

Insegnando la matematica agli altri finisci per comprenderla meglio tu. (Ian Stewart)

**JOIN** 

L'operatore JOIN nell'algebra relazionale:

$$R_1(X_1)$$
,  $R_2(X_2)$   
 $R_1$  JOIN  $R_2$  è una relazione su  $X_1X_2$ 

Questa definizione ci dice che le tuple del risultato sono ottenute combinando tuple degli operandi con valori uguali sugli attributi comuni

#### Esempio Prove scritte in un concorso

- I compiti sono anonimi e ad ognuno è associata una busta chiusa con il nome del candidato
- Ciascun compito e la relativa busta vengono contrassegnati con uno stesso numero

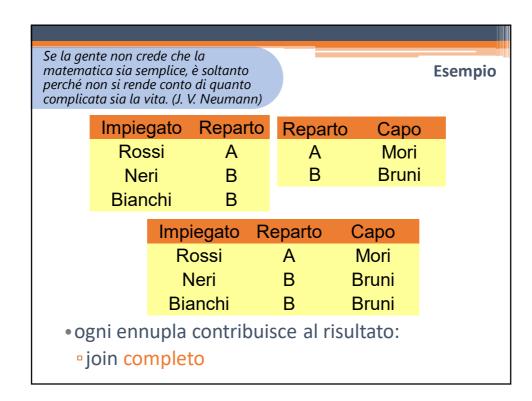
Numero	Voto	Numero	Candidato
1	25	1	Mario Rossi
2	13	2	Nicola Russo
3	27	3	Mario Bianchi
4	28	4	Remo Neri

Numero	Candidato	Voto
1	Mario Rossi	25
2	Nicola Russo	13
3	Mario Bianchi	27
4	Remo Neri	28

L'algebra e il denaro sono essenzialmente livellatori; la prima intellettualmente, l'altro effettivamente. (Simone Weil)

#### **JOIN NATURALE**

- Il JOIN naturale essenzialmente utile per riflessioni astratte ha la caratteristica di correlare dati di relazioni diverse sulla basi di valori uguali in **ATTRIBUTI CON LO STESSO NOME**
- operatore binario (generalizzabile)
- produce un risultato
  - sull'unione degli attributi degli operandi
  - con ennuple costruite ciascuna a partire da una ennupla di ognuno degli operandi



#### Prodotto cartesiano

 $R_1(X_1)$ ,  $R_2(X_2)$  con i due insiemi di attributi  $X_1$  e  $X_2$  sono disgiunti è delicato

il risultato è sempre  $X_1X_2$  e ciascuna tupla deriva sempre da due tuple degli operandi, ma atteso che  $X_1$  e  $X_2$  non hanno attributi in comune, non viene di fatto richiesta nessuna condizione per partecipare al JOIN (che normalmente ottiene tuple del risultato combinando tuple degli operandi con valori uguali sugli attributi comuni), degenera in una condizione sempre verificata.

In effetti, potremmo affermare che il JOIN degenera nel prodotto cartesiano

Ovvero l'operare prodotto cartesiano è definito esattamente come il JOIN NATURALE su relazioni senza attributi in comune. Ovviamente parliamo di concatenazione di tuple e non di coppie.

contiene sempre un numero di ennuple pari al prodotto delle cardinalità degli operandi (le ennuple sono tutte combinabili)

Esemp	io Prodotto c	artesiano			
	Impiegati		Reparti		
	Impiegato	Reparto	Codice	Capo	
	Rossi	Α	Α	Mori	
	Neri	В	В	Bruni	
	Bianchi	В			
	Impiegati Jo	OIN Repart	i		
	Impiegato	Reparto	Codice	Capo	
	Rossi	Α	Α	Mori	
	Rossi	Α	В	Bruni	
	Neri	В	Α	Mori	
	Neri	В	В	Bruni	
	Bianchi	В	Α	Mori	
	Bianchi	В	В	Bruni	

Esemp	io un join no	n comple	to		
	Impiegato	Reparto	Reparto	Capo	
	Rossi	Α	В	Mori	
	Neri	В	С	Bruni	
	Bianchi	В			
	Impie	gato Rep	oarto Ca	00	
	Ne		B Mo		
	Bian		B Mo		
				•••	





## Cardinalità del join

- Il join di R<sub>1</sub> e R<sub>2</sub> contiene un numero di ennuple ...
  - □ compreso fra zero e il prodotto di |R<sub>1</sub>| e |R<sub>2</sub>|
- se il join coinvolge una chiave di R<sub>2</sub>, allora il numero di ennuple è ...
  - □ compreso fra zero e |R<sub>1</sub>|
- se il join coinvolge una chiave di R<sub>2</sub> e un vincolo di integrità referenziale, allora il numero di ennuple è
  - pari a | R<sub>1</sub>|

