5.0- Le operazioni

Basi di Dati
per la gestione dell'Informazione

A. Chianese, V. Moscato, A. Picariello,
L. Sansone

Introduzione

- Le operazioni del modello relazionale sono di due tipi:
  - Operazioni di modifica delle base di dati
    - *updates*
  - Interrogazioni per il recupero delle informazioni
    - *query* e *retrieval*
- Outline:
  - Estensione delle operazioni insiemistiche classiche alle relazioni
  - Operazioni di modifica
  - Operazioni relazionali
  - Query SQL

OPERAZIONI INSIEMISTICHE Notazione Classica

- Siano *r1* e *r2* due relazioni **definite sullo stesso insieme di attributi** *X={A1,A2,…,An}*, è possibile definire le seguenti operazioni:
  - **Unione di *r1* e *r2***: una nuova relazione *r* ottenuta considerando sia le tuple di *r1* che quelle di *r2*

$$r = r_1 \cup r_2 \equiv \{ t: t \in r_1 \vee t \in r_2 \}$$

  - **Intersezione di *r1* e *r2*** : una nuova relazione *r* ottenuta considerando le tuple di *r1* che sono anche tuple di *r2* e viceversa

$$r = r_1 \cap r_2 \equiv \{ t: t \in r_1 \wedge t \in r_2 \}$$

  - **Differenza di *r1* e *r2***: una nuova relazione *r* ottenuta considerando le tuple di *r1* escluse quelle che appartengono anche ad *r2*

$$r = r_1 - r_2 \equiv \{ t: t \in r_1 \wedge t \notin r_2 \}$$

Esempi di operazioni insiemistiche: Unione

Matricola	Nome	Cognome	Indirizzo
150	Alex	Del Piero	via dei Palloni, 30
151	Martina	Stellina	via del Cielo, 40
142	Giovanni	SenzaTerra	via delle Crociate, 30

r1 - Studenti di Basi di Dati

Matricola	Nome	Cognome	Indirizzo
142	Giovanni	SenzaTerra	via delle Crociate, 30
149	Marco	Rossi	via dei colori, 40

r2 - Studenti di Analisi I

r1 ur2 - Studenti che frequentano o il corso di Basi di Dati o il corso di Analisi I

Matricola	Nome	Cognome	Indirizzo
150	Alex	Del Piero	via dei Palloni, 30
151	Martina	Stellina	via del Cielo, 40
142	Giovanni	SenzaTerra	via delle Crociate, 30
149	Marco	Rossi	via dei colori, 40

Esempi di operazioni insiemistiche: Intersezione

Matricola	Nome	Cognome	Indirizzo
150	Alex	Del Piero	via dei Palloni, 30
151	Martina	Stellina	via del Cielo, 40
142	Giovanni	SenzaTerra	via delle Crociate, 30

r1 - Studenti di Basi di Dati

Matricola	Nome	Cognome	Indirizzo
142	Giovanni	SenzaTerra	via delle Crociate, 30
149	Marco	Rossi	via dei colori, 40

r2 - Studenti di Analisi I

r1 ∩ r2 - Studenti che frequentano sia il corso di Basi di Dati sia il corso di Analisi I

Matricola	Nome	Cognome	Indirizzo
142	Giovanni	SenzaTerra	via delle Crociate, 30

Esempi di operazioni insiemistiche: Differenza

Matricola	Nome	Cognome	Indirizzo
150	Alex	Del Piero	via dei Palloni, 30
151	Martina	Stellina	via del Cielo, 40
142	Giovanni	SenzaTerra	via delle Crociate, 30

r1 - Studenti di Basi di Dati

Matricola	Nome	Cognome	Indirizzo
142	Giovanni	SenzaTerra	via delle Crociate, 30
149	Marco	Rossi	via dei colori, 40

r2 - Studenti di Analisi I

r1 - r2 - Studenti che frequentano il corso di Basi di Dati e non il corso di Analisi I

Matricola	Nome	Cognome	Indirizzo
150	Alex	Del Piero	via dei Palloni, 30
151	Martina	Stellina	via del Cielo, 40

