Sistemi Informativi – Lezione 1

Prof. Paolo Ciaccia
pciaccia@deis.unibo.it

DEIS – Università degli Studi di Bologna

Obiettivi della lezione

- Introdurre gli elementi essenziali relativi a:
  - DBMS, componente chiave di qualsiasi Sistema Informativo
  - Modello relazionale, rappresentazione standard dei dati in un DBMS
  - SQL, linguaggio standard per interagire con un DBMS

Sistemi Informativi

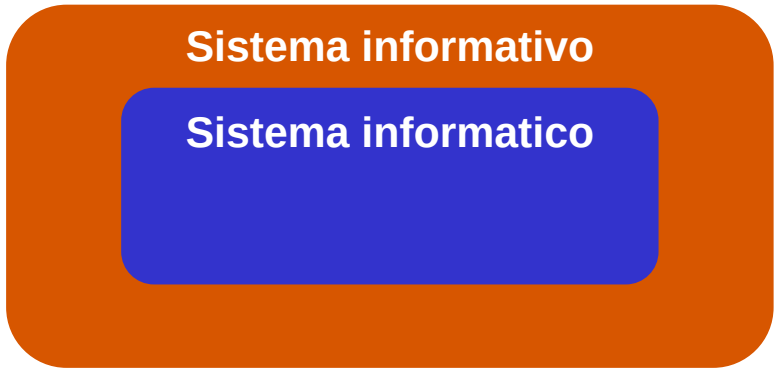
- Un sistema informativo (information system, IS) è un componente di una organizzazione (azienda, ente, …) il cui scopo è gestire le informazioni utili per gli scopi dell'organizzazione stessa

  GESTIRE = acquisire, elaborare, conservare, produrre, distribuire

- Un SI gestisce informazioni, ma ciò non significa necessariamente fare ricorso a strumenti automatici propri della tecnologia dell'informazione

  Banche e servizi anagrafici esistono da secoli!

- La parte automatizzata di un IS viene più propriamente denominata Sistema Informatico



Dati e Basi di Dati

- Il modo più comune con cui un sistema informatico gestisce le informazioni è attraverso la rappresentazione codificata dei dati di interesse
- Intuitivamente, una Base di Dati (DB - Data Base o Database) può pensarsi come una collezione di dati che rappresentano le informazioni di interesse per un'organizzazione
- In termini più precisi, un DB è una collezione di dati gestita da un DBMS

DBMS = Data Base Management System

DBMS: caratteristiche di base

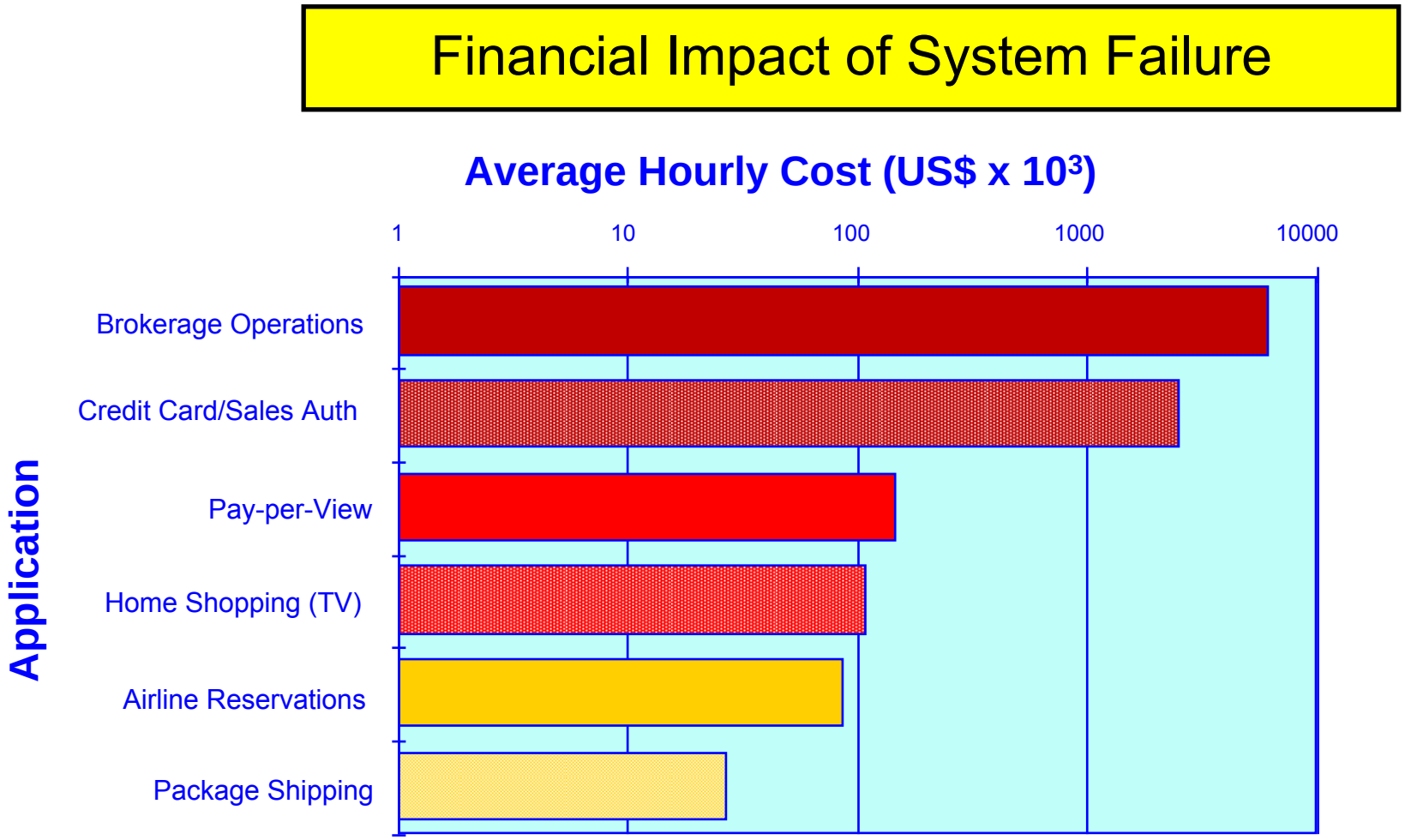
- Un DBMS è un sistema software in grado di gestire collezioni di dati che sono condivise da più applicazioni e utenti (e molto altro ancora…)

- Un DBMS deve essere in grado:
  - di gestire grandi quantità di dati (Giga-Tera byte e oltre)
  - di garantirne la persistenza (anche a fronte di guasti)
  - di offrire una "visione strutturata" dei dati stessi, che dipende dal modello (logico) dei dati supportato

RDBMS = DBMS che supporta il modello relazionale dei dati

Modello relazionale ≈ i dati sono rappresentati in forma tabellare

Il valore dei dati



DBMS: principali funzionalità

- Le caratteristiche fondamentali di un DBMS sono 3, riassumibili dicendo che:

  un DBMS è un sistema software che gestisce
  - grandi quantità di dati
  - persistenti
  - e condivisi
- La gestione di grandi quantità di dati richiede particolare attenzione ai problemi di efficienza (ottimizzazione delle richieste, ma non solo!)
- La persistenza e la condivisione richiedono che un DBMS fornisca dei meccanismi per garantire l'affidabilità dei dati (fault tolerance), per il controllo degli accessi e per il controllo della concorrenza
- Diverse altre funzionalità vengono messe a disposizione per motivi di efficacia, ovvero per semplificare la descrizione dei dati, lo sviluppo delle applicazioni, l'amministrazione di un DB, ecc.

Perché non i file system?

- Per gestire grandi quantità di dati in modo persistente e condiviso, sarebbe anche possibile fare uso dei file system, ma ciò ha una serie di inconvenienti, tra cui:
  - Non sono disponibili i servizi aggiuntivi offerti da un DBMS
  - I meccanismi di condivisione sono limitati, in particolare il livello di granularità è quello del file

    Es: due utenti non possono modificare contemporaneamente parti (record) diverse di uno stesso file
  - L'accesso a file condivisi richiede una descrizione degli stessi nel codice delle applicazioni, con rischi di descrizioni errate e quindi inconsistenti
- Per contro, la gestione dei dati mediante file system può risultare più efficiente che con un DBMS, proprio per la maggiore semplicità dei primi

Il modello dei dati

- Dal punto di vista utente un DB è visto come una collezione di dati che modellano una certa porzione della realtà di interesse
- L'astrazione logica con cui i dati vengono resi disponibili all'utente definisce un modello dei dati; più precisamente:

  un modello dei dati è una collezione di concetti che vengono utilizzati per descrivere i dati, le loro associazioni, e i vincoli che questi devono rispettare

- Un ruolo di primaria importanza nella definizione di un modello dei dati è svolto dai meccanismi che possono essere usati per strutturare i dati (cfr. i costruttori di tipo in un linguaggio di programmazione)
  - Ad es. esistono modelli in cui i dati sono descritti (solo) sotto forma di alberi (modello gerarchico), di grafi (modello reticolare) e di oggetti complessi (modello a oggetti)

Indipendenza fisica e logica

- Tra gli obiettivi di un DBMS vi sono quelli di fornire caratteristiche di:

Indipendenza fisica

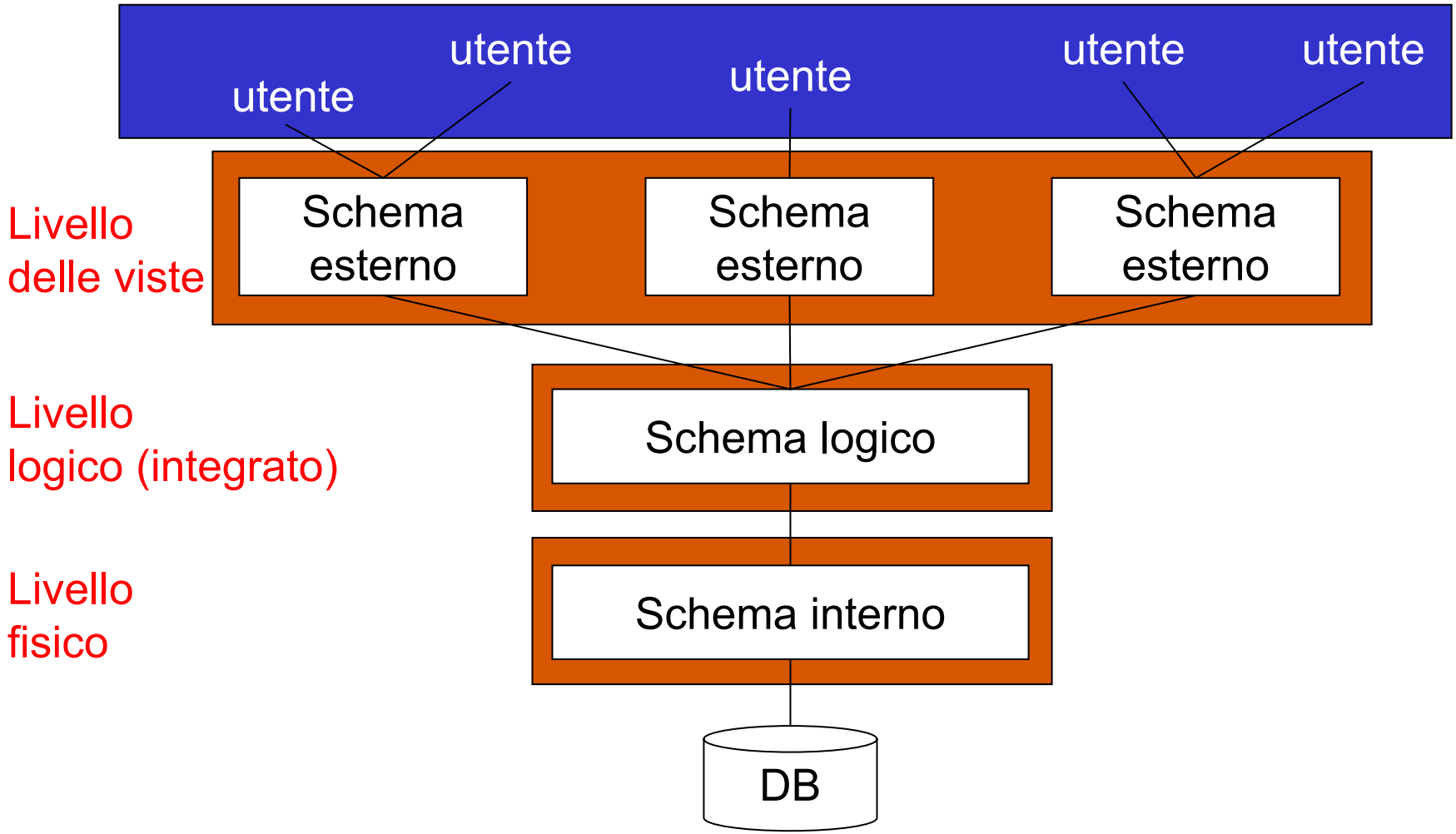
- L'organizzazione fisica dei dati dipende da considerazioni legate all'efficienza delle organizzazioni adottate. La riorganizzazione fisica dei dati non deve comportare effetti collaterali sui programmi applicativi