# Cardinalità del join

- R<sub>1</sub>(A,B), R<sub>2</sub>(B,C)
- in generale

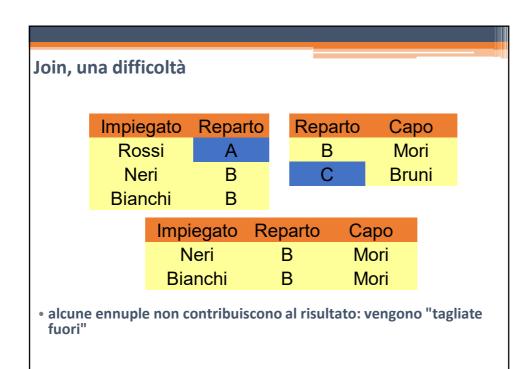
$$0 \le |R_1| |R_2| \le |R_1| \times |R_2|$$

• se B è chiave in R<sub>2</sub>

$$0 \le |R_1 \text{ JOIN } R_2| \le |R_1|$$

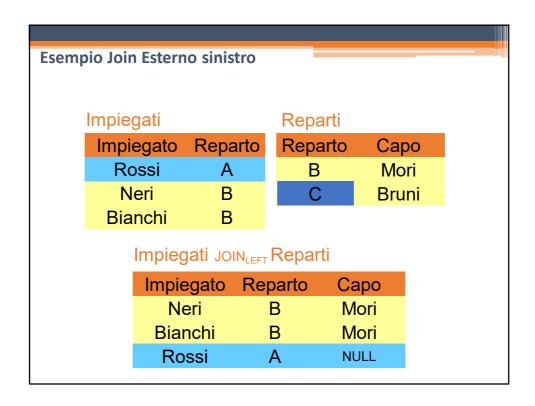
• se B è chiave in R<sub>2</sub> ed esiste vincolo di integrità referenziale fra B (in R<sub>1</sub>) e R<sub>2</sub>:

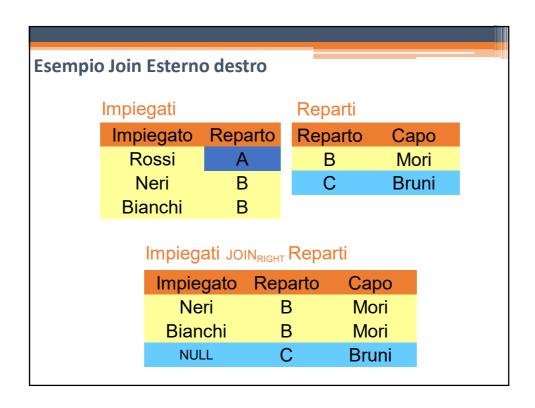
$$|R_1 \text{ JOIN } R_2| = |R_1|$$

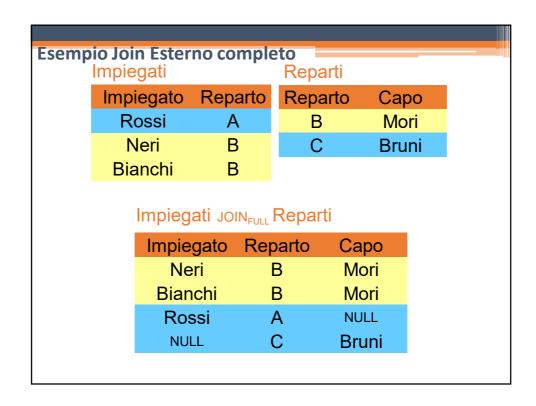


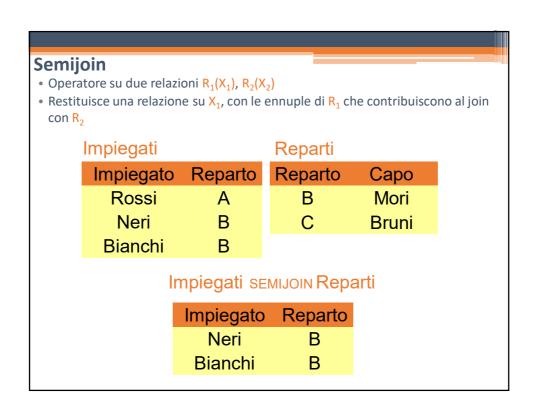
#### Join esterno

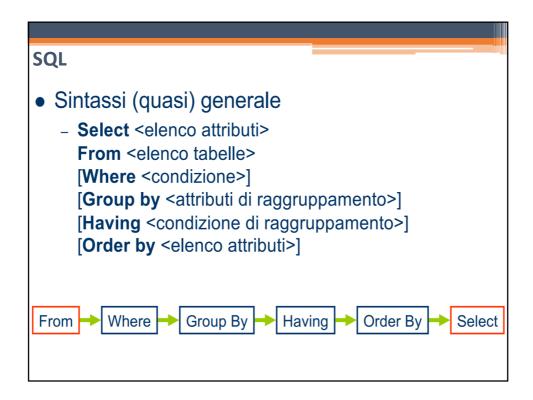
- Il join esterno estende, con valori nulli, le ennuple che verrebbero tagliate fuori da un join (interno)
- esiste in tre versioni:
  - sinistro, destro, completo
  - sinistro: mantiene tutte le ennuple del primo operando, estendendole con valori nulli, se necessario
  - destro: ... del secondo operando ...
  - ocompleto: ... di entrambi gli operandi ...











