Laboratorio di Basi di Dati	**Matricola:**	FILA
Appello del 10/11/2014 (100 minuti)	**Nome e cognome:**	A
Service:	Login: esame___ Password: _____	
❍ Oracle8i ESAMESI_SI-ORACLESRV01		
❍ Oracle11g ESAMESI_ si-oracle-11.csr.unibo.it		

1) Il sistema automatico di prenotazione dei posti di TicketTwo si basa sul seguente database.

EVENTI(ID, Titolo, Data, Struttura:STRUTTURE, Tipologia, DisponibilitàResiduaPosti)

STRUTTURE(IDStruttura,Indirizzo, Tipo, CapienzaTotale)

DISPONIBILITA(Evento:EVENTI, Fila, NumPoltronaIniziale, NumPosti)

Per ogni evento gestito da TicketTwo si memorizza la struttura in cui l'evento si svolgerà e le disponibilità dei biglietti. Queste ultime sono memorizzate come gruppi di posti liberi contigui. Quindi la tupla di **DISPONIBILITA** con valori (1, 2, 10, 5) indica che per l'evento 1, nella fila 2 c'è un gruppo di 5 posti liberi contigui a partire dalla poltrona 10. Per semplicità si assuma che non possono esistere due gruppi di posti contigui consecutivi. Per esempio non sono ammesse le tuple (1, 2, 10, **5**), (1, 2, 15, **3**). La precedente disponibilità sarebbe memorizzata come (1, 2, 10, **8**)

a) Si definisca la base di dati su ORACLE.
b) Si scriva la procedura Prenota(vIDEvento, vNumPosti) che prenota vNumPosti posti contigui per l'evento vIDEvento. La scelta del gruppo di posti da prenotare è basata sulle seguenti regole:

- I posti devono necessariamente essere contigui (quindi NumPosti>= vNumPosti)
- Sono da preferire i posti più vicini al palco (valore del campo Fila basso)
- A parità di fila va preferito uno slot composto dall'esatto numero di posti richiesto. Nel caso questo non esista va prenotato lo slot con dimensione massima (*politica worst-fit*)

Se non esistono vNumPosti posti contigui disponibili l'informazione viene riportata a video altrimenti vengono aggiornate le relazioni **DISPONIBILITA** ed **EVENTI**

```
create table EVENTI (
E_ID int,
E_Titolo varchar2(20),
E_Data date,
E_Struttura int,
E_Tipologia varchar2(10),
E_DisponibilitàResiduaPosti int,
primary key (E_ID),
foreign key (E_Struttura) references STRUTTURE(S_IDStruttura));

create table STRUTTURE (
S_IDStruttura int,
S_Indirizzo varchar2(20),
S_Tipo varchar2(10),
S_CapienzaTotale int,
primary key (S_IDStruttura)
);

create table DISPONIBILITA (
D_Evento int,
D_Fila int,
D_NumPoltronaIniziale int,
D_NumPosti int,
primary key (D_Evento,D_Fila),
foreign key (D_Evento) references EVENTI(E_ID)
);

create or replace
procedure Prenota(vIDEvento int, vNumPosti int) IS
--cursore
cursor cDisp is
select D_Fila, D_NumPoltronaIniziale, D_NumPosti, SIGN(D_NumPosti-vNumPosti) as c1 from DISPONIBILITA
where D_Evento=vIDEvento and D_NumPosti>=vNumPosti
order by 1, 4 desc, 3 desc;

vDisp cDisp%ROWTYPE;

begin
open cDisp;
fetch cDisp into vDisp;
if cDisp%NOTFOUND then
  DBMS_OUTPUT.PUT_LINE('Non ci sono abbastanza posti attigui disponibili');
else
  update EVENTI set E_DisponibilitàResiduaPosti=  E_DisponibilitàResiduaPosti-vNumPosti
  where E_ID = vIDEvento;
  if (vDisp.D_NumPosti-vNumPosti)>0 then
    update DISPONIBILITA set D_NumPosti=D_NumPosti-vNumPosti
    where D_Evento=vIDEvento and D_Fila=vDisp.D_Fila and D_NumPoltronaIniziale=vDisp.D_NumPoltronaIniziale;
  else
    delete DISPONIBILITA
    where D_Evento=vIDEvento and D_Fila=vDisp.D_Fila and D_NumPoltronaIniziale=vDisp.D_NumPoltronaIniziale;
 end if;
 DBMS_OUTPUT.PUT_LINE('Prenotati ' || vNumPosti || ' nella fila ' || vDisp.D_Fila || ' a partire dal posto ' ||
vDisp.D_NumPoltronaIniziale + vDisp.D_NumPosti - vNumPosti);

end if;
close cDisp;
end;
```

```
create or replace
procedure Prenota2(vIDEvento int, vNumPosti int) IS
--cursore
cursor cDisp is
select D_Fila, D_NumPoltronaIniziale, D_NumPosti from DISPONIBILITA
where D_Evento=vIDEvento and D_NumPosti>=vNumPosti
order by 1, 3 desc;

vDisp cDisp%ROWTYPE;
bestDisp cDisp%ROWTYPE;
Ciclo1 int:=0;


begin
bestDisp.D_NumPosti:=0;
for vDisp in cDisp loop
	if Ciclo1=0 then
		bestDisp=vDisp;
		Ciclo1:=1;
	end if;
	if vDisp.D_NumPosti = vNumPosti then
		bestDisp:=vDisp;
		exit;
	end if;
end loop;
if bestDisp.D_NumPosti=0 then
	DBMS_OUTPUT.PUT_LINE('Non ci sono abbastanza posti attigui disponibili');
else
	update EVENTI set E_DisponibilitàResiduaPosti=  E_DisponibilitàResiduaPosti-vNumPosti
	where E_ID = vIDEvento;
	if (bestDisp.D_NumPosti-vNumPosti)>0 then
		update DISPONIBILITA set D_NumPosti=D_NumPosti-vNumPosti
		where D_Evento=vIDEvento and D_Fila=bestDisp.D_Fila and
D_NumPoltronaIniziale=bestDisp.D_NumPoltronaIniziale;
	else
		delete DISPONIBILITA
		where D_Evento=vIDEvento and D_Fila=bestDisp.D_Fila and
D_NumPoltronaIniziale=bestDisp.D_NumPoltronaIniziale;
	end if;
	DBMS_OUTPUT.PUT_LINE('Prenotati ' || vNumPosti || ' nella fila ' || bestDisp.D_Fila || ' a partire dal posto ' ||
bestDisp.D_NumPoltronaIniziale + bestDisp.D_NumPosti - vNumPosti);

end if;

end;
```

