

# L'Algebra Relazionale

[alcune slide tratte da <http://www-db.disi.unibo.it/courses/SIG/algebra2.pdf>  
e <https://dbdmg.polito.it/wordpress/wp-content/uploads/2010/12/2.2-AlgebraRelazionale-PS.pdf> ]

# Linguaggi di interrogazione per il modello relazionale

- Un linguaggio di interrogazione (*query language*) per il modello relazionale è un linguaggio specializzato per manipolare (tipicamente estrarre) dati di una base di dati relazionale
- Tali linguaggi possono essere distinti in:
  - **Procedurali**: ogni interrogazione descrive passo a passo cosa fare per ottenere la risposta desiderata
  - **Dichiarativi**: l'interrogazione descrive il risultato desiderato, senza però descrivere cosa deve essere fatto per ottenere la risposta
- L'Algebra Relazionale (AR) è un linguaggio procedurale

# Perché studiare l'Algebra Relazionale?

- **Fondamenta teoriche per le operazioni sui dati**
  - Base teorica solida per comprendere e utilizzare le operazioni fondamentali sui dati
- **Ottimizzazione delle query**
  - Ottimizzazione delle query per migliorare le prestazioni del database
- **Portabilità tra sistemi di database**
  - L'algebra relazionale è un modello teorico che non dipende da un particolare sistema di gestione di database (DBMS)
- **Comprensione degli algoritmi di esecuzione**
  - Comprendere come i DBMS eseguono fisicamente le operazioni sui dati

## AR: concetti generali

- In AR, input e output di ogni interrogazione sono *relazioni* (nel senso del modello relazionale)
- Il risultato di un'operazione è una *nuova relazione*, che viene generata a partire da una o più relazioni in *input*
- Questa proprietà rende l'AR un'algebra “chiusa”
  - Tutti gli oggetti in AR sono relazioni

## AR: concetti generali (cont.)

- Le relazioni ottenute da una qualsiasi operazione di AR possono a loro volta diventare input di nuove operazioni della stessa algebra
- Una sequenza di operazioni di AR forma un'**espressione di AR**. Per esempio:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'}Boats) \bowtie Reserves) \bowtie Sailors)$$

- Il risultato di un'espressione di AR è a sua volta una relazione che rappresenta il risultato di un'interrogazione a una base di dati

# Le operazioni dell'AR

- Operazioni unarie
  - SELECT (simbolo:  $\sigma$  (sigma))
  - PROJECT (simbolo:  $\pi$  (pi greco))
  - RENAME (simbolo:  $\rho$  (rho))
- Operazioni insiemistiche di AR
  - UNIONE (  $\cup$  ), INTERSEZIONE (  $\cap$  ), DIFFERENZA o SOTTRAZIONE (  $-$  )
  - PRODOTTO CARTESIANO (  $\times$  )
- Operazioni binarie
  - JOIN
  - DIVISIONE
- Altre operazioni (che vedremo solo in parte)
  - OUTER JOINS, OUTER UNION
  - FUNZIONI AGGREGATE (come SUM, COUNT, AVG, MIN, MAX)

# ESEMPIO

- Nel seguito useremo il seguente schema:

Sailors(sid: integer, sname: string, rating: integer, age: real)  
Boats(bid: integer, bname: string, color: string)  
Reserves(sid: integer, bid: integer, day: date)

assumendo che contenga i seguenti dati:

*S1*

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

*S2*

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

*R1*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

# Selezione

- L'operatore di **selezione**,  $\sigma$ , permette di selezionare un **sottoinsieme delle tuple di una relazione**, applicando a ciascuna di esse una formula booleana  $F$

Espressione:  $\sigma_F(R)$

Schema	$R(X)$	$X$
Istanza	$r$	$\sigma_F(r) = \{ t \mid t \in r \text{ AND } F(t) = \text{vero} \}$
	Input	Output

