SQL
Corso base

INDICE

1

CAPITOLO

OVERVIEW

Il modello relazionale

Il modello relazionale realizza nel modo più appropriato il concetto di indipendenza dei dati in modo tale che l'utente veda solamente il contenuto delle informazioni e non la loro rappresentazione fisica.
Tale modello si basa su di una rappresentazione matematica che ha origine dal concetto di relazione standardizzata, introdotto per la prima volta da Codd.

Il termine "relazione" è in matematica così definito:

Dati gli insiemi ***D1, D2, …, Dn*** *(non necessariamente distinti) si definiscono relazioni su tali insiemi* ***R(D1, D2, …,Dn)*** *le enuple* ***(d1, d2, …, dn)*** *tali che* ***d1*** *sia contenuto* ***in D1, d2 in D2, …, dn in Dn****.*

Una relazione può essere ben rappresentata in una tabella in cui ogni riga rappresenta una tupla; ogni riga inoltre risulta composta da **n** colonne.

- Una colonna il cui valore definisce in modo univoco una relazione è detta **Chiave**.
  E' possibile che una relazione abbia più chiavi, in questo caso è compito dell'utente definire la **Chiave Primaria**.

INSIEMI

	D_1	D_2	.....	D_n
$R_1(D_1...D_n)$	d_{11}	d_{21}	.....	d_{n1}
	.....		.....	
	.....		.....	
	.....		.....	
	.....		.....	
$R_m(D_1...D_n)$	d_{1m}	d_{2m}	.....	d_{nm}

RELAZIONE

- I file sono visti come tabelle che forniscono differenti tipi di aggregazione di dati.
- Ciascuna riga della tabella corrisponde ad un record del file.
- Gli stessi campi possono apparire in diverse tabelle (relazioni) con differenti aggregazioni.

Chiavi

Un attributo di relazione può essere utilizzato come identificatore univoco di una tupla (riga) in una relazione (tabella).

Tale attributo è detto **CHIAVE PRIMARIA**.

La Chiave Primaria può anche essere identificata da un insieme di più attributi.

Quando le combinazioni di attributi hanno caratteristiche di chiave, nel senso sopra citato, si parla di **CHIAVE CANDIDATA**.

Una Chiave Candidata che non sia Chiave Primaria è detta **CHIAVE ALTERNATIVA.**

Gli attributi di una tabella che sono Chiave Primaria in altre tabelle, sono chiamati **CHIAVE ESTERNA.**

Normalizzazione di tabelle

Perché i dati siano organizzati razionalmente all'interno delle tabelle, esistono dei criteri da adottare quando si definiscono i contenuti delle tabelle stesse. Tali criteri sono conosciuti come *normalizzazioni*.

PRIMA FORMA NORMALE

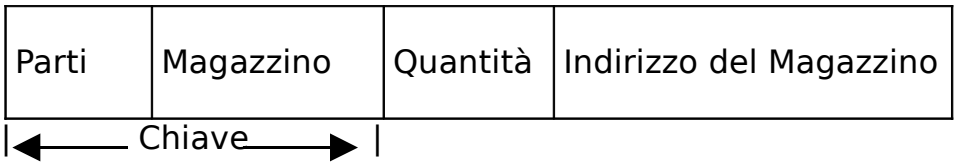
In ciascuna posizione di una tabella identificata da (riga, colonna) deve trovarsi un solo valore e non un insieme di valori.

Più che un criterio di progettazione, questo è un elemento che caratterizza i Database Relazionali.

SECONDA FORMA NORMALE

Un campo (che non sia Chiave) deve dipendere da una chiave unica e non da un sottoinsieme dei campi che compongono una chiave.

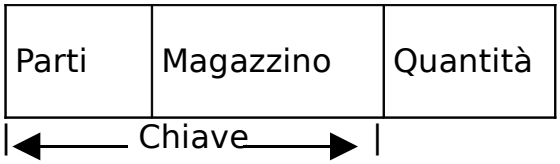
Esempio:



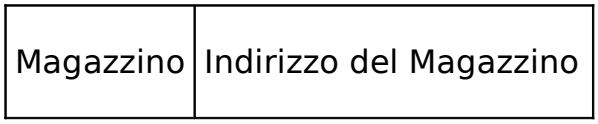
Questa non è una forma standard, perché Indirizzo del Magazzino è funzione solo di Magazzino. Tale situazione genera i seguenti problemi:

- L'Indirizzo del Magazzino viene puntualmente ripetuto in tutti i record.
- Modifiche dell'Indirizzo del Magazzino devono essere apportate a tutti i record per evitare inconsistenza dei dati.
- Se un magazzino non contiene alcuna parte, non vi è modo di ottenere informazioni sul suo indirizzo.

Un’informazione NORMALIZZATA ha invece la seguente forma:

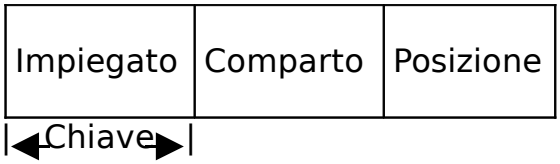


Una seconda tabella basata su due record che sono collegati sulla base del contenuto del campo Magazzino, potrebbe, se richiesto, ricostruire l’informazione.



TERZA FORMA NORMALE
Un campo (che non sia Chiave) non può dipendere da un altro campo non-Chiave (regola della non-transitività).

Esempio:



In questo caso il campo Posizione è funzione del campo Comparto che non è un campo Chiave e quindi la tabella non è normalizzata.

Stessi problemi del caso precedente, la soluzione è dividere i record:

Database per gli esempi

Negli esempi illustrati durante il corso, verranno utilizzate principalmente le due tabelle seguenti:

- EMP: contiene i dati dei dipendenti

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-81	1300		10

- DEPT: contiene i dati sui dipartimenti dove lavorano i dipendenti

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

I dati presenti nelle due tabelle sono collegabili mediante i valori comuni della colonna “DEPTNO”, presente in entrambe.

2

CAPITOLO

SQL IN AMBIENTE ORACLE

Oracle SQL statements

- Interrogazione e manipolazione dati

SELECT	Estrae righe da una o più tabelle
INSERT	Inserisce nuove righe in una tabella
UPDATE	Modifica i dati di una o più righe
DELETE	Elimina righe da una tabella

- Controllo transazioni e concorrenza

COMMIT	Rende effettive le modifiche ai dati
ROLLBACK	Annulla le modifiche dall'ultimo COMMIT
SAVEPOINT	Annulla le modifiche fino al punto indicato
LOCK TABLE	Controlla la condivisione dei dati

- Data Definition

CREATE TABLE	Crea una nuova tabella
ALTER TABLE	Aggiunge o ridefinisce colonne
CREATE VIEW	Crea una vista logica su una o più tabelle
CREATE SYNONYM	Crea un sinonimo per un nome tabella/view
CREATE SEQUENCE	Crea un generatore di numeri di sequenza
RENAME	Cambia il nome di una tabella/view/sinonimo
DROP	Distrugge una tabella, view, ecc.

COMMENT ON	Assegna commenti per una tabella o colonna

- Controllo strutture fisiche

CREATE INDEX	Crea un indice per una tabella
VALIDATE INDEX	Verifica l’integrità di un indice
CREATE CLUSTER	Crea un cluster per più tabelle
CREATE DATABASE	Crea le strutture iniziali del DB
CREATE TABLESPACE	Crea un tablespace addizionale
CREATE ROLLBACK	Crea un segmento per i rollbacks
ALTER ……	Modifica le caratteristiche di …..

- Controllo accessi

GRANT	Assegna dei privilegi a ruoli e/o utenti
REVOKE	Annulla dei privilegi a ruoli e/o utenti
AUDIT	Attiva la sorveglianza su attività utente

Attivazione di SQL*Plus

Pre-requisiti:

- Il DataBase Oracle deve essere stato istanziato
- Bisogna disporre di un nome utente e di una password validi, creati dal DBA, per collegarsi a ORACLE
  (es. NOME=scott PASSWORD=tiger)

Al terminale introdurre:

SQLPLUS (nome/password)

Se viene omessa l'indicazione di 'nome/password', verrà comunque richiesta interattivamente (la password verrà nascosta durante la digitazione).
Attendere il prompt SQLPLUS:

SQL>

A questo punto si possono far eseguire due distinti tipi di comandi:

- **Comandi SQL (il linguaggio SQL)**
- **Comandi SQL*Plus(direttive ausiliarie)**

Per uscire da SQL*Plus basta digitare la direttiva:

EXIToppure **QUIT**

Osservazione: Sqlplus, al momento in cui viene richiamato, cerca nella Working un file "login.sql" e, se esiste, esegue le direttive in esso codificate.

Sintassi SQL in SQL*Plus

Comandi SQL:

- Possono essere introdotti su una o più righe, andando normalmente a capo quando si vuole.
- Possono scriversi indifferentemente in maiuscolo o minuscolo.
- Vengono memorizzati in un buffer, in grado di contenerne uno alla volta (l'ultimo eseguito).
- Per essere eseguiti, vanno terminati con il segnale

;

oppure

- Terminare col punto (**.**)

- una volta tornati al prompt SQL, si può introdurre il comando di esecuzione:

RUN oppure **/**

Esempi:

```
SQL> SELECT alfa,beta
       FROM gamma
       WHERE alfa > 10;


SQL> /
```

Comandi SQL*Plus:

- Possono essere introdotti su una o più righe, andando a capo dopo il carattere “**-**”.
- Possono scriversi indifferentemente in maiuscolo o minuscolo.
- Non vengono memorizzati in un buffer, ma vengono immediatamente eseguiti digitando enter.

3

CAPITOLO

INTERROGAZIONE DEL DATABASE

Join di tabelle

Il Join permette di ottenere in una query i dati provenienti da due o più tabelle, aventi colonne di significato simile.

Oracle consente tre tipi di Join:

- **Equi-Join** Basato sulla uguaglianza dei contenuti delle colonne di configurazione

- **Non Equi-Join** Basato su una comparazione delle colonne di congiunzione con operatori
  > < >= <= != BETWEEN LIKE

- **Outer-Join** Come Equi-Join, ma restituisce anche le righe che non soddisfano l'Equi-Join
- **Self-Join** Come Equi-Join, ma tra la tabella e se stessa nel caso di dipendenza gerarchica.

Esempio di Equi-Join:

Dove lavorano i dipendenti con mansione = 'CLERK'

```
SELECT emp, deptno, dname, loc, ename, job
FROM emp, dept
WHERE emp.deptno = dept.deptno        ←—— equi-join
      AND job = 'CLERK'
ORDER BY emp.deptno;
```

DEPTNO	DNAME	LOC	ENAME	JOB
10	ACCOUNTING	NEW YORK	MILLER	CLERK
20	RESEARCH	DALLAS	SMITH	CLERK
30	SALES	CHICAGO	ADAMS	CLERK
40	OPERATIONS	BOSTON	JAMES	CLERK

Il join può essere eseguito anche su una unica tabella, (Self Join) purché venga vista con due nomi differenti (alias) nella query.

Per ciascuna riga delle tabella, verranno esaminate tutte le righe della tabella stessa.

Esempio:
come si chiamano i capi dei vari dipendenti

```
SELECT i.ename NOME_IMP, m.ename NOME_MGR
FROM emp i, emp m
WHERE i.mgr =  m.empno;
```

⟵ equi-join sulla stessa tabella con due nomi differenti

NOME_IMP	NOME_MGR
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING
MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
TURNER	BLAKE
.....	.....

Il join può essere realizzato avvalendosi di qualsiasi tipo di comparazione tra colonne delle tabelle esaminate.

Esempio:
a quale livello retributivo appartiene ogni dipendente

```
SELECT * FROM salgrade;
```

⟵ Tabella livelli retributivi

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

```
SELECT ename, sal, grade
FROM emp, salgrade
WHERE sal BETWEEN losal AND hisal;
```

⟵ Non equi-join

ENAME	SAL	GRADE
SMITH	800	1
ADAMS	1100	1
JAMES	950	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
ALLEN	1600	3
TURNER	1500	3
JONES	2975	4
BLAKE	2850	4
CLARK	2450	4
SCOTT	3000	4
FORD	3000	4
KING	5000	5

L’outer-join permette di restituire le righe che, nel join, non trovano corrispondenza, associandole a righe nulle.

Utilizza il simbolo (+) sulla colonna di join dove si permette che manchi la corrispondenza.

