7. PARTIȚIONĂRI

Cuprins

7. Partiționări

- Partiționarea datelor permite divizarea tabelelor și a indecșilor de dimensiuni mari în submulțimi mai mici și mai ușor de administrat.
  - În acest capitol sunt exemplificate metodele de partiționare pentru tabele.
  - Partiționarea indecșilor se realizează în mod asemănător tabelelor.

❖ Fiecare submulțime a unei partiții poate fi administrată individual și poate funcționa independent față de celelalte submulțimi.

❖ Partiționarea este transparentă aplicațiilor, acestea folosind tabelele partiționate ca și când ar fi tabele normale, dar având performanțe mai bune.

❖ Orice tabelă poate fi partiționată, cu excepția celor care au coloane de tip *LONG* sau *LONG RAW*.

- Tipurile de date *LONG* și *LONG RAW* nu sunt recomandate pentru aplicațiile de generație nouă. Acestea sunt păstrate din motive de compatibilitate cu versiunile anterioare. În locul acestora se pot utiliza tipurile de date *CLOB* sau *BLOB*.

- Partiționarea determină creșterea vitezei de accesare a datelor și îmbunătățește performanța aplicației.
  - Aceasta se întâmplă în mod deosebit în cazul aplicațiilor care accesează tabele cu milioane de înregistrări.
  - Comenzile *SELECT* și *LMD* pot rula în paralel pe submulțimi.
- Partiționarea îmbunătățește în mod considerabil prelucrarea datelor stocate în baze de date de dimensiuni foarte mari și reduce timpul necesar pentru operații administrative, precum *backup* și *restore*.
- Recomandări privind alegerea tabelelor candidat pentru partiționare:
  - tabele care au dimensiunea mai mare de 2 GB;
  - tabele care conțin date istorice, în care datele noi ar fi adăugate în submulțimea cea mai recentă (de exemplu, sunt actualizate numai datele submulțimii corespunzătoare lunii curente, iar datele corespunzătoare celorlalte luni sunt *read-only*);
  - tabele al căror conținut trebuie divizat și distribuit pe medii de stocare diferite.

- Folosirea metodelor de partiționare descrise în această secțiune, determină optimizarea instrucțiunilor *SQL*.
  - Se evită operațiile care nu sunt necesare pe întreaga tabelă, utilizându-se doar acele submulțimi care conțin datele utile comenzii.
  - Se poate îmbunătăți performanța operațiilor masive de *join*, atunci când cantități mari de date sunt unite folosind *join*-uri pe submulțimi în loc de tabele globale.

7.1. Metode de partiționare

- Sistemul *Oracle* permite mai multe metode de partiționare:
  - partiționarea *range* (prin ordonare);
  - partiționarea *hash*;
  - partiționarea *list* (prin listă);
  - partiționarea compusă.
- Fiecare metodă de partiționare are diferite avantaje și modele de proiectare.
  - Astfel, o anumită metodă de partiționare este caracteristică unei situații particulare.

Cheia de partiționare

- Mapează înregistrările tabelei într-o singură submulțime.
- Poate fi compusă dintr-una sau mai multe coloane ale tabelei (maxim 16).
- Este folosită de *server*-ul *Oracle* pentru a detecta automat ce submulțimi vor fi utilizate de comenzile *LMD* sau *SELECT*.

❖ Pentru o tabelă se poate defini o singură cheie de partiționare.

Specificarea tipului de partiționare

- Tipul de partiționare se specifică folosind clauza *PARTITION BY*.

```
PARTITION BY RANGE (listă_coloane)
PARTITION BY HASH  (listă_coloane)
PARTITION BY LIST  (coloană)
```

7.1.1. Partiționarea prin ordonare

- Este cel mai folosit tip de partiționare.
- Partiționarea prin ordonare, mapează înregistrările în submulțimi în funcție de valorile coloanelor după care se realizează partiționarea.
  - De exemplu, datele despre vânzări pot fi partiționate în submulțimi lunare.
- Partiționarea prin ordonare se definește precizând:
  - tipul de partiționare al tabelei

    ```
    PARTITION BY RANGE (listă_coloane)
    ```

    - *Listă_coloane* reprezintă o listă ordonată de coloane pe baza cărora se va realiza partiționarea.
    - Aceste coloane se numesc coloane de partiționare.
    - Valorile coloanele de partiționare ale unui anumite înregistrări constituie cheia de partiționare a acesteia.
  - specificațiile pentru fiecare submulțime individuală a partiției

