Finanziato dall'Unione europea NextGenerationEU

Agenzia Nazionale Politiche Attive del Lavoro

REGIONE PIEMONTE

GOL Garanzia Occupabilità Lavoratori

DATABASE ORACLE
Corso Base
#2 SQL (Data Query Language)

IMMAGINAZIONE E LAVORO DAL 1978

COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

SOFTWARE UTILIZZATO

ORACLE XE DB

E’ il pacchetto di installazione del DBMS.

Il software è scaricabile dal sito della Oracle, previa autenticazione gratuita.

https://www.oracle.com/database/technologies/appdev/xe.html

ORACLE SQL DEVELOPER

E’ il client di accesso ufficiale.

Il software è scaricabile dal sito della Oracle, previa autenticazione gratuita.

https://www.oracle.com/tools/downloads/sqldev-downloads.html

APEX (Oracle Application Express)

E’ un framework per lo sviluppo di siti web con infrastruttura basata su Database Oracle.

Il software è scaricabile dal sito della Oracle, previa autenticazione gratuita.

https://www.oracle.com/tools/downloads/apex-downloads/

RISORSE DISPONIBILI

Ulteriori informazioni, spiegazioni e risorse possono essere ricavate direttamente dal repository di riferimento:

https://github.com/pmarconcini/DB_Oracle_Corso_Base/

DOCUMENTAZIONE UFFICIALE

Documentazione ufficiale generale Oracle Database

https://docs.oracle.com/en/database/oracle/oracle-database/index.html

Documentazione specifica della versione 23

https://docs.oracle.com/en/database/oracle/oracle-database/23/books.html

Per ogni ambito è disponibile la versione navigabile HTML e la versione PDF.

SQL e PL/SQL

https://docs.oracle.com/en/database/oracle/oracle-database/23/adfns/index.html#Oracle%C2%AE-Database

SQL*Plus® User's Guide and Reference

https://docs.oracle.com/en/database/oracle/oracle-database/23/sqpug/index.html#SQL*Plus%C2%AE

APEX - Getting started:

https://apex.oracle.com/en/learn/getting-started/

APEX - Documentazione ufficiale Oracle

https://docs.oracle.com/en-us/iaas/Content/services.htm

DQL – Clausole SELECT e FROM

La clausola SELECT stabilisce le regole dell'output ottenuto dall'interrogazione. La clausola FROM stabilisce l'origine dei dati.

In entrambi i casi le informazioni devono essere separate da virgola e possono essere identificate da un alias per evitare riferimenti ambigui nei nomi (che devono essere necessariamente unici).

Esempio:

```
1.      SELECT
2.       sysdate data,       /* ⇒ esposto valore non derivato dalla tabella */
3.       i.deptno,           /* ⇒ è necessario specificare l'alias di tabella perché
4.                  il nome di colonna è presente in entrambe le tabelle coinvolte */
5.       i.empno, /* ⇒ nome della colonna mantenuta, valore del campo non variato */
6.       i.ename Nome,  /* ⇒ alias di colonna considerato comunque maiuscolo, valore
7.                         del     campo non variato*/
8.       INITCAP(i.ename) "Nome Dip",  /* ⇒ alias di colonna come specificato tra
9.                            virgolette, valore del campo variato (funzione) */
10.      i.ename || i.empno,  /* ⇒ alias corrispondente al calcolo (da evitare perché
11.                           causa l'allargamento della colonna in output, valore
12.                           variato (concatenamento) */
13.      (i.sal * 14 + comm) * (1 - 0.40) netto_annuo,  /* ⇒ operazioni secondo le
14.                                                        regole dell'algebra */
15.      (select count(*) from emp) tot_dip, /* ⇒ valore ottenuto da una subquery */
16.       d.*          /* ⇒ l'asterisco indica di considerare tutti i campi della
17.                       tabella (o vista) */
18.      FROM emp i, dept d
19.      WHERE i.deptno = d.deptno AND i.job     = 'SALESMAN';
```

Output:

	DATA	DEPTNO	EMPNO	NOME	Nome Dip	I.ENAME||I.EMPNO	NETTO_ANNUO	TOT_DIP	DEPTNO_1	DNAME	LOC
1	21-FEB-20	30	7499	ALLEN	Allen	ALLEN7499	23989,8	14	30	SALES	CHICAGO
2	21-FEB-20	30	7844	TURNER	Turner	TURNER7844	22321,74	14	30	SALES	CHICAGO
3	21-FEB-20	30	7654	MARTIN	Martin	MARTIN7654	19441,38	14	30	SALES	CHICAGO
4	21-FEB-20	30	7521	WARD	Ward	WARD7521	18901,38	14	30	SALES	CHICAGO

DQL – Formato di visualizzazione

Oracle ha delle formattazioni predefinite per quanto riguarda alcuni tipi di dato, ma è possibile impostarne una diversa per la sessione attraverso, per esempio, le seguenti istruzioni (che valgono per SQL Plus e SQL Developer, mentre non è detto per altri client non "garantiti" dalla Oracle stessa):

- ALTER SESSION SET NLS_DATE_FORMAT='dd/mm/yyyy hh24:mi:ss'; ==> per impostare la formattazione a video delle date
- ALTER SESSION SET NLS_LANGUAGE ='english'; ==> per impostare la lingua nei riferimenti temporali (giorno della settimana e mese)
- ALTER SESSION SET NLS_TERRITORY='ITALY'; ==> per impostare la territorialità (per esempio per l'identificazione del primo giorno della settimana)
- ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ', '; ==> per impostare il carattere di separazione dei decimali e delle migliaia (questo secondo solo nell'utilizzo della funzione di conversione TO_CHAR)

Esempio:

```
1.      ALTER SESSION SET NLS_DATE_FORMAT='dd/mm/yyyy hh24:mi:ss';
2.      ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ', ';
3.      SELECT sysdate Oggi, 12345.9 numero, TO_CHAR(12345.9,'999G999D99') formattato
        FROM DUAL;
```

Output:

⇨ Session modificato.
⇨ Session modificato.

⇨ OGGI	NUMERO	FORMATTATO
07/12/2022 18:59:27	12345,9	12 345,90

DQL – Funzioni Single-Row #1

Sono funzioni che vengono elaborate per ogni singola riga (un set di dati in ingresso, un dato in uscita). Possono essere utilizzate anche le funzioni utente o quelle contenute nei package, se pubbliche.

Seguono esempi di utilizzo delle funzioni incontrate frequentemente.

