5. Indecși

Cuprins

5. Indecși

- Un index este un obiect al baze de date care este independent față de datele stocate în tabele, atât din punct de vedere logic, cât și fizic. Deoarece este o structură independentă, acesta necesită spațiu de stocare dedicat.
- Pentru a accesa datele dintr-o tabelă optimizatorul poate utiliza două căi de acces:
  - *rowid*-urile (identificatorii unici) înregistrărilor (*rowid scan*);
  - scanarea secvențială a tabelei (*full table scan* / *sample table scan*).
- Indexarea este utilizată pentru a facilita regăsirea rapidă a informației, fiind o tehnică de optimizare. De asemenea, *server*-ul *Oracle* poate utiliza indecșii pentru a impune constrângerile de integritate (*primary key* / *unique*).
- Un index furnizează *rowid*-uri către înregistrările care conțin o anumită valoare a cheii de indexare, implicând în acest mod acces direct și rapid la informație. În lipsa indexării, optimizatorul va alege să acceseze tabela folosind ca tehnică scanarea secvențială a acesteia.
- În *Oracle* se pot defini indecși de tip *B*Tree* sau *Bitmap*.

- O formă simplificată a comenzii *CREATE INDEX* este următoarea:

```
CREATE [ {UNIQUE | BITMAP} ] INDEX nume_index
 ON  nume_tabel (nume_coloana [, nume_coloana]…);
```

  - Dacă este omisă clauza *BITMAP*, atunci este creat un index de tip *B*Tree*, care poate fi *unique* sau nu.
  - Clauza *BITMAP* determină crearea unui index de tip *Bitmap*.

- ❖ Cei doi indecși sunt diferiți ca structură, crearea și utilizarea acestora depinzând de specificul bazei de date, al datelor menținute în coloana ce se dorește a fi indexată și al interogărilor care utilizează această coloană.
- ❖ Pentru o coloană particulară se poate preta fie indexarea de tip *B*Tree*, fie indexarea din tip *Bitmap*.

- ❖ Mai mulți indecși definiți asupra unei tabele nu implică întotdeauna interogări mai rapide.
- ❖ De asemenea, fiecare operație *LMD* care este permanentizată asupra unei tabele cu indecși asociați determină actualizarea indecșilor respectivi. Această operație

va determina costuri suplimentare.

- ❖ Prin urmare, crearea indecșilor este recomandată numai în anumite situații, în urma unei analize specifice asupra bazei de date și a aplicațiilor care rulează pe aceasta.

- Pentru a elimina un index se utilizează comanda următoare:

```
DROP INDEX nume_index;
```

- ❖ Eliminarea unei tabele implică automat eliminarea tuturor indecșilor definiți asupra acesteia.
- ❖ Eliminarea unui index este independentă față de tabelă.

Tipurile de date *ROWID*

- Aceste tipuri de date sunt permise atât în *SQL*, cât și în *PL/SQL*, permițând în ambele cazuri aceeași dimensiune maximă.

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
ROWID	Adresele fizice ale înregistrărilor (OOOOOO.FFF.BBBBBB.III)	obiect. fișier. bloc. înregistrare	obiect. fișier. bloc. înregistrare
UROWID [(n)]	Adresele logice și fizice ale înregistrărilor. Implicit 4000*bytes*	4000 bytes	4000 bytes

- ❖ *ROWID*-urile fizice stochează adresa înregistrărilor din tabelele obișnuite (care nu sunt de tip *index-organized*), *cluster*-e, partiții și subpartiții ale tabelelor, indecși, partiții și subpartiții ale indecșilor.