OPERAZIONI DI MODIFICA DELLO STATO DELLA BASE DI DATI

- Premessa: data una relazione r
  - Il modello proposto è valido nel caso di operazioni che coinvolgono **una sola tupla di r**
  - Dopo l'introduzione delle query SQL tale modello sarà esteso ad operazioni che coinvolgono **più tuple di r**
- Operazioni considerate:
  - **insert** – inserimento di una nuova tupla all'interno di una relazione
  - **delete** – cancellazione di una tupla da una relazione
  - **update** – modifica dei valori di una tupla all'interno di una relazione

Inserimento

- Un'operazione di **inserimento** permette di inserire una nuova tupla all'interno di una data relazione **R** definita su un insieme di attributi **X={A1,A2,…,An}** con $v_i \in dom(A_i)$

$$INSERT ::= \langle v_1, v_2, \ldots, v_n \rangle \cup R \rightarrow R$$

- L'inserimento può causare la violazione di:
  - Vincoli di dominio
  - Vincoli di chiave primaria
  - Vincoli di integrità referenziale
- …a valle della violazione di un vincolo di norma un DBMS rifiuta l'operazione

Inserimento in SQL

Sintassi SQL

INSERT INTO *NOME TABELLA* **VALUES** (*Elenco Valori Tupla*)

Inserimento in SQL: esempi

- Inserimento di un nuovo studente del corso di Basi di Dati con tutte le informazioni ad esso relative

```
INSERT INTO STUDENTI_BASIDATI
  VALUES (155,'Carla','Fracci','Via
  del Campo 21')
```

- Inserimento di un nuovo studente corso di Basi di Dati con le sole informazioni relative a Matricola e Nome

```
INSERT INTO
  STUDENTI_BASIDATI(Matricola,Nome)
  VALUES (160,'Carla')
```


Cancellazione

- Un'operazione di **cancellazione** elimina una tupla da una data relazione **R** definita su un insieme di attributi **X={A1,A2,…,An}** con **vi ∈ dom(Ai)**

$$\textbf{DELETE} ::= R - \langle v_1, v_2, \ldots, v_n \rangle \rightarrow R$$

- La cancellazione può causare la violazione di un vincolo di integrità referenziale
- A valle della violazione di un vincolo un DBMS può adottare 3 politiche di reazione:
  - *Rifiuto della cancellazione* o *no action* – la cancellazione non viene eseguita
  - *Propagazione in cascata* o *cascade* – vengono cancellate in cascata tutte le tuple della tabella referente che referenziano la tupla cancellata nella tabella riferita
  - *Modifica del valore degli attributi riferiti* o *set* – ogni valore nella relazione referente è posto a NULL (se non esiste un vincolo di NOT NULL) oppure ad un valore valido nella tabella riferita

Cancellazione in SQL

Sintassi SQL

DELETE FROM *NOME TABELLA* **WHERE** *Condizione*

Esempio:

Cancellazione dello studente del corso di Basi di Dati avente matricola pari a 155

```
DELETE FROM STUDENTI_BASIDATI
WHERE Matricola=155
```

Modifica

- Un'operazione di **modifica** cambia i valori su uno o più attributi di una tupla di una data relazione **R** definita su un insieme di attributi **X={A1,A2,…,An}** con **vi ϵ dom(Ai)**

UPDATE ::= R - $\langle v_1,v_2,\ldots,v_n\rangle \rightarrow R$;

$\langle v_1,v_2,\ldots,v_n\rangle \cup R \rightarrow R$.

- In caso di aggiornamento, il DBMS deve:
  - verificare la condizione *vi ϵ dom(Ai)*
  - se la modifica coinvolge una chiave primaria, si hanno effetti simili alle operazioni combinate di DELETE e INSERT
  - se la modifica coinvolge una chiave esterna il DBMS deve verificare che il nuovo valore nella tabella referente sia presente nella tabella riferita, oppure sia NULL

Modifica in SQL

Sintassi SQL

UPDATE *NOME TABELLA* **SET** *Attributo = Espressione* {,*Attributo = Espressione* }
[**WHERE** *Condizione*]

Esempio

Modifica del nome e del cognome dello studente del corso di Basi di Dati avente matricola pari a 155.

```
UPDATE STUDENTI_BASIDATI SET Nome='Carletta',
       Cognome='Fraccini'
WHERE Matricola=155
```