2) Utilizzando il database TPCD, si disegni l'albero di esecuzione proposto da ORACLE e si calcoli il costo di accesso della seguente query.

```
select P_BRAND, AVG(PS_SUPPLYCOST)
from PART,PARTSUPP
where P_PARTKEY=PS_PARTKEY
and P_TYPE='LARGE BRUSHED COPPER'
group by P_BRAND
order by 2;
```

Si facciano le seguenti assunzioni e si estraggano dal DB eventuali dati mancanti:

D = 4096 byte len(P) = len(K) = 4 byte `NB = 101` u = 0.69

Si assuma inoltre che ORACLE non applichi proiezioni sui risultati intermedi e che non esegua operazioni in pipeline.

Soluzione

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			4682
SORT		ORDER BY	4682
SORT		GROUP BY	4682
TABLE ACCESS	PARTSUPP	BY INDEX ROWID	3
NESTED LOOPS			4577
TABLE ACCESS	PART	FULL	578
INDEX	IX_PART_PARTSUPP	RANGE SCAN	2

Si ricoda nuovamente a tutti gli studenti che il quesito "si disegni l'albero di esecuzione" non significa riportare sul foglio lo schema precedente già prodotto da Oracle, ma tradurlo in operazioni dell'algebra relazionale estesa (es Slide 6 -Ottimizzazione) indicando inoltre le tecniche di join e di accesso a tabella utilizzate. Questi esercizi sono stati ripetutamente svolti a lezione.

$NP_P = \lceil 200.000 \times 133/ (4096 \times 0{,}69) \rceil = 9.412$
$NP_{PS} = \lceil 800.000 \times 143/ (4096 \times 0{,}69) \rceil = 40.478$
Sel(`P_TYPE='LARGE BRUSHED COPPER'`) = 1/150

Si accede all'indice su PS_PARTKEY (XI_PART_PARTSUPP) in base al valore di una parte al fine di recuperare i record relativi a una parte. L'operazione è ripetuta per tutte le parti selezionate dal predicato su `P_BRAND`

$NL_{PS\_PARTKEY} = \lceil (200.000 \times 4\ \ 800.000\ \ 4) / (4.096 \times 0{,}69) \rceil = 1.416$
Costo di accesso mediante l'indice su PS_PARTKEY:
$2 + \lceil 1/200.000 \times 1.416 \rceil + \lceil 1/200.000 \times 40.478 \rceil = 2+1+1 = 4$