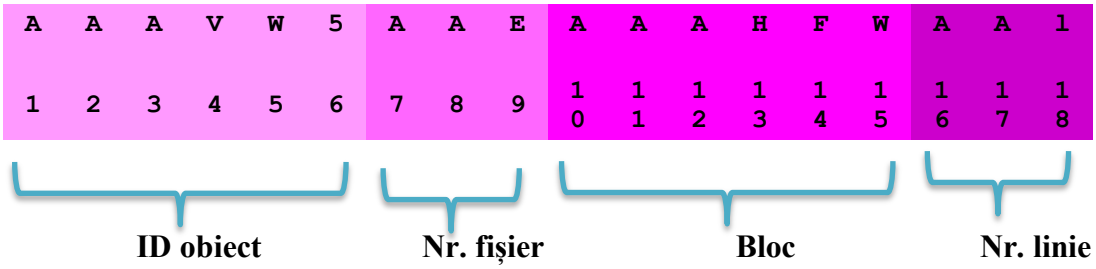

Fig. 5.1 Componentele unui *ROWID*

- ❖ *ROWID*-urile logice stochează adresa înregistrărilor din tabele de tip *index-organized*.

- ❖ Tipul de date *UROWID* (*Universal ROWID)* permite atât adresele fizice, cât și logice ale înregistrărilor dintr-o bază de date *Oracle*, dar și adresele înregistrărilor din tabele externe *non-Oracle*.

- ❖ Tabelele relaționale obișnuite stochează datele nesortate.
- ❖ O tabelă de tip *index-organized* este un tip de tabelă care stochează datele într-o structură de index *B*Tree*, sortate logic după cheia primară. Față de indexul standard creat automat la definirea unei chei primare, care stochează doar coloanele incluse în definiția cheii, indexul tabelei de tip *index-organized* stochează în general toate coloanele tabelei (coloanele care sunt accesate rar pot fi mutate în alte segmente față de cel principal).

5.1. Indecși *B*Tree*

- Un index *B*Tree* are structură de arbore balansat (*Balanced Tree*).

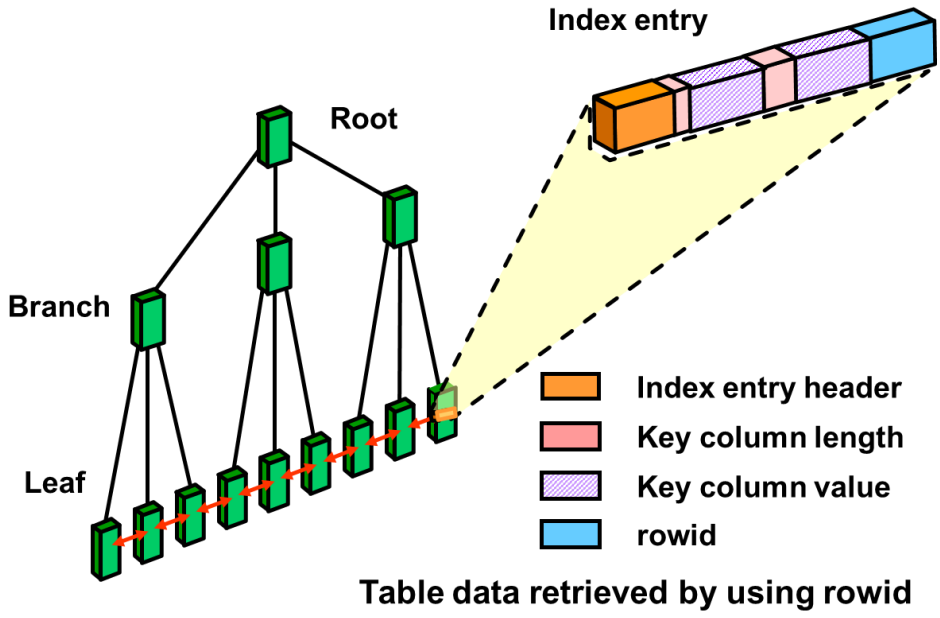

Fig. 5.1. Reprezentarea unui index *B*Tree* [11]

- ❖ Crearea unui index *B*Tree* se recomandă în următoarele situații:
  - coloana conține o varietate mare de valori, adică are cardinalitate distinctă mare (numărul de valori distincte este mare, comparativ cu numărul total de înregistrări din tabelă) ;

- coloana conține un număr mare de valori *null* (valorile *null* nu sunt indexate);
- una sau mai multe coloane sunt utilizate frecvent împreună într-o clauză *WHERE* sau într-o condiție de *join*;
- tabela este mare și este de așteptat ca majoritatea interogărilor asupra acesteia să regăsească mai puțin de 2-4% dintre linii.

