il rapido sviluppo di applicazioni, l’integrità dei dati e la sicurezza
- I livelli di astrazione portano all’indipendenza dei dati
- I DBA svolgono un lavoro di responsabilità e ben pagato!

Categorie di utenti

- **DataBase Administrator (DBA):** è il responsabile della progettazione, controllo ed amministrazione del database. Deve garantire i servizi associati al database, l'affidabilità e la gestione delle autorizzazioni d'accesso.
- **Progettisti** di applicazioni e **programmatori**: utilizzano DML e strumenti complementari per la generazione di interfacce al database.
- **Utenti**: utilizzano il database per i loro scopi; si suddividono in:
  - **End users**: sfruttano programmi speciali che implementano operazioni ripetitive e predefinite.
  - **Casual users**: usano i linguaggi interattivi per l'esecuzione di interrogazioni ed operazioni particolari ("casual" indica che tali interazioni non sono predefinite).

Vantaggi/svantaggi dei DBMS

- Vantaggi:
  - I DBMS consentono l'accesso dei soli utenti autorizzati alla risorsa centralizzata dei dati di un'organizzazione.
  - Il modello dei dati rappresenta uno standard che codifica le informazioni rilevanti per l'organizzazione, rendendone agevole l'integrazione in applicazioni esistenti e future.
  - Il controllo centralizzato dei dati riduce i costi, consentendo un'economia di scala.
  - La condivisione dei dati elimina il problema della ridondanza e della consistenza di questi ultimi.
  - I DBMS, tramite l'indipendenza dei dati, favoriscono lo sviluppo di applicazioni flessibili e facilmente modificabili.
- Svantaggi:
  - I DBMS sono prodotti costosi e complessi la cui introduzione in una realtà esistente comporta notevoli investimenti in termini di hardware, software ed addestramento di personale.
  - Spesso i DBMS forniscono un numero di servizi maggiore di quelli richiesti; la difficoltà di isolare soltanto quelli necessari comporta molte volte costi e perdita di efficienza dell'attività.

Il modello relazionale

- Il modello relazionale è stato introdotto nel 1970 da E.F. Codd.
- Soltanto a metà degli anni ottanta ha trovato una buona diffusione sul mercato, in quanto all'epoca della sua introduzione non esistevano soluzioni efficienti per implementare le strutture relazionali.
- Il modello relazionale si basa su due concetti chiave strettamente correlati:
  - La nozione matematica di relazione permette la formulazione e lo sviluppo di una teoria efficace a supporto del modello.
  - La nozione di tabella è semplice, supportando l'intuizione anche dei non "addetti ai lavori".
- Il successo del modello relazionale rispetto ad altri modelli (gerarchico, a grafo ecc.) è anche dovuto all'indipendenza dei dati e dell'interazione con essi dall'implementazione fisica delle relative strutture.

Relazioni

- Dati due insiemi D_1 e D_2, il loro *prodotto cartesiano* si indica con $D_1 \times D_2$ ed è l'insieme di tutte le possibili *coppie* il cui primo elemento appartiene a D_1 ed il cui secondo elemento appartiene a D_2.
- Una relazione su D_1 e D_2 (detti domini della relazione) è un sottoinsieme di $D_1 \times D_2$. Ad esempio se $D_1=\{1,2,4\}$ e $D_2=\{a,b\}$, allora $D_1 \times D_2=\{(1,a), (1,b), (2,a), (2,b), (4,a), (4,b)\}$. In questo caso un esempio di relazione su D_1 e D_2 è $\{(1,a), (1,b), (4,b)\}$.
- Le nozioni di prodotto cartesiano e relazione possono essere generalizzati al caso di n insiemi $D_1, D_2, \ldots, D_n$(non necessariamente distinti fra loro). In questo caso invece di coppie si avrà a che fare con n-tuple, i.e., elementi del tipo $(v_1, v_2, \ldots, v_n)$ dove v_1 appartiene a D_1 v_n a D_2, …, v_n a D_n.
- In teoria non vi sono vincoli sulla natura degli insiemi coinvolti. Tuttavia, siccome i computer possono memorizzare soltanto un numero finito di informazioni, si assume che le relazioni siano insiemi finiti. I domini invece possono essere infiniti in modo da poter assumere sempre l'esistenza di un valore non contenuto nel database.

Relazioni e tabelle

- Dati gli insiemi D_1={1,2,4} e D_2={a,b}, il prodotto cartesiano $D_1 \times D_2$={(1,a), (1,b), (2,a), (2,b), (4,a), (4,b)} e la relazione {(1,a), (1,b), (4,b)} ammettono la seguente rappresentazione sotto forma di tabelle:

1	a
1	b
2	a
2	b
4	a
4	b

1	a
1	b
4	b

- Dato il prodotto cartesiano $D_1 \times D_2 \times \ldots \times D_n$ il numero n è il grado del prodotto e delle relazioni sugli n domini. Il numero di n-tuple di una relazione costituisce invece la cardinalità di quest'ultima.

Relazioni con attributi

- Data una relazione, non ha importanza l'ordine in cui sono elencate le n-tuple che la compongono non è importante (l'ordine delle righe nelle tabelle è soltanto ai fini della presentazione). Si noti inoltre che ogni n-tupla è *unica*.
- Una n-tupla (v_1, v_2, …, v_n) stabilisce delle connessioni fra i dati che contiene; tali legami sono intrinsecamente dipendenti dalla posizione dei dati all'interno della n-tupla (notazione *posizionale*).
- Infatti, cambiando l'ordine dei componenti di una n-tupla, cambia anche l'interpretazione delle connessioni fra i dati (ogni posizione corrisponde ad un certo dominio).

Relazioni con attributi

- Per evitare i problemi connessi all'ordinamento dei componenti delle n-tuple, si definisce una notazione non posizionale che distingue i vari domini (e quindi i componenti) assegnando loro dei nomi univoci.
- Nella visualizzazione per mezzo di tabelle gli attributi diventano le intestazioni delle colonne:

HomeTeam	VisitorsTeam	HomeGoals	VisitorsGoals
Real Madrid	Liverpool	3	1
Liverpool	Milan	2	0
Real Madrid	Roma	1	2

Relazioni e database

- Solitamente un database è composto da più relazioni con valori comuni laddove vi siano delle corrispondenze fra di esse.
- Esempio:

Studenti

Matricola	Cognome	Nome	DataDiNascita
276545	Rossi	Mario	25/11/1980
485745	Verdi	Luisa	23/04/1981
200768	Gialli	Luigi	12/02/1981
587614	Rosa	Maria	10/10/1980

Esami

Studente	Voto	Corso
276545	24	01
276545	27	04
587614	30	01
200768	28	04

Corsi

Codice	Titolo	Docente
01	Fisica	Tizio
03	Chimica	Caio
04	Algebra	Sempronio

Relazioni e database

- Nell'esempio precedente la prima relazione contiene dei dati su un insieme di studenti (n. di matricola, cognome, nome, data di nascita).
- La terza relazione contiene dei dati su un insieme di corsi (codice, titolo e docente).
- La seconda relazione contiene dei dati (n. di matricola, voto e codice del corso) relativi agli esami sostenuti dagli studenti. Tramite il primo ed il terzo attributo vengono stabilite due corrispondenze con la prima e la terza relazione rispettivamente.
- Siccome le corrispondenze tra n-tuple appartenenti a relazioni diverse sono fondate sui valori di certi attributi, si dice che il modello relazionale è **value-based**, ossia, basato sui valori.

Informazioni incomplete e valori NULL

- Il modello relazionale impone una certa rigidità in quanto l'informazione deve essere sempre rappresentata da tuple di valori omogenei.
- Spesso tuttavia capita che i dati disponibili non corrispondano esattamente allo schema di relazione in quanti manca un dato.
- L'utilizzo di particolari valori del dominio associato ad un attributo per denotare che il valore corrispondente manca non è una buona idea in quanto può generare confusione.
- Il concetto di relazione viene quindi esteso in modo da permettere ad ogni tupla di assumere per ogni attributo o un valore del dominio corrispondente oppure un valore speciale detto *null value* (indicato dalla stringa NULL).
- Il significato del null value è quello dell'assenza di informazione, i.e., il valore potrebbe non esistere affatto od essere semplicemente sconosciuto.