Costo(Nested Loop P-PS) = $9.412 + \lceil 1/150 \times 200.000 \rceil \times 4 =$ **14.748**

Il numero di tuple in risultato al join è pari alla cardinalità di PARTSUPP a cui è applicato il filtro di selezione

$NT_{P+PS} = NT_{PS} \times$ Sel(`P_TYPE='LARGE BRUSHED COPPER'`) $= \lceil 800.000 /150 \rceil = 5.334$
$NP_{P+PS} = \lceil 5.334 \times (133+143)/ (4096 \times 0{,}69) \rceil = 521$

Costo (GB)=$2 \times 521 \times (\lceil \log_{100}(521) \rceil + 1) = 2 \times 521 \times 3 =$ **3.126**

Per il calcolo delle tuple attese dopo il GROUP BY si dovrebbe utilizzare Cardenas ma considerando che la cardinalità di `P_BRAND` è molto limitata il risultato è scontato

NT_{GB} = `NK(P_BRAND)` = 25
$NP_{GB} = \lceil 25 \times (21+4) / (4096 \times 0{,}69) \rceil = 1$

L'ordinamento per la clausila ORDER BY si può fare in memoria poiché il numero di pagine di NP_{GB} è minore del numero di buffer NB a disposizione, quindi non operando in piplene, il costo di ordinamento è pari al costo di lettura della relazione

Costo totale = **14.748 + 3.126 + 1 = 17.875**

Laboratorio di Basi di Dati	**Matricola:**	FILA
Appello del 10/11/2014 (100 minuti)	**Nome e cognome:**	B
Service:	Login: esame___ Password: _____	
❍ Oracle8i ESAMESI_SI-ORACLESRV01		
❍ Oracle11g ESAMESI_ si-oracle-11.csr.unibo.it		

1) Il provider di risorse Hardware-As-A-Service GreenForest mantiene su un database la disponibilità delle risorse HW che può allocare.

DATACENTER(ID, Nome, Continente)

DISPONIBILITA(DataCenter:DATACENTER, Cluster, DataOraDa, DataOraA, NumCPU)

La tupla di **DISPONIBILITA (**1, 2, 10/11/2014:00:00, 10/11/2014:14:00, 5) indica che presso il cluster 2 del datacenter 1 sono disponibili NumCPU CPU nella fascia oraria [00:00 – 14:00] del 10 novembre 2014. Per semplicità si assuma che non possono esistere due disponibilità consecutive. Per esempio non sono ammesse le tuple

(1, 2, 10/11/2014:**00:00**, 10/11/2014:**14:00**, 5), **(**1, 2, 10/11/2014:**14:00**, 10/11/2014:**17:00**, 5). La precedente disponibilità sarebbe memorizzata come **(**1, 2, 10/11/2014:**00:00**, 10/11/2014:**17:00**, 5)

(1, 2, 10/11/2014:00:00, 10/11/2014:14:00, **5**), **(**1, 2, 10/11/2014:00:00, 10/11/2014:14:00, **3**). La precedente disponibilità sarebbe memorizzata come **(**1, 2, 10/11/2014:00:00, 10/11/2014:14:00, **8**).

a) Si definisca la base di dati su ORACLE.
b) Si scriva la procedura Prenota(vContinente, vNumCPU, vDataOraDa, vDataOraA) che prenota vNumCPU CPU dello stesso cluster in uno dei data center siti in vContinente. La scelta delle CPU da prenotare è basata sulle seguenti regole:

- Le CPU devono necessariamente appartenere allo stesso cluster (quindi NumCPU>= vNumCPU) e devono essere disponibili per tutto l'intervallo temporale richiesto
- Sono da preferire i cluster più nuovi (valore del campo Cluster basso)
- A parità di cluster va preferita uno slot con l'esatto numero di CPU. Nel caso questo non esista va prenotato lo slot con il massimo numero di CPU (*politica worst-fit*)

Se non esistono vNumCPU disponibili nello stesso cluster e per l'intervallo temporale richiesto l'informazione viene riportata a video altrimenti viene aggiornata la relazione **DISPONIBILITA**

