

# SQL - Structured Query Language

SELECT (I)

Prof. Alfredo Pulvirenti

Prof. Salvatore Alaimo

(Atzeni-Ceri Capitolo 4)

# Generalità

- SQL sviluppato alla IBM nel 1973 è lo standard per tutti i sistemi commerciali ed open source (Oracle, Informix, Postgres, MySql, Sybase, DB2 etc.).
- Esistono sistemi commerciali che utilizzano interfacce tipo QBE:ACCESS. Tuttavia hanno sistemi per la traduzione automatica in SQL.
- SQL: Structured Query Language (lo standard definisce che adesso deve essere considerato come un nome proprio)
- Esistono diversi standard SQL-86/89/, SQL-2 del 1992 e SQL 3 del 99/2003. SQL-3 introduce i trigger, le viste ricorsive, e il supporto per il paradigma ad oggetti. In queste slide faremo riferimento fondamentalmente a SQL-92.

# SQL, operazioni sui dati

- interrogazione:
  - SELECT

# SELECT: sintassi generale

```
SELECT  [DISTINCT]  Attributi  
FROM    Tabelle  
[WHERE  Condizione]
```

# SELECT

- La query:
  1. Considera il prodotto cartesiano tra le *tabelle* in *Tabelle*
  2. Fra queste seleziona solo le righe che soddisfano la *Condizione*
  3. Infine valuta le espressioni specificate nella target list *Attributi*
- La SELECT implementa gli operatori Ridenominazione Proiezione, Selezione e Join dell'algebra relazionale
  - Più altro che vedremo più avanti

```
SELECT [DISTINCT] Attributi  
FROM Tabelle  
[WHERE Condizione]
```

*Attributi* ::= \* | *Attributo* {, *Attributo*}  
*Tabelle* ::= *Tabella* {, *Tabella*}

- Dove *Tabella* sta per una determinata relazione ed *Attributo* è uno degli attributi delle tabelle citate nella clausola FROM

# Semantica del SELECT

```
SELECT DISTINCT A1, A2, ..., An  
FROM R1, R2, ..., Rm  
WHERE C
```

- Equivale a

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_C(R_1 \times R_2 \times \dots \times R_m))$$

# QUERY SU UNA TABELLA

## STUDENTI

Nome	Matricola	Indirizzo	Telefono
Mario Rossi	123456	Via Etnea 1	222222
Ugo Bianchi	234567	Via Roma 2	333333
Teo Verdi	345678	Via Enna 3	444444

Vorrei conoscere  
indirizzo e telefono di Teo Verdi

```
SELECT Indirizzo, Telefono
FROM Studenti
WHERE Nome='Teo Verdi'
```

Indirizzo	Telefono
Via Enna 3	444444



# Scrittura Comandi SQL

- I comandi SQL non sono **case sensitive**.
- Possono essere distribuiti in una o più righe.
- Le parole chiave non possono essere abbreviate o spezzate in due linee.
- Clausole sono usualmente inserite in linee separate.

# Espressioni Aritmetiche

- Creare espressioni attraverso l'uso di operatori aritmetici.

Operatore	Descrizione
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione

# Uso degli operatori Aritmetici

```
SQL> SELECT ename, sal, sal+300  
2 FROM emp;
```

ENAME	SAL	SAL+300
KING	5000	5300
BLAKE	2850	3150
CLARK	2450	2750
JONES	2975	3275
MARTIN	1250	1550
ALLEN	1600	1900
...		

14 rows selected.

# Precedenza Operatori

```
SQL> SELECT ename, sal, 12*sal+100  
2 FROM emp;
```

ENAME	SAL	12*SAL+100
-----	-----	-----
KING	5000	60100
BLAKE	2850	34300
CLARK	2450	29500
JONES	2975	35800
MARTIN	1250	15100
ALLEN	1600	19300
...		

14 rows selected.

# Uso delle parentesi

```
SQL> SELECT ename, sal, 12*(sal+100)  
2 FROM emp;
```

ENAME	SAL	12*(SAL+100)
KING	5000	61200
BLAKE	2850	35400
CLARK	2450	30600
JONES	2975	36900
MARTIN	1250	16200
...		

14 rows selected.