Notazione

- Uno schema di relazione consiste in un simbolo R (nome della relazione) ed in un insieme di nomi di attributi $X=\{A_1, A_2, \ldots, A_n\}$. Complessivamente uno schema di relazione è denotato dall'espressione R(X).
- Uno schema di database consiste in un insieme di schemi di relazioni con nomi distinti e si denota con l'espressione $\mathbf{R}=\{R_1(X_1), R_2(X_2), \ldots, R_m(X_m)\}$.
- Un'istanza di relazione (o semplicemente una relazione) sullo schema R(X) è un insieme r di tuple su X.
- Un'istanza di database (o semplicemente un database) sullo schema $\mathbf{R}=\{R_1(X_1), R_2(X_2), \ldots, R_m(X_m)\}$ è un insieme di relazioni $\mathbf{r}=\{r_1, r_2, \ldots, r_m\}$ tale che ogni r_i è una relazione sullo schema $R_i(X_i)$ per $1 \le i \le m$.
- Esempio:

  R={Studenti(Matricola,Cognome,Nome,DataDiNascita),
  Esami(Studente,Voto,Corso), Corsi(Codice,Titolo,Docente)
  }

Correttezza dell'informazione

- Nonostante il modello relazionale permetta di rappresentare ed organizzare l'informazione in modo conveniente per le applicazioni, non è detto che ogni insieme di tuple rappresenti dell'informazione *corretta* rispetto ad un'applicazione.
- Esempio:

Studenti

Matricola	Cognome	Nome	DataDiNascita
276545	Rossi	Mario	25/11/1980
276545	Verdi	Luisa	23/04/1981
200768	Gialli	Luigi	12/02/1981
587614	Rosa	Maria	10/10/1980

Esami

Studente	Voto	Lode	Corso
276545	30	No	**07**
276545	27	**Si**	04
587614	30	No	01
300000	28	No	04

Corsi

Codice	Titolo	Docente
01	Fisica	Tizio
03	Chimica	Caio
04	Chimica	Sempronio

Vincoli di integrità

- Per evitare situazioni in cui le tuple delle relazioni di un database non rappresentano informazioni corrette, viene introdotto il concetto di *vincolo di integrità*.
- Ogni vincolo è un predicato che associa il valore vero o il valore falso ad ogni istanza di database a seconda che quest'ultimo soddisfi o meno il vincolo stesso.
- I vincoli di integrità sono classificabili come segue:
  - *Vincoli intra-relazionali* (ogni vincolo di questo tipo coinvolge una sola relazione del database); si possono suddividere ulteriormente in:
    - *Vincoli di tupla* (ogni vincolo va valutato su ogni tupla presa singolarmente).
    - *Vincoli di valore o di dominio* (impongono delle restrizioni sui domini degli attributi): coinvolgono un singolo attributo.
  - *Vincoli inter-relazionali* (ogni vincolo di questo tipo coinvolge più relazioni del database).

Vincoli di tupla

- I vincoli sui valori che una singola tupla può assumere vengono indicati con una sintassi che contempla gli operatori logici AND, OR, NOT e degli atomi che specificano dei confronti fra espressioni costruite a partire dai valori degli attributi.
- Ad esempio i vincoli sulla relazione Corsi possono essere specificati come segue:
  (Voto≥0) AND (Voto≤30)
  (NOT(Lode='Si')) OR (Voto=30)
  Il primo vincolo è di fatto un vincolo di dominio in quanto coinvolge soltanto l'attributo Voto.
- Si possono specificare anche vincoli più complessi.
  Ad esempio dato lo schema di relazione
  Pagamenti(Data,Imponibile,Tasse,ValoreNetto)
  si può specificare il vincolo di integrità
  ValoreNetto=Imponibile-Tasse

Chiavi

- Intuitivamente una chiave è un insieme di attributi che consente di individuare in modo univoco le tuple di una relazione.
- Formalmente si hanno le seguenti definizioni:
  - Un insieme di attributi K è una *superchiave* per la relazione r se quest'ultima non contiene due tuple distinte t_1 e t_2 tali che $t_1[K]=t_2[K]$.
  - Un insieme di attributi K è una *chiave* per la relazione r se K è la superchiave minimale per r (i.e., se non esiste un'altra superchiave K' per r contenuta in K).
- Il vincolo intra-relazionale più importante è quello di chiave che si specifica elencando gli attributi che costituiscono la chiave stessa. Ad esempio per la relazione
  Studenti(Matricola,Cognome,Nome,DataDiNascita)
  i vincoli di chiave possibili sono i seguenti (assumendo che non possano esistere due persone con lo stesso nome, lo stesso cognome e la stessa data di nascita):
  Matricola
  Cognome, Nome, DataDiNascita

Chiavi

- Ogni relazione ha sempre una chiave; infatti ogni tupla di una relazione r(X) è unica, quindi l'insieme di attributi X è una superchiave per essa. A questo punto si hanno due casi:
  - La superchiave è anche la chiave per la relazione r(X).
  - Esiste un'altra superchiave K' contenuta in K: si ripete il ragionamento rispetto a K' fino ad individuare una superchiave minimale. Tale procedimento termina sicuramente visto che gli attributi di una relazione sono in numero finito.
- L'esistenza di una chiave per ogni relazione ha le seguenti importanti conseguenze:
  - È garantito l'accesso in modo non ambiguo a tutti i valori del database.
  - È possibile avere delle connessioni fra dati contenuti in relazioni distinte.

Chiave primaria

- Nel caso in cui i valori di una chiave siano nulli (NULL), non è più possibile riferirsi in modo univoco alla tupla corrispondente. Inoltre non è più possibile stabilire delle corrispondenze con tuple di altre relazioni.
- Per evitare questi effetti indesiderati si stabilisce che i valori NULL non possano occorrere in una delle chiavi di una relazione, detta *chiave primaria* (mentre possono eventualmente occorrere nelle rimanenti chiavi).
- Negli schemi di relazione e nelle rappresentazioni sotto forma di tabella le chiavi primarie vengono evidenziate sottolineando gli attributi che le compongono (ad esempio Studenti(Matricola,Cognome,Nome,DataDiNascita)).
- Le connessioni fra tuple di relazioni distinte sono quindi solitamente realizzate per mezzo delle chiavi primarie.
- Se non è possibile trovare un insieme di attributi per la chiave primaria, si introduce un attributo apposito con la funzione di codice che viene generato in modo univoco al momento dell'inserzione delle tuple.

Vincoli referenziali

- I vincoli referenziali (foreign key) sono il tipo di vincolo inter-relazionale più importante.

Agenti

NumReg	Cognome	Nome
567	Brun	Jean
456	Larue	Henri
638	Larue	Jacques

Contravvenzioni

Codice	Data	Agente	Dipartimento	Registrazione
143256	25/10/92	567	75	5694 FR
987554	26/10/92	456	75	5694 FR
987557	26/10/92	456	75	6544 XY
630876	15/10/92	456	47	6544 XY
539856	12/10/92	567	47	6544 XY

Veicoli

Registrazione	Dipartimento	Proprietario	Indirizzo
6544 XY	75	Cordon Eduard	Rue du Pont
7122 HT	75	Cordon Eduard	Rue du Pont
5694 FR	75	Latour Hortense	Avenue Foch
6544 XY	47	Mimault Bernard	Avenue FDR

Vincoli referenziali

- Un vincolo referenziale permette di stabilire delle corrispondenze fra tuple appartenenti a due relazioni distinte specificando un insieme di attributi X della prima in modo tale che
  - i valori in X di ogni tupla della prima relazione appaiano come valori della chiave primaria di una tupla della seconda relazione.
- Nell'esempio precedente vi sono due vincoli referenziali:
  - Il primo vincolo è fra l'attributo Agente della relazione Contravvenzioni e l'attributo NumReg della relazione Agenti.
  - Il secondo vincolo è fra gli attributi Registrazione e Dipartimento della relazione Contravvenzioni e gli attributi con gli stessi nomi della relazione Veicoli.
- I vincoli referenziali sono diretti verso la chiave primaria della seconda relazione

Vincoli referenziali

- Indicando con $X=A_1A_2\ldots A_p$ un insieme di attributi di R_1, con $K=B_1B_2\ldots B_p$ gli attributi che compongono la chiave primaria di R_2 si ha che un vincolo referenziale fra gli attributi X di R_1 e R_2 è soddisfatto se
  - per ogni tupla t_1 di R_1 che non contenga valori nulli (NULL) in corrispondenza degli attributi X, esiste una tupla t_2 di R_2 tale che $t_1[A_i]=t_2[B_i]$ per $1 \le i \le p$.
- L'ordine in cui vengono specificati gli attributi di X e K è importante. Infatti supponendo di avere una relazione con schema

  Incidenti(Codice, Dipartimento1, Registrazione1, Dipartimento2, Registrazione2)

  per stabilire un vincolo referenziale con la relazione Veicoli bisogna necessariamente usare l'ordinamento (dato che i nomi degli attributi in gioco non coincidono con quelli della chiave primaria di Veicoli).