Indipendenza logica

- Pur in presenza di uno schema logico integrato non è utile o conveniente (ad es. per motivi di sicurezza) che ogni utente ne abbia una visione uniforme

- La soluzione porta a quella che è comunemente nota come…

Architettura a 3 livelli di un DBMS



I linguaggi dei DBMS

- Un DBMS mette a disposizione diversi linguaggi per interagire con le BD. Il livello di astrazione di tali linguaggi dipende fortemente dal modello dei dati cui ci si riferisce
- Una comune distinzione classifica i linguaggi sulla base delle funzioni svolte:
  - DDL (Data Definition Language)
    - Serve per definire gli schemi (logici, esterni, interni)
  - DML (Data Manipulation Language)
    - Serve per interrogare e modificare le istanze delle BD
  - DCL (Data Control Language)
    - Include comandi di vario tipo, ad es. per il controllo degli accessi

SQL riunisce in sé istruzioni di tutte le tre tipologie
(per cui si parla del DDL di SQL, del DML di SQL e del DCL di SQL)

Il modello relazionale

Relazionale, Gerarchico e Reticolare

- Il modello relazionale è stato introdotto nel 1970 da E.F. Codd (un ricercatore dell'IBM di San Jose, CA, USA) allo scopo di favorire l'indipendenza dei dati

- I modelli preesistenti (gerarchico e reticolare) erano fortemente influenzati da considerazioni di natura fisica, che enfatizzavano quindi aspetti di efficienza rispetto a quelli di semplicità d'uso:

  VELOCI MA COMPLICATI!!

- Rispetto agli altri modelli, quello relazionale si caratterizza per:
  - Semplicità concettuale
  - Precisione formale

Un po' di storia…

Anni '70: definizione del modello, prima versione del linguaggio SQL (allora SEQUEL), studi fondamentali sulla tecnologia relazionale (ottimizzazione, transazioni, recovery, …) e primi prototipi di DBMS relazionali:

- System R (IBM, laboratorio di ricerca di San Jose, CA, USA)
- Ingres (Università di Berkeley, CA,USA), il "nonno" di PostgreSQL

Anni '80: prima standardizzazione di SQL, primi prototipi commerciali:

- SQL/DS (derivato da System R)
- Oracle
- IBM DB2

Anni '90: standard ISO-ANSI SQL-2 (quello attualmente di riferimento, anche noto come SQL-92)

- Esiste già lo standard SQL:1999 (o SQL-3), ma non è ancora completamente recepito dai costruttori
- …e sono allo studio diverse altre estensioni del linguaggio

Un po' di storia… DBMS F/OS

- L'evoluzione dei DBMS F/OS ha di recente registrato notevoli progressi, a fronte del sempre maggiore utilizzo che ne viene fatto per gestire i dati di web applications
- Gli aspetti principali su cui valutare l'evoluzione sono:
  - Efficienza
  - Caratteristiche avanzate
- Sistemi quali MySQL, tradizionalmente efficienti per applicazioni prevalentemente read-only, si sono di recente muniti di caratteristiche proprie di DBMS più complessi (SQL più esteso, fault-tolerance, …)
- Sistemi quali PostgreSQL, tradizionalmente pensati per fornire caratteristiche innovative, sono continuamente migliorati per garantire alti livelli prestazionali

Relazioni = insiemi di oggetti

- Una relazione serve a rappresentare un

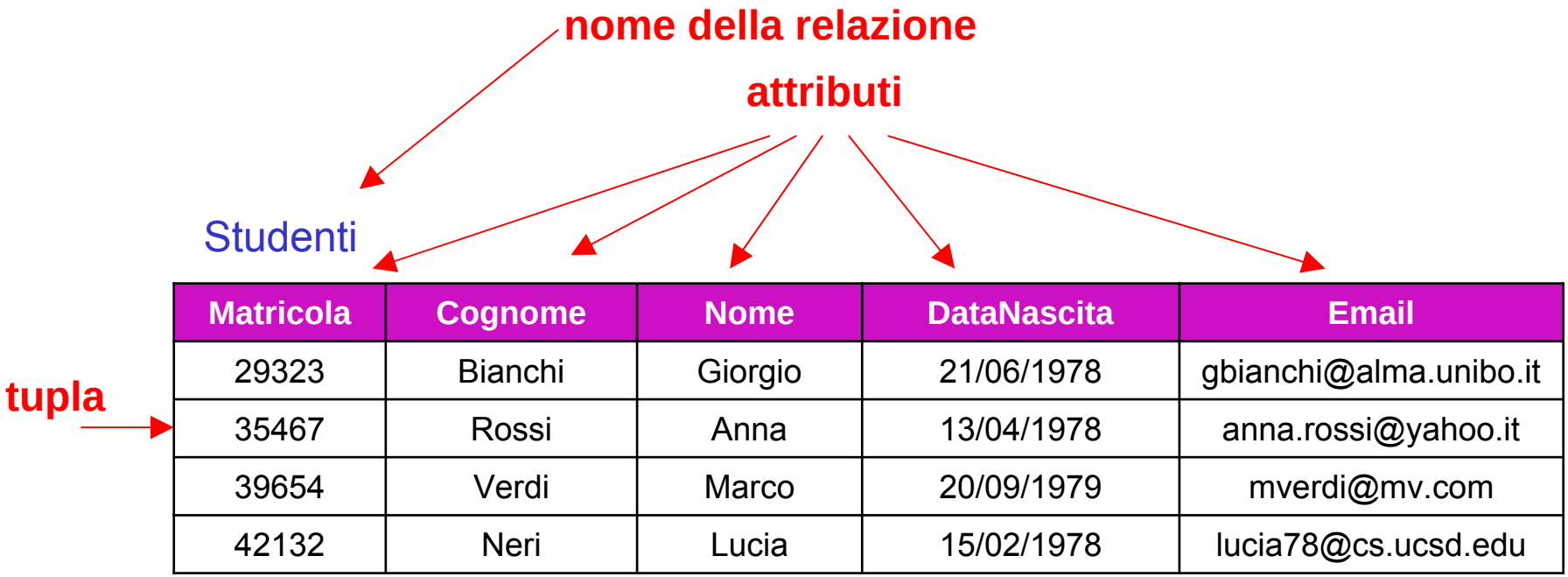
insieme di oggetti della realtà di interesse
che possiedono caratteristiche (proprietà) comuni

- Ogni specifico oggetto si caratterizza per i valori specifici che assume per tali proprietà

"Lo studente Giorgio Bianchi, nato il 21 Giugno 1978, ha numero di matricola 29323 ed email gbianchi@alma.unibo.it"

Relazioni: come sono fatte

- Una relazione può essere informalmente definita come una “tabella” le cui colonne (attributi) rapresentano le proprietà di interesse, e le cui righe (o tuple) rappresentano ciascuna uno specifico oggetto presente nel DB
- Una relazione ha necessariamente un nome, univoco all’interno del DB



Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

Relazione: alcune precisazioni (1)

- Il contenuto di una relazione non dipende dall'ordinamento delle tuple, quindi questa:

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

e questa:

Matricola	Cognome	Nome	DataNascita	Email
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it

sono uguali (la stessa relazione!)

Relazione: alcune precisazioni (2)

- Analogamente, anche l'ordine degli attributi non è rilevante, ovvero questa:

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

e questa:

Cognome	Nome	Matricola	DataNascita	Email
Bianchi	Giorgio	29323	21/06/1978	gbianchi@alma.unibo.it
Rossi	Anna	35467	13/04/1978	anna.rossi@yahoo.it
Verdi	Marco	39654	20/09/1979	mverdi@mv.com
Neri	Lucia	42132	15/02/1978	lucia78@cs.ucsd.edu

sono uguali

Relazione = schema + istanza

- Quando si parla di “relazione” in realtà ci si riferisce a un oggetto composto di 2 parti:

Lo SCHEMA, formato dal nome della relazione e dal nome degli attributi:

Studenti

Matricola	Cognome	Nome	DataNascita	Email

L’ISTANZA, formata dai dati veri e propri:

29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

Relazione = schema + istanza: sempre?

- Possiamo avere una relazione formata dal solo schema?

SI' e NO, in realtà l'istanza c'è, ma non contiene nessuna tupla (= istanza vuota)

  - E' quello che normalmente succede quando si crea una relazione!

Studenti

Matricola	Cognome	Nome	DataNascita	Email

- Possiamo avere una relazione formata dalla sola istanza?

ASSOLUTAMENTE NO, i nostri dati non significherebbero nulla!

Rappresentare uno schema

- Sinteticamente, uno schema si può esprimere come:

Studenti(Matricola,Cognome,Nome,DataNascita,Email)

- … in realtà per definire compiutamente uno schema bisogna aggiungere più informazioni…

Data Base relazionale

- Cos'è un DB relazionale? E' un insieme di relazioni, ovvero:
- Lo schema di un DB relazionale è un insieme di schemi di relazioni con nomi distinti, più un nome per il DB
- L' istanza è un insieme di istanze di relazioni, una per ogni schema di relazione nel DB

Un semplice DB relazionale

Università

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

Corsi

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Vincoli di integrità dei dati

- Una relazione non deve (e non può) essere vista come un contenitore di dati arbitrari
- Se così fosse non riusciremmo più a interpretare correttamente i dati e molte operazioni non si potrebbero eseguire in maniera affidabile
- Si rende pertanto necessaria un'attività di analisi rivolta a evidenziare quali sono i vincoli che le nostre istanze devono soddisfare affinché si possano considerare valide (o "legali", "corrette" "ammissibili", ecc.)