## **SQL** e Algebra Relazionale

In Pratica dati R1(A,B) e R2(C,D) Attraverso lo statement SQL realizzo:

FROM R1, R2 (prodotto cartesiano)
WHERE R1.B = R2.C (selezione)

 $PROJ_{A,D}$  (SEL<sub>B=C</sub> (R1 JOIN R2))

## SQL e le ridenominazioni

- Possono essere necessarie ridenominazioni
- Esse posso essere eseguite:
  - nel prodotto cartesiano e/o
  - nella target list (ovvero proiezione)

select X.A1 AS B1, ... from R1 X, R2 Y, R1 Z where X.A2 = Y.A3 AND ...

## SQL Esecuzione e specifica delle query

- Le espressioni SQL sono dichiarative e noi ne stiamo vedendo la semantica
- ➢ i DBMS eseguono le operazioni in modo efficiente, ad esempio:
  - ✓ eseguono le selezioni al più presto
  - ✓ se possibile, eseguono join e non prodotti cartesiani
  - La capacità dei DBMS di "ottimizzare" le interrogazioni, rende (di solito) non necessario preoccuparsi dell'efficienza quando si specifica un'interrogazione
  - È perciò più importante preoccuparsi della chiarezza (anche perché così è più difficile sbagliare ...)

#### **JOIN**

Join (SQL)

Il JOIN è una clausola del linguaggio SQL che serve a combinare le tuple di due o più relazioni di una base di dati.

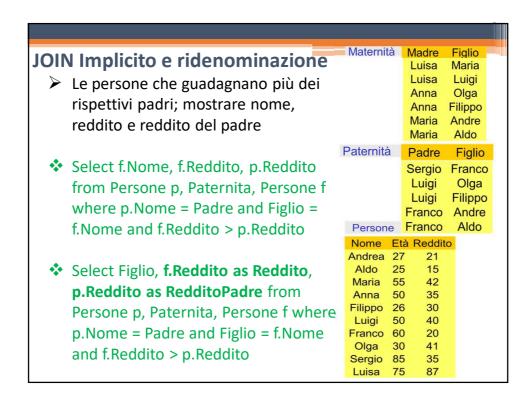
**SELECT** [DISTINCT] *lista Attributi* 

**FROM** Tabella

{Tipo JOIN Altra Tabella ON Condizione di Join}

[WHERE altra Condizione]

Lo standard ANSI definisce alcune specifiche per il linguaggio SQL sul tipo di JOIN da effettuare: INNER, FULL, LEFT e RIGHT, alle quali diversi DBMS aggiungono CROSS.





### JOIN

Lo standard ANSI definisce alcune specifiche per il linguaggio SQL sul tipo di JOIN da effettuare:

- ✓ INNER JOIN;
- ✓ FULL JOIN;
- ✓ LEFT JOIN;
- ✓ RIGHT JOIN

alle quali diversi DBMS aggiungono CROSS.

In alcuni casi è possibile, spesso è necessario, che una tabella debba essere combinata con se stessa, in questo caso si parlerà di SELF JOIN.

```
JOIN - Tabelle di riferimento
DROP TABLE IF EXISTS DIPARTIMENTI CASCADE;
CREATE TABLE DIPARTIMENTI (
ID_DIP CHAR(4) NOT NULL,
NOME VARCHAR (20) NOT NULL UNIQUE,
DIP IND VARCHAR (50),
CITTA VARCHAR (20),
CONSTRAINT PK DIPARTIMENTI PRIMARY KEY (ID DIP) );
/* INSERIRE OCCORRENZE IN UNA TABELLA DIPARTIMENTI */
INSERTINTO DIPARTIMENTI (ID DIP, NOME, DIP IND, CITTA)
VALUES ('AMMZ', 'AMMINISTRAZIONE', 'VIA TITO LIVIO, 27', 'MILANO');
INSERT INTO DIPARTIMENTI (ID_DIP, NOME, DIP_IND, CITTA)
VALUES ('PROD', 'PRODUZIONE', 'P.LE LAVATER, 3', 'TORINO');
INSERTINTO DIPARTIMENTI (ID_DIP, NOME, DIP_IND, CITTA)
VALUES ('DIST', 'DISTRIBUZIONE', 'VIA SEGRE, 9', 'ROMA');
INSERTINTO DIPARTIMENTI (ID DIP, NOME, DIP IND, CITTA)
VALUES ('DIRE', 'DIREZIONE', 'VIA TITO LIVIO, 27', 'MILANO');
INSERTINTO DIPARTIMENTI (ID DIP, NOME, DIP IND, CITTA)
VALUES ('RICE', 'RICERCA', 'VIA VENOSA, 6', 'MILANO');
```

