TABLIČNE FUNKCIJE NA BAZI ZA BOLJE PERFORMANSE ETL-A

Nataša Dvoršak
ULJANIK IRI d.o.o., Pula
e-mail: natasa.dvorsak@uljanik.hr

SAŽETAK

Postupak generiranja skladišta podataka obuhvaća transformaciju kompleksnih skupova podataka. Tablične funkcije na Oracle bazi mogu imati široku primjenu, od enkapsulacije poslovnih pravila na bazi, do modularnog izvođenje kompleksnih SQL upita s ciljem poboljšanja performansi. U prezentaciji će se prikazati mogućnosti poboljšanja performansi ETL-a pomoću "pipelined" opcije izvođenja tabličnih funkcija. Zanimljivo je i kako se popularni Map Reduce model programiranja može realizirati direktno na Oracle bazi uz pomoć tabličnih funkcija.

Improving ETL performance with table functions

The process of generating a data warehouse includes the transformation of complex data sets. Table functions in Oracle database can have broad applications, encapsulation of business logic, and modular construction of complex SQL queries to improve performance. The presentation will show the possibilities of improving ETL performance using pipelined table functions. It is interesting how popular Map Reduce programming model can be implemented directly into Oracle database using table functions.

UVOD

Posljednjih godina bilježi se linearni rast količine podataka. Statistički podaci kažu da je 2011.g. proizvedeno deset puta više podataka nego 2006 [2]. Obrada podataka serijskim načinom ponekad nije izvediva u 86400 sekundi koje su dnevno na raspolaganju. U radu će se dati osvrt na paralelne mogućnosti rada na Oracle bazi korištenjem "pipelined" PL/SQL tabličnih funkcija. Uobičajeno je da ETL (Extract Transform Load) procesi zahtijevaju i nekoliko međukoraka u transformaciji podataka do njihovog završnog oblika. Uporabom baznih tablica za pohranjivanje međurezultata dolazi do značajnog povećenja diskovnog prometa. Brža alternativa mogu biti pipelined tablične funkcije koje nude prečac kojim se iz eksterne tablice podaci direktno upisuju u odredišnu tablicu.

1 PARALELNE MOGUĆNOSTI RADA NA ORACLE BAZI

1.1 Paralelno izvršenje (Parallel Execution)

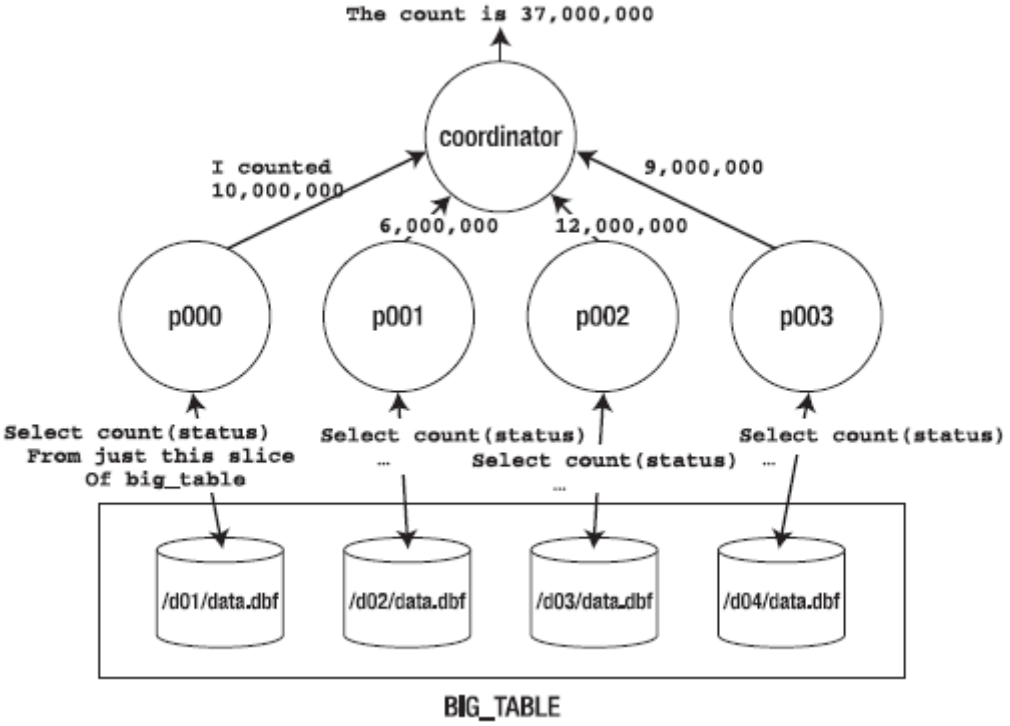
Paralelno izvršenje (parallel execution) je mogućnost enterprise edicije(Enterprise Edition) Oracle baze. Uvedena je u verziji 7.1.6. Oracle baze. Ukratko, to je mogućnost fizičkog dijeljenja dužeg serijskog procesa na manje dijelove koji se izvode istovremeno. Prije osvrta na PL/SQL rješavanje paralelnog izvođenja potrebno je spomenuti neke od mogućnosti paraleliziranja SQL, DML i DDL operacija bez posebnog programiranja. Paralelno izvršenje može se primijeniti na različite operacije čitanja i manipuliranja podataka na Oracle bazi. U ovom radu osvrnuti ćemo se na sljedeće tri mogućnosti palalelnog izvršavanja[1]:

- **Parallel query**: Oracle baza ima mogućnost da izvođenja jednog upita izvede pomoću nekoliko procesa operativnog sustava (npr. na Linuxu), odnosno dretvi (npr. na Windowsu). Ta je mogućnost primjenjiva za upite koji uključuju pretragu po čitavoj tablici ili particijama (full table scan), kao i za pretrage po čitavom indeksu (index fast full scan). Agregacije i skupovne operacije kao `GROUP BY`, `NOT IN`, `UNION`, `UNION ALL`, `CUBE`, `ROLLUP` mogu se također paralelizirati.
- **Parallel DML**: `INSERT AS SELECT`, `UPDATE`, `DELETE` i `MERGE` operacije mogu se izvršavati paralelno.
- **Parallel DDL**: `CREATE TABLE AS SELECT`, `CREATE INDEX`, `REBUILD INDEX`, `REBUILD INDEX PARTITION`, `MOVE`, `SPLIT` ili `COALESCE PARTITION` mogu se pokrenuti paralelno.