Studenti

Matricola	Cognome	Nome	DataNascita	Reddito	Email
29323	Bianchi	Giorgio	21/06/1978	(500, gen), (800, feb), (300, mar),...	gbianchi@alma.unibo.it
29323	Rossi	Anna	13/04/1978		
	1234	bababa	20/09/1979	1200	mverdi@mv.com
42132	Neri	Lucia	15/13/2026	Basso	Lucia78!£$cs

Vincoli di dominio

- Il vincolo di base che va specificato per ogni attributo riguarda il suo dominio, ovvero che tipo di dati “ha senso” per quell’attributo

Matricola	Cognome	Nome	DataNascita	Reddito	Email
	1234	bababa	20/09/1979	1200	mverdi@mv.com
42132	~~Neri~~	~~Lucia~~	15/13/2026	Basso	Lucia78!£$cs

- In pratica i domini che si possono usare dipendono dal DBMS e includono i tipi più comuni di dati (interi, stringhe, date)
- Come vedremo vi è tuttavia la possibilità di specificare ulteriori restrizioni

1NF, ovvero solo domini semplici

- Il modello relazionale non permette di usare domini arbitrari per la definizione delle relazioni; in particolare non è in generale possibile usare domini strutturati (array, set, liste, ...)

Reddito
(500, gen),
(800, feb),
(300, mar),…

- Vi sono però delle eccezioni notevoli (ad es. le date e le stringhe)
- Concisamente, una relazione in cui ogni dominio è “atomico” (non ulteriormente decomponibile) si dice che è in

  Prima Forma Normale, o 1NF (1st Normal Form)

- In molti casi è pertanto richiesta un’attività di normalizzazione dei dati che dia luogo a relazioni in 1NF e che preservi l’informazione originale

Normalizzazione dei dati (1)

- Supponiamo di avere il campo Reddito con i vari redditi mensili dell'ultimo anno:

Studenti

Matricola	Cognome	Nome	DataNascita	Reddito	Email
29323	Bianchi	Giorgio	21/06/1978	(500, gen), (800, feb), (300, mar),…	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	(1200, gen), (1100, feb),...	anna.rossi@yahoo.it

- Per prima cosa "estraiamo" Reddito dalla relazione Studenti, che così diventa:

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it

ed è in 1NF

Normalizzazione dei dati (2)

- Quindi “spezziamo” l’elenco dei redditi, creando per ogni mese una tupla con 2 attributi:

Mese	Reddito
gen	500
feb	800
mar	300
gen	1200
feb	1100
…	…

- Ora ci resta solo da “collegare” ciascuna tupla al suo studente. Come?

Normalizzazione dei dati (3)

- La scelta più semplice (ed è anche quella giusta!) è usare la Matricola:

Redditi

Matricola	Mese	Reddito
29323	gen	500
29323	feb	800
29323	mar	300
35467	gen	1200
35467	feb	1100
…	…	…

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it

Normalizzazione dei dati: altro esempio

Ricevuta n. 231 del 12/02/2002		
Coperti	2	3,00
Antipasti	1	5,80
Primi	2	11,45
Secondi	2	22,30
Caffè	2	2,20
Vino	1	8,00
Totale (Euro)		52,75

Ricevuta n. 352 del 13/02/2002		
Coperti	1	1,50

Ricevute

Numero	Data	Totale
231	12/02/2002	52,75
352	13/02/2002	…
…	…	…

Dettaglio

Numero	Quantità	Descrizione	Prezzo
231	2	Coperti	3,00
231	1	Antipasti	5,80
231	2	Primi	11,45
231	2	Secondi	22,30
231	2	Caffè	2,20
231	1	Vino	8,00
352	1	Coperti	1,50

Considerazioni sulla normalizzazione

- Il fatto che una rappresentazione normalizzata sia adeguata o meno dipende (molto) dal contesto
  - Ad es.: l'ordine delle righe nella ricevuta è rilevante o meno?
- Lo stesso dicasi per eventuali ridondanze che si possono venire ad osservare
  - Ad es.: il coperto ed il caffè hanno un prezzo che non varia da ricevuta a ricevuta?
- In generale è bene ricordare che ogni caso presenta una sua specificità, e quindi non va trattato "automaticamente"

- Normalizzare in 1NF è, a tutti gli effetti, un'attività di progettazione (logica), e in quanto tale può essere solo oggetto di "regole guida" che però non hanno validità assoluta

Informazione incompleta

- Le informazioni che si vogliono rappresentare mediante relazioni non sempre corrispondono pienamente allo schema prescelto, in particolare per alcune tuple e alcuni attributi potrebbe non essere possibile specificare, per diversi motivi, un valore del dominio

Matricola	Cognome	Nome	DataNascita	Reddito	Email
29323	Rossi	Anna	13/04/1978		
42132	Neri	Lucia	15/13/2026	500	

Lucia Neri non ha un'email (valore non applicabile)

Anna Rossi ha un reddito, ma non lo conosciamo (applicabile ma ignoto)

Anna Rossi non si sa se ha un'email (ignota l'applicabilità)

Cosa si fa nel modello relazionale?

- In diversi casi, in mancanza di informazione, si tende a usare un "valore speciale" del dominio (0, "", "-1", ecc.) che non si utilizza per altri scopi
- Questa pratica è fortemente sconsigliata, in quanto, anche dove possibile:
  - Valori inutilizzati potrebbero successivamente diventare significativi
  - Le applicazioni dovrebbero sapere "cosa significa in realtà" il valore usato allo scopo

Esempio (reale!): nel 1998, analizzando i clienti di un'assicurazione, si scoprì una strana concentrazione di ultra-novantenni… tutte le date di nascita ignote erano state codificate con "01/01/00"!!

- Nel modello relazionale si opera in maniera pragmatica: si adotta il concetto di valore nullo (NULL), che denota assenza di un valore nel dominio (e non è un valore del dominio)

Valori nulli: considerazioni

Matricola	Cognome	Nome	DataNascita	Reddito	Email
29323	Rossi	Anna	13/04/1978	NULL	NULL
42132	Neri	Lucia	15/13/2026	500	NULL

- La presenza di un valore nullo non fornisce alcuna informazione sull'applicabilità o meno
- È importante ricordare che NULL non è un valore del dominio; in particolare, se due tuple hanno entrambe valore NULL per un attributo, non si può inferire che esse abbiano lo stesso valore per quell'attributo, ovvero:

$$NULL \neq NULL$$

Valori nulli: restrizioni

- Possiamo sempre tollerare la presenza di valori nulli? NO!

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
NULL	729	30	SÌ
29323	913	**NULL**	NO
35467	913	30	NO

  - Il valore nullo per Matricola non permette di sapere chi ha sostenuto l'esame
  - Il valore nullo per Voto non è ammissibile nel contesto considerato

Istanze di questo tipo non sono accettabili!

- In questi casi si può (e si deve!) imporre l'assenza di valori nulli

Vincoli di chiave

- Un tipo importantissimo di vincoli sono i vincoli di chiave, che vietano la presenza di tuple distinte che hanno lo stesso valore su uno o più attributi

Matricola	Cognome	Nome	DataNascita
29323	Bianchi	Giorgio	21/06/1978
29323	Rossi	Anna	13/04/1978

  - Il valore di Matricola identifica univocamente uno studente, quindi non è ammissibile avere due studenti con la stessa matricola
  - Si noti che anche ogni insieme di attributi che include Matricola identifica uno studente
    - ad es. Matricola e Cognome

Relazioni con più chiavi

- E' possibile che una relazione abbia più di una chiave

Studenti

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
29323	BNCGRG78F21A	Bianchi	Giorgio	21/06/1978
35467	RSSNNA78D13A	Rossi	Anna	13/04/1978
39654	VRDMRC79I20A	Verdi	Marco	20/09/1979
42132	VRDMRC79I20B	Verdi	Marco	20/09/1979

- Sia Matricola che CodiceFiscale sono chiavi

Chiavi formate da più attributi (1)

- In alcuni casi una chiave è formata da due o più attributi

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

- Supponiamo che non vi siano verbalizzazioni di esami non superati
- Allora uno studente non può verbalizzare due o più volte lo stesso esame
- Quindi (Matricola,CodCorso) è una chiave
- …ma né Matricola né CodCorso, presi singolarmente, lo sono!

Chiavi formate da più attributi (2)

Redditi

Matricola	Mese	Reddito
29323	gen	500
29323	feb	800
29323	mar	300
35467	gen	1200
35467	feb	1100
…	…	…

- In questo caso la chiave è (Matricola,Mese)

- E se volessimo mantenere redditi per più anni?
- …introduciamo l'attributo Anno e la chiave diventa (Matricola,Mese,Anno)

Redditi

Matricola	Mese	Anno	Reddito
29323	gen	2004	100
29323	gen	2005	500
	…	…	…

Chiavi e valori nulli

- In presenza di valori nulli la funzione di identificazione svolta da una chiave viene meno

Studenti

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
NULL	NULL	Bianchi	Giorgio	21/06/1978
35467	RSSNNA78D13A	Rossi	Anna	13/04/1978
NULL	VRDMRC79I20A	Verdi	Marco	20/09/1979
42132	NULL	Verdi	Marco	20/09/1979

- La prima tupla non è identificabile in alcun modo, pertanto:

  È necessario specificare il valore di almeno una chiave!

- La terza e quarta tupla non sappiamo se si riferiscano o meno allo stesso studente, pertanto:

  Non è sufficiente specificare il valore di una chiave!

Chiave primaria

- Per evitare i problemi visti è necessario scegliere una chiave, detta chiave primaria, su cui non si ammettono valori nulli
- Convenzionalmente, gli attributi della chiave primaria vengono sottolineati

Studenti

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
29323	NULL	Bianchi	Giorgio	21/06/1978
35467	RSSNNA78D13A	Rossi	Anna	13/04/1978
39654	VRDMRC79I20A	Verdi	Marco	20/09/1979
42132	NULL	Verdi	Marco	20/09/1979

Nei casi in cui per nessuna chiave si possa garantire la disponibilità di valori, è necessario introdurre un nuovo attributo (un "codice") che svolga le funzioni di chiave primaria (si pensi ad esempio al caso in cui non si riesce a identificare un paziente al pronto soccorso ospedaliero)

Relazioni = legami tra oggetti

- Come già visto nel caso di Esami, una relazione può anche servire per rappresentare "legami" (associazioni) tra oggetti (es. un corso e uno studente)

  "Lo studente Giorgio Bianchi, nato il 21 Giugno 1978, con numero di matricola 29323 ed email gbianchi@alma.unibo.it, ha superato con voto 28 (senza lode) l'esame del corso di Analisi, codice 483, tenuto dal Prof. Biondi al primo anno"

- Osserviamo che CodCorso è la chiave primaria di Corsi, e che Matricola è la chiave primaria di Studenti. Quindi, in forma più compatta:

  "Lo studente con numero di matricola 29323 ha superato con voto 28 (senza lode) l'esame del corso con codice 483"

Rappresentare un'associazione

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

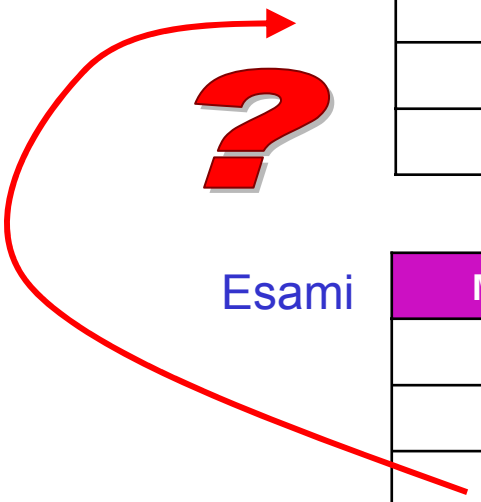
- Cosa abbiamo fatto?
- Abbiamo costruito una relazione con gli attributi con cui intendiamo descrivere un esame (Voto e Lode)
- Abbiamo poi "importato" le chiavi primarie dei (2) insiemi associati
- Una tupla di esami rappresenta quindi un singolo legame, identificato mediante i valori di chiave primaria delle relazioni coinvolte
- Ci manca però qualcosa, perché ora le nostre relazioni sono "indipendenti" tra loro…

Vincoli di integrità referenziale (1)