FUNZIONI NUMERICHE che restituiscono un valore numerico:

```
1.      SELECT i.sal,
2.      ABS(i.sal) F1,              --valore assoluto
3.      MOD(i.sal, 600) F2,          --resto della divisione (modale)
4.      POWER(i.sal, 2) F3,          --elevamento a potenza (anche i.sal^2)
5.      ROUND(i.sal) F4,           --arrotondamento all'intero
6.      ROUND(i.sal, -2) F5,          --arrotondamento alle centinaia
7.      ROUND(i.sal, 2) F6,          --arrotondamento ai centesimi
8.      SIGN(i.sal) F7,           --segno (0:0, 1:positivo, -1:negativo)
9.      SQRT(i.sal) F8,           --radice quadrata
10.     TRUNC(i.sal) F9,           --troncamento all'intero
11.     TRUNC(i.sal, 2) F10,          --troncamento alle centinaia
12.     TRUNC(i.sal, -2) F11          --troncamento ai centesimi
13.     FROM emp i
14.     WHERE empno = 7698;
```

Output:

	SAL	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
1	5048,96	5048,96	248,96	25491997,0816	5049	5000	5048,96	1	71,056034226517314129085127378101142833 74	5048	5048,96	5000

DQL – Funzioni Single-Row #2

FUNZIONI TESTUALI che restituiscono un valore testuale:

```
1.      SELECT i.ename,
2.       CHR(50) F1,             --carattere corrispondente al dato codice ascii
3.       INITCAP(i.ename) F2,     --Iniziale delle parole maiuscola e resto minuscolo
4.       LOWER(i.ename) F3,         --tutto minuscolo
5.       LPAD(i.ename, 6, 'X') F4,     --riempimento (o taglio) con X a sinistra
6.                                        --per arrivare a 6 byte
7.       LTRIM(i.ename) F5,         --eliminazione dei blank a sinistra
8.       REPLACE(i.ename, 'IN', 'OR') F6, --sostituzione di ogni occorrenza IN con OR
9.       RPAD(i.ename, 6, 'X') F7,     --riempimento (o taglio) con X a destra
10.                                     --per arrivare a 6 byte
11.      RTRIM(i.ename) F8,         --eliminazione dei blank a destra
12.      SUBSTR(i.ename, 2, 3) F9,     --estrazione dal carattere 2 di 3 byte
13.      SUBSTR(i.ename, -2) F10,     --estrazione di 2 caratteri dal fondo
14.      TRANSLATE(i.ename, 'KANGI', 'CORQ') F11,--sostituzione di ogni carattere con
15.                              --il corrispondente per posizione (nullo, in assenza)
16.      TRIM(i.ename) F12,         --eliminazione dei blank a destra e sinistra
17.      UPPER(i.ename) F13         --tutto maiuscolo
18.     FROM emp i
19.     WHERE ename= 'KING';
```

Output:

	ENAME	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13
1	KING	2	King	king	XXKING	KING	KORG	KINGXX	KING	ING	NG	CRQ	KING	KING

DQL – Funzioni Single-Row #3

FUNZIONI TESTUALI che restituiscono un valore numerico:

```
1.      SELECT i.ename,
2.       ASCII(i.ename) F1,          --codice ascii del primo carattere del testo
3.       INSTR(i.ename, 'N') F2,   --posizione della prima occorrenza di N nel testo.
4.       INSTR(i.ename, 'Q') F3,     --0 in assenza di occorrenze di Q nel testo.
5.       INSTR('CANNONE', 'N', 5) F4,--posizione della prima occorrenza di N nel
6.                                    -- testo a partire dal byte 5
7.       INSTR('CANNONE', 'N', 1, 2) F5, --posizione della seconda occorrenza di N
8.                                      --nel testo a partire dal byte 1
9.       INSTR('CANNONE', 'N', -1) F6,--posiz. dell'ultima occorrenza di N nel testo
10.      LENGTH(i.ename) F7          --lunghezza del testo
11.     FROM emp i
12.     WHERE ename= 'KING';
```

Output:

ENAME	F1	F2	F3	F4	F5	F6	F7
KING	75	3	0	6	4	6	4

DQL – Funzioni Single-Row #4

FUNZIONI PER LE DATE che restituiscono varie tipologie di valori:

```
1.      SELECT i.hiredate,
2.          ADD_MONTHS(i.hiredate, -2) F1, --addizione o sottrazione del numero di
        mesi
3.          CURRENT_DATE F2,          --data corrente della sessione di Oracle
4.          CURRENT_TIMESTAMP F3,      --timestamp corrente della sessione di Oracle
5.          LAST_DAY(i.hiredate) F4,    --ultimo giorno del mese
6.          LOCALTIMESTAMP F5,          --timestamp locale
7.          MONTHS_BETWEEN(i.hiredate, sysdate) F6,    --numero di mesi tra due date
8.          ROUND(i.hiredate) F7,         --arrotondamento della data
9.          ROUND(i.hiredate + 0.6) F8, --arrotondamento della data
10.         SYSDATE F9,                  --data di sistema
11.         SYSTIMESTAMP F10,           --timestamp di sistema
12.         TO_CHAR(i.hiredate, 'DD/MM/RRRR HH24:MI:SS/ssss') F11,--format in testo
13.         TO_CHAR(i.hiredate, 'DAY D MONTH RR') F12,    --formattazione in testo
14.         TRUNC(i.hiredate) F13,         --troncamento della data
15.         EXTRACT(YEAR FROM i.hiredate) F14, --Estrazione dell'anno
16.         EXTRACT(MONTH FROM i.hiredate) F15, --Estrazione del mese
17.         EXTRACT(DAY FROM i.hiredate) F16 --Estrazione del giorno
18.     FROM emp i
19.     WHERE ename = 'KING';
```

Output:

HIREDATE	F1	F2	F3	F4	F5
17-NOV-81	17-SET-81	13-DIC-22	13-DIC-22 16:12:03,228000000 EUROPE/BERLIN	30-NOV-81	13-DIC-22 16:12:03,228000000

F6	F7	F8	F9	F10
-492,892743055555555555555555555555555556	17-NOV-81	18-NOV-81	13-DIC-22	13-DIC-22 16:12:03,228000000 +01:00

F11	F12	F13	F14	F15	F16
17/11/1981 00:00:00/0000	MARTEDÌ 2 NOVEMBRE 81	17-NOV-81	1981	11	17

DQL – Funzioni Single-Row #5

FUNZIONI VARIE (COMPARAZIONE, CONVERSIONE E ALTRO) che restituiscono varie tipologie di valori:

```
1.      SELECT i.sal,
2.       i.comm,
3.       i.job,
4.       NVL(i.comm, 1000) F1,                    --Sostituzione del nullo
5.       GREATEST(NVL(i.comm, 1000), i.sal, 500) F2,      --il maggiore
6.       LEAST(NVL(i.comm, 1000), i.sal, 500) F3,          --il minore
7.       DECODE(i.job, 'PRESIDENT', 'CAPO', 'MANAGER', 'CAPETTO', 'SCHIAVO') F4,
8.       --(deprecata) dato il primo parametro, verifiche per
9.       -- uguaglianza (test|valore) con elemento finale jolly facoltativo
10.      CASE i.job WHEN 'PRESIDENT' THEN 'CAPO' WHEN 'MANAGER' THEN 'CAPETTO'
11.       ELSE 'SCHIAVO' END F5,             --scrittura analoga alla DECODE
12.      CASE WHEN i.sal > 2000 THEN 'RICCO' ELSE 'POVERO' END F6,
13.      --CASE con espressione di verifica
14.      TO_DATE ('27092016', 'DDMMYYYY') F7,          --conversione in data
15.      TO_NUMBER('02,780') F8,
16.     --conversione in numero (il carattere predefinito
17.     -- per i decimali è il punto, ma dipende dalla configurazione del database)
18.     - formattare aggiungendo , '999999D99999', 'NLS_NUMERIC_CHARACTERS = '',.''')
19.      TO_CHAR (i.sal, 'U99G999D99MI') F9,          --formattazione dei numeri
20.      COALESCE(i.comm, NULL, NULL, 1000) F10,      --Sostituzione del nullo
21.                                          -- fino al primo non nullo dell'elenco
22.      TO_CLOB(i.ename) F11,                --conversione in clob
23.      NVL2(i.comm, 'Se NON nullo', 'Se nullo') F12      --verifica se nullo
24.     FROM emp i
25.     WHERE ENAME = 'KING';
```

Output:

SAL	COMM	JOB	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
8857,81	(null)	PRESIDENT	1000	8857,81	500	CAPO	CAPO	RICCO	27-SET-16	2,78	€8.857,81	1000	KING	Se nullo

DQL – Clausole WHERE e ORDER BY #1

La **clausola opzionale WHERE** ha il duplice scopo di filtrare i singoli record e di indicare la relazione tra tabelle.

