Planovi izvršavanja

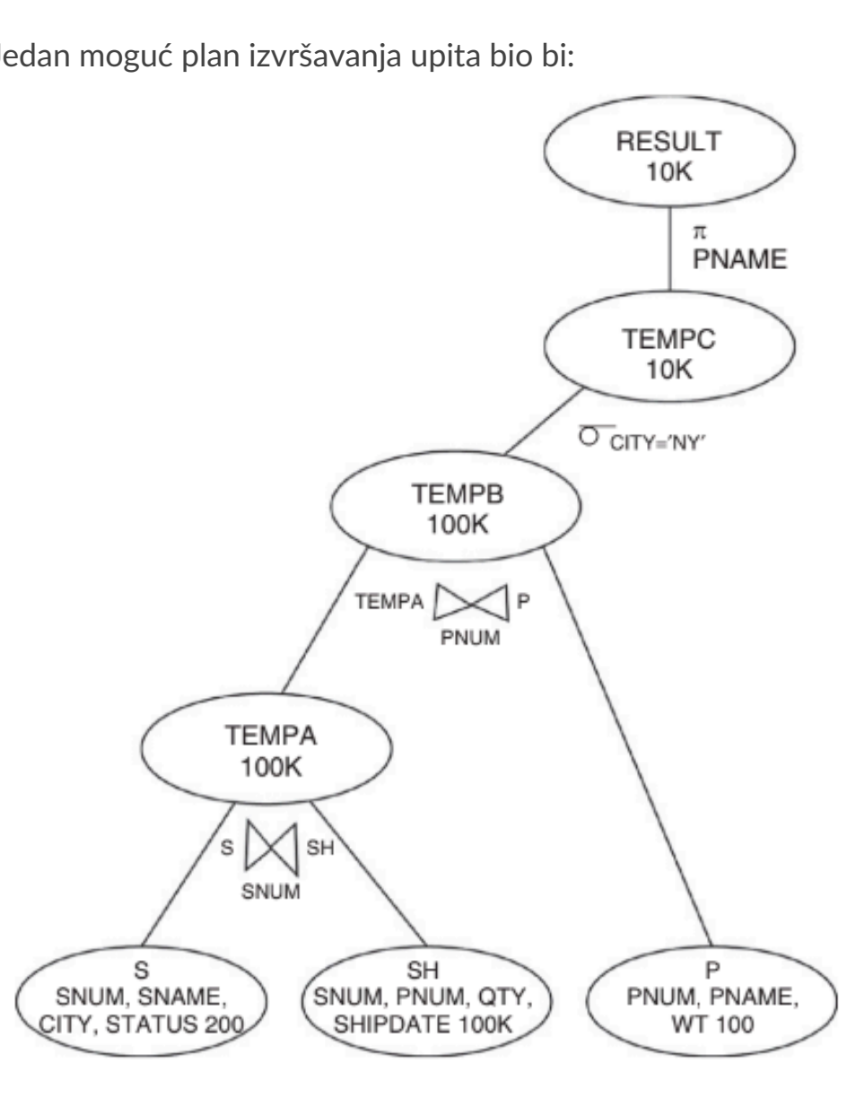
Pre samog izvršavanja upita, tj. dovlačenja redova koji zadovoljavaju taj upit iz memorije, sistem za upravljanje bazom podataka će kreirati tzv. plan izvršavanja upita. Naime, ovaj plan predstavlja konkretne korake koji će se izvršiti radi dohvatanja zahtevanih podataka - na koji način i kojim redosledom će se podaci filtrirati, tabele spajati, da li i koji će se indeksi koristiti i slično.

Za jedan upit je moguće napraviti ogroman broj ekvivalentnih (po njihovom rezultatu) planova izvršavanja. Međutim, neki od njih će biti efikasniji od drugih, a pronalaženje što boljeg plana je posao optimizatora upita u okviru subp-a. Da bi to izvršio, optimizator na raspolaganju ima informacije o statistikama podataka u tabelama kojima se pristupa upitom. U okviru MySQL-a, neke od dostupnih statistika su broj redova u tabeli, procenat NULL vrednosti po kolonama, raspodela vrednosti u kolonama u vidu histograma i još toga. Radi što boljih rezultata rada optimizatora, poželjno je povremeno ažurirati ove statistike, što se može uraditi ANALYZE TABLE naredbom.