- Dobbiamo specificare dei vincoli di integrità referenziale, in modo che la funzione di correlazione operata dalle chiavi sia garantita
- In pratica, vogliamo vincolare i valori dell'attributo (o degli attributi) che "referenziano" (si riferiscono a) la chiave (quelli "importati")

Studenti

Matricola	Cognome	Nome	DataNascita
29323	Bianchi	Giorgio	21/06/1978
35467	Rossi	Anna	13/04/1978
39654	Verdi	Marco	20/09/1979
42132	Neri	Lucia	15/02/1978



Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
41235	913	26	NO
35467	913	30	NO

Vincoli di integrità referenziale (2)

- Diciamo che un attributo di una relazione è una foreign key (o “chiave importata”) se, in ogni istante, i suoi valori devono essere un sottoinsieme di quelli della chiave primaria di un’altra relazione
- In Esami, Matricola è una foreign key che referenzia la chiave primaria di Studenti; idem per CodCorso, che referenzia la chiave primaria di Corsi

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it

Corsi

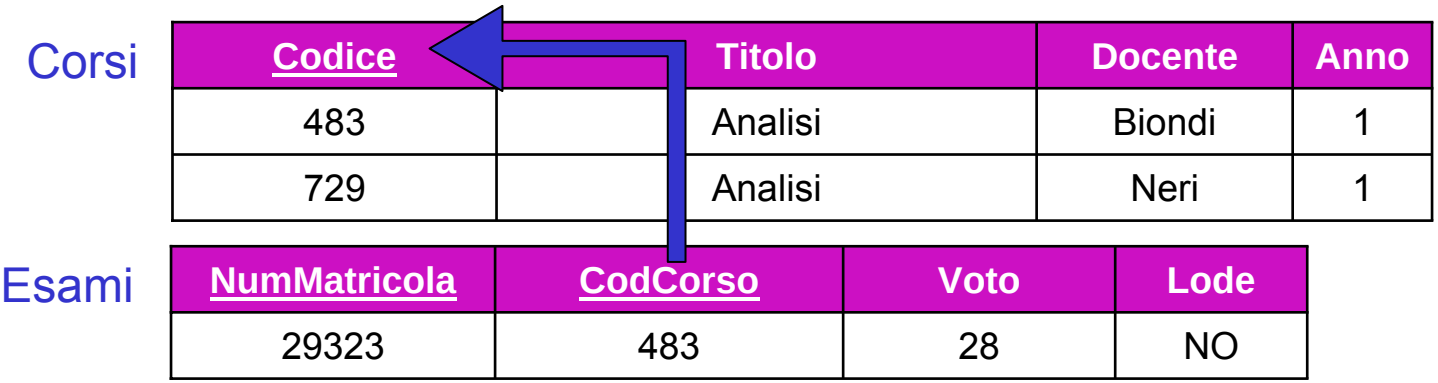
CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ

Foreign key: alcune precisazioni (1)

- In generale la foreign key e la primary key possono includere attributi con nomi diversi



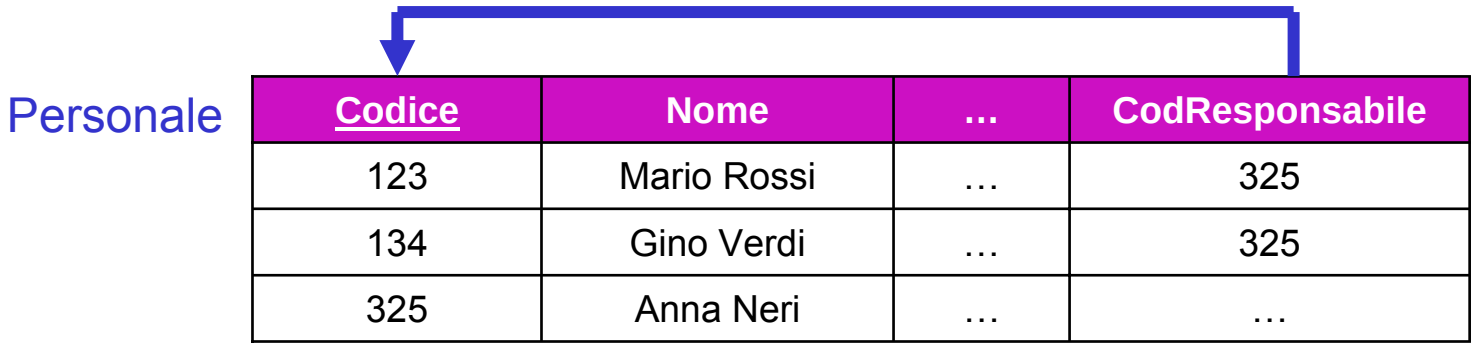
Corsi

Codice		Titolo	Docente	Anno
483		Analisi	Biondi	1
729		Analisi	Neri	1

Esami

NumMatricola	CodCorso	Voto	Lode
29323	483	28	NO

- Foreign key e primary key talvolta fanno parte della stessa relazione



Personale

Codice	Nome	…	CodResponsabile
123	Mario Rossi	…	325
134	Gino Verdi	…	325
325	Anna Neri	…	…

Foreign key: alcune precisazioni (2)

- In presenza di valori nulli, i vincoli di integrità referenziale si possono parzialmente rilassare

Personale

Codice	Nome	…	CodResponsabile
123	Mario Rossi	…	325
134	Gino Verdi	…	325
325	Anna Neri	…	NULL

- Nei DBMS un vincolo di integrità referenziale può anche esprimersi con riferimento a una generica chiave (quindi anche non primaria)

Studenti

Matricola	CodiceFiscale	Cognome	Nome	DataNascita
29323	BNCGRG78F21A	Bianchi	Giorgio	21/06/1978
35467	RSSNNA78D13A	Rossi	Anna	13/04/1978

Redditi

CF	Imponibile
BNCGRG78F21A	10000

Vincoli di tupla

- I vincoli di tupla esprimono condizioni che devono essere soddisfatte da ciascuna tupla (generalizzano quelli di dominio)

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	SÌ
39654	729	30	SÌ
29323	913	31	NO
35467	913	30	FORSE

  - Il Voto deve essere compreso tra 18 e 30

    $(\text{Voto} \geq 18)\ \text{AND}\ (\text{Voto} \leq 30)$
  - La Lode può solo assumere i valori `SÌ' o `NO'

    (Lode = `SÌ') OR (Lode = `NO')
  - La Lode si può assegnare solo se il Voto è 30:

    (Voto = 30) OR (Lode = `NO')
  - Nello schema Pagamenti(Data,ImportoLordo,Ritenute,Netto) si ha:

    ImportoLordo = Netto + Ritenute

Il linguaggio SQL

SQL: caratteristiche generali

- SQL (Structured Query Language) è il linguaggio standard *de facto* per DBMS relazionali, che riunisce in sé funzionalità di DDL, DML e DCL
- SQL è un linguaggio dichiarativo (non-procedurale), ovvero non specifica la sequenza di operazioni da compiere per ottenere il risultato
- Il modello dei dati di SQL è basato su tabelle anziché relazioni, in particolare:
  - Possono essere presenti righe (tuple) duplicate
- …il motivo è pragmatico, in particolare legato a considerazioni sull'efficienza (eliminare sempre i duplicati è costoso!)

SQL: standard e dialetti

- Il processo di standardizzazione di SQL è iniziato nel 1986
- Nel 1992 è stato definito lo standard SQL-2 (o SQL-92) da parte dell'ISO (International Standards Organization), e dell'ANSI (American National Standards Institute), rispettivamente descritti nei documenti ISO/IEC 9075:1992 e ANSI X3.135-1992 (identici!)
- Del 1999 è lo standard SQL:1999, che rende SQL un linguaggio computazionalmente completo (e quindi con istruzioni di controllo!) per il supporto di oggetti persistenti…
- Allo stato attuale ogni sistema ha (ancora) un suo dialetto, ovvero:
  - supporta (in larga parte) SQL-2
  - ha già elementi di SQL:1999
  - ha anche costrutti non standard
- Quella che vediamo è la parte "più diffusa" dello standard

DB di riferimento per gli esempi

…vedremo dopo come si può definire in SQL

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

Corsi

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

L'istruzione SELECT

- È l'istruzione che permette di eseguire interrogazioni (*query*) sul DB
- La forma di base è:

```
SELECT   ...
FROM     ...
WHERE    ...
```

ovvero:

- clausola SELECT(cosa si vuole come risultato)
- clausola FROM (da dove si prende)
- clausola WHERE (che condizioni deve soddisfare)

Estrarre tutti i dati

- La sintassi: `SELECT *`
  `FROM Corsi`

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

restituisce l'istanza della relazione Corsi (* significa: tutti gli attributi)

- Con `SELECT CodCorso, Titolo, Anno`
  `FROM Corsi`

otteniamo informazioni solo per gli attributi specificati ("proiezione")

CodCorso	Titolo	Anno
483	Analisi	1
729	Analisi	1
913	Sistemi Informativi	2

Risultati replicati (1)

- Se tra le colonne su cui si proietta non compare nessuna chiave, può capitare che si generino delle righe duplicate
- Ad esempio:

```
SELECT  Titolo
FROM    Corsi
```

Titolo
Analisi
Analisi
Sistemi Informativi

- La keyword **DISTINCT** elimina dal risultato le righe duplicate:

```
SELECT  DISTINCT Titolo
FROM    Corsi
```

Titolo
Analisi
Sistemi Informativi

Ridenominare le colonne

- Per chiarezza, leggibilità, ecc., è possibile dare un altro nome (uno "pseudonimo") alle colonne in output:

```
SELECT  Titolo AS NomeCorso, Docente AS Prof
FROM    Corsi
```

NomeCorso	Prof
Analisi	Biondi
Analisi	Neri
Sistemi Informativi	Castani

- La keyword **AS** può anche essere omessa:

```
SELECT  Titolo NomeCorso, Docente Prof
FROM    Corsi
```

Calcolare espressioni

- In output è anche possibile ottenere risultati di espressioni (numeriche, su stringhe, ecc.), che vengono valutate sulle tuple della relazione

```
SELECT  Matricola, Nome + ' ' + Cognome AS NomeCognome
FROM    Studenti
```

Matricola	NomeCognome
29323	Giorgio Bianchi
35467	Anna Rossi
39654	Marco Verdi
42132	Lucia Neri

```
SELECT  Voto / 3 AS Decimi
FROM    Esami
```

Decimi
9
10
8
10

R

A	B
20	12
15	8
8	3
14	17

```
SELECT  A + B
FROM    R
```

32
23
11
31

Filtrare il risultato: la clausola WHERE

- Per selezionare le sole tuple di interesse dobbiamo scrivere una condizione, che sia vera (soddisfatta) per tali tuple, ma non per le altre

```
SELECT Matricola, Voto, Lode
FROM   Esami
WHERE  CodCorso = 913
```

Matricola	Voto	Lode
29323	26	NO
35467	30	NO

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Clausola WHERE = espressione logica

- La clausola WHERE consiste, nel caso generale, di una espressione logica (operatori AND, OR, NOT) di predicati (condizioni)
- Una tupla soddisfa la clausola WHERE se e solo se l'espressione risulta vera per tale tupla

```
SELECT *
FROM   Esami
WHERE  CodCorso = 913
AND    Voto > 28
```

Matricola	CodCorso	Voto	Lode
35467	913	30	NO

Alcuni utili operatori (1)

- L'operatore LIKE, mediante le "wildcard"
  _ (corrisponde a un carattere arbitrario) e
  % (corrisponde a una stringa arbitraria),
  permette di trovare stringhe che soddisfano un certo "pattern"