❖ Coloane candidate pentru crearea unui index *B*Tree*:

- Cheia primară (indexul este creat automat de sistem);
- Cheia unică (indexul este creat automat de sistem);
- Cheia externă (indexul nu este creat automat de sistem).

5.2. Indecși *BITMAP*

- Structura unui index *Bitmap* este total diferită de cea a unui index *B*Tree*. Acesta este reprezentat printr-o hartă de biți bidimensională:
  - numărul de coloane este egal cu numărul de valori distincte ale cheii de indexare;
  - numărul de linii este egal cu numărul de înregistrări din tabela indexată;
  - dacă un *bit* are valoarea 1, atunci înregistrarea cu *ROWID*-ul corespunzător conține valoarea cheii.

❖ Reprezentarea internă a *bitmap*-urilor este adecvată aplicațiilor cu niveluri scăzute ale tranzacțiilor concurente.

❖ Indecșii *Bitmap* sunt eficienți pentru cererile care conțin multe condiții în clauza *WHERE*. Înregistrările care satisfac anumite condiții, dar nu pe toate, sunt filtrate înainte ca tabela să fie accesată. Aceasta îmbunătățește semnificativ timpul de răspuns.

- Indecșii de tip *Bitmap* sunt des utilizați în bazele de date depozit deoarece:
  - reduc timpul de răspuns pentru multe clase de cereri *ad hoc*;
  - reduc dimensiunile spațiului de stocare necesar, comparativ cu indecșii *B*Tree*;
  - permit mărirea performanțelor chiar dacă din punct de vedere *hardware* sistemul are un număr mic de procesoare sau un volum mic de memorie disponibilă;
  - permit menținerea eficientă a sistemului în timpul tranzacțiilor *LMD* paralele și în timpul încărcărilor cu date.

- Indexarea folosind indecși *B*Tree* a tabelelor de dimensiuni mari poate fi costisitoare din punct de vedere al spațiului de stocare, deoarece indecșii pot fi de câteva ori mai mari decât datele indexate din tabele. Din acest punct de vedere, indecșii *Bitmap* reprezintă doar o parte din spațial ocupat de datele indexate.
- Indecșii *Bitmap* sunt creați special pentru aplicațiile care preponderent interoghează datele. Aceste tipuri de indecși nu sunt recomandate pentru aplicațiile *OLTP* cu multe tranzacții concurente care actualizează datele.
- Interogările și operațiile *LMD* paralele utilizează indecșii *Bitmap* la fel ca și indecșii *B*Tree*.

Cardinalitatea coloanei indexate

- Cele mai multe avantaje ale utilizării indecșilor *Bitmap* se obțin atunci când sunt utilizați pe coloane cu cardinalitate distinctă mică (numărul de valori distincte este mic, comparativ cu numărul total de înregistrări din tabelă).
  - De exemplu, o coloană booleană (care are cardinalitatea distinctă 2) este ideală pentru un index *Bitmap*.
  - Totuși se creează indecși *Bitmap* și pentru coloane cu cardinalități distincte mari. De exemplu, într-o tabelă cu 1 milion de înregistrări, o coloană cu 10.000 de valori distincte reprezintă un candidat bun pentru un index *Bitmap*.

