

BASI DI DATI I

- Query complesse in SQL
- Aggiornamenti in SQL

QUERY COMPLESSE IN SQL



QUERY ANNIDATE E CONFRONTO DI INSIEMI

- La query usata per mostrare la **UNION** può essere così riformulata:

```
SELECT DISTINCT PNUMBER
FROM PROJECT
WHERE PNUMBER IN
  (SELECT PNUMBER
   FROM PROJECT, DEPARTMENT, EMPLOYEE
   WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME= 'Smith')
```

OR

```
PNUMBER IN
  (SELECT PNO
   FROM WORKS_ON, EMPLOYEE
   WHERE ESSN=SSN AND LNAME='Smith');
```



QUERIES ANNIDATE E CONFRONTO DI INSIEMI (2)

- La prima query seleziona il numero dei progetti che hanno uno '*Smith*' come manager, mentre la seconda seleziona il numero di progetto dei progetti che hanno uno '*Smith*' come lavoratore.
- Nella query esterna selezioniamo una tupla PROJECT se il valore PNUMBER compare nel risultato di una delle query annidate.



L'OPERATORE IN

- L'operatore **IN** confronta un valore con un insieme di tuple union-compatibili.
- L'operatore **IN** permette di specificare valori multipli nella clausola WHERE.

- **Esempio:**

```
SELECT DISTINCT ESSN
  FROM WORKS_ON
 WHERE (PNO, HOURS) IN
       (SELECT PNO, HOURS
        FROM WORKS_ON
       WHERE ESSN='123456789');
```



ALTRI OPERATORI

▪ **ANY (o SOME)**

- confronta un singolo valore (attributo) **v** con un multiset **V**, restituendo TRUE se **v** è uguale a qualche valore in **V**.
- **ANY e SOME** sono equivalenti. Possono essere combinati con { $>$, \geq , $<$, \leq , $<>$ }.
- $= ANY$ è equivalente ad usare l'operatore **IN**.

▪ Anche **ALL** può essere combinato con questi operatori.

- $v > ALL V$ è TRUE se il valore **v** è maggiore di tutti i valori in **V**.



OPERATORE ALL - ESEMPIO

- Trovare tutti i nomi degli impiegati il cui salario è maggiore del salario di tutti gli impiegati del dipartimento 5:

```
SELECT LNAME, ENAME  
      FROM EMPLOYEE  
 WHERE SALARY > ALL (SELECT SALARY  
                         FROM EMPLOYEE  
                        WHERE DNO=5);
```



OPERATORE ANY - ESEMPIO

```
SELECT LNAME, ENAME  
FROM EMPLOYEE  
WHERE SALARY > ANY (SELECT SALARY  
                 FROM EMPLOYEE);
```

- Trovare tutti i nomi degli impiegati tranne quelli il con il salario minimo.



CASI DI AMBIGUITÀ

- **Ambiguità nei nomi di attributi:**
 - Si ha, se esistono attributi con lo stesso nome, uno in una relazione nella clausola FROM della query esterna e l'altro in una relazione della clausola FROM della query interna.
- **Regola:** un riferimento a un attributo non qualificato riferisce alla relazione dichiarata nella query annidata più interna.



CASI DI AMBIGUITÀ - ESEMPIO

- Trovare il nome di ogni impiegato che ha una persona a carico con lo stesso nome e lo stesso sesso dell'impiegato:

```
SELECT E.ENAME, E.LNAME  
FROM EMPLOYEE AS E  
WHERE E.SSN IN (SELECT ESSN  
         FROM DEPENDENT  
         WHERE ESSN=E.SSN  
         AND DEPENDENT_NAME = E.FNAME  
         AND SEX=E.SEX);
```



È necessario qualificarlo altrimenti farebbe riferimento alla relazione DEPENDENT.