- $F$  si compone di **predicati** connessi da AND ( $\wedge$ ), OR ( $\vee$ ) e NOT ( $\neg$ )
- Ogni predicato è del tipo  $A \theta c$  o  $A \theta B$ , dove:
  - $A$  e  $B$  sono attributi in  $X$
  - $c \in \text{dom}(A)$  è una costante
  - $\theta$  è un operatore di confronto,  $\theta \in \{=, \neq, <, >, \leq, \geq\}$



# SELECT: esempio

- Trovare in S2 tutti i SAILOR che hanno giudizi superiori a 8:

$$\sigma_{\text{rating} > 8}(S2)$$

- Risultato:

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0



sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

- Stesso numero di attributi, selezionate le due (sole) tuple che soddisfano la condizione specificata nel SELECT (ovvero *rating* > 8)

# Proprietà dell'operatore SELECT

- $\sigma_{\langle \text{select\_cond} \rangle}(R)$  produce una relazione S che ha lo stesso schema (cioè gli stessi attributi) di R
- SELECT è commutativo:
  - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(R)) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R))$
- Di conseguenza, l'ordine di applicazioni non conta:
  - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R)))$
- Una sequenza di SELECT può essere sostituita da una singola operazione con congiunzione delle condizioni:
  - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond1} \rangle \text{ AND } \langle \text{cond2} \rangle \text{ AND } \langle \text{cond3} \rangle}(R))$
- Il numero di tuple prodotte dal SELECT è  $\leq$  al numero di tuple dell'istanza della relazione R in input

# Selezione: esempi (1)

Esami

Matricola	CodCorso	Voto	Lode
29323	483	28	NO
39654	729	30	Sì
29323	913	26	NO
35467	913	30	NO
31283	729	30	NO

$\sigma_{(\text{Voto} = 30) \text{ AND } (\text{Lode} = \text{NO})}(\text{Esami})$

Matricola	CodCorso	Voto	Lode
35467	913	30	NO
31283	729	30	NO

$\sigma_{(\text{CodCorso} = 729) \text{ OR } (\text{Voto} = 30)}(\text{Esami})$

Matricola	CodCorso	Voto	Lode
39654	729	30	Sì
35467	913	30	NO
31283	729	30	NO

## Selezione: esempi (2)

Partite

Giornata	Casa	Ospite	GolCasa	GolOspite
4	Venezia	Bologna	0	1
5	Brescia	Atalanta	3	3
5	Inter	Bologna	1	0
5	Lazio	Parma	0	0

$\sigma_{(\text{Giornata} = 5) \text{ AND } (\text{GolCasa} = \text{GolOspite})}(\text{Partite})$

Giornata	Casa	Ospite	GolCasa	GolOspite
5	Brescia	Atalanta	3	3
5	Lazio	Parma	0	0

$\sigma_{(\text{Ospite} = \text{Bologna}) \text{ AND } (\text{GolCasa} < \text{GolOspite})}(\text{Partite})$

Giornata	Casa	Ospite	GolCasa	GolOspite
4	Venezia	Bologna	0	1

# Proiezione

- L'operatore di **proiezione**,  $\pi$ , è ortogonale alla selezione, in quanto permette di selezionare un **sottoinsieme Y degli attributi di una relazione**

Espressione:  $\pi_Y(R)$

Schema	R(X)	Y
Istanza	r	$\pi_Y(r) = \{ t[Y] \mid t \in r \}$

Input                      Output



# PROJECT: esempio

- Trovare nome e giudizio di tutti i SAILORS (istanza S2):

$\pi_{\text{sname}, \text{rating}}(\text{S2})$

- Risultato:

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0



<i>sname</i>	<i>rating</i>
yuppy	9
Lubber	8
guppy	5
Rusty	10

# PROJECT: proprietà

- La forma generale dell'operazione *project* è:

$$\pi_{\langle \text{lista\_di\_attributi} \rangle}(R)$$

- $\pi$  è l'operatore di proiezione
  - $\langle \text{lista\_di\_attributi} \rangle$  è la lista degli attributi di R che si vogliono mantenere nella relazione in output
- 
- L'operatore PROJECT **rimuove le tuple duplicate!!**
    - Questo perché **il risultato del PROJECT deve essere un insieme di tuple**
      - Matematicamente, un insieme non ammette elementi duplicati!