JOI	IN – Tabelle	di riferimento		
40 41 42	FROM DIPA			
Dat	id_dip  [PK] character (4)	Notifications  nome character varying (20)	dip_ind character varying (50)	citta character varying (20)
1	AMMZ	AMMINISTRAZIONE	VIA TITO LIVIO, 27	MILANO
2	PROD	PRODUZIONE	P.LE LAVATER, 3	TORINO
3	DIST	DISTRIBUZIONE	VIA SEGRE, 9	ROMA
4	DIRE	DIREZIONE	VIA TITO LIVIO, 27	MILANO
5	RICE	RICERCA	VIA VENOSA, 6	MILANO

## JOIN - Tabelle di riferimento

#### JOIN - Tabelle di riferimento

/\* INSERIRE OCCORRENZE IN UNA TABELLA IMPIEGATI \*/

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000001', 'MARIO', 'ROSSI', 'AMMZ', 10, 45, 'MILANO');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000002', 'CARLO', 'BIANCHI', 'PROD', 20, 36, 'TORINO');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000003', 'GIOVANNI', 'VERDI', 'AMMZ', 20, 40, 'ROMA');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000004', 'FRANCO', 'NERI', 'DIST', 16, 45, 'NAPOLI');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000005', 'CARLO', 'ROSSI', 'DIRE', 14, 80, 'MILANO');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000006', 'LORENZO', 'GIALLI', 'DIRE', 7, 73, 'GENOVA');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000007', 'PAOLA', 'ROSATI', 'AMMZ', 75, 40, 'VENEZIA');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000008', 'MARCO', 'FRANCO', 'PROD', 20, 46, 'ROMA');

INSERT INTO IMPIEGATI (MATRICOLA, NOME, COGNOME, ID\_DIP, UFFICIO, STIPENDIO, CITTA) VALUES ('000009', 'MATTEO', 'GAETA', NULL, 20, 46, 'SALERNO');

9	99 SELECT *										
10	e FRO										
10	91										
10	0.3										
Dat	a Output	Explain N	Messages .	Notificat	Notifications						
4	matricola [PK] charact	nome character va	cognome character var	id_dip character	ufficio numeric	stipendio numeric (9	citta character				
1	000001	MARIO	ROSSI	AMMZ	10	45	MILANO				
2	000002	CARLO	BIANCHI	PROD	20	36	TORINO				
3	000003	GIOVANNI	VERDI	AMMZ	20	40	ROMA				
4	000004	FRANCO	NERI	DIST	16	45	NAPOLI				
5	000005	CARLO	ROSSI	DIRE	14	80	MILANO				
6	000006	LORENZO	GIALLI	DIRE	7	73	GENOVA				
7	000007	PAOLA	ROSATI	AMMZ	75	40	VENEZIA				
8	000008	MARCO	FRANCO	PROD	20	46	ROMA				
	000009	MATTEO	GAFTA	[null]	20	46	SALERNO				

## JOIN - Tabelle di riferimento

Nella Tabella IMPIEGATI vi possono essere tra zero e N tuple che fanno riferimento alla chiave primaria della tabella DIPARTIMENTI.

Infatti un valore di ID\_DIP in tabella IMPIEGATI può ripetersi più volte, ma deve sempre essere presente nella Tabella DIPARTIMENTI e corrispondere ad un valore presente nella chiave primaria di DIPARTIMENTI oppure può assumere il valore NULL (l'impiegato non è stato ancora assegnato a nessun dipartimento)

Nota: Il Dipartimento "*Ricerca*" ID\_DIP="RICE" della tabella "DIPARTIMENTI" non ha alcuna corrispondenza nella tabella "IMPIEGATI". Mentre l'impiegato "Gaeta" non è stato assegnato ad alcun DIPARTIMENTO, infatti ID\_DIP=NULL.

#### **INNER JOIN**

L'Inner Join si distingue da tutti gli altri tipi di Join per il suo fornire un **risultato minimale**.

Il risultato di una Inner Join è costituito solamente dai record di dati (Tuple, Righe) dell'unione incrociata che soddisfano la condizione di selezione.

Considerando che l'Inner Join è il Join SQL più importante, si può omettere la parola chiave "inner".

#### **INNER JOIN**

Realizzando una Inner Join tra due Tabelle, se si confronta la tabella dei risultati con le due tabelle di output, si noterà che possono mancare alcune Tuple per ogni tabella coinvolta nel Join.

Si tratta di quei record di dati che nei campi dichiarati dalla condizione di Inner Join non contengono valori corrispondenti tra le Tabelle coinvolte nel Join.

Quando vogliamo rilevare esattamente tali irregolarità e renderle visibili nell'interrogazione, dobbiamo utilizzare una outer join al posto di una inner join.

## **INNER JOIN**

Una INNER JOIN crea una nuova tabella combinando i valori delle due tabelle di partenza (DIPARTIMENTI, IMPIEGATI) basandosi su una certa regola di confronto.

La query compara ogni riga della tabella DIPARTIMENTI con ciascuna riga della tabella IMPIEGATI cercando di soddisfare la regola di confronto definita.

Quando la REGOLA DI JOIN viene soddisfatta, i valori di tutte le colonne delle tabelle DIPARTIMENTI e IMPIEGATI vengono combinate in un'unica riga nella costruzione della tabella risultante.