A seguire gli aspetti da considerare:

- Le espressioni utilizzate a tale scopo sono tipicamente costituite da due valori intervallati da un operatore di confronto, ma sono comunque presenti alcune eccezioni.
- I valori possono essere dati presenti nel record, elaborazione degli stessi, costanti o risultati di subquery.
- Le espressioni sono logicamente legate tra loro dagli operatori AND e OR e dall'uso delle parentesi tonde secondo la logica applicata in matematica. In assenza di parentesi l'operatore AND è considerato prima dell'operatore OR.
- L'operatore NOT nega la verifica dell'espressione (attenzione, le espressioni "a = 1" e "NOT a = 1" NON sono complementari perchè entrambe escludono i casi in cui "a" è nullo).
- L'operatore di confronto IN è abitualmente utilizzato con elenchi definiti e statici di valori semplici (costituiti da una singola colonna), che non possono però eccedere una numericità ben definita (nell'attuale versione di Oracle si tratta di 1000 valori); l'eccezione generata è la ORA-01795.

NB: l'eccezione NON si verifica nel caso di utilizzo con una subquery. Gli operatori di confronto IN, ALL, ANY, SOME, EXISTS saranno affrontati nel paragrafo destinato alle subqueries. In tutti questi casi il confronto avviene con un elenco (anche vuoto) di dati.

DQL – Clausole WHERE e ORDER BY #2

La **clausola opzionale ORDER BY** permette di stabilire l'ordine di esposizione dei record elencando i criteri per priorità; i criteri seguono le stesse regole dell'esposizione nella clausola SELECT e quindi possono essere valori presenti in campi, elaborazioni di essi, subqueries, gli alias utilizzati o la posizione della colonne nella SELECT e, per ognuno, è possibile stabilire:

- la cardinalità aggiungendo la sigla ASC (valore predefinito e quindi in genere omesso) per l'ordine crescente o DESC per l'ordine decrescente, secondo la logica del dato stesso (alfabetico per i testi, dimensioni per i numeri e temporale per le date)
- il posizionamento dei campi nulli indicando NULLS FIRST o NULLS LAST (che è il valore predefinito e si può quindi omettere)

Nel caso in cui siano eseguiti dei raggruppamenti secondo le modalità specificate nei paragrafi seguenti, nella clausola ORDER BY possono essere specificati solo i raggruppamenti stessi (o loro elaborazioni).

DQL – Clausole WHERE e ORDER BY #3

IMMAGINAZIONE E LAVORO DAL 1978

Esempio:

```
1.      SELECT i.ename, i.deptno, i.sal, i.comm, i.sal * 12 + nvl(i.comm, 0) annuo,
2.      i.mgr, i.hiredate, d.loc
3.      FROM emp i, dept d
4.      WHERE i.deptno = d.deptno                    --condizione di JOIN
5.       AND (                    --l'uso delle parentesi rende alternative le coppie
6.                                 --di condizioni seguenti
7.       d.LOC = 'DALLAS' AND i.sal > 1000
8.       OR
9.       d.LOC LIKE '%O%' AND i.sal > 1250 --LIKE confronta testi con caratteri
10.                                        -- jolly % e _
11.      )
12.      AND mgr IS NOT NULL       --verifica del NULLO (alternativa è la scrittura
13.                                 --NVL(mgr, 0) = 0 per esempio
14.      AND i.sal + NVL(i.comm, 0) BETWEEN 1000 AND 3000--BETWEEN permette di
15.                       --verificare l'appartenenza di un valore rispetto a un range
16.      AND i.hiredate >= TO_DATE('01071981', 'ddmmyyyy') --è necessario convertire
17.                                                        -- in data
18.     ORDER BY annuo desc,          --riferimento all'alias di colonna
19.      1, --prima colonna della SELECT in ordine crescente (anche senza ASC)
20.      i.empno NULLS FIRST; --campo non usato in SELECT con NULLI prima
```

Output:

SAL	COMM	JOB	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
8857,81	(null)	PRESIDENT	1000	8857,81	500	CAPO	CAPO	RICCO	27-SET-16	2,78	€8.857,81	1000	KING	Se nullo

DQL – Clausole WHERE e ORDER BY #4

ESPRESSIONI REGOLARI: Oracle ha implementato l'utilizzo delle espressioni regolari secondo le specifiche POSIX, Extended Regular Expression (ERE) specification. In particolare REGEXP_LIKE è utilizzabile in maniera specifiche della clausola di filtro dei dati (WHERE).

Le funzioni disponibili sono le seguenti:

- **REGEXP_LIKE** (<espressione>, <pattern> [, <case sensitivity>])
- **REGEXP_REPLACE** (<espressione>, <pattern> [, <nuovo testo> [, <posizione iniziale> [, <iesima occorrenza> [, <case sensitivity>]]]])
- **REGEXP_INSTR** (<espressione>, <pattern> [, <posizione iniziale> [, <iesima occorrenza> [, <ritorno 0:match, 1:post match> [, <case sensitivity>]]]]])
- **REGEXP_SUBSTR** (<espressione>, <pattern> [, <posizione iniziale> [, <iesima occorrenza> [, <case sensitivity>]]]])

Esempio:

```
1.      SELECT ename,
2.      REGEXP_REPLACE (ename, '([aeiou])','1', 1, 1, 'i') F1,
3.       --sostituzione della prima occorrenza di una vocale con 1
4.       REGEXP_INSTR (ename, '[a-c]', 1, 2, 0, 'i') F2,
5.      --ricerca della posizi. della seconda occorrenza di una lettera tra a, b e c
6.       REGEXP_INSTR ('ABCD1234efgh', '([[:lower:]])', 1, 2, 0, 'c') F3,
7.      --ricerca la posiz. della seconda occorrenza di un caratt. Alfab. minuscolo
8.       REGEXP_SUBSTR('..Z-ABCD12-34567YYY', '([[:alnum:]]){3,4}', 1, 2) F4,
9.      --ricerca considerando solo gli alfanumerici
10.      REGEXP_SUBSTR('Testo1, Testo2, Testo3', '[^,]+', 1, 2, 'i') F5
11.     --selezione del testo
12.     FROM emp
13.     --WHERE REGEXP_LIKE (ename, 'AR')             --corrispondente a like %AR%
14.     --WHERE REGEXP_LIKE (ename, 'N$')             --che termina con il testo
15.     --WHERE REGEXP_LIKE (ename, '^K')             --che inizia con il testo
16.     --WHERE REGEXP_LIKE (ename, '(AN|EN|IN|ON)')
17.     --che contiente uno dei valori alternativi
18.     --WHERE REGEXP_LIKE (ename, '[AEIOU]N')
19.     --che contiente uno dei valori alternativi seguito da N
20.     --WHERE REGEXP_LIKE (ename, '([cln])\1', 'i')
21.     --che contiente uno doppia C/N/L, non case sensitive
22.     --WHERE REGEXP_LIKE (ename, '[cln]{2}', 'i')
23.     --che contiente uno doppia C/N/L, non case sensitive
24.     WHERE REGEXP_LIKE (ename, '[AE]')
25.     ;
```