OPERAZIONI RELAZIONALI

- Operazioni classiche del modello relazionale:
  - **Proiezione**
  - **Selezione**
  - **Join**
- Approccio proposto:
  - **ad un descrizione formale di tipo procedurale delle operazioni seguirà la descrizione della sintassi – di tipo dichiarativo - SQL.**

Proiezione

- Sia dato uno schema di relazione *R(X)*; sia *Y* un sottoinsieme di attributi di *X* e sia *r* una istanza di R. Si definisce **proiezione** della relazione *r* rispetto a *Y* l'insieme dei valori di *Y* di tutte le tuple t della relazione

$\pi_Y(r) = \{t[Y], t \in r\}$

Matricola,nome e cognome di tutti gli studenti che frequentano il corso di Basi di Dati

Matricola	*Nome*	*Cognome*
150	Alex	Del Piero
151	Martina	Stellina
142	Giovanni	SenzaTerra

Osservazioni sull'operatore di proiezione

- Si noti che se dopo la proiezione si hanno due tuple uguali (e potrebbe capitare se la proiezione non contiene attributi chiave), allora le tuple ripetute vengono fuse in un'unica tupla.

ANAGRAFICA

Matricola	Nome	Cognome	DataNascita
150	Alex	Del Piero	10/3/1975
151	Alex	Del Piero	11/2/2004
142	Giovanni	SenzaTerra	3/4/1250

Nomi e cognomi degli studenti presenti in anagrafica

$\pi_{\text{Nome, Cognome}}(\text{ANAGRAFICA})$

Nome	Cognome
Alex	Del Piero
Giovanni	SenzaTerra

Proiezione in SQL

Sintassi SQL

SELECT [**DISTINCT**] *Elenco Attributi* FROM *NOME TABELLA*

Nomi e cognomi (senza ripetizioni) di tutti gli studenti presenti in anagrafica

SELECT DISTINCT Nome,Cognome
FROM ANAGRAFICA

Nomi e cognomi (con ripetizioni) di tutti gli studenti presenti in anagrafica

SELECT Nome,Cognome
FROM ANAGRAFICA

Selezione

- Definizione di selezione
  - Sia data una relazione $r \in R(X)$ ed una condizione di selezione χ Si Definisce **selezione** l'insieme:

$\sigma_{\chi}(r)=\{t:t \models \chi\}$

Studenti in anagrafica che hanno come nome 'Alex'

$\sigma_{\text{Nome='Alex'}}(\text{ANAGRAFICA})$

Matricola	Nome	Cognome	DataNascita
150	Alex	Del Piero	10/3/1975
151	Alex	Del Piero	11/2/2004

Soddisfacimento condizione selezione χ dalla tupla t : $t \models \chi$

Condizione Atomica
Sia *r* una relazione su *R(X)* e siano $A \in X$ e $B \in X$ due attributi di *r*, e sia $\theta \in \{=,\neq,<,>,\leq,\geq\}$ una qualsiasi relazione definita sul dominio di *A* e *B*. Le seguenti sono condizioni atomiche: (i) $A\ \theta\ B$; (ii) $A = c$, per ogni valore costante *c* del dominio di *A*

Condizione di Selezione
Ogni condizione atomica è una Condizione di Selezione. D'altro canto, se *vc1* e *vc2* sono due condizioni di selezione, lo sono pure $vc1 \wedge vc2$, $vc1 \vee vc2$, e $\neg vc1$

Soddisfacimento di una condizione di selezione
Sia *R(X)* uno schema di relazione, *r* una istanza di *R*, *t* una tupla in r, *A* e *B* attributi in *X*, e *c* una costante. Sia χ una qualsiasi condizione di selezione su *A*, *B* e *c*. Il soddisfacimento di χ da *t* ($t \models \chi$) è definita per induzione sulla struttura di χ, come segue:

- Paragone tra attributi: $t \models A\ \theta\ B$ se e solo se $t[A]\ \theta\ t[B]$
- Paragone tra costanti: $t \models A\ \theta\ c$ se e solo se $t[A]\ \theta\ c$