1 Postoje još i sljedeće mogućnosti: Parallel Data Loading, Parallel Recovery, Parallel Replication

Kako Oracle implementira paralelno izvršenje naredbi? Posao izvršavanja SQL izraza podijeli se na manje dijelove, svaki od kojih se odvija kao izdvojeni proces. Sesija iz koje se pokreće naredba preuzima ulogu koordinatora koji dinamički podijeli proces na manje dijelove. Zatim pribavlja dovoljan broj paralelnih procesa koji se nazivaju "parallel slave processes" ili "parallel execution servers". Svakom od njih dodjeli dio posla, prikuplja njihove rezultate i vraća ih korisničkom procesu. Kada je posao obavljen koordinator oslobađa zavisne procese.

Na sljedećoj slici prikazana je skica paralelnog izvršenja SQL upita. Potrebno je napomenuti da broj paralelnih procesa i datoteka podataka ne mora biti u odnosu 1:1 kao na skici, tj. nisu u direktnoj vezi. Podaci tablice mogu npr. biti pohranjeni u jednoj datoteci, a upit se i tada može pokrenuti paralelno.



Slika 1: Skica paralelnog izvršenja upita "select count(status) ..." [4]

U nastavku je primjer poziva paralelnog izvršenja SQL upita i praćenja rada analizom podataka iz kataloga.

```
select /*+ parallel (l,10)*/ l.* from ad_log_in l;

SQL> select sid, username, program
 2   from v$session
 3   where sid in (select sid from v$px_session where qcsid = 250);

     SID USERNAME                       PROGRAM
  ------ ------------------------------ ----------------------------
     173 WORKBOSS                       oracle@busrv01 (P004)
     174 WORKBOSS                       oracle@busrv01 (P006)
     193 WORKBOSS                       oracle@busrv01 (P002)
     195 WORKBOSS                       oracle@busrv01 (P005)
     196 WORKBOSS                       oracle@busrv01 (P007)
     197 WORKBOSS                       oracle@busrv01 (P001)
     200 WORKBOSS                       oracle@busrv01 (P003)
     224 WORKBOSS                       oracle@busrv01 (P008)
     230 WORKBOSS                       oracle@busrv01 (P000)
     250 WORKBOSS                       PlSqlDev.exe

  10 rows selected
```

Postavke stupnja paralelnog izvršenja, tzv. "degree of parallelism" ili DOP mogu se definirati na tri razine:

1. **Naredba** (statement)
   Uporabom hinta PARALLEL ili PARALLEL_INDEX. Npr.

```
select /*+ parallel(emp,4) */ count(*) from emp;
SELECT /*+ PARALLEL_INDEX(mytab, myindex, 3) */ stuff FROM mytab;
```

2. **Instanca**
   Izmjenom parametara sesije, npr.

```
alter session force parallel query;
```

3. **Objekt**
   Izmjenom u definiciji tablice ili indeksa, npr.:

```
alter table emp parallel (degree 4);
```

```
CREATE INDEX
   myidx
   ON
   customer(sex, hair_color, customer_id)
PARALLEL 35;
```

```
ALTER INDEX myidx parallel 35;
```

U nastavku je tablica u kojoj je prikazano kojim prioritetom Oracle prihvaća postavke paralelnog izvršenja (DOP).

Tabela 1: Prioritet postavki paralelnog izvršenja [5]

Table 25–3 Parallelization Priority Order: By Clause, Hint, or Underlying Table/Index Declaration

Parallel Operation	PARALLEL Hint	PARALLEL Clause	ALTER SESSION	Parallel Declaration
Parallel query table scan (partitioned or nonpartitioned table)	1) PARALLEL		2) FORCE PARALLEL QUERY	3) of table
Parallel query index range scan (partitioned index)	1) PARALLEL_ INDEX		2) FORCE PARALLEL QUERY	2) of index
Parallel UPDATE or DELETE (partitioned table only)	1) PARALLEL		2) FORCE PARALLEL DML	3) of table being updated or deleted from
INSERT operation of parallel INSERT... SELECT (partitioned or nonpartitioned table)	1) PARALLEL of insert		2) FORCE PARALLEL DML	3) of table being inserted into
SELECT operation of INSERT ... SELECT when INSERT is parallel	Takes degree from INSERT statement	Takes degree from INSERT statement	Takes degree from INSERT statement	Takes degree from INSERT statement
SELECT operation of INSERT ... SELECT when INSERT is serial	1) PARALLEL			2) of table being selected from
CREATE operation of parallel CREATE TABLE ... AS SELECT (partitioned or nonpartitioned table)	Note: Hint in the SELECT clause does not affect the CREATE operation	2)	1) FORCE PARALLEL DDL	
SELECT operation of CREATE TABLE ... AS SELECT when CREATE is parallel	Takes degree from CREATE statement	Takes degree from CREATE statement	Takes degree from CREATE statement	Takes degree from CREATE statement
SELECT operation of CREATE TABLE ... AS SELECT when CREATE is serial	1) PARALLEL or PARALLEL_ INDEX			2) of querying tables or partitioned indexes

Prije implementiranja paralelnih alternativa dobro je sa sigurnošću utvrditi jesu li serijske operacije dovoljno efikasno primijenjene. Većina tehnika paralelnog rada povećava efikasnost za manje od 100%, a zahtijeva korištenje dodatnih sistemskih resursa. Stoga je potrebno dobro odvagnuti koristi, posebno kod sustava gdje je zauzetost kapaciteta blizu gornjih granica.