CASI DI AMBIGUITÀ (2)

- In generale, una query scritta con blocchi annidati **SELECT...FROM...WHERE** e con operatori di confronto **=** o **IN** può essere sempre espressa come un singolo blocco.
- La precedente query può anche essere scritta come:

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E, DEPENDENT AS D
WHERE E.SSN=D.SSN AND E.SEX=D.SEX AND
E.FNAME=D.DEPENDENT_NAME
```



L'OPERATORE **CONTAINS**

- L'implementazione originale SQL su **systemR** prevedeva un operatore **CONTAINS** per confrontare due insiemi.
- È stato poi eliminato per motivi di efficienza.



L'OPERATORE CONTAINS - ESEMPIO

- Ritrovare il nome e cognome di ciascun impiegato che lavora su tutti i progetti controllati dal dipartimento 5:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE AS E  
WHERE (( SELECT PNO  
          FROM WORKS_ON  
          WHERE E.SSN=ESSN)  
        CONTAINS  
        ( SELECT PNUMBER  
          FROM PROJECT  
          WHERE DNUM=5));
```



EXIST E NOT EXIST

- **EXISTS e NOT EXISTS:**

- Per verificare se il risultato di una query annidata correlata è vuota.

- **Esempio:** Ritrovare il nome degli impiegati che non hanno persone a carico:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE NOT EXISTS ( SELECT *  
    FROM DEPENDENT  
    WHERE SSN=ESSN);
```



INSIEMI ESPLICITI

- Trovare il SSN di tutti gli impiegati che lavorano sui progetti 1, 2 o 3:

```
SELECT DISTINCT ESSN  
FROM WORKS_ON  
WHERE PNO IN (1, 2, 3);
```

- Si può anche testare se un valore è **NULL**:

- = e ≠ sono scritti come 'IS' e 'IS NOT' per confronti con NULL.
- **Esempio:** Trovare il nome di tutti gli impiegati che non hanno supervisori:

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE SUPERSSN IS NULL;
```



LA KEYWORD AS

- È possibile rinominare qualsiasi attributo che compare in una query con la keyword **AS**:

```
SELECT E.LNAME AS EMPLOYEE_NAME,
      S.LNAME AS SUPERVISOR_NAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;
```



TABELLE JOINED

- Il concetto di tabella joined compare con SQL2 per specificare un'operazione di **JOIN** nella clausola FROM.
- Possono essere usati i seguenti tipi di join:
 - **NATURAL JOIN**
 - **INNER JOIN**
 - **LEFT OUTER JOIN / RIGHT OUTER JOIN**
 - **FULL OUTER JOIN**



TABELLE JOINED - *ESEMPIO*

- Trovare il nome e l'indirizzo di ogni impiegato che lavora per il Dipartimento '*Research*':

```
SELECT FNAME, LNAME, ADDRESS  
FROM (EMPLOYEE JOIN DEPARTMENT ON  
DNO=DNUMBER)  
WHERE DNAME='Research';
```



AGGREGAZIONE E RAGGRUPPAMENTO

- Le funzioni di aggregazione e di raggruppamento sono diffusissime nella gestione di basi di dati. SQL incorpora le seguenti funzioni:
 - **COUNT**: conteggio tuple.
 - **COUNT(DISTINCT ...)**: conteggio di tuple distinte.
 - **SUM**: somma dei valori di un attributo in una tabella.
 - **MAX**: valore massimo tra gli attributi di una tabella.
 - **MIN**: valore minimo tra gli attributi di una tabella.
 - **AVG**: valore medio tra gli attributi di una tabella.
 - **STD**: deviazione standard tra gli attributi di una tabella.