# Alias delle colonne

- Rinominare il nome di una colonna
- Utile con dei calcoli
- Deve seguire immediatamente il nome di una colonna; può essere usata opzionalmente la parola chiave AS tra il nome della colonna e l'alias
- Richiede doppio apice se l'alias ha degli spazi

# Uso dell'Alias

```
SQL> SELECT   ename AS name, sal salary
2 FROM      emp;
```

NAME	SALARY
-----	
...	

```
SQL> SELECT   ename "Name",
2             sal*12 "Annual Salary"
3 FROM      emp;
```

Name	Annual Salary
-----	
...	

## Alias nelle tabelle

```
SELECT p.Professore
FROM Corsi p, Esami e
WHERE p.Corso = e.Corso AND
      Matricola = '123456'
```

- Professori con cui 123456 ha fatto esami
- Se i nomi degli attributi non sono univoci tra le tabelle si deve usare il nome della tabella nella select!



# Eliminazione delle righe duplicate

- E' consentito dall'uso della parola chiave **DISTINCT** nella clausola **SELECT**

```
SQL> SELECT DISTINCT deptno  
2 FROM emp ;
```

DEPTNO

-----  
10  
20  
30

Una relazione nell'algebra relazionale e' un insieme di tuple.  
In SQL i duplicati sono mantenuti.

# Esempio

## IMPIEGATI

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

**"...selezionare  
tutti gli impiegati  
del dipartimento 10"**



## IMPIEGATI

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

# Limitare le righe selezionate

- Limitare le righe tramite l'uso della clausola WHERE.

```
SELECT      [DISTINCT] { * | colonna [alias], ... }  
FROM        tabella  
[WHERE      condizione(i)];
```

- La clausola WHERE segue la clausola FROM.

# Uso della clausola WHERE

```
SQL> SELECT ename, job, deptno  
2 FROM emp  
3 WHERE job='CLERK' ;
```

ENAME	JOB	DEPTNO
-----	-----	-----
JAMES	CLERK	30
SMITH	CLERK	20
ADAMS	CLERK	20
MILLER	CLERK	10

# Predicati di confronto

Operatore	Significato
=	Uguale a
>	più grande di
>=	maggiore o uguale di
<	minore di
<=	minore o uguale a
<>	diverso

# Uso dei predicati di confronto

```
SQL> SELECT ename, sal, comm  
2 FROM emp  
3 WHERE sal<=comm;
```

ENAME	SAL	COMM
MARTIN	1250	1400

# Altri Predicati di Confronto

Operatori	Significato
<b>BETWEEN ...AND...</b>	compreso tra due valori
<b>IN(list)</b>	Corrisp. ad uno dei valori nella lista
<b>LIKE</b>	Operatore di pattern matching
<b>IS NULL</b>	Valore nullo

# Uso dell'operatore BETWEEN

- BETWEEN consente la selezione di righe con attributi in un particolare range.

```
SQL> SELECT  ename, sal  
2  FROM      emp  
3  WHERE     sal BETWEEN 1000 AND 1500;
```

ENAME	SAL	Limite inferiore	Limite superiore
-----	-----		
MARTIN	1250		
TURNER	1500		
WARD	1250		
ADAMS	1100		
MILLER	1300		



# Predicato BETWEEN

*Espr1* [NOT] BETWEEN *Espr2* AND *Espr3*

- Equivale a

[NOT] (*Espr2* <= *Espr1* AND *Espr1* <= *Espr3*)

# Uso dell'operatore IN

- E' usato per selezionare righe che hanno un attributo che assume valori contenuti in una lista.

```
SQL> SELECT empno, ename, sal, mgr  
2 FROM emp  
3 WHERE mgr IN (7902, 7566, 7788);
```

EMPNO	ENAME	SAL	MGR
7902	FORD	3000	7566
7369	SMITH	800	7902
7788	SCOTT	3000	7566
7876	ADAMS	1100	7788

# Uso dell'operatore LIKE

- LIKE è usato per effettuare ricerche *wildcard* di una stringa di valori.
- Le condizioni di ricerca possono contenere sia letterali, caratteri o numeri.
  - % denota zero o più caratteri.
  - \_ denota un carattere.

```
SQL> SELECT   ename  
2  FROM      emp  
3  WHERE     ename LIKE 'S%';
```

# Uso dell'operatore LIKE

- Il pattern-matching di caratteri può essere combinato.

```
SQL> SELECT  ename  
      2 FROM    emp  
      3 WHERE   ename LIKE ' _A% ' ;
```

ENAME

-----

MARTIN

JAMES

WARD

- l'identificatore ESCAPE (\) deve essere usato per cercare "%" o "\_".