    ```
    VALUES LESS THAN (listă_valori)
    ```

    - *Listă_valori* reprezintă o listă ordonată de valori pentru coloanele care apar în *listă_coloane*.
    - Este admisă numai clauza *VALUES LESS THAN*. Aceasta specifică limita superioară a submulțimii (care nu este inclusă). Toate submulțimile, cu excepția primei submulțimi, au o valoare minimă implicită, specificată de clauza *VALUES LESS THAN valoare_limită_anterioară* a submulțimii anterioare. Orice valori ale cheii de partiționare egale sau mai mari decât *valoare_limită_anterioară* și mai mici decât *valoare_limită_curentă* (specificată pentru submulțimea curentă prin clauza *VALUES LESS THAN valoare_limită_curentă*) sunt incluse în următoarea submulțime.
    - Cea mai mare submulțime este cea pentru care apare literalul *MAXVALUE*. Acesta reprezintă o valoare virtuală infinită care este mai mare decât orice altă valoare pentru un anumit tip de date, incluzând valoarea *null*.

Exemplul 7.1

Următoarea instrucțiune creează tabela *vanzari_ord* care este partiționată prin ordonare, cheia de partiționare fiind coloana *timp_id*.

a. Coloana *timp_id* este de tip *DATE*, reprezentând data calendaristică la care este emisă factura.

Observații:

- Submulțimea *vanzari_jan2007* conține toate înregistrările pentru care *timp_id* < '01/02/2007'. Deoarece facturile înregistrate sunt emise începând cu 01/01/2007, această submulțime conține toate înregistrările corespunzătoare facturilor emise în luna ianuarie din anul 2007.
- Submulțimea *vanzari_feb2007* conține toate înregistrările pentru care *timp_id* >= '01/02/2007' și *timp_id* < '01/03/2007', adică toate înregistrările corespunzătoare facturilor emise în luna februarie din anul 2007 (*timp_id* ∈ ['01/02/2007', '01/03/2007']).
- Submulțimea *vanzari_rest* conține toate înregistrările pentru care *timp_id* >= '01/05/2007'.

```
CREATE TABLE vanzari_ord
(id                  NUMBER,
 produs_id           NUMBER(4),
 client_id           NUMBER(4),
 timp_id             DATE,
 factura             NUMBER(4),
 cantitate           NUMBER(4),
 pret_unitar_vanzare NUMBER(10,2))
PARTITION BY RANGE(timp_id)
(
   PARTITION vanzari_jan2007
     VALUES LESS THAN(TO_DATE('01/02/2007','DD/MM/YYYY')),
   PARTITION vanzari_feb2007
     VALUES LESS THAN(TO_DATE('01/03/2007','DD/MM/YYYY')),
   PARTITION vanzari_mar2007
     VALUES LESS THAN(TO_DATE('01/04/2007','DD/MM/YYYY')),
   PARTITION vanzari_apr2007
     VALUES LESS THAN(TO_DATE('01/05/2007','DD/MM/YYYY')),
   PARTITION vanzari_rest
     VALUES LESS THAN (MAXVALUE)
);
```

b. Coloana *timp_id* este de tip *NUMBER*, reprezentând *Julian Day* corespunzătoare datei calendaristice la care este emisă factura (data calendaristică exprimată ca număr de zile scurse începând de la 1 ianuarie 4712 B.C.).

```
CREATE TABLE vanzari_ord
(id                  NUMBER,
 produs_id           NUMBER(4),
 client_id           NUMBER(4),
 timp_id             NUMBER,
 factura             NUMBER(4),
 cantitate           NUMBER(4),
 pret_unitar_vanzare NUMBER(10,2))
PARTITION BY RANGE(timp_id)
(
   PARTITION vanzari_jan2007 VALUES LESS THAN (2454133),
     -- 2454133 =
     --      TO_CHAR(TO_DATE('01/02/2007','DD/MM/YYYY'),'J')
   PARTITION vanzari_feb2007 VALUES LESS THAN (2454161),
     -- 2454161 =
     --      TO_CHAR(TO_DATE('01/03/2007','DD/MM/YYYY'),'J')
   PARTITION vanzari_mar2007 VALUES LESS THAN (2454192),
     -- 2454192 =
     --      TO_CHAR(TO_DATE('01/04/2007','DD/MM/YYYY'),'J')
   PARTITION vanzari_apr2007 VALUES LESS THAN (2454222),
     -- 2454222 =
     --      TO_CHAR(TO_DATE('01/05/2007','DD/MM/YYYY'),'J')
   PARTITION vanzari_rest
     VALUES LESS THAN (MAXVALUE)
);
```

