



Basi di Dati

## BASE DI DATI

Ogni organizzazione è dotata di un SISTEMA INFORMATIVO che gestisce e organizza le informazioni necessarie all'organizzazione.

Per indicare la parziale AUTOMATIZZAZIONE del sistema, di solito viene utilizzato il termine **SISTEMA INFORMATICO**.

le informazioni vengono rappresentate per mezzo di **DATI**.  
dato che consente di avere  
conoscenza esatta dei fatti

elementi di informazione costituiti da  
simboli che devono essere elaborati.



(i dati da soli non hanno alcun significato)

➡ **BASE DI DATI**: è una collezione di dati, utilizzati per rappresentare le informazioni per un sistema informatico.

DBMS: Un **DATA BASE MANAGEMENT SYSTEM** è un sistema software in grado di gestire collezioni di dati:  
• GRANDI,  
assicurando la loro:  
• AFFIDABILITÀ,  
• EFFICIENTE,  
• PRIVATEZZA,  
• EFFICACE.

• CONDIVISE,  
• PERSISTENTI;

• **GRANDI**: possono avere grandissime dimensioni, molto più grandi della memoria centrale disponibile, infatti devono prevedere una gestione dei dati in memoria secondaria. I sistemi devono poter gestire i dati senza pose limite alle dimensioni (l'unica limite è la dimensione fisica del dispositivo)

• **CONDIVISE**: applicazioni e utenti devono poter accedere, secondo opportune modalità, a dati comuni. In questo modo si riduce la RIDONDANZA dei dati, si evitano ripetizioni e si riduce la possibilità di INCONSENSENZE (se esistono copie degli stessi dati è possibile che in qualche momento non siano uguali)

• **PERSISTENTI**: hanno un tempo di vita che non è limitato a quello delle singole esecuzioni dei programmi che le utilizzano.  
(i dati gestiti da un programma in memoria centrale hanno una vita che inizia e termina con l'esecuzione del programma)

• **AFFIDABILITÀ**: capacità del sistema di conservare intatto il contenuto della base di dati (o di permetterne la ricostruzione) in caso di malfunzionamenti hardware e software.

• **PRIVATEZZA**: ciascun utente, opportunamente riconosciuto, viene abilitato a svolgere determinate azioni sui dati, attraverso meccanismi di AUTORIZZAZIONE.

• **EFFICIENTI**: capaci di svolgere le operazioni utilizzando un insieme di risorse (tempo e spazio) accettabile per gli utenti.  
(dipende dall'implementazione del DBMS e i DBMS forniscono un ampio insieme di funzionalità richiedendo molte risorse e garantiscono l'efficienza solo se il sistema informatico su cui è installato è adeguato)

• **EFFICACI**: capaci di rendere produttive le attività dei loro utenti.

• **TRANSAZIONE**: sequenza ordinata di operazioni da considerare INVIOVISIBILE (atomico),

corretto anche in presenza di concorrenza e con effetti definitivi. Le transazioni concorrenti devono essere coerenti e i risultati sono permanenti.

Es (atomico). "Spostare i soldi dal conto bancario A a B: o il ritiro viene effettuato su A e trasferito su B oppure nessuna operazione viene eseguita."

Es (concorrente). "Se due agenzie diverse vogliono prenotare lo stesso (disponibile) posto su un treno, esso non deve essere prenotato due volte."

## DBMS VS FILE SYSTEM

da gestione di collezioni di dati è possibile anche con sistemi più semplici, come il **FILE SYSTEM**, presente in tutti i sistemi operativi. I file gestiscono insiemi di dati "localmente" a una specifica procedura o applicazione. I DBMS estendono

le funzioni del file system, fornendo l'accesso condiviso agli stessi dati da parte di più utenti e applicazioni, garantendo inoltre altri servizi e utilizzando i file con un'organizzazione dei dati più sofisticata.

### RAPPRESENTAZIONE DEI DATI

I dati vengono rappresentati a livelli diversi, permettendo così l'INDIPENDENZA dei dati dalla loro rappresentazione fisica.

Ogni software (che non utilizza DBMS) contiene una diversa rappresentazione dei dati, causando così incoerenza nella loro rappresentazione. Mentre in DBMS una porzione del database contiene una descrizione centralizzata dei dati ed è utilizzata tra tutte le applicazioni software.

→ Ogni programma può interpretare, ad alto livello, i dati in modo diverso, indipendentemente da come sono rappresentati a basso livello: **MODELLO DEI DATI**

insieme di costrutti utilizzati per organizzare i dati di interesse e descrivere la dinamica.

→ **MODELLO RELAZIONALE**: permette di definire tipi per mezzo dei costrutti RELAZIONE, che consente di organizzare i dati in insiemi di record a struttura fissa.

viene rappresentato per mezzo di una TABELLA

l'ordine è irrilevante { RIGHE: rappresentano specifici record

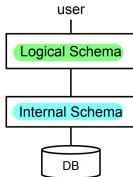
COLONNE: corrispondono ai campi del record

Nella base di dati troviamo:  
• **SCHEMA**, parte invariante nel tempo, costituita dalle caratteristiche dei dati (l'intestazione tabella)  
• **ISTANZA**, parte variabile nel tempo, costituita dai valori effettivi (corpo della tabella)

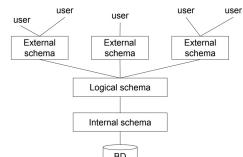
da notare che il modello può essere ulteriormente sviluppato tenendo presenti diversi LIVELLI DI ASTRATTO nella progettazione del database:

- **MODELLO CONCETUALE**: livello più alto di astrazione, rappresenta la struttura dei dati in modo indipendente dal DBMS utilizzato per implementarlo. Descrive cosa rappresentano i dati e COME SONO COLLEGATI tra loro, utilizzando un linguaggio comprensibile. Utilizzato nelle fasi preliminari di progettazione, il più diffuso è il modello **ENTITY-RELATIONSHIP (ER)**.
- **MODELLO LOGICO**: è il livello successivo di astrazione e rappresenta il database in modo più dettagliato, ma ancora indipendentemente dalle specifiche tecnologie di implementazione. Si basa sul modello CONCETUALE e definisce come i dati saranno strutturati nel database, senza tenere conto delle limitazioni fisiche o delle funzionalità specifiche del DBMS. (es. modello RELAZIONALE).

### ARCHITETTURA DBMS



- **SCHEMA LOGICO**: descrizione dell'intera base di dati nel modello logico principale del DBMS. (struttura tabella)
- **SCHEMA INTERNO (FISICO)**: rappresenta lo schema logico a livello di archiviazione, utilizzando specifiche strutture dati (guzzi), come file, records e data pointers.