# PROJECT: proprietà

- Il numero di tuple in  $\pi_{\langle \text{list} \rangle}(R)$  è uguale al numero di tuple di  $R$  o minore (se sono stati eliminati eventuali duplicati)
  - Se *list* include una *chiave* di  $R$ , allora il numero di tuple restituite da PROJECT sarà sempre *uguale* al numero di tuple di  $R$
- **PROJECT non è commutativo!**
  - $\pi_{\langle \text{list1} \rangle}(\pi_{\langle \text{list2} \rangle}(R)) = \pi_{\langle \text{list2} \rangle}(\pi_{\langle \text{list1} \rangle}(R))$  solo se  $\langle \text{list2} \rangle$  contiene gli attributi di  $\langle \text{list1} \rangle$



# Proiezione: esempi (1)

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

$\pi_{\text{CodCorso, Docente}}(\text{Corsi})$

CodCorso	Docente
483	Biondi
729	Neri
913	Castani

$\pi_{\text{CodCorso, Anno}}(\text{Corsi})$

CodCorso	Anno
483	1
729	1
913	2

# Proiezione: esempi (2)

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

$\pi_{\text{Titolo}}(\text{Corsi})$

Titolo
Analisi
Sistemi Informativi

$\pi_{\text{Docente}}(\text{Corsi})$

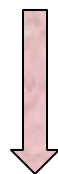
Docente
Biondi
Neri
Castani

# Espressioni in AR

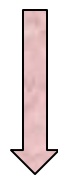
- E' possibile applicare in sequenza diversi operatori di AR:
  - Un'espressione può essere ottenuta:
    1. annidando (*nesting*) gli operatori in un'unica espressione, o
    2. applicando gli operatori uno alla volta, creando di volta in volta **relazioni intermedie** (vedremo come quando introdurremmo la ridenominazione)
- Nel secondo caso, dobbiamo assegnare nomi alle relazioni intermedie.

# Espressioni in AR

$\pi_{sname, rating}(\sigma_{rating > 8}(S2))$



sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0



sname	rating
yuppy	9
rusty	10

# Operazioni insiemistiche

- **UNIONE** ( $R \cup S$ ): è la relazione che include tutte le tuple che sono in R o in S o in entrambe
  - I duplicati sono eliminati
- **INTERSEZIONE** ( $R \cap S$ ): è la relazione che include tutte le tuple che sono sia in R sia in S
- **DIFFERENZA** ( $R - S$ ): è la relazione che include tutte le tuple che sono in R ma non in S
  - Per convenzione, se i nomi degli attributi sono diversi in R e in S, l'output utilizza i nomi di R
- **PRODOTTO CARTESIANO** ( $R \cdot S$ ): è la relazione che ha come schema l'unione degli attribute di R e S e una tupla  $\langle r, s \rangle$  (la concatenazione di  $r$  e  $s$ ) per ogni coppia di tuple  $r \in R$  e  $s \in S$

# UNIONE, INTERSEZIONE, DIFFERENZA: compatibilità dei domini

## NOTA BENE:

- Le relazioni in input R e S devono essere “compatibili”, ovvero:
  - R ed S devono avere lo stesso numero  $n$  di attributi
  - gli attributi corrispondenti di R e S devono avere lo stesso dominio (o domini compatibili)  
$$\text{dom}(A_i) = \text{dom}(B_i) \text{ per } i=1, 2, \dots, n)$$

# Esempi:

$S1$

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

$S2$

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

INPUT

---

OUTPUT

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

# Unione e differenza: esempi

VoliCharter

Codice	Data
XY123	21/07/2001
SC278	28/07/2001
XX338	18/08/2001

VoliNoSmoking

Codice	Data
SC278	28/07/2001
SC315	30/07/2001

VoliCharter  $\cup$  VoliNoSmoking

Codice	Data
XY123	21/07/2001
SC278	28/07/2001
XX338	18/08/2001
SC315	30/07/2001

VoliCharter - VoliNoSmoking

Codice	Data
XY123	21/07/2001
XX338	18/08/2001

VoliNoSmoking - VoliCharter

Codice	Data
SC315	30/07/2001



# Alcune proprietà

- UNION e INTERSECTION sono operazioni *commutative*:
  - $R \cup S = S \cup R$  e  $R \cap S = S \cap R$
- UNION e INTERSECTION sono operazioni *associative* e possono quindi essere pensate come operazioni *n-arie*:
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- SET DIFFERENCE non è commutativa, per cui in generale:
  - $R - S \neq S - R$