Exemplul 7.2

Exemplu de cereri care utilizează doar o submulțime a partiției.

a. Utilizarea corespunzătoarea a predicatului cererii astfel încât optimizatorul să scaneze doar submulțimea ce conține informațiile necesare cererii.

```
SELECT *
FROM   vanzari_ord
WHERE  timp_id >= TO_DATE('01/03/2007','DD/MM/YYYY')
AND    timp_id <  TO_DATE('01/04/2007','DD/MM/YYYY');
```

b. Specificarea în cerere a submulțimii necesare cererii.

```
SELECT *
FROM   vanzari_ord PARTITION (vanzari_mar2007);
```

7.1.2. Partiționarea *hash*

- Acest tip de partiționare determină maparea datelor în submulțimi utilizând un algoritm *hash*, pe care sistemul îl aplică pentru o cheie de partiționare specificată. Algoritmul *hash* distribuie în mod egal înregistrările, rezultând submulțimi de aproximativ aceeași dimensiune.
- Partiționarea *hash* reprezintă o metodă ideală pentru distribuirea datelor în mod egal. De asemenea, acest tip de partiționare reprezintă o alternativă ușor de utilizat pentru partiționarea prin ordonare, atunci când datele nu sunt istorice și nu există nicio coloană evidentă, pentru care optimizarea prin eliminarea scanării submulțimilor logice (obținute prin partiționarea prin ordonare) să fie avantajoasă.
- Sistemul *Oracle* folosește algoritmi *hash* liniari și nu pot fi definiți alți algoritmi *hash* pentru partiționare. Se recomandă ca numărul de submulțimi ale partiției să fie o putere a lui 2.

Exemplul 7. 3

Următoarea instrucțiune creează tabela *vanzari_hash* care este partiționată *hash*, cheia de partiționare fiind coloana *produs_id*.

```
CREATE TABLE vanzari_hash
   (id                   NUMBER,
    produs_id            NUMBER(4),
    client_id            NUMBER(4),
    timp_id              DATE,
    factura              NUMBER(4),
    cantitate            NUMBER(4),
    pret_unitar_vanzare NUMBER(10,2))
 PARTITION BY HASH(produs_id)
 PARTITIONS 4;
```

Exemplul 7.4

Exemplu de cereri care utilizează doar o submulțime a partiției.

a. Utilizarea corespunzătoarea a predicatului cererii astfel încât optimizatorul să scaneze doar submulțimea ce conține informațiile necesare cererii.

```
SELECT *
FROM   vanzari_hash
WHERE  produs_id = 3010;
```

b. Specificarea în cerere a submulțimii necesare cererii.

```
SELECT *
FROM   vanzari_hash PARTITION (SYS_P381);
```

Observații:

- *SYS_P381* reprezintă un numele atribuit de sistem submulțimii.
- Vizualizarea *user_tab_partitions* din dicționarul datelor furnizează informații despre submulțimile partiției.
- Pentru a afla numele submulțimii *hash* care se dorește a fi scanată se poate utiliza următoarea cerere:

```
SELECT PARTITION_NAME
FROM   USER_TAB_PARTITIONS
WHERE  TABLE_NAME = UPPER('VANZARI_HASH')
AND    PARTITION_POSITION = ORA_HASH('3010', 4 - 1) + 1;
```

unde, *ORA_HASH*('*valoare_căutată*', *număr_submulțimi_partiție* - 1) este funcția *hash* care este utilizată de sistem pentru partiționarea *hash* (depinzând de versiunea server-ului de baze de date utilizat). În acest exemplu se determină numele submulțimii în care se găsește produsul cu codul 3010, tabela *vanzari_hash* fiind partiționată *hash* în 4 submulțimi.

7.1.3. Partiționarea prin listă

- Partiționarea prin listă oferă posibilitatea de a controla în mod explicit cum să se mapeze înregistrările în submulțimi. Pentru aceasta se specifică o listă de valori, pentru cheia de partiționare, la definirea fiecărei submulțimi a partiției.
- Această metodă este diferită față de partiționarea prin ordonare, pentru care o clasă ordonată de valori este asociată cu o submulțime și de partiționarea *hash*, caz în care nu există control asupra distribuției înregistrărilor în submulțimi.
- Utilizând partiționarea prin listă se pot grupa și organiza mulțimi de date neordonate și nerelaționate într-un mod foarte natural.