### INDIPENDENZA DEI DATI

d'accesso ai dati avviene solo tramite il livello esterno

• **INDIPENDENZA FISICA**: il livello logico e quello esterno sono indipendenti da quello fisico: una relazione è utilizzata nello stesso modo qualunque sia la realizzazione fisica. (DBMS)

- **INDIPENDENZA LOGICA**: il livello esterno è indipendente da quello logico. Aggiunte o modifiche non richiedono modifiche a livello logico.

(tabelle di base)

## LINGUAGGI PER BASI DI DATI

Un altro aspetto che rende efficaci le basi di dati è la sua disponibilità attraverso diversi linguaggi ed interfacce:

- **SQL**: (STRUCTURED QUERY LANGUAGE), linguaggio testuale interattivo;
- **DML**: (DATA MANIPULATION LANGUAGE), per l'interrogazione e l'aggiornamento di istanze nella base di dati;
- **DDL**: (DATA DEFINITION LANGUAGE), per la definizione di schemi (logici, esterni, fisici) e altre operazioni generali.

## PERSONAGGI E INTERPRETI

Categorie di persone che interagiscono con una base di dati o DBMS:

- **DATABASE ADMINISTRATOR**: (DBA) l'amministratore della base di dati è:
  - responsabile della progettazione, controllo e amministrazione del DB;
  - media le esigenze espresse dagli utenti, garantendo un controllo CENTRALIZZATO sui dati;
  - garantisce sufficienti prestazioni, assicura l'affidabilità del sistema e gestisce le autorizzazioni di accesso ai dati
- **PROGETTISTI E PROGRAMMATORI**: definiscono e realizzano i programmi che accedono alla base di dati utilizzando DML, oppure altri strumenti di supporto alla generazione di interfacce.
- **UTENTI**: utilizzano la base di dati per le proprie attività, possono essere divisi in due categorie:
  - **UTENTI FINALI**: utilizzano TRANSAZIONI, cioè programmi che realizzano attività predefinite e di frequenza elevata (previste a priori)
  - **UTENTI CASUALI**: utilizzano linguaggi interattivi per l'accesso al DB ed eseguono operazioni non previste (es. interrogazioni e aggiornamenti)

## VANTAGGI E SVANTAGGI DEI DBMS

### PROS

- permettono di considerare i dati come una risorsa comune a disposizione di tutte le componenti dell'organizzazione
- Disponibilità di SERVIZI INTEGRATI (insieme di funzionalità e strumenti offerto dal DBMS)
- Controllo CENTRALIZZATO dei dati (sono raccolti, gestiti e mantenuti in un'unica posizione logica o fisica)
- La condivisione permette di ridurre RIBONDANZE e INCONSENSENZE.
- d'INDIPENDENZA DEI DATI: favorisce lo sviluppo di applicazioni più flessibili e facilmente modificabili

### CONS

- Sono costosi e complessi.
- Formiscono, in forma INTEGRATA, una serie di servizi associati a un costo, scardinale quelli non necessari da quelli richiesti comporta una riduzione delle prestazioni

## MODELLO logici

- Tre modelli logici TRADIZIONALI:
  - MODELLO GERARCHICO

• NETWORK



• RELAZIONALE → basato sul concetto matematico di RELAZIONE. Rappresentazione per mezzo di tabelle.

Indipendenza dei dati

- Altri modelli logici recenti:
  - OBJECT ORIENTED

• XML

- RELAZIONE MATEMATICA = RELAZIONE + RELATIONSHIP

↓  
Rappresenta una classe di fatti, nel modello ENTITY-RELATIONSHIP (ASSOCIAZIONE)

↓  
secondo il modello relazionale dei dati

come nella teoria degli insiemi

- Una tabella rappresenta una relazione se:
  - 1) le righe sono diverse fra loro
  - 2) le intestazioni delle colonne sono diverse fra loro
  - 3) i valori di ogni colonna sono fra loro omogenei

## RELAZIONI E TABELLE

Dati  $m > 0$  insiemi  $D_1, D_2, \dots, D_m$ , non necessariamente distinti, il prodotto cartesiano indicato con  $D_1 \times D_2 \times \dots \times D_m$  è costituito dall'insieme delle  $m$ -uple  $(v_1, v_2, \dots, v_m)$  t.c.  $v_i \in D_i$  per  $1 \leq i \leq m$ . Una relazione matematica sui domini  $D_1, \dots, D_m$  è un sottoinsieme del prodotto cartesiano  $D_1 \times D_2 \times \dots \times D_m$ .

Queste relazioni possono essere utilizzate per rappresentare i dati di interesse per qualche applicazione.

Esempio: relazione che contiene i dati relativi ai risultati di un insieme di partite di calcio

$\text{Matches} \subseteq \text{String} \times \text{String} \times \text{int} \times \text{int}$

Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

←  
ciascuna  $m$ -upla contiene dati fra

loro collegati e stabilisce un legame fra loro. Se  $m$ -uple sono ORDINATE, è definito un ordinamento fra i domini

↳ L'ORDINAMENTO è una caratteristica insoddisfacente del concetto di relazione matematica, solitamente in informatica si tende a privilegiare metizioni NON-POSIZIONALI

## STRUTTURA DATI NON POSIZIONALE

Ogni nome, unico, nella tabella (ATTRIBUTO) è associato ad un dominio. L'attributo fornisce così il "ruolo" del dominio.

Name	Age	Goals	Goals
Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

## TABELLE E RELAZIONI

Una tabella rappresenta una relazione: • ogni riga può assumere qualunque posizione

• ogni colonna può assumere qualunque posizione

- Una tabella rappresenta una relazione se:
- tutte le righe sono diverse
  - tutte le colonne sono diverse
  - i valori nelle colonne sono omogenei

I riferimenti tra dati memorizzati in diverse relazioni sono rappresentate attraverso valori dei domini che compaiono nelle tuple.

Vantaggi:

- Indipendenza dalle strutture fatiche che possono cambiare dinamicamente

- Si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione

- L'utente finale vede gli stessi dati dei programmati

- Portabilità dei dati

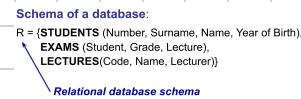
- Puntatori unidirezionali

## DEFINIZIONI

- **SCHEMA DI RELAZIONE**: una relazione detta R con un insieme di attributi  $A_1, \dots, A_m : R(A_1, \dots, A_m)$



- **SCHEMA DI BASE DI DATI**: insieme di schemi di relazione  $R = \{R_1(X_1), \dots, R_k(X_k)\}$



- **N-UPLA** su un insieme di attributi X: funzione che associa a ciascun attributo  $A \in X$  un valore del dominio di A.