La INNER JOIN è la forma di JOIN usata più di frequente nelle applicazioni e rappresenta la modalità predefinita.

Il termine INNER si può omettere.

#### **INNER JOIN**

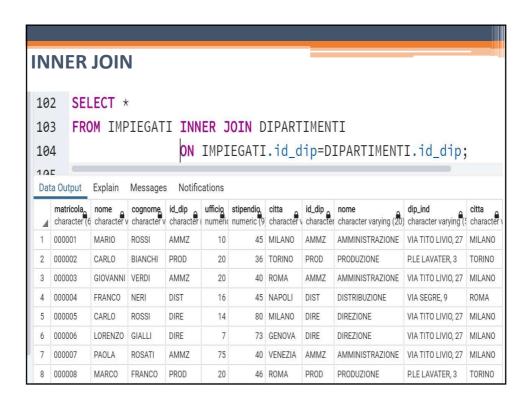
EFFETTUARE UN INNER JOIN TRA LA TABELLA DIPARTIMENTI E LA TABELLA IMPIEGATI SULLE COLONNE DIPARTIMENTI.**ID DIP** 

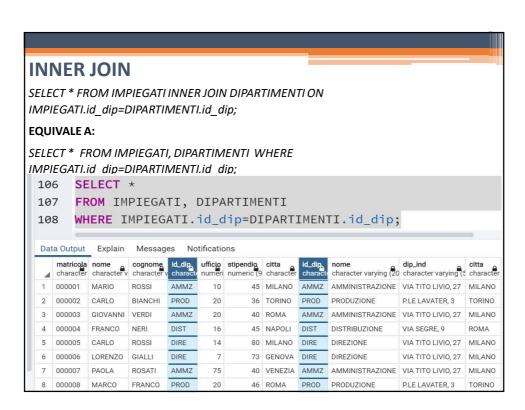
Ε

IMPIEGATI.**ID\_DIP**CON LA REGOLA JOIN
<< DIPARTIMENTI.ID DIP = IMPIEGATI.ID DIP >>

Esempio di INNER JOIN in forma esplicita:

SELECT \* FROM IMPIEGATI **INNER JOIN** DIPARTIMENTI **ON** IMPIEGATI.id\_dip=DIPARTIMENTI.id\_dip;





#### **INNER JOIN**

Osserviamo che l'impiegato "Gaeta" e il Dipartimento "Ricerca" non sono presenti nel risultato della Query

La tupla di «Gaeta» ha NULL come valore nella colonna ID\_DIP nella Tabella IMPIEGATI,

mentre la tupla di «Ricerca» pur essendo presente in tabella DIPARTIMENTI e correttamente valorizzata in ogni suo campo non ha nessun valore corrispondente in nessuna tupla della Tabella IMPIEGATI alla colonna ID\_DIP.

Quando come risultato finale si desidera avere anche i record che non hanno corrispondenza si una query con **OUTER JOIN**.

#### **INNER JOIN**

Le inner join si possono realizzare sotto forma di theta join, equi join, non equi join e natural join.

#### Theta join, equi join e non equi join

L'inner join della terminologia SQL corrisponde al theta join dell'algebra relazionale.

La theta join si differenzia da equi join e non equi join, in quanto fornisce agli utenti un numero illimitato di operatori di confronto tra cui scegliere.

**Le equi join**, invece, limitano la condizione di selezione per le query sull'uguaglianza dei valori delle colonne.

Le non equi join, invece, ammettono tutti gli operatori di confronto ad eccezione del segno uguale.

THETA JOIN - O	peratori di	Confronto
----------------	-------------	-----------

Tipo di

join Operatori di confronto ammessi

Theta join = (uguale) < (minore) > (maggiore)≤ (minore o uguale)≥ (maggiore o uguale)<> (diseguale)!= (diseguale)

Equijoin = (uguale)

Non equi < (minore) > (maggiore)≤ (minore o uguale)≥ (maggiore o uguale)<> (diseguale)!= (diseguale) ioin

## THETA JOIN - Operatori di Confronto

<u>Se due tabelle sono collegate da colonne con lo stesso</u> <u>nome,</u> le inner join vengono solitamente convertite in natural join.

### Le natural join sono una sottocategoria di equi join.

Come anche l'equi join, la natural join prevede l'uguaglianza dei valori delle due colonne come condizione di selezione.

L'operatore natural join collega automaticamente le tabelle utilizzando colonne con lo stesso nome.

La condizione di selezione non deve essere definita esplicitamente.

## THETA JOIN – Operatori di Confronto

#### **OSSERVAZIONE IMPORTANTE**

La tabella dei risultati della natural join si differenzia da quella della classica inner join, in quanto le colonne con lo stesso nome nelle tabelle di output non sono elencate due volte, ma vengono unite in una colonna comune.

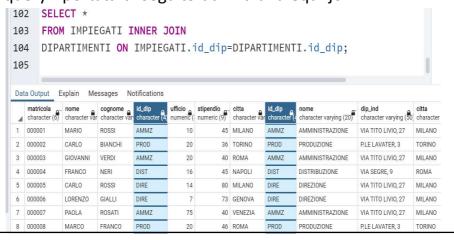
Per le natural join è disponibile una notazione abbreviata che non richiede una clausola USING.