Selezione in SQL

Sintassi SQL

SELECT * **FROM** *NOME TABELLA* **WHERE** *Condizione*

Studenti presenti in anagrafica con nome 'Alex'

```
SELECT  *
FROM    ANAGRAFICA
WHERE   Nome =Alex
```


Query SQL

- Una query SQL su una tabella r si ottiene combinando un'operazione di proiezione con una di selezione

$$\pi_Y(\sigma_\chi(r))$$

Sintassi SQL

SELECT *Y* **FROM** *r* **WHERE** χ

Confronto tra stringhe

- Per quanto attiene i valori del tipo stringa, uso far riferimento anche ad operatori differenti rispetto a quelli classici di confronto tra elementi di un insieme
- Spesso si richiede di effettuare confronti solo su parti di stringhe di caratteri e non sull'intera stringa: ciò avviene in SQL attraverso l'operatore **LIKE**.
  - Tale operatore si utilizza assieme a due caratteri riservati
    - il simbolo di "%" che sostituisce un numero arbitrario di caratteri
    - il carattere "_" che sostituisce un singolo carattere

Cognomi degli studenti in anagrafica che hanno la lettera 'e' in seconda posizione e terminano con la lettera 'a'

```
SELECT Nome, Cognome
FROM   ANAGRAFICA
WHERE Cognome  LIKE  '_e%a'
```

Nome	*Cognome*
Giovanni	Senzaterra

Operatori aritmetici e di ordinamento

- E' possibile:
  - applicare gli operatori aritmetici agli attributi di proiezione aventi dominio numerico
  - Oridinare le tuple risultato di un'interrogazione attraverso la clausola **ORDER BY** in maniera crescente (**ASC**) o decrescente (**DESC**)

Nome, cognome e stipendio incrementato di 10 euro degli impiegati di un'azienda

```
SELECT Nome,Cognome,Stipendio+10
FROM   ANAGRAFICA
```

Cognomi (ordinati alfabeticamente) e nomi (ordinati in senso decrescente), e degli studenti in anagrafica

```
SELECT Cognome, Nome
FROM   ANAGRAFICA
ORDER BY Cognome,Nome DESC
```

Prodotto cartesiano

- Date due relazioni *r1* e *r2* definite sugli schemi *R(X1)* e *R(X2)* con $X1 \cap X2 = \emptyset$, il **prodotto cartesiano** tra *r1* e *r2* restituisce restituisce un'istanza di relazione r definita su $R(X1 \cup X2)$ e contenente le tuple *t* formate dalla **concatenazione** di ciascuna tupla di *r1* con ciascuna tupla di *r2*

$$r = r_1 \times r_2 = \{ t = \langle t_1, t_2 \rangle, t_1 \in r_1 \wedge t_2 \in r_2 \}$$

r1

Nome	*Cognome*
carlo	rossi
mirko	verdi
paolo	paolone

r2

Codice	*Qualifica*
001	impiegato
002	dirigente

r1 x r2

Nome	*Cognome*	*Codice*	*Qualifica*
carlo	rossi	001	impiegato
carlo	rossi	002	dirigente
mirko	verdi	001	impiegato
mirko	verdi	002	dirigente
paolo	paolone	001	impiegato
paolo	paolone	002	dirigente

Dot Notation

- Qualora le due relazioni posseggano attributi con lo stesso nome si utilizza la cosiddetta *dot notation* per distinguere gli attributi comuni nel prodotto cartesiano
  - *NomeRelazione. NomeAttributo*
    - Es. *r1.nome*, *r2.codice*, ...

Theta Join ed Equi Join

- Date due relazioni *r1* su *R(X1)* ed *r2* su *R(X2)*, con:
  X1 ∩ X2 = Ø
  $\chi \equiv a_{r1}\ \vartheta\ a_{r2}$
  si definisce **THETA JOIN** l'operazione:

$$\sigma_{\chi}(r_1 \times r_2) \equiv r_1 \bowtie_{\chi} r_2$$

  - Se ϑ è una uguaglianza tra i valori di un attributo di *r1* e quelli di *r2* si parla di **EQUI JOIN**