Esempio:

listare gli addetti di tutti i dipartimenti, evidenziando anche i dipartimenti senza addetti.

```
SELECT dept.deptno, dname, ename
FROM  dept, emp
WHERE dept.deptno = emp.deptno(+);      <-- Outer-Join
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	CLARK
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
20	RESEARCH	SMITH
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	ADAMS
20	RESEARCH	FORD
30	RESEARCH	ALLEN
30	SALES	BLAKE
30	SALES	TURNER

30	SALES	JAMES
30	SALES	MARTIN
30	SALES	WARD
40	OPERATIONS	

Si possono effettuare join su più di due tabelle, aggiungendo nella WHERE nuove condizioni in “AND”.

Si possono mescolare tipi diversi di join nella stessa query.

Esempio:

visualizzare nome-dipartimento, nome-impiegato e livello retributivo di ciascun impiegato.

```
SELECT dname, ename, grade
FROM emp, dept, salgrade          <-- Join di tre tabelle

WHERE emp.deptno = dept.deptno    <-- equi-join

  AND sal BETWEEN losal AND hisal; <-- non equi-join
```

DNAME	ENAME	GRADE
RESEARCH	SMITH	1
RESEARCH	ADAMS	1
SALES	JAMES	1
SALES	WARD	2
SALES	MARTIN	2
ACCOUNTING	MILLER	2
SALES	ALLEN	3
SALES	TURNER	3
RESEARCH	JONES	4
SALES	BLAKE	4
ACCOUNTING	CLARK	4
RESEARCH	SCOTT	4
RESEARCH	FORD	4
ACCOUNTING	KING	5

Titoli delle colonne

Nell'ambito di una 'SELECT' si possono far comparire titoli di colonna diversi rispetto al nome effettivo della colonna o al testo dell'espressione formulata, che rappresentano i defaults per SQL*Plus.

Basta indicare il titolo dopo il dato stesso, eventualmente racchiuso tra virgolette se contiene spazi.

Esempio:

```
SQL>SELECT ename NOME, sal STIPENDIO, comm,
          sal+NVL(comm,0)  "COMPENSO TOTALE"
      FROM emp
      WHERE deptno = 30 AND SAL > 800;
```

NOME	STIPENDIO	COMM	COMPENSO TOTALE
ALLEN	1600	300	1900
WARD	1250	500	1750
MARTIN	1250	1400	2650
BLAKE	2850		2850
TURNER	1500	0	1500
JAMES	950		950

6 records selected.

N.B. Questi titoli non possono essere usati al posto dei veri nomi colonna nelle 'WHERE' o 'ORDER BY'.

Funzioni aritmetiche

ABS(n) Valore assoluto di ‘n’
Es. ABS(-17.34) ⟶ 17.34

CEIL(n) Minimo intero > = ‘n’
Es. CEIL(-17.34) ⟶ -17
CEIL(17.34) ⟶ 18

FLOOR(n) Massimo intero <= ‘n’
Es. FLOOR(17.34)⟶ 17
FLOOR(-17.34) ⟶ -18

GREATEST(n1, .. , n-esimo) Ritorna il maggiore tra n1, ... ,n-esimo valore
Es. GREATEST(37,50,45) ⟶50

LEAST(n1, .. , n-esimo) Ritorna il minore tra n1, ... ,n-esimo valore
Es. LEAST(37,50,45) ⟶ 37

MOD(m,n) Resto della divisione intera (m/n)
Es. MOD(17,5) ⟶ 2

POWER(m,n) Eleva ‘m’ alla n-esima potenza
Es. POWER(2,4) ⟶ 16

ROUND(n[,m]) Arrotonda ‘n’ con ‘m’ decimali
Es. ROUND(26.68,1) ⟶ 26,7

SIGN(n) Ritorna -1 se n < 0 , +1 se n > 0 o se n=0
Es. SIGN(-15) ⟶ -1

SQRT(n) Radice quadrata di ‘n’
Es. SQRT(25) ⟶ 5

TRUNC(n[,m]) Tronca ‘m’ decimali sul valore di ‘n’.
Se omesso ‘m’, elimina tutti i decimali

Es. TRUNC(26.63) $\longrightarrow$ 26

Gli argomenti delle funzioni possono essere espressioni qualsiasi, purché il loro valore risulti numerico.

Esempio:

```
SELECT ename, TRUNC(GREATEST(sal/2, comm)) FROM
emp;
```

Funzioni di manipolazione caratteri

str1 || str2 || ...
Concatenazione di stringhe di caratteri.
Es. 'ABCD' || 'EFG' ⟶ ABCDEFG

CHR(n)
Genera il carattere ASCII/EBCDIC corrispondente al valore decimale 'n'.
Es. CHR(65) ⟶ A

INITCAP(s)
Pone in maiuscolo il carattere iniziale della stringa 's'.
Es. INITCAP('smith') ⟶ Smith

INSTR(s1,s2 [,m[,n]])
Ritorna la posizione della n-esima occorrenza della stringa 's2' entro la stringa 's1' a partire dalla posizione 'm'.
Se omesso 'n' cerca la prima occorrenza.
Se omesso 'm' cerca dall'inizio di 's1'.
Es. INSTR('MISSISSIPPI','S',5,2) ⟶ 7

LENGTH(s)
Ritorna la lunghezza, in caratteri, di 's'.
Es. LENGTH('SMITH') ⟶ 5

LOWER(s)
Pone in minuscolo tutti i caratteri di 's'.
Es. LOWER('SMITH') ⟶ smith

LPAD(s, n [,c])
Ritorna la stringa 's' completa a sinistra con occorrenze del carattere 'c', per una lunghezza totale di 'n' caratteri.
Se 'c' è omesso, si assume che il carattere sia blank.
Es. LPAD('XYZW',7,'*') ⟶ ***XYZW

LTRIM(s, c)
Elimina dalla stringa 's' tutte le occorrenze del set di caratteri 'c' presenti a sinistra e sino al primo carattere diverso da 'c'.
Es. LTRIM('***XYZW**','*') ⟶ XYZW**

REPLACE(s1, s2 [,s3])
Cambia nella stringa 's1' tutte le occorrenze della stringa 's2' con altrettante di 's3'
Es. REPLACE('ABACAB','AB','XXX') ⟶ XXXACXXX

RPAD(s, n [,c]) Ritorna la stringa ‘s’ completa a destra con occorrenze del carattere ‘c’, per una lunghezza totale di ‘n’ caratteri.
Se ‘c’ è omesso, si assume che il carattere sia blank.
Es. RPAD(‘XYZW’,7,’*’) ⟶ XYZW***

RTRIM(s, c) Elimina dalla stringa ‘s’ tutte le occorrenze del set di caratteri ‘c’ presenti sulla destra e sino al primo carattere diverso da ‘c’.
Es. RTRIM(‘***XYZW**’,’*’) ⟶ ***XYZW

SUBSTR(s, m [,n]) Estrae dalla stringa ‘s’ una sottostringa di ‘n’ caratteri a partire dalla posizione ‘m’.
Se omesso, estrae sino alla fine di ‘s’.
Es. SUBSTR(‘ABCDEF’,2,3) ⟶ BCD

TRANSLATE(s1, s2, s3) Nella stringa ‘s1’ cambia le corrispondenze dei caratteri elencati nella stringa ‘s2’ con quelli elencati nella stringa ‘s3’.
Es. TRANSLATE(‘ACBCAB’,’AB’,’12’) ⟶ 1C2C12

UPPER(s) Pone in maiuscolo tutti i caratteri di ‘s’
Es. UPPER(‘smith’) ⟶ SMITH

Gli argomenti delle funzioni possono essere espressioni qualsiasi, purché il loro valore risulti del tipo richiesto dalla natura degli argomenti previsti.

Esempio:

```
SQL> SELECT deptno, RPAD(dname,10)||'  -  '||INITCAP(LOWER(LOC))
                    DEPARTMENT
       FROM dept;

    DEPTNO   DEPARTMENT
    ----------   ----------------------
          10  ACCOUNTING       New York
          20  RESEARCH         Dallas
          30  SALES            Chicago
          40  OPERATIONS       Boston
```

Funzioni di conversione

DECODE(e,v1,d1 [,v2,d2] ...[,def])

Genera ‘d1’ se l’espressione ‘e’ risulta = ‘v1’,
genera ‘d2’ se l’espressione ‘e’ risulta = ‘v2’,
e similmente per tutte le altre coppie [v-esime, d-esime].
Se ad ‘e’ non corrisponde alcuno dei valori indicati viene generato il default ‘def’ (NULL se omesso).

Esempi:

```
SELECT ename, job
DECODE(job,’SALESMAN’,100,’ANALYST’,200,300)
      JOB_CODE
FROM emp;
```

ENAME	JOB	JOB_CODE
SMITH	CLERK	300
WARD	SALESMAN	100
BLAKE	MANAGER	300
.....	.....	.....

```
SELECT ename, job, DECODE(job, ’SALESMAN’, sal +
comm, sal) COMPENSO
FROM emp;
```

ENAME	JOB	COMPENSO
SMITH	CLERK	800
ALLEN	SALESMAN	1900
WARD	SALESMAN	1750
JONES	MANAGER	2975
.....	.....	.....

TO_CHAR(n, formato) Converte il dato NUMBER o DATE ‘n’ in una stringa, secondo il ‘formato’ indicato (vedi formati per numeri/date).
Es. TO_CHAR(12345,’9999.99’) ⟶ 123.45
TO_CHAR(HIREDATE,’MM/YY’) ⟶ 09/85

TO_DATE(v, formato) Converte il dato CHARACTER o NUMBER ‘v’ in una data, secondo il ‘formato’ specificato (vedi formati per le date).
Es. TO_DATE(‘270990’, ‘DD_MM_YY’) ⟶ 27-SEP-90

TO_NUMBER(s) in	Converte la stringa di caratteri ‘s’ in un dato formato numerico. Es. TO_NUMBER(‘123’)+1 $\longrightarrow$ 124
NVL(dato, vnull)	Converte l’eventuale NULL in un valore specificato. Es. NVL(comm,0)

4

CAPITOLO

INTERFACCIA SQL*Plus

Caratteristiche SQL*Plus

Collegandosi a SQL*Plus si ha la possibilità di utilizzare due distinti tipi di comandi:

- **Istruzioni SQL;**
- **Direttive interne a SQL*Plus;**

Mentre le direttive interne a SQL*Plus, una volta introdotte ed eseguite, non restano memorizzate, una istruzione SQL viene mantenuta in un buffer e può essere ulteriormente esaminata, editata e rieseguita.

Il buffer delle istruzioni SQL è comunque in grado di gestirne una alla volta. Digitando una nuova istruzione SQL, si cancella quella precedentemente inserita.

E’ però possibile salvare il contenuto del buffer su file esterno e richiamarlo quando occorre.

Editing del command-buffer

Per editare il contenuto del command buffer di SQL*Plus sono disponibili le seguenti direttive:

- **LIST** **L** Per listare una o più linee del buffer
- **CHANGE** **C** Per modificare la linea corrente
- **APPEND** **A** Per aggiungere testo alla linea corrente
- **INPUT** **I** Per inserire righe dopo la linea corrente
- **DELETE** **D** Per cancellare la linea corrente

Una linea diventa corrente quando viene evidenziata, come unica o ultima linea, da una “LIST”.

Esiste poi la direttiva di esecuzione del contenuto del buffer:

- **RUN** **R** Esegue l’istruzione SQL contenuta nel buffer (“/” per evitare la visualizzazione del comando).

Direttiva LIST

L Lista tutte le linee

L n Lista solo la linea ‘n’

L m n Lista da linea ‘m’ a linea ‘n’

In luogo di ‘m’ e ‘n’ si possono indicare:

***** Per indicare la linea corrente

LAST Per indicare l’ultima linea del buffer

Esempi:

```
SQL>L
      SELECT deptno, ename, sal
      FROM emp
      * WHERE deptno = 10

SQL>L1
      * SELECT deptno, ename, sal
```

La linea corrente viene identificata con il simbolo ‘*’.

Direttiva CHANGE

Formato:

c / stringa1/ stringa2/ Sostituisce la prima occorrenza di 'stringa1' con 'stringa2' nella linea corrente

Esempio:

```
SQL>L1
      * SELECT dptno, ename, sal

SQL> c/dptno/deptno/
      * SELECT deptno, ename, sal
```

Nel caso in cui il testo da cambiare contenga il carattere "/" si può usare come separatore un qualsiasi carattere speciale non presente nel testo medesimo.

Direttiva APPEND

Formato:

A ***testo*** Aggiunge ‘testo’ in fondo alla linea corrente.

Esempio:

```
SQL> L1
       * SELECT deptno, ename, sal

SQL> a ,comm
      * SELECT deptno, ename, sal, comm
```

Direttiva INPUT

Formato:

I ***testo*** Aggiunge ‘testo’ su nuove righe dopo la linea corrente.

Esempio:

```
SQL> L last
     * WHERE deptno = 10

SQL> i
     * ORDER BY sal
                                   ←—— Una linea vuota termina l’input del testo

SQL> r
```

```
SELECT deptno, ename, sal, comm
FROM emp
WHERE deptno = 10
* ORDER BY sal
```

```
DEPTNO   ENAME      SAL     COMM
------ ----------  -------  -----
    10  MILLER      1300
    10  CLARK       2450
    10  KING        5000
```

Tramite la “INPUT” si possono, in effetti, caricare nel buffer più istruzioni SQL, ma al momento del RUN si avrà un messaggio di errore, poiché il contenuto del buffer verrà interpretato come un singolo statement.

Direttiva DELETE

Formato:

del Elimina la linea corrente

Esempio:

```
SQL> L last
      * ORDER BY sal

SQL> L *
      * ORDER BY sal

SQL> del

SQL> l
      SELECT deptno, ename, sal, comm
      FROM emp
      * WHERE deptno = 10
```

Direttiva SAVE

Permette di salvare su un file esterno il contenuto del buffer di SQL*Plus

Formato:

save *nome file* [create | replace | append]

nome file Se fornito senza suffisso, viene aggiunto il default “.sql”

create Il file non deve esistere già

replace Il file se esiste viene riscritto

append Il contenuto del buffer viene accodato al contenuto attuale del file

Direttiva GET

Permette di caricare il contenuto del file nel buffer SQL*Plus, al fine di editarlo e poi riscriverlo.

Formato:

get *nome_file* [list]

nome file Se fornito senza suffisso, viene aggiunto il default ".sql"

Direttiva START

Permette di leggere ed eseguire una dopo l'altra le istruzioni SQL o direttive SQL*Plus contenute in un file.

Formato:

start *nome_file* oppure **@*nome_file***

nome_file se fornito senza suffisso, viene aggiunto il default ".sql"

Il file richiamato potrà essere stato creato anche con un text editor esterno, cosa senz'altro necessaria se contiene direttive SQL*Plus.

Se sono presenti più istruzioni SQL, ciascuna di esse dovrà essere marcata con il segnale di esecuzione “;” oppure essere seguita dalla direttiva di run “/”.

Si può ottenere la start di un file direttamente al momento del richiamo di SQL*Plus, con la sintassi:

sqlplus [-s] username/password @*nome_file*

Il file da eseguire potrà contenere, se non specificato sulla riga di comando, come prima riga, la stringa “utente/password” e, come ultima, una “EXIT” nel caso sia richiesta la disconnessione a fine esecuzione.

L’opzione “-s” (leggi -SILENT), evita se necessario, di far comparire i messaggi di attivazione di SQL*Plus.

Variabili di sostituzione

SQL*Plus consente di rendere parametriche le istruzioni SQL delle altre direttive, introducendo una o più componenti di tipo variabile, identificate da nomi che iniziano con “&”.
Ogni volta che una istruzione contenente variabili viene eseguita, SQL*Plus chiede all’operatore i valori effettivi da sostituire al loro posto.
Le variabili possono sostituire costanti, nomi di colonne, nomi di tabelle e qualsiasi altro argomento specificabile.
Esempio:

```
SQL> set verify on

SQL> SELECT deptno, ename, job, sal
       FROM emp
       WHERE deptno =  &DEPTNUM AND job = ‘&JOB’;

Enter value for deptnum:  20
Enter value for job : CLERK

old    3: WHERE deptno =  &DEPTNUM AND job = ‘&JOB’
new  3: WHERE deptno =  20 AND job =  ‘CLERK’;

 DEPTNO   ENAME    JOB          SAL
       ------  ----------   ---------       -----
         20  SMITH     CLERK      1300
         20  ADAMS    CLERK      2450

SQL> set verify off

SQL> SELECT empno, ename, job, sal
       FROM emp
       WHERE deptno =  &DEPTNUM
       ORDER BY  &ORDERBY;

Enter value for deptnum:  20
Enter value for orderby :   SAL
```

```
EMPNO  ENAME    JOB              SAL
-------  ----------   --------------    -------
  7369  SMITH    CLERK             800
  7876  ADAMS    CLERK            1100
  7566  JONES    MANAGER          2975
  7902  FORD     ANALYST          3000
  7788  SCOTT    ANALYST          3100
```

Se la stessa variabile viene usata in più punti, per evitare molteplici richieste di valorizzazione basta denominarla con il prefisso “&&”, livello sessione, invece del semplice “&”, livello statement.

Esempio: file richiamato con una ‘START’

```
SELECT deptno, ename, job
FROM emp
WHERE deptno = &&DEPTNUM;

SELECT * FROM dept
WHERE DEPTNO = &&DEPTNUM;
```

```
Enter value from deptnum: 10
```

```
DEPTNO   ENAME    JOB
------  ------       --------------
    10  CLARK    MANAGER
    10  KING     PRESIDENT
    10  MILLER   CLERK


DEPTNO   DNAME           LOC
------  ------------------   --------------
    10  ACCOUNTING   NEW YORK
```

Il processo di sostituzione delle variabili può essere inibito e riabilitato con la direttiva:

SET SCAN OFF|ON

Direttive DEFINE e UNDEFINE

Quando una variabile riceve un valore, essa viene allocata in un buffer di SQL*Plus.
Per vedere i valori correnti delle variabili di sostituzione, si può utilizzare la direttiva:

DEF [*nome-variabile*]

Se viene omesso il "*nome-variabile*" si ottiene la lista dei valori correnti di tutte le variabili attive.

Per assegnare esplicitamente un valore ad una variabile si usa:

DEF nome-variabile = *valore*

Per disattivare una variabile si può usare la direttiva:

UNDEFINE nome-variabile

Esempio:

```
SQL> DEF deptnum=10

SQL> SELECT * FROM dept WHERE deptno = &DEPTNUM;

 DEPTNO    DNAME              LOC
------  ------------------    ---------------
     10  ACCOUNTING    NEW YORK

SQL> DEF deptnum

      DEFINE   DEPTNUM  = "10" (CHAR)

SQL> UNDEFINE deptnum

SQL> /

      Enter value for deptnum:  10

  DEPTNO  DNAME              LOC
  ------------  ----------------------    ----------------
      10  ACCOUNTING    NEW YORK
```

Direttiva ACCEPT

Consente di acquisire i valori delle variabili interattivamente prima della esecuzione delle istruzioni SQL che le utilizzano.

Formato:

ACCEPT *nome-variabile* [NUMBER | CHAR] [PROMPT 'messaggio']

NUMBER | CHAR Controlla il tipo dei dati in input

PROMPT 'messaggio' Visualizza un messaggio per l'operatore

L'uso della sola clausola 'PROMPT' permette di inviare messaggi al terminale durante l'esecuzione di una procedura.

Esempio:
file richiamato con una "START"

```
PROMPT  '**** LISTA DIPENDENTI  ****'

ACCEPT deptnum NUMBER PROMPT  'Inserire N.
            Dipartimento:'

SELECT empno, ename FROM emp WHERE deptno =
            &DEPTNUM;
```

output prodotto:

```
**** LISTA DIPENDENTI  ****

Inserire N. Dipartimento: 10

 EMPNO    ENAME
------  ----------
  7782  CLARK
  7839  KING
  7934  MILLER
```

Valorizzazione non interattiva delle variabili

I valori da assegnare alle variabili di sostituzione possono essere specificati direttamente nel comando che manda in esecuzione il file.

In tal caso, i nomi delle variabili devono essere costituiti dal prefisso “&” e da un numero progressivo, poiché il meccanismo di sostituzione diventa posizionale.

Esistono due diverse possibilità:

- Nella “START” quando si è già collegati a SQL*Plus

  START *nome-file* valore-1 valore-2 .....

- Nel comando di attivazione SQL*Plus

  SQLPLUS [-SILENT] @ *nome-file* valore-1 valore-2 .....

In questo caso il file eseguito deve contenere nella prima riga la stringa “utente/password” e la direttiva “EXIT” nell’ultima riga.

Esempio:
dato il file “rep1.sql” contenente

```
SCOTT/TIGER

SELECT * FROM emp
            WHERE deptno = &1
            AND  job = ‘&2’;
EXIT
```

Per eseguirlo, al richiamo di SQL*Plus si richiederà, ad es.:

```
SQLPLUS @rep1 30 SALESMAN
```

“&1” verrà sostituito da “30” e “&2” verrà sostituito da “SALESMAN”

Gestione degli errori

Nel caso si verifichino condizioni anomale durante l'esecuzione di istruzioni SQL, SQL*Plus visualizza un messaggio di errore, in genere ritornato dal Kernel Oracle che interpreta ed esegue le istruzioni SQL.

In molti casi un "*" punta all'elemento sintattico all'origine della condizione di errore.

Esempio:

```
SQL> SELECT ename, job FROM emps;

SELECT ename, job FROM emps
                      *
ERROR at line 1: ORA-0942:  table or view does not exist
```

Il codice di errore e le ulteriori spiegazioni si trovano sul manuale degli "ERROR MESSAGES AND CODES".

Qualora si voglia controllare esplicitamente il comportamento di SQL*Plus a seguito di condizioni di errore, a partire dalla V3 esiste la direttiva:

WHENEVER SQLERROR

[EXIT [SUCCESS | FAILURE | WARNING | n | variable] | CONTINUE]

L'opzione "EXIT" provoca l'uscita da SQL*Plus quando fallisce una successiva istruzione SQL, mentre la "CONTINUE" ripristina la condizione abituale, che tiene l'utente collegato anche in caso di errori.

L'uscita può essere accompagnata dalla segnalazione al sistema operativo di una specifica condizione o codice di ritorno:

SUCCESS	Ritorna un codice di successo (default)
FAILURE	Ritorna un codice di errore pre-definito
WARNING	Ritorna un codice di warning pre-definito
n	Ritorna il valore "n" come codice di uscita
variable di	Ritorna il contenuto della variabile indicata come codice uscita (ex. SQL.SQLCODE)

Anche quando si esce volutamente da SQL*Plus con la direttiva “EXIT” o “QUIT” si possono ritornare al S.O. codici di uscita in modo simile alla “WHENEVER EXIT ....”, usando il formato:

EXIT [SUCCESS | FAILURE | WARNING | n | variable]

Connessione ad un altro utente

Mentre si è collegati a SQL*Plus con un certo nome utente, si può commutare verso un altro identificativo utente, a patto che si conoscano nome e password di accesso, con la direttiva:

CONNECT username/password

Esempio:

```
SQL> CONNECT system/manager
```

Accesso al sistema operativo

Mentre si è collegati a SQL*Plus si possono richiamare i comandi del sistema operativo in cui Oracle lavora (Es. Unix), mediante la direttiva:

HOST *comando-sistema*

Generalmente, invece della parola “HOST” si possono utilizzare anche caratteri di abbreviazione, come “$” oppure “!”.
Dopo l’esecuzione del comando si torna automaticamente in SQL*Plus.

Esempio (Unix):

```
SQL> !ls *.sql

abcd.sql

SQL> !ed abcd.sql
        .....
        .....