SQL

- originariamente "Structured Query Language", ora "nome proprio"
- linguaggio con varie funzionalità:
  - contiene sia il DDL sia il DML
- ne esistono varie versioni
- vediamo gli aspetti essenziali, non i dettagli

SQL: "storia"

- prima proposta SEQUEL (1974);
- prime implementazioni in SQL/DS e Oracle (1981)
- dal 1983 ca. "standard di fatto"
- standard “de jure“ ANSI
  - 1986, esteso con integrità referenziali nel 1989 (SQL-89)
  - 1992: molte funzionalità (SQL-2)
  - 1999: trigger, oggetti, viste ricorsive ... (SQL:1999, o SQL-3)
- Il più diffuso: SQL-2, ma recepito solo in parte (!!)

Definizione dei dati in SQL

- Istruzione CREATE TABLE:
  - definisce uno schema di relazione e ne crea un’istanza vuota
  - specifica attributi, domini e vincoli

CREATE TABLE, esempio

```
CREATE TABLE Impiegato (
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Dipart CHAR(15),
  Stipendio NUMERIC(9) DEFAULT 0,
  FOREIGN KEY (Dipart) REFERENCES
       Dipartimento(NomeDip),
  UNIQUE (Cognome,Nome)
)
```

Domini elementari

- Carattere: singoli caratteri o stringhe, anche di lunghezza variabile
- Bit: singoli booleani o stringhe
- Numerici, esatti e approssimati
- Data, ora, intervalli di tempo
- Introdotti in SQL:1999:
  - Boolean
  - BLOB, CLOB (binary/character large object): per grandi immagini e testi

Vincoli intrarelazionali

- NOT NULL
- UNIQUE definisce chiavi
- PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
- CHECK, vedremo più avanti

UNIQUE e PRIMARY KEY

- due forme:
  - nella definzione di un attributo, se forma da solo la chiave
  - come elemento separato

CREATE TABLE, esempio

```
CREATE TABLE Impiegato(
   Matricola CHAR(6) PRIMARY KEY,
   Nome CHAR(20) NOT NULL,
   Cognome CHAR(20) NOT NULL,
   Dipart CHAR(15),
   Stipendio NUMERIC(9) DEFAULT 0,
   FOREIGN KEY(Dipart) REFERENCES
         Dipartimento(NomeDip),
   UNIQUE (Cognome,Nome)
 )
```

PRIMARY KEY, alternative

```
Matricola CHAR(6) PRIMARY KEY

Matricola CHAR(6),
…,
PRIMARY KEY (Matricola)
```


CREATE TABLE, esempio

```
CREATE TABLE Impiegato(
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Dipart CHAR(15),
  Stipendio NUMERIC(9) DEFAULT 0,
  FOREIGN KEY(Dipart) REFERENCES
       Dipartimento(NomeDip),
  UNIQUE (Cognome,Nome)
 )
```


Chiavi su più attributi, attenzione

```
Nome          CHAR(20) NOT NULL,
Cognome   CHAR(20) NOT NULL,
UNIQUE (Cognome,Nome),

Nome          CHAR(20) NOT NULL UNIQUE,
Cognome   CHAR(20) NOT NULL UNIQUE,
```

- Non è la stessa cosa!

Vincoli interrelazionali

- CHECK, vedremo più avanti
- REFERENCES e FOREIGN KEY permettono di definire vincoli di integrità referenziale
- di nuovo due sintassi
  - per singoli attributi
  - su più attributi
- E' possibile definire politiche di reazione alla violazione

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

Prov	Numero	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

CREATE TABLE, esempio

```
CREATE TABLE Infrazioni(
  Codice CHAR(6) NOT NULL PRIMARY KEY,
  Data DATE NOT NULL,
  Vigile INTEGER NOT NULL
            REFERENCES Vigili(Matricola),
  Provincia CHAR(2),
  Numero CHAR(6) ,
  FOREIGN KEY(Provincia, Numero)
            REFERENCES Auto(Provincia, Numero)
)
```


Modifiche degli schemi

ALTER DOMAIN
ALTER TABLE
DROP DOMAIN
DROP TABLE
...

Definizione degli indici

- è rilevante dal punto di vista delle prestazioni
- ma è a livello fisico e non logico
- in passato era importante perché in alcuni sistemi era l'unico mezzo per definire chiavi
- CREATE INDEX

DDL, in pratica

- In molti sistemi si utilizzano strumenti diversi dal codice SQL per definire lo schema della base di dati

SQL, operazioni sui dati

- interrogazione:
  - SELECT
- modifica:
  - INSERT, DELETE, UPDATE

Istanze di esempio

- Useremo nei nostri esempi queste istanze delle relazioni Velisti e Prenota
- Se la chiave per la relazione Prenota contenesse solo gli attributi *vid* e *bid*, in che modo la semantica sarebbe diversa?

P1

vid	bid	giorno
22	101	10/10/96
58	103	11/12/96

V1

vid	vnome	espe-rienza	età
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

V2

vid	vnome	espe-rienza	età
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Interrogazioni SQL di base

```
SELECT [DISTINCT]   lista-attributi
FROM                lista-relazioni
[WHERE              qualificazioni]
```

- *Lista-relazioni*. Una lista di nomi di relazioni (eventualmente con una *variabile di range* dopo ciascun nome)
- *Lista-attributi*. Una lista di attributi delle relazioni in *lista-relazioni*
- *Qualificazioni*. Confronti (attr *op* cost oppure attr1 *op* attr2, dove *op* è uno tra <, >, =, ≤, ≥, ≠) combinati usando AND, OR e NOT
- DISTINCT è una parola chiave opzionale che indica che la risposta non dovrebbe contenere duplicati. Per impostazione predefinita i duplicati *non* sono eliminati!

Strategia di valutazione concettuale

- La semantica di una interrogazione SQL è definita in termini della seguente strategia di valutazione concettuale:
  - calcolare il prodotto scalare di *lista-relazioni*
  - scartare le tuple risultanti se non passano le *qualificazioni*
  - cancellare gli attributi che non sono in *lista-attributi*
  - se è specificato DISTINCT, eliminare le righe duplicate
- Questa strategia è probabilmente il modo meno efficiente di calcolare una interrogazione! Un ottimizzatore troverà strategie più efficienti per calcolare *le stesse risposte*.

Esempio di valutazione concettuale

```
SELECT    V.vnome
FROM      Velisti V, Prenota P
WHERE     V.vid = P.vid AND P.bid = 103
```

(vid)	vnome	espe-rienza	età	(vid)	bid	giorno
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Trovare gli id dei velisti che hanno prenotato almeno una barca

```
SELECT    V.vid
FROM      Velisti V, Prenota P
WHERE     V.vid = P.vid
```

- Farebbe differenza aggiungere DISTINCT a questa interrogazione?
- Qual è l'effetto della sostituzione di V.vid con V.vnome nella clausola SELECT? Farebbe differenza aggiungere DISTINCT a questa variante dell'interrogazione?

Espressioni e stringhe

```
SELECT    V.età, età1 = V.età -5, 2 * V.età AS eta2
FROM      Velisti V
WHERE     V.vnome LIKE 'B_%B'
```

- Illustra l'uso delle espressioni aritmetiche e del pattern matching di stringhe: trovare le triple (le età dei velisti e due campi definiti da espressioni) per quei velisti il cui nome inizia e termina con B e contiene almeno tre caratteri)
- AS e = sono due modi di dare un nome ai campi del risultato
- LIKE è usato per il matching di stringhe. "_" indica qualunque carattere singolo e "%" sta per 0 o più caratteri arbitrari

Trovare i vid dei velisti che hanno prenotato una barca rossa o una barca verde

- UNION: può essere usato per calcolare l'unione di qualunque coppia di insiemi di tuple (essi stessi risultati di interrogazioni SQL) *compatibili rispetto all'unione*
- Se nella prima versione sostituiamo OR con AND, cosa otteniamo?
- Disponibile anche: EXCEPT (cosa otteniamo se sostituiamo UNION con EXCEPT?)

```
SELECT  P.sid
FROM  Barche B, Prenota R
WHERE  P.bid=B.bid
  AND (B.colore='rosso' OR B.colore='verde')
```

```
SELECT   P.vid
FROM     Barche B, Prenota P
WHERE    P.bid = B.bid
      AND (B.colore = 'rosso' OR B.colore =
'verde')
UNION
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                    AND B.colore = 'verde'
```


Trovare i vid dei velisti che hanno prenotato una barca rossa e una barca verde

- INTERSECT: può essere usato per calcolare l'intersezione di qualunque coppia di insiemi di tuple *compatibili rispetto all'unione*
- Incluso nello standard SQL/92, ma alcuni sistemi non lo supportano
- Confrontate la simmetria delle interrogazioni con UNION e INTERSECT con la diversità delle altre versioni

```
SELECT    V.vid
FROM      Velisti V, Barche B1, Prenota P1,
                    Barche B2, Prenota P2
WHERE     V.vid = P1.vid AND P1.bid = B1.bid
          AND V.vid = P2.vid AND P2.bid = B2.bid
          AND (B1.colore = 'rosso' AND B2.colore = 'verde')
```