- Indecșii *B-tree* sunt mai eficienți pe coloane cu cardinalități mari, adică pentru date care au multe valori posibile. În *data warehouse*, indecșii *B-tree* ar trebui utilizați pentru coloanele cu valori unice sau pentru coloanele cu cardinalități mari. Majoritatea indecșilor în *data warehouse* trebuie să fie de tip *bitmap*.

- În cazul cererilor *ad hoc*, indecșii *Bitmap* îmbunătățesc considerabil performanțele interogărilor.
  - Condițiile *AND* sau *OR* din clauzele *WHERE* pot fi rapid rezolvate prin execuția operațiilor booleene corespunzătoare direct pe indecșii *Bitmap*, înainte de a converti hărțile de biți rezultate în *ROWID*-iuri. Dacă numărul de linii rezultate este mic, atunci răspunsul cererii este dat rapid, fără a fi necesară o scanare totală a întregii tabele.

Exemplul 5.1

Presupunem că tabela *clienti* conține următoarele atribute:

- atributul *tip*, care poate avea 2 valori distincte

  S - standard

  V - VIP

- atributul *stare_civila*, care poate avea 3 valori distincte

  C - căsătorit

  N - necăsătorit

  N/A - nespecificat

- atributul *grila_venit*, care poate avea 5 valori distincte

  A - salariul în intervalul [1000, 2000)

  B - salariul în intervalul [2000, 3000)

  C - salariul în intervalul [3000, 4000)

  D - salariul în intervalul [4000, 5000)

  E - salariul >=5000

Deoarece coloanele *tip*, *stare_civila* și *grila_venit* au cardinalități distincte mici, indecșii *Bitmap* sunt ideali pentru aceste coloane. Aceștia vor fi definiți folosind comenzile următoare:

```
CREATE BITMAP INDEX ind_clienti_tip
ON clienti(tip);

CREATE BITMAP INDEX ind_clienti_sc
ON clienti(stare_civila);

CREATE BITMAP INDEX ind_clienti_gv
ON clienti(grila_venit);
```

Pentru exemplificare sunt utilizate următoarele înregistrări ale tabelei *clienti*:

ROWID	CLIENT_ID	TIP	STARE_CIVILA	GRILA_VENIT
AAAR5iAAFAAAAC9AAG	1	S	N/A	A
AAAR5iAAFAAAAC9AAF	2	S	C	D
AAAR5iAAFAAAAC9AAC	3	V	N	D
AAAR5iAAFAAAAC9AAA	4	S	N/A	B
AAAR5iAAFAAAAC9AAB	5	S	C	C
AAAR5iAAFAAAAC9AAE	6	V	N	A
AAAR5iAAFAAAAC9AAD	7	V	N/A	C
AAAR5iAAFAAAAC9AAR	8	V	C	E

În tabelul următor este exemplificat indexul *bitmap* pentru coloana *tip*.

ROWID	'V'	'S'
AAAR5iAAFAAAAC9AAG	0	1
AAAR5iAAFAAAAC9AAF	0	1
AAAR5iAAFAAAAC9AAC	1	0
AAAR5iAAFAAAAC9AAA	0	1
AAAR5iAAFAAAAC9AAB	0	1
AAAR5iAAFAAAAC9AAE	1	0
AAAR5iAAFAAAAC9AAD	1	0
AAAR5iAAFAAAAC9AAR	1	0

În tabelul următor este exemplificat indexul *bitmap* pentru coloana *stare_civila*.

ROWID	'C'	'N'	'N/A'
AAAR5iAAFAAAAC9AAG	0	0	1
AAAR5iAAFAAAAC9AAF	1	0	0
AAAR5iAAFAAAAC9AAC	0	1	0
AAAR5iAAFAAAAC9AAA	0	0	1
AAAR5iAAFAAAAC9AAB	1	0	0
AAAR5iAAFAAAAC9AAE	0	1	0
AAAR5iAAFAAAAC9AAD	0	0	1
AAAR5iAAFAAAAC9AAR	1	0	0

În tabelul următor este exemplificat indexul *bitmap* pentru coloana *grila_venit*.