ANAGRAFICA_DIPENDENTI

Nome	*Cognome*	*Mansione*
carlo	rossi	001
mirko	verdi	001
paolo	paolone	002

MANSIONI

Codice	*Qualifica*
001	impiegato
002	dirigente

ANAGRAFICA_DIPENDENTI $\bowtie_{Codice=Mansione}$ MANSIONI

Nome	*Cognome*	*Mansione*	*Codice*	*Qualifica*
carlo	rossi	001	001	impiegato
mirko	verdi	001	001	impiegato
paolo	paolone	002	002	dirigente

Join in SQL

- In SQL, l'operazione di JOIN può essere effettuata in:
  - *modo implicito*
  - *modo esplicito*
- Il join implicito si ottiene applicando la definizione data di equi-join, come prodotto cartesiano seguito da una selezione:

```
SELECT *
FROM r1, r2
WHERE Condizione
```

  - **Esempio:**

```
SELECT *
FROM ANAGRAFICA DIPENDENTI, MANSIONI
WHERE Mansione=Codice
```

- **Il join esplicito richiede invece che una istruzione SQL esprima l'operazione di join, come nel seguente modo:**

```
SELECT *
FROM r1 JOIN r2 ON Condizione
```

  - **Esempio:**

```
SELECT *
FROM ANAGRAFICA DIPENDENTI JOIN MANSIONI
ON Mansione=Codice
```

Tuple Dondolanti

- Se si considerano le istanze di relazioni sulla destra si può notare:
  - La terza tupla della relazione MANSIONI non è coinvolta nelloperazione di equi join
    - Non esiste un impiegato con la mansione di amministratore
    - Si parla di "**tupla dondolante**" (*dangling tuple*)

ANAGRAFICA_DIPENDENTI

Nome	*Cognome*	*Mansione*
carlo	rossi	001
mirko	verdi	001
paolo	paolone	002

MANSIONI

Codice	*Qualifica*
001	impiegato
002	dirigente
003	amministratore

Join Naturale

- Un caso particolare di theta join è il **JOIN NATURALE**, in cui la condizione *di uguaglianza* è implicitamente verificata su *tutte le coppie di attributi delle due relazioni aventi lo stesso nome*
- Se, ad esempio, *r*1 ed *r*2 hanno il nome di un attributo comune, chiamiamolo *Ac*, vale la seguente equivalenza:

$$r_1 \bowtie r_2 \equiv r_1 \bowtie_{r_1.A_c=r_2.A_c} r_2$$

Join Esterno (1/2)

- La presenza di tuple dangling che non partecipano alle relazioni di join può essere a volte dannosa in quanto tralascia alcune informazioni utili
- Per evitare ciò, esiste una variante dell'operazione di join detto **JOIN ESTERNO**
- In questo caso, date due relazioni *rs* e *rd* – dette anche relazione destra e relazione sinistra, in quanto compaiono nella parte destra e nella parte sinistra del join valgono le seguenti operazioni di esterno:
  - ***join destro*** $r_s \bowtie_{\text{RIGHT}} r_d$
  - ***join sinistro*** $r_s \bowtie_{\text{LEFT}} r_d$
  - ***join completo*** $r_s \bowtie_{\text{FULL}} r_d$

Join Esterno (2/2)

- **Il join esterno prevede che tutte le tuple della relazione *rs* o della relazione *rd* o di entrambe concorrano alla formazione del risultato, ponendo NULL dove l'operazione di join coinvolge tuple dondolanti.**
- **In particolare nel:**
  - **join esterno sinistro vengono considerate tutte le tuple della relazione sinistra**
  - **join esterno destro vengono considerate tutte le tuple della relazione destra**
  - **join completo vengono considerate tutte le tuple della relazione sinistra e della relazione destra.**

r_S

Nome	*Cognome*	*Mansione*
carlo	rossi	001
paolo	paolone	002

r_D

Codice	*Qualifica*
001	impiegato
002	dirigente
003	amministratore

$r = r_S \bowtie_{\text{RIGHT}} r_D$

Nome	*Cognome*	*Codice=Mansione*	*Qualifica*
carlo	rossi	001	impiegato
paolo	paolone	002	dirigente
NULL	NULL	003	amministratore