Campo chiave!

```
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                    AND B.colore = 'rosso'
INTERSECT
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                    AND B.colore = 'verde'
```


Design di un database

- Progettare un database implica definire quanto i seguenti aspetti:
  - Struttura
  - Caratteristiche
  - Contenuti
- Il ciclo di design di un database si suddivide in tre fasi principali:
  - progettazione concettuale
  - progettazione logica
  - progettazione fisica
- A sua volta la progettazione di un database è soltanto una delle fasi che costituiscono il ciclo di vita di un sistema informativo:
  - Studio di fattibilità
  - Raccolta ed analisi dei requisiti
  - Progettazione (design)
  - Implementazione
  - Validazione e testing
  - Funzionamento (operatività)

Metodologia di design

- Una buona metodologia di progettazione di un database deve prevedere quanto segue:
  - una decomposizione dell'attività complessiva di design in una successione di fasi indipendenti l'una dall'altra;
  - delle strategie da seguire nelle varie fasi di progettazione e dei criteri che consentano di effettuare una scelta nel caso in cui vi siano più alternative valide da seguire;
  - dei modelli di riferimento per descrivere input ed output (i.e., prerequisiti e risultati) delle varie fasi.
- Le proprietà che una buona metodologia deve garantire sono le seguenti:
  - generalità rispetto al sistema ed alle applicazioni in uso,
  - qualità del prodotto (accuratezza, completezza ed efficienza),
  - facilità di utilizzo delle strategie e dei modelli di riferimento.

Metodologia di design

- Nel campo di ricerca sui database si è consolidata una metodologia che soddisfa tutte le proprietà precedentemente descritte.
- Il principio su cui si basa è effettuare una chiara divisione fra le decisioni che riguardano *cosa* rappresentare e quelle che riguardano *come* farlo.

Fasi di progetto

- Progettazione concettuale: il modello di dati utilizzato è quello *concettuale*, che permette di descrivere l'organizzazione dei dati ad un alto livello di astrazione, producendo come risultato uno schema concettuale.
- Progettazione logica: lo schema concettuale viene tradotto in uno schema logico, adottando un *modello di dati logico* che sia supportato dal DBMS scelto.
- Progettazione fisica: lo schema logico viene completato con tutti i dettagli necessari per l'implementazione fisica sul DBMS scelto. Il risultato è uno schema fisico dipendente da un *modello fisico* strettamente dipendente dalle caratteristiche specifiche (e.g. strutture per l'organizzazione dei dati) del DBMS scelto.

Requisiti

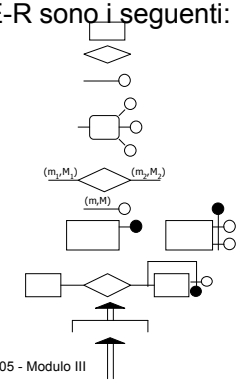
- I requisiti possono essere suddivisi in
  - Requisiti dei dati (riguardano il contenuto del database)
  - Requisiti operazionali (riguardano l'uso del database da parte di utenti e programmi)
- Per quanto riguarda la fase di progettazione concettuale ciò che conta sono i requisiti dei dati, mentre i requisiti operazionali sono utili soltanto per verificare che lo schema concettuale prodotto sia completo.
- Nella fase di progettazione logica lo schema concettuale ricevuto in input rappresenta i requisiti dei dati, mentre i requisiti operazionali sono utilizzati per costruire lo schema logico in modo da garantire un'esecuzione efficiente delle operazioni stesse.
- Nella fase di progettazione fisica lo schema logico ricevuto in input ed i requisiti operazionali sono utilizzati per ottimizzare l'efficienza del sistema informativo in base alle caratteristiche del particolare DBMS scelto.

Il modello Entità-Relazione

- Il modello Entità-Relazione (E-R) è un modello di dati concettuale.
- Fornisce dei costrutti per descrivere i requisiti dei dati di un'applicazione in modo indipendente da come vengano organizzati e gestiti i dati in un sistema.
- Ad ogni costrutto è associata una rappresentazione grafica.
- In questo modo è possibile rappresentare tramite dei diagrammi (ottenuti combinando i costrutti di base) i requisiti che i dati devono soddisfare.

Costrutti del modello E-R

- I costrutti del modello E-R sono i seguenti:
  - Entità
  - Relazione
  - Attributo semplice
  - Attributo composto
  - Cardinalità di una relazione
  - Cardinalità di un attributo
  - Identificatore interno
  - Identificatore esterno
  - Generalizzazione
  - Sottoinsieme



Entità

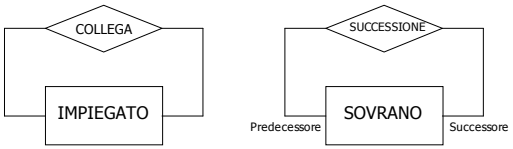
- Le entità rappresentano insiemi di oggetti (studenti, impiegati, città ecc.) che hanno delle proprietà comuni ed esistono autonomamente.
- Un'occorrenza di un'entità è un oggetto dell'insieme che quest'ultima rappresenta (e.g., la città di Roma è un'occorrenza dell'entità Città).
- Un'occorrenza di un'entità non è un valore che identifichi l'oggetto, ma l'oggetto stesso. Quindi un'occorrenza di un'entità esiste indipendentemente dalle sue proprietà.
- In uno schema ogni entità ha un nome univoco ed è rappresentata da un rettangolo contenente quest'ultimo:

Relazioni

- Le relazioni rappresentano corrispondenze logiche fra due o più entità.
- Un'occorrenza di una relazione che coinvolge n entità è una n-tupla in cui ogni elemento è un'occorrenza dell'entità corrispondente. Quindi un'occorrenza di una relazione binaria è una coppia di elementi di cui il primo è un'occorrenza della prima entità, mentre il secondo è un'occorrenza della seconda entità.
- Esempi di possibili relazioni sono la relazione RESIDENZA fra le entità CITTÀ e IMPIEGATO e la relazione ESAME fra le entità STUDENTE e CORSO.
- Graficamente una relazione si rappresenta con un rombo che racchiude il suo nome (univoco all'interno dello schema):

Relazioni ricorsive

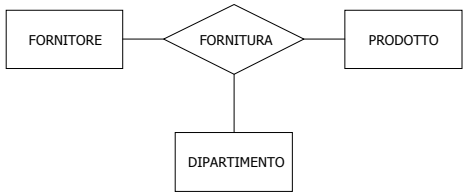
- Le relazioni ricorsive collegano un'entità a se stessa:



- Nel caso della seconda relazione (SUCCESSIONE), esiste un'asimmetria. Quindi si rende necessario distinguere i due ruoli che l'entità SOVRANO gioca, associando degli identificatori (Predecessore e Successore) alle linee che escono dalla relazione ricorsiva.

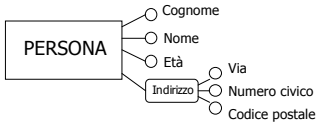
Relazioni fra più di due entità

- Un esempio di relazione fra più di due entità è quella tale che ogni sua occorrenza descrive il fatto che un fornitore fornisce un dato prodotto ad un dipartimento:

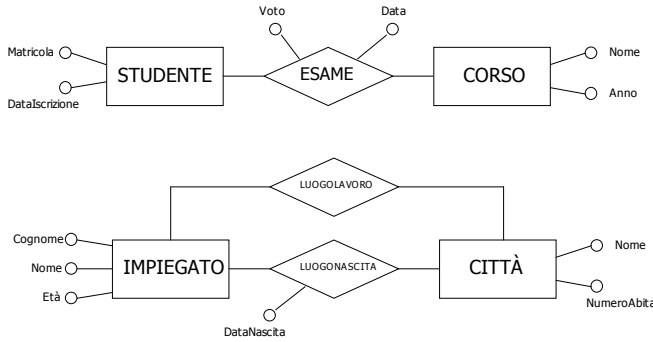


Attributi

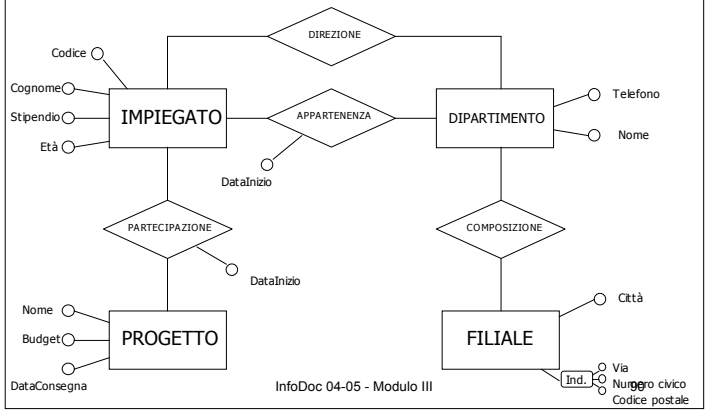
- La funzione degli attributi è quella di descrivere le proprietà elementari di entità e relazioni.
- Un attributo associa ad ogni occorrenza dell'entità o relazione su cui è definito un valore appartenente ad un insieme (il dominio dell'attributo).
- A volte può essere conveniente raggruppare diversi attributi della stessa entità o relazione, se sono strettamente correlati. In questo caso si ottiene un attributo composto.