Interno, plan izvršavanja se može predstaviti kao graf, tačnije stablo, gde savaki od čvorova predstavlja neki korak prilikom izvršavanja upita (pa time i neki vid međurezultata) i koren predstavlja konačni rezultat upita. Svim čvorovima su pridružene dodatne informacije poput procenjenog broja redova međurezultata, njegovih kolona, kao i procenjene cene za njegovo kreiranje. Primera radi, razmotrimo naredni upit nad tabelama *parts*, *shipments* i *suppliers*:

```
SELECT  p.name
FROM    parts AS p JOIN
        shipments AS sh ON p.pnum = sh.pnum JOIN
        suppliers AS s ON sh.snum = s.snum
WHERE   s.city = 'NY';
```

Jedan moguć plan izvršavanja upita bio bi:



Primetimo da listovi plana izvršavanja u ovom slučaju predstavljaju bazne tabele koje se koriste u upitu, međurezultati TEMPA, TEMPB rezultate operacije spajanja, a TEMPC međurezultat dobijen filtriranjem po uslovu s.city = 'NY'.

Procena cene plana izvršavanja

Jedna od metrika koja se može koristiti za određivanje cene plana izvršavanja je procenjen broj I/O operacija potrebnih za njegovo sprovođenje. Međutim, da bi se došlo do tog broja, prvo je potrebno zapravo proceniti broj redova u svakom od međurezultata, tj. njihova kardinalnost. Definišimo, pre toga, pojam selektivnosti.

Selektivnost

Selektivnost predikata *P*, u oznaci *S(P)*, primenjenog u selekciji nad tabelom *T* (npr. u WHERE klauzuli upita), predstavlja udeo redova tabele *T* koji zadovoljavaju predikat *P*. U zavisnosti od vrste predikata i pod pretpostavkom uniformne raspodele vrednosti atributa nad kojima se vrši selekcija (što najčešće nije realna situacija, ali o tome kasnije), selektivnost se računa na sledeći način:

- $S(A = a) = \frac{1}{card_A(T)}$, gde $card_A(T)$ predstavlja broj jedinstvenih vrednosti atributa A u tabeli T
- $S(A > a) = \frac{max_A(T) - a}{max_A(T) - min_A(T)}$, gde $max_A(T)$ i $min_A(T)$ redom predstavljaju maksimalnu i minimalnu vrednost atributa A u tabeli T
- $S(A < a) = \frac{a - min_A(T)}{max_A(T) - min_A(T)}$
- $S(P \wedge Q) = S(P) \times S(Q)$
- $S(P \vee Q) = S(P) + S(Q) - S(P) \times S(Q)$

Neka je dat naredni upit:

```
SELECT  *
FROM    employees
WHERE   role = 'executive' AND salary > 1500;
```

Ukoliko u tabeli employees imamo četiri različite vrednosti za atribut role ('support', 'worker', 'manager', 'executive') i plate se kreću iz opsega od 1000 do 3000, selektivnost predikata u WHERE klauzuli bi mogli da procenimo na sledeći način:

$$\begin{aligned} &S(role = executive \wedge salary > 1500) \\ &= S(role = executive) \times S(salary > 1500) \\ &= \frac{1}{4} \times \frac{3000 - 1500}{3000 - 1000} = \frac{1}{4} \times \frac{3}{4} = \frac{3}{16} \end{aligned}$$