# Predicato IS [NOT] NULL

- L'operatore **IS NULL** controlla l'esistenza di valori null
- Sintassi:

```
Espr IS [NOT] NULL
```

- Esempio:

```
SELECT Nome  
FROM Studenti  
WHERE Telefono IS NOT NULL
```

# Uso dell'operatore IS NULL

```
SQL> SELECT  ename, mgr  
2  FROM      emp  
3  WHERE     mgr IS NULL;
```

ENAME	MGR
-----	-----
KING	

# Operatori Logici

Operatore	Significato
AND	Restituisce TRUE if <i>entrambe</i> le condizioni sono TRUE
OR	Restituisce TRUE se <i>almeno</i> una delle condizioni è TRUE
NOT	Restituisce TRUE se la condizione è FALSE

# Uso dell'operatore AND

```
SQL> SELECT empno, ename, job, sal  
2   FROM emp  
3   WHERE sal >= 1100  
4   AND job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7876	ADAMS	CLERK	1100
7934	MILLER	CLERK	1300



# Uso dell'operatore OR

```
SQL> SELECT empno, ename, job, sal  
2 FROM emp  
3 WHERE sal >= 1100  
4 OR job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7566	JONES	MANAGER	2975
7654	MARTIN	SALESMAN	1250
...			
7900	JAMES	CLERK	950
...			

14 rows selected.

# Uso dell'operatore NOT

```
SQL> SELECT ename, job  
2 FROM emp  
3 WHERE job NOT IN ('CLERK', 'MANAGER', 'ANALYST');
```

ENAME	JOB
-----	-----
KING	PRESIDENT
MARTIN	SALESMAN
ALLEN	SALESMAN
TURNER	SALESMAN
WARD	SALESMAN

# Regole di precedenza

Ordine di val.	Operatore
1	Tutti gli operatori di confronto
2 NOT	
3 AND	
4 OR	

- La modifica delle regole di precedenza è ottenuta con l'uso delle parentesi.

# Regole di precedenza

```
SQL> SELECT ename, job, sal
  2  FROM    emp
  3  WHERE   job='SALESMAN'
  4  OR      job='PRESIDENT'
  5  AND     sal>1500;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
MARTIN	SALESMAN	1250
ALLEN	SALESMAN	1600
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

# Regole di precedenza

## L'uso delle parentesi forza la priorità

```
SQL> SELECT      ename, job, sal
  2  FROM          emp
  3  WHERE         (job='SALESMAN'
  4  OR            job='PRESIDENT')
  5  AND          sal>1500;
```

ENAME	JOB	SAL
-----	-----	-----
KING	PRESIDENT	5000
ALLEN	SALESMAN	1600

# Clausola ORDER BY

- La clausola ORDER BY ordina le righe
  - ASC: ordine crescente, default
  - DESC: ordine decrescente
- La clausola ORDER BY è inserita per ultima nei comandi SELECT.

```
SQL> SELECT      ename, job, deptno, hiredate
  2  FROM          emp
  3  ORDER BY hiredate;
```

ENAME	JOB	DEPTNO	HIREDATE
-----	-----	-----	-----
SMITH	CLERK	20	17-DEC-80
ALLEN	SALESMAN	30	20-FEB-81
...			

14 rows selected.

# Ordinamento decrescente

```
SQL> SELECT      ename, job, deptno, hiredate  
  2  FROM        emp  
  3  ORDER BY hiredate DESC;
```

ENAME	JOB	DEPTNO	HIREDATE
-----	-----	-----	-----
ADAMS	CLERK	20	12-JAN-83
SCOTT	ANALYST	20	09-DEC-82
MILLER	CLERK	10	23-JAN-82
JAMES	CLERK	30	03-DEC-81
FORD	ANALYST	20	03-DEC-81
KING	PRESIDENT	10	17-NOV-81
MARTIN	SALESMAN	30	28-SEP-81

...

14 rows selected.

# Ordinamento tramite Alias

```
SQL> SELECT    empno, ename, sal*12 annsal  
2  FROM      emp  
3  ORDER BY  annsal;
```

EMPNO	ENAME	ANNSAL
7369	SMITH	9600
7900	JAMES	11400
7876	ADAMS	13200
7654	MARTIN	15000
7521	WARD	15000
7934	MILLER	15600
7844	TURNER	18000

...

14 rows selected.



# Ordinamento su Colonne Multiple

- L'ordine nella ORDER BY induce l'ordine dell'ordinamento.

