DODATAK A

Podsetnik za analitičke funkcije

U receptima iz ove knjige koriste se prednosti koje pružaju analitičke funkcije (engl. *window functions*), dodate ISO SQL standardu 2003. godine i analitičke funkcije specifične za pojedine platforme. U ovom dodatku je ponuđen kratak opis načina na koji rade analitičke funkcije. Analitičke funkcije znatno olakšavaju mnoge teže poslove (teže u smislu „ako koristite standardni SQL“). Spisak analitičkih funkcija na raspolaganju, opis sintakse i detaljna objašnjenja o tome kako one rade, naći ćete u dokumentaciji svog sistema.

Grupisanje redova

Pre nego što pređemo na analitičke funkcije, ključno je da shvatite kako se grupišu u SQL-u redovi. Prema mom iskustvu, koncept grupisanja rezultata u SQL-u za mnoge je kamen spoticanja. Izvor problema je nepotpuno razumevanje kako deluje odredba GROUP BY i zašto određeni upiti vraćaju određene rezultate kad sadrže odredbu GROUP BY.

Jednostavno rečeno, grupisanje je način organizovanja sličnih redova u celine. Kada u upitu zadate odredbu GROUP BY, svaki red u skupu rezultata zapravo je grupa i predstavlja jedan ili više redova koji sadrže iste vrednosti u jednoj ili više kolona koje zadate. To je cela suština.

Ako je grupa samo red koji predstavlja jedan ili više redova koji sadrže istu vrednost u određenoj koloni (ili kolonama), primeri grupa koje se mogu formirati od redova tabele EMP su *svi zaposleni u odeljenju 10* (zajednička vrednost za sve te zaposlene na osnovu koje bivaju organizovani u grupu, jeste DEPTNO=10) ili *svi referenti* (zajednička vrednost za sve te zaposlene na osnovu koje bivaju organizovani u grupu jeste JOB='CLERK'). Pogledajte naredne upite. Prvi prikazuje sve zaposlene u odeljenju 10; drugi upit grupiše sve zaposlene u odeljenju 10 i daje sledeće podatke o toj grupi: ukupan broj redova (članova) grupe, najveću platu i najmanju platu:

```
select deptno,ename
  from emp
 where deptno=10
```

```
DEPTNO ENAME
------ ----------
    10 CLARK
    10 KING
    10 MILLER

select deptno,
       count(*) as cnt,
       max(sal) as hi_sal,
       min(sal) as lo_sal
  from emp
 where deptno=10
 group by deptno

DEPTNO        CNT     HI_SAL     LO_SAL
------ ---------- ---------- ----------
    10          3       5000       1300
```

Kada ne biste imali mogućnost da grupišete sve zaposlene iz odeljenja 10, kako biste dobili podatke koje učitava drugi navedeni upit, morali biste ručno da ispitujete redove koji se odnose na to odeljenje (što je trivijalan posao kada imate tri reda, ali šta ako imate tri miliona redova?). Dakle, zbog čega bismo uopšte poželeli da formiramo grupe redova? Može biti više razloga da to činite; možda želite da utvrdite koliko grupa postoji ili koliko članova (redova) ima u svakoj grupi. Kao što se vidi iz navedenog jednostavnog primera, grupisanje omogućava da dobijete podatke o više redova iz tabele a da ih ne ispitujete jedan po jedan.

Definicija grupe u SQL-u

U matematici, grupa je definisana najčešće kao (G, •, e), gde je G skup, • je binarna operacija nad skupom G, a e je član skupa G. Po analogiji, grupu u SQL-u definisaćemo kao (G, e), gde je G skup rezultata jednog samostalnog upita koji sadrži odredbu GROUP BY, e je član skupa G, a važe sledeći aksiomi:

- Svako e u G je različito i predstavlja jedan ili više primeraka e.
- Za svako e u G, agregatna funkcija COUNT vraća vrednost > 0.

Skup rezultata je pomenut u definiciji SQL-ove grupe kako bi se istakla činjenica da želimo da definišemo šta su grupe, ali samo u kontekstu upita. Zbog toga možemo zameniti „e“ u oba aksioma s rečju „red“ jer, tehnički gledano, redovi u skupu rezultata upita jesu grupe.

Pošto su ti uslovi suštinski za ono što smatramo grupom, važno je dokazati da su ispunjeni (što ćemo učiniti pomoću nekoliko primera SQL upita).

Grupe nisu prazni skupovi

Po samoj definiciji, grupa mora da ima barem jednog člana (ili red). Ako tu tvrdnju prihvatimo kao istinitu, onda kažemo da se grupa ne može formirati od sadržaja prazne tabele. Kako bismo dokazali da je ta tvrdnja istinita, pokušaćemo da dokažemo da je neistinita. U narednom primeru pravimo praznu tabelu, a zatim pokušavamo da od redova prazne tabele formiramo grupe pomoću tri upita:

```
create table fruits (name varchar(10))

select name
  from fruits
 group by name

(no rows selected)


select count(*) as cnt
  from fruits
 group by name

(no rows selected)


select name, count(*) as cnt
  from fruits
 group by name

(no rows selected)
```

Kao što se vidi iz ovih upita, od sadržaja prazne tabele nije moguće formirati ono što se u SQL-u smatra grupom.

Groupe su jedinstvene

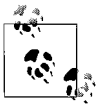
Sada ćemo dokazati da su grupe formirane pomoću upita koji sadrže odredbu GROUP BY, jedinstvene. U narednom primeru, tabeli FRUITS dodajemo pet redova, a zatim od njih formiramo grupe:

```
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Apple')
insert into fruits values ('Peach')

select *
  from fruits

NAME
----------
Oranges
Oranges
Oranges
```

```
Apple
Peach


select name
  from fruits
 group by name

NAME
----------
Apple
Oranges
Peach


select name, count(*) as cnt
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
```

Prvi upit pokazuje da se vrednost „Oranges“ triput pojavljuje u tabeli FRUITS. Međutim, drugi i treći upit (koji sadrže odredbu GROUP BY) prikazuju samo jedan primerak vrednosti „Oranges.“ Sva tri upita uzeta zajedno dokazuju da su redovi skupa rezultata (*e* u *G*, u definiciji) jedinstveni, a svaka vrednost u koloni NAME predstavlja jedan ili više svojih primeraka koji postoje u tabeli FRUITS.

Činjenica da su grupe jedinstvene važna je jer to uglavnom znači da listi kolona u odredbi SELECT nećete dodati rezervisanu reč DISTINCT ako upit sadrži odredbu GROUP BY.



Ni na koji način ne tvrdim da su GROUP BY i DISTINCT jedno te isto. To su dva potpuno različita koncepta. Samo tvrdim da će stavke navedene u odredbi GROUP BY biti jedinstvene u skupu rezultata i da je rezervisana reč DISTINCT u kombinaciji s GROUP BY suvišna.