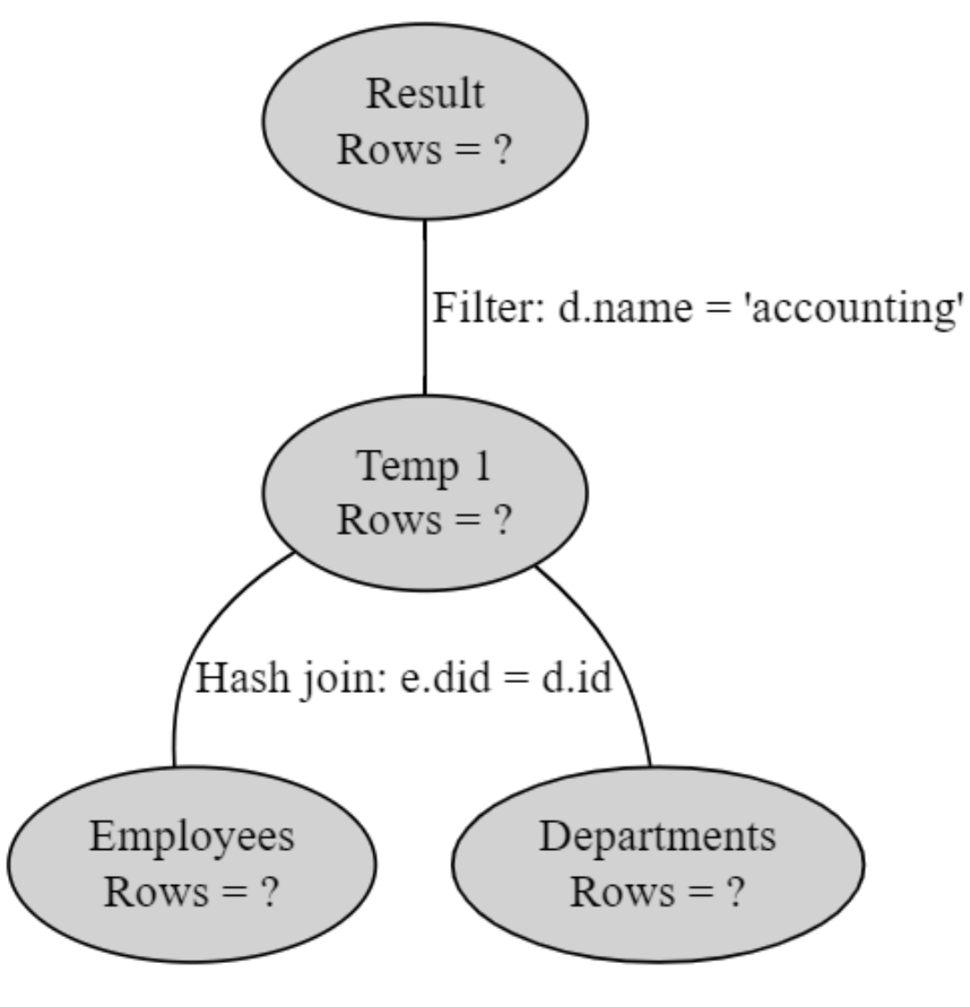
Primetimo da je nerealno da četvrtina zaposlenih ima direktorsku poziciju, te prethodna procena selektivnosti vrlo verovatno ne odgovara stvarnom stanju u podacima. Ovako loša procena može navesti optimizator na loš izbor plana izvršavanja, npr. da zbog preslabe procenjene selektivnosti ne primeni neki indeks, iako bi on dosta pomogao u izvršavanju upita. Kao što je već pomenuto, moderni subp-ovi imaju mogućnost održavanja histograma koji aproksimiraju raspodelu vrednosti nekog atributa, tj. kolone, u okviru tabele, što omogućava mnogo precizniju procenu selektivnosti. Na primer, ukoliko bi u održavanim statistikama postojala informacija da zaposlenih sa direktorskom pozicijom ima samo petoro od ukupno hiljadu zaposlenih, za $S(role = executive)$ bi se mogla uzeti vrednost $\frac{5}{1000} = \frac{1}{200}$, pa bi celokupna procenjena selektivnost prethodnog predikata imala 50 puta manju vrednost (tj. bila bi 50 puta jača).

Kardinalnost

Sada kad smo upoznati sa pojmom selektivnosti i time kako se ona procenjuje, možemo preći i na procenjivanje **kardinalosti** međurezultata u okviru plana izvršavanja. Razmotrimo naredni upit:

```
SELECT  *
FROM    employees AS e JOIN
        departments AS d ON e.did = d.id
WHERE   d.name = 'accounting';
```

Jedan od mogućih planova izvršavanja izgledao bi ovako:



Kako da procenimo broj redova, tj. kardinalnost, svakog od međurezultata? Pretpostavimo da je broj redova u tabeli *Employees* 200, a u tabeli *Departments* 10. Prema tome, za listove ovog plana izvršavanja možemo uzeti upravo te vrednosti. Preostalo je da izračunamo kardinalnosti za *Temp 1* i *Result*. Naime, *Temp 1* međurezultat nastaje spajanjem tabele *Employees* sa tabelom *Departments* po njenom primarnom ključu, a *Result* filtriranjem redova iz *Temp 1*. Kardinalnosti u ovim situacijama ćemo procenjivati na sledeći način:

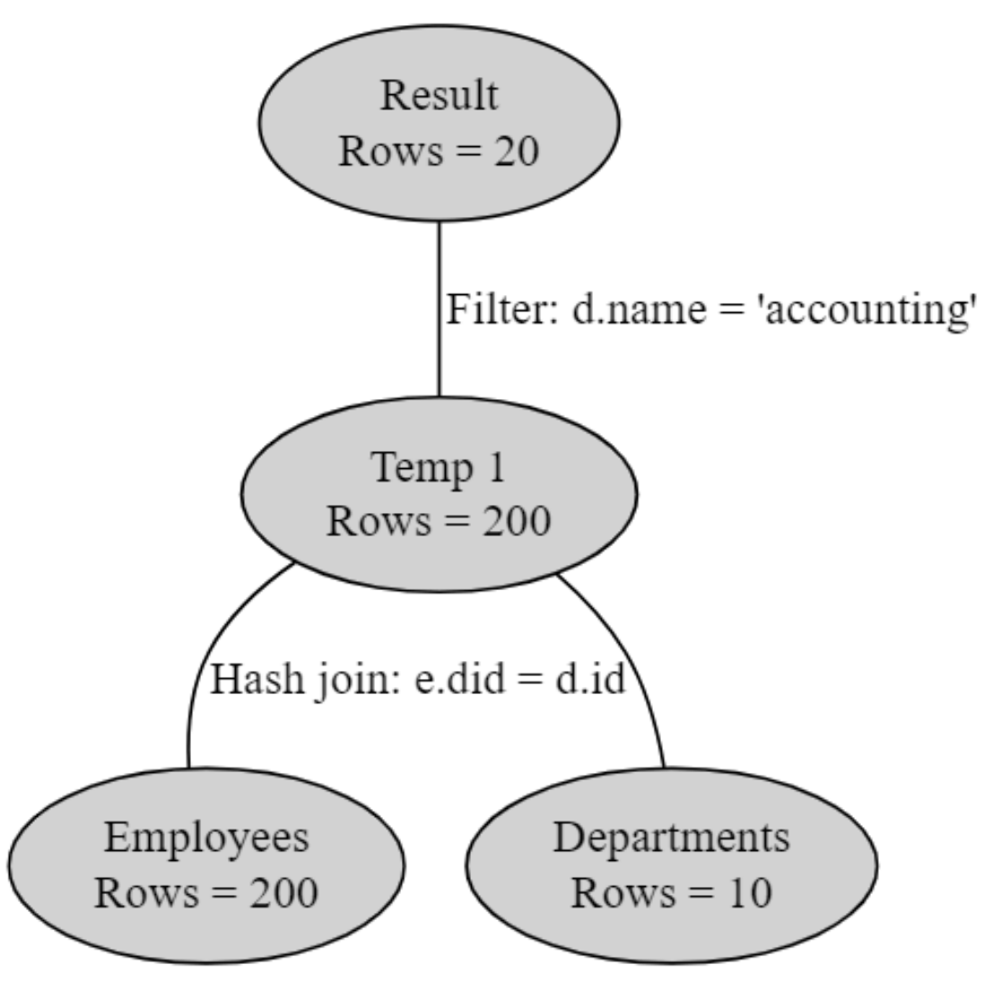
- $card(T_1\ join\ T2\ on\ P) = S(P) \times card(T_1) \times card(T_2)$, gde $S(P)$ predstavlja selektivnost predikata po kojem se vrši spajanje
- $card(T\ filter\ by\ P) = S(P) \times card(T)$

Prema ovome, dobijamo sledeće procene (napomena - pretpostavljamo da svaki od 10 departmana ima jedinstveno ime):

$$\begin{aligned} card(Temp\ 1) &= S(e.did = d.id) \times card(Employees) \times card(Departments) \\ &= \frac{1}{10} \times 200 \times 10 = 200, \end{aligned}$$

$$\begin{aligned} card(Result) &= S(d.name = accounting) \times card(Temp\ 1) \\ &= \frac{1}{10} \times 200 = 20. \end{aligned}$$

Predstavljeno planom izvršavanja, to bi bilo:



Broj stranica

Iako ukupan broj redova svih međurezultata plana izvršavanja potencijalno može da se iskoristi kao metrika koja opisuje cenu tog plana, u dosta slučajeva ona nije dovoljno precizna. Naime, najveći deo izvršavanja upita odlazi na I/O operacije, te bi bilo dobro osmisliti neku metriku koja to i oslikava. Jedna od mogućnosti je procenjen broj pristupa stranicama (niza većeg broja uzastopnih slogova, tj. redova, koji se dovlači u radnu memoriju sa diska prilikom pristupa tabeli), a za takvu procenu nam nije dovoljan samo broj redova u međurezultatima, već i veličina tih redova, kao i veličina stranica.