Studenti la cui email finisce con '.it' e hanno una 'b' in seconda posizione

```
SELECT  *
FROM    Studenti
WHERE   Email LIKE '_b%.it'
```

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it

gbianchi@alma.unibo.it

Alcuni utili operatori (2)

- L'operatore BETWEEN permette di esprimere condizioni di appartenenza a un intervallo (estremi inclusi)

Esami con voto tra 26 e 29

```
SELECT  *
FROM    Esami
WHERE   Voto BETWEEN 26 AND 29
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
29323	913	26	NO

- Lo stesso risultato si può ottenere scrivendo:

```
SELECT  *
FROM    Esami
WHERE   Voto >= 26
AND     Voto <= 29
```

Alcuni utili operatori (3)

- L'operatore IN permette di esprimere condizioni di appartenenza a un insieme di valori

Esami dei corsi con codici 483 e 729

```
SELECT   *
FROM     Esami
WHERE    CodCorso IN (483,729)
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ

- Lo stesso risultato si può ottenere scrivendo:

```
SELECT   *
FROM     Esami
WHERE    CodCorso = 483
OR       CodCorso = 729
```

Valori nulli (1)

- La presenza di valori nulli può dar luogo a “strani risultati”
- Supponiamo che il DB non riporti la data di nascita di Anna Rossi

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	NULL	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

```
SELECT
FROM    Studenti
WHERE   DataNascita > ‘31/12/1978’
```

restituisce ovviamente

Matricola	Cognome	Nome	DataNascita	Email
39654	Verdi	Marco	20/09/1979	mverdi@mv.com

Valori nulli (2)

- Se ora volessimo vedere “tutti gli altri” potremmo scrivere

```
SELECT
FROM    Studenti
WHERE   DataNascita <= ‘31/12/1978’
-- oppure NOT (DataNascita > ‘31/12/1978’)
```

che restituisce

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

- In nessun caso riusciamo a selezionare Anna Rossi con una condizione sulla data di nascita!

Verifica di valori nulli

- Né la condizione **`DataNascita > '31/12/1978'`** né il suo contrario (**`DataNascita <= '31/12/1978'`**) sono vere se il valore della data di nascita è NULL

- Per verificare se un valore è NULL si deve usare l'operatore IS

```
SELECT  *
FROM    Studenti
WHERE   DataNascita IS NULL
```

Matricola	Cognome	Nome	DataNascita	Email
35467	Rossi	Anna	NULL	anna.rossi@yahoo.it

- **`NOT (A IS NULL)`**, che è vera se il valore dell'attributo A non è NULL, si scrive anche **`A IS NOT NULL`**

Espressioni logiche e valori nulli

- Nel caso di più condizioni bisogna fare attenzione alla presenza di valori nulli (logica a 3 valori: TRUE, FALSE, UNKNOWN))

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	NULL
35467	Rossi	Anna	NULL	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

```
SELECT Matricola
FROM   Studenti
WHERE  DataNascita <= '31/12/1978'
AND    Email LIKE '%.edu'
```

Giorgio Bianchi non siamo sicuri che soddisfi entrambe le condizioni, quindi non viene selezionato!

Matricola
42132

```
WHERE  DataNascita <= '31/12/1978'
OR     Email LIKE '%.edu'
```

Giorgio Bianchi ora va bene, perché siamo sicuri che soddisfa almeno una condizione!

Matricola
29323
42132

Ordinamento del risultato

- Per ordinare il risultato di una query secondo i valori di una o più colonne si introduce la clausola **ORDER BY**, e per ogni colonna si specifica se l'ordinamento è per valori "ascendenti" (**ASC**, il default) o "discendenti" (**DESC**). Si ordina sulla prima colonna, a parità di valori di questa sulla seconda, e così via

```
SELECT   *
FROM     Esami
ORDER BY CodCorso, Voto DESC
```

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
35467	913	30	NO
29323	913	26	NO

```
SELECT   *
FROM     Esami
ORDER BY Voto DESC, CodCorso
```

Matricola	CodCorso	Voto	Lode
39654	729	30	SÌ
35467	913	30	NO
29323	483	28	NO
29323	913	26	NO

Definizione di tabelle

- Mediante l'istruzione **CREATE TABLE** si definisce lo schema di una tabella e se ne crea un'istanza vuota

- Per ogni attributo va specificato
  - il dominio (obbligatorio!)
  - un eventuale valore di default
  - ed eventuali vincoli

- Infine possono essere espressi altri vincoli a livello di tabella

- Mediante l'istruzione **DROP TABLE** è possibile eliminare lo schema di una tabella (e conseguentemente la corrispondente istanza)

```
DROP TABLE Imp
```

Definizione di tabelle: esempio (1)

```
CREATE TABLE Studenti (
   Matricola char(5)      PRIMARY KEY, -- chiave primaria
   CF        char(16)     UNIQUE NOT NULL,  -- chiave
   Cognome   varchar(30)  NOT NULL,
   Nome      varchar(30)  NOT NULL,
   DataNascita date       NOT NULL,
   Email     varchar(100)       )

CREATE TABLE Corsi (
   CodCorso  int          PRIMARY KEY, -- chiave primaria
   Titolo    varchar(50)  NOT NULL,
   Docente   varchar(30),
   Anno      int          DEFAULT 1 CHECK (Anno > 0))
```

Definizione di tabelle: esempio (2)

```
CREATE TABLE Esami (
   Matricola char(5)      NOT NULL REFERENCES Studenti,
   CodCorso  int          NOT NULL REFERENCES Corsi,
   Voto      int          NOT NULL
                          CHECK (Voto BETWEEN 18 AND 30),
   Lode      char(2)      NOT NULL
                          CHECK (Lode IN ('SI','NO')),
   PRIMARY KEY (Matricola,CodCorso) )
```

- Per Esami è forzato specificare la chiave primaria in questo modo, in quanto è formata da più di un attributo, per Studenti si può scegliere

Valori nulli e valori di default

- Per vietare la presenza di valori nulli, è sufficiente imporre il vincolo NOT NULL

```
CF          char(16)     NOT NULL
```

- Per ogni attributo è possibile specificare un valore di default, che verrà usato se all'atto dell'inserimento di una tupla non viene fornito un valore per quell'attributo

```
Anno        int    DEFAULT 1

Email       varchar(100) DEFAULT 'guest@alma.unibo.it'
```

Chiavi

- La definizione di una chiave avviene esprimendo un vincolo UNIQUE, che si può specificare “in linea”, se la chiave consiste di un singolo attributo

```
CF          char(16)     UNIQUE
```

o dopo aver dichiarato tutti gli attributi (“vincolo di tabella”), se la chiave consiste di uno o più attributi:

```
UNIQUE (Cognome,Nome)
```

- Si noti che specificare

```
UNIQUE (Cognome),
UNIQUE (Nome)
```

sarebbe molto più restrittivo (perché?)

Chiavi primarie

- La definizione della chiave primaria di una tabella avviene specificando un vincolo PRIMARY KEY, o in linea o come vincolo di tabella

```
Matricola char(5)      PRIMARY KEY

PRIMARY KEY (Matricola,CodCorso)
```

- Va osservato che:
  - La specifica di una chiave primaria non è obbligatoria
  - Si può specificare al massimo una chiave primaria per tabella
  - Non è necessario specificare NOT NULL per gli attributi della primary key

Chiavi importate (“foreign key”)

- La definizione di una foreign key avviene specificando un vincolo FOREIGN KEY, e indicando quale chiave viene referenziata

```
CodCorso  int   REFERENCES Corsi(CodCorso)
```

- Ovvero, come vincolo di tabella:

```
FOREIGN KEY (CodCorso) REFERENCES Corsi(CodCorso)
```

- Nell’esempio, Esami è detta tabella di riferimento e Corsi tabella di destinazione (analoga terminologia per gli attributi coinvolti)
- Le colonne di destinazione devono essere una chiave della tabella destinazione (non necessariamente la chiave primaria)
- Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria

```
CodCorso  int   REFERENCES Corsi
```

Vincoli generici (“check constraint”)

- Mediante la clausola CHECK è possibile esprimere vincoli di tupla arbitrari, sfruttando tutto il potere espressivo di SQL
- La sintassi è: **`CHECK (<condizione>)`**
- Il vincolo è violato se esiste almeno una tupla che rende falsa la <condizione>. Pertanto

```
Stipendio int          CHECK (Stipendio > 0),
```

non permette tuple con stipendio negativo, ma ammette valori nulli per l’attributo Stipendio (perché non posso essere sicuro che sia negativo!)

- Se CHECK viene espresso a livello di tabella (anziché nella definizione dell’attributo) è possibile fare riferimento a più attributi della tabella stessa

```
CHECK ((Voto = 30) OR (Lode = ‘NO’))
```

Vincoli con nomi

- A fini diagnostici (e di documentazione) è spesso utile sapere quale vincolo è stato violato a seguito di un'azione sul DB
- A tale scopo è possibile dare dei nomi ai vincoli, ad esempio:

```
Voto  int          NOT NULL
                   CONSTRAINT VotoValido
                   CHECK (Voto BETWEEN 18 AND 30),


CONSTRAINT ForeignKeyCorsi
    FOREIGN KEY (CodCorso)  REFERENCES Corso
```

Istruzioni di aggiornamento dei dati

- Le istruzioni che permettono di aggiornare il DB sono

INSERT	inserisce nuove tuple nel DB
DELETE	cancella tuple dal DB
UPDATE	modifica tuple del DB

- INSERT può usare il risultato di una query per eseguire inserimenti multipli
- DELETE e UPDATE possono fare uso di condizioni per specificare le tuple da cancellare o modificare
- In ogni caso gli aggiornamenti riguardano una sola relazione

Inserimento di tuple: caso singolo

- Per inserire una nuova tupla bisogna specificarne i valori, dicendo quale valore va assegnato a quale attributo

```
INSERT INTO Corsi(Titolo,CodCorso,Docente,Anno)
VALUES              ('StoriaAntica',456,'Grigi',3)
```

- Se la lista degli attributi viene omessa vale l'ordine con cui sono stati definiti

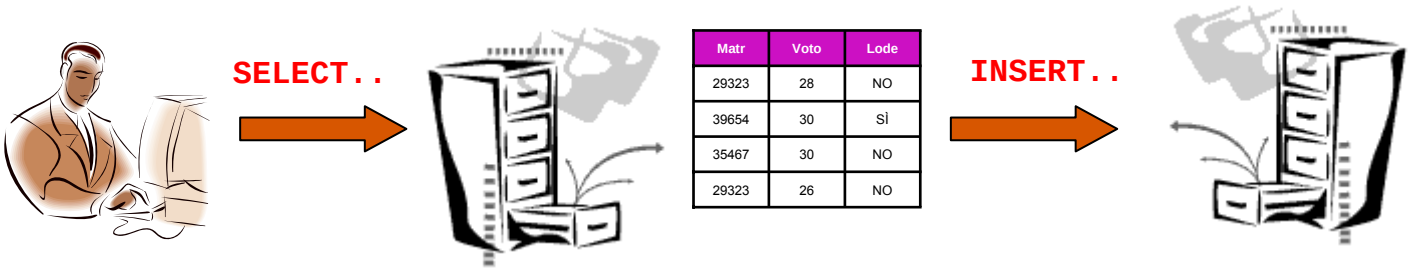
```
INSERT INTO Corsi
VALUES      (456,'StoriaAntica','Grigi',3)
```

- Se la lista non include tutti gli attributi, i restanti assumono valore NULL (se ammesso) o il valore di default (se specificato)

```
INSERT INTO Corsi(CodCorso,Titolo)
VALUES              (456,'StoriaAntica')
```

Inserimento di tuple: caso multiplo

- In alcuni casi si rende necessario inserire in una tabella le tuple che risultano da una query



- Si può fare direttamente! Ad esempio:

```
INSERT INTO StudentiSenzaEmail(Matr,Cog,Nom)
SELECT Matricola,Cognome,Nome
FROM   Studenti
WHERE  Email IS NULL
```

- Gli schemi del risultato e della tabella in cui si inseriscono le tuple possono essere diversi, l'importante è che i tipi delle colonne siano compatibili

Cancellazione di tuple

- L'istruzione DELETE può fare uso di una condizione per specificare le tuple da cancellare

```
DELETE FROM Corsi -- elimina i corsi di Biondi
WHERE  Docente = 'Biondi'
```

- Per cancellare tutte le tuple (attenzione!):

```
DELETE FROM Corsi
```

- Che succede se la cancellazione porta a violare il vincolo di integrità referenziale? (ad es.: che accade agli esami dei corsi di Biondi?)