Rezultat funkcije COUNT nije nikad nula

Upiti i rezultati u prethodnom odeljku takođe dokazuju poslednji aksiom da rezultat agregatne funkcije COUNT neće nikad biti nula ako je upotrebite u upitu koji sadrži odredbu GROUP BY i koji se izvršava u tabeli koja nije prazna. Ne bi trebalo da bude iznenađujuće to što rezultat prebrojavanja članova grupe ne može nikad da bude nula. Pošto smo već dokazali da se grupa ne može formirati od sadržaja prazne tabele, grupa mora imati barem jedan red. Ako postoji barem jedan red, rezultat prebrojavanja redova grupe uvek će biti barem 1.

Fregeov aksiom i Raselov paradoks

Za one koje to zanima, Fregeov *aksiom apstrakcije*, koji se zasniva na Kantorovom rešenju za definisanje članstva u skupovima s beskonačnim ili neprebrojivim brojem članova, određuje da, za dato svojstvo koje identifikuje članove skupa, postoji skup čiji su članovi isključivo elementi koji imaju to svojstvo. Izvor nevolja, kao što je naveo Robert Stol, „jeste neograničena upotreba načela apstrakcije". Bertrand Rasel je zatražio od Gotloba Fregea da razmotri skup čiji su članovi skupovi, a definišuće svojstvo im je da nisu članovi samih sebe.

Kao što je Rasel istakao, aksiom apstrakcije pruža previše slobode: kada članstvo u skupu definišete tako što zadajete uslov ili definišete svojstvo, može se pronaći protivrečnost. Da bi bolje objasnio kako se može pronaći protivrečnost, izmislio je „berberinovu slagalicu". Berberinova slagalica glasi ovako:

> U jednom gradu postoji berberin koji brije sve muškarce, i samo one muškarce koji se ne briju sami. Ako je to tačno, ko onda brije berberina?

Konkretniji primer je skup koji se može opisati kao:

Svi članovi x unutar skupa y koji ispunjavaju određeni uslov (P)

Matematička notacija za ovaj opis izgleda ovako:

```
{x ε y | P(x)}
```

Pošto opisani skup čine *samo oni x-ovi unutar y koji ispunjavaju uslov* (*P*), možda ćete smatrati kako je intuitivnije da skup opišete kao *x je član skupa y ako i samo ako x ispunjava određeni uslov* (*P*).

A sada ćemo definisati uslov *P*(*x*) kao da *x nije član x*:

```
( x ε x )
```

Skup je sada definisan kao *x je član skupa y ako i samo ako x nije član x*:

```
{x ε y | ( x ε x )}
```

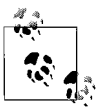
Možda vam Raselov paradoks još nije jasan, ali zapitajte se sledeće: da li skup može biti član samog sebe? Pretpostavimo da *x* = *y* i pogledajmo ponovo navedeni skup. Naredni skup može se definisati kao *y je član skupa y ako i samo ako y nije član skupa y*:

```
{y ε y | ( y ε y )}
```

(*nastavlja se*)

Jednostavno rečeno, Raselov paradoks nas dodvodi u situaciju da imamo skup koji istovremeno i jeste i nije član samog sebe, što je protivrečnost. Intuitivno se zaključuje da to ne predstavlja problem: kako skup može biti član samog sebe? Na kraju krajeva, skup svih knjiga nije knjiga. Pa kako onda ovaj paradoks postoji i kako može biti uzrok problema? To se može dogoditi ako razmotrite apstraktniju primenu teorije skupova. Na primer, „praktična“ primena Raselovog paradoksa može se pokazati ako razmotrite skup svih skupova. Ako prihvatimo postojanje takvog koncepta, onda takav skup, po samoj svojoj definiciji, mora biti član samog sebe (jer, na kraju krajeva, to je skup svih skupova). Šta se dešava kad primenite navedeni uslov $P(x)$ na skup svih skupova? Jednostavno rečeno, Raselov paradoks bi propisao da je skup svih skupova član samog sebe ako i samo ako nije član samog sebe – što je jasna protivrečnost.

Za one koje to zanima, Ernst Zermelo je razvio aksiomatsku šemu razdvajanja (poznatu kao aksiomatska šema podskupova ili aksiom specifikacije) koja elegantno zaobilazi Raselov paradoks u aksiomatskoj teoriji skupova.



Imajte u vidu da je ovde reč o upotrebi funkcije COUNT u kombinaciji sa odredbom GROUP BY, a ne o samostalnoj upotrebi te funkcije. Razume se, ako nad praznom tabelom izvršite upit koji poziva funkciju COUNT bez odredbe GROUP BY, rezultat će biti nula.

Paradoksi

> „Teško bi se našla veća nesreća koja može zadesiti pisca u naučnoj oblasti nego kad se deo temelja građevine koju je podigao zatrese nakon završetka posla... U tu situaciju me je postavilo pismo gospodina Bertranda Rasela, upravo u času kad je štampanje ove knjige bilo pred završetkom.“

Ovo je citat iz odgovora Gotloba Fregea kada je Bertrand Rasel otkrio protivrečnost u Fregeovom aksiomu apstrakcije u teoriji skupova.

Paradoksi često naizgled protivreče prihvaćenim teorijama ili idejama. U mnogim slučajevima te protivrečnosti su ograničene i mogu se „zaobići“ ili važe u toliko malom broju izuzetnih slučajeva da se slobodno mogu zanemariti.

Možda ste dosad shvatili da je suština pominjanja paradoksa bila namera da pokažemo kako postoji paradoks u vezi s našom definicijom grupe u SQL-u i izvesnost da taj paradoks moramo objasniti. Iako se sada bavimo grupama, naš krajnji cilj je razmatranje SQL upita. U odredbi GROUP BY, upit može sadržati više vrsta vrednosti, kao što su konstante, izrazi ili, što je uobičajenije, kolone iz određene tabele. Cena te fleksibilnosti je činjenica da je NULL ispravna „vrednost“ u SQL-u. NULL je problem pošto ga agregatne funkcije zanemaruju. Ali u tom slučaju, ako imamo tabelu s jednim redom koji sadrži samo NULL, šta će biti rezultat agregatne funkcije COUNT kada je upotrebimo u upitu koji sadrži odredbu GROUP BY? Po samoj definiciji, kada se odredba GROUP BY nalazi u upitu u kojem se poziva funkcija COUNT, rezultat

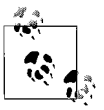
mora biti vrednost >= 1. Šta se u tom slučaju dešava s vrednostima koje funkcije kao što je COUNT zanemaruju i šta to znači za definiciju grupe? Pogledajte sledeći primer, koji ilustruje paradoks grupe koju čine vrednosti NULL (funkcija COALESCE je upotrebljena gde je neophodno, kako biste lakše razumeli rezultate):

```
select *
  from fruits

NAME
----------
Oranges
Oranges
Oranges
Apple
Peach


insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)

select coalesce(name,'NULL') as name
  from fruits

NAME
----------
Oranges
Oranges
Oranges
Apple
Peach
NULL
NULL
NULL
NULL
NULL


select coalesce(name,'NULL') as name,
       count(name) as cnt
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
NULL                0
Oranges             3
Peach               1
```

Na prvi pogled, izgledalo bi da postojanje NULL vrednosti u tabeli uvodi protivrečnost, ili paradoks, u našu definiciju SQL grupe. Srećom, ta protivrečnost nije razlog za brigu jer taj paradoks više utiče na praktičnu realizaciju agregatnih funkcija nego na našu definiciju. Pogledajte završni upit u skupu iz prethodnog pasusa; opis upita koji bi bio problematičan izgledao bi ovako:

> *Prebrojite koliko puta se svako ime voća pojavljuje u tabeli FRUITS* ili *prebrojite koliko članova ima svaka grupa*.