Exemplul 7.5

Următoarea instrucțiune creează tabela *vanzari_lista* care este partiționată prin listă, cheia de partiționare fiind coloana *oras_id*.

```
CREATE TABLE vanzari_lista
   (id                  NUMBER,
    produs_id           NUMBER(4),
```

```
  client_id           NUMBER(4),
  timp_id             DATE,
  oras_id          NUMBER(4),
  factura             NUMBER(4),
  cantitate           NUMBER(4),
  pret_unitar_vanzare NUMBER(10,2))
PARTITION BY LIST(oras_id)(
PARTITION vanzari_moldova
    VALUES(70, 255),
    -- 70 = Iași, 255 = Bacău
PARTITION vanzari_muntenia
    VALUES(247, 18),
    -- 247 = Bucuresti, 18 = Pitesti
PARTITION vanzari_oltenia
    VALUES(719, 503 , 277),
    -- 719 = Craiova, 503 = Targu Jiu, 277 = Ramnicu Valcea
PARTITION vanzari_altele VALUES(DEFAULT));
```

Observație:

- În cazul partiționării prin listă se poate utiliza o submulțime predefinită, astfel încât o înregistrare care nu se încadrează în niciuna dintre submulțimile definite să nu genereze o eroare.

Exemplul 7.6

Exemplu de cereri care utilizează doar o submulțime a partiției.

a. Utilizarea corespunzătoarea a predicatului cererii astfel încât optimizatorul să scaneze doar submulțimea ce conține informațiile necesare cererii.

```
SELECT *
FROM   vanzari_lista
WHERE  oras_id IN (247, 18);
```

b. Specificarea în cerere a submulțimii necesare cererii.

```
SELECT *
FROM   vanzari_lista PARTITION (vanzari_muntenia);
```

7.1.4. Partiționarea compusă

- Partiționarea compusă combină două tipuri de partiționare. Sistemul distribuie mai întâi datele în submulțimi corespunzătoare primul tip de partiționare, iar apoi fiecare submulțime este distribuită la rândul său în submulțimi, conform celui de al doilea tip de partiționare specificat.

- Sunt permise următoarele tipuri de partiționări compuse:
  - partiționare *range – range*;
  - partiționare *range – hash*;
  - partiționare *range – list*;
  - partiționare *list – range*;
  - partiționare *list – hash*;
  - partiționare *list – list.*

Exemplul 7.7

Instrucțiunea următoare creează tabela *vanzari_ord_hash*, care este partiționată prin ordonare, utilizând cheia de partiționare *timp_id*, iar fiecare submulțime obținută este divizată în subpartiții *hash*, utilizând cheia de partiționare *produs_id.*

```
CREATE TABLE vanzari_ord_hash
   (id                  NUMBER,
    produs_id           NUMBER(4),
    client_id           NUMBER(4),
    timp_id             DATE,
    factura             NUMBER(4),
    cantitate           NUMBER(4),
    pret_unitar_vanzare NUMBER(10,2))
PARTITION BY RANGE (timp_id)
SUBPARTITION BY HASH (produs_id) SUBPARTITIONS 4
(PARTITION vanz2007t1
  VALUES LESS THAN (TO_DATE('01/04/2007','DD/MM/YYYY')),
 PARTITION vanz2007t2
  VALUES LESS THAN (TO_DATE('01/07/2007','DD/MM/YYYY')),
 PARTITION vanz2007t3
  VALUES LESS THAN (TO_DATE('01/10/2007','DD/MM/YYYY')),
 PARTITION vanz2007t4
  VALUES LESS THAN (TO_DATE('01/01/2008','DD/MM/YYYY')));
```

Exemplul 7.8

Exemplu de cereri care utilizează doar o submulțime a partiției.

a. Utilizarea corespunzătoarea a predicatului cererii astfel încât optimizatorul să scaneze doar submulțimea ce conține informațiile necesare cererii.

```
SELECT *
FROM   vanzari_ord_hash
WHERE  timp_id >= TO_DATE('01/03/2007','DD/MM/YYYY')
AND    timp_id <  TO_DATE('01/04/2007','DD/MM/YYYY')
AND    produs_id = 3010;
```

b. Specificarea în cerere a submulțimii necesare cererii.

```
SELECT *
FROM   vanzari_ord_hash
SUBPARTITION (SYS_SUBP385);
```