- **$t[A]$** : dimostra le varie delle m-uple t sull'attributo A. es.  $t[Home] = Barca$

- **ISTANZA DI RELAZIONE SU UNO SCHEMA  $R(x)$** : insieme r di m-uple su X

- **ISTANZA DI BASE DI DATI** su uno schema  $R = \{R_1(X_1), \dots, R_k(X_k)\}$ : insieme di relazioni  $r = \{r_1, \dots, r_m\}$  (con  $r_i$  relazione su  $R_i$ )

## LEZIONE 02 - Relational Model

### 2.1.15 Informazione Incompleta e Valori Nulli

Modello relazionale impone un certo grado di rigidità.

informazioni rappresentate con

**TUPLE di dati sfaccendati**.

es. Persona (Cognome, Nome, Indirizzo, Telefono)

Non sempre tutti gli attributi sono disponibili in ogni tupla (es. manca il campo "Telefono"), non è utile utilizzare un valore del dominio per rappresentare l'assenza dell'informazione in quanto potrebbero causare delle ambiguità.

→ utilizziamo il **VALORE NULLO**, valore aggiuntivo rispetto a quelli del dominio.

Studenti

Matricola	Cognome	Nome	Data di nascita
276545	Rossi	Maria	null
null	Neri	Anna	23/04/2001
null	Verdi	Fabio	12/02/2001

Esami

Studente	Voto	Corso
276545	28	01
null	27	null
200768	24	null

Corsi

Codice	Titolo	Docente
01	Analisi	Giani
03	Chimica	null
null	Chimica	Belli

Il valore nullo non è sempre utilizzabile, infatti se il mappo sul numero di matricola o sul codice del corso può generare problemi, in quanto sono utilizzati per stabilire correlazioni fra tuple di relazioni diverse.

Tre casi per la mancanza di un valore: 1) UNKNOWN VALUE

2) INEXISTANT VALUE

3) UNINFORMATIVE VALUE

nei moderni DBMS non ci sono specifiche distinzioni

## 2.2 VINCOLI DI INTEGRITÀ

Per evitare situazioni in cui i dati rappresentati non sono corretti viene introdotto il **VINCOLO DI INTEGRITÀ**

proprietà che deve essere soddisfatta dalle istanze  
ogni vincolo può essere visto come una funzione booleana (o predicato)

Possiamo classificare i vincoli in due categorie:

- **VINCOLO INTRARELACIONALE**: se il suo soddisfacimento è definito rispetto a singole relazioni della base di dati.
  - **VINCOLO DI TUPLA**: vincolo che può essere valutato su ciascuna tupla, indipendentemente dalle altre.
  - **VINCOLO SU VALORI**: (o vincolo di dominio) impone una restrizione sul dominio dell'attributo definito con riferimento a singoli valori (es. voto >= 30 in "esami")
- **VINCOLO INTERRELACIONALE**: vinkolo che coinvolge più relazioni (es. viene richiesto che m° di matricola compare nella relazione "esami" solo se compare nella relazione "studenti")

I DBMS non supportano tutti i tipi di vincoli:

- possiamo indicare i tipi di vincoli e il DBMS impedisce la loro violazione
- quando non sono supportati sono l'utente e il programmatore a configurare i vincoli al di fuori del DBMS.

		Intra-relational over values					
		EXAM	Student	Grade	Lauda	Lecture	
Inter-relational		276545	276545	32	yes	01	
		276545	787643	30	yes	02	
		787643	739430	27	yes	03	
		739430		24		04	
		Intra-relational over tuples					
		STUDENT	Number	Surname	Name		
		276545	Rossi	Mario			
		787643	Neri	Piero			
		787643	Bianchi	Luca			

### 2.2.1 VINCOLO DI TUPLA

La sintassi è esprimibile utilizzando espressioni booleane ( $\wedge, \vee, \neg$ ) con atomi che confrontano valori di attributo o espressioni aritmetiche (uguaglianza, disegualanza e ordinamento)

es.  $(voto >= 18) \wedge (voto <= 30) \quad (\text{not}(Lode = "lode")) \text{ or } (voto = 30)$

### 2.2.2 CHIAVE

→ insieme di attributi per identificare univocamente le tuple di una relazione.

Distinguiamo:

- **SUPERCHIAVE**: un insieme K di attributi è superchiave su r se r non contiene due tuple distinte t<sub>1</sub> e t<sub>2</sub>

con  $t_1[K] = t_2[K]$

- **CHIAVE**: K è chiave di r se è una superchiave minimale di r, ovvero non esiste un'altra superchiave K' di r che sia contenuta in K come sottinsieme proprio.

Matricola	Cognome	Nome	Nascita	Corso
4328	Rossi	Luigi	29/04/2000	Ing. Informatica
6328	Rossi	Dario	29/04/2000	Ing. Informatica
4766	Rossi	Luca	01/05/2001	Ing. Civile
4856	Neri	Luca	01/05/2001	Ing. Meccanica
5536	Neri	Luca	05/03/1999	Ing. Meccanica

es.  
l'insieme {Matricola} è superchiave, è anche superchiave minimale in quanto contiene un solo attributo, quindi {Matricola} è chiave.  
{Cognome, Nome, Nascita} è superchiave, ma non è superchiave minimale, perché nessuno dei suoi sottointesi è superchiave. (possono esistere tuple con cognome e nascita uguali, ecc...)  
{Matricola, Corso} è superchiave, ma non è una superchiave minimale, perché il sottointeso {Matricola} è una superchiave minimale  
{Nome, Corso} non è superchiave perché possono comparire tuple fra loro uguali sia su nome che su corso.

de chiavi che ci "interessano" sono quelle corrispondenti ai vincoli di integrità  $\Rightarrow$  definendo uno SCHEMA, associamo ad es. SCHEMA: Studente (Matricola, Nome, Cognome, Nascita, Corso)

VINCOLO: 2 chiavi {Matricola}

{Cognome, Nome, Nascita}

N.B. Ciascuna relazione ha una chiave

## 2.2.3 CHIAVI E VALORI NULLI

In presenza di valori nulli non è sempre possibile identificare univocamente una tupla.

$\hookrightarrow$  CHIAVE PRIMARIA: chiave in cui vietiamo la presenza di valori nulli.

$\hookrightarrow$  gli attributi che la compagno vengono evidenziati con una SOTTOUNIVERSITÀ.

$\hookrightarrow$  La maggior parte dei riferimenti tra relazioni viene realizzata con i suoi valori.

CHIAVE  $\xrightarrow{\text{sempre}}$  SUPERCHIAVE (ma il contrario)

$\hookrightarrow$  è chiave solo se è SUPERCHIAVE MINIMALE.

## 2.2.4 VINCOLI DI INTEGRITÀ REFERENZIALE

FOREIGN KEY (REFERENTIAL INTEGRITY CONSTRAINT) fra un insieme di attributi X di una relazione R<sub>1</sub> e un'altra relazione R<sub>2</sub> è soddisfatto se i valori su X di ciascuna tupla dell'istanza R<sub>1</sub> compaiono come valori della CHIAVE PRIMARIA dell'istanza R<sub>2</sub>.