Esempi



Esempio



Cardinalità delle relazioni

- Data una relazione è possibile specificare una cardinalità minima ed una massima per ognuna delle entità coinvolte.
- La cardinalità minima specifica il minimo numero di occorrenze della relazione a cui deve partecipare ogni istanza dell'entità considerata.
- Viceversa, la cardinalità massima specifica il massimo numero di occorrenze della relazione a cui può partecipare ogni istanza dell'entità considerata.
- Esempio:

Cardinalità delle relazioni

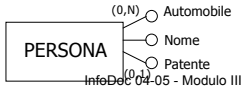
- In generale si può specificare qualsiasi numero intero non negativo come cardinalità.
- L'unico vincolo da rispettare è che la cardinalità minima sia minore od uguale alla cardinalità massima.
- Tuttavia, nella maggior parte dei casi sono sufficienti tre valori: zero, uno o N (detto anche "many", i.e., "molti") che indica un intero generico maggiore di uno.
- Per la cardinalità minima si ha quanto segue:
  - Se vale zero, si dice che la partecipazione alla relazione dell'entità relativa è opzionale.
  - Se vale uno, si dice che la partecipazione alla relazione dell'entità relativa è obbligatoria.
- Per la cardinalità massima si ha quanto segue:
  - Se vale uno, la partecipazione alla relazione dell'entità relativa può essere vista come una funzione (al più un'occorrenza della relazione può essere associata ad ogni occorrenza dell'entità).
  - Se vale N, ogni occorrenza dell'entità relativa è associata con un numero arbitrario di occorrenze della relazione.

Cardinalità delle relazioni

- Nel caso delle relazioni binarie (i.e., relazioni che coinvolgono due entità), si ha la seguente classificazione, ottenuta osservando le cardinalità massime:
  - Se entrambe le entità partecipano con cardinalità massima uguale a uno, la relazione è detta *uno-a-uno (one-to-one)*.
  - Se un'entità partecipa con cardinalità massima uguale a uno e l'altra con cardinalità massima uguale a N, la relazione è detta *uno-a-molti (one-to-many)*.
  - Se entrambe le entità partecipano con cardinalità massima uguale a N, la relazione è detta *molti-a-molti (many-to-many)*.

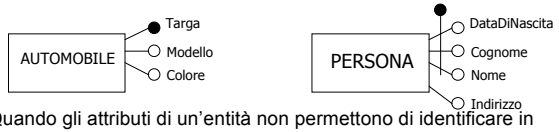
Cardinalità degli attributi

- Le cardinalità di un attributo specificano il minimo ed il massimo numero di valori dell'attributo che possono essere associati ad un'occorrenza dell'entità a cui si riferiscono.
- Se la cardinalità minima di un attributo è zero, esso viene detto opzionale.
- Se la cardinalità minima di un attributo è uno, esso viene detto obbligatorio.
- Se la cardinalità massima di un attributo è N, esso viene detto multivalore.
- Quando la cardinalità di un attributo è (1,1) viene omessa.
- Per quanto riguarda gli attributi multivalore, è opportuno notare che la situazione che rappresentano può essere modellata introducendo nuove entità e delle relazioni uno-a-molti o molti-a-molti.



Identificatori

- Ad ogni entità è possibile associare degli identificatori che permettono di distinguere senza ambiguità ogni singola occorrenza dell'entità stessa.
- Se l'identificatore è composto da uno o più attributi dell'entità a cui si riferisce, è detto *interno (o chiave)*.

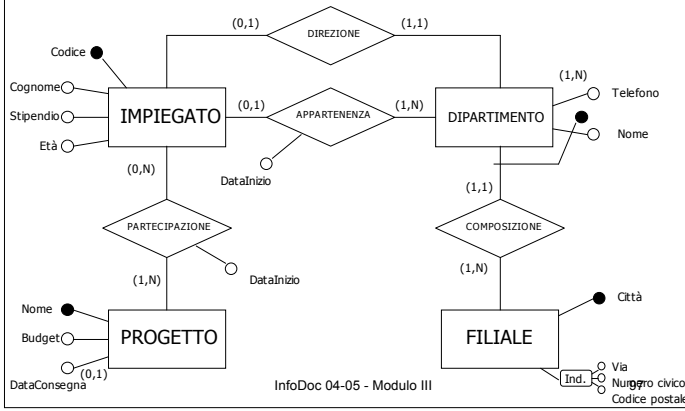


- Quando gli attributi di un'entità non permettono di identificare in modo univoco le sue occorrenze, si può ricorrere ad un attributo di un'altra entità collegata tramite una relazione uno-a-molti (identificatore esterno).

Identificatori

- Per quanto riguarda gli identificatori vale quanto segue:
  - Ogni identificatore può essere composto da uno o più attributi, se ognuno di questi ha cardinalità (1,1).
  - Un identificatore esterno può coinvolgere varie entità, se l'entità per cui viene definito partecipa con cardinalità (1,1) ad ogni relazione che coinvolge le entità esterne.
  - Un identificatore esterno può coinvolgere un'entità a sua volta identificata esternamente, a patto che non si generino dei cicli.
  - Ogni entità deve avere almeno un identificatore. Se ne ha più di uno, allora gli attributi e le entità coinvolte possono essere opzionali.

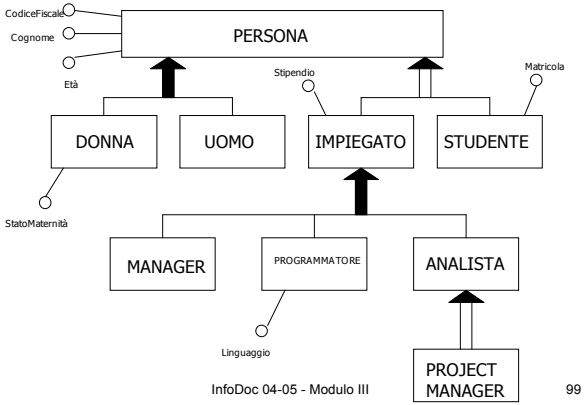
Esempio



Generalizzazioni

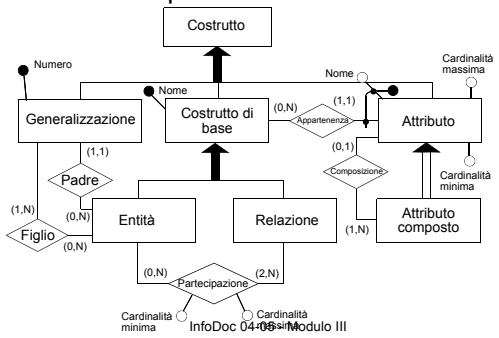
- Una generalizzazione lega un'entità padre E con delle entità figlie $E_1,\ldots,E_n$.
- E è detta generalizzazione di $E_1,\ldots,E_n$, mentre queste ultime sono dette specializzazioni di E.
- Tre le entità coinvolte in una generalizzazione valgono le seguenti proprietà:
  - Ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità padre.
  - Ogni proprietà (e.g., attributi identificatori ecc.) dell'entità padre è anche una proprietà delle entità figlie. Tale fatto viene detto *ereditarietà*.
- Una generalizzazione è detta *totale* se ogni occorrenza dell'entità padre è anche un'occorrenza di una delle entità figlie, altrimenti è detta *parziale*.
- Una generalizzazione è detta *esclusiva* se ogni occorrenza dell'entità padre è al più un'occorrenza di una delle entità figlie, altrimenti è detta *sovrapposta*.
- Quando una generalizzazione ha una sola entità figlia si parla di *sottoinsieme*.

Generalizzazioni



Riepilogo del modello E-R

- Il modello E-R è talmente generale da poter essere usato per descrivere se stesso:



Raccolta ed analisi dei requisiti

- Per raccolta dei requisiti di un'applicazione si intende:
  - L'individuazione dei problemi che l'applicazione deve risolvere.
  - La definizione delle caratteristiche dell'applicazione:
    - Aspetti statici (dati).
    - Aspetti dinamici (operazioni sui dati).
- La raccolta dei requisiti avviene generalmente in linguaggio naturale.
- Per analisi dei requisiti si intende invece la razionalizzazione e l'organizzazione dei requisiti raccolti.
- Le due attività sono fortemente connesse alimentandosi a vicenda.