Si utilizza invece l'operatore natural join.

#### **EQUIJOIN**

La EQUI JOIN, ha un particolare tipo di comparatore, detto THETA JOIN, che utilizza come metodo di verifica, solamente l'uguaglianza matematica come regola di confronto.

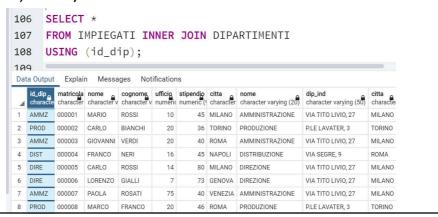
La query riportata di seguito utilizza una equi-join:



#### **EQUIJOIN**

SQL fornisce altre opzioni attraverso la keyword **USING**SELECT \* FROM IMPIEGATI **INNER JOIN** DIPARTIMENTI **USING** (**id\_dip**);

<u>PS: Ricordarsi delle parentesi quando si utilizza la USING</u>
USING è supportata da Microsoft, MySQL, Oracle, PostgreSQL, SQLite, e DB2/400.



#### **Natural Join**

il **natural join** offre ulteriori specializzazioni di equi-join. Solitamente il JOIN confronta colonne di tabelle diverse che hanno lo stesso nome.

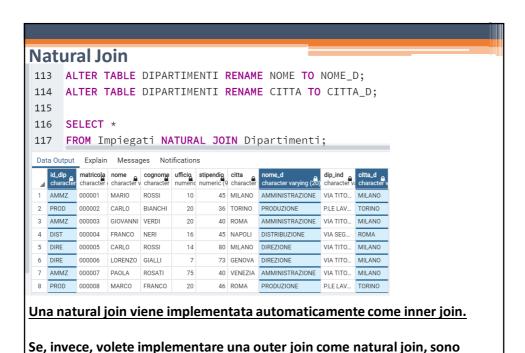
il natural join effettua questa operazione.

Se considerate che nel nostro caso, le Tabella IMPIEGATI e DIPARTIMENTI hanno i seguenti campi in comune: IN TABELLA DIPARTIMENTI ID\_DIP CHAR(4) NOT NULL, NOME VARCHAR(20) – contiene il nome del Dipartimento; CITTA VARCHAR (20)contiene la citta sede del Dipartimento, mentre in TABELLA IMPIEGATI NOME VARCHAR(20) – contiene il nome dell'impiegato; CITTA VARCHAR(20)contiene magari la città dove risiede -, osserverete che se si effettua un Natural Join sulle tabelle precedentemente definite il risultato sarà:



Provate a MODIFICARE i nomi delle colonne con lo stesso nome in entrambe le tabelle, ovvero modificate il nome delle colonne NOME e CITTA in tabella DIPARTIMENTI ad es. NOME in NOME\_D e CITTA in CITTA\_D.

In tal modo esse avranno solo il campo ID-DIP in comune. Poi verificate che questo natural join equivale alla precedente equi-join (USING) che abbiamo realizzato



necessarie parole chiave aggiuntive (ad esempio una natural left outer join).

#### **Cross Join**

Una cross join, cartesian join o product fornisce le strutture attraverso cui tutti i tipi di inner join operano.

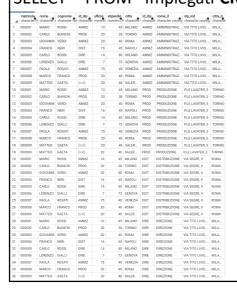
Il risultato di una cross join è il prodotto cartesiano di tutte le righe delle tabelle che concorrono alla query di join.

È come dire che stiamo facendo una inner join senza impostare la regola di confronto o in cui la regola di confronto ritorna sempre vero.

## **Cross Join**

Esempio di cross join esplicito:

SELECT \* FROM Impiegati CROSS JOIN Dipartimenti;



36	000009	MATTEO	GAETA	[null]	20	46	SALER	DIRE	DIREZIONE	VIA TITO LIVIO,	MILA
37	000001	MARIO	ROSSI	AMMZ	10	45	MILANO	RICE	RICERCA	VIA VENOSA, 6	MILA
38	000002	CARLO	BIANCHI	PROD	20	36	TORINO	RICE	RICERCA	VIA VENOSA, 6	MILA
39	000003	GIOVANNI	VERDI	AMMZ	20	40	ROMA	RICE	RICERCA	VIA VENOSA, 6	MILA
40	000004	FRANCO	NERI	DIST	16	45	NAPOLI	RICE	RICERCA	VIA VENOSA, 6	MILA
41	000005	CARLO	ROSSI	DIRE	14	80	MILANO	RICE	RICERCA	VIA VENOSA, 6	MILA
42	000006	LORENZO	GIALLI	DIRE	7	73	GENOVA	RICE	RICERCA	VIA VENOSA, 6	MILA
43	000007	PAOLA	ROSATI	AMMZ	75	40	VENEZIA	RICE	RICERCA	VIA VENOSA, 6	MILA
44	800000	MARCO	FRANCO	PROD	20	46	ROMA	RICE	RICERCA	VIA VENOSA, 6	MILA
	000000	MATTER	CAPTA	familie	20		CALED	nuce	DICEDCA	100 ISSUED A	140.4

#### **Outer Join**

Una Outer Join non richiede che ci sia corrispondenza esatta tra le righe di due tabelle.