1.2 Paralelno izvršenje SQL upita

Ovo je najčešće korištena mogućnost paralelnog izvršavanja na Oracle bazi. Ovaj oblik paralelnog izvođenja može značajno smanjiti vrijeme potrebno za izvršavanje upita nad velikim skupom podataka, ali nije primjenjiv za sve upite. Da bi se SQL upit izvodio paralelno potrebno je zadovoljiti sljedeće uvjete:

- Barem jednoj tablici iz upita pristupa se s "full table scan", ili se indeksu pristupa s "range scan" koji uključuje više particija.
- Uz "full table scan" pristup potrebno je koristiti PARALLEL hint točno specificirajući tablicu:

  ```
  select /*+ parallel(emp,4) */ count(*) from emp;
  ```

  ili tablica treba imati "parallel" deklaraciju u svojoj definiciji:

  ```
  alter table emp parallel (degree 4);
  ```

- Ako se izvršenje oslanja na "index range scan" koji se grana na više particija, treba koristiti PARALLEL_INDEX hint specificirajući odgovarajući indeks, ili indeks mora imati paralelnu deklaraciju u svojoj definiciji.

1.3 Paralelno izvršenje DML naredbi

Kada se izdaje DML naredba kao INSERT, UPDATE, ili DELETE, Oracle primjenjuje skup pravila kako bi odredio da li se naredba može paralelizirati. Za UPDATE i DELETE naredbe vrijede jednaka pravila, dok INSERT naredbe imaju svoj skup pravila.

Pravila za UPDATE i DELETE naredbe su:

- Oracle može paralelno izvoditi UPDATE i DELETE na particioniranim tablicama, ali samo kada je u operaciju uključeno više particija.
- Ne može se paralelizirati UPDATE ili DELETE operacija ne neparicioniranim tablicama ili kada ta operacija uključuje samo jednu particiju.

Pravila za INSERT naredbe su:

- Standardna INSERT naredba koja koristi VALUES klauzu ne može se paralelizirati
- Oracle može paralelizirati samo INSERT . . . SELECT . . . FROM naredbe.

DML naredbe izvršavati će se paralelno samo ako se u okviru sesije eksplicitno to zatraži, tj. postavi se parametar sesije (ALTER SESSION ENABLE PARALLEL DML). U nastavku se prikazuje primjer poziva paralelnog izvršenja DML naredbe:

```
alter session enable parallel dml;
insert /*+ parallel (emp_big,4,1) */ into emp_big select * from emp;
commit;
alter session disable parallel dml;
```

1.4 Paralno izvršenje DDL naredbi

Poziv paralelnog izvršenja DDL naredbi može se primijeniti na tablice i indekse bez obzira na to da li su particionirani. Nad neparticioniranim tablicama i indeksima samo se ovi tipovi DDL naredbi mogu paralelno izvršavati:

```
CREATE TABLE...AS SELECT
CREATE INDEX
ALTER INDEX...REBUILD
```

Kada se radi s particioniranim tablicama ili indeksima podrška za paralelni rad je nešto šira.

Sljedeći oblici DDL naredbi mogu se za takve slučajeve paralelno izvršavati:

```
CREATE TABLE...AS SELECT
ALTER TABLE...MOVE PARTITION
ALTER TABLE...SPLIT PARTITION
CREATE INDEX
ALTER INDEX...REBUILD PARTITION
ALTER INDEX...SPLIT PARTITION
```

Potrebno je napomenuti kako za tablice koje sadrže LOB ili objektne kolone Oracle ne može primijeniti paralelni rad DDL naredbi. U nastavku je primjer poziva paralelnog izvršenja DDL naredbi:

```
create table big_emp parallel (degree 4) as select * from emp;
```

```
CREATE INDEX emp_ix ON emp (emp_id)
TABLESPACE ind
STORAGE (INITIAL 1M NEXT 1M PCTINCREASE 0 MAXEXTENTS 20)
PARALLEL (DEGREE 4);
```

2 PL/SQL I PARALELNE MOGUĆNOSTI IZVRŠENJA

2.1 Paralelno izvršavanje PL/SQL-a

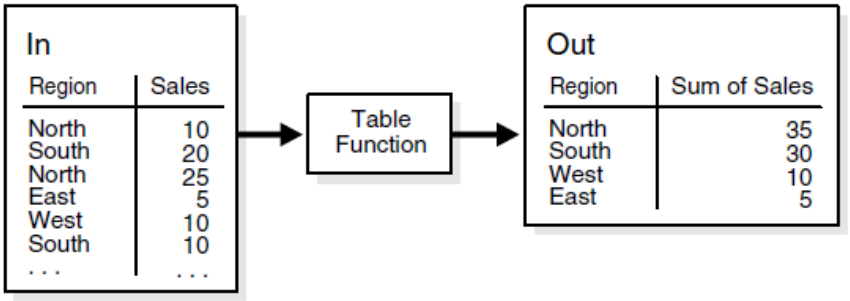
Postoji nekoliko mogućnosti implementiranja paralelnog izvršavanja PL/SQL procedura. Na Oracle bazi konkurentne aktivnosti događaju se kroz sesije koje se obično oslanjaju na jednu konekciju i proces operativnog sustava. Raspoložive su sljedeće opcije:

- **Parallel Execution Servers:** To su procesi operativnog sustava koje baza automatski podešava kako bi podržala paralelno izvršenje SQL naredbi. Parametrima `parallel_min_servers` i `parallel_max_servers` podešava se fond raspoloživih procesa operativnog sustava pojedine instance baze. Ova opcija koristi se u paralelnim mogućnostima izvršenja SQL naredbi, a koriste je i tablične funkcije u svom paralelnom načinu rada.
- **Job Queue Processes:** Uporabom paketa `dbms_job` i `dbms_scheduler` PL/SQL procedure mogu se poslati na izvođenje u redu čekanja. Njihovim izvršavanjem, na asinkron način, upravlja server zadužen za procese u redu čekanja. Ne postoji zagarantirani raspoloživi broj procesa u bilo koje vrijeme, a maksimalni broj paralelnih procesa podešen je parametrom `job_queue_processes`.
- **User Session Processes:** Ručnim kreiranjem procesa operativnog sustava koji imaju konekciju prema bazi moguće je stvoriti vlastito rješenje upravljanja paralelnim procesima.