```
SQL> SELECT      ename, deptno, sal  
2 FROM          emp  
3 ORDER BY deptno, sal DESC;
```

ENAME	DEPTNO	SAL
KING	10	5000
CLARK	10	2450
MILLER	10	1300
FORD	20	3000

...

14 rows selected.

- L'ordinamento può essere fatto anche con colonne che non sono nella target list

# Ottenere dati da più Tabelle

EMPNO	ENAME	...	DEPTNO
-----	-----	...	-----
7839	KING	...	10
7698	BLAKE	...	30
...			
7934	MILLER	...	10

IMPIEGATI

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

DIPARTIMENTI

EMPNO	DEPTNO	LOC
-----	-----	-----
7839	10	NEW YORK
7698	30	CHICAGO
7782	10	NEW YORK
7566	20	DALLAS
7654	30	CHICAGO
7499	30	CHICAGO
...		
14 rows selected.		

# Giunzione (JOIN)

- La join viene usata per effettuare query su più tabelle.

```
SELECT    tabella1.colonna, tabella2.colonna
FROM      tabella, tabella2
WHERE     tabella1.colonna1 = tabella2.colonna2;
```

- La condizione di join va scritta nella clausola WHERE.
- Mettere come prefisso della colonna il nome della tabella se questa stessa colonna appare in più di una tabella.

# Esempio

Nome	Matricola	Indirizzo	Telefono
Mario Rossi	123456	Via Etnea 1	222222
Ugo Bianchi	234567	Via Roma 2	333333
Teo Verdi	345678	Via Enna 3	444444

Quali esami ha superato Mario Rossi?

Corso	Matricola	Voto
Programmazione	345678	27
Architettura	123456	30
Programmazione	234567	18
Matematica Discreta	345678	22
Architettura	345678	30

```
SELECT Corso
FROM Esami,Studenti
WHERE Esami.Matricola = Studenti.Matricola
AND Nome='Mario Rossi'
```

Corso
Architettura

# Esempio

Nome	Matricola	Indirizzo	Telefono
Mario Rossi	123456	Via Etnea 1	222222
Ugo Bianchi	234567	Via Roma 2	333333
Teo Verdi	345678	Via Enna 3	444444

Corso	Professore
Programmazione	Ferro
Architettura	Pappalardo
Matematica Discreta	Lizzio

Corso	Matricola	Voto
Programmazione	345678	27
Architettura	123456	30
Programmazione	234567	18
Matematica Discreta	345678	22
Architettura	345678	30

Quali Professori hanno dato più di 24 a Teo Verdi ed in quali corsi?

```
SELECT Professore, Corsi.Corso
FROM Corsi, Esami, Studenti
WHERE Corsi.Corso = Esami.Corso AND Esami.Matricola =
  Studenti.Matricola AND Nome='Teo Verdi' AND Voto > 24
```

Corso	Professore
Programmazione	Ferro
Architettura	Pappalardo

# Uso degli alias nelle query su più tabelle

Professori che hanno fatto esami in almeno 2 corsi diversi allo stesso studente.