SQL> START abcd
```

Per quanto riguarda l’uso di un text editor esterno, si può anche utilizzare un’apposita direttiva di SQL*Plus:

ED [*nome-file*]

Questa fa riferimento al valore di una variabile “_editor” che contiene il nome del text editor da richiamare.
Ad esempio se è stata data una direttiva “DEF _editor = vi”, ogni volta che in SQL*Plus verrà richiamata la direttiva “ED” si richiamerà l’editor “vi”.
Se non si specifica il “*nome-file*”, l’editor richiamato agirà direttamente sui contenuti del buffer di SQL*Plus.
Specificando invece un file, il contenuto attuale del buffer di SQL*Plus viene salvato su un file “afiedt.buf” e ripreso da esso in uscita dalla sessione di editor.

5

CAPITOLO

QUERY CON FUNZIONI DI GRUPPO

Query-selezione con raggruppamento

SELECT dato-1,……., dato-n
FROM tabella-1, ……., tabella-n
WHERE condizioni su righe individuali
GROUP BY grp-1, ……., grp-n
HAVING condizioni su funzioni di gruppo
ORDER BY ord-1 [ASC | DESC], ….., ord-n [ASC | DESC]

dato-1,……., dato-n Possono essere solo chiavi di raggruppamento (grp-1,...) o funzioni di gruppo (SUM, AVG,..).

grp-1, ……., grp-n Colonne o espressioni sui cui valori vengono effettuati i raggruppamenti (es. GROUP BY deptno, job).
Se si omette la 'GROUP BY ..' si ottiene un unico gruppo coincidente con tutte le righe selezionate dalla 'WHERE'.

HAVING Impone condizioni sui gruppi di righe, mentre la 'WHERE' lavora su ciascuna riga individuale della tabella
(es. HAVING AVG(sal) > 1000)

Funzioni di gruppo

AVG(dato) Valuta la media dei valori del ‘dato’ sui gruppi generati dalla query.
Es. AVG(sal), AVG(sal*12)

COUNT(dato) Conteggia il numero di valori non nulli per il ‘dato’ in ciascun gruppo generato.
Es. COUNT(comm)

COUNT(*) Conteggia il numero di righe per ciascun gruppo generato dalla query.

MAX(dato) Valuta il massimo dei valori del ‘dato’ sui gruppi generati dalla query.
Es. MAX(sal), MAX(LENGTH(ename))

MIN(dato) Valuta il minimo dei valori del ‘dato’ sui gruppi generati dalla query.
Es. MIN(sal), MIN(LENGTH(ename))

STDDEV(dato) Valuta la deviazione standard sui valori non nulli di ciascun gruppo.
Es. STDDEV(sal)

SUM(dato) Valuta la somma dei valori del ‘dato’ per ogni gruppo generato dalla query.
Es. SUM(sal)

VARIANCE(dato) Valuta la varianza sui valori non nulli di ciascun gruppo.
Es. VARIANCE(sal)

Tutte le funzioni di gruppo, tranne la ‘COUNT(*)’, ammettono una opzione ‘DISTINCT’ per escludere i valori duplicati del dato su cui la funzione opera.

Esempi:

1. Calcolo somma, media, max e min SAL sull'intera tabella

```
SELECT SUM(sal), AVG(sal), MAX(sal), MIN(sal)
FROM emp;
```

SUM(SAL)	AVG(SAL)	MAX(SAL)	MIN(SAL)
29025	2073	5000	800

2. Calcolo somma, media, max, e min SAL per ciascun dipartimento

```
SELECT deptno, SUM(sal), AVG(sal), MAX(sal), MIN(sal)
FROM emp GROUP BY deptno;
```

DEPTNO	SUM(SAL)	AVG(SAL)	MAX(SAL)	MIN(SAL)
10	8750	2917	5000	1300
20	10875	2175	3000	800
30	9400	1567	2850	950

3. Somma, media, max e min SAL per ciascun dipartimento, ma stampa solo se la media del dipartimento è > 2000

```
SELECT deptno, SUM(sal), AVG(sal), MAX(sal), MIN(sal)
FROM emp
GROUP BY deptno
HAVING AVG(sal) > 2000;
```

DEPTNO	SUM(SAL)	AVG(SAL)	MAX(SAL)	MIN(SAL)
10	8750	2917	5000	1300
20	10875	2175	3000	800

4. Numero di distinte mansioni (job) presenti in ciascun dipartimento.

```
SELECT deptno, COUNT(distinct job) JOBS
FROM emp
GROUP BY deptno;
```

DEPTNO	JOBS
10	3
20	3
30	3

5. Le funzioni di gruppo possono essere utilizzate anche in presenza di join su più tabelle.

Esempio:
totale compensi per dipartimento

```
SELECT emp.deptno, SUM(sal + NVL(comm,0)) COMPENSI
FROM emp, dept
WHERE emp.deptno=dept.deptno
GROUP BY emp.deptno;
```

DEPTNO	COMPENSI
10	8750
20	10875
30	11600

6. Si possono fare raggruppamenti su più colonne, ad es.:

```
SELECT deptno, job, AVG(sal)
FROM emp
WHERE deptno > 10
GROUP BY deptno, job;
```

DEPTNO	JOB	AVG(SAL)
20	ANALYST	3050
20	CLERK	950
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850

```
        30  SALESMAN            1400
```

Non si possono selezionare colonne che non sono funzioni di gruppo o componenti della chiave di raggruppamento.

Ad esempio:

```
SELECT emp.deptno, dname, SUM(sal+NVL(comm,0)) COMPENSI
FROM emp, dept
WHERE emp.deptno = dept.deptno
GROUP BY emp.deptno;
```

Produce l'errore:

```
SELECT emp.deptno, dname, SUM(sal+NVL(comm,0)) COMPENSI
                          *
Error at line 1: ORA-0979: not a GROUP BY expression
```

In questo caso basta aggiungere "dname" nella "GROUP BY".

6

CAPITOLO

SUBQUERY

Subquery

Nella clausola ‘WHERE’ di una select, una o più condizioni possono utilizzare un’altra SELECT per generare i valori su cui eseguire un confronto.
La SELECT interna viene classificata come ‘SUBQUERY’ e viene eseguita prima della query principale.
La SELECT di una subquery può attingere dati da qualsiasi tabella disponibile, compresa quella esaminata a livello di query principale.
Può contenere tutte le clausole di una normale SELECT, ad eccezione della ‘ORDER BY’.
I dati restituiti da una subquery non vengono visualizzati, a meno che non siano espressamente selezionati a livello della SELECT principale.
Le subquery sono utilizzabili anche nelle istruzioni di manipolazione (INSERT, UPDATE, DELETE).

Esempio:

visualizzare il nome degli impiegati che hanno la stessa mansione di ‘JONES’;

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job FROM emp
                    WHERE ename=‘JONES’);
```

ENAME	JOB
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

Se la subquery ritorna più valori, occorre effettuare il confronto con gli operatori ‘ANY’, ‘ALL’, o ‘IN’, che sono in grado di considerare liste di valori.
‘ANY’ opera con logica di tipo ‘OR’ sugli n valori ritornati, mentre ‘ALL’ lavora in modo ‘AND’.
Entrambi vanno preceduti da un operatore di comparazione (= > < ecc.) che ne completa il significato.

Esempi:

1. Visualizzare dipartimento, nome e stipendio degli impiegati che hanno stipendio maggiore del più basso stipendio pagato nel dipartimento 30

```
SELECT dname, ename, sal
FROM emp, dept
WHERE emp.deptno=dept.deptno
      AND  sal > ANY  (SELECT sal FROM emp WHERE
                              deptno=30);
```

DNAME	ENAME	SAL
SALES	ALLEN	1600
SALES	WARD	1250
RESEARCH	JONES	2975
SALES	MARTIN	1250
SALES	BLAKE	2850
ACCOUNTING	CLARK	2450
RESEARCH	SCOTT	3000
ACCOUNTING	KING	5000
SALES	TURNER	1500
RESEARCH	ADAMS	1100
RESEARCH	FORD	3000
ACCOUNTING	MILLER	1300

2. Visualizzare dipartimento, nome e stipendio di quanti hanno stipendio maggiore del più alto stipendio pagato nel dipartimento 30, ordinati per stipendio decrescente

```
SELECT dname, ename, sal
FROM emp, dept
WHERE emp.deptno=dept.deptno
      AND sal > ALL (SELECT sal FROM emp WHERE
                              deptno=30)
ORDER BY sal DESC;
```

DNAME	ENAME	SAL
ACCOUNTING	KING	5000
RESEARCH	SCOTT	3000
RESEARCH	FORD	3000
RESEARCH	JONES	2975

3. Visualizzare nome e mansione di coloro che hanno una mansione che si ritrova tra quelle svolte da quanti lavorano nella sede di Dallas

```
SELECT ename, job
FROM emp
WHERE job IN
      (SELECT job FROM emp,dept
      WHERE emp.deptno = dept.deptno
            AND loc = 'DALLAS');
```

ENAME	JOB
SCOTT	ANALYST
FORD	ANALYST
SMITH	CLERK
MILLER	CLERK
ADAMS	CLERK
JAMES	CLERK
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

Quando il confronto con i dati ritornati dalla subquery coinvolge più colonne, invece di introdurre un 'AND' di più subquery, si possono elencare le colonne stesse.

Esempio:

visualizzare chi svolge la stessa mansione e lavora nello stesso dipartimento di 'ALLEN'

```
SELECT deptno, job, ename, sal
FROM emp
WHERE (deptno, job) = (SELECT deptno,job FROM emp
                       WHERE ename='ALLEN');
```

DEPTNO	JOB	ENAME	SAL
30	SALESMAN	ALLEN	1600
30	SALESMAN	WARD	1250
30	SALESMAN	MARTIN	1250
30	SALESMAN	TURNER	1500

Si possono avere più subquery allo stesso livello rispetto alla query principale, combinati in 'OR' o in 'AND', con altre eventuali clausole della 'WHERE'

Esempio:

visualizzare chi svolge la stessa mansione di 'JONES' o guadagna uno stipendio maggiore o uguale a quello di 'FORD'

```
SELECT ename, job
FROM emp
WHERE job =
      (SELECT job FROM emp WHERE ename ='JONES')
      OR sal >=
      (SELECT sal FROM emp WHERE ename ='FORD');
```

ENAME	JOB
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
FORD	ANALYST

Una subquery può comparire anche all'interno di un altra, senza limitazioni al livello di profondità

Esempio:

visualizzare chi lavora nel dipartimento 10 con una mansione identica ad una qualsiasi tra quelle presenti nel dipartimento di nome 'SALES'

```
SELECT ename, job
FROM emp
WHERE deptno = 10
   AND job IN
       (SELECT job FROM emp
        WHERE deptno =
              (SELECT deptno FROM dept
               WHERE dname='SALES'));
```

ENAME	JOB
MILLER	CLERK
CLARK	MANAGER

Query coordinate o cicliche

Quando occorre che la subquery venga ripetuta per ogni riga esaminata a livello di query principale, bisogna introdurre un alias sul nome tabella della query principale e utilizzarlo nella ‘WHERE’ della subquery.

Esempio:

visualizzare i dati degli impiegati che percepiscono uno stipendio superiore allo stipendio medio del loro dipartimento

```
SELECT deptno, ename, sal
FROM emp E
WHERE sal >
      (SELECT AVG(sal) FROM emp WHERE
                              deptno=E.deptno);
```

DEPTNO	ENAME	SAL
30	ALLEN	1600
20	JONES	2975
30	BLAKE	2850
20	SCOTT	3000
10	KING	5000
20	FORD	3000

Operatore EXISTS

Ritorna un valore “VERO” se la subquery che lo segue ritorna almeno una riga, “FALSO” altrimenti.
E’ disponibile anche in forma negata “NOT EXISTS”.

Esempi:

1. Visualizzare i dati di coloro che lavorano in un dipartimento dove è presente qualche analista.

```
SELECT ename, job, deptno
FROM emp E
WHERE  EXISTS
      (SELECT * FROM emp WHERE job = ‘ANALYST’
            AND deptno = E.deptno);
```

ENAME	JOB	SAL
SMITH	CLERK	3000
JONES	MANAGER	20
SCOTT	ANALYST	20
ADAMS	CLERK	20
FORD	ANALYST	20

2. Visualizzare i dati di coloro che sono stati assunti prima del loro capo

```
SELECT ename, hiredate, mgr
FROM emp E
WHERE  EXISTS
      (SELECT * FROM emp WHERE empno = E.mgr
      AND hiredate > E.hiredate);
```

ENAME	HIREDATE	MGR
SMITH	17-DEC-80	7902
ALLEN	20-FEB-81	7698
WARD	20-FEB-81	7698
JONES	02-APR-81	7839

UNION, INTERSECT, MINUS

Sono operatori su insiemi che permettono di combinare i risultati di più SELECT entro un'unica query.

Le 'SELECT' coinvolte devono ritornare lo stesso numero e tipo di colonne, mentre i loro nomi possono essere differenti.

UNION Righe ritornate da almeno una delle SELECT

INTERSECT Righe ritornate da tutte le SELECT

MINUS Righe ritornate dalla prima select e da altre

Le 'SELECT' individuali operano implicitamente scartando le righe di contenuto duplicato.

La 'ORDER BY' può essere specificata solo per il risultato globale e non per le 'SELECT' componenti. Occorre indicare il numero colonna, poiché i nomi delle colonne omologhe nelle singole 'SELECT' possono essere differenti.