2.2 PL/SQL tablične funkcije

Tablične funkcije su PL/SQL funkcije koje kao rezultat vraćaju skupove redova koji igraju ulogu tablice. Od verzije baze Oracle9i tablične funkcije mogu primati cursor, tj. rezultat SQL upita kao ulazni parametar. Nad njima se mogu izvoditi SQL upiti kao i nad tablicama baze uporabom TABLE funkcije u FROM klauzuli SQL upita.

Slika 2 ilustrira operaciju agregacije podataka pomoću tablične funkcije kojoj ulaz predstavlja skup redova, a izlaz je novi skup redova nastalih nakon izvođenja SUM operacije.



Slika 2: Shematski prikaz rada tablične funkcije [5]

Pseudokod za izvođenje ovakve operacije izgledao bi otprilike ovako:

```
INSERT INTO Out SELECT * FROM TABLE(Table_Function(SELECT * FROM In));
```

Obične tablične funkcije zahtijevaju da se čitava kolekcija izlaznih podataka popuni prije izlaza iz funkcije. Takav način rada može zauzimati značajne memorijske resurse (Program Global Area - PGA) i zahtijevati mnogo vremena prije produciranja prvih rezultata. Rezultat obične tablične funkcije mora biti SQL tip na bazi. U nastavku je primjer kôda kojim se kreira izrazito jednostavna tablična funkcija [11]:

```
-- Create the types to support the table function.
DROP TYPE t_tf_tab;
DROP TYPE t_tf_row;

CREATE TYPE t_tf_row AS OBJECT (
  id           NUMBER,
  description  VARCHAR2(50)
);
/

CREATE TYPE t_tf_tab IS TABLE OF t_tf_row;
/

-- Build the table function itself.
CREATE OR REPLACE FUNCTION get_tab_tf (p_rows IN NUMBER) RETURN  t_tf_tab AS
  l_tab  t_tf_tab := t_tf_tab();
BEGIN
FOR i IN 1 .. p_rows LOOP
    l_tab.extend;
    l_tab(l_tab.last) := t_tf_row(i, 'Description for ' || i);
  END LOOP;

  RETURN l_tab;
END;
/
```

Poziv funkcije izgleda ovako:

```
SELECT *
FROM   TABLE(get_tab_tf(10))
ORDER BY id DESC;

        ID DESCRIPTION
---------- --------------------------------------------------
        10 Description for 10
         9 Description for 9
         8 Description for 8
         7 Description for 7
         6 Description for 6
         5 Description for 5
         4 Description for 4
         3 Description for 3
         2 Description for 2
         1 Description for 1

10 rows selected.
```

2.3 "Pipelined" tablične funkcije

"Pipelined" tablične funkcije svojevrsni su spoj mogućnosti PL/SQL-a s performansama SQL-a. Nedostatak "običnih" tabličnih funkcija je u zauzimanju značajnih memorijskih resursa potrebnih za pohranjivanje rezultata, kao i u vremenu potrebnom za vraćanje prvog retka rezultata. Na sreću postoje tzv. pipelined tablične funkcije koje vraćaju rezultate u obliku red po red. Razlikuju se od običnih tabličnih funkcija u klauzuli PIPELINED i pozivanjem PIPE ROW za vraćanje pojedinog reda iz

funkcije.

Obične tablične funkcije kao rezultat koriste SQL tipove definirane na bazi, dok "pipelined" funkcije mogu koristiti i PL/SQL tipove koje tada Oracle implicitno kreira kao tipove na bazi.

Tablične funkcije ne mogu se koristiti preko DB linka. Razlog je taj što je izlazni tip podataka SQL tip na bazi (eksplicitni ili implicitni), koji se može koristiti samo na bazi na kojoj je definiran. U nastavku je primjer pipelined funkcije koja radi isti posao kao i obična tablična funkcija iz prethodnog poglavlja, umjesto da puni kolekciju ova funkcija odmah vraća rezultat (PIPE ROW) i završava praznom RETURN naredbom:

```
CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN  t_tf_tab
PIPELINED AS
BEGIN
  FOR i IN 1 .. p_rows LOOP
    PIPE ROW(t_tf_row(i, 'Description for ' || i));
  END LOOP;

  RETURN;
END;
/
```

2.4 Paralelno izvršenje pomoću "pipelined" tabličnih funkcija

Kako bi pipelined tablična funkcija koristila mogućnosti paralelnog izvršenja potrebno je da zadovolji sljedeće uvjete:

- Mora sadržavati klauzulu PARALLEL_ENABLE
- Mora imati barem jedan REF CURSOR ulazni parametar
- Mora sadržavati PARTITION BY klauzulu kojom se definira metoda raspoređivanja redova po paralelnim instancama. Ref kursori slabog tipa mogu koristiti samo PARTITION BY ANY klauzulu kojom se slučajnim odabirom redovi raspoređuju po paralelnim instancama.
- Cursor parametar mora se izvršavati paralelno, tj. imati definiran DOP (degree of parallelism).

Opcionalno se dodatno može koristiti i klauzula CLUSTER ili ORDER za grupiranje i sortiranje podataka unutar paralelnog procesa.