```
SELECT p1.Professore
FROM Corsi p1, Corsi p2, Esami e1, Esami e2
WHERE p1.Corso = e1.Corso AND
      p2.Corso = e2.Corso AND
      p1.Professore = p2.Professore AND
      e1.Matricola = e2.Matricola AND
      NOT p1.Corso = p2.Corso
```

## Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

## Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

## Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

# Join naturale

- Padre e madre di ogni persona

```
SELECT paternita.figlio, padre, madre  
FROM maternita, paternita  
WHERE paternita.figlio =  
maternita.figlio
```



# Prodotto Cartesiano

- Il prodotto cartesiano è ottenuto quando:
  - Una condizione join è omessa
  - Tutte le righe della prima tabella ammettono join con tutte le righe della seconda
  - Per evitare il prodotto cartesiano, includere sempre condizioni join valida nella clausola WHERE.

# Generare un Prodotto Cartesiano

**IMPIEGATI (14 righe)**

EMPNO	ENAME	...	DEPTNO
-----	-----	...	-----
7839	KING	...	10
7698	BLAKE	...	30
...			
7934	MILLER	...	10

**DIPARTIMENTI (4 righe)**

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON



**“Prodotto  
Cartesiano:  
14\*4=56 rows”**



ENAME	DNAME
-----	-----
KING	ACCOUNTING
BLAKE	ACCOUNTING
...	
KING	RESEARCH
BLAKE	RESEARCH
...	
56 rows selected.	

# Natural join

## IMPIEGATI

EMPNO	ENAME	DEPTNO
-----	-----	-----
7839	KING	10
7698	BLAKE	30
7782	CLARK	10
7566	JONES	20
7654	MARTIN	30
7499	ALLEN	30
7844	TURNER	30
7900	JAMES	30
7521	WARD	30
7902	FORD	20
7369	SMITH	20
...		
14 rows selected.		

## DIPARTIMENTI

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
20	RESEARCH	DALLAS
20	RESEARCH	DALLAS
...		
14 rows selected.		

# Estrarre Record con Natural join

```
SQL> SELECT  emp.empno,   emp.ename, emp.deptno,
2           dept.deptno, dept.loc
3   FROM      emp, dept
4   WHERE     emp.deptno=dept.deptno;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS

...

14 rows selected.

# Join esplicito (JOIN-ON)

- Sintassi:

```
SELECT ...  
FROM Tabella { ... JOIN Tabella ON CondDiJoin }, ...  
[ WHERE AltraCondizione ]
```

- Esempio: padre e madre di ogni persona (le due versioni):

```
SELECT paternita.figlio, padre, madre  
FROM maternita, paternita  
WHERE paternita.figlio = maternita.figlio
```

```
SELECT madre, paternita.figlio, padre  
FROM maternita JOIN paternita ON  
paternita.figlio = maternita.figlio
```

# Selezione, proiezione e join

- I padri di persone che guadagnano più di 20

$\pi_{padre}(paternità \bowtie_{figlio=nome} (\sigma_{reddito>20}(persone)))$

```
SELECT distinct padre  
FROM persone, paternita  
WHERE figlio = nome AND reddito > 20
```

```
SELECT distinct padre  
FROM persone  
JOIN paternita ON figlio = nome  
WHERE reddito > 20
```

# Necessità di ridenominazione

- Le persone che guadagnano più dei rispettivi padri;  
mostrare nome, reddito e reddito del padre

$$\pi_{nome,reddito,RP}(\sigma_{reddito > RP}(\delta_{NP,EP,RP \leftarrow Nome, Et\grave{a}, Reddito}(persone) \bowtie_{NP=padre} (paternit\grave{a} \bowtie_{figlio=nome} persone))))$$

```
SELECT f.nome, f.reddito, p.reddito
FROM persone p, paternita, persone f
WHERE p.nome = padre AND
      figlio = f.nome AND
      f.reddito > p.reddito
```

# SELECT, con ridenominazione del risultato

```
SELECT figlio, f.reddito AS reddito,  
       p.reddito AS redditoPadre  
FROM persone p, paternita, persone f  
WHERE p.nome = padre AND  
       figlio = f.nome AND  
       f.reddito > p.reddito
```

Oppure

```
SELECT x.figlio, f.reddito AS reddito,  
       p.reddito AS redditoPadre  
FROM persone p, paternita x, persone f  
WHERE p.nome = x.padre AND x.figlio = f.nome  
       AND f.reddito > p.reddito
```



Trovare le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

```
SELECT f.nome, f.reddito, p.reddito
FROM persone p, paternita, persone f
WHERE p.nome = padre AND
      figlio = f.nome AND
      f.reddito > p.reddito
```

```
SELECT f.nome, f.reddito, p.reddito
FROM persone p JOIN paternita ON p.nome = padre
      JOIN persone f ON figlio = f.nome
WHERE f.reddito > p.reddito
```