SELECT *
FROM r_s RIGHT JOIN r_d ON r_d.Codice=r_s.Mansione

Ridenominazione

- Sia *r* una relazione definita su uno schema *R(X)* e sia *Y* un insieme di attributi avente la stessa cardinalità di *X*.
- Sia definito un ordinamento tra gli attributi di *X* e quelli di *Y* , diciamolo *A1,A2…Ak* e *B1,B2…Bk*, con la proprietà che *dom(Ai) = dom(Bi)* con *i=1…k*.
- L'operazione di **ridenominazione** è la seguente:

$$\varrho_{B_1 B_2 \ldots B_k \leftarrow A_1 A_2 \ldots A_k}(r) = \{t' : \exists t \in r \wedge t[A_i] = t'[B_i], \forall i = 1 \ldots k\}$$

Ridenominazione: esempio

Nome	*Cognome*	*Mansione*
carlo	rossi	001
mirko	verdi	001
paolo	gialli	002

- Ridenominiamo in inglese gli attributi

$\varrho_{Name,Surname,Position \leftarrow Nome,Cognome,Mansione}$

Name	*Surname*	*Position*
carlo	rossi	001
mirko	verdi	001
paolo	gialli	002

Ridenominazione in SQL

- In SQL possiamo effettuare una ridenominazione attraverso il meccanismo degli *alias*.

```
SELECT A1 [AS] alias1, A2 [AS] alias2, ...,
       An [AS] aliasn
FROM NOME TABELLA
```

- Un esempio dato dalla seguente query:

```
SELECT Nome AS Name, Cognome AS Surname
FROM ANAGRAFICA_DIPENDENTI
```

- E' possibile usare il meccanismo degli alias anche sui nomi di tabella che permette di associare un nome alla tabella nella clausola FROM

```
SELECT T1.Nome, T1.Cognome, T2.Qualifica
FROM ANAGRAFICA DIPENDENTI T1, MANSIONI T2
WHERE T1.Mansione=T2.Codice
```


Uso di alias

- **L'uso di alias su variabile ed attributi permette di utilizzare più volte una stessa tabella in una interrogazione**
- **Ad esempio se si vogliono trovare i nomi e cognomi dei dipendenti la cui mansione è uguale a quelle di un dipendente assegnato, allora è possibile usare la seguente query SQL**

```
SELECT T1.Nome, T1.Cognome
FROM ANAGRAFICA DIPENDENTI T1, ANAGRAFICA DIPENDENTI T2
WHERE (T1.Mansione=T2.Mansione) AND (T2.Nome='carlo')
AND (T2.Cognome='rossi') AND (T1.Nome <> 'carlo') AND
(T1.Cognome <>'rossi')
```

Aggregazione e raggruppamento (1/4)

- **Per *funzioni aggregate* si intende un insieme di funzioni che operano su "*collezioni di valori della base di dati*". Classiche funzioni aggregate sono le operazioni di COUNT, MIN, MAX, e AVG**
- **Si può prevedere di raggruppare dapprima le tuple sulla base del valore di uno o più attributi, quindi di applicare a ciascun gruppo così ottenuto, le funzioni di aggregazione. Ciò si ottiene attraverso opportune clausole dette *di raggruppamento***

Aggregazione e raggruppamento (2/4)

- **Le operazioni di *aggregazione e raggruppamento* non sono definite in algebra relazionale, mentre sono di uso molto comune in SQL, dal momento che aggregazione e raggruppamento sono operazioni comuni in molte applicazioni di basi di dati.**
- **SQL supporta cinque diverse funzioni di aggregazione, che possono essere applicate ad un qualsiasi attributo di una relazione. In particolare:**
  - **COUNT(*): restituisce il numero di tuple o di valori presenti nel risultato di una interrogazione**
  - **MIN(), MAX(), MIN(), AVG(): sono funzioni che applicate ad un insieme di valori numerici ne restituisce la somma, il massimo, il minimo e la media aritmetica**

Aggregazione e reggruppamento (3/4)

- Le operazioni di raggruppamento si ottengono attraverso le clausole:
  - GROUP BY *ElencoDiAttributi*: permette di raggruppare le tuple della relazione in sottoinsiemi caratterizzati dallo stesso valore degli attributi che compaiono come argomento della clausola stessa
  - HAVING (*Condizioni*): permette di specificare delle condizioni logiche (predicati) che devono essere verificate sul sottoinsieme di tuple precedentemente raggruppate con la clausola GROUP BY