Osnovna sintaksa izgleda ovako:

```
CREATE FUNCTION function-name(parameter-name ref-cursor-type)
  RETURN rec_tab_type PIPELINED
  PARALLEL_ENABLE(PARTITION parameter-name BY [{HASH | RANGE} (column-list) |
ANY ])
  [ORDER | CLUSTER] parameter-name BY (column-list) IS
BEGIN
  ...
END;
```

U nastavku je i primjer zaglavlja paketa s funkcijama koje podržavaju paralelno izvršavanje. Čitavi paket može se potražiti u izvoru [11].

```
1  CREATE OR REPLACE PACKAGE parallel_ptf_api AS
2
3           TYPE t_parallel_test_row IS RECORD (
4             id              NUMBER(10),
5             country_code    VARCHAR2(5),
6             description     VARCHAR2(50),
7             sid             NUMBER
8           );
9
10        TYPE t_parallel_test_tab IS TABLE OF t_parallel_test_row;
11
12         TYPE t_parallel_test_ref_cursor IS REF CURSOR
13           RETURN parallel_test%ROWTYPE;
14
15          FUNCTION test_ptf_any (p_cursor  IN  t_parallel_test_ref_cursor)
16            RETURN t_parallel_test_tab PIPELINED
```

```
17            PARALLEL_ENABLE(PARTITION p_cursor BY ANY);
18
19          FUNCTION test_ptf_hash (p_cursor  IN  t_parallel_test_ref_cursor)
20            RETURN t_parallel_test_tab PIPELINED
21            PARALLEL_ENABLE(PARTITION p_cursor BY HASH (country_code))
22            ORDER p_cursor BY (country_code, created_date);
23
24          FUNCTION test_ptf_range (p_cursor  IN  t_parallel_test_ref_cursor)
25            RETURN t_parallel_test_tab PIPELINED
26            PARALLEL_ENABLE(PARTITION p_cursor BY RANGE (country_code))
27            CLUSTER p_cursor BY (country_code, created_date);
28
29        END parallel_ptf_api;
30        /
```

PARTITION BY klauzula definira metodu koja će se koristiti za dijeljenje redova po paralelnim instancama funkcije, dok ORDER/CLUSTER klauzula određuje na koji način su redovi raspoređeni unutar instance. U nastavku će se detaljnije pojasniti ovi pojmovi:

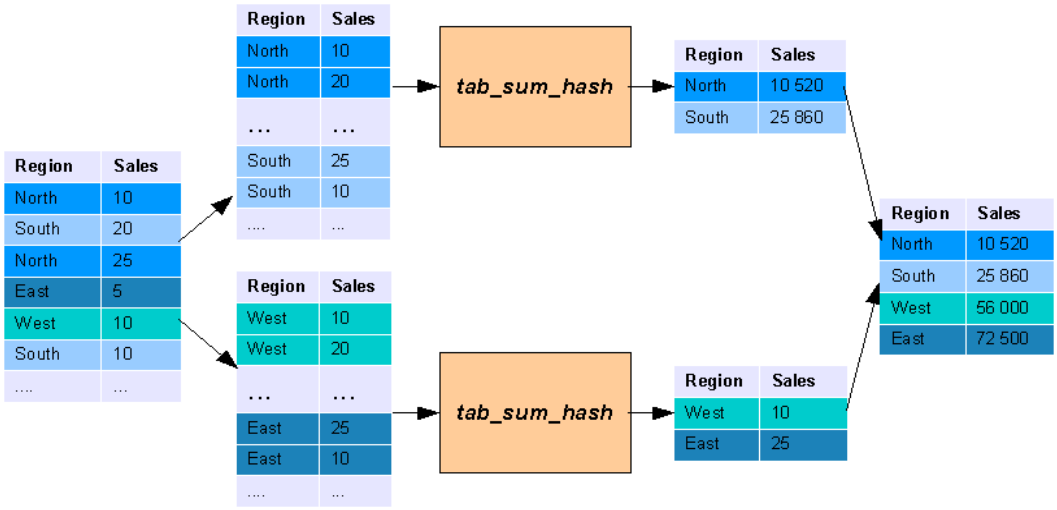
- ORDER, CLUSTER: Ovim opcijama definira se na koji način su redovi dostavljeni u svaku od paralelnih instanci nakon što su prethodno podijeljeni. Pretpostavimo li da je podskup podataka koji su raspoređenu u neku instancu sljedeći:
  5,4,5,6,8,3,2,3,4,5,6,1,2,4,6.
  CLUSTER opcijom podaci će funkciji možda biti predani ovakvim redoslijedom:
  6,6,6,3,3,8,1,4,4,4,5,5,5,2,2,
  dok će ORDER opcijom podaci biti sortirani u obliku:
  1,2,2,3,3,4,4,4,5,5,5,6,6,6,8.
  Oba postupka zahtijevaju dodatni posao za proces instance, a sortiranje je nešto zahtjevnije od grupiranja.
- PARTITION BY ANY: Ovom metodom redovi se slučajnim odabirom dijele po paralelnim instancama i svaka od instanci dobiva otprilike jednaki broj redova. Dodatno grupiranje ili sortiranje tako dobivenih podataka ima ograničeni smisao budući da se ne može pretpostaviti na koji način će redovi biti podijeljeni po instancama.
- PARTITION BY HASH: Podaci se grupiraju u podskupove prema zadanim kolonama. Čitav podskup takvih podataka biti će dodijeljen jednoj instanci. Jedna instanca može zaprimiti i nekoliko različitih podskupova, ali može se računati na činjenicu da je čitav podskup podataka u toj instanci. U osnovi, metoda se može usporediti s CLUSTER opcijom u smislu podjele podataka. Ova metoda omogućava uravnoteženu podjelu podataka po instancama, a istodobno i sigurnost da su podaci s određenom vrijednošću na okupu. Uz tu pretpostavku daljnje grupiranje ili sortiranje podataka unutar instance može imati smisla.
- PARTITION BY RANGE: Ovom metodom podaci se sortiraju prema zadanim vrijednostima prije podjele po instancama, nešto kao ORDERED opcija, ali za podjelu podataka. Osigurava da su podaci u potpunosti grupirani unutar pojedine instance prema rastućem redosljedu. Manjkavost ove metode je u tome što distribucija podataka po instancama može biti neravnomjerna. Kao i kod PARTITION BY HASH metode, grupiranje ili sortiranje podataka unutar instance ima smisla.