Output:

ENAME	F1	F2	F3	F4	F5
BLAKE	BL1KE	3	10	3456	Testo2
CLARK	CL1RK	3	10	3456	Testo2
JONES	J1NES	0	10	3456	Testo2
ALLEN	1LLEN	0	10	3456	Testo2
WARD	W1RD	0	10	3456	Testo2
MARTIN	M1RTIN	0	10	3456	Testo2
TURNER	T1RNER	0	10	3456	Testo2
ADAMS	1DAMS	3	10	3456	Testo2
JAMES	J1MES	0	10	3456	Testo2
MILLER	M1LLER	0	10	3456	Testo2

DQL – Clausole WHERE e ORDER BY #5

IMMAGINAZIONE E LAVORO DAL 1978

Clausola di **LIMITAZIONE DELLE RIGHE**: dalle ultime versioni di Oracle (12.1 e seguenti) è possibile limitare il numero di righe selezionate utilizzando una apposita clausola applicata all'ordinamento (TOP-N Queries).

Esempio:

```
1.      SELECT ename, sal
2.      FROM   emp
3.      ORDER BY sal DESC
4.      --FETCH FIRST 11 ROWS ONLY --prime 11 righe
5.      --FETCH FIRST 2 ROWS WITH TIES --prime 2 righe più eventuali pareggi
6.      --FETCH FIRST 20 PERCENT ROWS ONLY; --primo 20% dei record
7.      --OFFSET 4 ROWS FETCH NEXT 5 ROWS ONLY; --dopo i primi 4 record, 5 record
8.      ;
```

Output:

	ENAME	SAL
1	KING	8857,81
2	FORD	5314,68
3	SCOTT	5314,68
4	JONES	5270,4
5	BLAKE	5048,96
6	CLARK	4340,34
7	ALLEN	2834,5
8	TURNER	2657,35
9	MILLER	2303,03
10	WARD	2214,45
11	MARTIN	2214,45
12	ADAMS	1948,72
13	SMITH	1714,88
14	JAMES	1682,99

Nelle versioni precedenti di Oracle l'unica possibilità è utilizzare una query di selezione e ordinamento innestata su cui poi applicare il filtro di paginazione sul ROWNUMBER prodotto.

Esempio:

```
1.      SELECT x2.*
2.      FROM (
3.          SELECT ROWNUM ord, x1.*
4.          FROM (
5.                  SELECT rownum n, ename, sal
6.                  FROM   emp
7.                  ORDER BY sal DESC
8.                ) x1
9.          ) x2
10.     WHERE ord > 7
11.     ;
```

Output:

	ORD	N	ENAME	SAL
1	8	11	TURNER	2657,35
2	9	14	MILLER	2303,03
3	10	9	WARD	2214,45
4	11	10	MARTIN	2214,45
5	12	12	ADAMS	1948,72
6	13	7	SMITH	1714,88
7	14	13	JAMES	1682,99

DQL – Clausole GROUP BY e HAVING

Tramite la clausola opzionale **GROUP BY è possibile aggregare record**, al solito secondo il valore presente nei campi o elaborazioni dello stesso, stabilendo l'ordine logico di aggregazione.
In presenza di raggruppamenti nella clausola SELECT è possibile fare riferimento esclusivamente agli stessi criteri di aggregazione utilizzati nella GROUP BY e/o a eventuali valori ottenuti da funzioni aggregate.
Discorso analogo per la clausola ORDER BY, ma senza vincoli nell'ordine di utilizzo.

La clausola opzionale **HAVING** può essere presente solo in presenza della clausola GROUP BY ed è una clausola di filtro dei raggruppamenti; ne consegue che in essa si possono usare esclusivamente funzioni aggregate applicate ai criteri di raggruppamento e gli operatori logici e di confronto già visti nel caso della clausola WHERE.

E' possibile utilizzare le funzioni aggregate anche senza raggruppamenti (quindi senza la clausola GROUP BY) purché nella SELECT siano esposti solo dati aggregati.

Esempi:

```
1.      SELECT d.loc,
2.       CASE i.job WHEN 'PRESIDENT' THEN 'Capi'
3.                  WHEN 'MANAGER' THEN 'Capi' ELSE 'Schiavi' END ruolo,
4.       AVG (i.sal) F1,          --media del gruppo
5.       COUNT (d.loc) F2,          --conteggio dei record con valore per gruppo
6.       COUNT (*) F3,          --conteggio dei record per gruppo
7.       COUNT (i.sal) F4,          --conteggio dei record con valore per gruppo
8.       MAX (i.sal) F5,          --massimo
9.       MIN (i.sal) F6,          --minimo
10.      SUM (i.sal) F7          --sommatoria
11.     FROM emp i, dept d
12.     WHERE i.deptno (+)= d.deptno
13.     --outer join per forzare l'utilizzo dei record di dept senza emp collegati
14.     GROUP BY d.loc,
15.              CASE i.job WHEN 'PRESIDENT' THEN 'Capi'
16.                         WHEN 'MANAGER' THEN 'Capi' ELSE 'Schiavi' END
17.     HAVING MAX (NVL(i.sal,0)) < 5000 and COUNT(i.sal) BETWEEN 0 AND 4
18.     ORDER BY ruolo, AVG(i.sal) DESC, d.loc
```

Output:

LOC	RUOLO	F1	F2	F3	F4	F5	F6	F7
BOSTON	Schiavi	(null)	1	1	0	(null)	(null)	(null)
NEW YORK	Schiavi	2303,03	1	1	1	2303,03	2303,03	2303,03

```
1.      SELECT
2.          COUNT(*) F1,                --conteggio generale (record)
3.          COUNT(DISTINCT d.loc) F2,
4.           --la parola chiave DISTINCT permette di ignorare i valori ripetuti
5.           -- ma non può essere utilizzata in presenza di dati di tipo CLOB
6.          COUNT(d.loc) F3,
7.          --conteggio dei record con valore non nullo di un campo
8.          AVG (i.sal) F4              --media generale
9.      FROM emp i, dept d
10.     WHERE i.deptno (+)= d.deptno --outer join per forzare l'utilizzo dei record
        di dept senza emp collegati
11.     ;
```

F1	F2	F3	F4
15	4	15	3694,0885714285714285714285714285714285714285

DQL – JOINS #1

Per **JOIN** si intende una query che collega i dati di due o più tabelle; in particolare, oltre alle tabelle che utilizziamo nelle clausole SELECT, WHERE, GROUP BY, HAVING e ORDER BY, è **necessario "coinvolgere" tutte le tabelle intermedie** che le collegano: l'effetto è che **saranno necessarie un numero di join minore di uno rispetto al numero di tabelle considerate**.

Nelle ultime versioni Oracle si è adeguata allo standard ANSI e quindi è possibile utilizzare la clausola ON per specificare la relazione.

Per la "**SIMPLE (o INNER) JOIN**" il collegamento specificando tutti i campi coinvolti nella clausola WHERE collegando logicamente le espressioni necessarie tramite l'operatore logico AND.
Per esempio, date due tabelle T1 e T2 in relazione tramite i campi A e B (NB: non necessariamente i campi devono avere lo stesso nome sulle tabelle coinvolte), il risultato sarà la presenza nella clausola WHERE della doppia espressione T1.A = T2.A AND T1.B = T2.B al netto di eventuali condizioni di filtro.
Record non coinvolti dalla relazione (dipartimenti senza dipendenti, nel db di esempio) saranno esclusi dalla query.
NB: nel caso della simple join con standard ANSI la parola chiave INNER si può omettere.