# Join esterno

- Padre e, se nota, madre di ogni persona

```
SELECT paternita.figlio, padre, madre  
FROM paternita LEFT JOIN maternita  
    ON paternita.figlio = maternita.figlio
```

## join esterno

```
SELECT paternita.figlio, padre, madre  
FROM maternita JOIN paternita  
ON maternita.figlio = paternita.figlio
```

```
SELECT paternita.figlio, padre, madre  
FROM maternita LEFT JOIN paternita  
ON maternita.figlio = paternita.figlio
```

```
SELECT paternita.figlio, padre, madre  
FROM maternita FULL JOIN paternita  
ON maternita.figlio = paternita.figlio
```

## Ancora su Join

- CROSS JOIN è il prodotto cartesiano
- UNION JOIN...ON è l'unione esterna cioè si estendono le due tabelle con le colonne dell'altro con valori nulli e si fa l'unione.

# Ancora sulle Join

- **NATURAL JOIN**
- **JOIN...USING (...)** è la natural join sugli attributi specificati (un sottoinsieme di quelli in comune) presenti in entrambe le tabelle
- **JOIN...ON** su quelli che soddisfano una data condizione
- **[LEFT|RIGHT|FULL]** usato con Natural Join o Join è la giunzione esterna nelle tre modalità sinistra ,destra o completa.

- Natural Join

```
SELECT  
Studenti.Nome, Esami.Corso, Esami.Voto  
FROM Esami NATURAL JOIN Studenti
```

# Esempio

Agenti (CodiceAgente, Nome, Zona, Supervisore, Commissione)

Clienti (CodiceCliente, Nome, Città', Sconto)

Ordini (CodiceOrdine, CodiceCliente, CodiceAgente, Articolo, Data, Ammontare)

# Esempio

Agenti (CodiceAgente, Nome, Zona, Supervisore, Commissione)

Clienti (CodiceCliente, Nome, Città', Sconto)

Ordini (CodiceOrdine, CodiceCliente, CodiceAgente, Articolo, Data, Ammontare)



# Esempio di Join On

Codice agente ed ammontare degli ordini dei supervisori

```
SELECT Agenti.CodiceAgente,Ordini.Ammontare  
FROM Agenti JOIN Ordini  
ON Agenti.Supervisore = Ordini.CodiceAgente
```

# Giunzione Esterna

Codice agente ed ammontare degli agenti incluso quelli che non hanno effettuato ordini (avranno ammontare NULL)

```
SELECT Agenti.CodiceAgente,Ordini.Ammontare  
FROM Agenti NATURAL LEFT JOIN Ordini
```

# Unione, intersezione e differenza

- La **SELECT** da sola non permette di fare unioni; serve un costrutto esplicito:

```
SELECT ...  
UNION  
SELECT ...
```

- i duplicati vengono eliminati, per mantenerli bisogna specificarlo **UNION ALL**.

# Notazione posizionale!

```
SELECT padre  
FROM paternita  
UNION  
SELECT madre  
FROM maternita
```

- quali nomi per gli attributi del risultato?
  - nessuno
  - quelli del primo operando
  - ...

## Notazione posizionale, 2

SELECT padre, figlio

FROM paternita

UNION

SELECT figlio, madre

FROM maternita

SELECT padre, figlio

FROM paternita

UNION

SELECT madre, figlio

FROM maternita

# Notazione posizionale, 3

- Anche con le ridenominazioni non cambia niente:

```
SELECT padre as genitore, figlio  
FROM paternita  
UNION  
SELECT figlio, madre as genitore  
FROM maternita
```

- Corretta:

```
SELECT padre as genitore, figlio  
FROM paternita  
UNION  
SELECT madre as genitore, figlio  
FROM maternita
```

- Analogamente
  - INTERSECT [ALL]
  - EXCEPT [ALL]

Aggregazione dati



# Operatori aggregati

- Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple
- SQL-2 prevede 5 possibili operatori di aggregamento:
  - conteggio, minimo, massimo, media, somma
- Gli operatori di aggregazione NON sono rappresentabili in Algebra Relazionale

# Cosa sono?

- Operano su insiemi di righe per dare un risultato per gruppo.