U nastavku je skicirani prikaz parelelnog izvršenja tablične funkcije "tab_sum_hash". Funkcija kao ulazni parametar prima detaljne redove prodaje, a vraća zbirne podatke po regijama. Definicija funkcije je sljedeća:

```
FUNCTION tab_sum_hash (p_cursor  IN t_ref_cursor)
      RETURN t_tab PIPELINED
      PARALLEL_ENABLE(PARTITION p_cursor BY HASH (region))
      CLUSTER p_cursor BY (region);
```

Pseudokôd poziva sa skice izgleda ovako:

```
SELECT * FROM TABLE(tab_sum_hash (SELECT /*+parallel (s,10)*/ s.*
                                    FROM sales s));
```



Slika 3: Shematski prikaz paralelnog izvršenja tablične funkcije

2.5 Izvođenje DML operacija u tabličnim funkcijama

Da bi tablična funkcija mogla izvoditi DML operacije u deklaraciji treba biti postavljena kao autonomna transakcija. Tada se funkcija izvršava u zasebnoj sesiji koju ne dijeli s glavnim procesom.

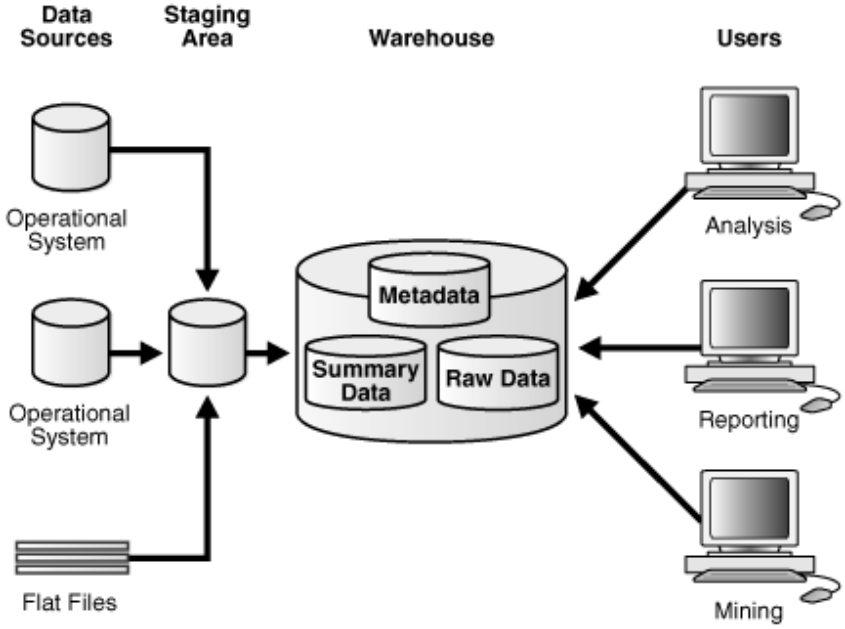
Ovo je sintaksa jedne takve funkcije:

```
CREATE FUNCTION f(p SYS_REFCURSOR) return CollType PIPELINED IS
       PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN ... END;
```

U paralelnom radu, svaka od instanci funkcije tada kreira nezavisnu transakciju.

2.6 Primjena u ETL procesu

Standardni ETL (Extract Transfom Load) postupak uobičajeno se rješava na način da se podaci iz vanjskih sustava najprije prepišu u radno područje (Staging Area), a zatim prolaze niz koraka koji ih transformiraju u oblik u kojem se upisuju u skladište. Zapisivanjem podataka u radne tablice stvara se značajan diskovni promet. Pipelined tablične funkcije pružaju alternativu kojom se npr. podaci iz eksterne tablice direktno prepisuju u odredišnu tablicu eliminacijom većine diskovnog prometa.



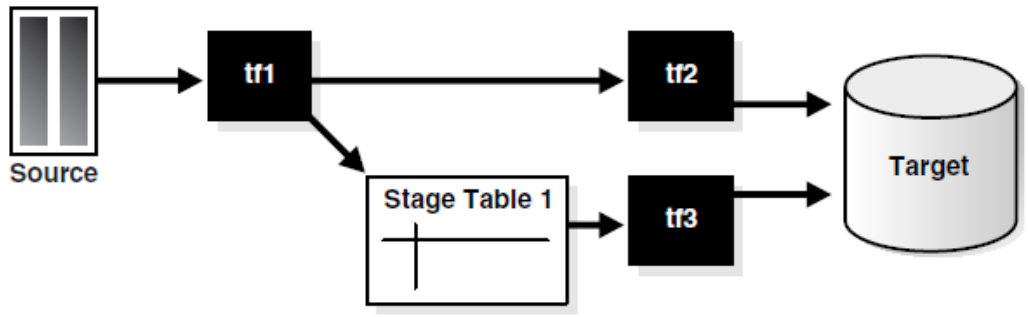
Slika 4: Skica arhitekture skladišta podataka [5]

Uz poznavanje činjenica da tablične funkcije:

- vraćaju redove kao rezultate,
- rezultati SQL upita ili neke druge tablične funkcije mogu se prosljeđivati funkciji kao CURSOR parametar,
- skupovi izlaznih podataka vraćaju se postepeno u trenutku kada se kreiraju,
- izvođenje se može paralelizirati

jednostavno se može predočiti scenarij njihove primjene u ETL procesu.

Proces transformacije podataka obično se sastoji od niza podijeljenih operacija, komunikacija se obično održava podacima u radnim tablicama. Relativno je jednostavno i elegantno neki od koraka zamijeniti tabličnom funkcijom. Tablična funkcija može npr. čitati podatke direktno iz vanjskih tablica, transformirati ih i predati odredišnim tablicama skladišta, radnoj tablici ili drugoj tabličnoj funkciji na daljnju transformaciju.



Slika 5: Skica ETL procesa koji koristi tablične funkcije

Pseudo kod za transformaciju sa slike 5 sastojao bi se iz ova dva koraka:

```
INSERT INTO target
   SELECT * FROM TABLE(tf2 (SELECT * FROM TABLE(tf1 (SELECT * FROM source))));

INSERT INTO target SELECT * FROM TABLE(tf3(SELECT * FROM stage_table1));
```

Brojne su kombinacije koje se mogu složiti, a najveća prednost je u tome što se proces može paralelizirati bez posebnog kodiranja. Dovoljno je u tabličnoj funkciji definirati klauzule paralelnog rada i proslijediti joj kursor koji se izvršava paralelno.