Ako pažljivije proučite prethodne naredbe INSERT, uočićete da imamo pet redova s vrednostima NULL, što znači da postoji NULL grupa s pet članova.



Iako NULL svakako ima svojstva po kojima se razlikuje od drugih vrednosti, to je ipak vrednost i može zapravo da čini grupu.

Kako bismo onda napisali upit čiji bi rezultat prebrojavanja bio 5 a ne 0, odnosno koji daje tražene podatke i pri tome je usklađen s našom definicijom grupe? Naredni primer pokazuje kako se može zaobići paradoks NULL grupe:

```
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
NULL                5
```

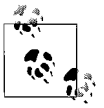
Rešenje je da upotrebite COUNT(*), a ne COUNT(NAME) da biste izbegli paradoks NULL grupe. Agregatne funkcije će zanemariti NULL vrednosti ako postoje u kolonama koje prosledite funkciji. Prema tome, da biste izbegli rezultat nula kada koristite funkciju COUNT, nemojte joj proslediti ime kolone; umesto toga, prosledite joj zvezdicu (*). Znak * čini da funkcija COUNT prebrojava redove umesto stvarnih vrednosti u kolonama, pa je zato nevažno da li redovi sadrže NULL ili neku drugu vrednost.

Drugi paradoks potiče od aksioma da je svaka grupa u skupu rezultata (za svako *e* unutar *G*) jedinstvena. Zbog prirode SQL-ovih skupova rezultata i tabela koji su tačnije definisani kao multiskupovi ili „vreće“, a ne kao „pravi“ skupovi (zato što su dozvoljeni duplirani redovi), moguće je formirati skup rezultata s dupliranim grupama. Pogledajte sledeće upite:

```
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 union all
select coalesce(name,'NULL') as name,
       count(*) as cnt
```

```
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
NULL                5
Apple               1
Oranges             3
Peach               1
NULL                5


select x.*
  from (
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
       ) x,
       (select deptno from dept) y

NAME              CNT
---------- ----------
Apple               1
Apple               1
Apple               1
Apple               1
Oranges             3
Oranges             3
Oranges             3
Oranges             3
Peach               1
Peach               1
Peach               1
Peach               1
NULL                5
NULL                5
NULL                5
NULL                5
```

Kao što se iz upita uočava, grupe se zapravo ponavljaju u konačnim rezultatima. Srećom, nema puno razloga za brigu jer je u pitanju samo delimičan paradoks. Prvo svojstvo grupe određuje da za (G,e), G je skup rezultata jednog, odnosno samostalnog upita koji sadrži odredbu GROUP BY. Jednostavno rečeno, skup rezultata svakog upita koji sadrži odredbu GROUP BY slaže se s našom definicijom grupe. Tek kad kombinujete skupove rezultata dva upita koji sadrže odredbe GROUP BY da biste formirali multiskup, mogu se pojaviti duplirane grupe. Prvi upit u prethodnom primeru sadrži operator UNION ALL, što nije operacija koja daje skup, već multiskup, a takođe dvaput poziva odredbu GROUP BY, čime izvršava dva upita.



Ako upotrebite operator UNION, rezultat te operacije je skup, što znači da nećete imati duplirane grupe.

Drugi upit u prethodnoj grupi koristi Dekartov proizvod, što je izvodljivo samo ako prvo materijalizujete grupu a zatim izračunate Dekartov proizvod. To znači da je svaki samostalan upit sa odredbom GROUP BY usklađen s našom definicijom. Nijedan od dva primera ne oduzima ništa od definicije SQL grupe. Oni su navedeni celine radi, a i da biste imali u vidu kako je u SQL-u gotovo sve izvodljivo.

Povezanost između odredaba SELECT i GROUP BY

Nakon definisanja i dokazivanja koncepta grupe, vreme je da pređemo na praktičnije probleme u vezi sa upitima koji sadrže odredbu GROUP BY. Važno je da razumete povezanost između odredbe SELECT i odredbe GROUP BY kada u SQL-u grupišete podatke. Imajte u vidu sledeće: kada koristite agregatne funkcije kao što je COUNT, svaki element naveden na listi u odredbi SELECT, a koji nije zadat kao argument neke agregatne funkcije, mora biti deo neke grupe. Na primer, ako napišete odredbu SELECT nalik na sledeću:

```
select deptno, count(*) as cnt
  from emp
```

kolonu DEPTNO morate navesti i u odredbi GROUP BY:

```
select deptno, count(*) as cnt
  from emp
 group by deptno

DEPTNO    CNT
------ ------
    10      3
    20      5
    30      6
```

Izuzeci od ovog pravila su konstante, skalarne vrednosti izračunate pomoću funkcija koje sami napišete, analitičke funkcije i nekorelativni skalarni podupiti. Pošto se odredba SELECT obrađuje posle odredbe GROUP BY, te konstrukcije su dozvoljene u odredbi SELECT i ne moraju se (a u nekim slučajevima se i ne mogu) navesti u odredbi GROUP BY. Na primer:

```
select 'hello' as msg,
       1 as num,
       deptno,
       (select count(*) from emp) as total,
       count(*) as cnt
  from emp
 group by deptno
```

```
MSG   NUM DEPTNO TOTAL CNT
----- --- ------ ----- ---
hello   1     10    14   3
hello   1     20    14   5
hello   1     30    14   6
```

Neka vas ovaj upit ne zbunjuje. Elementi navedeni na listi u odredbi SELECT, ali kojih nema na listi u odredbi GROUP BY, ne utiču na promenu ukupnog broja redova po odeljenju (kolona CNT), niti na vrednosti u koloni DEPTNO. Na osnovu rezultata ovog upita, možemo preciznije definisati pravilo o elementima koji se moraju navesti i na listi u odredbi SELECT, i u odredbi GROUP BY kada se koriste agregatne funkcije:

> Stavke na listi u odredbi SELECT koje mogu da izmene grupu ili izmene vrednost koju vraća agregatna funkcija, moraju se navesti i u odredbi GROUP BY.

Dodatne stavke na listi odredbe SELECT u navedenom primeru upita ne menjaju vrednosti u koloni CNT ni za jednu grupu (odeljenje), niti menjaju same grupe.