Esempi:

1. Unire i dati di due tabelle 'SALESMAN' e 'ANALYSTS', di struttura simile a 'EMP', contenenti rispettivamente i dati degli impiegati con job = 'SALESMAN' e 'ANALYST'.

```
SELECT empno, ename, sal FROM salesman;
```

```
 EMPNO   ENAME        SAL
------- ---------- ------
  7499  ALLEN       1600
  7521  WARD        1250
  7654  MARTIN      1250
  7844  TURNER      1500
```

```
SELECT empno,ename,sal FROM analysts;
```

EMPNO	ENAME	SAL
7788	SCOTT	3000
7902	FORD	3000

```
SELECT empno, ename, job, sal+comm
FROM salesman
UNION
SELECT empno, ename, job, sal
FROM analysts
ORDER BY 3;
```

EMPNO	ENAME	JOB	SAL
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000
7499	ALLEN	SALESMAN	1900
7521	WARD	SALESMAN	1750
7654	MARTIN	SALESMAN	2650
7844	TURNER	SALESMAN	1500

2. Verificare in quali anni sono stati assunti sia analisti che venditori.

```
SELECT  TO_CHAR(hiredate,'YYYY') ANNO_ASS
FROM salesman
INTERSECT
SELECT  TO_CHAR(hiredate,'YYYY') ANNO_ASS
FROM analysts;
```

ANNO_ASS
1981

3. Verificare quali valori di codice dipartimento DEPTNO presenti in ‘DEPT’ non trovano corrispondenza in ‘EMP’

```
SELECT deptno FROM dept
MINUS
SELECT deptno FROM emp;
```

```
    DEPTNO
----------
        40
```

7

CAPITOLO

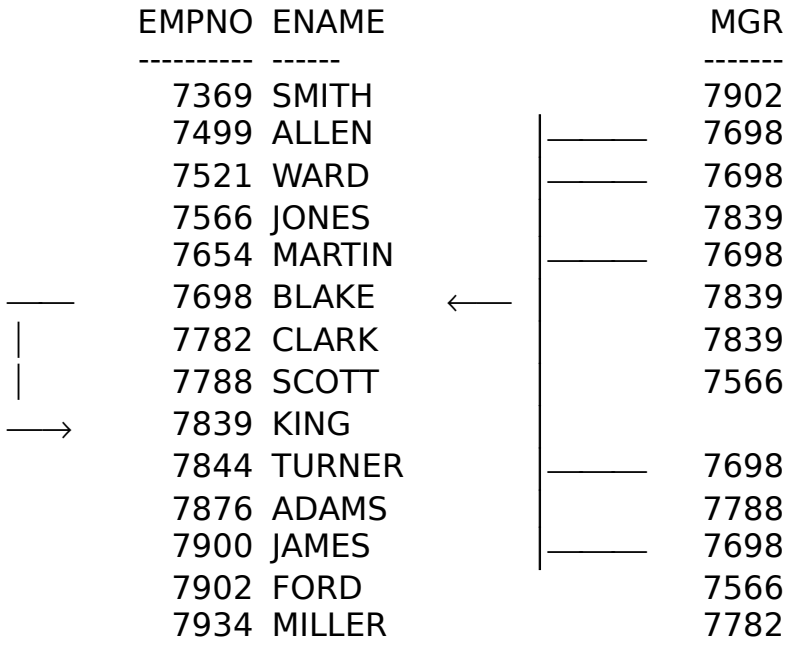
QUERY SU GERARCHIE

Struttura gerarchica

Sono tabelle in cui due o più colonne sono in rapporto reciproco e determinano l'esistenza di una correlazione "uno a molti" tra le righe della tabella stessa.

Ad esempio, nella tabella "EMP" la colonna "MGR" contiene il numero dell'impiegato che svolge il ruolo di manager nei confronti dell'impiegato cui la riga si riferisce.

```
SQL>  SELECT empno, ename, mgr
        FROM emp;
```



Oracle permette di ricostruire la gerarchia implicita tra le righe di una tabella, con una query che restituisce il livello gerarchico in cui si trova ciascuna riga.

Tale livello è restituito dalla funzione "**LEVEL**".

La modalità di scansione delle righe viene stabilita con due clausole speciali della SELECT:

START WITH Definisce la riga di partenza

CONNECT BY PRIOR Definisce il criterio con cui attribuire il valore di LEVEL di una riga rispetto alla riga precedente

Esempio:

gerarchia TOP-DOWN

```
SQL> SELECT LEVEL, ename, empno, mgr
     FROM emp
     START WITH ename='KING'
     CONNECT BY PRIOR empno = mgr;
```

←—— LEVEL superiore a chi ha empno = mgr del LEVEL corrente

LEVEL	ENAME	EMPNO	MGR
1	KING	7839	
2	JONES	7566	7839
3	SCOTT	7788	7566
4	ADAMS	7876	7788
3	FORD	7902	7566
4	SMITH	7369	7902
2	BLAKE	7698	7839
3	ALLEN	7499	7698
3	WARD	7521	7698
3	MARTIN	7654	7698
3	TURNER	7844	7698
3	JAMES	7900	7698
3	CLARK	7782	7839
3	MILLER	7934	7782

Per evidenziare la gerarchia dal basso all'alto :

```
SQL>SELECT LEVEL, ename, empno, mgr
      FROM emp
      START WITH ename='SMITH'
      CONNECT BY empno =  PRIOR mgr;
```

LEVEL	ENAME	EMPNO	MGR
1	SMITH	7369	7902
2	FORD	7902	7566
3	JONES	7566	7839
4	KING	7839	

Se interessa evidenziare la gerarchia anche visivamente, si possono usare funzioni che elaborano il "LEVEL", ad esempio:

```
SELECT LPAD(' ',LEVEL*2)||ename ORGANIGRAMMA, level, job
FROM emp
START WITH ename='KING'
CONNECT   BY   PRIOR  empno= mgr;
```

ORGANIGRAMMA	LEVEL	JOB
KING	1	PRESIDENT
JONES	2	MANAGER
SCOTT	3	ANALYST
ADAMS	4	CLERK
FORD	3	ANALYST
SMITH	4	CLERK
BLAKE	2	MANAGER
ALLEN	3	SALESMAN
WARD	3	SALESMAN
MARTIN	3	SALESMAN
TURNER	3	SALESMAN
JAMES	3	CLERK
CLARK	2	MANAGER
MILLER	3	CLERK

Questo genere di query può applicarsi a vari tipi di problemi, quali le composizioni di prodotti, strutture ad albero, ecc.
Nel caso fosse utile, la clausola "START" può usare una subquery.

Le clausole "START" e "CONNECT" possono coesistere con le altre clausole della SELECT (WHERE, ORDER BY, ecc.)
Va comunque tenuto presente che, nella scrittura di una SELECT, le clausole vanno scritte rispettando il seguente ordine:

1 - SELECT [DISTINCT] ....

2 - FROM .....

3 - [WHERE .... [subquery-SELECT] ....]

4 - [CONNECT ... | START... | GROUP BY ... | HAVING ...]

5 - [ORDER BY ..]

Le clausole allo stesso livello possono comparire in qualsiasi ordine.

8

CAPITOLO

TRATTAMENTO DELLE DATE

Generalità

Definendo una colonna con il tipo “DATE”, il suo contenuto viene considerato una data e si possono applicare tutte le funzioni ORACLE specifiche per le date.
Una colonna “DATE” occupa sette byte, ciascuno associato ad una unità temporale:

[secolo] [anno] [mese] [giorno] [ora] [minuto] [secondo]

Le date rappresentabili vanno dal 4712 A.C. fino al 4712 D.C.
Quando una colonna di tipo “DATE” viene selezionata, il suo formato standard di presentazione è sotto forma di stringa di caratteri, del tipo:

27-OCT-90

Per ottenere la data in un diverso formato esterno, si può utilizzare la funzione:

TO_CHAR(data, ‘formato’)

Dove “formato” è una sequenza di caratteri di editing tra quelli indicati alla pagina seguente.
Quando una componente della data viene presentata in forma nominale (nomi dei mesi o dei giorni), viene utilizzata la lingua nazionale prevista nella configurazione Oracle.
La funzione “SYSDATE” restituisce data e ora di sistema.
Se vengono inserite date senza specificare ore e minuti, saranno automaticamente a zero (=mezzanotte)

Editing delle date

Caratteri	Significato
CC o SCC	Secolo – Secolo con segno
SYYYY	Anno su 4 cifre con segno (es. -47)
YYYY	Anno con 4 cifre (es. 1989)
YYY	Anno su 3 cifre (es. 989)
YY	Anno su 2 cifre (es. 89)
YEAR	Nome dell'anno (es. NINETEEN-EIGHTY-NINE
Q	Trimestre dell'anno (es. 4)
MM	Mese dell'anno (es. 10)
MONTH	Nome del mese (es. JUNE)
Month	Nome del mese (es. June)
MON	Iniziali del mese (es. JUN)
Mon	Iniziali del mese (es. Jun)
WW	Settimana nell'anno (es. 51)
W	Settimana nel mese (es. 2)
DDD	Giorno nell'anno (es. 325)
DD	Giorno nel mese (es. 2)
D	Giorno nella settimana (es. 1 = SUNDAY)
DAY	Nome del giorno (es. SATURDAY)
Day	Nome del giorno (es. Saturday)
DY	Iniziali del giorno (es. SAT)
Dy	Iniziali del giorno (es. Sat)
HH O HH12	Ora del giorno (da 1 a 12)
HH24	Ora del giorno (da 1 a 24)
MI	Minuti
SS	Secondi
SSSSS	Secondi da mezzanotte (0 - 86399)
J	Giorni trascorsi dal 1-1-4713 A.C. (*)
AM o PM	Indicatori meridiani (AM o PM)
BC O AD	Indicatori A.C. D.C.

Nel formato si possono inserire, come parti costanti, tutti i caratteri non usati come caratteri di edizione (es. spazi, trattini, slash, ecc.), eventualmente racchiusi tra " " per evitare interferenze.

Esempi:

```
SQL> COLUMN  DATA_ASS   FORMAT  A10
SQL> COLUMN  GIORNO_ORA  FORMAT  A20
SQL> COLUMN  TRIM_SETT  FORMAT  A10
SQL>
SQL> SELECT ename,
            TO_CHAR(hiredate, 'DD/MM/YY')  DATA_ASS,
            TO_CHAR(hiredate, 'Day - HH24:MM')  GIORNO_ORA,
            TO_CHAR(hiredate, 'Q - WW') TRIM_SETT
       FROM emp
       WHERE deptno = 10;


       ENAME    DATA_ASS    GIORNO_ORA          TRIM_SETT
       ------   --------    ----------------    ---------
       CLARK    09/06/81    Tuesday  -  00:06   2 - 23
       KING     17/11/81    Tuesday  -  00:11   4 - 46
       MILLER   23/01/81    Friday   -   00:01   1 - 04


SQL> COLUMN  NUMERO_ASS  FORMAT  9999
SQL>
SQL> SELECT TO_CHAR(hiredate, 'YYYY')  DATA_ASS,
            COUNT(*)  NUMERO_ASS
       FROM emp
       GROUP  BY  TO_CHAR(hiredate, 'YYYY');


    ANNO_ASS    NUMERO_ASS
    --------    -----------
    1980                  1
    1981                 10
    1986                  3
```

Calcoli con le date

Quando una espressione coinvolge colonne di tipo “DATE”, Oracle opera in modo conseguente, secondo le regole:

Data + N ⟶ Data posteriore di “N” giorni

Data - N ⟶ Data anteriore di “N” giorni

Data1 - Data2 ⟶ N° di giorni tra le due date

Esempio:

lista impiegati per i quali non sono ancora trascorsi 90 giorni dall’assunzione.

```
SQL> COLUMN  DATA_SCAD  FORMAT  A12
SQL>
SQL>SELECT ename, hiredate  DATA_ASS,
                  SYSDATE  OGGI,
                  hiredate + 90   DATA_SCAD,
      FROM emp
      WHERE hiredate + 90 > SYSDATE
      ORDER BY hiredate;        ⟵ Ordine Cronologico
```

ENAME	DATA_ASS	OGGI	DATA_SCAD
SCOTT	09-APR-86	11-APR-86	12-APR-86
ADAMS	12-APR-86	11-APR-86	16-MAY-86
MILLER	15-APR-86	11-APR-86	08-JUL-86