Observații:

- *SYS_SUBP385* reprezintă un numele atribuit de sistem submulțimii.
- Vizualizarea *user_tab_subpartitions* din dicționarul datelor furnizează informații despre submulțimile partiției.
- Pentru a afla numele submulțimii *hash* care se dorește a fi scanată se poate utiliza următoarea cerere:

```
SELECT SUBPARTITION_NAME, SUBPARTITION_POSITION
FROM   USER_TAB_SUBPARTITIONS
WHERE  TABLE_NAME = UPPER('vanzari_ord_hash')
AND    PARTITION_NAME = UPPER('vanz2007t1')
AND    SUBPARTITION_POSITION = ORA_HASH('3010', 4 - 1) + 1;
```

7.2. Alegerea unei anumite metode de partiționare

7.2.1 Când este recomandată metoda de partiționare prin ordonare

- Partiționarea prin ordonare este o metodă bună pentru partiționarea datelor istorice.
  - Partiționarea prin ordonare permite organizarea datelor după intervalele de timp pe baza unei coloane de tipul *DATE*.
  - Instrucțiunile *SQL* care accesează aceste submulțimi și utilizează factorul timp pot fi optimizate folosind partiționarea prin ordonare.
    - De exemplu, dacă partiționarea se realizează în funcție de lună, atunci interogarea care afișează datele lunii dec-2007 are nevoie să acceseze numai submulțimea Decembrie a anului 2007. Aceasta reduce cantitatea de date scanată la o fracțiune din totalul de date disponibil în tabelă.
- Partiționarea prin ordonare este, de asemenea, ideală atunci când, în mod periodic, se încarcă date noi și în același timp se elimină de datele vechi. Adăugarea sau eliminarea submulțimilor se realizează ușor.

❖ Partiționarea prin ordonare este utilizată în anumite cazuri.
- Tabele foarte mari sunt scanate frecvent pe baza unui predicat (care poate determina o ordonare) pe o coloană ce poate fi folosită pentru partiționare.
- Se dorește menținerea anumitor date disponibile oricând.
- Nu se pot îndeplini operațiile administrative (precum cele de *backup* și *restore*) pe tabele foarte mari în timpul alocat, dar se pot diviza în submulțimi mai mici bazate pe o cheie de partiționare.

7.2.2. Când este recomandată metoda de partiționare *hash*

- Modul în care sistemul *Oracle* distribuie datele în submulțimile *hash* nu corespunde cu o vizualizare logică a datelor, așa acum este în cazul partiționării prin ordonare. Prin urmare partiționarea *hash* nu reprezintă un mod eficient de prelucrare a datelor istorice.
- Ca regula generală, partiționările *hash* se folosesc în următoarele scopuri:
  - pentru a îmbunătăți disponibilitatea și prelucrarea tabelelor mari sau pentru a permite execuția în paralel a comenzilor *LMD* asupra tabelelor care nu mențin date istorice;
  - pentru a evita repartizarea asimetrică a datelor în submulțimi;
  - pentru a permite anumite metode de optimizare:
    - scanarea doar a submulțimilor partiției ce sunt necesare comenzii (submulțimile ce nu sunt utile comenzii nu vor fi utilizate de optimizator);
    - realizarea operațiilor de *join* pe submulțimi (dacă cheia de partiționare este coloana de *join*);
- Dacă unei partiții *hash* i se adaugă o submulțime sau dacă două submulțimi ale sale sunt contopite (*merge*), atunci în mod automat sistemul *Oracle* rearanjează înregistrările astfel încât să reflecte schimbarea numărului de submulțimi.
  - Funcția *hash* folosită este proiectată în special pentru a limita costul reorganizării datelor. Dacă partiției i se adaugă o submulțime, în loc să regrupeze toate înregistrările din tabelă, sistemul divizează o singură submulțime deja existentă. dacă două submulțimi ale partiției sunt contopite, atunci sistemul renunță la o submulțime prin contopirea celor 2 submulțimi ce se doresc a fi contopite.
  - Deși această tehnică, îmbunătățește vizibil prelucrarea tabelelor partiționate *hash*, se pot crea discrepanțe dacă numărul submulțimilor nu este o putere a lui 2. În cel mai defavorabil caz, cea mai mare submulțime poate ajunge să aibe dublul dimensiunii celei mai mici submulțimi.