Bilo bi korisno da se zapitamo: koji elementi u odredbi SELECT mogu da izmene grupisanje ili vrednost koju vraća agregatna funkcija? Odgovor je jednostavan: druge kolone iz tabele (ili tabela) iz kojih učitavate redove. Pogledajte šta bi se dogodilo ako dodate kolonu JOB upitu koji smo dosad razmatrali:

```
select deptno, job, count(*) as cnt
  from emp
 group by deptno, job

DEPTNO JOB       CNT
------ --------- ---
    10 CLERK       1
    10 MANAGER     1
    10 PRESIDENT   1
    20 CLERK       2
    20 ANALYST     2
    20 MANAGER     1
    30 CLERK       1
    30 MANAGER     1
    30 SALESMAN    4
```

Dodavanjem još jedne kolone, JOB, iz tabele EMP, menjamo grupu i skup rezultata upita; zbog toga moramo navesti kolonu JOB u odredbi GROUP BY zajedno s kolonom DEPTNO, inače se upit ne može izvršiti. Dodavanjem kolone JOB u odredbe SELECT/GROUP BY menjamo upit „Koliko zaposlenih ima u svakom odeljenju?“ u „Koliko zaposlenih ima po svakom radnom mestu i u svakom odeljenju?“ Obratite pažnju na to da su grupe jedinstvene; vrednosti iz kolona DEPTNO i JOB *pojedinačno* nisu jedinstvene, ali njihova kombinacija (što je navedeno i u odredbi GROUP BY i na listi u odredbi SELECT, pa zato imamo grupu, primera radi) jeste kombinacija 10 i CLERK pojavljuje se samo jedanput.

Ako na listu odredbe SELECT ne stavite ništa drugo osim agregatnih funkcija, u odredbi GROUP BY možete navesti svaku postojeću kolonu koju smatrate potrebnom. Pogledajte sledeća dva upita koji ilustruju tu činjenicu:

```
select count(*)
  from emp
 group by deptno

  COUNT(*)
----------
         3
         5
         6

select count(*)
  from emp
 group by deptno,job

  COUNT(*)
----------
         1
         1
         1
         2
         2
         1
         1
         1
         4
```

Nije obavezno da na listi odredbe SELECT navedete i druge elemente osim agregatnih funkcija, ali se tako često poboljšava razumljivost i upotrebljivost rezultata upita.

Uglavnom važi sledeće pravilo: kada u upitu imate odredbu GROUP BY i agregatne funkcije, svaki element naveden na listi odredbe SELECT [iz tabele (ili tabela) navedene u odredbi FROM], koji nije upotrebljen kao argument neke agregatne funkcije, morate takođe navesti u odredbi GROUP BY. Međutim, u MySQL-u postoji „mogućnost“ da prekršite to pravilo i da na listi odredbe SELECT navedete elemente [kolone iz tabele ili tabela iz kojih učitavate podatke] koji nisu zadati kao argumenti agregatnih funkcija, niti su navedeni u odredbi GROUP BY. Reč „mogućnost“ napisao sam između navodnika jer se radi o grešci koja vam se lako može potkrasti, pa vas zato molim da je izbegavate. U stvari, ako koristite MySQL i stalo vam je do tačnosti upita koje pišete, predlažem vam da zaboravite tu tzv. „mogućnost“.

Analitičke funkcije

Pošto shvatite koncept grupisanja i upotrebe agregatnih funkcija u SQL-u, lako ćete razumeti i *analitičke funkcije* (engl. *window functions*). Analitičke funkcije, slično agregatnim funkcijama, deluju na definisan skup (ili grupu) redova, ali umesto da vrate samo jednu vrednost po grupi, mogu da vrate više vrednosti za svaku grupu.

Grupa redova s kojima funkcija radi zove se *okvir* ili *prozor* (engl. *window*), od čega potiče ime „window functions“. U DB2 se takve funkcije zovu *OLAP* (od *online analytic processing*) funkcij, u Oracleu su one poznate kao *analitičke funkcije*, a u ISO standardu za SQL zovu se „funkcije prozora (engl. *window function*)“. U Prevodu ove knjige koristili smo izraz „analitičke funcije“.

Jednostavan primer

Pretpostavimo kako želite da prebrojite koliko ima zaposlenih u svim odeljenjima. Uobičajen metod da to uradite jeste upit u kojem ćete upotrebiti funkciju COUNT(*) za celu tabelu EMP:

```
select count(*) as cnt
  from emp

   CNT
------
    14
```

To je vrlo jednostavno, ali često će vam zatrebati takvi podaci izračunati na osnovu redova koji ne čine grupu ili čine drugačiju grupu. Analitičke funkcije pružaju jednostavna rešenja za tu vrstu problema. Na primer, naredni upit pokazuje kako da pomoću analitičke funkcije izračunate zbirne podatke (ukupan broj zaposlenih), pojedinačno u svakom redu tabele (jedan po zaposlenom):

```
select ename,
       deptno,
       count(*) over( ) as cnt
  from emp
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10     14
KING           10     14
MILLER         10     14
SMITH          20     14
ADAMS          20     14
FORD           20     14
SCOTT          20     14
JONES          20     14
ALLEN          30     14
BLAKE          30     14
MARTIN         30     14
JAMES          30     14
TURNER         30     14
WARD           30     14
```

U ovom primeru poziva se analitička funkcija COUNT(*) OVER( ). Rezervisana reč OVER pokazuje da se funkcija COUNT poziva kao analitička, a ne kao agregatna funkcija. Standard za SQL dozvoljava da sve agregatne funkcije budu i analitičke, a rezervisana reč OVER omogućava da se pri upotrebi u SQL-u pravi razlika između njih.

Dakle, šta je tačno uradila analitička funkcija COUNT(*) OVER ()? Za svaki red koji je upit učitao, funkcija je vratila *ukupan broj redova* u tabeli. Kao što prazne zagrade pokazuju, rezervisana reč OVER prihvata dodatne odredbe koje uslovljavaju redove na koje će delovati analitička funkcija. Ako nije zadata nijedna takva odredba, analitička funkcija obrađuje sve redove u skupu; zbog toga se u svakom redu rezultata upita ponavlja vrednost 14.

Nadam se da počinjete da uviđate koliko su analitičke funkcije korisne – omogućavaju da radite s različitim nivoima grupisanja za isti red. Tokom čitanja preostalog dela ovog dodatka još bolje ćete uvideti koliko ta mogućnost može biti izuzetno korisna.

Redosled obrade

Pre nego što dublje „zaronimo“ u odredbu OVER, važno je imati u vidu da se u SQL-u analitičke funkcije pozivaju kao poslednji korak pre odredbe ORDER BY. Kao primer izvršavanja analitičke funkcije u poslednjem koraku, uzmimo upit iz prethodnog odeljka kojem ćemo dodati odredbu WHERE da bismo zanemarili zaposlene iz odeljenja (kolona DEPTNO) 20 i 30:

```
select ename,
       deptno,
       count(*) over( ) as cnt
  from emp
 where deptno = 10
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
```

Vrednost u koloni CNT u svakom redu više nije 14, nego 3. U ovom primeru odredba WHERE ograničava skup rezultata na tri reda, pa zato analitička funkcija prebrojava samo tri reda (u trenutku obrade odredbe SELECT upita, analitička funkcija „vidi“ samo tri reda). Iz ovog primera jasno je da se analitičke funkcije izvršavaju tek posle obrade odredaba kao što su WHERE i GROUP BY.