Pogledajmo detaljniju strukturu tabela *Employees* i *Departments* iz prethodnih primera:

Employees

id	name	surname	role	salary	did
INT	VARCHAR(40)	VARCHAR(40)	VARCHAR(20)	INT	INT

Departments

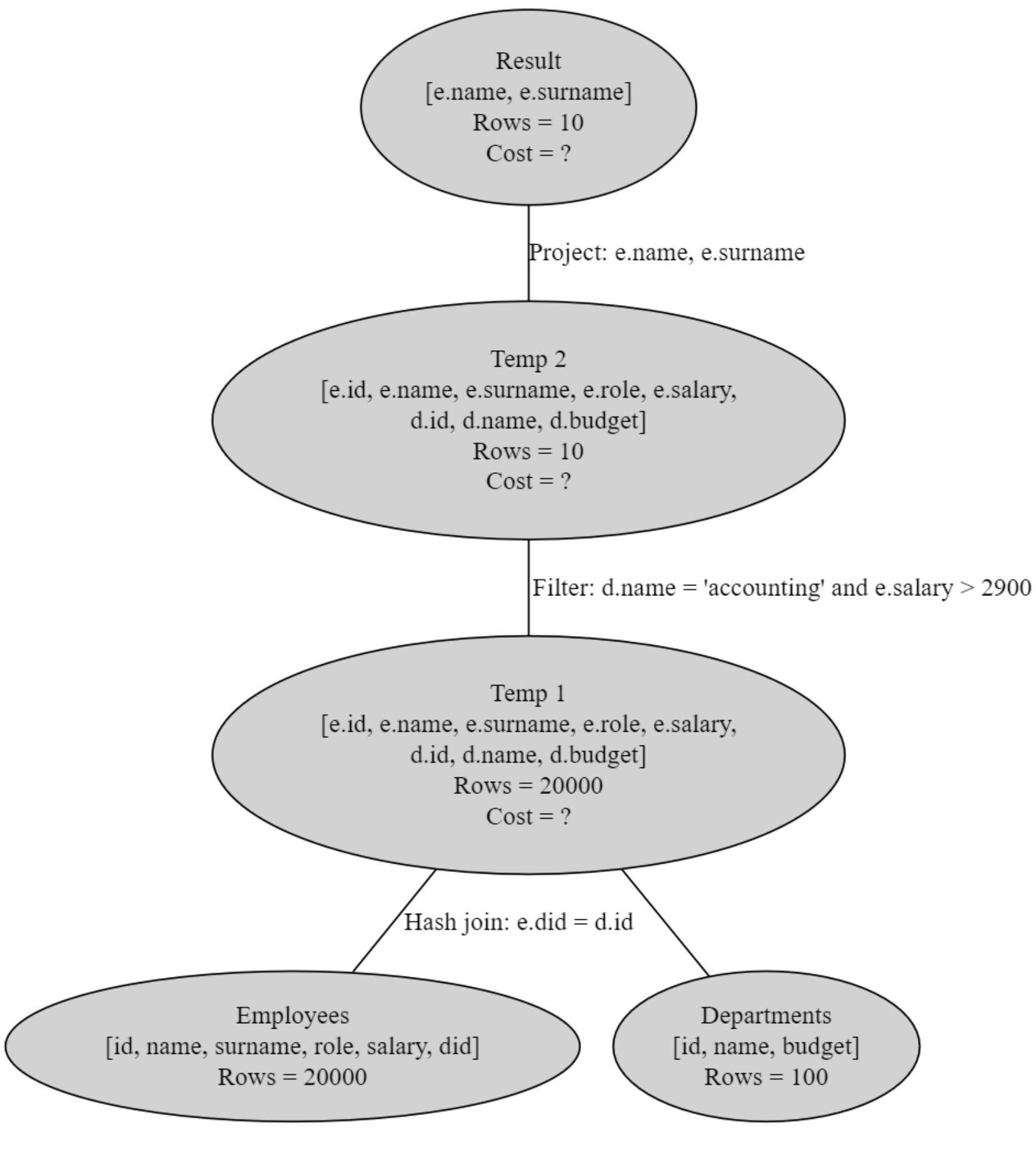
id	name	budget
INT	VARCHAR(40)	INT

Uzmimo da je, na primer, veličina stranica podešena na 4KB (za detaljnije informacije u kontekstu MySQL-a, pogledati sledeći dokument). Ako bi zanemarili zaglavlje stranica, dobili bi da u jednu stranicu u proseku može da stane $\lfloor\frac{4KB}{4B+20B+20B+10B+4B+4B}\rfloor = 66$ *Employees* slogova ili $\lfloor\frac{4KB}{4B+20B+4B}\rfloor = 146$ *Departments* slogova (napomena - za VARCHAR(n) atribute je uzeto da u proseku zauzimaju $\frac{n}{2}$ bajta, a za INT atribute 4 bajta). Označimo ovaj broj kao $PF(T)$ za tabelu T. Prosečan broj stranica potrebnih za skladištenje cele tabele T je onda $pages(T) = \lceil\frac{rows(T)}{PF(T)}\rceil$.