# Operazioni insiemistiche: il PRODOTTO CARTESIANO (cross product)

- Serve a combinare tuple di due relazioni e si indica con il simbolo  $\times$

$$R(A_1, \dots, A_n) \times S(B_1, \dots, B_m)$$

- Il risultato è una nuova relazione  $Q$  di grado  $n + m$ :

$$Q(A_1, \dots, A_n, B_1, \dots, B_m)$$

con gli attributi esattamente in questo ordine

- Se  $R$  ha  $n_R$  tuple ( $|R| = n_R$ ) e  $S$  ha  $n_S$  tuple ( $|S| = n_S$ ), allora  $R \times S$  avrà  $n_R * n_S$  tuple ( $|R \times S| = n_R * n_S$ ).

# IL PRODOTTO CARTESIANO: esempio

*S1*

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

X

*R1*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96



***S1 x R1***

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

# L'operatore RENAME

- Nell'esempio di PRODOTTO CARTESIANO, è chiaro che c'è un conflitto di nomi tra gli attributi della relazione S1xR1 (sulla colonna *sid*)
- Per risolvere problemi di questo tipo, viene introdotto un operatore di ridenominazione (*rename*), denotato dal simbolo  $\rho$  (rho)

## RENAME (cont.)

- L'operatore di ridenominazione (RENAME) ha la seguente forma:
  - $\rho(R(F_1, \dots, F_n), E)$  dove:
    - E è una qualunque espressione in algebra relazionale
    - R è una nuova relazione che ha le stesse tuple di E ma con alcuni attributi rinominati
    - $(F_1, \dots, F_n)$  è la *lista di ridenominazione* e contiene espressioni della forma *vecchionome*  $\rightarrow$  *nuovonome* o *posizione*  $\rightarrow$  *nuovonome*
- Se il nome degli attributi non viene modificato, si può omettere la lista di ridenominazione

# Esempio di ridenominazione

***S1 x R1***

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

sid1	sname	rating	age	sid2	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

# Operazioni insiemistiche: il JOIN

- L'operazione di JOIN di due relazioni R e S (  $R \bowtie_c S$  ) può essere definita in termini di PRODOTTO CARTESIANO e SELEZIONE come segue:

$$\sigma_c(R \times S)$$

dove c è una condizione espressa con una formula booleana

- Il risultato è l'insieme delle combinazioni di tuple di R e S che soddisfano la condizione (il predicato) c
- Questa forma di JOIN (a.k.a.  $\theta$ -join / theta-join) è la forma più generale e permette di combinare in modo semanticamente sensato tuple che appartengono a relazioni diverse

# Alcune proprietà del JOIN

- Il JOIN è commutativo e associativo
- La cardinalità  $| R \bowtie_c S |$  della relazione risultante dal JOIN sarà  $\leq$  della cardinalità del prodotto cartesiano delle due relazioni ( $|R \times S|$ ), grazie al fatto che alcune delle combinazioni di tuple di  $R \times S$  potrebbero non rispettare la condizione  $c$  del JOIN (e normalmente è così!)