Particije

Odredba PARTITION BY omogućava definisanje *particije* (engl. *partition*), to jest grupe redova s kojima će raditi analitička funkcija. Kao što ste već videli, ako zadate prazne zagrade, particija s kojom će raditi analitička funkcija biće ceo skup rezultata. Odredbu PARTITION BY možete zamisliti kao „pokretni GROUP BY“ jer, za razliku od uobičajene odredbe GROUP BY, grupa koja se formira odredbom PARTITION BY nije jedinstvena u skupu rezultata. Odredbu PARTITION BY možete upotrebiti za izračunavanje određene zbirne vrednosti za definisanu grupu redova

(koja se ponovo izračunava za novu grupu); i umesto da jedna grupa predstavlja sve primerke te vrednosti koji postoje u tabeli, učitava se svaki primerak pojedinačno (svaki član u svakoj grupi). Pogledajte sledeći upit:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as cnt
  from emp
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
SMITH          20      5
ADAMS          20      5
FORD           20      5
SCOTT          20      5
JONES          20      5
ALLEN          30      6
BLAKE          30      6
MARTIN         30      6
JAMES          30      6
TURNER         30      6
WARD           30      6
```

Upit i dalje učitava 14 redova, ali sada se funkcija COUNT poziva za svako odeljenje kao rezultat odredbe PARTITION BY DEPTNO. Svi zaposleni u istom odeljenju (istoj particiji) imaće istu vrednost u koloni CNT, zato što se zbirna vrednost (prebrojavanje) neće ponovo izračunavati dok se ne promeni odeljenje. Obratite pažnju na to da upit prikazuje izračunate podatke u svakoj grupi, zajedno s članovima grupe. Taj upit možete smatrati efikasnijom verzijom narednog upita:

```
select e.ename,
       e.deptno,
       (select count(*) from emp d
         where e.deptno=d.deptno) as cnt
  from emp e
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
SMITH          20      5
ADAMS          20      5
FORD           20      5
SCOTT          20      5
JONES          20      5
ALLEN          30      6
```

```
BLAKE          30      6
MARTIN         30      6
JAMES          30      6
TURNER         30      6
WARD           30      6
```

Lepa je odlika odredbe PARTITION BY to što svoje proračune obavlja nezavisno od drugih analitičkih funkcija i u istoj naredbi SELECT može da formira particije po različitim kolonama. Pogledajte sledeći upit, koji učitava prezime zaposlenog, šifru odeljenja u kojem radi, ukupan broj zaposlenih u tom odeljenju, naziv njegovog radnog mesta i ukupan broj zaposlenih koji rade na istom radnom mestu:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as dept_cnt,
       job,
       count(*) over(partition by job)    as job_cnt
  from emp
 order by 2

ENAME      DEPTNO DEPT_CNT JOB       JOB_CNT
---------- ------ -------- --------- -------
MILLER         10        3 CLERK           4
CLARK          10        3 MANAGER         3
KING           10        3 PRESIDENT       1
SCOTT          20        5 ANALYST         2
FORD           20        5 ANALYST         2
SMITH          20        5 CLERK           4
JONES          20        5 MANAGER         3
ADAMS          20        5 CLERK           4
JAMES          30        6 CLERK           4
MARTIN         30        6 SALESMAN        4
TURNER         30        6 SALESMAN        4
WARD           30        6 SALESMAN        4
ALLEN          30        6 SALESMAN        4
BLAKE          30        6 MANAGER         3
```

Iz ovog skupa rezultata vidi se da svi zaposleni u istom odeljenju imaju istu vrednost u koloni DEPT_CNT, a svi zaposleni koji rade na istom radnom mestu (bez obzira na odeljenje) imaju istu vrednost u koloni JOB_CNT.

Trebalo bi da dosad bude jasno da odredba PARTITION BY deluje slično odredbi GROUP BY, ali pri tome na nju ne utiču drugi elementi navedeni u odredbi SELECT a da ne mora se dodavati i odredba GROUP BY.

Efekti NULL vrednosti

Slično odredbi GROUP BY, odredba PARTITION BY smešta sve NULL vrednosti u jednu grupu, to jest particiju. Zbog toga je uticaj NULL vrednosti na rezultate odredbe PARTITION BY sličan onom na odredbu GROUP BY. Naredni upit, pomoću analitičke funkcije prebrojava zaposlene po iznosu isplaćene provizije (ako je provizija NULL, prikazuje se –1 da bi bilo razumljivije):

```
select coalesce(comm,-1) as comm,
       count(*)over(partition by comm) as cnt
  from emp

  COMM        CNT
------ ----------
     0          1
   300          1
   500          1
  1400          1
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
```

Budući da je zadato COUNT(*), ta funkcija prebrojava redove. Vidi se da je za desetoro zaposlenih iznos provizije NULL. Međutim, ako umesto simbola * zadate kolonu COMM, rezultati su prilično drugačiji:

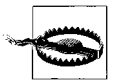
```
select coalesce(comm,-1) as comm,
       count(comm)over(partition by comm) as cnt
  from emp

COMM        CNT
---- ----------
   0          1
 300          1
 500          1
1400          1
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
```

U ovom upitu zadato je COUNT(COMM), što znači da se u koloni COMM prebrojavaju samo vrednosti različite od NULL. Postoji jedan zaposleni čiji je iznos provizije 0, jedan kome je isplaćena provizija od 300 itd. Obratite pažnju na broj onih čiji je iznos provizije NULL! Ima ih 0. Zašto? Zato što agregatne funkcije zanemaruju NULL vrednosti, ili tačnije rečeno, agregatne funkcije obrađuju samo vrednosti koje nisu NULL.

Kada koristite funkciju COUNT, razmislite da li ćete ubrojati i redove koji sadrže NULL vrednosti. Zadajte COUNT(ime kolone) da biste zanemarili NULL vrednosti. Zadajte COUNT(*) ako želite da ubrojite i NULL vrednosti (jer u tom slučaju ne prebrojavate stvarne vrednosti u koloni, već redove tabele).

Kada je redosled važan

Ponekad je redosled kojim analitička funkcija obrađuje redove važan za rezultate koje želite da dobijete pomoću upita. Iz tog razloga, sintaksa analitičkih funkcija prihvata odredbu ORDER BY koja se može zadati unutar odredbe OVER. Upotrebite odredbu ORDER BY da biste zadali redosled redova unutar particija (imajte u vidu sledeće: kada izostavite odredbu PARTITION BY, „particija“ znači ceo skup rezultata).



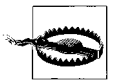
Neke analitičke funkcije *zahtevaju* da zadate redosled redova unutar particija, što znači da je za neke analitičke funkcije obavezna odredba ORDER BY.

Kada u odredbi OVER analitičke funkcije zadate odredbu ORDER BY, time zadajete dve stvari:

1. redosled redova unutar particije;
2. koji će redovi biti obuhvaćeni proračunom.