Nel caso in cui la chiave di R<sub>2</sub> è unica e composta di un solo attributo B e l'insieme X è a sua volta costituito di un solo attributo A, il vincolo tra R<sub>1</sub> e R<sub>2</sub> è soddisfatto se:  $\forall$  tupla t<sub>1</sub>  $\in$  R<sub>1</sub> per cui t<sub>1</sub>[A] ≠ Null,  $\exists$  t<sub>2</sub>  $\in$  R<sub>2</sub> t.c. t<sub>1</sub>[A] = t<sub>2</sub>[B].

+ ciascuno degli attributi in X deve corrispondere a un preciso attributo della PK K di R<sub>2</sub>,

X = A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>p</sub> e K = B<sub>1</sub>, B<sub>2</sub>, ..., B<sub>p</sub> il vincolo di integrità è soddisfatto se

$\forall$  tupla t<sub>1</sub>  $\in$  R<sub>1</sub> dove t<sub>1</sub>[A<sub>i</sub>] ≠ Null  $\exists$  t<sub>2</sub>  $\in$  R<sub>2</sub> t.c. t<sub>1</sub>[A<sub>i</sub>] = t<sub>2</sub>[B<sub>i</sub>]  $\forall$  i  $1 \leq i \leq p$

Im caso siamo presenti più chiavi  $\rightarrow$  una deve essere sottointesa come PK  $\rightarrow$  i riferimenti vengono fatti verso la PK

Se una tupla viene eliminata  $\rightarrow$  VIOLAZIONE vincolo: - rigetto dell'op.

- eliminazione in cascata

- introduzione valori nulli

## LEZIONE 03. ALGEBRA AND RELATIONAL CALCULUS

• DATABASE LANGUAGE: - op. sugli schemi: DDL, Data Definition Language

- op. sui dati: DML, Data Manipulation Language - query instructions, per estrarre i dati di interesse

update instructions, per inserire nuovi dati o modificare quelli già presenti

- LINGUAGGI DI INTERROGAZIONE:
  - DICHIAIRATIVI: specificano le proprietà del risultato "che cosa"
  - PROCEDURALI: specificano le modalità di generazione del risultato "come"

es. algebra relazionale  $\rightarrow$  procedurale

calcolo relazionale  $\rightarrow$  dichiarativo (teoricamente, non implementato)

SQL (structured query language)  $\rightarrow$  parzialmente dichiarativo (implementato)

QBE (query by example)  $\rightarrow$  dichiarativo (implementato)

### 3.1 ALGEBRA RELAZIONALE

distinguendo PROCEDURALE basato su comandi di tipo algebrico. Formato da OPERATORI:

#### 1) INSIEMISTICI TRADIZIONALI: UNIONE, INTERSEZIONE, DIFFERENZA

RELAZIONE  $\rightarrow$  insieme di tuple omogenee definite sugli stessi attributi.

- l'**UNIONE** di due relazioni  $r_1$  e  $r_2$  definite sullo stesso insieme di attributi  $X$  è indicata come  $r_1 \cup r_2$  ed è una relazione ancora su  $X$  contenente le tuple che appartengono a  $r_1$ , oppure a  $r_2$ , oppure a entrambe (no duplicati)
- l'**INTERSEZIONE** di  $r_1(X)$  e  $r_2(X)$  è indicata come con  $r_1 \cap r_2$  ed è una relazione su  $X$  contenente le tuple che appartengono contemporaneamente a  $r_1$  e  $r_2$ .
- la **DIFERENZA** di  $r_1(X)$  e  $r_2(X)$  è indicata con  $r_1 - r_2$  ed è una relazione su  $X$  contenente le tuple che appartengono a  $r_1$  e non appartengono a  $r_2$ .

#### 2) OPERATORI SPECIFICI: RIDENOMINAZIONE, SELEZIONE, PROIEZIONE

- RIDENOMINAZIONE: aggiorna i nomi degli attributi, a seconda delle necessità, lasciando inalterato il contenuto delle relazioni.

**def**  $P$  relazione definita sull'insieme di attributi e sia  $Y$  un altro insieme di attributi con  $\#X = \#Y$ . Siamo

$A_1, \dots, A_k$  e  $B_1, \dots, B_k$  rispettivamente un ordinamento per gli attributi in  $X$  e in  $Y$ . Allora:

$$P_{B_1, \dots, B_k \leftarrow A_1, \dots, A_k} (r)$$

es.  $P$    
 genitore  $\leftarrow$  padre (Paterno)

- SELEZIONE: produce un sottoinsieme delle tuple su tutti gli attributi. ( $\rightarrow$  genera decomposizioni orizzontali). È indicata

con  $\sigma$  e si pedisce inseriamo la combinazione di selezione.

formula proposizionale  $F$  su  $X$  ottenuta combinando:

- connettivi AND, OR, NOT ( $\wedge, \vee, \neg$ )

- combinazioni atomiche:  $A \oplus B$   $A \otimes C$



La selezione  $\sigma_F(r)$ , dove  $r$  relazione e  $F$  formula proposizionale, produce una relazione sugli stessi attributi di  $r$

che contiene le tuple di  $r$  su cui  $F$  è vera.

- PROIEZIONE: sul risultato troviamo tutte le tuple, ma su un sottoinsieme degli attributi ( $\rightarrow$  genera decomposizioni verticali)

**def** Dati una relazione  $r(x)$  e un sottoinsieme  $Y$  di  $X$ , la proiezione di  $r$  su  $Y$ , indicata con  $\pi_Y(r)$ , è l'insieme di

tuple di  $r$  considerando solo i valori su  $Y$ :

Impiegati				
Cognome	Nome	Età	Stipendio	
Rossi	Mario	25	2000,00	
Neri	Luca	40	3000,00	
Verdi	Nico	36	4500,00	
Rossi	Marco	40	3900,00	

$\sigma_{\text{Eta}>30 \wedge \text{Stipendio}>4000,00}(Impiegati)$

Cognome	Nome	Età	Stipendio
Verdi	Nico	36	4500,00



$R_1 \bowtie_{\text{condizione}} R_2$  (prodotto cartesiano seguito da una selezione)

**EQUI-JOIN:** e' un THETA-JOIN in cui la condizione di selezione e' una congiungione di atomi di uguaglianza, permette di specificare su quale attributo unire senza utilizzare la ridenominazione.

N.B.: se usiamo il theta-join su due relazioni con attributi in comune, non me tener conto, prenderà solo quelli specificati nella condizione.

