Query complesse

Query annidate

Le **query annidate:** uno strumento sintattico molto importante per effettuare interrogazioni complesse sui database. Un'**interrogazione nidificata** (o subquery) è una query all'interno di altra interrogazione

In generale una query SQL può avere delle subquery (o sottoquery) sia nella clausola **SELECT**, sia nella clausola **FROM** sia nella clausola **WHERE** (o Having).

È possibile utilizzare una **sottoquery come alias di campo**, ad esempio quando si desidera utilizzare i risultati della sottoquery come un campo nella query principale.

```
(SELECT MAX([Order Date])
FROM [Product Orders] AS [Old Orders]
WHERE [Old Orders].[Order Date]
 < [Product Orders].[Order Date]
AND [Old Orders].[Product ID]
 = [Product Orders].[Product ID])  AS Prior Date
```

Utilizzabile in altra query come **campo**

È possibile utilizzare una sottoquery come **criterio di campo**, ad esempio quando si desidera utilizzare i risultati della sottoquery per limitare i valori visualizzati nel campo.

```
IN (SELECT [ID] FROM [Employees]
WHERE [Job Title]<>'Sales Representative')
```

SUBQUERY: query nidificate vere e proprie e **inline view**

query nidificate
inline view o *tabelle derivate*: subquery in clausola FROM di una SELECT (referenziate tramite un ALIAS)

Nel formulare query per interrogare le tabelle di un database, se ne esamini la sintassi:

- il comando SELECT per selezionare i campi (le colonne) di nostro interesse;
- la clausola FROM per indicare le tabelle su cui eseguire la query;
- la clausola WHERE per indicare condizioni di filtro e prelevare i soli dati che soddisfano tali condizioni;
- la clausola ORDER BY[1] per impostare un ordine di visualizzazione dei dati.

Una "subquery" è una query inclusa in un'altra, ovvero un'interrogazione all'interno di altre interrogazioni.

Una subquery ritorna dei dati (*tabelle* o *dati singoli*) necessari all'esecuzione di query ad un livello più alto.

Una query così strutturata rispetta l'ordine gerarchico di esecuzione: di solito il DBMS esegue prima le query più interne e, una volta completate, quelle situate a "*livelli*" superiori.

Una query situata a un livello più basso rispetto ad un'altra viene definita "*child*", al contrario la query situata ad un livello più alto rispetto alla child viene definita "*parent*".

Una subquery può contenere altre query nidificate al suo interno. Questa nidificazione può proseguire per molti livelli (nel caso di **query nidificate** si possono avere al massimo 255 livelli di nidificazione o "*strati*" di interrogazione).

[1] Query con ordinamento, a volte, elencate tra query complesse

- Bisogna essere consapevoli, di volta in volta, di **quali dati sono restituiti** da una query nidificata

Le query nidificate possono restituire un **valore singolo** (subquery ***scalari***) o **valori multipli**:

Il **caso più semplice** accade quando una subquery restituisce una *relazione di grado uno (unica riga)*. Se una subquery **restituisce un singolo valore**, allora è lecito usare la subquery come operando di operatori non insiemistici.

Tipicamente una subquery **restituisce più valori**, è per questo che una subquery appare come operando di operatori insiemistici;

Subquery a riga singola

Hanno la particolarità di restituire sempre una sola riga di dati.

Per esempio cerchiamo l'impiegato (tabella employees) con lo stipendio più alto.

```
SELECT last_name AS cognome, first_name AS nome, salary AS salario
 FROM employees
  WHERE salary = (SELECT MAX (salary) FROM employees);
```

Una nidificazione di questo tipo usa sempre un operatore a singola riga, solitamente quello di uguaglianza (=).

Subquery a riga multipla

Le subquery di questo tipo restituiscono più di una riga di risultati. È necessario usare *operatori a riga multipla* come `IN`, `EXISTS`, `ANY` e `ALL`.