Funzioni per le date

ADD_MONTHS(data,n) Genera la data posteriore di “n” mesi rispetto a quella contenuta in “data” (anteriore se “n” ha valore negativo)
Esempio: ADD_MONTH(hiredate,12) ⟶ + 1 anno

MONTH_BETWEEN(d1,d2) Ritorna il numero di mesi tra le due date “d1” e “d2” (positivo se “d1” è posteriore a “d2”)
Esempio: MONTH_BETWEEN(‘02-FEB-86’,’01-GEN 86’)⟶ 1.03225806

LAST_DAY(data) Ritorna la data dell’ultimo giorno del mese contenuto in “data”
Esempio: LAST_DAY(‘02-FEB-88’) ⟶ ‘29-FEB-88’

NEXT_DAY(data,giorno) Sposta la data contenuta in “data” al successivo “giorno” specificato.
Esempio: NEXT_DAY(‘17-MAR-89’,’TUESDAY’) ⟶ ‘24-MAR-89’

TRUNC(data [, formato]) Ritorna la data contenuta in “data” limitatamente alla porzione specificata dal “formato” indicato.
DEFAULT: toglie ore, minuti e secondi
Esempio: SYSDATE = 19-MAR-90: TRUNC(SYSDATE,’YYYY’) ⟶ 01-JAN-90

ROUND(data, formato) Arrotonda la data contenuta in “data” secondo il “formato” indicato.
Esempio: ROUND(SYSDATE,’YYYY’) ⟶ 01-JAN-90

Esempio:

```
SQL> SELECT ename, hiredate+90-SYSDATE  GIORNI_RESIDUI
     FROM emp
     WHERE hiredate+90 > SYSDATE;
```

ENAME	GIORNI_RESIDUI
SCOTT	.8888
ADAMS	34.89
MILLER	87.89

```
SQL>SELECT ename,
          TRUNC(hiredate+90) - TRUNC(SYSDATE)GIORNI_RESIDUI
  FROM emp
  WHERE hiredate+90 > SYSDATE;
```

ENAME	GIORNI_RESIDUI
SCOTT	1
ADAMS	35
MILLER	88

Date come caratteri o numeri

Qualora una data fosse memorizzata in una colonna di tipo carattere o numerico, è possibile beneficiare di quanto visto per la manipolazione delle date, purché la colonna venga convertita al formato “DATE” con la funzione:

TO_DATE(numero | stringa , ‘formato’)

Dove ‘formato’ usa gli stessi caratteri di editing visti per la “TO_CHAR”, ma allo scopo di specificare come deve essere interpretato il contenuto del dato numerico o stringa per trasformarlo in data.
Esempio:

```
SQL>SELECT   TO_DATE(10199006, ‘MMYYYYDD’)  DATA1,
                TO_DATE(‘901020’, ‘YYMMDD’)  DATA2
      FROM  dual;

      DATA1          DATA2
      ---------------  ---------------
      06-OCT-90    20-OCT-90

SQL>SELECT
          TO_DATE(‘901020’,‘YYMMDD’) - TO_DATE
                   (10199006,’MMYYYYDD’)  DG
      FROM  dual;

      DG
      ----
      14

SQL>SELECT
      TO_CHAR(TO_DATE(10199006, ‘MMYYYYDD’),
                ‘DD-MM-YYYY’) DATA1
      FROM  dual;

      DATA1
      ----------
      06-10-1990
```

9

CAPITOLO

MANIPOLAZIONE DEI DATI

Aspetti generali

Sono disponibili tre istruzioni SQL per modificare il contenuto del database:

INSERT Per inserire nuove righe in una tabella.

UPDATE Per modificare il contenuto di una o più righe di una tabella.

DELETE Per eliminare una o più righe da una tabella.

L'esecuzione di queste istruzioni non produce effetti definitivi sul database fino a che non viene eseguita l'istruzione SQL:

COMMIT

Fino a che non viene richiesto il "COMMIT", tutte le modifiche sono provvisorie e visibili solo all'utente che le ha eseguite.
Questo permette di eseguire più operazioni che costituiscono un'unità logica di lavoro (transazione) e di convalidarle nel loro insieme.

Se le modifiche per le quali non è stato ancora richiesto "COMMIT" devono essere invece annullate, occorre richiedere:

ROLLBACK

Il "ROLLBACK" annulla tutte le modifiche effettuate dall'utente dopo l'ultimo "COMMIT", o dall'inizio del lavoro se non è mai stato richiesto un "COMMIT".

Dalla versione 6 in poi di Oracle , si possono assegnare specifici punti di ripristino per successivi "ROLLBACK", con l'istruzione SQL:

SAVEPOINT nome-savepoint

Si potrà ritornare al punto in cui era stata eseguita una tale “SAVEPOINT” con un’istruzione:

ROLLBACK TO SAVEPOINT nome-savepoint

In SQL*Plus è possibile far richiedere automaticamente il “COMMIT” dopo ogni istruzione “INSERT”, “UPDATE”, “DELETE”, usando la direttiva:

SET AUTOCOMMIT ON|OFF

dove:
ON = Attivazione autocommit
OFF = Disattivazione autocommit

Blocchi PL/SQL

Dalla Versione 6 in poi di Oracle, è stata introdotta una estensione al linguaggio SQL, detta “PL/SQL”, che permette di codificare più istruzioni SQL entro un unico blocco strutturato con costrutti procedurali (IF, ELSE, FOR, ecc.) e variabili locali.
La sintassi è derivata dal linguaggio ADA.
Un blocco PL/SQL viene elaborato con una singola richiesta al KERNEL Oracle, aumentando l’efficienza dell’applicazione.
Un blocco PL/SQL inizia con un “DECLARE”, “BEGIN” o un nome-blocco.
Il “;” non ha più un significato abituale in SQL*Plus, poiché é utilizzato dalla sintassi del PL/SQL.
In SQL*Plus un blocco PL/SQL viene terminato con una riga finale contenente solo “.” Per la sua esecuzione si usa la direttiva “RUN” o “/”.
Esempio:

```
SQL>    DECLARE
            avg_sal         NUMBER(7,2);
        BEGIN
            SELECT avg(sal) INTO avg_sal FROM emp
                WHERE job <> ‘PRESIDENT’;
            IF avg_sal < 3000  THEN
                UPDATE emp SET sal =  sal + 50
                    WHERE job <> ‘PRESIDENT’;
                INSERT  INTO sal_history(avg_date,
                                avg_value)
                    SELECT sysdate, avg(sal)
                    FROM emp
                    GROUP BY  sysdate;
            ELSE
                INSERT INTO sal_history(avg_date,
                                avg_value)
                    VALUES (sysdate, avg_sal);
            END IF;
            COMMIT;
        END;
        .
SQL>    /
PL/SQL procedure successfully completed.
```

Utilizzo concorrente dei dati

Quando più utenti o programmi manipolano contemporaneamente i dati di una medesima tabella, Oracle provvede automaticamente ad impedire ad un utente di modificare i dati che sono in corso di aggiornamento da parte di un altro utente.
Quando viene eseguita una “UPDATE”, ”DELETE”, o “INSERT”, le righe interessate da tali operazioni vengono bloccate fino al “COMMIT” o “ROLLBACK” delle transazioni effettuate.
Le righe bloccate continuano però ad essere accessibili in query, mostrando lo stato relativo all’ultimo COMMIT valido.
Dopo il COMMIT le righe modificate divengono visibili nelle loro nuova situazione e aggiornabili da parte di altri utenti o programmi.
Oracle garantisce inoltre la consistenza delle query, nel senso che durante tutto lo svolgimento di una SELECT, se una riga viene esaminata più volte, essa appare sempre nello stato in cui era all’inizio della query.
Eventuali modifiche intervenute durante l’espletamento della query non vengono percepite al suo interno.
In talune circostanze può risultare utile cambiare la gestione standard dei lock, ad esempio se si desidera:

1. Bloccare preventivamente le righe su cui si intende fare successive modifiche.
2. Bloccare preventivamente l’intera tabella per svolgere modifiche su tutte o gran parte delle sue righe.
3. Garantire che una query possa ritornare dati su cui non sono pendenti modifiche.

Nel caso 1. si può effettuare una “SELECT” delle righe interessate con la speciale clausola

“FOR UPDATE”.

Ad esempio:

```
SQL>SELECT empno FROM emp
      WHERE deptno = 30
      FOR UPDATE OF sal, comm NOWAIT;
```

La “FOR UPDATE” deve precisare il nome di almeno una colonna che si intende modificare e l’opzione “NOWAIT” se si vuole che la “SELECT” termini con un messaggio di errore, anziché rimanere in attesa, nel caso esista qualche LOCK attivo sulle righe coinvolte.
Anche se poi non si effettuano modifiche, occorre eseguire “COMMIT” o “ROLLBACK” per sbloccare le righe.

Nei casi 2. e 3. si può utilizzare l'istruzione "LOCK TABLE" per richiedere un LOCK esclusivo sull'intera tabella.
Ad esempio:

```
SQL> LOCK TABLE emp IN EXCLUSIVE MODE NOWAIT;
```

L'opzione "NOWAIT" ha lo stesso significato visto in precedenza un successivo "COMMIT" o "ROLLBACK" sbloccherà la tabella.

10

CAPITOLO

GESTIONE TABELLE

Aspetti generali

La gestione delle tabelle, intesa come creazione e controllo delle strutture fisiche che ospitano i dati, si avvale delle seguenti istruzioni SQL:

CREATE TABLE	Per creare una nuova tabella
ALTER TABLE	Per modificare la struttura di una tabella
DROP TABLE	Per eliminare una tabella
RENAME	Per cambiare il nome ad una tabella
CREATE SYNONYM	Per creare un nome alternativo
CREATE INDEX	Per creare un indice
DROP INDEX	Per eliminare un indice

Per creare una tabella, l'utente deve avere un privilegio di "RESOURCE" nel database, assegnatogli dal DB administrator.

Una volta creata, una tabella è visibile e manipolabile solo dall'utente che l'ha creata, il quale , tuttavia può concedere ad altri utenti specifici diritti di accesso (GRANTS).

Chi accede ad una tabella altrui, deve far precedere il nome del proprietario al nome della tabella, ad esempio:

```
SQL> SELECT * FROM scott.emp;
```

L'utente che riceve un diritto su una tabella altrui può crearsi un sinonimo, al fine di non dover specificare sempre il nome del proprietario della tabella, ad esempio:

```
SQL> CREATE SYNONYM emp FOR scott.emp;

SQL> SELECT * FROM emp;
```

Creazione di una tabella

L’istruzione “CREATE TABLE” ha il seguente formato di base:

```
CREATE  TABLE  nome_tabella
      (colonna-1 datatype(size)  [NOT  NULL],
      colonna-2 datatype(size)  [NOT  NULL],
      ..........................                    )
      [ AS select-statement ]
```

I nomi delle tabella e colonna devono iniziare con una lettera alfabetica e possono contenere lettere, numeri ed i segni “$” e “_”.

Gli stessi nomi di tabella possono essere usati da utenti diversi e gli stessi nomi colonna in tabelle diverse, anche del medesimo utente.

La clausola “AS select-statement” permette di creare una tabella con colonne dello stesso tipo e lunghezza di quelle specificate nella SELECT. I nomi delle colonne possono essere presi da quelle usate nella select oppure fornite esplicitamente.

Dopo la creazione, la tabella viene anche caricata con le righe estratte dalla SELECT, se ve sono.
Questo formato consente di allocare la tabella con caratteristiche fisiche di default, generalmente adeguate per tabelle di media dimensione.

Ulteriori opzioni legate a particolari modalità di allocazione saranno illustrate nel corso di administration.

Esempio:

```
SQL> CREATE  TABLE proj
                  (PROJNO     NUMBER(2)  NOT NULL,
                  PNAME       CHAR(10),
                  BUDGET     NUMBER(8,2));

Table created.
```

Tipi di dati per le colonne

Per ogni colonna va specificato il tipo di dati contenuti (Formato interno al DB) e la dimensione massima dei dati.

Oracle memorizza i dati solo per lo spazio strettamente necessario e non per la lunghezza massima specificata.