	UNIONE ( $R_1 \cup R_2$ )
INSIEMISTICI	INTERSEZIONE ( $R_1 \cap R_2$ )
	DIFERENZA ( $R_1 - R_2$ )
SPECIFICI	SELEZIONE ( $\Theta_F(R)$ ) sulle tuple
	PROIEZIONE ( $\pi_F(R)$ ) sugli attributi
	RIDENOMINAZIONE ( $P_{X_1 \leftarrow X_2}(R)$ )
JOIN	NATURALE ( $R_1 \bowtie R_2$ )
	THETA-JOIN ( $R_1 \bowtie_F R_2$ )

### ■ ESPRESSIONI DI ALGEBRA RELAZIONALE EQUIVALENTE

= se due espressioni  $E_1$  e  $E_2$  generano lo stesso risultato

$$1. \Theta_{C1 \text{ AND } C2}(R) = \Theta_{C1}(\Theta_{C2}(R))$$

$$2. x, y \in R : \pi_x(R) = \pi_x(\pi_{x \rightarrow y}(R))$$

$$3. \pi_{x \rightarrow y}(\Theta_x(R)) = \Theta_x(\pi_{x \rightarrow y}(R))$$

$$4. \text{CER}_2 : \Theta_C(R_1 \bowtie R_2) = R_1 \bowtie (\Theta_C(R_2)) \quad (\text{riduzione dimensione intermedia del risultato})$$

$$5. x_1 \in R_1, x_2 \in R_2, y_2 \in R_2 \text{ e } (x_1 \cap x_2) \subseteq y_2 \Rightarrow x_2 - y_2 \text{ non e' coinvolto nel JOIN} : \pi_{x_1, y_2}(R_1 \bowtie R_2) = R_1 \bowtie \pi_{y_2}(R_2)$$

$$6. \Theta_C(R_1 \bowtie R_2) = R_1 \bowtie R_2$$

$$7. \Theta_C(R_1 \cup R_2) = \Theta_C(R_1) \cup \Theta_C(R_2)$$

$$8. \Theta_C(R_1 - R_2) = \Theta_C(R_1) - \Theta_C(R_2)$$

$$9. \pi_x(R_1 \cup R_2) = \pi_x(R_1) \cup \pi_x(R_2) \quad \text{N.B. da proiezione non e' distributiva sulla differenza}$$

$$10. \Theta_{C1 \text{ OR } C2}(R) = \Theta_{C1}(R) \cup \Theta_{C2}(R)$$

$$11. \Theta_{C1 \text{ AND } C2}(R) = \Theta_{C1}(R) \cap \Theta_{C2}(R) = \Theta_{C1}(R) \bowtie \Theta_{C2}(R)$$

$$12. \Theta_{C1 \text{ AND } T_{C2}}(R) = \Theta_{C1}(R) - \Theta_{C2}(R)$$

$$13. R_1 \bowtie (R_2 \cup R_3) = (R_1 \bowtie R_2) \cup (R_1 \bowtie R_3)$$

### 3.1.9 ALGEBRA CON VALORI NULLI

Possiamo "rimediarci" alla presenza di valori nulli in 2 modi:

1. LOGICA A TRE VALORI: VERO

FALSO  
UNKNOWN

} non e' necessario

A is null

2. traduciamo i null simbolicamente introducendo 2 condizioni atomiche

A is not null

### 3.1.10 VISTE O RELAZIONI DERIVATE

↳ relazioni definite per mezzo di funzioni (create da un'interrogazione)

↳ rappresentazioni diverse per gli stessi dati

Due tipi: • **VISTE MATERIALIZZATE**: relazioni derivate effettivamente memorizzate nel DB.

PRO: subito disponibili per le interrogazioni

CONTRO: - ridondanza dei dati      - eccessivamente supportate dal DBMS

- appesantiscono gli aggiornamenti

• **VISTE VIRTUALI**: relazioni definite per mezzo di interrogazioni, non memorizzate nel DB.

↳ le interrogazioni sulle viste vengono eseguite sostituendo alla vista la sua definizione (avendo, componendo le due interrogazioni)

PRO: supportate da tutti i DBMS.

- Ogni utente vede quindi: ciò che gli interessa

ciò che è autorizzato a visualizzare

Strumento di programmazione: possiamo semplificare la sintassi delle interrogazioni

↳ l'utilizzo di viste non influenza sull'efficienza delle interrogazioni.

AGGIORNARE LE VISTE := cambiare la tauta base in modo che la vista aggiornata riferisca l'aggiornamento.

### 3.2 CALCOLO RELAZIONALE

↳ famiglia di linguaggi di interrogazione, basati sui calcoli dei predicati del primo ordine, che hanno la caratteristica di essere DICHIASTRATIVI

↳ specificano la proprietà del risultato delle interrogazioni, anziché la procedura seguita

Due definizioni: • **CALCOLO RELAZIONALE SU DOMINI**

• **CALCOLO SU TUPLE CON DICHIARAZIONI DI RANGE**

#### 3.2.1 CALCOLO RELAZIONALE SU DOMINI

Le espressioni hanno forma:  $\{A_1 : x_1, \dots, A_k : x_k \mid f\}$ , dove:

•  $A_1, \dots, A_k$  sono attributi distinti (possono non compiere nello schema del DB)

•  $x_1, \dots, x_k$  sono variabili (supponiamo distinte)

•  $f$  è una formula che utilizza operatori booleani e quantificatori.

La lista di copie  $A_1 : x_1, \dots, A_k : x_k$  viene chiamata TARGET LIST, definisce la struttura del risultato.

ESEMPIO:

Impiegati (Matricola, Nome, Età, Stipendio)

Supervisione (Capo, Impiegato)

A.R.: Ø Stipendio > 40 (Impiegati)

C.R.D.: {Matricola: m, Nome: n, Età: e, Stipendio: s |

Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s)

$\wedge (s > 40)$ ] predicato x selezione le tuple

costituito dalla relazione su  $A_1, \dots, A_k$  che

contiene le tuple i cui valori, sostituiti a

$x_1, \dots, x_k$ , rendono vera la formula rispetto a

un'istanza di DB a cui l'espressione viene applicata

- Return the chiefs' number and name having all employees earning more than 40

$$\begin{aligned} & \Pi_{Number, Name} (\text{EMPLOYEE} \bowtie_{\substack{\text{Number} = \text{Chief} \\ (\text{Chief} \in \text{SUPERVISOR})}} \text{Employee} = \text{Number} \sigma_{Wage > 40} (\text{EMPLOYEE}))) \\ & \{ \text{Number: } c, \text{Name: } n \mid \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \neg \exists m' (\exists n' (\exists a' (\exists w' (\text{EMPLOYEE}(\text{Number: } m', \text{Name: } n', \text{Age: } a', \text{Wage: } w') \wedge \\ & w' \leq 40 \wedge \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m'))))) \end{aligned}$$

- Return the employees earning more money than their boss; for both such employees and chiefs return the number, name and salary

$$\begin{aligned} & \Pi_{Number, Name, Wage, NumC, NameC, WageC} (\sigma_{Wage > WageC} (\rho_{NumC, NameC, WageC, AgeC \leftarrow \text{Number}, \text{Name}, \text{Wage}, \text{Age}} (\text{EMPLOYEE}))) \\ & \bowtie_{\substack{\text{NumC} = \text{Chief} \\ (\text{SUPERVISOR} \bowtie_{\text{Employee} = \text{Number}} \text{EMPLOYEE}))}} \\ & \{ \text{Number: } m, \text{Name: } n, \text{Wage: } w, \text{NumC: } c, \text{NameC: } nc, \text{WageC: } wc \mid \\ & \text{EMPLOYEE}(\text{Number: } m, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } nc, \text{Age: } ac, \text{Wage: } wc) \wedge w > sc \} \end{aligned}$$