ROWID	'A'	'B'	'C'	'D'	'E'
AAAR5iAAFAAAAC9AAG	1	0	0	0	0
AAAR5iAAFAAAAC9AAF	0	0	0	1	0
AAAR5iAAFAAAAC9AAC	0	0	0	1	0
AAAR5iAAFAAAAC9AAA	0	1	0	0	0
AAAR5iAAFAAAAC9AAB	0	0	1	0	0
AAAR5iAAFAAAAC9AAE	1	0	0	0	0
AAAR5iAAFAAAAC9AAD	0	0	1	0	0
AAAR5iAAFAAAAC9AAR	0	0	0	0	1

Se dă următoarea cerere: „Câți dintre clienții de tip *VIP* căsătoriți câștigă un venit de nivel *D* sau *E*?“

```
SELECT COUNT(*)
FROM   clienti
WHERE  tip = 'VIP' AND stare_civila = 'C'
AND    grila_venit IN ('D', 'E');
```

Indecșii *bitmap* pot eficientiza procesarea acestei cereri prin aplicarea de operații logice pe biți și numărarea aparițiilor numărului 1 în rezultatul final.

ROWID	'V'	'C'	'D'	'E'	'V' AND 'C' OR ('D','E')
AAAR5iAAFAAAAC9AAG	0	0	0	0	0
AAAR5iAAFAAAAC9AAF	0	1	1	0	0
AAAR5iAAFAAAAC9AAC	1	0	1	0	0
AAAR5iAAFAAAAC9AAA	0	0	0	0	0
AAAR5iAAFAAAAC9AAB	0	1	0	0	0
AAAR5iAAFAAAAC9AAE	1	0	0	0	0
AAAR5iAAFAAAAC9AAD	1	0	0	0	0
AAAR5iAAFAAAAC9AAR	1	1	0	1	1

Valorile *null*

- Comparativ cu indecșii *B*Tree*, indecșii *Bitmap* includ și înregistrările care conțin valori *null*. Indexarea valorilor *null* poate fi utilă pentru anumite tipuri de cereri *SQL*, ca de exemplu cererile care conțin funcția grup *COUNT*.

Exemplul 5.2

Presupunem că tabela *clienti* conține și atributul *gen*, care poate avea 3 valori distincte:

M - masculin

F - feminin

Null - nespecificat

Deoarece coloana *gen* are cardinalitate distincte mică, se poate defini un index *Bitmap* pe aceasta, utilizând comanda următoare:

```
CREATE BITMAP INDEX ind_clienti_gen
ON clienti(gen);
```

În tabelul următor este exemplificat indexul *bitmap* pentru coloana *gen*.

ROWID	'F'	'M'	'null'
AAAR5iAAFAAAAC9AAG	1	0	0
AAAR5iAAFAAAAC9AAF	0	1	0
AAAR5iAAFAAAAC9AAC	0	0	1
AAAR5iAAFAAAAC9AAA	1	0	0
AAAR5iAAFAAAAC9AAB	0	1	0
AAAR5iAAFAAAAC9AAE	0	1	0
AAAR5iAAFAAAAC9AAD	0	0	1
AAAR5iAAFAAAAC9AAR	1	0	0

Cererea următoare va folosi un indexul *bitmap* definit pe coloana *gen*.

```
SELECT COUNT(*)
FROM   clienti
WHERE  gen IS NULL;
```

Pentru cererea următoare orice index *bitmap* poate fi folosit, deoarece toate înregistrările tabelei sunt indexate, incluzându-le pe cele care conțin valori *null*. Dacă valorile *null* nu sunt indexate, optimizatorul poate folosi indecșii doar pe coloanele care au constrângeri *NOT NULL*.

```
SELECT COUNT(*)
FROM   clienti;
```

Indecși *bitmap join*

- Se pot crea indecși *bitmap* asupra unei singure tabele sau asupra unui *join* de două sau mai multe tabele (indecși *bitmap join*).
- Un index *bitmap join* reprezintă o modalitate eficientă pentru a reduce volumul de date care intră în *join*, deoarece restricțiile se execută înainte. Acest tip de index stochează pentru fiecare valoare a coloanei dintr-o tabelă valorile *ROWID* corespunzătoare dintr-unul sau mai multe tabele.