Aggregazione e reggruppamento (4/4)

- Valgono le seguenti osservazioni.
  - **In una interrogazione SQL contenente la clausola GROUP BY, un attributo può comparire come argomento della clausola SELECT solo se è presente nell'elenco degli attributi della clausola GROUP BY**
  - **Un errore molto comune nella scrittura delle interrogazioni SQL quello di confondere nei raggruppamenti la clausola WHERE con la clausola HAVING:**
    - ***Una semplice regola da seguire è la seguente: se le condizioni sono verificabili a livello delle singole tuple, allora si usa la clausola WHERE; se le condizioni sono verificabili su un gruppo, si usa la clausola HAVING.***

Esempi di funzioni di aggregazione e raggruppamento (1/2)

ANAGRAFICA(Matricola, Nome, Cognome, NomeDip, Stipendio)
PROGETTI(Codice, Nome, Anno, Fondi)
TEAM(Mat:ANAGRAFICA, Prog:PROGETTI)

- Trovare il numero totale di impiegati presenti in anagrafica

```
SELECT COUNT(*) AS Numero_Impiegati
FROM ANAGRAFICA
```

- Trovare il numero totale di impiegati per ogni dipartimento

```
SELECT NomeDip AS Nome_Dipartimento,
COUNT(Matricola) AS Numero_Impiegati
FROM ANAGRAFICA
GROUP BY NomeDip
```

- Trovare il numero totale di impiegati per ogni dipartimento ed il loro stipendio medio

```
SELECT NomeDip AS Nome_Dipartimento,
COUNT(Matricola) AS Numero_Impiegati,
AVG(Stipendio) AS StipendioMedio
FROM ANAGRAFICA
GROUP BY NomeDip
```


Esempi di funzioni di aggregazione e raggruppamento (2/2)

ANAGRAFICA(Matricola, Nome, Cognome, NomeDip, Stipendio)
PROGETTI(Codice, Nome, Anno, Fondi)
TEAM(Mat:ANAGRAFICA, Prog:PROGETTI)

- Trovare il nome dei progetti su cui lavorano più di 3 persone

```
SELECT    P.Nome AS Nome_Progetto, COUNT(*) AS
          Numero_Impiegati
FROM      TEAM T JOIN PROGETTO P ON T.Prog=P.Codice
GROUP BY P.Nome
HAVING    COUNT(*)>=3
```


Operatori insiemistici in SQL

- SQL permette di usare all'interno di interrogazioni anche gli operatori insiemistici già visti. Le parole chiave utilizzabili sono:
  - UNION, per l'unione
  - INTERSECT, per l'intersezione
  - EXECEPT, per la differenza

```
SELECT *
FROM STUDENTI_BASIDATI
UNION
SELECT *
FROM STUDENTI_ANALISI1

SELECT *
FROM STUDENTI_BASIDATI
INTERSECT
SELECT *
FROM STUDENTI_ANALISI1

SELECT *
FROM STUDENTI_BASIDATI
EXCEPT
SELECT *
FROM STUDENTI_ANALISI1
```


Query nidificate

- In alcuni casi può essere utile esprimere le interrogazioni SQL con una struttura più complessa in cui, nella clausola WHERE, la condizione è costituita dal **confronto** del **valore di un attributo** con il risultato di un'altra query SQL (*query nidificate*).
- Si noti che ciò possibile se ai normali ***operatori di confronto*** binari se ne aggiungono altri come di seguito elencato.
  - ***op* ALL**: la condizione specificata da *op*, con *op* operatore di confronto, è verificata se e solo se verificata da tutti gli elementi restituiti dalla query nidificata;
  - ***op* ANY**: la condizione specificata da *op*, con *op* operatore di confronto, è verificata se e solo se verificata da almeno uno degli elementi restituiti dalla query nidificata;
  - **IN** e **NOT IN**: sono del tutto identici a =ANY e ≠ALL, rispettivamente
- Inoltre **gli operatori di confronto unari**:
  - EXISTS, NOT EXISTS: verificano se un insieme è non vuoto (oppure vuoto)