**RECALL - Leggi di DeMorgan**

- $\neg(f \wedge g) = \neg f \vee \neg g$
- $\neg(f \vee g) = \neg f \wedge \neg g$

**quantificatori:**

- $\neg \forall x A = \exists x \neg A$
- $\forall x A = \neg \exists x \neg A$
- $\neg \exists x A = \forall x \neg A$
- $\exists x A = \neg \forall x \neg A$

**+**:  $\neg A \vee B \rightarrow \text{if } A \text{ then } B$

### 3.2.2 PROS AND CONS del calcolo su domini

**PRO:** Dichiarativo

**CONS:** • "verboso", troppe variabili

• ammette espressioni che non fanno senso. es:  $\{A: x \mid \neg R(A: x)\}, \{A: x, B: y \mid R(A: x)\}$

**PLUS:** DRC e RA sono equivalenti:

per ogni espressione dipendente dal dominio

im DRC, esiste un'espressione equivalente im RA,

e viceversa.

queste espressioni sono dipendenti dal dominio e le

dovremo evitare + nel A.R. Non le possiamo usare perché

sono dipendenti dal dominio.

### 3.2.3 CALCOLO SU TUPLE CON DICHIARAZIONI DI RANGE

→ ha forma:  $\{T \mid L \mid f\}$ , dove:

- T è la TARGET LIST, con elementi del tipo  $Y: x_1 \dots x_n$ , con x variabile e Y e  $x_1 \dots x_n$  sequenze di attributi, gli attributi in Y devono comporre tutto lo schema della relazione che costituisce il RANGE di X.
- L è la RANGE LIST, elenca le variabili libere della formula f con i relativi range. L è una lista di elementi del tipo  $x(R)$ , con x variabile e R nome di relazione.
- f è una formula con ATOMI, CONNETTIVI, QUANTIFICATORI.

⇒ viene ridotto il numero di variabili

• tutti i valori devono venire dati da,

- Return the chiefs' number and name having all employees earning more than 40

$$\begin{aligned} & \{ \text{Number: } c, \text{Name: } n \mid \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \neg \exists m' (\exists n' (\exists a' (\exists w' (\text{EMPLOYEE}(\text{Number: } m', \text{Name: } n', \text{Age: } a', \text{Wage: } w') \wedge \\ & w' \leq 40))) \} \\ & \{ e.(\text{Number}, \text{Name}) \mid s(\text{SUPERVISOR}), e(\text{EMPLOYEE}) \mid \\ & s.\text{Chief} = e.\text{Number} \wedge \neg \exists e'(\text{EMPLOYEE})(\exists s'(\text{SUPERVISOR}) \\ & (s.\text{Chief} = s'.\text{Chief} \wedge s'.\text{Employee} = e'.\text{Number} \wedge e'.\text{Wage} \leq 40)) \} \end{aligned}$$

ES. Matricola, Nome, Età e Stipendio degli impiegati che guadagnano più di 40 mila euro:

$$\{ i * \mid i \mid (\text{Impiegati}) \mid i.\text{Stipendio} > 40 \}$$

restituisce tutti gli attributi

## LIMITI:

Il calcolo su tuple con dichiarazioni di range non permette di esprimere le interrogazioni i cui risultati possono provenire indifferentemente da due o più relazioni (come l'operatore di unione).

→ Infatti, essendo SQL basato sul calcolo su tuple con dichiarazioni di range, prevede un costrutto esplicito di UNIONE, mentre l'INTERSEZIONE e la DIFFERENZA li possiamo esprimere in altri modi.

## CALCOLO e ALGEBRA

- Sono praticamente EQUIVALENTI.
- Alcune query utili non possono essere espresse:
  - possiamo solo estrarre valori, non possiamo calcolare nuovi.
  - ES: - su ogni tupla: (conversioni, somme, differenze, ...)
  - su un set di tuple: (sommatoria, media, ...)
  - estensioni adottate però da SQL
  - query ricorsive, come la chiusura transitiva (TRANSITIVE CLOSURE)

## CHIUSURA TRANSITIVA

La chiusura transitiva  $R^+$  di una relazione binaria  $R$  su un set  $X$  è la più piccola relazione su  $X$  che contiene  $R$  ed è transitiva. Data una relazione  $R$  su  $A \times A$ , la chiusura transitiva è una relazione  $R^+$  tale che

$$R^+ = \{ \langle x, y \rangle \mid \exists y_1, \dots, y_m \in A, m \geq 2, y_1 = x, y_m = y, \langle x, y_i \rangle, \langle y_i, y_{i+1} \rangle \in R, i=1, \dots, m-1 \}$$

Se  $X$  è un set di aeroporti  $x \in X$  significa "c'è un volo diretto da aeroporto  $x$  a aeroporto  $y$ ", la chiusura transitiva  $x \in R^+ y$  significa "è possibile volare da  $x$  a  $y$  in uno o più voli"

- We could use both joins with renaming in order to express such relations

- But:

Employee	Chief	Employee	Superior
Rossi	Lupi	Rossi	Lupi
Neri	Bruni	Neri	Bruni
Lupi	Falchi	Lupi	Falchi
Falchi	Leoni	Falchi	Leoni

Nell'algebra relazionale non possiamo esprimere la chiusura transitiva per ogni relazione binaria, ma si darebbe ricreare ogni volta un'espressione diversa.

128

## 3.3 DATALOG

L'idea su cui si basa DATALOG è quella di adattare alle basi di dati il linguaggio di programmazione logica PROLOG.

Si basa sul calcolo dei predicati del primo ordine, possiamo trovare due tipi di predicati:

- i predicati **ESTENSIONALI**, corrispondono alle relazioni nella base di dati.
- i predicati **INTENSIONALI**, sono specificati per mezzo di regole logiche. Questi predicati degli insieme viste (relazioni virtuali) sulla base di dati.

Le REGOLE DATALOG hanno la forma: **testa ← corpo**

• la TESTA è un predicato atomico

Sono imposte delle CONDIZIONI:

• il CORPO è una lista di condizioni atomiche.

- i predicati estensioniali possono comparire solo nel corpo delle regole
- se una variabile compare nella testa di una regola, allora deve comparire anche nel corpo della stessa regola.
- se una variabile compare in un atomo di confronto, allora deve comparire anche in un atomo nel corpo della stessa regola.