Il **caso più frequente** è quello di subquery nella clausola **WHERE**: la `Select` nidificata che fornisce più di un valore è preceduta da **IN** o **EXIST**

Ad esempio visualizziamo gli impiegati che lavorano nei dipartimenti 'Administration' e 'Marketing'.

```
SELECT last_name AS cognome, first_name AS nome, salary
FROM employees
WHERE department_id IN
 (SELECT department_id FROM departments
   WHERE (department_name = 'Administration' OR
          department_name = 'Marketing'));
```

Gli operatori di confronto che introducono una sottoquery possono essere ***modificati*** tramite le parole chiave **ALL** o **ANY**. SOME è l'equivalente di ANY nello standard ISO.

```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ANY
    (SELECT MAX (ListPrice)
     FROM Production.Product
     GROUP BY ProductSubcategoryID) ;
```

Le sottoquery introdotte da un **operatore di confronto modificato** restituiscono un elenco di zero o più valori e possono includere una clausola GROUP BY o HAVING. Tali sottoquery *possono essere riformulate* con **EXISTS**.

Utilizzando l'operatore di confronto > come esempio,

>ALL indica maggiore di **qualsiasi valore**, ovvero maggiore del valore massimo.
Ad esempio, >ALL (1, 2, 3) significa maggiore di 3.

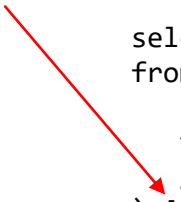
>ANY significa maggiore di **almeno un valore**, ovvero, maggiore del valore minimo.
Ad esempio, >ANY (1, 2, 3) significa maggiore di 1

Anche http://www.edatlas.it/scarica/SIA_4/Capitolo6/MaterialiOnLine/3Predicati.pdf

Esercizi_risolti http://corsiadistanza.polito.it/on-line/sistemi_info/esercizi/es_U3L3.pdf

Le subquery usate nella clausola FROM di una SELECT si chiamano **inline view** o **tabelle derivate**.

> Una **tabella derivata**, è una tabella che viene creata durante l'esecuzione della query esterna, quindi è un semplice *result-set*, cioè un insieme di tuple (o record) ottenute da una query (referenziato tramite un ALIAS)



```
select MAX(tot_salario)
from (
   select SUM(salario) as tot_salario
   from Lavoratori
   group by settore
) AS temporanea;
```

Usare le subquery in altri statement DML

Possiamo usare le subquery anche nelle operazioni di **inserimento**, **modifica** e **cancellazione** di dati (`INSERT, UPDATE, DELETE`).

Ad esempio **inseriamo** nella tabella "padre" un nuovo nominativo. Alla colonna "idpadre" (PRIMARY KEY), sarà attribuito un nuovo valore utilizzando una subquery. Questa calcola il valore massimo presente nella colonna "idpadre" ed aggiunge un'unità per rispettare il vincolo di chiave primaria.

```
INSERT INTO padre
  VALUES ((SELECT MAX (IDPADRE)+1 FROM padre),
          'Ferraris Nicola', TO_DATE ('23/03/1954', 'DD/MM/YYYY'));
```

Ancora **aggiorniamo**, nella tabella "padre", la data di nascita del nominativo 'Rominelli Giacomo'. Non conoscendo a priori il suo id, useremo la subquery per ottenere il relativo valore di "idpadre".

```
 UPDATE padre
  SET DATANASCITA = TO_DATE ('25/11/1964', 'DD/MM/YYYY')
  WHERE idpadre = (SELECT idpadre FROM padre
  WHERE nominativo = 'Rominelli Giacomo');
```

Infine **cancelliamo** le righe della tabella "padre" in cui il valore di "idpadre" è maggiore di 100.

```
DELETE FROM
  (SELECT * FROM padre WHERE idpadre > 100);
```

funzioni SQL di aggregazione

http://www.w3schools.com/sql/sql_functions.asp

Gli **operatori aggregati** (o **funzioni di aggregazione**) sono di fondamentale importanza per effettuare **subquery** complesse.