Esempio standard Oracle:

```
1.      SELECT count(*) k
2.      --conta 14 record, cioè gli impiegati associati a un dipartimento
3.      FROM emp i, dept d
4.      WHERE i.deptno = d.deptno;
```

Esempio standard ANSI:

```
1.      SELECT COUNT(*) k
2.      --conta 14 record, cioè gli impiegati associati a un dipartimento
3.      FROM dept d INNER JOIN emp i
4.      ON i.deptno = d.deptno;
```

DQL – JOINS #2

La “**NATURAL JOIN**” è una join in cui la relazione tra i campi della tabella è automaticamente interpretata da Oracle in base a nome e tipo di dato.

Esempio:

```
1.      SELECT COUNT(*) k
2.          --conta 14 record, cioè gli impiegati associati a un dipartimento
3.      FROM dept d NATURAL JOIN emp i;
4.          --per la join viene automaticamente considerato l'unico campo
5.          --presente in entrambe le tabelle: DEPTNO
```

L’ “**OUTER JOIN**” è una join in cui viene forzata la restituzione dei record di una tabella NON collegati all’altra. Il nome **LEFT o RIGHT** deriva dalla posizione della tabella forzata nella clausola FROM, cioè quella in cui manca il record per completare la join. La parola chiave OUTER si può omettere.

Esempio RIGHT JOIN standard Oracle:

```
1.      SELECT COUNT(*) k
2.          --conta 15 record, cioè i 14 impiegati associati e il dipartimento vuoto
3.      FROM emp i, dept d
4.      WHERE i.deptno (+) = d.deptno;
```

Esempio RIGHT JOIN standard ANSI:

```
1.      SELECT COUNT(*) k
2.      --conta 15 record, cioè i 14 impiegati associati e il dipartimento vuoto
3.      FROM dept d RIGHT JOIN emp i
4.      ON i.deptno = d.deptno;
```

Esempio LEFT JOIN standard ANSI:

```
1.      SELECT COUNT(*) k
2.          --conta 15 record, cioè i 14 impiegati associati e il dipartimento vuoto
3.      FROM emp d LEFT JOIN dept i
4.      ON i.deptno = d.deptno;
```

In alcuni DBMS esiste, secondo standard ANSI, anche la FULL OUTER JOIN che forza la restituzione dei record non legati di entrambe le tabelle.

DQL – JOINS #3

La "**SELF JOIN**" è una join che mette in relazione uno o più campi di una tabella con uno o più campi della tabella stessa, come nell'esempio seguente. Di fatto è una INNER o OUTER JOIN tra una tabella e la replica di se stessa (tramite alias)

Esempio:

```
1.      SELECT i.ename NOME, i.empno MATRICOLA, i2.ename NOME_SUP,
2.             i2.empno MATRICOLA_SUP
3.      FROM emp i, emp i2
4.      WHERE i.mgr = i2.empno   (+);
```

Output:

NOME	MATRICOLA	NOME_SUP	MATRICOLA_SUP
BLAKE	7698	KING	7839
CLARK	7782	KING	7839
JONES	7566	KING	7839
ALLEN	7499	BLAKE	7698
WARD	7521	BLAKE	7698
MARTIN	7654	BLAKE	7698
TURNER	7844	BLAKE	7698
JAMES	7900	BLAKE	7698
MILLER	7934	CLARK	7782
SCOTT	7788	JONES	7566
FORD	7902	JONES	7566
ADAMS	7876	SCOTT	7788
SMITH	7369	FORD	7902
KING	7839	(null)	(null)

Il PRODOTTO CARTESIANO è considerato una NON-JOIN (o NON EQUIJOIN, in contrasto con le EQUIJOIN che presentano una uguaglianza di confronto) perchè non esiste una relazione di uguaglianza tra i campi delle due tabelle, pur essendoci un legame logico (spesso esplicato nella clausola WHERE). All'atto pratico il prodotto cartesiano produce l'incrocio di tutti i record delle tabelle coinvolte (nel database di esempio 14 impiegati per 5 dipartimenti per un totale di 70 record). L'esempio classico di utilizzo corretto è l'incrocio tra salari e livelli salariali come da codice seguente.

Esempio:

```
1.      SELECT i.ename, i.sal, s.losal, s.hisal, s.grade
2.      FROM emp i, salgrade s
3.      WHERE i.sal BETWEEN s.losal AND s.hisal;
```

Output:

ENAME	SAL	LOSAL	HISAL	GRADE
JAMES	1682,99	1000,01	2000	2
SMITH	1714,88	1000,01	2000	2
ADAMS	1948,72	1000,01	2000	2
MARTIN	2214,45	2000,01	3000	3
WARD	2214,45	2000,01	3000	3
MILLER	2303,03	2000,01	3000	3
TURNER	2657,35	2000,01	3000	3
ALLEN	2834,5	2000,01	3000	3
CLARK	4340,34	4000,01	5000	5
BLAKE	5048,96	5000,01	6000	6
JONES	5270,4	5000,01	6000	6
FORD	5314,68	5000,01	6000	6
SCOTT	5314,68	5000,01	6000	6
KING	8857,81	8000,01	9000	9

La query produce correttamente 14 record (gli impiegati), purchè gli scaglioni corrispondenti ai livelli siano continui e non sovrapposti.

DQL – DESCRIBE

Tramite l'istruzione **DESC <oggetto>** è possibile ottenere le relative informazioni. L'istruzione espone informazioni diverse a seconda del client utilizzato.

DESC dept;

Output in APEX:

Object Type TABLE Object DEPT

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
DEPT	DEPTNO	NUMBER	-	2	0	1	-	-	-
	DNAME	VARCHAR2	14	-	-	-	✓	-	-
	LOC	VARCHAR2	13	-	-	-	✓	-	-

Output in Oracle SQL Developer:

```
Nome    Nullo?   Tipo
------  --------  ------------
DEPTNO  NOT NULL  NUMBER(2)
DNAME             VARCHAR2(14)
LOC               VARCHAR2(13)
```

DESC emp;

Output in APEX:

Object Type TABLE Object EMP

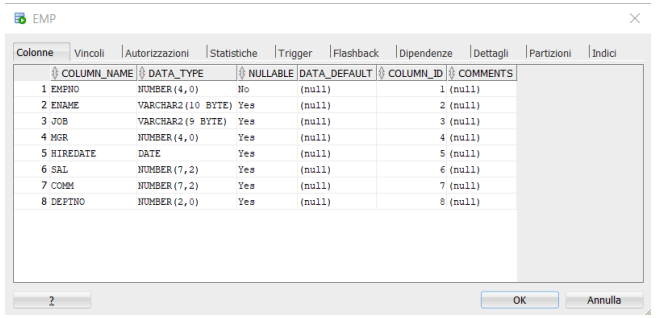
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	EMPNO	NUMBER	-	4	0	1	-	-	-
	ENAME	VARCHAR2	10	-	-	-	✓	-	-
	JOB	VARCHAR2	9	-	-	-	✓	-	-
	MGR	NUMBER	-	4	0	-	✓	-	-
	HIREDATE	DATE	7	-	-	-	✓	-	-
	SAL	NUMBER	-	7	2	-	✓	-	-
	COMM	NUMBER	-	7	2	-	✓	-	-
	DEPTNO	NUMBER	-	2	0	-	✓	-	-

Output in Oracle SQL Developer:

```
Nome      Nullo?   Tipo
--------  --------  ------------
EMPNO     NOT NULL  NUMBER(4)
ENAME               VARCHAR2(10)
JOB                 VARCHAR2(9)
MGR                 NUMBER(4)
HIREDATE            DATE
SAL                 NUMBER(7,2)
COMM                NUMBER(7,2)
DEPTNO              NUMBER(2)
```

Ogni software dotato di interfaccia grafica ha, generalmente, una combinazione di tasti per ottenere a video tutte le informazioni legate a un oggetto.