AGGREGAZIONE E RAGGRUPPAMENTO

- **Esempio:** Trovare la somma dei salari di tutti gli impiegati, il massimo, il minimo e la media dei salari:

```
SELECT SUM(SALARY), MAX(SALARY),  
       MIN(SALARY), AVG(SALARY)  
  FROM EMPLOYEE;
```



COUNT - ESEMPI

- Conta il numero di impiegati:

```
SELECT COUNT(*)  
FROM EMPLOYEE;
```

- Restituisce il numero di tuple nel risultato della query (*):

```
SELECT COUNT(*) AS Conteggio  
FROM EMPLOYEE, DEPARTMENT  
WHERE DNO=DNUMBER AND DNAME='Research';
```



COUNT - ESEMPI (2)

- Conta il numero di valori di stipendi distinti:

```
SELECT COUNT (DISTINCT SALARY)  
FROM EMPLOYEE;
```

- Elencare il nome ed il cognome degli impiegati che hanno due o più persone a carico:

```
SELECT LNAME, FNAME  
FROM EMPLOYEE  
WHERE ( SELECT COUNT(*)  
        FROM DEPENDENT  
        WHERE SSN=ESSN)>=2;
```



ORDINAMENTO DI TUPLE

- Per ordinare le tuple nel risultato della query si usa la clausola **ORDER BY**.
- **Esempio:** Ritrovare una lista di impiegati e dei progetti su cui lavorano, ordinati per dipartimento, e nell'ambito di ciascun dipartimento, alfabeticamente per cognome e nome:

```
SELECT DNAME, FNAME, LNAME, PNAME  
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT  
WHERE DNUMBER=DNO AND SSN=ESSN AND  
PNO=PNUMBER  
ORDER BY DNAME, LNAME, FNAME;
```



ORDINAMENTO DI TUPLE (2)

- L'ordine di default è crescente:
 - **ASC** per crescente.
 - **DESC** decrescente.
- **Esempio:** per avere un ordine decrescente di dipartimento e crescente per nome e cognome:

...

ORDER BY DNAME DESC, LNAME ASC, FNAME ASC;



GROUP BY

- Raggruppiamo le tuple che hanno lo stesso valore per alcuni attributi.
- **Esempio:**

```
SELECT DNO, COUNT(*), AVG(SALARY)
  FROM EMPLOYEE
 GROUP BY DNO;
```
- Le tuple sono divise in gruppi, ogni gruppo ha lo stesso valore per DNO.
- Le funzioni COUNT e AVG sono applicate ad ogni gruppo di queste tuple.



GROUP BY (2)

- Risultato:

DNO	COUNT(*)	AVG(SALARY)
1	4	23000
4	3	25000
3	4	22000



GROUP BY (3)

- **Esempio:** Per ogni progetto, visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on  
WHERE pnumber = pno  
GROUP BY pnumber;
```



GROUP BY (4)

- **Esempio:** Per ogni progetto visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati del dipartimento n.5 che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on, employee  
WHERE pnumber = pno AND ssn = essn AND dno = 5  
GROUP BY pnumber, pname;
```



GROUP BY (5) – USO DI HAVING

- **Esempio:** Per ogni progetto su cui lavorano più di due impiegati, visualizzare il numero del progetto, il nome del progetto ed il numero di impiegati che lavorano su quel progetto:

```
SELECT pnumber, pname, COUNT(*)  
FROM project, works_on  
WHERE pnumber = pno  
GROUP BY pnumber, pname  
HAVING COUNT(*) > 2;
```



GROUP BY (6) – USO DI HAVING (2)

- **Esempio:** Determinare, per ogni dipartimento che ha più di 6 impiegati, il numero totale degli impiegati il cui stipendio è maggiore di \$40.000:

```
SELECT dname, COUNT(*)  
FROM department, employee  
WHERE dnumber = dno AND salary > 40000  
GROUP BY dname  
HAVING COUNT(*) > 6;
```



RIEPILOGO DELLE INTERROGAZIONI

SELECT <elenco attributi e funzioni>
FROM <elenco delle tavelle>
[**WHERE** <condizioni>]
[**GROUP BY** <attributo o attributi di raggruppamento>]
[**HAVING** <condizione di raggruppamento>]
[**ORDER BY** <elenco attributi>]



AGGIORNAMENTI IN SQL



AGGIORNAMENTI IN SQL

- In SQL sono previsti tre comandi per modificare il database:
 - **INSERT**
 - **DELETE**
 - **UPDATE**



IL COMANDO INSERT

- Il comando **INSERT INTO** inserisce nuove righe in una relazione.