# Esempio di JOIN

$S1 \times R1$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

$S1 \bowtie R1$   
 $S1.sid < R1.sid$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

# Variante 1: EQUIJOIN

- È la forma di JOIN più comunemente utilizzata
- La condizione  $c$  include **soltanto** confronti di uguaglianza (=)
- Esempio:

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{R.sid=S.sid} R1$$

- NB: la colonna usata nella condizione dell'EQUIJOIN compare solo una volta nel risultato (nella parte di tupla che appartiene alla relazione di sinistra)

## Variante : il NATURAL JOIN

- Il NATURAL JOIN (denotato da  $R \bowtie S$ ) è un EQUIJOIN in cui l'uguaglianza è automaticamente definita su **tutti** gli attributi comuni di R e S
- La definizione standard assume che tutte le coppie di attributi su cui si effettua il JOIN abbiano lo stesso nome in entrambe le relazioni (es. *S1.sid* e *R1.sid*)
- Se così non fosse, va applicata prima un'operazione di ridenominazione

## Join naturale: definizione

- Ogni tupla che compare nel risultato del join naturale di  $r_1$  e  $r_2$ , istanze rispettivamente di  $R_1(X_1)$  e  $R_2(X_2)$ , è ottenuta come combinazione (“match”) di una tupla di  $r_1$  con una tupla di  $r_2$  sulla base dell’uguaglianza dei valori degli attributi comuni (cioè quelli in  $X_1 \cap X_2$ )