Razmotrimo upit sličan prethodnom, samo što se sada vrši projekcija na $e.name$ i $e.surname$, kao i dodatna selekcija po $e.salary$:

```
SELECT  e.name, e.surname
FROM    employees AS e JOIN
        departments AS d ON e.did = d.id
WHERE   d.name = 'accounting' AND
        e.salary > 2900;
```

Malo detaljniji i sa većim brojem redova po tabeli (u odnosu na prethodni) plan izvršavanja bi mogao da izgleda ovako (pretpostavka da su plate u opsegu od 1000 do 3000, kao i to da su nazivi departmana jedinstveni, pa je selektivnost predikata u WHERE klauzuli jednaka $\frac{1}{100} \times \frac{100}{2000} = \frac{1}{2000}$):



Kako da procenimo broj I/O operacija, tj. dobavljenih stranica, za svaki od međurezultata? To zavisi od konkretne operacije koja se izvršava da bi se stiglo do tog međurezultata. U zavisnosti od korišćenog algoritma za spajanje tabela, koristićemo naredne procene broja I/O operacija (radi jednostavnosti zapisa, neka važi $PT_n = pages(T_n)$) :

Algoritam	Cena spajanja T_1 i T_2
Block Nested Loop Join	$PT_1 + PT_1 \times PT_2$, napomena - za T_1 se onda obično bira ona tabela sa manje stranica
Sort-Merge Join	$PT_1 + PT_2 + 2 \times \lceil PT_1 \times log_2(PT_1)\rceil + 2 \times \lceil PT_2 \times log_2(PT_2)\rceil$, napomena - u zavisnosti od toga da li su tabele T_1 i/ili T_2 već sortirane po atributu spajanja (npr. ako se spaja po primarnom ključu, tj. stranom ključu, što je najčešće slučaj), cena može biti dosta manja (gube se, jedan ili oba, sabirci koji predstavljaju cenu sortiranja)
(GRACE) Hash Join	$3 \times (PT_1 + PT_2)$

(Detaljnije obrazloženje iza ovih cena možete videti ovde)

Prema ovome, za cenu međurezultata $Temp\ 1$ možemo uzeti

$$\begin{aligned} cost(Temp\ 1) &= 3 \times (pages(Employees) + pages(Departments)) \\ &= 3 \times (\lceil\frac{rows(Employees)}{PF(Employees)}\rceil + \lceil\frac{rows(Departments)}{PF(Departments)}\rceil) \\ &= 3 \times (\lceil\frac{20000}{66}\rceil + \lceil\frac{100}{146}\rceil) = 3 \times (304 + 1) = 915 \end{aligned}$$

Primetimo da je po ovoj proceni, zato što je tabela *Departments* dosta mala (samo 100 redova i staje u jednu stranicu), bolje bilo iskoristiti neku drugu strategiju za spajanje - običan *Block Nested Loop Join* bi bio jeftiniji.

Da bi se međurezultat *Temp 2* formirao, potrebno je pročitati sve redove iz *Temp 1* (pesimistična pretpostavka da nakon svake operacije vraćamo međurezultat na disk, pa je potrebno njegovo ponovno dovlačenje) i upisati 10 redova koji zadovoljavaju zadati uslov nazad na disk. Pre toga, obratimo pažnju na to da se u *Temp 1* i *Temp 2* sada nalaze i kolone tabele *Employees* i tabele *Departments*, pa je broj potrebnih stranica za njihovo skladištenje veća nego u baznim tabelama, tačnije, $PF(Temp\ 1) = PF(Temp\ 2) = \lfloor\frac{4KB}{4B+20B+20B+10B+4B+4B+20B+4B}\rfloor = 47$. Prema tome, cena je:

$$\begin{aligned} cost(Temp\ 2) &= pages(Temp\ 1) + pages(Temp\ 2) \\ &= \lceil\frac{20000}{47}\rceil + \lceil\frac{10}{47}\rceil = 426 + 1 = 427 \end{aligned}$$

Za dobijanje konačnog rezultata *Result* vrši se projekcija iz *Temp 2* na ime i prezime zaposlenog. Pošto je u pitanju konačni rezultat (a ne neki međurezultat), ovde neće biti izvršen njegov upis na disk, pa je njegova cena jednaka ceni čitanja iz tabele *Temp 2*, to jest:

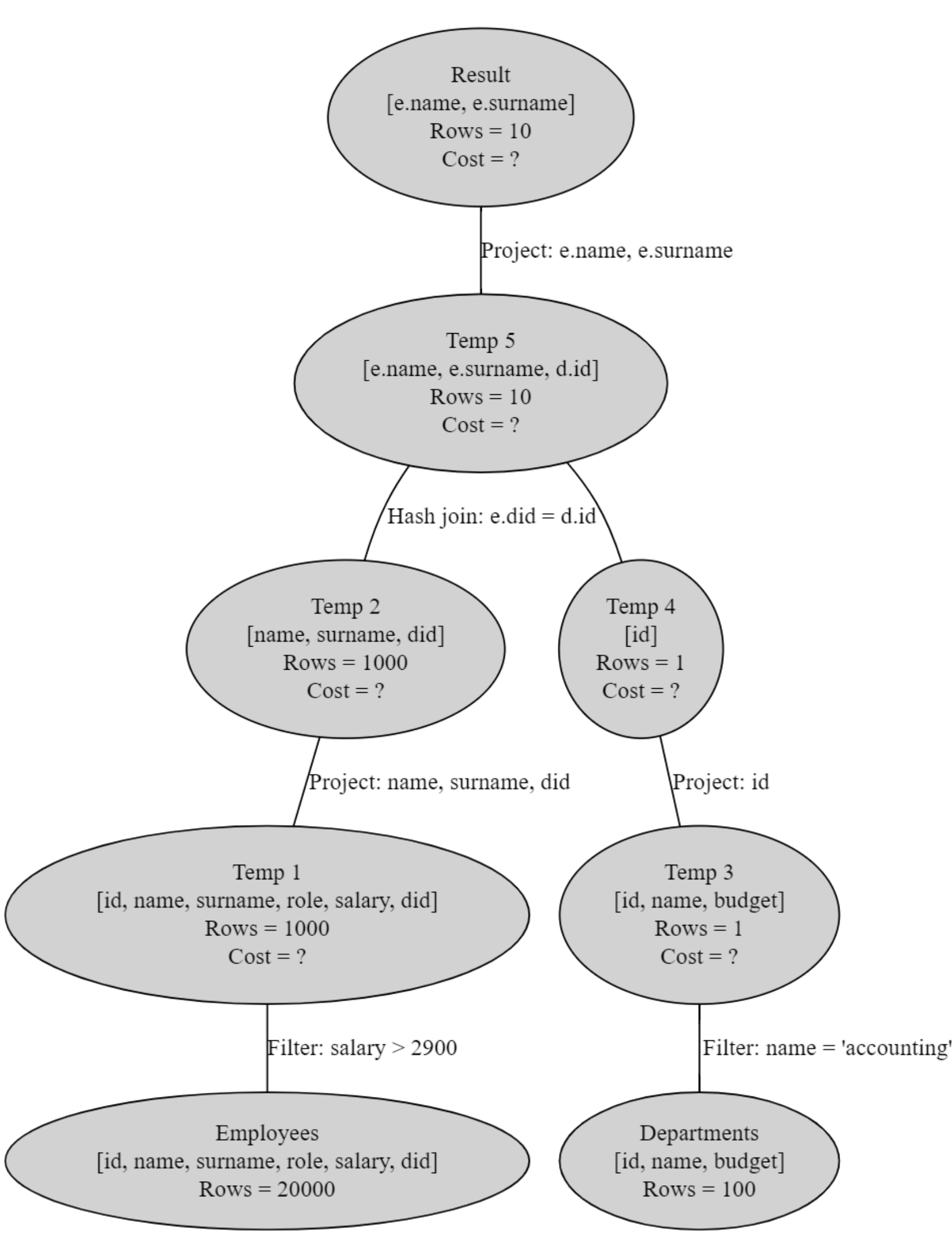
$$cost(Result) = \lceil\frac{10}{47}\rceil = 1$$

Prema ovom računu, ukupna cena plana izvršavanja je procenjena na $cost(Temp\ 1) + cost(Temp\ 2) + cost(Result) = 915 + 427 + 1 = 1343$ I/O operacija, tj. dovlačenja stranica sa diska.

Možemo li bolje?

Ako bolje pogledamo prethodni plan izvršavanja i njegovu cenu, možemo da primetimo da je za najveći deo cene krivo upravo spajanje tabela. Naime, u najvećem broju slučajeva, spajanje tabela predstavlja najskuplju operaciju u okviru plana izvršavanja, te je dobra ideja fokusirati se upravo na njih prilikom optimizacije.