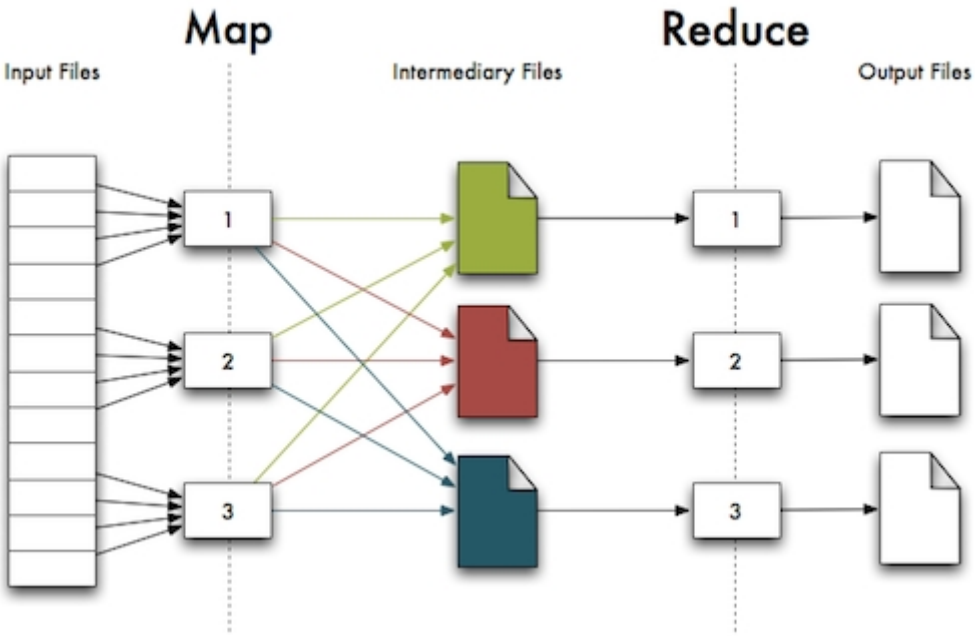
3 MAPREDUCE U ORACLE BAZI

3.1 MapReduce model

MapReduce programski model popularizirao je Google koji je ujedno i vlasnik patenta. Mnogi (i Oracle) žale se na dodjelu tog patenta jer smatraju da je Google samo popularizirao ideju koji i drugi već odavno imaju u primjeni. Osnovna ideja modela je dijeljenje ulaznih podataka u manje skupine koje se prosljeđuju na paralelna izvršavanje, međurezultati se grupiraju i takvi ponovno paralelno prosljeđuju funkciji koja producira izlazne podatke. MapReduce proces primjenjiv je za struktirirane i nestrukturirane sadržaje. Tijek podataka MapReduce algoritmom karakterizira pet osnovnih faza:

1. Input – Ulazni podaci dijele se na manje nezavisne skupove podataka koji se paralelno prosljeđuju *map* funkciji.
2. Map – Tipična map funkcija preuzima podskup ulaznih podataka, obrađuje ih, i kao rezultat vraća podskupove uređenih parova ključ/vrijednost (key/value).
3. Partition/Sort – Međurezultati se grupiraju, a mogu se i sortirati, te se u takvim manjim podskupovima paralelno prosljeđuju *reduce* funkciji.
4. Reduce – Reduce funkcija izvodi operacije agregacije nad podskupom ulaznih podataka i producira izlazne podatke.