Modifica di tuple

- Anche l'istruzione UPDATE può fare uso di una condizione per specificare le tuple da modificare e di espressioni per determinare i nuovi valori

```
UPDATE Corsi
SET    Docente = 'Bianchi',
       Anno = 2
WHERE  Docente = 'Biondi'

UPDATE Dipendenti
SET    Stipendio = 1.1*Stipendio -- aumento del 10%
WHERE  Ruolo = 'Programmatore'
```

- Anche l'UPDATE può portare a violare il vincolo di integrità referenziale

Politiche di “reazione”

- Anziché lasciare al programmatore il compito di garantire che a fronte di cancellazioni e modifiche i vincoli di integrità referenziale siano rispettati, si possono specificare opportune politiche di reazione in fase di definizione degli schemi

```
CREATE TABLE Esami (
   Matricola char(5)     NOT NULL,
   CodCorso  int         NOT NULL,
   ...
   FOREIGN KEY CodCorso REFERENCES Corsi
       ON DELETE CASCADE        -- cancellazione in cascata
       ON UPDATE NO ACTION      -- modifiche non permesse
```

- Altre politiche: SET NULL e SET DEFAULT

Interrogazioni: tutto qui?

- Quanto visto sinora ci consente, per una data relazione, di:
  - Definire il suo schema, con tutti i vincoli opportuni
  - Inserire i dati, modificarli e cancellarli
  - Scrivere delle interrogazioni che operano su tale relazione
- Le interrogazioni “interessanti” includono tuttavia almeno altre 2 tipologie notevoli di casi:
- Interrogazioni su più relazioni

  I docenti dei corsi di cui lo studente con matricola 29323 ha sostenuto l’esame
- Interrogazioni di sintesi

  Quanti esami ha verbalizzato ciascun docente?

Interrogazioni su più tabelle (1)

E se volessimo trovare i docenti dei corsi di cui lo studente con matricola 29323 ha sostenuto l'esame?

```
SELECT   CodCorso
FROM     Esami
WHERE    Matricola = '29323'

SELECT   Docente
FROM     Corsi
WHERE    CodCorso IN (483,913)
```

CodCorso
483
913

Docente
Biondi
Castani

- E se lo studente avesse sostenuto 20 esami!? Molto poco pratico!!!
- In più c'è un altro problema…

Interrogazioni su più tabelle (2)

E se volessimo generare un elenco con il seguente formato (schema)?

Matricola	Cognome	Nome	CodCorso	Voto	Lode

- Operando su una singola tabella alla volta non ce la faremmo mai!

Interrogazioni su più tabelle (3)

- Per prima cosa cerchiamo di capire cosa conterrebbe il risultato…

Studenti

Matricola	Cognome	Nome	DataNascita	Email
29323	Bianchi	Giorgio	21/06/1978	gbianchi@alma.unibo.it
35467	Rossi	Anna	13/04/1978	anna.rossi@yahoo.it
39654	Verdi	Marco	20/09/1979	mverdi@mv.com
42132	Neri	Lucia	15/02/1978	lucia78@cs.ucsd.edu

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Matricola	Cognome	Nome	CodCorso	Voto	Lode
29323	Bianchi	Giorgio	483	28	NO
39654	Verdi	Marco	729	30	SÌ
29323	Bianchi	Giorgio	913	26	NO
35467	Rossi	Anna	913	30	NO

Interrogazioni su più tabelle (5)

- Come abbiamo fatto?
  1. Come prima cosa abbiamo preso le tabelle Esami e Studenti
  2. Poi abbiamo "accoppiato" ciascuna tupla di Esami con la corrispondente tupla di Studenti, usando la Matricola
  3. Abbiamo poi mantenuto solo gli attributi che ci interessavano
- I punti 1 e 3 sono semplici da realizzare, e sono molto simili a quello che si fa quando si opera su una relazione sola
- Per il punto 2 dobbiamo esplicitare il "criterio di accoppiamento" che, in termini più tecnici, si chiama condizione di join ("*giunzione*")

Abbiamo eseguito un join di Studenti ed Esami

Matricola	Cognome	Nome	CodCorso	Voto	Lode
29323	Bianchi	Giorgio	483	28	NO
39654	Verdi	Marco	729	30	SÌ
29323	Bianchi	Giorgio	913	26	NO
35467	Rossi	Anna	913	30	NO

Esprimere la condizione di join (1)

- Per il punto 2 dobbiamo esplicitare il "criterio di accoppiamento" che, in termini più tecnici, si chiama condizione di join ("*giunzione*")
- Nel nostro caso il criterio è, a parole

  Accoppia una tupla di Esami con una tupla di Studenti
  se hanno la stessa (uguale) Matricola

- Ma se scriviamo **`Matricola = Matricola`** non significa nulla!

- Perché? Perché il sistema, quando scrivo **`Matricola`**, non sa se intendo quella della tabella Studenti o quella della tabella Esami

Esprimere la condizione di join (2)

Laureati

Matricola	VotoFinale
29323	89
35467	95
39654	102

Concorsi

Codice	VotoFinale
ABC	105
XYZ	88
GHJ	99

- In questo caso il criterio è, a parole

  Accoppia una tupla di Laureati con una tupla di Concorsi se il VotoFinale del primo è maggiore o uguale al VotoFinale minimo richiesto per l'ammissione al concorso stesso

Matricola	Codice
29323	XYZ
35467	XYZ
39654	XYZ
39654	GHJ

- Ma se scriviamo **`VotoFinale >= VotoFinale`** non significa nulla!

Indicare correttamente gli attributi

- Quando lavoriamo su 2 o più tabelle che hanno attributi con lo stesso nome, se vogliamo usare tali attributi nelle interrogazioni dobbiamo indicare esplicitamente la loro tabella, in questo modo:

```
Esami.Matricola
```

- Quindi, negli esempi visti, scriveremmo

```
Esami.Matricola = Studenti.Matricola
```

```
Laureati.VotoFinale >= Concorsi.VotoFinale
```

- Questa cosa si può sempre fare, anche quando non è necessario (ad es. perché si opera su una sola tabella, o se gli attributi hanno nomi diversi)

Pseudonimi per i nomi delle relazioni

- Se le nostre relazioni hanno nomi lunghi è noioso scriverli per esteso ogni volta
- E' possibile usare degli pseudonimi, o alias, più brevi, che si inseriscono nella clausola FROM (tipicamente l'iniziale del nome, i primi caratteri,ecc.)

```
SELECT ...
FROM   Esami E ...
WHERE  E.Matricola ...
```

- Adesso abbiamo tutto quello che ci serve…

…e finalmente…

1. Come prima cosa prendiamo le tabelle Esami e Studenti

```
FROM    Esami E, Studenti S
```

2. Poi “accoppiamo” ciascuna tupla di Esami con la corrispondente tupla di Studenti, usando la Matricola

```
WHERE   E.Matricola = S.Matricola
```

3. Infine manteniamo solo gli attributi che ci interessano

```
SELECT  S.Matricola,S.Cognome,S.Nome,E.Voto,E.Lode
```

```
SELECT  S.Matricola,S.Cognome,S.Nome,E.Voto,E.Lode
FROM    Esami E, Studenti S
WHERE   E.Matricola = S.Matricola
```

Altri esempi (1)

i numeri di matricola degli studenti
che hanno sostenuto l'esame di Analisi con il Prof. Biondi

```
SELECT  E.Matricola
FROM    Corsi C, Esami E
WHERE   C.CodCorso = E.CodCorso
AND     C.Titolo = 'Analisi'
AND     C.Docente = 'Biondi'
```

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Matricola	CodCorso	Voto	Lode	Titolo	Docente	Anno
29323	483	28	NO	Analisi	Biondi	1
39654	729	30	SÌ	Analisi	Neri	1
29323	913	26	NO	Sistemi Informativi	Castani	2
35467	913	30	NO	Sistemi Informativi	Castani	2

Altri esempi (2)

i docenti dei corsi di cui lo studente con matricola 29323 ha sostenuto l'esame

```
SELECT  C.Docente
FROM    Corsi C, Esami E
WHERE   C.CodCorso = E.CodCorso
AND     E.Matricola = '29323'
```

CodCorso	Titolo	Docente	Anno
483	Analisi	Biondi	1
729	Analisi	Neri	1
913	Sistemi Informativi	Castani	2

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	SÌ
29323	913	26	NO
35467	913	30	NO

Matricola	CodCorso	Voto	Lode	Titolo	Docente	Anno
29323	483	28	NO	Analisi	Biondi	1
~~39654~~	~~729~~	~~30~~	~~SÌ~~	~~Analisi~~	~~Neri~~	~~1~~
29323	913	26	NO	Sistemi Informativi	Castani	2
~~35467~~	~~913~~	~~30~~	~~NO~~	~~Sistemi Informativi~~	~~Castani~~	~~2~~

Più di 1 tabella = 2,3,4,…!

- Quanto fatto con 2 tabelle si può generalizzare al caso di 3 o più tabelle

i docenti dei corsi di cui lo studente Giorgio Bianchi ha sostenuto l'esame

```
SELECT C.Docente
FROM   Corsi C, Esami E, Studenti S
WHERE  C.CodCorso = E.CodCorso
AND    E.Matricola = S.Matricola
AND    S.Cognome = 'Bianchi'
AND    S.Nome = 'Giorgio'
```

Self Join

- In alcuni casi è necessario fare il join di una tabella con se stessa
- Esseenziale fare uso di alias

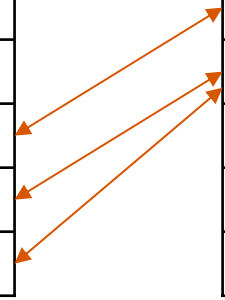
Chi sono i nonni di Anna?