- Inoltre, lo schema del risultato è l’unione degli schemi degli operandi

Espressione: $R_1 \bowtie R_2$		
Schema	$R_1(X_1), R_2(X_2)$	$X_1 X_2$
Istanza	$r_1, r_2$	$r_1 \bowtie r_2 = \{ t \mid t[X_1] \in r_1 \text{ AND } t[X_2] \in r_2 \}$
	Input	Output

# Join naturale: osservazioni

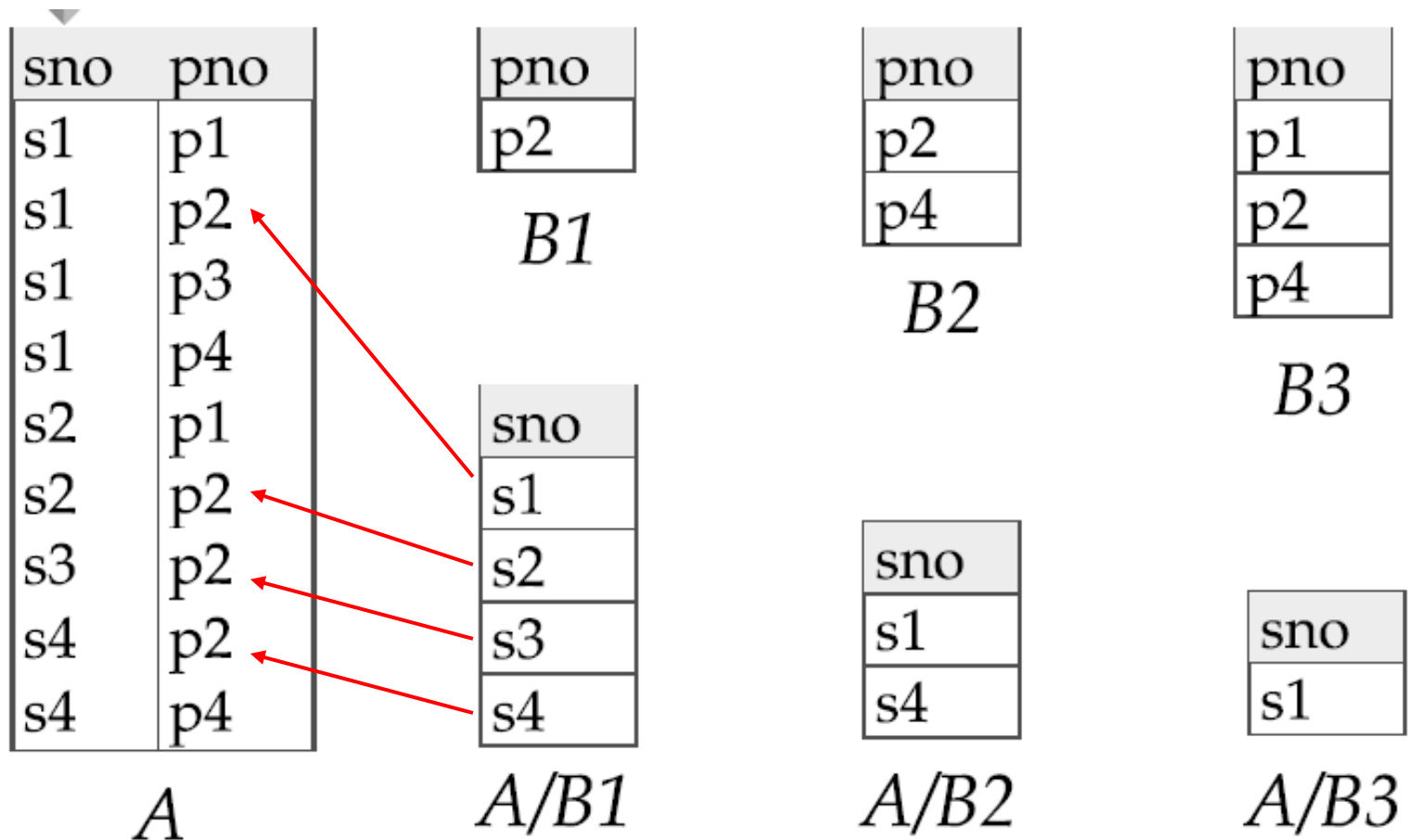
- È possibile che una tupla di una delle relazioni operande non faccia match con nessuna tupla dell'altra relazione; in tal caso tale tupla viene detta “**dangling**”
- Nel caso limite è quindi possibile che il risultato del join sia vuoto; all'altro estremo è possibile che ogni tupla di  $r_1$  si combini con ogni tupla di  $r_2$
- Ne segue che  
la cardinalità del join,  $|r_1 \bowtie r_2|$ , è compresa tra 0 e  $|r_1| * |r_2|$
- Se il join è eseguito su una superchiave di  $R_1(X_1)$ , allora ogni tupla di  $r_2$  fa match con al massimo una tupla di  $r_1$ , quindi  $|r_1 \bowtie r_2| \leq |r_2|$
- Se  $X_1 \cap X_2$  è la chiave primaria di  $R_1(X_1)$  e foreign key in  $R_2(X_2)$  (e quindi c'è un vincolo di integrità referenziale) allora  $|r_1 \bowtie r_2| = |r_2|$