Fonti di informazione

- La raccolta dei requisiti avviene attingendo da varie fonti di informazione:
  - Utenti dell'applicazione
  - Documentazione esistente
  - Eventuali versioni preesistenti dell'applicazione
- Nella fase di raccolta dei requisiti è fondamentale l'interazione con gli utenti del sistema informativo al fine di individuare gli aspetti essenziali, isolando i problemi marginali a raffinamenti e studi successivi.
- Siccome viene usato il linguaggio naturale, è importante analizzare profondamente i testi raccolti in modo da eliminare eventuali ambiguità ed inesattezze.

Esempio: società di formazione

Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei **partecipanti** ai corsi e dei **docenti**. Per i **partecipanti** (circa 5000), identificati da un codice, si vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, il **luogo** di nascita, il nome dei loro attuali datori di lavoro, i **posti** dove hanno lavorato in precedenza insieme al periodo, l'indirizzo ed il **numero di telefono**, i corsi che hanno frequentato (i corsi sono circa 200) e il **giudizio** finale. Rappresentiamo anche i **seminari** che stanno frequentando e, per ogni giorno, i **luoghi** e le ore dove sono tenute le lezioni. I corsi hanno un codice, un titolo e possono avere varie **edizioni** con date di inizio e fine e numero di partecipanti. Se gli **studenti** sono liberi professionisti, vogliamo conoscere l'area di interesse e, se lo possiedono, il **titolo**. **Per quelli che lavorano alle dipendenze di altri**, vogliamo conoscere invece il loro livello e la posizione ricoperta. Per gli **insegnanti** (circa 300) rappresentiamo il cognome, l'età, il posto dove sono nati, il nome del corso che insegnano, quelli che hanno insegnato nel passato e quelli che possono insegnare. Rappresentiamo anche tutti i loro **recapiti telefonici**. I **docenti** possono essere dipendenti interni della società o collaboratori esterni.

Regole generali

- Scegliere il livello giusto di astrazione: nell'esempio precedente "titolo" andrebbe specificato meglio come "titolo professionale" e "giudizio" come "votazione in decimi".
- Utilizzare una struttura standard delle frasi: lo stile deve essere uniforme; ad esempio "per <dato> rappresentiamo <insieme di proprietà>".
- Evitare frasi contorte: al posto di "a quelli che lavorano alle dipendenze di altri" è preferibile usare "lavoratori dipendenti".
- Individuare i sinonimi/omonimi e unificare i termini: nell'esempio precedente si usa sia il termine "docente" che il termine "insegnante". Inoltre "posto" è usato sia riferito a impiego, che a città, che ad aula.
- Esplicitare il riferimento tra termini: nell'esempio non è chiaro se "indirizzo" e "numero di telefono" si riferiscano ai partecipanti o ai loro datori di lavoro. Inoltre non è chiaro se l'espressione "Per quelli che lavorano" si rivolga ai partecipanti o ai docenti.
- Costruire un glossario dei termini: la definizione di un glossario con una descrizione, sinonimi e termini correlati logicamente per ogni termine usato nei requisiti può essere molto utile per agevolare la comprensione.

Esempio: società di formazione

Termine	Descrizione	Sinonimi	Collegamenti
Partecipante	Partecipante ai corsi. Può essere un dipendente o un professionista.	Studente	Corso, Datore
Docente	Docente dei corsi. Può essere un collaboratore esterno	Insegnante	Corso
Corso	Corso offerto. Può avere varie edizioni.	Seminario	Docente, Partecipante
Datore	Datori di lavoro attuali e passati dei partecipanti ai corsi.	Posto	Partecipante

Esempio: società di formazione

- Frasi di carattere generale:
  - Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti.
- Frasi relative ai partecipanti:
  - Per i partecipanti (circa 5000), identificati da un codice, si vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, la città di nascita, il nome dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato nel passato, con la relativa votazione in decimi.
- Frasi relative ai datori di lavoro:
  - Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono.

Esempio: società di formazione

- Frasi relative ai corsi:
  - Per i corsi (circa 200), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove sono tenute le lezioni.
- Frasi relative ai tipi specifici di partecipanti:
  - Per i partecipanti che sono liberi professionisti, rappresentiamo l'are di interesse e, se lo possiedono, il titolo professionale. Per i partecipanti che sono dipendenti, rappresentiamo invece il loro livello e la posizione ricoperta.
- Frasi relative ai docenti:
  - Per i docenti (circa 300), rappresentiamo il cognome, l'età, la città di nascita, tutti i numeri di telefono, il titolo del corso che insegnano, di quelli che hanno insegnato nel passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.

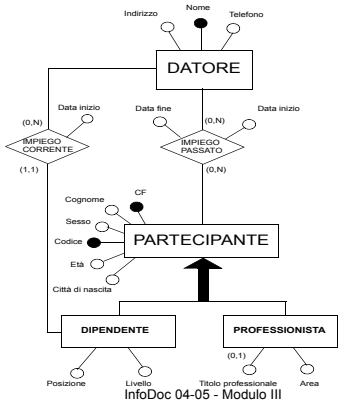
Qualità di uno schema

- Ogni schema concettuale deve garantire alcune proprietà generali che ne garantiscano la qualità:
  - Correttezza: i costrutti del modello usato sono utilizzati in modo appropriato, i.e., non ci sono errori sintattici (e.g., generalizzazioni fra relazioni invece che tra entità) o semantici (uso di una relazione per descrivere il fatto che un'entità è la specializzazione di un'altra entità).
  - Completezza: tutti i dati di interesse devono essere rappresentati e tutte le operazioni devono poter essere eseguite utilizzando i concetti descritti nello schema.
  - Leggibilità: lo schema deve essere autoesplicativo; quindi vanno usati nomi significativi per i vari costrutti e vanno seguite alcune regole pratiche:
    - I costrutti vanno disposti in modo uniforme, mettendo al centro quelli che partecipano a più relazioni.
    - Le linee tracciate devono essere perpendicolari e bisogna cercare di minimizzarne le intersezioni.
    - Le entità padri vanno poste sopra alle entità figlie (nel caso delle generalizzazioni).
  - Minimalità: tutte le specifiche dovrebbero essere rappresentate una sola volta nello schema, i.e., non ci devono essere ridondanze.

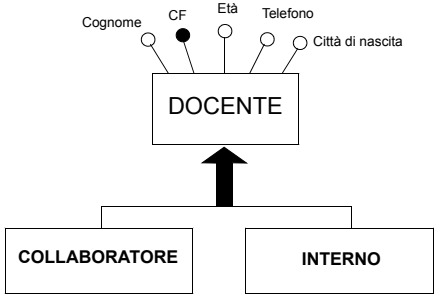
Metodologia generale

- Analisi dei requisiti:
  - Costruzione di un glossario dei termini
  - Analisi dei requisiti ed eliminazione delle eventuali ambiguità
  - Raggruppamento dei requisiti in insiemi omogenei
- Passo base:
  - Individuazione dei concetti maggiormente rilevanti e loro rappresentazione in uno schema scheletro.
- Passo di decomposizione (opzionale):
  - Decomposizione dei requisiti relativamente ai concetti presenti nello schema scheletro.
- Passo iterativo (da ripetere per ogni sottoschema fino alla rappresentazione di tutte le specifiche):
  - Raffinamento dei concetti presenti in base alle loro specifiche.
  - Aggiunta di nuovi concetti allo schema per descrivere nuove cose.
- Passo di integrazione:
  - Integrazione in un unico schema dei vari sottoschemi, con riferimento allo schema scheletro.
- Analisi di qualità:
  - Verifica della correttezza, completezza, minimalità e leggibilità dello schema (con eventuale ristrutturazione di quest'ultimo)

