FUNZIONI SQL

*Manipolazione **date***
extract(year FROM data_nascita)
date_part('year', data_nascita)
EXTRACT(year FROM **AGE**(ur.data_nascita)) as "Età"
EXTRACT(**EPOCH** FROM AGE(fine, inizio)) [restituisce il numero di secondi]

*Gestione **duplicati***
select **DISTINCT**
    Attributo

*Gestione **valori nulli***
WHERE attributo **IS NULL**
WHERE attributo **IS NOT NULL**

***Ordinamento** dei risultati*
ORDER BY
    Attributo **DESC;**
ORDER BY
    Attributo **ASC;**
ORDER BY
    Attributo **NULLS FIRST;**

Join interno
INNER JOIN
    tabella **AS** nome **ON** tabella.attributo = altra_tabella.attributo

Se gli attributi di join hanno lo stesso nome

select * from S [inner] **JOIN** SP **USING** (SNum) where S.City='London' and SP.PNum='P4';

Join naturale
Se tutti (e soli) gli attributi su cui applicare la condizione di join hanno lo stesso nome, si può utilizzare il natural join

select * from S **NATURAL JOIN** SP where S.City='London' and SP.PNum='P4';

***Left** join*
Si hanno nel risultato tutte le righe della tabella che nella query compare a sinistra
Se per una riga della tabella di sinistra non ci sono righe della tabella di destra per cui la clausola è vera, nel risultato i valori corrispondenti agli attributi della tabella di destra avranno valore nullo

LEFT JOIN
    tabella **AS** nome **ON** tabella.attributo = altra_tabella.attributo

***Right** join*
Opposto del left join

RIGHT JOIN
tabella **AS** nome **ON** tabella.attributo = altra_tabella.attributo

***Full** join*
Riporta nel risultato tutte le righe di entrambe le tabelle

FULL JOIN
tabella **AS** nome **ON** tabella.attributo = altra_tabella.attributo

***Self** join*
È possibile effettuare il join (interno o esterno) di una tabella con se stessa
In questo caso è necessario usare degli alias per assegnare nomi diversi alla tabella

```
select distinct
        Fornit1.SNum Fornitore1, Fornit2.SNum Fornitore2, Fornit1.City
from
        S Fornit1
join
        S Fornit2 on Fornit1.City= Fornit2.City
where
        Fornit1.Snum < Fornit2.SNum;
```

***Conteggio** tuple*
COUNT(attributo)

Funzioni matematiche
SUM(attributo)
MAX(attributo)
MIN(attributo)
AVG(attributo)
ROUND(AVG(attributo), **2**)

Alternativa a MAX/MIN
SELECT colonna
FROM tabella
ORDER BY colonna **DESC/ASC**
LIMIT 1;

Utilizzare **all** per controllare che il valore sia il più grande, piccolo di tutti

Alternativa a AVG
SELECT **SUM(colonna) / COUNT(colonna)** AS Media
FROM tabella;

***Raggruppamento** attributi*

GROUP BY

attributo

Having

HAVING

Operatore aggregato ><= valore

Differenza tra having e where:

• nella clausola **having** generalmente vanno messe solo condizioni in cui compaiono operatori aggregati

• altrimenti i predicati possono essere espressi nella clausola **where**

***Unione** insiemistica*

Date due relazioni A e B, A ∪ B è l'insieme delle tuple x tali che x appartiene ad A o appartiene a B (o appartiene a entrambi)

SELECT nome_utente FROM utente_registrato
UNION
SELECT nome_streamer FROM streamer

***Intersezione** insiemistica*

Date due relazioni A e B, A ∩ B è l'insieme delle tuple x tali che x appartiene ad A e appartiene anche a B

SELECT nome_utente FROM utente_registrato
INTERSECT
SELECT nome_streamer FROM streamer

***Sottrazione** insiemistica*

Date due relazioni A e B, A except B è l'insieme delle tuple x tali che x appartiene ad A ma non appartiene a B

SELECT nome_utente FROM utente_registrato
EXCEPT
SELECT nome_streamer FROM streamer

Sottoquery

- **Semplici** *(o stratificate): è possibile valutare prima l'interrogazione più interna (una volta per tutte), poi, sulla base del suo risultato, valutare l'interrogazione più esterna*
- **Correlate** (o incrociate)**:** l'interrogazione più interna fa riferimento a una delle tabelle appartenenti all'interrogazione più esterna. Per ciascuna riga candidata alla selezione nell'interrogazione più esterna, è necessario valutare nuovamente la sottoquery

Possibile utilizzare le sottoquery:

- nella clausola **where** (utilizzando test di selezione come **all/any, in, exists**)
- nella clausola **from** (permettono di valutare query su tabelle derivate)
- nella clausola **select** (standard SQL:2003, non supportata da tutti i DBMS)

where City = **(select City from S where SNum='S1')**;

Operatori di confronto:

- **any**: il predicato deve essere soddisfatto da almeno una riga restituita dalla sottointerrogazione
- **all**: il predicato deve essere soddisfatto da tutte le righe restituite dalla sottointerrogazione

Verificare che un valore sia presente:

- in un insieme possiamo utilizzare le condizioni **IN** e **NOT IN**

ur.nome_utente **NOT IN** (sottoquery)

*Creare una tabella **senza** Create Table*
SELECT *
INSERT INTO nuova_tabella
FROM tabella_esistente

*Operatore **like***
Confrontare una stringa con un'altra stringa in cui possono comparire i seguenti caratteri speciali:
_ (trattino basso) = un carattere qualsiasi
% (percentuale) = una sequenza di lunghezza arbitraria (eventualmente anche zero) di caratteri arbitrari

SELECT * FROM utente_registrato where nome **like** 'N%';

Come creare le viste
CREATE VIEW NomeVista as
(select … from … where …)
With check option;
Può essere usata come se fosse una tabella che ha come attributi quelli definiti dalla select

È possibile creare viste a partire da altre viste
Check option:

- **Local**: vengono annullate solo le modifiche che violano le condizioni della vista che si sta modificando
- **Cascaded:** vengono annullate anche le modifiche che violano le condizioni delle viste da cui la vista è originata