I tipi di dato utilizzati da Oracle sono:

Tipo colonna	Max size	Può ricevere
varchar2(size)	2000	Stringa di caratteri ASCII/EBCDIC, di lunghezza variabile con lunghezza max. = *size*
number(p,s)	38	1.0*10E-130 and 9.9..9*10E+125
Long	2Gbyte	dato di tipo carattere di lunghezza variabile fino a 2 Gbyte
Date	7	date comprese nel range 01/01/4712 B.C. e 31/12/4712 A.D.
Raw(size)	255	Raw binary data di lunghezza = *size* e max. = 255
Long row	2Gbyte	Raw binary data di lunghezza variabile fino a 2 Gbyte
Rowid		Stringa esadecimale rappresentante l'indirizzo univoco dei record
Char(size)	255	Stringa di caratteri ASCII/EBCDIC, di lunghezza fissa e max. = 255 (default = 1)
Mlslabel		tipo di dato in formato binario

Esempi:

varchar2(30)	Fino a 30 caratteri (“SALES” occupa 4 byte)
number(5)	Numero intero fino 5 cifre
number(8,2)	Numero di 8 cifre, 6 intere e 2 decimali
number(8,-2)	Numero di 8 cifre arrotondato alle centinaia
char(40)	Fino a 40 caratteri (“SALES” occupa 40 byte)

Al momento della creazione della tabella, Oracle accetta anche tutti i tipi di dato previsti dallo standard SQL e da DB2 (es. NUMERIC, DECIMAL, FLOAT, ecc.) salvo convertirli comunque nei formati sopra indicati.

L’opzione “NOT NULL” obbliga ad inserire un valore nella colonna, impedendo l’inserimento o aggiornamento della riga in caso contrario.

Modifica di struttura di una tabella

Quando una tabella è già stata creata e caricata con i dati, si può comunque modificarne alcuni aspetti di struttura con l'istruzione "ALTER TABLE".

La "ALTER TABLE" presenta due opzioni:

- **ALTER TABLE nome tabella MODIFY …** Per modificare le caratteristiche di una colonna
- **ALTER TABLE nome tabella ADD …** Per aggiungere delle colonne

Nel caso di aggiunta di nuova colonna, questa si ritroverà con valori "NULL" in tutte le righe della tabella, sicché non si può aggiungere una colonna con opzione "NOT NULL", a meno che non si specifichi un valore di default.

Esempi:

```
SQL>ALTER TABLE proj
      MODIFY (budget  NUMBER(9,2));

SQL>ALTER TABLE emp
      ADD( projno NUMBER(2));
```

Indici

Un indice è una struttura fisica addizionale, completamente invisibile all'utente, che ha due utilizzi principali:

1. Velocizzare gli accessi ai dati su tabelle medio-grandi.

2. Garantire l'unicità delle chiavi primarie.

Si possono creare più indici sulla stessa tabella per ottimizzare differenti tipi di ricerche (es. su "EMPNO" e su "ENAME").

Un indice può essere creato o eliminato quando si desidera, prima o dopo il caricamento di dati nella tabella (all'inizio andrà creato subito l'indice per la chiave primaria della tabella).

Come chiave per un indice si potranno utilizzare una colonna o una combinazione di colonne della tabella (in ordine qualsiasi), anche se già utilizzate in altri indici.

Il formato base per la creazione di un indice è:

CREATE [UNIQUE] INDEX nome-indice
ON nome-tabella (col1-1 (col-2]...)

Dove "col-1", "col-2", ... sono le colonne che compongono la chiave associata all'indice. L'ordine di specificazione determina il peso delle diverse componenti della chiave.
L'opzione "UNIQUE" impedisce l'inserimento di righe aventi valori duplicati nella colonne costituenti la chiave associata all'indice.

Esempi:

```
SQL> CREATE UNIQUE INDEX ixemp01 ON emp(empno);

SQL> CREATE INDEX ixemp02 ON EMP (deptno,ename);
```

Un indice sarà sfruttato da Oracle durante una select, solo se la prima o unica colonna della sua chiave viene coinvolta nelle condizioni presenti nella "WHERE"

Esempi:

SQL>SELECT * FROM emp WHERE empno > 7300 (Usa indice IXEMP01)

SQL>SELECT * FROM emp WHERE deptno = 20 (Usa indice IXEMP02)

SQL>SELECT * FROM emp WHERE ename = ‘WARD’ (non usa indici)

Per eliminare un indice si usa l’istruzione SQL:

DROP INDEX nome-indice;

11

CAPITOLO

VIEWS

Aspetti generali

Una view è una tabella fittizia corrispondente ad una QUERY, predisposta al fine di fornire una visione alternativa dei dati presenti nel database.
Una view non contiene dati, ma appare ed è manipolabile come una tabella reale. Può essere conveniente per ragioni di:

- SICUREZZA — Una view può restringere l'accesso solo ad alcune righe e/o colonne di una tabella.
- SEMPLICITA' vista facile da — Una interrogazione complessa può essere come una semplice tabella, più manipolare.

Una VIEW è definita mediante una query, secondo la sintassi:

CREATE VIEW nome-view [(viewcol-1, ... , viewcol-n)]
AS SELECT [DISTINCT] dato-1, ... , dato-n
FROM .....
[WHERE .....] [GROUP BY ....] [HAVING .....]
[UNION ..... | INTERSECT ..... | MINUS .....]
[WITH CHECK OPTION]

In pratica, la query che definisce la view può sfruttare tutte le possibilità e clausole viste in precedenza, tranne la "ORDER BY".
Se non vengono forniti i nomi delle colonne della view, essa eredita quelli delle colonne selezionate nella query.
Se la query ritorna dati corrispondenti a espressioni o costanti, i nomi delle colonne della view devono essere specificate.
La clausola "WITH CHECK OPTION" è significativa quando la view è usata anche per fare inserimenti/aggiornamenti di righe nella tabella reale su cui è definita.
Essa impedisce di inserire o aggiornare righe che non sono poi visibili attraverso la view stessa.

Esempi di views

1. Creazione di una view per accedere ad un sottoinsieme di righe e colonne della tabella “EMP”.

```
SQL>CREATE  VIEW emp10
            AS SELECT empno, ename, job
                  FROM emp
                  WHERE deptno = 10;
```

View created.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7839	KING	PRESIDENT		17-NOV-81	5000		10
7934	MILLER	CLERK	7782	23-JAN-81	1300		10
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7788	SCOTT	ANALYST	7566	09-DEC-81	3000		20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

La “SELECT” che definisce la view verrà eseguita quando si andrà ad interrogare la view stessa, ad esempio con:

```
SQL> SELECT * FROM emp10 WHERE empno > 7800
```

EMPNO	ENAME	JOB
7839	KING	PRESIDENT
7934	MILLER	CLERK

2.Una view che presenta dati raggruppati (stipendi medi di ciascun dipartimento)

```
SQL>CREATE VIEW avgsal (deptno, dept_avg_sal)
      AS SELECT deptno, AVG(sal)
      GROUP BY deptno;
```

View created

```
SQL>SELECT * FROM avgsal;
```

DEPTNO	DEPT_AVG_SAL
10	2917
20	2175
30	1567

La view può essere utilizzata in un Join con la tabella ‘EMP’ al fine di raffrontare stipendi individuali e medi:

```
SQL>SELECT ename, sal, dept_avg_sal
      FROM emp, avgsal
      WHERE emp.deptno = avgsal.deptno;
```

ENAME	SAL	DEPT_AVG_SAL
CLARK	2450	2917
MILLER	1300	2917
KING	5000	2917
SMITH	800	2175
SCOTT	3000	2175
JONES	2975	2175
ADAMS	1100	2175
FORD	3000	2175
ALLEN	1600	1567
BLAKE	2850	1567
TURNER	1500	1567
JAMES	950	1567
MARTIN	1250	1567
WARD	1250	1567

3.Creazione di una view che unifica i dati di sotto-tipi di un’entità (venditori + analisti)

```
SQL>CREATE  VIEW  sal_ana(nome, mansione)
              AS SELECT ename, job FROM salesman
                   UNION
                       SELECT ename,j ob FROM analysts
       /
```

View created.

```
SQL>  SELECT * FROM sal_ana;
```

NOME	MANSIONE
ALLEN	SALESMAN
FORD	ANALYST
MARTIN	SALESMAN
SCOTT	ANALYST
TURNER	SALESMAN
WARD	SALESMAN

4.Creazione di una view per mascherare un Join.

```
SQL> CREATE  VIEW  empdep (empno, ename, deptno, dname)
              AS SELECT empno, ename, emp.deptno, dname
                    FROM emp, dept
                    WHERE emp.deptno = dept.deptno;
```

View created.

```
SQL>  SELECT * FROM empdep WHERE dname = 'SALES';
```

EMPNO	ENAME	DEPTNO	DNAME
7499	ALLEN	30	SALES
7521	WARD	30	SALES
7654	MARTIN	30	SALES
7698	BLAKE	30	SALES
7844	TURNER	30	SALES
7900	JAMES	30	SALES

Aggiornamenti mediante views

Restrizioni sull'uso delle views per aggiornare i dati:

- Views definite con colonne virtuali o derivate da più tabelle (Join, union, ecc.) possono essere utilizzate solo per eseguire SELECT.

- Non si possono eseguire INSERT se la view non contempla colonne dichiarate NOT NULL nella tabella sottostante (Oracle lascia a NULL le colonne non visibili nella view)

Esempio:

```
UPDATE  emp10
        SET  JOB = 'CLERK'
        WHERE ENAME = 'MILLER'
```

Esempio di creazione Vista e suo utilizzo in inserimento, aggiornamento e cancellazione:

```
create view v1 as select empno,ename,job,deptno from emp;

insert into v1 values (1000,'Rossi','CLERK',20);
(le sole colonne NOT NULL sono empno e deptno)

update v1 set ename = 'Bianchi' where ename ='Rossi';

delete from v1 where ename ='Bianchi';
```

Così, attraverso le viste possiamo "**granulare**" la sicurezza sui dati:

Esempio:
Vogliamo permettere ad un altro utente la visibilità delle colonne definite nella vista V1, ma l' aggiornamento della sola colonna JOB.

```
grant select on v1 to capo_scott;

grant update (job) on v1 to capo_scott ;


connect capo_scott/<password>@<alias_tnsnames.ora>

select * from scott.v1;

EMPNO ENAME      JOB          DEPTNO
----- ---------- --------- ---------
```

```
 7839 KING       PRESIDENT         10
 7698 BLAKE      MANAGER           30
 7782 CLARK      MANAGER           10
 7566 JONES      MANAGER           20
 7654 MARTIN     SALESMAN          30
 7499 ALLEN      SALESMAN          30
 7844 TURNER     SALESMAN          30
 7900 JAMES      CLERK             30
 7521 WARD       SALESMAN          30
 7902 FORD       ANALYST           20
 7369 SMITH      CLERK             20
 7788 SCOTT      ANALYST           20
 7876 ADAMS      CLERK             20
 7934 MILLER     CLERK             10
 1000 Rossi      Operaio           20
```

```
update scott.v1 set job='Operaio' where ename ='Rossi';

1 row updated.

update scott.v1 set ename='Bianchi' where job='Operaio';
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

12

CAPITOLO

SECURITY DEI DATI

Assegnazione dei GRANTS

L'utente che crea una tabella o una view, ne diviene il Proprietario a tutti gli effetti, nel senso che a lui solo è possibile manipolarne i dati o la struttura.
Al tentativo di accedere ad una risorsa per la quale non si hanno diritti , Oracle risponde con un messaggio di errore del tipo:

ORA-00942: table or view does not exist

Il proprietario (OWNER) della risorsa può però concedere ad altri utenti il diritto di svolgere determinate operazioni sulla risorsa stessa, mediante l'istruzione SQL:

GRANT tipo-operazione
ON nome_tabella| nome view
TO nome-utente

I tipi di operazione che possono essere abilitati sono:

SELECT	
INSERT	
UPDATE	
DELETE	
ALTER	(solo per le tabelle)
INDEX	(solo per le tabelle)
REFERENCES	(solo per le tabelle)

Il diritto di eseguire il "DROP" della risorsa non è invece trasferibile a utenti diversi da chi ne ha eseguito la "CREATE".
In una stessa "GRANT" si possono abilitare più tipi di operazioni su una certa risorsa (tabella o view) a favore di più utenti.

Esempio:

```
SQL>GRANT SELECT, UPDATE
      ON  dept
      TO  adams, jones;
```

L'operazione di "UPDATE" può essere abilitata limitatamente ad alcune colonne.
Ad esempio:

```
SQL> GRANT  SELECT, UPDATE (hiredate, deptno)
      ON emp
      TO  adams;
```

Di conseguenza, l'utente "ADAMS" potrà modificare solo le colonne "HIREDATE" e "DEPTNO" nella tabella "EMP".
Non è possibile fare lo stesso per l'operazione di "SELECT", ma si può ovviare tramite una view che mostra solo le colonne volute.
Si possono concedere tutti i diritti con l'opzione "ALL".
Ad esempio:

```
SQL>GRANT ALL
      ON projects
      TO adams;
```

Si può anche assegnare un grant generalizzato a tutti gli utenti presenti e futuri indicando come nome utente "PUBLIC".
Ad esempio:

```
SQL>GRANT ALL
ON projects
TO  PUBLIC;
```

Si può fare in modo che l'utente beneficiario di un GRANT possa a sua volta estenderlo ad altri utenti ancora. A tale scopo va utilizzata la "GRANT OPTION" da parte del proprietario.
Ad esempio:

```
SQL>GRANT SELECT
      ON projects
      TO adams
      WITH  GRANT OPTION;
```

Annullamento dei GRANTS

I grants possono essere annullati con l'istruzione SQL:

REVOKE tipo-operazione
ON nome_tabella| nome view
FROM nome-utente

Le eventuali catene di grants generate dalla "GRANT OPTION" vengono automaticamente annullate.

Esempio:

SQL> REVOKE UPDATE
ON projects
FORM adams;

Nomi e sinonimi

L'utente che riceve un diritto su una tabella, deve referenziarla premettendo il nome del proprietario a quello della tabella.
Ad esempio l'utente "ADAMS" potrà accedere alla tabella "EMP" di "SCOTT" con sintassi del tipo:

SELECT * FROM scott.emp;

L'utente che riceve diritti di accesso su tabelle altrui può definire propri 'sinonimi' per tali tabelle con l'istruzione:

CREATE SYNONYM nome-sinonimo FOR nome-tabella\nome-view

Esempio (utente "ADAMS") :

```
SQL> CREATE SYNONYM emps  FOR scott.emp;
SQL>
SQL> SELECT * FROM emps;
```

Inoltre l'amministratore del database può creare sinonimi di tipo 'pubblico', ossia utilizzabile da tutti gli utenti, con la:

CREATE PUBLIC SYNONYM nome-sinonimo
FOR nome-tabella | nome-view

13

CAPITOLO

ESERCIZI

Database per gli esercizi

Tabella dipartimenti “DEPT“

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Tabella dipendenti “EMP”

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-81	1300		10

Tabella progetti “PROJECTS“

PROJNO	PNAME
1	ORACLE DEVELOPMENT
2	DOCUMENTATION
3	CUSTOMER TRAINING
10	TESTING
20	INSTALLATION
30	QUALITY CONTROL

Tabella assegnazioni “ASSIGNEMENT“

EMPNO	PROJNO	STARTDATE
7369	1	17-DEC-80
7369	2	01-JAN-81
7499	1	20-FEB-81
7521	10	22-FEB-81
7521	20	15-MAR-81
7521	30	15-MAR-81
7566	1	02-APR-81
7654	2	01-OCT-81
7698	10	01-MAY-81
7782	20	10-JUN-81
7788	10	09-NOV-81
7839	2	20-NOV-81
7844	3	12-SEP-81
7876	10	23-SEP-81
7900	20	04-DEC-81
7902	1	04-DEC-81
7934	1	10-JAN-81
7934	2	10-JAN-81
7934	3	22-APR-81
7934	10	23-JAN-81

Esercizi relativi ai capitoli 2/3/4/5/6

1. Visualizzare il risultato del calcolo 23*10/5 appoggiandosi alla tabella dual.

2. Visualizzare i distinti numeri dipartimento (DEPTNO) presenti nelle righe della tabella EMP.

3. Selezionare tutti i dati dei dipendenti che hanno matricola (EMPNO) minore di quella del proprio manager (MGR), oppure non hanno manager, ordinando per numero dipartimento (DEPTNO) crescente e stipendio (SAL) decrescente.

4. Visualizzare tutti i dati dei dipendenti che hanno mansione(JOB) uguale a “SALESMAN” e percepiscono uno stipendio (SAL) compreso fra 1500 e 2000 $.

5. Selezionare tutti i dati dei dipendenti il cui nome (ENAME) finisce con “S” e presenta una “A” come terza lettera.

6. Visualizzare ENAME, JOB, SAL, e la percentuale COMM/SAL*100 per tutti i dipendenti che hanno COMM non nulla, ordinando in modo decrescente sul valore della suddetta percentuale.

7. Stesso risultato dell’esercizio 6. con la percentuale arrotondata all’intero.

8. Visualizzare ENAME, JOB, e una colonna MASSIMO, contenente il maggiore tra i valori di SAL e COMM del dipendente.

9. Listare i nomi (ENAME) dei dipendenti, ordinati per lunghezza decrescente del nome stesso e, a parità di lunghezza, in modo alfabetico.

Esempio:

```
ENAME
----------
MARTIN
TURNER
ADAMS
SMITH
FORD
WARD
```

10. Listare i nomi (ENAME) dei dipendenti e una colonna ISTOGRAM contenente un numero di “*” pari alla lunghezza del nome che si trova in ENAME.

Esempio:

```
ENAME      ISTOGRAM
---------- ----------
SMITH      *****
WARD       ****
ADAMS      *****
MARTIN     ******
```

11. Listare ENAME, JOB, EMPNO per tutti i dipendenti, allineando a destra il contenuto della colonna ENAME, su 10 caratteri.

Esempio:

```
ENAME      JOB             EMPNO
---------- --------------- ----------
     SMITH CLERK            7369
    MARTIN SALESMAN         7654
      FORD ANALYST          7902
```

12. Visualizzare, in una sola colonna (v. esempio) il nome (ENAME) e la località di lavoro (LOC) di tutti i dipendenti.

Esempio:

```
SEDE_DI_LAVORO
----------------------------
CLARK - NEW YORK
FORD – DALLAS
MARTIN – CHCAGO
```

13. Visualizzare per ogni dipartimento i progetti ad esso affidati e i nomi dei dipendenti cui sono assegnati, ordinati per nome dipartimento.
Suggerimento: Si tratta di 3 **Equi Join**.

Esempio:

DNAME	PNAME	ENAME
RESEARCH	ORACLE DEVELOPMENT	SMITH
RESEARCH	DOCUMENTATION	SMITH
SALES	ORACLE DEVELOPMENT	ALLEN
SALES	TESTING	WARD

14. Selezionare il numero dipartimento (DEPTNO) e descrizione (DNAME) dei dipartimenti privi di dipendenti.
Suggerimento: Si può effettuare con **Outer Join MINUS Equi Join,** oppure con **Outer Join** e selezione per ENAME IS NULL.

Esempio:

DEPTNO	DNAME
40	OPERATIONS

15. Visualizzare, come in esempio, i dati dei dipendenti assunti in data (HIREDATE) anteriore a quella del loro manager.
Suggerimento: Si tratta di un **Self Join**.

Esempio:

N_DIP	NOME_DIP	DATA_DIP	N_MGR	NOME_MGR	DATA_MGR
7499	ALLEN	20-FEB-81	7698	BLAKE	01-MAY-81
7521	WARD	22-FEB-81	7698	BLAKE	01-MAY-81
7566	JONES	02-APR-81	7839	KING	17-NOV-81

16. Selezionare DEPTNO e DNAME dei dipartimenti in cui lavora almeno un dipendente con la qualifica di analista (‘ANALYST’).
Suggerimento: Può essere utilizzata una Query con Subquery oppure EquiJoin con selezione.

17. Creare un file .SQL, da richiamarsi con la direttiva “START”, che produce il risultato dell’es. 16 per qualsiasi valore di qualifica introdotto interattivamente dall’utente.
Suggerimento: Nel caso di una Query con Subquery, attenzione al ritorno di “troppe righe” (too many rows), quindi utilizzare = ANY (… ..) o IN (……).

18. Fare in modo che il file .SQL dell'es. 17 venga eseguito immediatamente all'attivazione di SQLPLUS, fornendo la qualifica nel comando di richiamo di SQLPLUS.
Suggerimento: Per NT, vi sono 2 files che vengono automaticamente eseguiti ad ogni nuova finestra SQLPlus, il primo ad essere eseguito è %ORACLE_HOME%\PLUSxx\GLOGIN.SQL, mentre il secondo è %ORACLE_HOME%\DBS\LOGIN.SQL. Inserendovi alcuni nostri Statements ……

19. Listare ENAME e SAL dei dipendenti il cui stipendio è inferiore a quello medio del loro dipartimento.

20. Visualizzare i numeri che compaiono sia come numeri dipartimento (DEPTNO) che come numeri progetto (PROJNO).
Esempio:

```
NUMERO
----------
        10
        20
        30
```

Esercizi relativi ai capitoli 9/10/11

1. Settare AUTOCOMMIT OFF e cancellare dalla tabella EMP le righe degli impiegati che lavorano nel dipartimento 20, senza dare COMMIT. ***Suggerimento:*** Ricordarsi di disabilitare prima la Foreign Key di ASSIGNEMENT riferita alla tabella EMP, oppure di indicare su di essa l'opzione ON DELETE CASCADE.

2. Verificare che le righe cancellate siano scomparse dalla tabella EMP.

3. Eseguire il ROLLBACK dell'operazione precedente e verificare che i dati degli impiegati del dipartimento 20 siano stati ripristinati nella tabella EMP.

4. Aumentare del 12% lo stipendio (SAL) ai dipendenti aventi la qualifica (JOB) uguale a "ANALYST". Verificare la modifica e dare COMMIT.

5. Inserire un nuovo dipartimento nella tabella DEPT, con numero dipartimento usuale a "50", descrizione usuale a "MARKETING" e località usuale a "ATLANTA". Verificare l'inserimento dei nuovi dati e dare il COMMIT.

6. Preparare un file .SQL contenente l'istruzione di INSERT parametrizzata con le variabili di sostituzione in luogo dei valori tale da consentire l'inserimento di nuove righe nella tabella EMP.

7. Impostare l'autocommit immediato.

8. Utilizzare il file preparato al punto 6. per inserire i dati di due nuovi impiegati attribuiti al dipartimento "50".

9. Modificare la struttura della tabella ASSIGNMENT, aggiungendo una nuova colonna di nome DURATA, di formato numerico intero su 4 cifre.

10. Per ogni riga della tabella ASSIGNMENT, valorizzare la colonna DURATA con il numero di giorni trascorsi dalla data presente nella colonna STARTDATE sino ad oggi.

11. Aumentare a 5 cifre la dimensione della colonna DURATA nella tabella ASSIGNEMENT.

12. Creare una tabella di nome “MANAGERS”, con le colonne:

MATRICOLA	Numerica intera di 4 cifre
NOME	Alfanumerica di 16 caratteri
STIPENDIO	Numerica di 7 cifre con 2 decimali
DIPARTIMENTO	Numerica intera di 2 cifre

13. Creare un indice univoco sulla tabella MANAGERS, avente come chiave il contenuto della MATRICOLA.

14. Inserire nella tabella MANAGERS i dati relativi ai dipendenti con qualifica uguale a “MANAGER”, presenti nella tabella EMP, secondo la corrispondenza delle colonne: MATRICOLA=EMPNO, NOME=ENAME, STIPENDIO=SAL*12, DIPARTIMENTO=DEPTNO.

15. Verificare il funzionamento dell’indice, tentando di inserire nella tabella MANAGERS una riga con valore di MATRICOLA già presente.

16. Creare una view che permetta di vedere, come se si trovassero in un’unica tabella, i seguenti dati:

DNAME	ENAME	PNAME	DURATA

Eseguire una SELECT di tutta la view.

17. Creare una view che permetta di vedere gli stipendi medi pagati in ciascun dipartimento.

Esempio:

DEPTNO	SALMED
10	2917
20	2175
30	1567

Eseguire una SELECT di tutta la view.

18. Utilizzare la view creata al punto precedente per visualizzare i dati degli impiegati che percepiscono uno stipendio minore dello stipendio medio del loro dipartimento, evidenziando:

DEPTNO	ENAME	SAL	SALMED
10	CLARK	2450	2917
10	MILLER	1300	2917

19. Listare per ogni dipartimento la percentuale dei salari in esso pagati, rispetto al totale stipendi aziendale.
Esempio:

DEPTNO	SALDEP	PERCENT	SALTOT
10	8750	30.10	29025
20	10875	37.50	29025
10	8750	32.40	29025