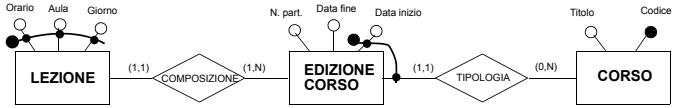
Esempio: società di formazione



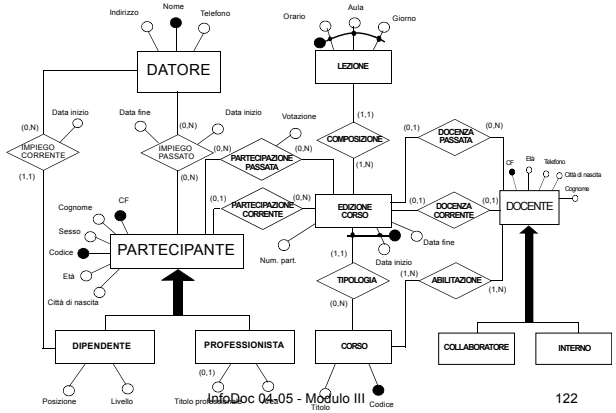
Esempio: società di formazione



Esempio: società di formazione



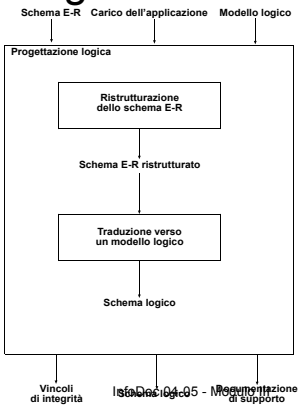
Esempio: società di formazione



Progettazione logica

- Lo scopo della fase di progettazione logica è di produrre uno schema logico, in accordo al modello dei dati scelto, che rappresenti fedelmente l'informazione descritta da uno schema E-R (prodotto nella fase di progettazione concettuale).
- Nel seguito assumiamo come modello dei dati il modello relazionale.
- Una procedura di traduzione diretta dallo schema E-R a quello logico corrispondente non è possibile per le seguenti ragioni:
  - Vi sono costrutti del modello E-R che non hanno un costrutto naturalmente corrispondente nel modello relazionale (e.g., le generalizzazioni).
  - Mentre uno schema concettuale è ad alto livello ed indipendente dalla tecnologia scelta per l'implementazione, lo schema logico è il punto di partenza di quest'ultimo e quindi deve tener conto del fattore prestazioni del prodotto finale.
- Quindi la fase di progettazione logica si divide in due parti:
  - Ristrutturazione dello schema E-R in base a criteri di ottimizzazione e di semplificazione.
  - Traduzione dello schema ristrutturato in uno schema logico ed eventuale ulteriore ottimizzazione.

Progettazione logica



Ristrutturazione degli schemi E-R

- Per quanto riguarda la fase di ristrutturazione possiamo individuare i seguenti passi:
  - **Analisi delle ridondanze**: si deve decidere se mantenere o meno le ridondanze (i.e., i concetti derivabili da altri) presenti nello schema.
  - **Eliminazione delle generalizzazioni**: bisogna sostituire le generalizzazioni presenti nello schema con altri costrutti che siano naturalmente traducibili in costrutti del modello relazionale.
  - **Partizionamento/accorpamento di entità e associazioni**: a seconda dei casi, potrebbe essere utile suddividere ulteriormente dei concetti od accorparne altri strettamente correlati.
  - **Scelta degli identificatori primari**: per le entità che hanno più di un identificatore si deve indicare quello principale.

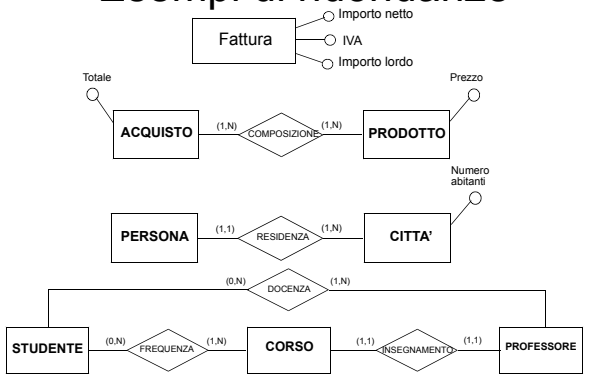
Analisi delle ridondanze

- Le ridondanze presentano il vantaggio di semplificare le interrogazioni, in quanto riducono il numero di accessi necessari per recuperare i dati necessari alle operazioni.
- Tuttavia vi sono dei lati negativi:
  - Le operazioni di aggiornamento risultano più onerose in presenza di concetti ridondanti (l'operazione va ripetuta per ogni occorrenza).
  - Vi è un maggior consumo di spazio (memoria).
- Quindi, di volta in volta, bisogna valutare i pro ed i contro in termini di guadagno di velocità delle operazioni sui dati e di consumo di memoria, prima di decidere se eliminare o mantenere una data ridondanza.

Tipologie di ridondanza

- I tipi di ridondanza che si possono avere in uno schema E-R sono i seguenti:
  - Attributi derivabili, occorrenza per occorrenza (di entità o associazione), da altri attributi dello stesso concetto.
  - Attributi derivabili da attributi di altre entità o associazioni (di solito rappresentano valori aggregati).
  - Attributi derivabili da operazioni di conteggio di occorrenze (è un caso particolare del precedente).
  - Associazioni derivabili dalla composizione di altre associazioni in presenza di cicli (si noti tuttavia che la semplice presenza di cicli in uno schema E-R non comporta necessariamente ridondanze di questo tipo).

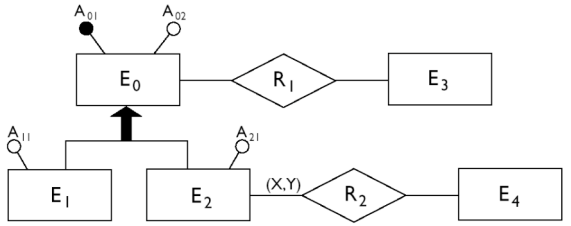
Esempi di ridondanze



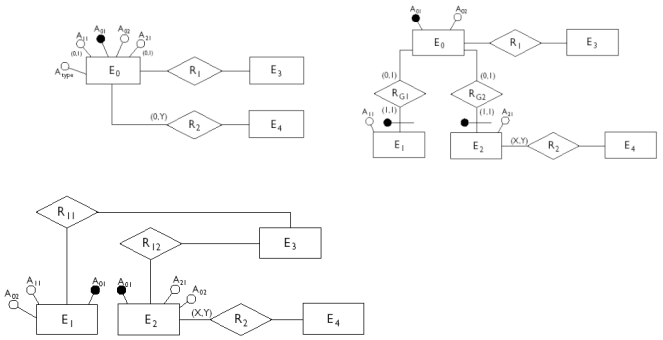
Eliminazione delle generalizzazioni

- Ci sono tre alternative possibili:
  - **Accorpamento delle entità figlie nell'entità padre**: le entità figlie spariscono ed i loro attributi e partecipazioni ad associazioni vengono aggiunti al padre; viene inoltre aggiunto a quest'ultimo un attributo che serve a distinguere il tipo di occorrenza (a seconda della "provenienza" dalle entità figlie).
  - **Accorpamento dell'entità padre nelle entità figlie**: l'entità padre sparisce ed i suoi attributi e le associazioni a cui partecipava vengono aggiunti alle entità figlie (ereditarietà).
  - **Sostituzione della generalizzazione con associazioni (uno-a-uno)**: non ci sono trasferimenti di attributi o associazioni, in quanto le figlie sono sempre identificate esternamente dall'entità padre, ma si rende necessario introdurre un vincolo che impedisca alle occorrenza del padre di partecipare a più di una delle nuove associazioni introdotte.

Esempio



Ristrutturazioni possibili



Eliminazione delle generalizzazioni

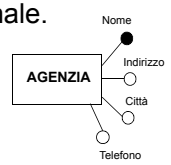
- Vantaggi/svantaggi delle alternative di eliminazione:
  - La prima alternativa è conveniente quando le operazioni non distinguono tra le occorrenze e gli attributi delle entità in gioco. C'è un aumento dello spreco di memoria (presenza di valori nulli) bilanciato da una diminuzione degli accessi necessari.
  - La seconda alternativa è possibile soltanto quando la generalizzazione è totale. Conviene quando le operazioni distinguono fra le entità figlie, usando soltanto occorrenze di una e non delle altre. In questo caso abbiamo sia un risparmio di memoria rispetto alla prima alternativa (assenza di valori nulli), sia una diminuzione degli accessi necessari (non è necessario visitare l'entità padre).
  - La terza alternativa conviene quando la generalizzazione non è totale e ci sono operazioni che distinguono fra entità padre ed entità figlie. Rispetto alla prima alternativa si risparmia memoria (per l'assenza di valori nulli), ma c'è un aumento degli accessi.
  - La terza alternativa ha il vantaggio (che esula dallo schema di analisi dei costi considerato) consistente nel fatto che genera entità con pochi attributi. Quindi le strutture logiche risultanti saranno di piccole dimensioni, consentendo di estrarre un elevato numero di risultati (tuple) ad ogni query.

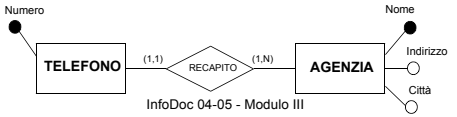
Partizionamento/accorpamento

- Per garantire una maggiore efficienza delle operazioni, entità e associazioni possono essere partizionate o accorpate in base al seguente principio:
  - Si riducono gli accessi separando gli attributi di uno stesso concetto che vengono utilizzati da operazioni distinte e raggruppando quelli di concetti diversi che vengono utilizzati dalle stesse operazioni.
- Per quanto riguarda i partizionamenti di entità si ha quanto segue:
  - I partizionamenti effettuati operando sugli attributi vengono detti decomposizioni verticali: hanno l'effetto di produrre entità con pochi attributi.
  - I partizionamenti effettuati operando sulle occorrenze di un'entità vengono detti decomposizioni orizzontali: possono avere delle ripercussioni negative sulle prestazioni del sistema in quanto comportano la potenziale duplicazione di tutte le relazioni a cui partecipava l'entità originale.