Pogledajte sledeći upit, koji izračunava ukupan zbir i tekući zbir plata zaposlenih u odeljenju (kolona DEPTNO) 10:

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over( ) as total2,
       sum(sal)over(order by hiredate) as running_total
  from emp
 where deptno=10

DEPTNO ENAME  HIREDATE      SAL TOTAL1 TOTAL2 RUNNING_TOTAL
------ ------ ----------- ----- ------ ------ -------------
    10 CLARK  09-JUN-1981  2450   8750   8750          2450
    10 KING   17-NOV-1981  5000   8750   8750          7450
    10 MILLER 23-JAN-1982  1300   8750   8750          8750
```



Samo da bih vas malo namučio, dodao sam funkciju SUM s praznim zagradama. Obratite pažnju na to da kolone TOTAL1 i TOTAL 2 sadrže iste vrednosti. Zašto? U ovom slučaju takođe, odgovor na pitanje leži u redosledu izvršavanja analitičkih funkcija. Odredba WHERE filtrira skup rezultata tako da se za izračunavanje ukupnog zbira uzimaju u obzir samo redovi u kojima kolona DEPTNO sadrži vrednost 10. U ovom slučaju imamo samo jednu particiju – ceo skup rezultata, koji se sastoji isključivo od plata iz odeljenja 10. Zbog toga kolone TOTAL1 i TOTAL2 sadrže jednake vrednosti.

Ako pogledate vrednosti u koloni SAL, lako ćete razumeti odakle potiču vrednosti u koloni RUNNING_TOTAL. Na osnovu njih možete i sami izračunati tekući zbir. Ali važnije je da shvatite zbog čega se dodavanjem odredbe ORDER BY u odredbu OVER dobija tekući zbir. Razlog je sledeći: kada u odredbi OVER zadate ORDER BY, time zadajete podrazumevani „pokretni" ili „klizni" prozor unutar particije, mada ne na izričit način. Odredba ORDER BY HIREDATE čini da se sabiranje završava s datumom u koloni HIREDATE tekućeg reda.

Sledeći upit je sličan prethodnom, ali sadrži odredbu RANGE BETWEEN (o njoj će više biti reči u nastavku teksta); ona izričito definiše redove koji će se obrađivati kao rezultat odredbe ORDER BY HIREDATE:

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over( ) as total2,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                      and current row) as running_total
  from emp
 where deptno=10

DEPTNO ENAME  HIREDATE      SAL TOTAL1 TOTAL2 RUNNING_TOTAL
------ ------ ----------- ----- ------ ------ -------------
    10 CLARK  09-JUN-1981  2450   8750   8750          2450
    10 KING   17-NOV-1981  5000   8750   8750          7450
    10 MILLER 23-JAN-1982  1300   8750   8750          8750
```

Odredba RANGE BETWEEN zadata u ovom upitu zove se u ANSI standardu *odredba* opsega (engl. *framing clause)*, što je i izraz koji ću koristiti u daljem tekstu. Sada bi trebalo da bude jasno zbog čega zadavanjem odredbe ORDER BY u odredbi OVER dobijamo tekući zbir; u upitu smo zadali (podrazumevano, ne izričito) da sabere sve redove počev od tekućeg i da u zbir uključi sve prethodne redove („prethodne" kako je definisano odredbom ORDER BY, što u ovom slučaju znači po redosledu vrednosti u koloni HIREDATE).

Odredba opsega

Odredba opsega iz upita u prethodnom odeljku primenićemo na skup rezultata, počev od prvog zaposlenog po datumu zapošljavanja, a to je CLARK.

1. Počev od iznosa CLARKOVE plate, 2450, i uključujući sve zaposlene koji su počeli da rade pre njega, izračunavamo ukupan zbir. Pošto je CLARK prvi zaposlen u odeljenju 10, ukupan zbir plata je jednak CLARKOVOJ plati, 2450, što je i prva vrednost koja se prikazuje u koloni RUNNING_TOTAL.
2. Prelazimo na sledećg zaposlenog po redosledu vrednosti u koloni HIREDATE, a to je KING, i ponovo primenjujemo odredbu opsega. Izračunavamo ukupan zbir u vrednosti koloni SAL počev od tekućeg reda, 5000 (KINGOVA plata), i ubrajamo sve prethodne redove (sve zaposlene koji su počeli da rade pre KINGA). Pošto je CLARK jedini koji je zaposlen pre KINGA, ukupan zbir je 5000 + 2450, odnosno 7450, što je sledeća vrednost prikazana u koloni RUNNING_TOTAL.
3. Prelazimo na MILLERA, poslednjeg zaposlenog u particiji definisanoj po vrednostima u koloni HIREDATE, i još jednom primenjujemo odredbu opsega. Izračunavamo ukupan zbir vrednosti u koloni SAL počev od tekućeg reda, 1300 (MILLEROVA plata), i ubrajamo sve prethodne redove (sve zaposlene koji su počeli da rade pre MILLERA). Pošto su i CLARK i KING zaposleni pre MILLERA, njihove plate se ubrajaju u iznos prikazan u koloni RUNNING_TOTAL za MILLERA: 2450 + 5000 + 1300 je 8750, što je vrednost prikazana u koloni RUNNING_TOTAL za MILLERA.

Kao što vidite, tekući zbir zapravo izračunava odredba opsega. Odredba ORDER BY definiše redosled izračunavanja a definiše i implicitnu oredbu opsega.

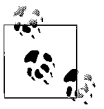
Okvirna odredba uglavnom omogućava definisanje različitih „podopsega“ podataka s kojima će se obavljati proračuni. Ti podopsezi mogu se zadati na više načina. Pogledajte sledeći upit:

```
select deptno,
       ename,
       sal,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                      and current row) as run_total1,
       sum(sal)over(order by hiredate
                     rows between 1 preceding
                      and current row) as run_total2,
       sum(sal)over(order by hiredate
                    range between current row
                      and unbounded following) as run_total3,
       sum(sal)over(order by hiredate
                     rows between current row
                      and 1 following) as run_total4
  from emp
 where deptno=10
```

```
DEPTNO ENAME    SAL RUN_TOTAL1 RUN_TOTAL2 RUN_TOTAL3 RUN_TOTAL4
------ ------ ----- ---------- ---------- ---------- ----------
    10 CLARK   2450       2450       2450       8750       7450
    10 KING    5000       7450       7450       6300       6300
    10 MILLER  1300       8750       6300       1300       1300
```

Neka vas ovaj upit ne zbuni jer nije tako komplikovan kao što izgleda. Već ste videli kako se izračunava kolona RUN_TOTAL1 i efekte odredbe opsega „UNBOUNDED PRECEDING AND CURRENT ROW“. Evo kratkog opisa onog što se odvija u drugim primerima:

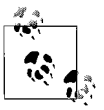
RUN_TOTAL2
: Umesto rezervisane reči RANGE, u ovoj odredbi opsega zadata je reč ROWS, što znači da se *okvir*, ili „prozor“, formira od zadatog broja redova. Izraz 1 PRECEDING znači da okvir počinje od reda koji neposredno prethodi tekućem redu. Opseg okvira se nastavlja do tekućeg reda, CURRENT ROW. Dakle, kolona RUN_TOTAL2 sadrži zbir plate tekućeg zaposlenog i onog koji mu neposredno prethodi prema redosledu vrednosti u koloni HIREDATE.