5. Output – Izlazni podaci se prikupljaju i producira rezultat modela.

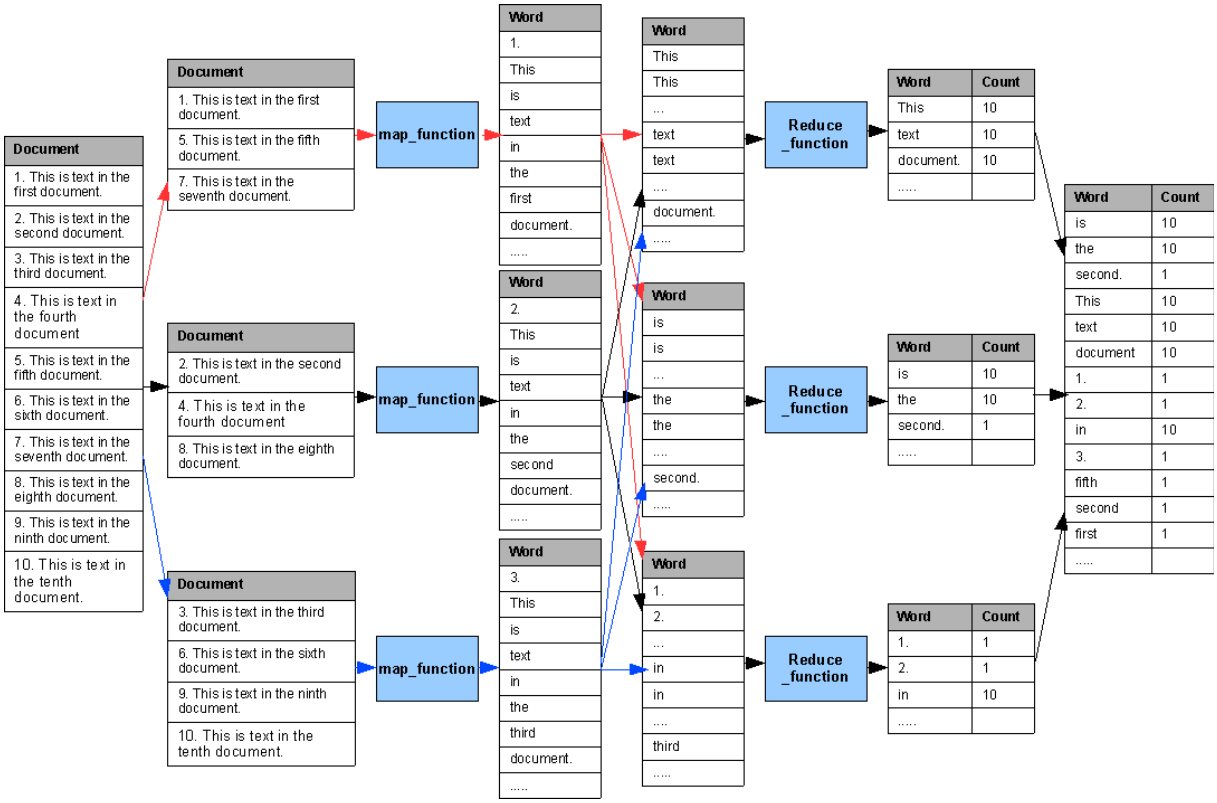


Slika 6: Shematski prikaz MapReduce modela [12]

Veoma jednostavno MapReduce model može se preslikati na Oracle bazu kao lančani poziv tabličnih funkcija u paralelnom načinu rada, npr.

```
SELECT *
  FROM TABLE(reduce_function
         (CURSOR(SELECT *
                   FROM TABLE(map_function (CURSOR(SELECT /*+parallel (i,10)*/
                                                     i.*
                                                FROM input_file i))))));
```

Za slikovitiju predodžbu prikazati će se primjer iz izvora [6] gdje je prikazan MapReduce model prebrojavanja riječi u dokumentima. U dokumentu se može pronaći i source kôd rješenja. Map funkcija prima listu dokumenata kao ulazni parametar, dijeli ih na riječi i vraća popis riječi iz dokumenta. Reduce funkcija prima popis riječi (u ORDER ili CLUSTER redoslijedu), prebrojava ih, te vraća rezultat. Cijeli posao odvija se u paralelnom načinu rada. Što se događa pozivom prikazuje sljedeća skica:



Slika 7: Skica paralelnog rada tabličnih funkcija u MapReduce modelu

Činjenica da se MapReduce model može implementirati pomoću PL/SQL-a na bazi ne znači da ga treba uvijek primjenjivati. Ponekad se do rezultata elegantnije i brže dolazi jednostavnom upotrebom SQL naredbi. U izvoru [2], Expert PL/SQL Practices, prikazan je MapReduce primjer brojanja pojavljivanja znakova u nazivima kolona. Prikazani primjer npr. može se vrlo jednostavno riješiti upotrebom SQL-a:

```
select letter key_item, count(*) value_item
  from (select substr(d.text, c.i, 1) letter
          from documents d,
               (select level i from dual connect by level <= 30) c)
 where letter is not null
 group by letter
 order by letter;
```

Prava vrijednost primjera je u razumijevanju paradigme paralelnog izvršenja PL/SQL-a na Oracle bazi i upoznavanje s mogućnostima primjene u specifičnim situacijama.

ZAKLJUČAK

U radu su prikazane mogućnosti programiranja paralelnog rada na Oracle bazi. Prikazan je jednostavni primjer primjene MapReduce modela pomoću PL/SQL tabličnih funkcija.

Za odluku o primjeni paralelnih mogućnosti izvršenja na Oracle bazi potrebno je dobro odvagnuti prednosti. Paralelni procesi, u pravilu troše više sistemskih resursa u odnosu na serijske. U situacijama kada postoje raspoloživi kapaciteti paralelnim izvođenjem može se ubrzati obrada velikih skupova podataka, tj. njihova obrada smjestiti u raspoložive vremenske okvire.

PL/SQL tablične funkcije nisu najbolje rješenje za sve probleme. Dobro ih je primijeniti u situacijama kada je potrebno implementirati kompleksne transformacije koje je teško prikazati upotrebom SQL-a. U radu nije dan osvrt na daljnje mogućnosti optimizacije rada s tabličnim funkcijama, a ima ih dosta. Zanimljiva je mogućnost korištenja tzv. "extensible optimizer"-a. To je interface za procjenu kardinalnosti kojim se funkcija povezuje sa specijalnim objektnim tipom koji CBO -u daje informaciju o kardinalnosti rezultata. Postupak je lijepo opisan u 21. poglavlju knjige Oracle

PL/SQL Programming [3].

LITERATURA

1. Oracle® Database PL/SQL User's Guide and Reference 10*g* Release 2 (10.2) B14261-01

2. John Beresniewicz, Adrian Billington, Martin Büchi, Melanie Caffrey, Ron Crisco, Lewis Cunningham, Dominic Delmolino, Sue Harper, Torben Holm, Connor McDonald, Arup Nanda, Stephan Petit, Michael Rosenblum, Robyn Sands, Riyaj Shamsudeen (2011): Expert PL/SQL Practices: for Oracle Developers and DBAs, Apress

3. Steven Feuerstein (2009): Oracle PL/SQL Programming, FIFTH EDITION, O'Reilly Media

4. Thomas Kyte: Expert Oracle Database Architecture Oracle Database 9*i*, 10*g*, and 11*g* Programming Techniques and Solutions Second Edition, Apress

5. Oracle® Database Data Warehousing Guide 10*g* Release 2 (10.2) B14223-02

6. Shrikanth Shankar, Jean-Pierre Dijcks: In-Database Map-Reduce, An Oracle White Paper, October 2009

7. Oracle® Database PL/SQL Language Reference 11*g* Release 2 (11.2) E25519-05

LINKOVI

8. https://blogs.oracle.com/datawarehousing/entry/mapreduce_oracle_tablefunction

9. http://www.oracle-developer.net/display.php?id=427

10. http://www.oracle-developer.net/display.php?id=429

11. http://www.oracle-base.com/articles/misc/pipelined-table-functions.php

12. http://blog.maxgarfinkel.com/archives/176

13. http://en.wikipedia.org/wiki/MapReduce

14. http://www.akadia.com/services/ora_parallel_processing.html