- Return the employees' number, name, age and salary being 30 years old

{ Number:  $m$ , Name:  $n$ , Age:  $a$ , Wage:  $w$  }  
EMPLOYEE(Number:  $m$ , Name:  $n$ , Age:  $a$ , Wage:  $w$ )  $\wedge a = 30$

DRC  
DATALOG  
132

- Return the chiefs' number and name having all employees earning more than 40

- We need negation

CHIEFSOFNORICHERS(Chief:  $c$ ) ←  
EMPLOYEE(Number:  $m$ , Name:  $n$ , Age:  $a$ , Wage:  $w$ ),  
 $w \leq 40$ , SUPERVISOR(Chef:  $c$ , Employee:  $m$ )  
CHIEFSOONLYRICHER(Number:  $c$ , Name:  $n$ ) ←  
EMPLOYEE(Number:  $c$ , Name:  $n$ , Age:  $a$ , Wage:  $w$ ),  
SUPERVISOR(Chef:  $c$ , Employee:  $m$ ),  
NOT CHIEFSOFNORICHERS(Chef:  $c$ )  
? CHIEFSOONLYRICHER(Number:  $c$ , Name:  $n$ )<sup>136</sup>

La definizione di query ricorsive diventa complicata nel caso della negazione:

- datalog mon ricorsivo senza negazione equivale al calcolo senza negazione e senza quantificatore universale.
- datalog mon ricorsivo con negazione equivale al calcolo e all'algebra
- non possiamo comparare datalog ricorsivo senza negazione e calcolo.
- datalog ricorsivo con negazione è più espressivo del calcolo e dell'algebra.

che cosa significa?

## SQL

### DATA DEFINITION

- CREATE DATABASE db\_name : ogni database creato contiene tabelle, viste, trigger, ...
- CREATE SCHEMA db\_schema: uno schema in SQL è identificato da un nome e descrive gli elementi che gli appartengono, come tabelle, tipi, vincoli, viste, domini... può essere seguito dalla keyword AUTHORIZATION per indicare il nome dell'utente proprietario dello schema.  

```
CREATE SCHEMA schema_name
AUTHORIZATION 'user_name'
```
- CREATE TABLE table\_name : ogni tabella viene definita associandole un nome ed elencando gli attributi che ne compongono lo schema. Per ogni attributo si definiscono un nome, un dominio ed eventualmente un insieme di vincoli che devono essere rispettati dai valori dell'attributo.

### BASIC DATA TYPES

- Domino character permette di rappresentare singoli caratteri o stringhe. La lunghezza può essere fissa o variabile.
- Tipi numerici, famiglia che contiene i domini che permettono di rappresentare valori esatti, interi o con una parte decimale.
- Istanti temporali: - date (year, month, day)
  - time (hour, minute, second)
  - interval, per rappresentare intervalli di tempo
- SQL-3: - boolean, per rappresentare singoli valori booleani
- BLOB e CLOB, permettono di rappresentare oggetti di grandi dimensioni, costituiti da una sequenza arbitraria di valori binari (BLOB, binary large object) o di caratteri (CLOB, character large object). È possibile memorizzare questi valori ma non possono essere utilizzati come criterio di selezione nelle query.

### CUSTOM DATA TYPES

Ogni tipo di dati standard può essere utilizzato per definire nuove relazioni, indicando vincoli e valori di default

Es. CREATE DOMAIN Grade  
AS SMALLINT DEFAULT NULL  
CHECK (value >= 18 AND value <= 30)

## VINCOLI INTRARELAZIONALI

Proprietà che devono essere verificate da ogni istanza della base di dati.

- NOT NULL**: indica che il valore null non è ammesso come valore dell'attributo. Nel caso in cui all'attributo è associato un valore di default (diverso da null), allora è possibile fare l'inserimento anche senza fornire un valore per l'attributo.

Cognome varchar(20) not null

- UNIQUE**: si applica ad un attributo, o ad un insieme di attributi, e impone che i valori dell'attributo siano una (super)chiave.

Esezione per il valore nullo → si assume che i valori null siano tutti diversi tra loro.

Matricola character(6) unique

Nome varchar(20) not null,  
Cognome varchar(20) not null,  
unique (Cognome, Nome)

- PRIMARY KEY**, è possibile specificare la chiave primaria una sola volta per tabella, può essere definita su un singolo attributo o su un insieme di attributi. Gli attributi che ne fanno parte non possono assumere valore null.

Nome varchar(20),  
Cognome varchar(20),  
primary key (Cognome, Nome)

Es.

```
CREATE TABLE EMPLOYEE (
    Number CHARACTER(6) PRIMARY KEY,
    Name CHARACTER(20) NOT NULL,
    Surname CHARACTER(20) NOT NULL,
    Dept CHARACTER(15),
    Wage NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dept) REFERENCES
        DEPARTMENT(Dept),
    UNIQUE (Surname, Name)
)
```

**WARNING:** Name CHARACTER(20) NOT NULL,  
Surname CHARACTER(20) NOT NULL,  
UNIQUE (Surname, Name)  
!=  
Name CHARACTER(20) NOT NULL UNIQUE,  
Surname CHARACTER(20) NOT NULL UNIQUE,

## VINCOLI INTERRELAZIONALI

Questo vincolo crea un legame tra i valori di un attributo della tabella su cui è definito (interna) e i valori di un attributo di un'altra tabella (esterna). L'unico requisito è che l'attributo cui si fa riferimento sia oggetto al vincolo **unique**.

Può essere definito in due modi:

1. Su un solo attributo: utilizziamo **REFERENCES**, dove si specifica la tabella esterna e l'attributo della tabella esterna al quale l'attributo in questione deve essere legato.
2. Su un insieme di attributi: utilizziamo **FOREIGN KEY**, dove vengono elencati gli attributi della tabella coinvolti, cui segue la definizione dei corrispondenti attributi della tabella esterna con **REFERENCES**.

```
CREATE TABLE OFFENCES (
    Code CHARACTER(6) PRIMARY KEY,
    Day DATE NOT NULL,
    Officer INTEGER NOT NULL REFERENCES OFFICER(Id),
    State CHARACTER(2),
    Number CHARACTER(6),
    FOREIGN KEY(State, Number) REFERENCES CAR(State, Number)
)
```

Dopo ogni vincolo referenziale, possiamo specificare l'azione da invocare se l'operazione è respinta:

- per le op. di modifica:

- **CASCADE**, il nuovo valore della t. esterna viene riportato sulle righe corrispondenti della t. interna.

- **SET NULL**, all'attributo riferente viene assegnato il valore nullo al posto di quello modificato nella t. esterna.

- **SET DEFAULT**, all'attributo riferente viene assegnato il valore di default al posto di quello modificato nella t. esterna.

- NO ACTION, l'azione di modifica non viene consentita

- violazioni prodotte dall'eliminazione:

- CASCADE: tutte le righe della t. intorno corrispondenti alla riga cancellata vengono cancellate

- SET NULL: dell'attributo riferente viene assegnato il valore nullo al posto del valore cancellato nella tabella esterna

- SET DEFAULT: dell'attributo riferente viene assegnato il valore di default al posto del valore cancellato nella tabella esterna

- NO ACTION: la cancellazione non viene consentita

### **MODIFICA DEGLI SCHEMI**

- **ALTER** : permette di modificare i domini e schemi di tabella.