# Operazioni binarie: la DIVISIONE

- La DIVISIONE non è un operatore primitivo e non tutti i DBMS la supportano
- Tuttavia, può essere utile per rappresentare query del tipo:  
*Trovare i marinai che hanno prenotato tutte le barche*
- Data una relazione A con due colonne x e y e B con una sola colonna y:

$$A/B = \{ \langle x \rangle \mid \exists \langle x, y \rangle \in \tilde{A} \ \forall \langle y \rangle \in B \}$$

# Esempio di DIVISIONE



# OUTER JOIN

- Nel NATURAL JOIN e EQUIJOIN, le tuple che **non** soddisfano le condizioni del JOIN vengono eliminate dal risultato.
  - Anche le tuple con valore NULL negli attributi del JOIN sono eliminate → Perdita di informazione
- Per superare questo limite, sono state introdotte delle operazioni, chiamate OUTER JOIN, che servono a mantenere alcune (o tutte) le tuple che non hanno un *match* nelle altre forme di JOIN

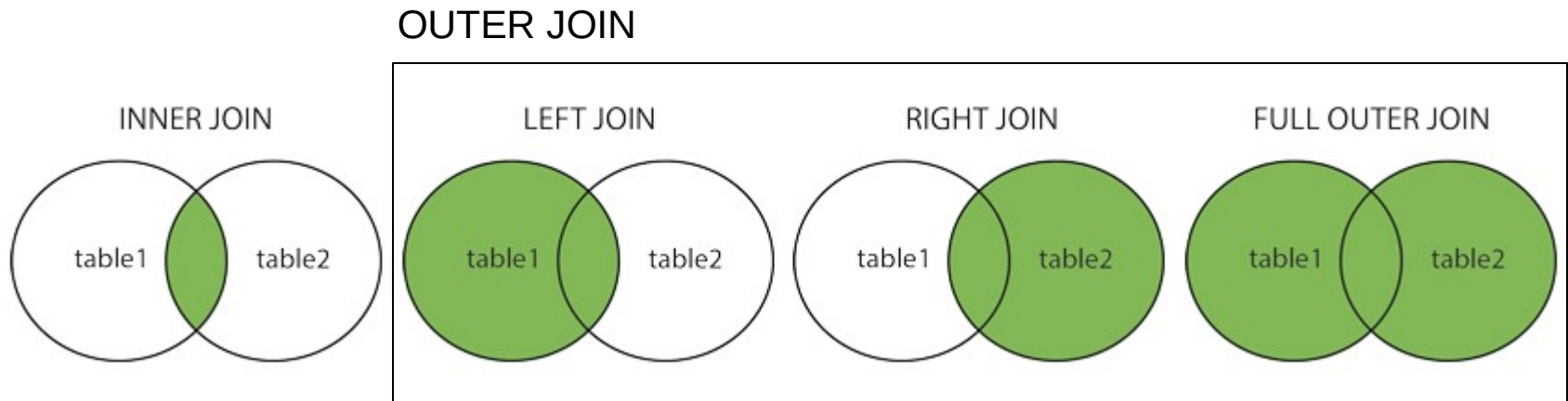


# OUTER JOIN

- Esistono 3 varianti dell'OUTER JOIN:
  - **LEFT OUTER JOIN:** vengono mantenute tutte le tuple del primo operando
  - **RIGHT OUTER JOIN:** vengono mantenute tutte le tuple del secondo operando
  - **FULL OUTER JOIN:** vengono mantenute le tuple di entrambi gli operandi
- In tutti e tre i casi, nella relazione risultante i valori mancanti sono sostituiti con il valore NULL

# I tre tipi di OUTER JOIN

- Dal punto di vista insiemistico, possiamo rappresentare come segue l'OUTER JOIN nelle sue possibili varianti:



# Outer join: esempi

Ricercatori

Nome	CodProgetto
Rossi	HK27
Bianchi	HK27
Verdi	HK28

Progetti

CodProgetto	Responsabile
HK27	Bianchi
HAL2000	Neri

Ricercatori  $\Rightarrow \Leftarrow$  Progetti

Nome	CodProgetto	Responsabile
Rossi	HK27	Bianchi
Bianchi	HK27	Bianchi
Verdi	HK28	NULL

Ricercatori  $\Rightarrow \Leftarrow =$  Progetti

Nome	CodProgetto	Responsabile
Rossi	HK27	Bianchi
Bianchi	HK27	Bianchi
Verdi	HK28	NULL
NULL	HAL2000	Neri

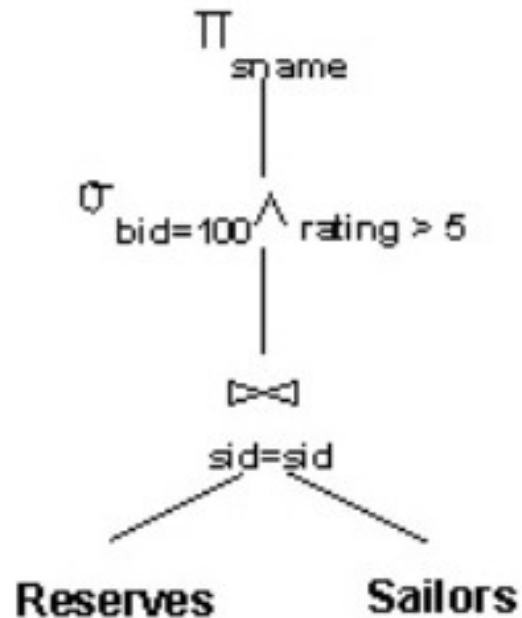
Ricercatori  $\triangleright \Leftarrow =$  Progetti

Nome	CodProgetto	Responsabile
Rossi	HK27	Bianchi
Bianchi	HK27	Bianchi
NULL	HAL2000	Neri

# Query Tree

- E' una struttura dati interna per rappresentare i passi di esecuzione di una query
- Standard per stimare il lavoro necessario per eseguire una query, la generazione di risultati intermedi e l'ottimizzazione dell'esecuzione
- I nodi stanno per le operazioni (selezione, proiezione, join, ....)
- Le foglie rappresentano la/le relazione/i di partenza
- Un albero dà una buona rappresentazione visiva della complessità della query e delle operazioni coinvolte

# Esempio di Query Tree



A quale domanda corrisponde questo Query Tree?