Genitori G1

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

Genitori G2

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria



```
SELECT   G1.Genitore AS Nonno
FROM     Genitori G1, Genitori G2
WHERE    G1.Figlio = G2.Genitore
  AND    G2.Figlio = 'Anna'
```

Nonno
Giorgio
Silvia
Enzo

Join espliciti

- Anziché scrivere i predicati di join nella clausola WHERE, è possibile "costruire" una *joined table* direttamente nella clausola FROM

```
SELECT   S.*, E.CodCorso, E.Voto, E.Lode
FROM     Studenti S JOIN Esami E
                    ON (S.Matricola = E.Matricola)
WHERE    E.Voto > 26
```

  in cui JOIN si può anche scrivere INNER JOIN

- Altri tipi di join espliciti sono:

```
LEFT [OUTER] JOIN
RIGHT [OUTER] JOIN
FULL [OUTER] JOIN
```

Outer join

- L'outer join è un tipo di join che permette di restituire anche le tuple che non soddisfano mai la condizione di join

```
SELECT   *
FROM     Studenti S LEFT JOIN Esami E
                    ON (S.Matricola = E.Matricola)
```

restituisce anche gli studenti senza esami (quindi tutti)

Qualche esercizio…

- Proviamo a scrivere le istruzioni SQL che restituiscono:

1) Gli indirizzi di posta elettronica degli studenti che hanno preso 30 e lode nei corsi di Analisi
2) Il nome e cognome degli studenti che hanno sostenuto almeno un esame (senza duplicati)

Operatori insiemistici

- L'istruzione SELECT non permette di eseguire unione, intersezione e differenza di tabelle
- Ciò che si può fare è combinare in modo opportuno i risultati di due istruzioni SELECT, mediante gli operatori

`UNION, INTERSECT, EXCEPT`

- In tutti i casi gli elementi delle SELECT list devono avere tipi compatibili e gli stessi nomi se si vogliono colonne con un'intestazione definita
- L'ordine degli elementi è importante (notazione posizionale)
- Il risultato è in ogni caso privo di duplicati, per mantenerli occorre aggiungere l'opzione ALL:

`UNION ALL, INTERSECT ALL, EXCEPT ALL`

Operatori insiemistici: esempi (1)

R

A	B
1	a
1	a
2	a
2	b
2	c
3	b

S

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT A
FROM R
UNION
SELECT C
FROM S
```

1
2
3
4

```
SELECT B
FROM R
UNION ALL
SELECT B
FROM S
```

B
a
a
a
b
c
b
a
b
a
c
c
d

```
SELECT A
FROM R
UNION
SELECT C AS A
FROM S
```

A
1
2
3
4

```
SELECT A,B
FROM R
UNION
SELECT B,C AS A
FROM S
```

Non corretta!

Operatori insiemistici: esempi (2)

R

A	B
1	a
1	a
2	a
2	b
2	c
3	b

S

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT B
FROM R
INTERSECT
SELECT B
FROM S
```

B
a
b
c

```
SELECT B
FROM S
EXCEPT
SELECT B
FROM R
```

B
d

```
SELECT B
FROM R
INTERSECT ALL
SELECT B
FROM S
```

B
a
a
b
c

```
SELECT B
FROM R
EXCEPT ALL
SELECT B
FROM S
```

B
a
b

Informazioni di sintesi

- Quanto sinora visto permette di estrarre dal DB informazioni che si riferiscono a singole tuple (eventualmente ottenute mediante operazioni di join)

  Esempio: gli esami dello studente con matricola 29323, i nomi degli studenti che hanno sostenuto un esame con il prof. Biondi, ecc.

- In molti casi è viceversa utile ottenere dal DB informazioni (di sintesi) che caratterizzano "gruppi" di tuple

  Esempio: il numero di esami sostenuti dallo studente con matricola 29323, la media dei voti degli esami del primo anno, ecc.

- A tale scopo SQL mette a disposizione due strumenti di base:
  - Funzioni aggregate
  - Clausola di raggruppamento (GROUP BY)

Un nuovo DB per gli esempi…

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Prog

CodProg	Citta
P01	Milano
P01	Bologna
P02	Bologna

Funzioni aggregate (1)

- Lo standard SQL mette a disposizione una serie di funzioni aggregate (o "di colonna"):
  - MIN minimo
  - MAX massimo
  - SUM somma
  - AVG media aritmetica
  - STDEV deviazione standard
  - VARIANCE varianza
  - COUNT contatore

```
SELECT   SUM(Stipendio) AS ToTStipS01
FROM     Imp
WHERE    Sede = 'S01'
```

TotStipS01
5100

Funzioni aggregate (2)

- L'argomento di una funzione aggregata è una qualunque espressione che può figurare nella SELECT list (ma non un'altra funzione aggregata!)

```
SELECT   SUM(Stipendio*12) AS ToTStipAnnuiS01
FROM     Imp
WHERE    Sede = 'S01'
```

TotStipAnnuiS01
61200

- Tutte le funzioni, ad eccezione di COUNT, ignorano i valori nulli
- Il risultato è NULL se tutti i valori sono NULL
- L'opzione DISTINCT considera solo i valori distinti

```
SELECT   SUM(DISTINCT Stipendio)
FROM     Imp
WHERE    Sede = 'S01'
```

4100

COUNT e valori nulli

- La forma COUNT(*) conta le tuple del risultato; viceversa, specificando una colonna, si omettono quelle con valore nullo in tale colonna

Imp

CodImp	Sede	…	Stipendio
E001	S01		2000
E002	S02		1500
E003	S01		1000
E004	S03		NULL
E005	S02		2500
E006	S01		NULL
E007	S01		1000
E008	S02		1200

```
SELECT   COUNT(*) AS NumImpS01
FROM     Imp
WHERE    Sede = 'S01'
```

NumImpS01
4

```
SELECT   COUNT(Stipendio)
                  AS NumStipS01
FROM     Imp
WHERE    Sede = 'S01'
```

NumStipS01
3

Clausola SELECT e funzioni aggregate

- Se si usano funzioni aggregate, la SELECT list non può includere altri elementi che non siano a loro volta funzioni aggregate

```
SELECT   Nome, MIN(Stipendio)
FROM     Imp
```

non va bene!
(viceversa, `SELECT MIN(Stipendio), MAX(Stipendio)..` è corretto)

- Il motivo è che una funzione aggregata restituisce un singolo valore, mentre il riferimento a una colonna è in generale un insieme di valori (eventualmente ripetuti)

Funzioni aggregate e raggruppamento

- I valori di sintesi calcolati dalle funzioni aggregate si riferiscono a tutte le tuple che soddisfano le condizioni delle clausola WHERE
- In molti casi è viceversa opportuno fornire tali valori per gruppi omogenei di tuple (es: impiegati di una stessa sede)
- La clausola GROUP BY serve a definire tali gruppi, specificando una o più colonne (di raggruppamento) sulla base della/e quale/i le tuple sono raggruppate per valori uguali

```
SELECT   Sede, COUNT(*) AS NumProg
FROM     Imp
WHERE    Ruolo = 'Programmatore'
GROUP BY Sede
```

Sede	NumProg
S01	2
S03	1
S02	1

- La SELECT list può includere le colonne di raggruppamento, ma non altre!

Come si ragiona con il GROUP BY

- Le tuple che soddisfano la clausola WHERE…

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

- …sono raggruppate per valori uguali della/e colonna/e presenti nella clausola GROUP BY…

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	**S01**	Programmatore	1000
E007	Violetti	**S01**	Programmatore	1000
E004	Gialli	**S03**	Programmatore	1000
E008	Aranci	**S02**	Programmatore	1200

- …e infine a ciascun gruppo si applica la funzione aggregata

Sede	NumProg
S01	2
S03	1
S02	1

GROUP BY: esempi

1) Per ogni ruolo, lo stipendio medio nelle sedi di Milano

```
SELECT   I.Ruolo, AVG(I.Stipendio) AS AvgStip
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Ruolo
```

Ruolo	AvgStip
Analista	2000
Sistemista	1100
Programmatore	1000

2) Per ogni sede di Milano, lo stipendio medio

```
SELECT   I.Sede, AVG(I.Stipendio) AS AvgStip
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Sede
```

Sede	AvgStip
S01	1275
S03	1000

3) Per ogni ruolo e sede di Milano, lo stipendio medio

```
SELECT   I.Sede, I.Ruolo, AVG(I.Stipendio)
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Sede, I.Ruolo
```

Ruolo	Sede	
Analista	S01	2000
Sistemista	S01	1100
Programmatore	S01	1000
Programmatore	S03	1000

Raggruppamento e proiezione

- Quando la SELECT list include solo le colonne di raggruppamento, il tutto è equivalente a ciò che si otterrebbe omettendo il GROUP BY e rimuovendo i duplicati con l'opzione DISTINCT

```
SELECT   Sede
FROM     Imp
GROUP BY Sede
```

Sede
S01
S02
S03

equivale pertanto a

```
SELECT   DISTINCT Sede
FROM     Imp
```

Condizioni sui gruppi

- Oltre a poter formare dei gruppi, è anche possibile selezionare dei gruppi sulla base di loro proprietà “complessive”

```
SELECT   Sede, COUNT(*) AS NumImp
FROM     Imp
GROUP BY Sede
HAVING   COUNT(*) > 2
```

Sede	NumImp
S01	4
S02	3

- La clausola HAVING ha per i gruppi una funzione simile a quella che la clausola WHERE ha per le tuple (attenzione a non confonderle!)

Tipi di condizioni sui gruppi

- Nella clausola HAVING si possono avere due tipi di condizioni:
  - Condizioni che fanno uso di funzioni aggregate (es. `COUNT(*) > 2`)
  - Condizioni che si riferiscono alle colonne di raggruppamento
    - Queste ultime si possono anche inserire nella clausola WHERE

```
SELECT   Sede, COUNT(*) AS NumImp
FROM     Imp
GROUP BY Sede
HAVING   Sede <> 'S01'
```

equivale a

```
SELECT   Sede, COUNT(*) AS NumImp
FROM     Imp
WHERE    Sede <> 'S01'
GROUP BY Sede
```

Sede	NumImp
S02	3
S03	1

Un esempio completo

Per ogni sede di Bologna in cui il numero di impiegati è almeno 3, si vuole conoscere il valor medio degli stipendi, ordinando il risultato per valori decrescenti di stipendio medio e quindi per sede

```
SELECT   I.Sede, AVG(Stipendio) AS AvgStipendio
FROM     Imp I, Sedi S
WHERE    I.Sede = S.Sede
  AND    S.Citta = 'Bologna'
GROUP BY I.Sede
HAVING   COUNT(*) >= 3
ORDER BY AvgStipendio DESC, Sede
```

- L'ordine delle clausole è sempre come nell'esempio
- Si ricordi che il GROUP BY non implica alcun ordinamento del risultato

Subquery

- Oltre alla forma "flat" vista sinora, in SQL è anche possibile esprimere delle condizioni che si basano sul risultato di altre interrogazioni (subquery, o query innestate o query nidificate)

```
SELECT   CodImp    -- impiegati delle sedi di Milano
FROM     Imp
WHERE    Sede IN (SELECT   Sede
                  FROM     Sedi
                  WHERE    Citta = 'Milano')
```

Sede
S01
S03

- La subquery restituisce l'insieme di sedi ('S01','S03'), e quindi il predicato nella clausola WHERE esterna equivale a

```
WHERE    Sede IN ('S01','S03')
```

Subquery scalari

- Gli operatori di confronto =, <,… si possono usare solo se la subquery restituisce non più di una tupla (subquery "scalare")

```
SELECT   CodImp    -- impiegati con stipendio minimo
FROM     Imp
WHERE    Stipendio = (SELECT   MIN(Stipendio)
                      FROM     Imp)
```

- La presenza di vincoli può essere sfruttata a tale scopo

```
SELECT   Responsabile
FROM     Sedi
WHERE    Sede = (SELECT Sede -- al massimo una sede
                 FROM   Imp
                 WHERE  CodImp = 'E001')
```

Subquery: caso generale

- Se la subquery può restituire più di un valore si devono usare le forme
  - <op> ANY: la relazione <op> vale per almeno uno dei valori
  - <op> ALL : la relazione <op> vale per tutti i valori

```
SELECT   Responsabile
FROM     Sedi
WHERE    Sede = ANY (SELECT   Sede
                     FROM     Imp
                     WHERE    Stipendio > 1500)

SELECT   CodImp    -- impiegati con stipendio minimo
FROM     Imp
WHERE    Stipendio <= ALL (SELECT   Stipendio
                           FROM     Imp)
```