La tabella risultante da una Outer Join trattiene tutti quei record che non hanno alcuna corrispondenza tra le tabelle.

Le Outer Join si suddividono in Left Outer Join, Right Outer Join, e Full Outer Join, in base a quale sia la tabella di cui intendiamo trattenere i valori in caso di mancata corrispondenza della regola di confronto da (sinistra, destra, o entrambi).

In questo caso left (sinistra) e right (destra) si riferiscono ai due lati della keyword JOIN.

Si sottolinea come esista un ordine di esecuzione tra le condizioni specificate nella WHERE e quelle presenti nella ON.

Specificatamente le clausole presenti nella ON sono valutate ANTERIORMENTE all'esecuzione del join mentre le clausole nella where sono valutate SUCCESSIVAMENTE all'esecuzione del join.

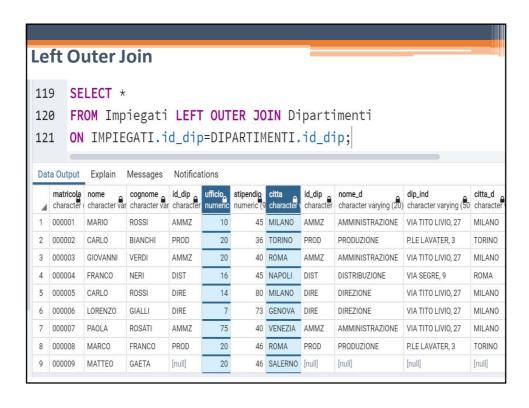
## Left Outer Join e Right Outer Join

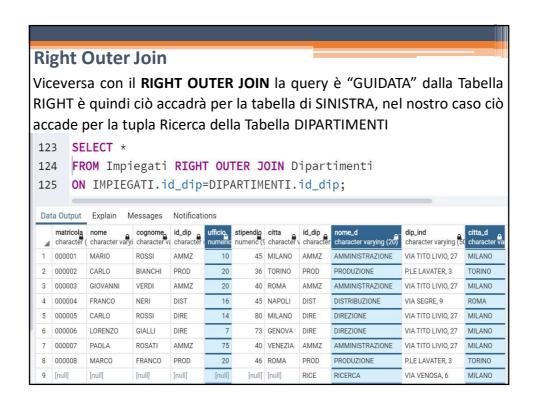
Il risultato di una query **LEFT OUTER JOIN** (o semplicemente left join) per le tabelle IMPIEGATI e DIPARTIMENTI contiene sempre tutti i record della tabella di sinistra ("left") IMPIEGATI, mentre vengono estratti dalla tabella di destra ("right") DIPARTIMENTI solamente le righe che trovano corrispondenza nella regola di confronto della join **ON IMPIEGATI.id dip=DIPARTIMENTI.id dip**.

Questo significa che la JOIN è "GUIDATA" dalla Tabella LEFT e che se la clausola ON trova zero righe nella tabella di DESTRA, la join mostrerà una riga risultante con valore NULL in tutte le colonne corrispondenti al risultato per le colonne di DESTRA.

Nel nostro caso ciò accade per la tupla Gaeta della Tabella IMPIEGATI.

In pratica la query con **RIGHT OUTER JOIN** (o right join) semplicemente ripropone il funzionamento della **LEFT OUTER JOIN**, a tabelle invertite.





#### **Full Outer Join**

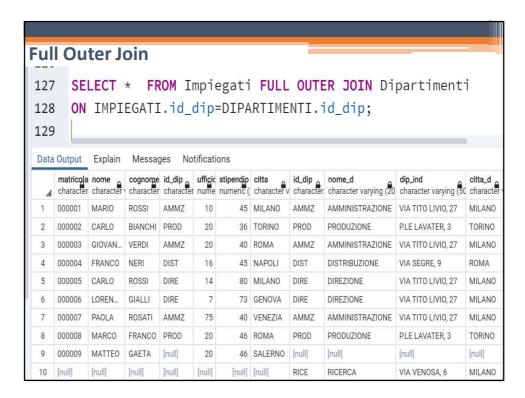
inoltre

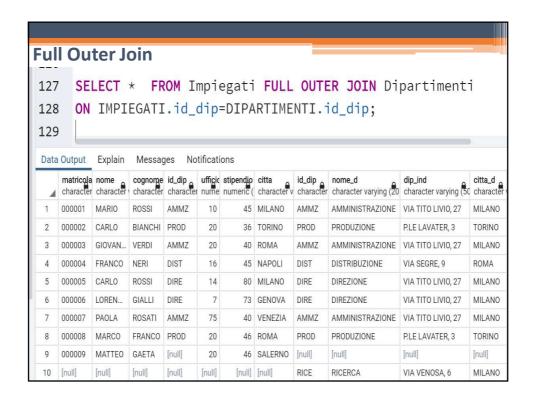
Una Query con Full Outer Join combina i risultati delle due tabelle di Sinistra e Destra tenendo conto di tutte le righe delle tabelle, anche di quelle che non hanno corrispondenza tra di loro.