- **ALTER DOMAIN** : permette di modificare domini già definiti. Viene utilizzato con:

- SET DEFAULT
- DROP DEFAULT
- ADD CONSTRAINT
- DROP CONSTRAINT

- **ALTER TABLE** : permette di modificare tabella già definite. Viene utilizzato con:

- ALTER COLUMN
- ADD COLUMN
- DROP COLUMN
- DROP CONSTRAINT
- ADD CONSTRAINT

- **DROP DOMAIN** : rimuove un tipo di dati definito dall'utente

- **DROP TABLE** : rimuove un'intera istanza di tabella con il suo schema e i suoi dati

### **DEFINIZIONE INDICI**

- **CREATE INDEX** utilizzato per creare un indice su una o più colonne di una tabella. Migliora la velocità delle operazioni di lettura,

- ma rallenta le op. di scrittura. (simile all'indice di un libro)

- CREATE INDEX idx\_surname  
ON OFFICER (Surname)

### **OPERAZIONI SUI DATI**

- Query: **SELECT**

- Edit: **INSERT, DELETE, UPDATE**

- **SELECT <AttributeList> //target post**  
**FROM <TableList> //statement**  
[ **WHERE <Condition>** ] //statement

- Data una relazione R(A,B)  
SELECT \*  
FROM R  
-> (\*) seleziona tutti gli attributi di R

- **SELECT \***  
FROM PEOPLE  
WHERE Name **LIKE 'J\_m'**

Ritorna le persone che hanno il nome che inizia con la 'J' e che hanno come terza lettera 'm'  
-> '\_' sostituisce un singolo carattere  
-> '%' sostituisce zero o più caratteri

- **SELECT DISTINCT <AttributeList>**  
FROM <Table>  
-> DISTINCT viene utilizzato per restituire solo valori unici in una query eliminando i duplicati

- **R1(A1, A2), R2(A3, A4)**  
SELECT DISTINCT R1.A1, R2.A4  
FROM R1, R2  
WHERE R1.A2 = R2.A3  
-> (FROM) prodotto cartesiano  
-> (WHERE) selezione  
-> (SELECT) proiezione

- **Alias & Ridenominazione**  
SELECT X.A1 AS B1...  
FROM R1 AS X, R2 AS Y,...  
WHERE X.A2 = Y.A3 AND...

## VALUTAZIONE DELLE QUERIES

SQL è un linguaggio dichiarativo, significa che indichiamo cosa vuoi ottenere, senza specificare come il DB debba eseguire il lavoro. I DBMS fanno piani di esecuzione per eseguire le query in modo efficiente:

- le selezioni vengono eseguite il prima possibile per ridurre il numero di righe da processare
- dove possibile, le join vengono eseguite invece dei prodotti cartesiani, in quanto le join sono più efficienti perché combinano solo le righe che formano una corrispondenza.

Infatti non dobbiamo scrivere necessariamente query efficienti poiché è il DBMS ad ottimizzarle, pertanto è più importante che le query siano facili da capire.

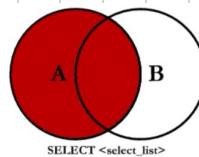
## JOIN STATEMENT

- **INNER JOIN**: esplicito : `SELECT ... FROM R JOIN R2 ON R.A = R2.B`, implicito: `SELECT ... FROM R, R2 WHERE R.A = R2.B`

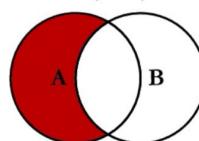
• **NATURAL JOIN**: si usa al posto del JOIN se due tabelle hanno colonne con lo stesso nome

• **OUTER JOIN**: per evitare la perdita di informazioni quando una tupla non trova il match

- a) **LEFT JOIN**: torna tutte le tuple della tabella a sinistra, quelle senza un match con la tabella a destra sono riempiti con valori NULL
- b) **RIGHT JOIN**: torna tutte le tuple della tabella a destra, quelle senza un match con la tabella a sinistra sono riempiti con valori NULL
- c) **FULL JOIN**: torna tutte le tuple sia della prima che della seconda tabella, dove non c'è un match troviamo valori NULL.



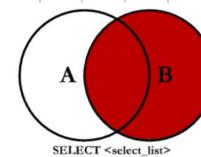
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



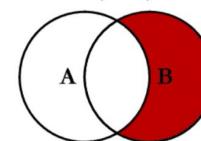
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



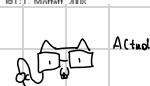
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



## ORDINAMENTO

→ ORDER BY AttrDiOrdinamento [ asc | desc ]

permette di specificare un ordinamento delle righe del risultato di una query

ASC → ascending order (default)

DESC → descending order

## UNION, INTERSECTION, DIFFERENCE

• UNION : SELECT ...

UNION [ ALL ] → si definisce tutte le righe sono uniche, tranne quando viene utilizzato ALL (multi-set union)

SELECT ...

quando due tabeles hanno schemi diversi si assumono i nomi degli attributi del primo operando

• DIFFERENCE : SELECT ... → è possibile esprimere la differenza anche attraverso query annidate

FROM ... • INTERSECTION SELECT ...

EXCEPT

FROM ...

SELECT R.A

SELECT ...

INTERSECT

FROM R, R1

FROM ...

SELECT ...

WHERE R.A=R1.B

FROM ...

## QUERY ANNIDATE

Possono essere formulate utilizzando i predicati ANY o ALL con gli operatori ( $>$ ,  $<$ ,  $=$ , ...)

Attribute op ANY / ALL (Expr)

• IN : la tupla della query esterna e' un match se il suo valore e' contenuto tra gli elementi ritornati da Expr

Attribute IN (Expr)

ANY, ALL, IN possono essere negati con IN

→ A IN (Expr)  $\equiv$  A = ANY (Expr)

A NOT IN (Expr)  $\equiv$  A  $\neq$  ALL (Expr)

## ■ VISIBILITÀ :

Non e' possibile fare riferimento a variabili dichiarate all'interno dei blocchi interni. Se il nome di una variabile e' omesso, prendiamo far dichiarazione più "vicina". Possiamo fare riferimento a variabili definite:

• mello scope della query in cui e' definita (blocchi esterni)

• mello scope di una query annidata, a qualsiasi livello (blocco interno) all'interno di esso.

La query interna viene eseguita una volta per ogni tupla dell'interno della query esterna, l'unico modo per evitare e' creare una vista, che va però a modificare lo schema del database.

## EXISTENTIAL QUANTIFICATION

usually query annidata

EXISTS ( Expr ) Il predicato e' vero se Expr torna almeno una tupla