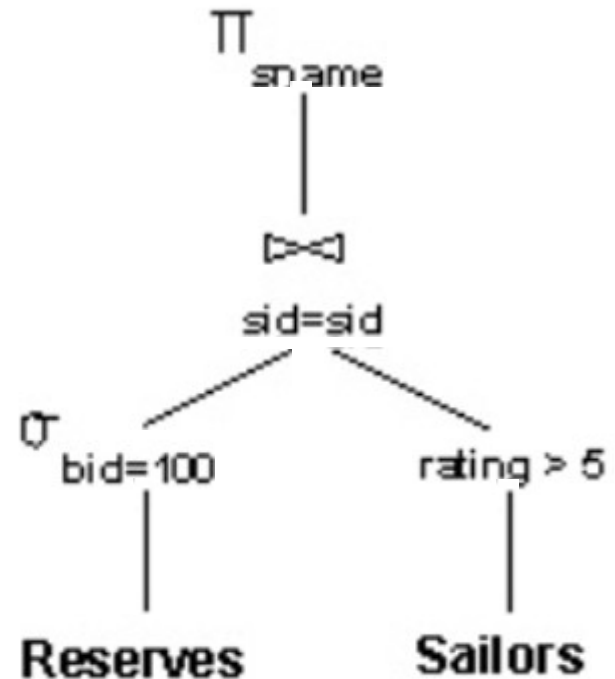
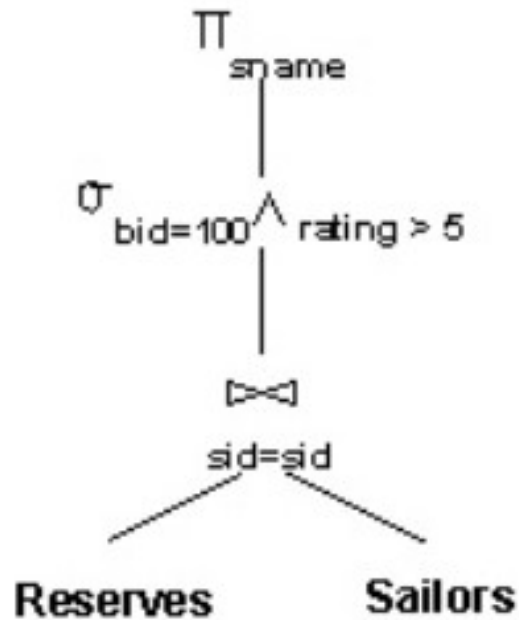
# Ottimizzazione algebrica dell'esecuzione di una query

- Obiettivo: minimizzare i costi di esecuzione di una query
- Si parte da un piano di esecuzione (rappresentato da un query tree)
- Si verifica se esista un altro piano di esecuzione che
  - produca lo stesso risultato di quello originale
  - abbia costi inferiori

# Ottimizzazione di esecuzione di query

• Possiamo ottimizzare il piano di esecuzione a sinistra?

• Perché (intuitivamente) il piano di esecuzione a destra è migliore di quello a sinistra?



# Esempi di (semplici) regole di ottimizzazione

- **Anticipazione delle selezioni** (*push selections down*):
  - spostare le operazioni di **selezione** il più vicino possibile alle tabelle di origine → applicare i filtri subito riduce il numero di righe coinvolte nelle operazioni successive, come i join o le proiezioni
- **Anticipazione delle proiezioni** (*push projections down*):
  - eliminando le colonne non necessarie prima di effettuare altre operazioni riduce lo spazio di memoria necessario per memorizzare i dati temporanei
- **Riordino dei join**
  - spesso è possibile riordinare le operazioni di join per eseguire prima quelle che producono meno righe intermedie, riducendo il numero di combinazioni da elaborare