Soluzione

```
create table DATACENTER (
C_ID int,
C_Nome varchar2(20),
C_Continente varchar2(10),
primary key (C_ID)
);

create table DISPONIBILITA (
D_DataCenter int,
D_Cluster int,
D_DataOraDa timestamp,
D_DataOraA timestamp,
D_NumCPU int,
primary key (D_DataCenter, D_Cluster, D_DataOraDa),
foreign key (D_DataCenter) references DATACENTER(C_ID)
);


create or replace
procedure Prenota(vContinente varchar2, vNumCPU int , vDataOraDa timestamp, vDataOraA timestamp) IS
--cursore
cursor cDisp is
select D_DataCenter, D_Cluster, D_NumCPU , SIGN(D_NumCPU-vNumCPU) ,D_DataOraDa,D_DataOraA from
DISPONIBILITA, DATACENTER
where D_DataCenter=C_ID and D_DataOraDa<=vDataOraDa and D_DataOraA >=vDataOraDa and D_NumCPU>=vNumCPU
order by 2, 4 desc, 3 desc;

vDisp cDisp%ROWTYPE;
vInFila int;
begin
open cDisp;
fetch cDisp into vDisp;
if cDisp%NOTFOUND then
  DBMS_OUTPUT.PUT_LINE('Non ci sono abbastanza risorse disponibili');
else

  delete DISPONIBILITA where D_DataCenter=vDisp.D_DataCenter and D_Cluster=vDisp.D_Cluster and
D_DataOraDa=vDisp.D_DataOraDa;
  if (vDisp.D_DataOraDa<vDataOraDa) then
    insert into DISPONIBILITA values (vDisp.D_DataCenter, vDisp.D_Cluster, vDisp.D_DataOraDa, vDataOraDa,
vDisp.D_NumCPU);
  end if;
  if (vDisp.D_DataOraA>vDataOraA)          then
    insert into DISPONIBILITA values (vDisp.D_DataCenter, vDisp.D_Cluster, vDataOraA, vDisp.D_DataOraA,
vDisp.D_NumCPU);
  end if;
  if (vDisp.D_NumCPU>vNumCPU) then
    insert into DISPONIBILITA values (vDisp.D_DataCenter, vDisp.D_Cluster,  vDataOraDa, vDataOraA, vDisp.D_NumCPU-
vNumCPU);
  end if;
end if;
DBMS_OUTPUT.PUT_LINE('Prenotate ' || vNumCPU || ' risorse nel cluster ' || vDisp.D_Cluster || ' del datacenter ' ||
vDisp.D_DataCenter);

end;
```

```
create or replace
procedure Prenota2(vContinente varchar2, vNumCPU int , vDataOraDa timestamp, vDataOraA timestamp) IS
--cursore
cursor cDisp is
select D_DataCenter, D_Cluster, D_NumCPU , D_DataOraDa,D_DataOraA from DISPONIBILITA, DATACENTER
where D_DataCenter=C_ID and D_DataOraDa<=vDataOraDa and D_DataOraA >=vDataOraDa and D_NumCPU>=vNumCPU
order by 2,  3 desc;

vDisp cDisp%ROWTYPE;
bestDisp cDisp%ROWTYPE;
vInFila int;
Ciclo1 int;
begin

bestDisp.D_NumCPU:=0;
for vDisp in cDisp loop
if Ciclo1=0 then
        bestDisp:=vDisp;
        Ciclo1:=1;
end if;
if vDisp.D_NumCPU = vNumCPU then
        bestDisp:=vDisp;
        exit;
end if;
en
d loop;

if bestDisp.D_NumCPU=0 then
  DBMS_OUTPUT.PUT_LINE('Non ci sono abbastanza risorse disponibili');
else

  delete DISPONIBILITA where D_DataCenter=bestDisp.D_DataCenter and D_Cluster=bestDisp.D_Cluster and
D_DataOraDa=bestDisp.D_DataOraDa;
  if (bestDisp.D_DataOraDa<vDataOraDa) then
    insert into DISPONIBILITA values (bestDisp.D_DataCenter, bestDisp.D_Cluster, bestDisp.D_DataOraDa, vDataOraDa,
bestDisp.D_NumCPU);
  end if;
  if (bestDisp.D_DataOraA>vDataOraA)      then
    insert into DISPONIBILITA values (bestDisp.D_DataCenter, bestDisp.D_Cluster, vDataOraA, bestDisp.D_DataOraA,
bestDisp.D_NumCPU);
  end if;
  if (bestDisp.D_NumCPU>vNumCPU) then
    insert into DISPONIBILITA values (bestDisp.D_DataCenter, bestDisp.D_Cluster,  vDataOraDa, vDataOraA,
bestDisp.D_NumCPU-vNumCPU);
  end if;
end if;
DBMS_OUTPUT.PUT_LINE('Prenotate ' || vNumCPU || ' risorse nel cluster ' || bestDisp.D_Cluster || ' del datacenter ' ||
bestDisp.D_DataCenter);
end;
```

2) Utilizzando il database TPCD, si disegni l'albero di esecuzione proposto da ORACLE e si calcoli il costo di accesso della seguente query.