- La forma = ANY equivale a IN

Subquery: livelli multipli di innestamento

- Una subquery può fare uso a sua volta di altre subquery. Il risultato si può ottenere risolvendo a partire dal blocco più interno

```
SELECT CodImp
FROM   Imp
WHERE  Sede IN (SELECT Sede
                FROM   Sedi
                WHERE  Citta NOT IN (SELECT Citta
                                     FROM   Prog
                                     WHERE  CodProg = 'P02'))
```

- Attenzione a non sbagliare quando ci sono negazioni! Nell'esempio, i due blocchi interni non sono equivalenti a:

```
WHERE  Sede IN (SELECT Sede
                FROM   Sedi, Prog
                WHERE  Sedi.Citta <> Prog.Citta
                  AND  Prog.CodProg = 'P02')
```

Subquery: quantificatore esistenziale

- Mediante EXISTS (SELECT * …) è possibile verificare se il risultato di una subquery restituisce almeno una tupla

```
SELECT   Sede
FROM     Sedi S
WHERE    EXISTS  (SELECT   *
                  FROM     Imp
                  WHERE    Ruolo = 'Programmatore')
```

- Facendo uso di NOT EXISTS il predicato è vero se la subquery non restituisce alcuna tupla
- In entrambi i casi la cosa non è molto "interessante" in quanto il risultato della subquery è sempre lo stesso, ovvero non dipende dalla specifica tupla del blocco esterno

Subquery correlate

- Se la subquery fa riferimento a “variabili” definite in un blocco esterno, allora si dice che è correlata

```
SELECT   Sede      -- sedi con almeno un programmatore
FROM     Sedi S
WHERE    EXISTS  (SELECT   *
                  FROM     Imp
                  WHERE    Ruolo = 'Programmatore'
                    AND    Sede = S.Sede)
```

- Adesso il risultato della query innestata dipende dalla sede specifica, e la semantica quindi diventa:

  Per ogni tupla del blocco esterno, considera il valore di S.Sede e risolvi la query innestata

Subquery: “unnesting” (1)

- È spesso possibile ricondursi a una forma “piatta”, ma la cosa non è sempre così ovvia. Ad esempio, nell’esempio precedente si può anche scrivere

```
SELECT   DISTINCT Sede
FROM     Sedi S, Imp I
WHERE    S.Sede = I.Sede
  AND    I.Ruolo = ‘Programmatore’
```

- Si noti la presenza del DISTINCT
- La forma innestata è “più procedurale” di quella piatta e, a seconda dei casi, può risultare più semplice da derivare
- Si ricordi comunque che in una subquery non si possono usare operatori insiemistici (UNION, INTERSECT e EXCEPT) e che una subquery può comparire solo come operando destro in un predicato

Subquery: “unnesting” (2)

- Con la negazione le cose tendono a complicarsi. Ad esempio, per trovare le sedi senza programmatori, nella forma innestata basta sostituire NOT EXISTS a EXISTS, ma nella forma piatta:

```
SELECT DISTINCT Sede
FROM   Sedi S LEFT OUTER JOIN Imp I ON
                (S.Sede = I.Sede) AND (I.Ruolo = ‘Programmatore’)
WHERE  I.CodImp IS NULL
```

- È facile sbagliare, ad esempio la seguente query non è corretta

```
SELECT DISTINCT Sede
FROM   Sedi S LEFT OUTER JOIN Imp I ON (S.Sede = I.Sede)
WHERE  I.Ruolo = ‘Programmatore’
  AND  I.CodImp IS NULL
```

perché la clausola WHERE non è mai soddisfatta!

Subquery: aggiornamento dei dati

- Le subquery si possono efficacemente usare per aggiornare i dati di una tabella sulla base di criteri che dipendono dal contenuto di altre tabelle

```
DELETE FROM Imp -- elimina gli impiegati di Bologna
WHERE  Sede IN (SELECT Sede
                FROM   Sedi
                WHERE  Citta = 'Bologna')

UPDATE Imp
SET    Stipendio = 1.1*Stipendio
WHERE  Sede IN (SELECT S.Sede
                FROM   Sede S, Prog P
                WHERE  S.Citta = P.Citta
                  AND  P.CodProg = 'P02')
```

Subquery e CHECK

- Facendo uso di subquery nella clausola CHECK è possibile esprimere vincoli arbitrariamente complessi

  Ogni sede deve avere almeno due programmatori

```
... -- quando si crea la TABLE Sedi
CHECK (2 <= (SELECT COUNT(*) FROM Imp I
             WHERE  I.Sede = Sede -- correlazione
               AND  I.Ruolo = 'Programmatore'))
```

  Supponendo di avere due tabelle ImpBO e ImpMI e di volere che uno stesso codice (CodImp) non sia presente in entrambe le tabelle:

```
... -- quando si crea la TABLE ImpBO
CHECK (NOT EXISTS (SELECT * FROM ImpMI
                   WHERE  ImpMI.CodImp = CodImp))
```

Definizione di viste

- Mediante l'istruzione CREATE VIEW si definisce una vista, ovvero una "tabella virtuale"
- Le tuple della vista sono il risultato di una query che viene calcolato dinamicamente ogni volta che si fa riferimento alla vista

```
CREATE VIEW ProgSedi(CodProg,CodSede)
AS     SELECT P.CodProg,S.Sede
       FROM   Prog P, Sedi S
       WHERE  P.Citta = S.Citta
```

ProgSedi

CodProg	CodSede
P01	S01
P01	S03
P01	S02
P02	S02

```
SELECT *
FROM   ProgSedi
WHERE  CodProg = 'P01'
```

CodProg	CodSede
P01	S01
P01	S03
P01	S02

Uso delle viste

- Le viste possono essere create a vari scopi, tra i quali si ricordano i seguenti:
  - Permettere agli utenti di avere una visione personalizzata del DB, e che in parte astragga dalla struttura logica del DB stesso
  - Far fronte a modifiche dello schema logico che comporterebbero una ricompilazione dei programmi applicativi
  - Semplificare la scrittura di query complesse
- Inoltre le viste possono essere usate come meccanismo per il controllo degli accessi, fornendo ad ogni classe di utenti gli opportuni privilegi
- Si noti che nella definizione di una vista si possono referenziare anche altre viste

Indipendenza logica tramite VIEW

- A titolo esemplificativo si consideri un DB che contiene la tabella

  EsamiSI(Matr,Cognome,Nome,DataProva,Voto)

- Per evitare di ripetere i dati anagrafici, si decide di modificare lo schema del DB sostituendo alla tabella **EsamiSI** le due seguenti:

  StudentiSI(Matr,Cognome,Nome)

  ProveSI(Matr,DataProva,Voto)

- È possibile ripristinare la "visione originale" in questo modo:

```
CREATE VIEW EsamiSI(Matr,Cognome,Nome,DataProva,Voto)
AS     SELECT S.*,P.DataProva,P.Voto
       FROM   StudentiSI S, ProveSI P
       WHERE  S.Matr = P.Matr
```

Aggiornamento di viste

- Le viste possono essere utilizzate per le interrogazioni come se fossero tabelle del DB, ma per le operazioni di aggiornamento ci sono dei limiti

```
CREATE VIEW NumImpSedi(Sede,NumImp)
AS     SELECT    Sede,COUNT(*)
       FROM      Imp
       GROUP BY  Sede

UPDATE NumImpSedi
SET    NumImp = NumImp + 1
WHERE  Sede = 'S03'
```

NumImpSedi

Sede	NumImp
S01	4
S02	3
S03	1

- Cosa significa? Non si può fare!
- In generale, ogni DBMS pone dei limiti su quelle che sono le viste aggiornabili

Viste non aggiornabili

- Le più comuni restrizioni riguardano la non aggiornabilità di viste definite mediante:
  - GROUP BY
  - Funzioni aggregate
  - DISTINCT
  - join
- Intuitivamente, una vista è aggiornabile se il DBMS riesce a stabilire come le relazioni del DB devono essere modificate a fronte di una modifica della vista

Viste con CHECK OPTION

- Per le viste aggiornabili si presenta un nuovo problema. Si consideri il seguente inserimento nella vista **ImpBO**

```
INSERT INTO ImpBO(CodImp,Nome,Sede,Ruolo,Stipendio)
VALUES ('E009','Azzurri','S03','Analista',1800)
```

  in cui il valore di Sede (**'S03'**) non rispetta la specifica della vista. Ciò comporta che una successiva query su **ImpBO** non restituirebbe la tupla appena inserita (!?)

- Per evitare situazioni di questo tipo, all'atto della creazione di una vista si può specificare, facendola seguire alla query che definisce la vista, la clausola **WITH CHECK OPTION**, che garantisce che ogni tupla inserita nella vista sia anche restituita dalla vista stessa

Tipi di CHECK OPTION

- Se la vista V1 è definita in termini di un'altra vista V2, e si specifica la clausola **WITH CHECK OPTION**, il DBMS verifica che la nuova tupla t inserita soddisfi sia la definizione di V1 che quella di V2, indipendentemente dal fatto che V2 sia stata a sua volta definita **WITH CHECK OPTION**

- Questo comportamento di default, che è equivalente a definire V1

  WITH CASCADED CHECK OPTION

  si può alterare definendo V1

  WITH LOCAL CHECK OPTION

- In modalità **LOCAL**, il DBMS verifica solo che t soddisfi la specifica di V1

Esempio

```
CREATE TABLE R (A INT)  CREATE VIEW V1
                        AS SELECT * FROM R WHERE A < 2
                        WITH CHECK OPTION

CREATE VIEW V2
AS     SELECT * FROM V1 WHERE A > 0
WITH LOCAL CHECK OPTION

CREATE VIEW V3
AS     SELECT * FROM V1 WHERE A > 0
WITH CASCADED CHECK OPTION

INSERT INTO V2(2)  -- OK
INSERT INTO V3(2)  -- Errore su V1
```

Supporto di SQL nei sistemi F/OS

- Il supporto di SQL nei sistemi F/OS (ma non solo!) è non omogeneo, e varia da una versione all'altra
- Considerando le ultime versioni dei due sistemi di riferimento (MySQL e PostgreSQL), si può tuttavia asserire che queste sono comparabili per ciò che riguarda le caratteristiche sinora viste, e che le principali differenze vanno invece ricercate in aspetti che si riferiscono a:
  - Organizzazione fisica dei dati (es. partizionamento)
  - Prestazioni (inclusa l'ottimizzazione delle interrogazioni)
  - Tool di amministrazione

MySQL

- A titolo di esempio, si riportano alcune delle caratteristiche di base di SQL che sono state introdotte solo nelle ultime versioni di MySQL

  (fonte: MySQL 5.1 Reference Manual,
  http://downloads.mysql.com/docs/refman-5.1-en.a4.pdf)

Feature	MySQL Series
UNION	4.0
Subqueries	4.1
Views	5.0
Foreign keys	5.2 (3.23 per InnoDB tables)

  - InnoDB table: tabella memorizzata in una struttura che permette il supporto di transazioni ACID e il lock a livello di tupla (lezione 2), a fronte di una riduzione di prestazioni
  - Il default è avere table di tipo MyISAM, che garantiscono le migliori prestazioni in lettura, ma rinunciano alle caratteristiche di cui sopra

PostgreSQL

- Per PostgresSQL un elenco come il precedente non è facilmente ottenibile. Per contro, le seguenti sono alcune delle lacune rispetto al "core" dello standard SQL:1999

  (fonte: PostgreSQL 8.2.0 Documentation,

  http://www.postgresql.org/files/documentation/pdf/8.2/postgresql-8.2-A4.pdf)

Feature
Subqueries in CHECK
Views with CHECK OPTION

  - Va tuttavia notato che PostgreSQL mette a disposizione un potente sistema di regole con cui è possibile sopperire (anche se in modo non standard) a tali lacune e a molte altre ancora