Kolone RUN_TOTAL1 i RUN_TOTAL2 sadrže iste vrednosti i za CLARKA i za KINGA. Zašto? Razmislite o tome koje se vrednosti sabiraju za njih dvoje, u svakoj od obe analitičke funkcije. Pažljivo razmislite, pa ćete doći do odgovora.

RUN_TOTAL3
: Analitička funkcija u koloni RUN_TOTAL3 deluje upravo na suprotan način od one u koloni RUN_TOTAL1; umesto da krene od tekućeg reda i da u zbir ubroji sve prethodne redove, sabiranje počinje od tekućeg reda a zatim se u zbir ubrajaju svi naredni redovi.

RUN_TOTAL4
: Analitička funkcija u ovoj koloni deluje na suprotan način od analitičke funkcije u koloni RUN_TOTAL2; umesto da vrednost iz tekućeg reda sabere s vrednošću iz reda koji mu prethodi, vrednost iz tekućeg reda sabira s vrednošću iz sledećeg reda.



Ako ste shvatili dosadašnja objašnjenja, nećete imati problema ni sa jednim receptom u ovoj knjizi. Međutim, ako noste se razumeli, vežbajte sa svojim primerima i podacima. Lično smatram da lakše učim kada pišem kôd u kojem koristim nove mogućnosti nego kada samo čitam o njima.

Finale

Kao završni primer dejstva odredbe opsega na rezultate upita, pogledajte sledeći upit:

```
select ename,
       sal,
       min(sal)over(order by sal) min1,
       max(sal)over(order by sal) max1,
```

```
       min(sal)over(order by sal
                    range between unbounded preceding
                      and unbounded following) min2,
       max(sal)over(order by sal
                    range between unbounded preceding
                      and unbounded following) max2,
       min(sal)over(order by sal
                    range between current row
                      and current row) min3,
       max(sal)over(order by sal
                    range between current row
                      and current row) max3,
       max(sal)over(order by sal
                     rows between 3 preceding
                      and 3 following) max4
  from emp

ENAME    SAL   MIN1   MAX1   MIN2   MAX2   MIN3   MAX3   MAX4
------ ----- ------ ------ ------ ------ ------ ------ ------
SMITH    800    800    800    800   5000    800    800   1250
JAMES    950    800    950    800   5000    950    950   1250
ADAMS   1100    800   1100    800   5000   1100   1100   1300
WARD    1250    800   1250    800   5000   1250   1250   1500
MARTIN  1250    800   1250    800   5000   1250   1250   1600
MILLER  1300    800   1300    800   5000   1300   1300   2450
TURNER  1500    800   1500    800   5000   1500   1500   2850
ALLEN   1600    800   1600    800   5000   1600   1600   2975
CLARK   2450    800   2450    800   5000   2450   2450   3000
BLAKE   2850    800   2850    800   5000   2850   2850   3000
JONES   2975    800   2975    800   5000   2975   2975   5000
SCOTT   3000    800   3000    800   5000   3000   3000   5000
FORD    3000    800   3000    800   5000   3000   3000   5000
KING    5000    800   5000    800   5000   5000   5000   5000
```

Razdvojimo upit na sastavne delove:

MIN1

Pošto u analitičkoj funkciji koja generiše ovu kolonu nije zadata odredba opsega, važi podrazumevana odredba opsega UNBOUNDED PRECEDING AND CURRENT ROW. Zašto kolona MIN1 sadrži 800 u svim redovima? Razlog je to što je najmanja plata prva po redosledu (ORDER BY SAL), a taj iznos ostaje najmanji, ili minimum, do kraja tabele.

MAX1

Vrednosti u koloni MAX1 znatno se razlikuju od onih u koloni MIN1. Zbog čega? Razlog je (ponovo) podrazumevana odredba opsega UNBOUNDED PRECEDING AND CURRENT ROW. U kombinaciji sa ORDER BY SAL, ta odredba opsega obezbeđuje da taj maksimalni iznos odgovara onome iz tekućeg reda.

Pogledajte prvi red, za SMITHA. Kada se SMITHOVA plata poredi sa svim prethodnim platama, MAX1 za SMITHA je SMITHOVA plata, pošto nema prethodnih plata za poređenje. Zatim prelazimo u sledeći red, JAMES, i poredimo

JAMESOVU platu sa svim prethodnim platama, što u ovom slučaju znači da je poredimo sa SMITHOVOM platom. Pošto je od ta dva iznosa JAMESOVA plata veća, to je maksimum. Ako tu logiku primenite na sve redove, videćete da je vrednost u koloni MAX1 u svakom redu iznos plate tekućeg zaposlenog.

MIN2 i MAX2
: Odredba opsega za ove kolone je UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING, što je isto kao kada zadate prazne zagrade. Zbog toga funkcije MIN i MAX deluju na sve redove u tabeli. Kao što biste i očekivali, vrednosti koje vraćaju funkcije MIN i MAX konstantne su za ceo skup rezultata, pa zato i te kolone sadrže iste vrednosti u svim redovima.

MIN3 i MAX3
: Odredba opsega za ove kolone je CURRENT ROW AND CURRENT ROW, što znači da se funkcijama MIN i MAX prosleđuje samo iznos plate tekućeg zaposlenog. Zato su u svakom redu vrednosti u kolonama MIN3 i MAX3 jednake onoj u koloni SAL. Ovo ste lako pogodili, zar ne?

MAX4
: Odredba opsega za kolonu MAX4 je 3 PRECEDING AND 3 FOLLOWING, što znači da se, za svaki red, uzimaju u obzir tri reda koji prethode tekućem i tri reda koji mu slede, kao i tekući red. Pozivanje funkcije MAX(SAL) daje najveći iznos plate iz tih sedam redova.

  Ako pogledate iznos u koloni MAX4 za zaposlenog MARTIN, shvatićete kako se primenjuje odredba opsega. MARTINOVA plata je 1250, a plate tri zaposlena koji mu prethode su: WARDOVA (1250), ADAMSOVA (1100) i JAMESOVA (950). Plate tri zaposlena koji slede iza MARTINA iznose: MILLEROVA (1300), TURNEROVA (1500) i ALLENOVA (1600). Od svih tih iznosa plata, uključujući i MARTINOVU, najveća je ALLENOVA, pa zato kolona MAX4 za MARTINA sadrži vrednost 1600.