Nell'esempio a lato la finestra PopUp visualizzata in Oracle SQL Developer dopo aver posizionato il cursore in prossimità del nome dell'oggetto e premuto la combinazione di tasti Shift + F4.



DQL – DUAL e PSEUDOCOLONNE

La tabella fittizia **DUAL è** una tabella non esistente monoriga e senza colonne definite. In alcuni contesti è estremamente utile (v. esempio relativo all'operatore EXISTS nel paragrafo delle subqueries)

Esempio:

```
1.      SELECT SYSDATE C1, CURRENT_TIMESTAMP C2, POWER(10, 2) C3, 10 * 10 C4, USER C5
2.      FROM DUAL;
```

Output:

C1	C2	C3	C4	C5
25-FEB-20	25-FEB-20 11:27:39,571000000 EUROPE/BERLIN	100	100	SCOTT

Le **PSEUDOCOLONNE** sono colonne fittizie i cui dati sono ottenuti da funzioni di sistema e sono coerenti con il singolo record. Le più frequentemente utilizzate sono:

- LEVEL: definisce la distanza dalla radice nelle queries gerarchiche (v. successivo)
- SEQUENCE: utilizza l'oggetto contatore (v. relativo capitolo)
- ROWNUM: indica l'ordine di selezione dei record (attenzione, non l'ordine di esposizione!)
- ROWID: indica l'indirizzo del record

Esempio:

```
1.      SELECT ROWNUM, e.ROWID, e.ename, d.ROWID, d.loc
2.      FROM emp e, dept d
3.      WHERE e.deptno = d.deptno
4.      ORDER BY ENAME;
```

Output:

ROWNUM	ROWID	ENAME	ROWID_1	LOC
8	AAAR/dAAHAAAAF1AAL	ADAMS	AAAR/TAAHAAAAFVAAB	DALLAS
13	AAAR/dAAHAAAAF1AAH	ALLEN	AAAR/TAAHAAAAFVAAC	CHICAGO
14	AAAR/dAAHAAAAF1AAB	BLAKE	AAAR/TAAHAAAAFVAAC	CHICAGO
1	AAAR/dAAHAAAAF1AAC	CLARK	AAAR/TAAHAAAAFVAAA	NEW YORK
4	AAAR/dAAHAAAAF1AAF	FORD	AAAR/TAAHAAAAFVAAB	DALLAS
12	AAAR/dAAHAAAAF1AAM	JAMES	AAAR/TAAHAAAAFVAAC	CHICAGO
6	AAAR/dAAHAAAAF1AAD	JONES	AAAR/TAAHAAAAFVAAB	DALLAS
3	AAAR/dAAHAAAAF1AAA	KING	AAAR/TAAHAAAAFVAAA	NEW YORK
10	AAAR/dAAHAAAAF1AAJ	MARTIN	AAAR/TAAHAAAAFVAAC	CHICAGO
2	AAAR/dAAHAAAAF1AAN	MILLER	AAAR/TAAHAAAAFVAAA	NEW YORK
5	AAAR/dAAHAAAAF1AAE	SCOTT	AAAR/TAAHAAAAFVAAB	DALLAS
7	AAAR/dAAHAAAAF1AAG	SMITH	AAAR/TAAHAAAAFVAAB	DALLAS
11	AAAR/dAAHAAAAF1AAK	TURNER	AAAR/TAAHAAAAFVAAC	CHICAGO
9	AAAR/dAAHAAAAF1AAI	WARD	AAAR/TAAHAAAAFVAAC	CHICAGO

Il valore del campo ROWID è unico e specifico per ogni singolo record del database e può essere utilizzato anche come criterio di ricerca.

DQL – Queries gerarchiche

Una forma più evoluta di self join è quella utilizzata per definire le queries gerarchiche; in particolare è possibile stabilire sia il punto di partenza (**la radice, tramite START WITH**) della gerarchia che la **regola di connessione tra righe (CONNECT BY)**. Sono disponibili delle funzioni peculiari, alcune delle quali visibili nella query di esempio seguente (tra quelle non utilizzate può essere utile la funzione CONNECT_BY_ROOT <campo> che restituisce il valore della radice per ogni riga (nel caso dell'esempio sarebbe sempre un dato della riga relativa a KING, essendoci una unica radice).

Esempio:

```
1.      SELECT ename Impiegato, empno, mgr,  CONNECT_BY_ISLEAF Foglia,
2.      LEVEL, SYS_CONNECT_BY_PATH(ename, '/') "Path"
3.      FROM emp
4.      --WHERE LEVEL <= 5 and deptno = 10
5.      START WITH empno = 7839
6.      CONNECT BY PRIOR empno = mgr AND LEVEL <= 5 and deptno = 10
7.      --ORDER BY LEVEL, Impiegato, Foglia;
```

Output:

IMPIEGATO	EMPNO	MGR	FOGLIA	LEVEL	Path
KING	7839	(null)	0	1	/KING
CLARK	7782	7839	0	2	/KING/CLARK
MILLER	7934	7782	1	3	/KING/CLARK/MILLER

Nella clausola ORDER BY è possibile inserire la parola chiave SIBLINGS che fa sì che i criteri seguenti siano considerati a parità di livello all'interno del singolo ramo, come nell'esempio seguente:

Esempio:

```
1.      SELECT ename Impiegato, empno, mgr,  CONNECT_BY_ISLEAF Foglia,
2.      LEVEL, SYS_CONNECT_BY_PATH(ename, '/') "Path"
3.      FROM emp
4.      START WITH mgr IS NULL
5.      CONNECT BY PRIOR empno = mgr
6.         ORDER SIBLINGS BY ename;
```

Output:

```
IMPIEGATO  EMPNO MGR FOGLIA LEVEL      PATH
KING       7839          0     1      /KING
BLAKE      7698 7839     0     2      /KING/BLAKE
ALLEN      7499 7698     1     3      /KING/BLAKE/ALLEN
JAMES      7900 7698     1     3      /KING/BLAKE/JAMES
MARTIN     7654 7698     1     3      /KING/BLAKE/MARTIN
TURNER     7844 7698     1     3      /KING/BLAKE/TURNER
WARD       7521 7698     1     3      /KING/BLAKE/WARD
CLARK      7782 7839     0     2      /KING/CLARK
MILLER     7934 7782     1     3      /KING/CLARK/MILLER
[…]
```

DQL – Operatori SET

Sono operatori che permettono di combinare il risultato di 2 o più queries; le colonne delle singole queries devono coincidere per numero, ordine e tipo di dato. L'alias di colonna proposto è quello utilizzato nella prima query.
Gli operatori utilizzabili sono:

- UNION: i record presenti nella prima e nella seconda, NON ripetuti se presenti in entrambe
- UNION ALL: i record presenti nella prima e nella seconda, ripetuti se presenti in entrambe
- MINUS: i record della prima query assenti nella seconda
- INTERSECT: i record comuni ad entrambe le query

Esempio:

```
1.      SELECT ename impiegato, empno matricola, deptno sede
2.      FROM emp
3.      WHERE deptno = 20
4.      UNION ALL
5.      SELECT ename, empno, deptno
6.      FROM emp
7.      WHERE ename like 'J%';
```

Output:

IMPIEGATO	MATRICOLA	SEDE
JONES	7566	20
SCOTT	7788	20
FORD	7902	20
SMITH	7369	20
ADAMS	7876	20
JONES	7566	20
JAMES	7900	30

DQL – Subqueries e operatori

Una subquery è una query annidata all'interno di un'altra query, detta parent (tale rapporto gerarchico può essere riproposto per più livelli); **deve essere racchiusa tra parentesi tonde** e può essere utilizzata **in tutte le clausole** e, in caso di necessità, può fare **riferimento ai campi della query parent** specificandone gli alias di tabella. Con gli stessi criteri le subqueries possono essere utilizzate **anche nelle DML di manipolazione dati**.

Alcuni operatori utilizzabili nella clausola WHERE utilizzano subqueries:

- **IN** (verificata se il valore è uno di quelli dell'elenco; non soggetto alla limitazione dei 1000 valori che genera l'eccezione ORA-01795)
- **<operatore di confronto> ANY** (condizione rispettata rispetto ad almeno uno dei valori)
- **EXISTS** (esistenza di almeno un valore)
- **<operatore di confronto> ALL** (condizione rispettata rispetto a tutti i valori)

NB: SOME ha esattamente lo stesso utilizzo di ANY e restituisce lo stesso risultato. Di fatto sono interscambiabili.

Il valore restituito da una subquery per ogni record può essere **sia un dato semplice** (quindi una unica colonna) **che complesso** (quindi sottoforma di riga); in questo secondo caso esistono dei limiti nella modalità di utilizzo:

- Nella clausola SELECT è necessario convertire i dati tramite l'utilizzo di oggetti TYPE (v. il paragrafo ad essi dedicato)
- Nella condizione IN le due parti del confronto devono avere lo stesso numero di colonne, corrispondenti per tipo di dato
- Non è possibile utilizzare dati complessi con gli operatori ANY, ALL e SOME.

DQL – Subqueries e operatori

Una subquery è una query annidata all'interno di un'altra query, detta parent (tale rapporto gerarchico può essere riproposto per più livelli); **deve essere racchiusa tra parentesi tonde** e può essere utilizzata **in tutte le clausole** e, in caso di necessità, può fare **riferimento ai campi della query parent** specificandone gli alias di tabella. Con gli stessi criteri le subqueries possono essere utilizzate **anche nelle DML di manipolazione dati**.

Alcuni operatori utilizzabili nella clausola WHERE utilizzano subqueries:

- **IN** (verificata se il valore è uno di quelli dell'elenco; non soggetto alla limitazione dei 1000 valori che genera l'eccezione ORA-01795)
- **<operatore di confronto> ANY** (condizione rispettata rispetto ad almeno uno dei valori)
- **EXISTS** (esistenza di almeno un valore)
- **<operatore di confronto> ALL** (condizione rispettata rispetto a tutti i valori)

NB: SOME ha esattamente lo stesso utilizzo di ANY e restituisce lo stesso risultato. Di fatto sono interscambiabili.

Il valore restituito da una subquery per ogni record può essere **sia un dato semplice** (quindi una unica colonna) **che complesso** (quindi sottoforma di riga); in questo secondo caso esistono dei limiti nella modalità di utilizzo:

- Nella clausola SELECT è necessario convertire i dati tramite l'utilizzo di oggetti TYPE (v. il paragrafo ad essi dedicato)
- Nella condizione IN le due parti del confronto devono avere lo stesso numero di colonne, corrispondenti per tipo di dato
- Non è possibile utilizzare dati complessi con gli operatori ANY, ALL e SOME.

DQL – Subqueries con IN, ESXIST e ALL

IMMAGINAZIONE E LAVORO DAL 1978

Esempio operatore IN:

```
1.      SELECT empno, ename, mgr,
2.      --non è necessario l'alias perchè non c'è ambiguità con le subqueries
3.      (SELECT ename FROM emp WHERE empno = e.mgr) MANAGER,
4.      --nella condizione si fa riferimento all'alias della parent
5.      sal - (SELECT AVG(sal) FROM emp) DELTA_SAL_MEDIO,
6.      --nessuna neccessità di legarsi alla parent
7.      sal - (SELECT AVG(sal) FROM emp WHERE deptno = e.deptno) DELTA_SAL_MEDIO_SEDE
8.      --neccessità di legarsi alla parent
9.      FROM emp e
10.     WHERE deptno IN (SELECT deptno FROM dept WHERE loc IN ('NEW YORK', 'DALLAS'))
11.     order by DELTA_SAL_MEDIO DESC;
```

Output:

EMPNO	ENAME	MGR	MANAGER	DELTA_SAL_MEDIO	DELTA_SAL_MEDIO_SEDE
7839	KING	(null)	(null)	5163,721428571428571428571428571428571429	3690,75
7902	FORD	7566	JONES	1620,591428571428571428571428571428571429	1402,008
7788	SCOTT	7566	JONES	1620,591428571428571428571428571428571429	1402,008
7566	JONES	7839	KING	1576,311428571428571428571428571428571429	1357,728
7782	CLARK	7839	KING	646,251428571428571428571428571428571429	-826,72
7934	MILLER	7782	CLARK	-1391,058571428571428571428571428571428571	-2864,03
7876	ADAMS	7788	SCOTT	-1745,368571428571428571428571428571428571	-1963,952
7369	SMITH	7902	FORD	-1979,208571428571428571428571428571428571	-2197,792

Esempio operatore EXISTS:

```
1.      SELECT deptno, count(*) k, AVG (sal) media,
2.             SUM(sal) somma, MAX(sal) massimo, MIN(sal) minimo
3.      FROM emp x
4.      WHERE EXISTS (SELECT 1 FROM emp WHERE x.deptno = deptno
5.            AND job IN ('SALESMAN', 'ANALYST'))
6.      GROUP BY deptno
```

Output:

DEPTNO	K	MEDIA	MASSIMO	MINIMO
30	6	2775,45	16652,7	1682,99
20	5	3912,672	19563,36	1714,88

Esempio operatore EXISTS:

```
1.      SELECT ename, job, sal
2.      FROM emp e
3.      WHERE sal > ALL (SELECT avg(sal)  FROM emp sq WHERE sq.job = e.job
4.                       UNION
5.                       SELECT avg(sal)  FROM emp sq WHERE sq.deptno = e.deptno
6.                       );
```

Output:

ENAME	JOB	SAL
BLAKE	MANAGER	5048,96
JONES	MANAGER	5270,4
ALLEN	SALESMAN	2834,5

DQL – Subqueries con ANY e in FROM

Esempio operatore ANY:

```
1.      SELECT ename, job, sal
2.      FROM emp e
3.      WHERE sal > ANY (SELECT avg(sal)  FROM emp sq WHERE sq.job = e.job
4.                       UNION
5.                       SELECT avg(sal)  FROM emp sq WHERE sq.deptno = e.deptno
6.                       );
```

Output:

ENAME	JOB	SAL
KING	PRESIDENT	8857,81
BLAKE	MANAGER	5048,96
JONES	MANAGER	5270,4
SCOTT	ANALYST	5314,68
FORD	ANALYST	5314,68
ALLEN	SALESMAN	2834,5
TURNER	SALESMAN	2657,35
ADAMS	CLERK	1948,72
MILLER	CLERK	2303,03

Esempio di utilizzo nella clausola FROM:

```
1.      SELECT empno, ename, e.deptno, sal, media, sal - media delta_sal
2.      FROM emp e, (SELECT deptno, AVG (sal) media FROM emp GROUP BY deptno) sq1
3.      WHERE e.deptno = sq1.deptno
4.      ORDER BY delta_sal DESC;
```

Output:

EMPNO	ENAME	DEPTNO	SAL	MEDIA	DELTA_SAL
7839	KING	10	8857,81	5167,06	3690,75
7698	BLAKE	30	5048,96	2775,45	2273,51
7902	FORD	20	5314,68	3912,672	1402,008
7788	SCOTT	20	5314,68	3912,672	1402,008
7566	JONES	20	5270,4	3912,672	1357,728
7499	ALLEN	30	2834,5	2775,45	59,05
7844	TURNER	30	2657,35	2775,45	-118,1
7521	WARD	30	2214,45	2775,45	-561
7654	MARTIN	30	2214,45	2775,45	-561
7782	CLARK	10	4340,34	5167,06	-826,72
7900	JAMES	30	1682,99	2775,45	-1092,46
7876	ADAMS	20	1948,72	3912,672	-1963,952
7369	SMITH	20	1714,88	3912,672	-2197,792
7934	MILLER	10	2303,03	5167,06	-2864,03

DQL – Inline View

Una particolare forma di subquery è la Inline View, in cui il recordset ottenuto dalla query è oggetto a sua volta di query e si trova, sempre racchiuso tra parentesi e identificato tramite alias, nella clausola FROM.

Nell'esempio seguente la query più interna serve per ordinare i dati, la seconda per associare l'ordinale grazie al ROWNUM e la terza per filtrare (è il meccanismo utilizzato per ottenere un set posizionato di record prima della versione 12.1 di Oracle).

Esempio:

```
1.      SELECT *
2.      FROM (
3.          SELECT ename, sal, rownum ordine
4.          FROM (
5.                  SELECT ename, sal
6.                  FROM emp
7.                  ORDER BY sal DESC
8.                  ) iv
9.           ) iv_ordinata
10.     WHERE ordine BETWEEN 4 and 11;
```

Output:

ENAME	SAL	ORDINE
JONES	5270,4	4
BLAKE	5048,96	5
CLARK	4340,34	6
ALLEN	2834,5	7
TURNER	2657,35	8
MILLER	2303,03	9
WARD	2214,45	10
MARTIN	2214,45	11

DQL – Clausola WITH

Una alternativa alla vista inline è l'utilizzo della **clausola WITH** in cui si deve definire **prima della clausola SELECT** la query necessaria per stabilire i dati che faranno da "bacino" nella DML.

E' preferibile all'utilizzo della vista inline soprattutto nei casi in cui sia necessario accedere più volte al contenuto di una data tabella

Esempio:

```
1.      WITH dept_medie AS (
2.        SELECT deptno, AVG(sal) sal_medio
3.        FROM   emp
4.        GROUP BY deptno)
5.      SELECT e.ename,       e.sal,       e.deptno,
6.             dm.sal_medio,       dm.sal_medio - e.sal sal_diff
7.      FROM   emp e,       dept_medie dm
8.      WHERE  e.deptno = dm.deptno
9.      ORDER BY e.ename;
```

Output:

ENAME	SAL	DEPTNO	SAL_MEDIO	SAL_DIFF
ADAMS	1948,72	20	3912,672	1963,952
ALLEN	2834,5	30	2775,45	-59,05
BLAKE	5048,96	30	2775,45	-2273,51
CLARK	4340,34	10	5167,06	826,72
FORD	5314,68	20	3912,672	-1402,008
JAMES	1682,99	30	2775,45	1092,46
JONES	5270,4	20	3912,672	-1357,728
KING	8857,81	10	5167,06	-3690,75
MARTIN	2214,45	30	2775,45	561
MILLER	2303,03	10	5167,06	2864,03
SCOTT	5314,68	20	3912,672	-1402,008
SMITH	1714,88	20	3912,672	2197,792
TURNER	2657,35	30	2775,45	118,1
WARD	2214,45	30	2775,45	561

DQL – Data Dictionary

E' possibile avere **informazioni sulle caratteristiche e sull'utilizzo degli oggetti del database** disponibili tramite delle viste di sistema; la "visibilità" delle informazioni dipende dai privilegi garantiti all'utente che esegue la ricerca. Il prefisso delle tabelle chiarisce (generalmente) l'ambito della ricerca: **USER_ per gli oggetti di proprietà dell'utente**, **ALL_ per gli oggetti su cui l'utente ha privilegi anche se non di sua proprietà**, **DBA_ per tutti gli oggetti presenti nel database** (sono tabelle ad accesso limitato), **V$ e GV$** (le cosiddette viste dinamiche, anch'esse ad accesso limitato) **per le informazioni variabili**.

SELECT * FROM dict;
➔ permette di esplorare l'intero contenuto del Data Dictionary

VISTE INFORMATIVE SULLE DEFINIZIONI DEGLI OGGETTI:

- CAT ➔ elenco di SEQUENCE/TABLE/VIEW di proprietà dell'utente (corrisponde alla vista di sistema USER_CATALOG)
- USER_OBJECTS ➔ Anagrafica degli oggetti (tutti i tipi)
- USER_TABLES ➔ Anagrafica delle tabelle
- USER_CONS_COLUMNS ➔ Legame Vincoli-Colonne
- USER_CONSTRAINTS ➔ Anagrafica dei vincoli
- USER_TAB_COLS ➔ Anagrafica delle colonne
- USER_VIEWS ➔ Anagrafica delle viste
- USER_INDEXES ➔ Anagrafica degli indici
- USER_SYNONYMS ➔ Anagrafica dei sinonimi
- USER_PROCEDURES ➔ Anagrafica delle procedure/funzioni/packages
- USER_DB_LINKS ➔ Anagrafica dei DB link
- USER_TYPES ➔ Anagrafica dei Type
- USER_TRIGGERS ➔ Anagrafica dei trigger

VISTE INFORMATIVE SUI DATI DI SISTEMA:

- USER_ROLE_PRIVS ➔ Ruoli associati all'utente
- USER_TAB_PRIVS ➔ Privilegi sulle tabelle associati all'utente
- USER_USERS ➔ Stato dell'utenza
- ALL_DIRECTORIES ➔ Directory disponibili per scrittura di files
- USER_ERRORS ➔ Errori nel codice associati alla compilazione
- USER_SOURCE ➔ Codice degli oggetti
- USER_DEPENDENCIES ➔ Legami di dipendenza tra oggetti
- USER_JOBS ➔ Cruscotto jobs
- NLS_DATABASE_PARAMETERS ➔ Parametri a livello di db
- NLS_INSTANCE_PARAMETERS ➔ Parametri a livello di istanza (anche V$SYSTEM_PARAMETER)
- NLS_SESSION_PARAMETERS ➔ Parametri di sessione (anche V$PARAMETER)
- V$VERSION ➔ Info sulla versione di Oracle