Le funzioni di aggregazioni sono funzioni standard native di SQL che permettono di ottenere valori numerici e/o effettuare calcoli in funzione di query specifiche. Di seguito l'elenco delle funzioni di aggregazione di SQL in schema tabellare (altre Transact-SQL):

Le funzioni di aggregazione eseguono un calcolo su un set di valori e restituiscono un singolo valore.

AVG()	Restituisce la media tra due valori specificati
COUNT()	Restituisce un intero che indica il numero di record trovati
MAX()	Restituisce il valore massimo tra due valori
MIN()	Restituisce il valore minimo tra due valori
SUM()	Restituisce la somma tra più record dello stesso campo

Ad eccezione della funzione COUNT, le funzioni di aggregazione ignorano i valori Null. Vengono spesso utilizzate con la clausola GROUP BY dell'istruzione SELECT.

Nell'utilizzo di una funzione di aggregazione è importante (consigliato, anche se non obbligatorio) specificare un *alias* per il risultato, con l'utilizzo della clausola AS.

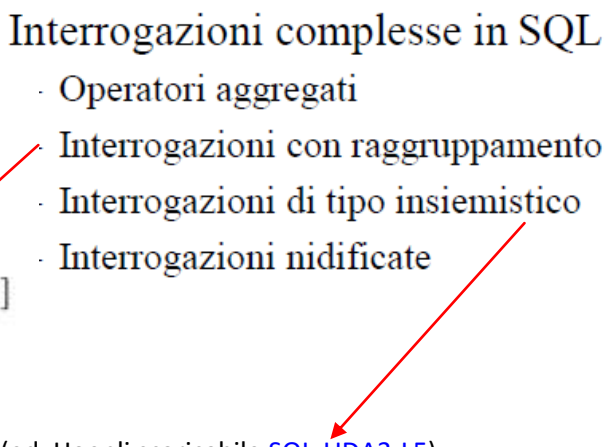
È possibile utilizzare le funzioni di aggregazione come espressioni solo nei casi seguenti:

- Nell'elenco di selezione di un'istruzione SELECT (una **sottoquery** o una query esterna).
- Nella clausola HAVING.

Appendice

SelectSQL ::=

SELECT *ListaAttributiOEspressioni*
FROM *ListaTabelle*
[**WHERE** *CondizioniSemplici*]
[**GROUP BY** *ListaAttributiDiRaggruppamento*]
[**HAVING** *CondizioniAggregate*]
[**ORDER BY** *ListaAttributiDiOrdinamento*]



Terminologia non sempre univoca nel definire query annidate (ed. Hoepli scaricabile SQL-UDA3-L5)

VIEW

Una vista, (o view), è un oggetto logico che consente di avere una rappresentazione dei dati "personalizzata" dagli utenti.

Per ottenere informazioni capita di costruire query complesse (con join, alias, nidificazioni, etc.) da dare magari ad un altro componente del team di sviluppo, che avrebbe bisogno di qualcosa di molto più semplice e meno lungo.

La vista può essere intesa come "l'alias di una query", ovvero un modo veloce di eseguire una certa query.

Se la query è definita ogni volta e non viene memorizzata in nessuna struttura fisica o logica, la vista invece è un **oggetto creato dall'utente** e **memorizzato nel dizionario dei dati**.

Ciò che resta memorizzato è solo la definizione della query e non i dati che continuano a essere memorizzati nelle rispettive tabelle. Le tabelle interrogate da una vista sono chiamate **base tables**, cioè tabelle di base.

Creazione di una vista di sola lettura

Oltre ad usare le viste come query, possiamo sfruttarle per eseguire alcune operazioni **DML** sulle tabelle base (es. UPDATE).

Per evitare che chi ha accesso alle viste possa modificare i dati nelle tabelle base, possiamo creare viste di sola lettura usando la clausola **WITH READ ONLY** in CREATE VIEW.

Vista che non consente operazioni di DML:

```
CREATE VIEW readonly_view AS
SELECT * FROM padre
WITH READ ONLY;
```

Anche: http://new345.altervista.org/DB/Il_concetto_di_vista.pdf