- Sintassi:

```
INSERT INTO Target [(FieldName,...)]  
VALUES (Value1,...);
```

oppure

```
INSERT INTO Target [(FieldName,...)]  
SELECT FieldName,...  
FROM TableExpression;
```



IL COMANDO INSERT - ESEMPIO

- Aggiungere una nuova tupla alla relazione '*Employee*':

```
INSERT INTO EMPLOYEE  
VALUES ('Richard', 'K', 'Marini', '654765876',  
'30-DEC-52', '98 Oak Forest, Katy, TX', 'M', 37000, '987654321', 4);
```



IL COMANDO INSERT (2)

- È possibile non assegnare valori a tutti gli attributi.
 - In tal caso, questi avranno il valore di **default** o **NULL**.
-
- **Esempio:**
 - **INSERT INTO** EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '654765876');



IL COMANDO INSERT - ESEMPIO

- Creare una tabella temporanea che ha nome, numero di impiegati e salari totali per ciascun dipartimento:

```
CREATE TABLE DEPTS_INFO ( DEPT_NAME VARCHAR(15),  
                           NO_OF_EMPS INTEGER,  
                           TOTAL_SAL INTEGER);
```

```
INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)  
SELECT DNAME, COUNT(*), SUM(SALARY)  
FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER=DNO  
GROUP BY DNAME;
```

- *Eventuali aggiornamenti successivi non influenzano la tabella originale. Per aggiornarle, è invece necessario definire una view.*



USO DI *AUTO_INCREMENT*

- L' attributo **AUTO_INCREMENT** può essere usato per generare un identificatore unico per le nuove righe:

```
CREATE TABLE animals (
    id INT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
) AUTO_INCREMENT=5;
```

```
INSERT INTO animals (name) VALUES
('dog'),('cat'),('penguin'),('wolf'),('whale'),('ostrich');
```

```
SELECT * FROM animals;
```

id	name
5	dog
6	cat
7	penguin
8	wolf
9	whale
10	ostrick



IL COMANDO DELETE

- Il comando **DELETE** rimuove una o più tuple da una relazione.
- Sintassi:

```
DELETE
FROM TableName
WHERE Criteria;
```



IL COMANDO DELETE - *ESEMPI*

```
DELETE FROM EMPLOYEE  
WHERE LNAME='Brown';
```

```
DELETE FROM EMPLOYEE  
WHERE DNO IN (SELECT DNUMBER  
         FROM DEPARTMENT  
         WHERE DNAME='Research');
```



IL COMANDO UPDATE

- Il comando **UPDATE** permette di modificare valori in una relazione:

```
UPDATE PROJECT  
SET PLOCATION='Bellaire', DNUM=5  
WHERE PNUMBER=10;
```

```
UPDATE EMPLOYEE  
SET SALARY=SALARY * 1.1  
WHERE DNO IN (SELECT DNUMBER  
FROM DEPARTMENT  
WHERE DNAME='Research');
```



VISTE IN SQL

- Le viste sono tabelle ‘virtuali’ derivate da tabelle esistenti nel db.
- Possono essere definite per nascondere dei dati da alcune tabelle (es. per questioni di privacy), per combinare più tabelle, per creare report, etc.
- Sintassi:
CREATE VIEW *ViewName*
AS *SelectStatement*;



VISTE IN SQL - ESEMPIO

```
CREATE VIEW WORKS_ON1
AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER;
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS
-------	-------	-------	-------



VISTE IN SQL - ESEMPIO (2)

```
CREATE VIEW DEPTS_INFO (DEPT_NAME,  
    NO_OF_EMPS, TOTAL_SAL)  
    SELECT DNAME, COUNT(*), SUM(SALARY)  
    FROM DEPARTMENT, EMPLOYEE  
    WHERE DNUMBER=DNO  
    GROUP BY DNAME;
```

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL
-----------	------------	-----------





FINE

Per eventuali domande: (in ordine di preferenza personale)

- Ora.
- Chat di Teams
- Mail: silvio.barra@unina.it