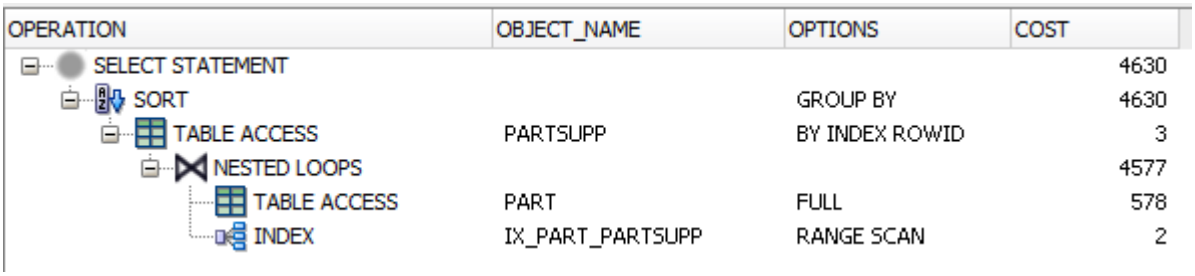
```
select P_BRAND, AVG(PS_SUPPLYCOST)
from PART,PARTSUPP
where P_PARTKEY=PS_PARTKEY
and P_TYPE='ECONOMY ANODIZED TIN'
group by P_BRAND;
```

Si facciano le seguenti assunzioni e si estraggano dal DB eventuali dati mancanti:

D = 4096 byte len(P) = len(K) = 4 byte `NB = 101` u = 0.69

Si assuma inoltre che ORACLE non applichi proiezioni sui risultati intermedi e che non esegua operazioni in pipeline.

Soluzione



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			4630
SORT		GROUP BY	4630
TABLE ACCESS	PARTSUPP	BY INDEX ROWID	3
NESTED LOOPS			4577
TABLE ACCESS	PART	FULL	578
INDEX	IX_PART_PARTSUPP	RANGE SCAN	2

Si ricoda nuovamente a tutti gli studenti che il quesito "si disegni l'albero di esecuzione" non significa riportare sul foglio lo schema precedente già prodotto da Oracle, ma tradurlo in operazioni dell'algebra relazionale estesa (es Slide 6 -Ottimizzazione) indicando inoltre le tecniche di join e di accesso a tabella utilizzate. Questi esercizi sono stati ripetutamente svolti a lezione.

$NP_P = \lceil 200.000 \times 133/ (4096 \times 0{,}69) \rceil = 9.412$
$NP_{PS} = \lceil 800.000 \times 143/ (4096 \times 0{,}69) \rceil = 40.478$
Sel(`P_TYPE='ECONOMY ANODIZED TIN'`) = 1/150

Si accede all'indice su PS_PARTKEY (XI_PART_PARTSUPP) in base al valore di una parte al fine di recuperare i record relativi a una parte. L'operazione è ripetuta per tutte le parti selezionate dal predicato su `P_BRAND`

$NL_{PS\_PARTKEY} = \lceil (200.000 \times 4 \ \ 800.000 \times 4) / (4.096 \times 0{,}69) \rceil = 1.416$
Costo di accesso all'indice su PS_PARTKEY:
$2 + \lceil 1/200.000 \times 1.416 \rceil + \lceil 1/200.000 \times 40.478 \rceil = 2+1+1 = 4$

Costo(Nested Loop P-PS) = $9.412 + \lceil 200.000/150 \rceil \times 4 =$ **14.748**

Il numero di tuple in risultato al join è pari alla cardinalità di PARTSUPP a cui è applicato il filtro di selezione

$NT_{P+PS} =$ `NT`$_{PS} \times$ Sel(`P_TYPE='LARGE BRUSHED COPPER'`) $= \lceil 800.000 /150 \rceil = 5.334$
$NP_{P+PS} = \lceil 5.334 \times (133+143)/ (4096 \times 0{,}69) \rceil = 521$

Costo totale = **14.748 + 3.126 = 17.874**