## IMPIEGATI

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

“Salario  
Massimo”

MAX (SAL)
5000

# Quali sono

- AVG
- COUNT
- MAX
- MIN
- SUM

```
SELECT      [column,] group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

# Uso di AVG e SUM

- Possono essere usati su dati numerici.

```
SQL> SELECT  AVG(sal), MAX(sal),  
2           MIN(sal), SUM(sal)  
3 FROM      emp  
4 WHERE     job LIKE 'SALES%';
```

AVG (SAL)	MAX (SAL)	MIN (SAL)	SUM (SAL)
-----	-----	-----	-----
1400	1600	1250	5600

# Uso di MIN e MAX

- Possono essere usati su qualsiasi tipo.

```
SQL> SELECT MIN(hiredate), MAX(hiredate)
2 FROM emp;
```

MIN (HIRED	MAX (HIRED
-----	-----
17-DEC-80	12-JAN-83

# Uso di COUNT

- COUNT(\*) ritorna il numero di righe di una tabella.

```
SQL> SELECT COUNT (*)  
2 FROM emp  
3 WHERE deptno = 30;
```

COUNT (\*)

-----

6

# Operatori aggregati: COUNT

- `count(*)` come detto restituisce il numero di righe
- `Count(attributo)` il numero di valori di un particolare attributo non null
- Esempio: Il numero di figli di Franco:

```
SELECT count(*) as NumFigliDiFranco  
FROM Paternita  
WHERE Padre = 'Franco'
```

- l'operatore aggregato (`count`) viene applicato al risultato dell'interrogazione:

```
SELECT *  
FROM Paternita  
WHERE Padre = 'Franco'
```



## **Paternità**

<b>Padre</b>	<b>Figlio</b>
<b>Sergio</b>	<b>Franco</b>
<b>Luigi</b>	<b>Olga</b>
<b>Luigi</b>	<b>Filippo</b>
<b>Franco</b>	<b>Andrea</b>
<b>Franco</b>	<b>Aldo</b>

**NumFigliDiFranco**

**2**

# COUNT e valori nulli

- Numero di tuple

```
SELECT count(*) FROM persone
```

- Numero di volte il campo 'reddito' non è NULL

```
SELECT count(reddito) FROM persone
```

- Numero di valori distinti del campo 'reddito' (senza i NULL)

```
SELECT count(distinct reddito) FROM persone
```

## **Persone**

<b>Nome</b>	<b>Età</b>	<b>Reddito</b>
<b>Andrea</b>	<b>27</b>	<b>21</b>
<b>Aldo</b>	<b>25</b>	<b>NULL</b>
<b>Maria</b>	<b>55</b>	<b>21</b>
<b>Anna</b>	<b>50</b>	<b>35</b>

## Osservazioni

- Se una colonna A contiene solo valori nulli, MAX, MIN, AVG, SUM restituiscono NULL, mentre Count vale zero.
- AVG, SUM ignorano i valori nulli

# Altri operatori aggregati

- SUM, AVG, MAX, MIN
- Media dei redditi di coloro che hanno meno di 30 anni:

```
SELECT avg(reddito)  
FROM persone  
WHERE eta < 30
```

- Uso del JOIN: media dei redditi dei figli di Franco:

```
SELECT avg(reddito)  
FROM persone JOIN paternita ON nome=figlio  
WHERE padre='Franco'
```

- Uso di più operatori di aggregamento nella target list:

```
SELECT avg(reddito), min(reddito), max(reddito)  
FROM persone  
WHERE eta < 30
```

# Operatori aggregati e valori nulli

```
SELECT avg(reddito) AS redditomedio  
FROM persone
```

**Persone**

<b>Nome</b>	<b>Età</b>	<b>Reddito</b>
<b>Andrea</b>	<b>27</b>	<b>30</b>
<b>Aldo</b>	<b>25</b>	<b>NULL</b>
<b>Maria</b>	<b>55</b>	<b>36</b>
<b>Anna</b>	<b>50</b>	<b>36</b>

# Creare gruppi di dati

## IMPIEGATI

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

2916.6667

2175

1566.6667

“salario  
medio  
in IMPIEGATI  
per ogni  
dipartimento”

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667

# Creare gruppi tramite: GROUP BY

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Divide le righe di una tabella in gruppi più piccoli.

# Uso di GROUP BY

- Tutte le colonne della SELECT che non sono in funzioni di gruppo devono essere nella GROUP BY.