Kako to da izvedemo? Već smo videli da je moguće da bi neki drugi algoritam spajanja bio jeftiniji u prethodnom slučaju, ali možemo da razvijemo i opštiju strategiju optimizacije od toga. Potencijalno interesantnija ideja je sledeća - šta ako bi se operacije selekcije i projekcije izvršile pre operacije spajanja? Naime, u prethodnom planu je u spajanju učestvovao veliki broj redova koji su naknadno izbačeni (predikatom nad platom je iz opticaja izbačeno 95% zaposlenih, a nad nazivom departmana njih 99%), a u međurezultatima su čuvani i podaci koji su apsolutno nepotrebni za dalje izvršavanje upita (npr. uloga zaposlenog, kao i naziv i budžet departmana se uopšte ne pominju u upitu). Pogledajmo naredni, alternativni plan izvršavanja:



Cena kreiranja *Temp 1* međurezultata podrazumeva cenu čitanja tabele *Employees* i zapisivanja rezultujućih kolona nakon filtriranja:

$$\begin{aligned} cost(Temp\ 1) &= pages(Employees) + pages(Temp\ 1) \\ &= \lceil\frac{rows(Employees)}{PF(Employees)}\rceil + \lceil\frac{rows(Temp\ 1)}{PF(Temp\ 1)}\rceil \\ &= \lceil\frac{20000}{66}\rceil + \lceil\frac{1000}{66}\rceil = 320 \end{aligned}$$

Za *Temp 2* je potrebno pročitati *Temp 1* i zapisati projektovane slogove. Pošto se projekcija vrši na *name*, *surname* i *did*, ukupna veličina sloga u *Temp 2* će biti $20B + 20B + 4B = 44B$, te u jednu stranicu od $4KB$ može da stane ukupno $PF(Temp\ 2) = \lfloor\frac{4KB}{44B}\rfloor = 93$ sloga. Prema tome, sledi:

$$\begin{aligned} cost(Temp\ 2) &= pages(Temp\ 1) + pages(Temp\ 2) \\ &= \lceil\frac{rows(Temp\ 1)}{PF(Temp\ 1)}\rceil + \lceil\frac{rows(Temp\ 2)}{PF(Temp\ 2)}\rceil \\ &= \lceil\frac{1000}{66}\rceil + \lceil\frac{1000}{93}\rceil = 27 \end{aligned}$$

Zato što, kao u prethodnom primeru, cela tabela $Departments$ staje u jednu stranicu, sledi da je $cost(Temp\ 3) = 1$ i $cost(Temp\ 4) = 1$.

Cena spajanja *Temp 2* i *Temp 4* će sada biti mnogo manja nego u prethodnom primeru gde su filtriranje i selekcija rađeni nakon spajanja. Naime, sledi da je nova cena:

$$\begin{aligned} cost(Temp\ 5) &= 3 \times (pages(Temp\ 2) + pages(Temp\ 4)) \\ &= 3 \times (\lceil\frac{rows(Temp\ 2)}{PF(Temp\ 2)}\rceil + 1) \\ &= 3 \times (\lceil\frac{1000}{93}\rceil + 1) = 3 \times (11 + 1) = 36 \end{aligned}$$

Primetimo da bi, kada bi se koristio *Block Nested Loop Join* gde je za spoljnu petlju izabrana *Temp 4* (zato što sadrži manje stranica), cena bi bila $1 + 1 \times 11 = 12$, što je još veća ušteda.

Konačno, cena za kreiranje krajnjeg rezultata *Result* je takođe 1, tj. $cost(Result) = 1$, zato što je potrebno samo pročitati 10 redova iz *Temp 5*, a oni svi staju u jednu stranicu.

Ukupna procenjena cena ovog plana izvršavanja je onda jednaka $cost(Temp\ 1) + cost(Temp\ 2) + cost(Temp\ 3) + cost(Temp\ 4) + cost(Temp\ 5) + cost(Result) = 320 + 27 + 1 + 1 + 36 + 1 = 386$, što je dosta manje nego prvobitnih 1343 I/O operacija.

Ova strategija optimizacije je skoro uvek primenljiva. Naime, teži se da se u planu izvršavanja što pre oslobodimo nepotrebnih redova (filtriranjem čim je to moguće) i nepotrebnih kolona (projekcijom na samo one kolone koje su zapravo potrebne za izvršavanje upita).