Eliminazione degli attributi multivalore

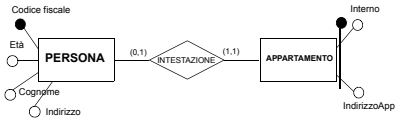
- Gli attributi multivalore vanno eliminati in quanto non hanno un corrispondente costrutto nel modello relazionale.

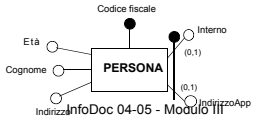




Accorpamento di entità

- Gli accorpamenti in genere si eseguono su entità che partecipano ad associazioni di tipo uno-a-uno. Negli altri casi (uno-a-molti e molti-a-molti) è facile che si generino ridondanze.
- Un possibile effetto collaterale di un accorpamento fra entità è la presenza di valori nulli.





Partizionamento/accorpamento di associazioni

- Quanto detto a proposito del partizionamento/accorpamento di entità può essere esteso anche alle associazioni.
- Il criterio da seguire è quello di decomporre un'associazione fra due entità in due o più associazioni fra le stesse entità per separare le occorrenze dell'associazione originale che erano sempre utilizzate in modo esclusivo dalle operazioni.
- Viceversa conviene accorpare due o più associazioni fra entità che si riferiscono ad aspetti diversi dello stesso concetto in un'unica associazione, se le varie occorrenze delle singole associazioni vengono sempre utilizzate contemporaneamente dalle operazioni.

Identificatori principali

- Dato che le chiavi nel modello relazionale vengono usate per stabilire delle corrispondenze fra dati appartenenti a relazioni distinte, è fondamentale stabilire quali siano gli identificatori principali.
- Infatti questi ultimi corrisponderanno poi alle chiavi primarie sulle quali i DBMS costruiscono delle strutture ausiliarie (indici) per il reperimento efficiente dei dati.
- Quindi, nel caso vi siano entità con più identificatori, bisogna sceglierne uno principale in base ai seguenti criteri:
  - Vanno scartati gli attributi che ammettono valori nulli.
  - In genere è preferibile scegliere identificatori composti da pochi attributi.
  - Gli identificatori interni con pochi attributi vanno preferiti a quelli esterni.
  - Gli identificatori utilizzati da molte operazioni sono da preferire agli altri.
- Nel caso non vi siano degli identificatori idonei, è possibile introdurne uno apposito che conterrà come valori dei codici, generati appositamente per distinguere le varie occorrenze dell'entità.

Traduzione verso il modello relazionale

- Dato lo schema E-R ristrutturato si procede alla sua traduzione in uno schema logico *equivalente*, i.e., uno schema che rappresenta le stesse informazioni.
- I casi contemplati sono i seguenti:
  - Entità.
  - Associazioni molti-a-molti.
  - Associazioni uno-a-molti.
  - Entità con identificatore esterno.
  - Associazioni uno-a-uno.

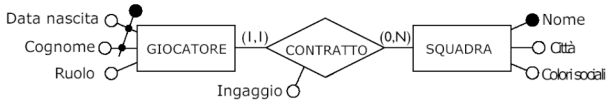
Associazioni molti-a-molti

IMPIEGATO(Matricola,Cognome, Stipendio)
PROGETTO(Codice,Nome,Budget)
PARTECIPAZIONE(Matricola,Codice,DataInizio)

Al fine di rendere più comprensibile lo schema è opportuno eseguire delle rinomine:

PARTECIPAZIONE(Impiegato,Progetto,DataInizio)

Associazioni uno a molti



In base alla modalità di traduzione per le associazioni molti-a-molti lo schema logico risultante dovrebbe essere il seguente:
GIOCATORE(Cognome,DataNascita, Ruolo)
SQUADRA(Nome,Città,ColoriSociali)
CONTRATTO(Giocatore,DataNascitaGiocatore,NomeSquadra,Ingaggio)

Tuttavia, siccome GIOCATORE E CONTRATTO hanno la stessa chiave è possibile fonderle, ottenendo lo schema seguente:
GIOCATORE(Cognome,DataNascita, Ruolo, NomeSquadra, Ingaggio)
SQUADRA(Nome,Città,ColoriSociali)

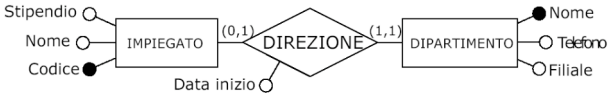
Entità con identificatore esterno

In questo caso, rappresentando l'identificatore esterno, si rappresenta automaticamente anche l'associazione fra le due entità:

STUDENTE(Matricola, NomeUniversità, Cognome, AnnoIscrizione)
UNIVERSITA'(Nome, Città, Indirizzo)

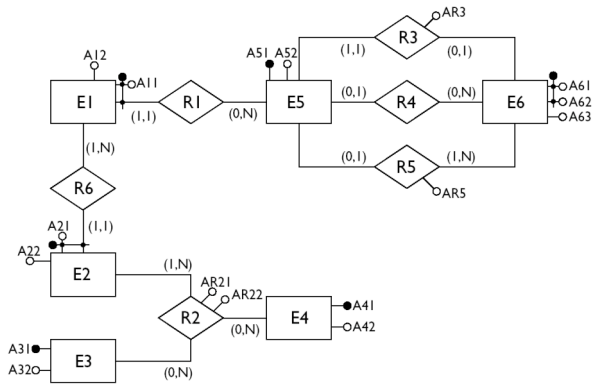
Associazioni uno-a-uno

DIRETTORE(Codice,Nome,Stipendio)
DIPARTIMENTO(Nome,Telefono,Filiale,Direttore,DataInizio)



IMPIEGATO(Codice,Nome,Stipendio)
DIPARTIMENTO(Nome,Telefono,Filiale,Direttore,DataInizio)

Esempio



Esempio

Lo schema logico corrispondente allo schema E-R precedente è quindi definito come segue:

E1(<u>A11</u>, <u>A51</u>, A12)

E2(<u>A21</u>, <u>A11</u>, <u>A51</u>, A22)

E3(<u>A31</u>, A32)

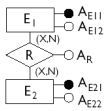
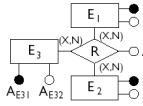
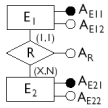
E4(<u>A41</u>,A42)

E5(<u>A51</u>, A52, A61R3, A62R3, AR3, A61R4, A62R4, A61R5, A62R5, AR5)

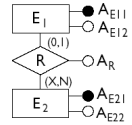
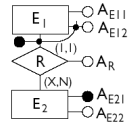
E6(<u>A61</u>, <u>A62</u>, A63)

R2(<u>A21</u>, <u>A11</u>, <u>A51</u>, <u>A31</u>, <u>A41</u>, AR21, AR22)

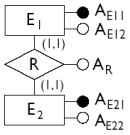
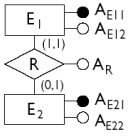
Schemi riassuntivi

Tipo	Concetto iniziale	Risultato possibile
Associazione binaria molti a molti		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$
Associazione ternaria molti a molti		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$
Associazione uno a molti con partecipazione obbligatoria		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

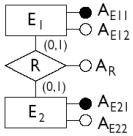
Schemi riassuntivi

Tipo	Concetto iniziale	Risultato possibile
Associazione uno a molti con partecipazione opzionale		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ Oppure: $E_1(\underline{A_{E11}}, A_{E21}, \underline{A^*_{E21}}, \underline{A^*_R})$ $E_2(\underline{A_{E21}}, A_{E22})$
Relazione con identificatore esterno		$E_1(\underline{A_{E12}}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

Schemi riassuntivi

Tipo	Concetto iniziale	Possibile risultato
Associazione uno a uno con partecipazione obbligatoria per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ Oppure: $E_2(\underline{A_{E21}}, A_{E22}, \underline{A_{E11}}, A_R)$ $E_1(\underline{A_{E11}}, A_{E12})$
Associazione uno a uno con partecipazione opzionale per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

Schemi riassuntivi

Tipo	Concetto iniziale	Possibile risultato
Associazione uno a uno con partecipazione opzionale per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E21})$ $E_2(\underline{A_{E21}}, A_{E22}, A^*_{E11}, A^*_R)$ Oppure: $E_1(\underline{A_{E11}}, A_{E12}, A^*_{E21}, A^*_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ Oppure: $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, A_{E21}, A_R)$