```
SQL> SELECT deptno, AVG(sal)
2 FROM emp
3 GROUP BY deptno;
```

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667



# Uso GROUP BY

- La colonna di GROUP BY non deve essere necessariamente nella SELECT.

```
SQL> SELECT      AVG(sal)
  2  FROM        emp
  3  GROUP BY deptno;
```

```
AVG (SAL)
-----
2916.6667
2175
1566.6667
```

# Raggruppare piu' di una colonna

## IMPIEGATI

DEPTNO	JOB	SAL
10	MANAGER	2450
10	PRESIDENT	5000
10	CLERK	1300
20	CLERK	800
20	CLERK	1100
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	SALESMAN	1600
30	MANAGER	2850
30	SALESMAN	1250
30	CLERK	950
30	SALESMAN	1500
30	SALESMAN	1250

“sommare i salari  
in IMPIEGATI  
per ogni lavoro,  
Raggruppati  
per dipartimento”

DEPTNO	JOB	SUM (SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

# Uso di GROUP BY su colonne multiple

```
SQL> SELECT    deptno, job, sum(sal)
  2  FROM      emp
  3  GROUP BY  deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
...		

9 rows selected.

# Operatori aggregati e target list

- un'interrogazione scorretta:

```
SELECT nome, max(reddito)  
FROM persone
```

- di chi sarebbe il nome?
- La target list deve essere omogenea

```
SELECT min(eta), avg(reddito)  
FROM persone
```

# Operatori aggregati e raggruppamenti

- Le funzioni possono essere applicate a partizioni delle relazioni
- Clausola **GROUP BY**
  - Sintassi: **GROUP BY** listaAttributi
- Il numero di figli di ciascun padre

```
SELECT padre, count(*) AS NumFigli  
FROM paternita  
GROUP BY Padre
```

**paternita**

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Padre	NumFigli
Sergio	1
Luigi	2
Franco	2

# Query errate con funzioni di raggruppamento

- Ogni colonna o espressione della SELECT che non è argomento di funzioni di gruppo deve essere nella GROUP BY.

```
SQL> SELECT deptno, COUNT(ename)
2 FROM emp;
```

```
SELECT deptno, COUNT(ename)
      *
ERROR at line 1:
-----: not a single-group group function
```

# Query illegal con funzioni di raggrup.

- Non può essere usata la WHERE per restringere i gruppi.

```
SQL> SELECT      deptno, AVG(sal)
  2  FROM          emp
  3  WHERE         AVG(sal) > 2000
  4  GROUP BY     deptno;
```

```
WHERE AVG(sal) > 2000
```

```
*
```

```
ERROR at line 3:
```

```
-----: group function is not allowed here
```

- Dobbiamo estendere la sintassi:
  - Deve essere usata una clausola nuova
    - HAVING.

# Escludere gruppi di ris.

## IMPIEGATI

DEPTNO	SAL
-----	
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

5000

3000

2850

**“salario  
massimo  
per dipartimento  
maggiore di  
\$2900”**

DEPTNO	MAX (SAL)
-----	
10	5000
20	3000



# Clausola HAVING

- Uso di HAVING per restringere gruppi
  - Le righe sono raggruppate.
  - La funzione di raggruppamento è applicata.

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column] ;
```

# Uso di HAVING

```
SQL> SELECT    deptno, max(sal)
  2  FROM      emp
  3  GROUP BY  deptno
  4  HAVING    max(sal)>2900;
```

DEPTNO	MAX (SAL)
10	5000
20	3000

# Uso di HAVING

```
SQL> SELECT      job, SUM(sal) PAYROLL
  2  FROM          emp
  3  WHERE         job NOT LIKE 'SALES%'
  4  GROUP BY     job
  5  HAVING        SUM(sal)>5000
  6  ORDER BY     SUM(sal);
```

JOB	PAYROLL
ANALYST	6000
MANAGER	8275

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column] ;
```

- Ordine di valutazione delle clausole:
  - WHERE
  - GROUP BY
  - HAVING