Jasnoća + Performanse = Moć

Kao što vidite, analitičke funkcije su izuzetno moćne jer omogućavaju da pišete upite koji prikazuju i detaljne i zbirne (izračunate) podatke. Pomoću analitičkih funkcija možete pisati upite kraće i efikasnije od onih u kojima morate spajati tabele same sa sobom ili upotrebljavati skalarne podupite. Pogledajte sledeći upit, koji na jednostavan način odgovara na sva naredna pitanja: „Koliko ima zaposlenih u svakom odeljenju? Koliko različitih radnih mesta ima u svakom odeljenju (tj. koliko referenata radi u odeljenju 10)? Koliko je zaposlenih navedeno u tabeli EMP?“

```
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over ( ) as total
  from emp
```

```
DEPTNO JOB          EMP_CNT    JOB_CNT      TOTAL
------ --------- ---------- ---------- ----------
    10 CLERK              3          1         14
    10 MANAGER            3          1         14
    10 PRESIDENT          3          1         14
    20 ANALYST            5          2         14
    20 ANALYST            5          2         14
    20 CLERK              5          2         14
    20 CLERK              5          2         14
    20 MANAGER            5          1         14
    30 CLERK              6          1         14
    30 MANAGER            6          1         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
```

Da biste isti rezultat dobili bez analitičkih funkcija, potrebno je malo više truda:

```
select a.deptno, a.job,
       (select count(*) from emp b
         where b.deptno = a.deptno) as emp_cnt,
       (select count(*) from emp b
         where b.deptno = a.deptno and b.job = a.job) as job_cnt,
       (select count(*) from emp) as total
  from emp a
 order by 1,2

DEPTNO JOB          EMP_CNT    JOB_CNT      TOTAL
------ --------- ---------- ---------- ----------
    10 CLERK              3          1         14
    10 MANAGER            3          1         14
    10 PRESIDENT          3          1         14
    20 ANALYST            5          2         14
    20 ANALYST            5          2         14
    20 CLERK              5          2         14
    20 CLERK              5          2         14
    20 MANAGER            5          1         14
    30 CLERK              6          1         14
    30 MANAGER            6          1         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
```

Mada je očigledno da u ovom slučaju nije teško napisati rešenje bez analitičkih funkcija, ono svakako nije tako jasno, niti efikasno (nećete primetiti razlike u performansama kada tabela ima 14 redova, ali pokušajte te upite u tabeli s recimo, 1000 ili 10000 redova, pa ćete shvatiti prednosti upotrebe analitičkih funkcija nad višestrukim spajanjem tabela samih sa sobom i skalarnim podupitima).

Izgradnja osnove

Osim jasnoće i performansi, analitičke funkcije su korisne i za izgradnju „osnove“ za složenije upite „u stilu izveštaja“. Pogledajte naredni upit „u stilu izveštaja“ koji u lokalnom prikazu koristi analitičke funkcije a zatim rezultate tog prikaza grupiše i prosleđuje spoljašnjem upitu. Upotreba analitičkih funkcija omogućava da istovremeno prikazujete i detaljne i zbirne podatke, što je korisno u izveštajima. U narednom upitu se pomoću analitičkih funkcija izračunavaju ukupni brojevi nečega po raznim particijama. Pošto se zbirne vrednosti izračunavaju za više redova, lokalni prikaz učitava sve redove iz tabele EMP, koje zatim izrazi CASE u spoljašnjem upitu transponuju da bi se dobio izveštaj:

```
select deptno,
       emp_cnt as dept_total,
       total,
       max(case when job = 'CLERK'
                then job_cnt else 0 end)  as clerks,
       max(case when job = 'MANAGER'
                then job_cnt else 0  end) as mgrs,
       max(case when job = 'PRESIDENT'
                then job_cnt else 0  end) as prez,
       max(case when job = 'ANALYST'
                then job_cnt else 0  end) as anals,
       max(case when job = 'SALESMAN'
                then job_cnt else 0  end) as smen
  from (
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over ( ) as total
  from emp
       ) x
 group by deptno, emp_cnt, total

DEPTNO DEPT_TOTAL TOTAL CLERKS MGRS PREZ ANALS SMEN
------ ---------- ----- ------ ---- ---- ----- ----
    10          3    14      1    1    1     0    0
    20          5    14      2    1    0     2    0
    30          6    14      1    1    0     0    4
```

Navedeni upit prikazuje šifru odeljenja, ukupan broj zaposlenih u svakom odeljenju, ukupan broj zaposlenih u tabeli EMP i raspodelu zaposlenih po radnim mestima u svakom odeljenju. Sve to obavlja isti upit, bez dodatnog spajanja tabele same sa sobom i bez privremenih tabela!

Kao završni primer iznete tvrdnje da se pomoću analitičkih funkcija može dati odgovor na više pitanja istovremeno, pogledajte sledeći upit:

```
select ename as name,
       sal,
       max(sal)over(partition by deptno) as hiDpt,
       min(sal)over(partition by deptno) as loDpt,
       max(sal)over(partition by job) as hiJob,
       min(sal)over(partition by job) as loJob,
       max(sal)over( ) as hi,
       min(sal)over( ) as lo,
       sum(sal)over(partition by deptno
                        order by sal,empno) as dptRT,
       sum(sal)over(partition by deptno) as dptSum,
       sum(sal)over( ) as ttl
  from emp
 order by deptno,dptRT
```

```
NAME     SAL HIDPT LODPT HIJOB LOJOB    HI   LO  DPTRT DPTSUM    TTL
------ ----- ----- ----- ----- ----- ----- ---- ------ ------ ------
MILLER  1300  5000  1300  1300   800  5000  800   1300   8750  29025
CLARK   2450  5000  1300  2975  2450  5000  800   3750   8750  29025
KING    5000  5000  1300  5000  5000  5000  800   8750   8750  29025
SMITH    800  3000   800  1300   800  5000  800    800  10875  29025
ADAMS   1100  3000   800  1300   800  5000  800   1900  10875  29025
JONES   2975  3000   800  2975  2450  5000  800   4875  10875  29025
SCOTT   3000  3000   800  3000  3000  5000  800   7875  10875  29025
FORD    3000  3000   800  3000  3000  5000  800  10875  10875  29025
JAMES    950  2850   950  1300   800  5000  800    950   9400  29025
WARD    1250  2850   950  1600  1250  5000  800   2200   9400  29025
MARTIN  1250  2850   950  1600  1250  5000  800   3450   9400  29025
TURNER  1500  2850   950  1600  1250  5000  800   4950   9400  29025
ALLEN   1600  2850   950  1600  1250  5000  800   6550   9400  29025
BLAKE   2850  2850   950  2975  2450  5000  800   9400   9400  29025
```

Ovaj upit odgovara na naredna pitanja lako, efikasno i jasno (i bez dodatnog spajanja tabele EMP same sa sobom!). Upit poredi zaposlenog i iznos njegove plate da bi prikazao:

1. ko ima najveću platu od svih zaposlenih (HI)
2. ko ima najmanju platu od svih zaposlenih (LO)
3. ko ima najveću platu u svom odeljenju (HIDPT)
4. ko ima najmanju platu u svom odeljenju (LODPT)
5. ko ima najveću platu za svoje radno mesto (HIJOB)
6. ko ima najmanju platu za svoje radno mesto (LOJOB)
7. koliki je zbir svih plata (TTL)
8. koliki je zbir svih plata po odeljenju (DPTSUM)
9. koliki je tekući zbir svih plata po odeljenju (DPTRT).