- Dacă nu se specifică numele submulțimilor, atunci sistemul va genera în mod automat numele acestora. De asemenea, se poate utiliza clauza *STORE IN* pentru a stoca submulțimile într-un anumit *tablespace*.

7.2.3. Când este recomandată partiționarea prin listă

- Partiționarea prin listă este utilizată atunci când se dorește specificarea modului de mapare a înregistrărilor în submulțimi, pe baza unor valori discrete.
- Spre deosebire de celelalte tipuri de partiționări, cheile de partiționare multi-coloană nu sunt permise pentru partiționarea prin listă.
  - Dacă o tabelă este partiționată prin listă, atunci cheia de partiționare nu poate fi constituită decât de o singură coloană a tabelei.

7.2.4. Când este recomandată partiționarea compusă

- Metoda partiționării compuse este utilizată dacă:
  - submulțimile subpartiției au un înțeles logic pentru a grupa eficient datele istorice;
  - conținutul unei submulțimi trebuie distribuit în mai multe *tablespace*-uri;
  - sunt necesare metode specifice de optimizare la nivel de submulțimi (scanarea doar a submulțimilor subpartiției ce sunt necesare comenzii și *join*-uri între submulțimi, chiar dacă sunt utilizate coloane diferite ale tabelei partiționate);
  - se cere un grad de paralelism mai mare decât numărul submulțimilor pentru operații paralele, *backup* și *recovery*.

- ❖ Majoritatea tabelelor mari dintr-un depozit de date ar trebui să folosească partiționarea prin ordonare.
- ❖ Partiționarea compusă ar trebui folosită pentru tabele foarte mari sau pentru depozite de date cu nevoi foarte bine definite pentru condițiile de mai sus.

7.2.5. Metoda de optimizare *Partition Pruning*

- Această metodă implică scanarea doar a submulțimilor partiției ce sunt necesare comenzii. Optimizatorul marchează submulțimile ce trebuie accesate de comandă și le elimină din listă pe cele care nu sunt necesare.
- Optimizatorul analizează clauzele *FROM* și *WHERE* din comenzile *SQL* pentru a elimina submulțimile de care nu este nevoie atunci când se construiește lista de acces a

partiției. Astfel, se efectuează operațiile numai pe acele submulțimi care sunt relevante pentru comanda *SQL*.

- Sistemul elimină submulțimile care nu sunt necesare atunci când sunt folosite:
  - predicatele de tip egalitate, *range*, *LIKE* și *IN-list* asupra coloanelor de partiționare ale unei partiții *range*;
  - predicatele de tip egalitate și *IN-list* asupra coloanelor de partiționare ale unei partiții *hash*.
- Folosind această metodă se reduce cantitatea de date accesate de pe disc și micșorează timpul de procesare, îmbunătățind performanța de interogare și utilizare a resurselor.

Bibliografie

1. Connolly T.M., Begg C.E., *Database Systems*: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Inmon W.H., *Building the Data Warehouse*, 4th Edition, Wiley, 2005
4. Kimball R., *The Data Warehouse Toolkit*, 3rd Edition, Wiley, 2013
5. Kimball R., Ross M., Thornthwaite W., Mundy J., Becker B., *The Data Warehouse Lifecycle Toolkit*, 2nd Edition Wiley, 2008
6. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2024
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2025
8. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2025
9. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2025
10. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2025
11. Oracle and/or its affiliates, *Oracle Database: SQL Tuning Workshop*, 2010, 2025
12. Oracle and/or its affiliates, *Oracle OLAP Customizing Analytic Workspace Manager*, 2006, 2019
13. Oracle and/or its affiliates, *Oracle OLAP DML Reference*, 1994, 2019
14. Oracle and/or its affiliates, *Oracle OLAP User's Guide*, 2003, 2019
15. Oracle and/or its affiliates, *Oracle Warehouse Builder Concepts*, 2000, 2021
16. Oracle and/or its affiliates, *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*, 2000, 2021
17. Oracle and/or its affiliates, *Oracle Database Data Warehousing Guide*, 2001, 2021
18. Oracle University, *Oracle Database*: *PL/SQL Fundamentals*, *Student Guide*, 2009, 2025
19. Poe V., Klauer P., Brobst S., *Building A Data Warehouse for Decision Support*, 2nd Edition, Prentice Hall; 1997
20. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004