Il risultato di un Full Outer Join per le tabelle IMPIEGATI e DIPARTIMENTI contiene tutte le righe della tabella left IMPIEGATI, estraendo dalla tabella right DIPARTIMENTI solamente le righe che trovano corrispondenza nella regola di confronto della join;

verranno estratti tutti i record della tabella di left IMPIEGATI che non trovano corrispondenza nella tabella right DIPARTIMENTI con NULL come valori di tutte le colonne della tabella DIPARTIMENTI

tutti i record della tabella di destra right DIPARTIMENTI che non trovano corrispondenza nella tabella di sinistra left IMPIEGATI con NULL come valori di tutte le colonne della tabella IMPIEGATI.





#### **Full Outer Join**

Alcuni database (MySQL) non supportano direttamente questa funzionalità, ma la si può emulare attraverso la combinazione di left e right outer join per utilizzando UNION.

SELECT \* FROM Impiegati LEFT JOIN Dipartimenti
ON IMPIEGATI.id\_dip=DIPARTIMENTI.id\_dip
UNION

SELECT \* FROM Impiegati RIGHT JOIN Dipartimenti
ON IMPIEGATI.id\_dip=DIPARTIMENTI.id\_dip
WHERE Impiegati.id\_dip IS NULL;

#### **Full Outer Join**

Allo stesso modo senza utilizzare nemmeno la query di **RIGHT OUTER JOIN**:

SELECT Impiegati.\*, Dipartimenti.\*

FROM Impiegati LEFT JOIN Dipartimenti ON Impiegati.id\_dip=Dipartimenti.id\_dip

UNION

SELECT Impiegati.\*, Dipartimenti.\*

FROM Dipartimenti LEFT JOIN Impiegati ON Impiegati.id\_dip=Dipartimenti.id\_dip
WHERE Impiegati.id\_dip IS NULL;

### **BASI DI DATI**

#### Materiale utilizzato e bibliografia

- > Le slide utilizzate dai docenti per le attività frontali sono in gran parte riconducibili e riprese dalle slide originali (con alcuni spunti parziali ripresi dai libri indicati) realizzate da:
- √ autori del libro Basi di Dati (Atzeni e altri) testo di riferimento del corso Basi di Dati e sono reperibili su internet su
  molteplici link oltre che laddove indicato dagli stessi autori del libro;
- ✓ Prof.ssa Tiziana Catarci e dal dott. Ing. Francesco Leotta corso di Basi di Dati dell'Università degli Studi La Sapienza di Roma al seguente link ed altri: <a href="http://www.dis.uniroma1.it/~catarci/basidatGEST.html">http://www.dis.uniroma1.it/~catarci/basidatGEST.html</a> (molto Interessanti anche le lezioni su YouTube).
- ✓ Proff. Luca Allulli e Umberto Nanni, Libro Fondamenti di basi di dati, editore HOEPLI (testo di facile lettura ed efficace).
- Diverse slide su specifici argomenti utilizzate dai docenti per le attività frontali sono anche in parte riconducibili e riprese dalle slide originali facilmente reperibili e accessibili su internet realizzate da:

Prof.ssa Roberta Aiello – corso Basi di Dati dell'Università di Salerno

Prof. Dario Maio - corso Basi di Dati dell'Università di Bologna al seguente link ed altri: http://bias.csr.unibo.it/maio

Prof. Marco Di Felice - corso Basi di Dati dell'Università di Bologna al seguente link ed altri: http://www.cs.unibo.it/difelice/dbsi/

Prof Marco Maggini e prof Franco Scarselli - corso Basi di Dati dell'Università di Siena ai seguenti link ed altri: http://staff.icar.cnr.it/pontieri/didattica/LabSI/lezioni/\_preliminari-DB1%20(Maggini).pdf

Prof.ssa Raffaella Gentilini - corso Basi di Dati dell'Università di Perugia al seguente link ed altri: http://www.dmi.unipg.it/raffaella.gentilini/BD.htm

Prof. Enrico Giunchiglia - corso Basi di Dati dell'Università di Genova al seguente link ed altri: http://www.star.dist.unige.it/~enrico/BasiDiDati/

Prof. Maurizio Lenzerini - corso Basi di Dati dell'Università degli Studi La Sapienza di Roma al seguente link ed altri <a href="http://didatticainfo.altervista.org/Quinta/Database2.pdf">http://didatticainfo.altervista.org/Quinta/Database2.pdf</a>

- The PostgreSQL Global Development Group PostgreSQL nn.xx Documentation
- > PostgreSQL (appendice scaricabile dal sito del libro (area studenti) e www.postgresql.org