Query nidificate

- Le query nidificate sono dunque costituite da una interrogazione che ha al suo interno un altra interrogazione, a volte detta *sottointerrogazione*.
- Chiaramente, una sottointerrogazione può contenere al suo interno una ulteriore sottointerrogazione

Esempi di query nidificate (1/2)

ANAGRAFICA(Matricola, Nome, Cognome, NomeDip, Stipendio)
PROGETTI(Codice, Nome, Anno, Fondi)
TEAM(Mat:ANAGRAFICA, Prog:PROGETTI)

- Trovare nomi e cognomi delle persone che partecipano al progetto con codice 'EUI05'

```
SELECT A.Nome, A.Cognome
FROM ANAGRAFICA A
WHERE A.Matricola IN (
        SELECT T.Mat
        FROM TEAM T
        WHERE T.Prog='EU105')
```

- Trovare nomi e cognomi delle persone che partecipano al progetto "Sistemi Informatici e Calcolo Parallelo"

```
SELECT A.Nome, A.Cognome
FROM ANAGRAFICA A
WHERE A.Matricola IN (
        SELECT T.Mat
        FROM TEAM T
        WHERE T.Prog IN (
                SELECT P.Codice
                FROM Progetti P
                WHERE P.Nome='Sistemi Informatici e Calcolo Parallelo')
```


Esempi di query nidificate (2/2)

- Trovare nomi e cognomi delle persone che partecipano al progetto con codice 'EUI05'

```
SELECT A.Nome, A.Cognome
FROM ANAGRAFICA A
WHERE EXISTS (
        SELECT *
        FROM TEAM T
        WHERE T.Prog='EU105' AND
T.mat=A.Matricola)
```

- Trovare il nome dei progetti i cui fondi sono maggiori del progetto "IDEA"

```
SELECT P1.Codice
FROM Progetto P1
WHERE P1.Fondi > ANY (
        SELECT P2.Fondi
        FROM Progetti P2
        WHERE P2.Nome='IDEA')
```

- Trovare il codice dei progetti con maggiori fondi

```
SELECT P1.Codice
FROM Progetto P1
WHERE P1.Fondo >= ALL (
        SELECT P2.Fondo
        FROM Progetti P2)
```


Viste

- Una *vista* è una tabella che viene descritta da viste (tabelle) precedentemente definite
- E' una *relazione virtuale*, nel senso che le sue tuple non sono effettivamente memorizzate nella base di dati, quanto piuttosto ricavabili attraverso interrogazioni da altre tuple presenti nella base di dati
- E' una *interfaccia* da mettere a disposizione di utenti o di applicazioni per le interrogazioni: si tratta di dati usati di frequente che possono anche non esistere fisicamente
  - …ciò limita le operazioni sulle viste, in particolare quelle di *aggiornamento*
- In SQL una vista viene creata attraverso la sintassi seguente:

```
CREATE VIEW NOME VISTA
AS QuerySQL
```

  - Esempio:

```
CREATE VIEW IMPIEGATI_DIS AS
SELECT *
FROM ANAGRAFICA
WHERE NomeDip='DIS'
```

Sintassi SQL semplificata

- Sintassi SQL per le query:

  SELECT *ElencoAttributi*
  FROM *ElencoTabelle*
  [**WHERE** *CondizioneLogica*]
  [**GROUP BY** *ElencoAttributi*]
  [**HAVING** *CondizioneDiRaggruppamento*]
  [**ORDER BY** *ElencoAttributi*]

Operazioni set-oriented

- Con l'introduzione dello statement SELECT, le operazioni di insert, update e delete già presentate, possono riferirsi anche ad insiemi di tuple, consentendo così l'inserimento, l'aggiornamento e la cancellazione di più informazioni all'interno della base di dati attraverso un unico costrutto (operazioni *set oriented*)

```
INSERT INTO IMPIEGATI DIS (
SELECT *
FROM ANAGRAFICA
WHERE NomeDip='DIS')

DELETE FROM ANAGRAFICA
WHERE NomeDip='DIS'

UPDATE ANAGRAFICA SET Cognome=''
WHERE NomeDip='DIS'
```