- ❖ În mediile *data warehouse*, condiția de *join* este un *equi-inner join* între coloana sau coloanele care fac parte din cheia primară a tabelelor dimensiune și coloana sau coloanele care fac parte din cheia externă a tabelei de fapte.

- ❖ Indecșii *bitmap join* sunt mai eficienți față de vizualizările materializate, din punct de vedere al spațiului de stocare, fiind o metodă de a materializa *join*-urile în avans.

Exemplul 5.2

Presupunem că tabela *vanzari* are structura și datele exemplificate în tabelul următor:

ROWID	ID	id_client	id_timp	id_produs	cantitate	valoare
AAAR5iAAFAAAAC9AAS	10	1	20201101	101	3	3
AAAR5iAAFAAAAC9AAK	20	2	20201101	167	1	10
AAAR5iAAFAAAAC9AAJ	30	1	20201102	256	4	80
AAAR5iAAFAAAAC9AAH	40	1	20201103	589	10	100
AAAR5iAAFAAAAC9AAI	50	4	20201104	936	2	90
AAAR5iAAFAAAAC9AAN	60	3	20201104	101	5	5
AAAR5iAAFAAAAC9AAM	70	2	20201104	832	3	900

AAAR5iAAFAAAAC9AAL	80	5	20201104	936	1	45
AAAR5iAAFAAAAC9AAP	90	1	20201105	256	2	40
AAAR5iAAFAAAAC9AAQ	100	7	20201105	256	10	200

Presupunem că se definește următorul index *bitmap join*

```
CREATE BITMAP INDEX ind_vanz_tip_client
ON     vanzari(tip)
FROM   vanzari v, clienti c
WHERE  c.id_client = v.client_id;
```

În tabelul următor este exemplificat indexul *bitmap join* pentru coloana *tip*.

ROWID_vanzare	ROWID_client	'S'	'V'
AAAR5iAAFAAAAC9AAS	AAAR5iAAFAAAAC9AAG	1	0
AAAR5iAAFAAAAC9AAK	AAAR5iAAFAAAAC9AAF	1	0
AAAR5iAAFAAAAC9AAJ	AAAR5iAAFAAAAC9AAG	1	0
AAAR5iAAFAAAAC9AAH	AAAR5iAAFAAAAC9AAG	1	0
AAAR5iAAFAAAAC9AAI	AAAR5iAAFAAAAC9AAA	1	0
AAAR5iAAFAAAAC9AAN	AAAR5iAAFAAAAC9AAC	0	1
AAAR5iAAFAAAAC9AAM	AAAR5iAAFAAAAC9AAF	1	0
AAAR5iAAFAAAAC9AAL	AAAR5iAAFAAAAC9AAE	0	1
AAAR5iAAFAAAAC9AAP	AAAR5iAAFAAAAC9AAG	1	0
AAAR5iAAFAAAAC9AAQ	AAAR5iAAFAAAAC9AAD	0	1

Următoarea cerere va utiliza indexul creat anterior:

```
SELECT id_timp, tip, cantitate
FROM   vanzari v, clienti c
WHERE  c.id_client = v.client_id;
```

Exemplul 5.3

Comanda următoare determină crearea unui index *bitmap join* care referă mai multe coloane a unei tabele.

```
CREATE BITMAP INDEX ind_vanz_tip_sc
ON vanzari(tip, stare_civila)
FROM   vanzari v, clienti c
WHERE  c.id_client = v.client_id;
```

Exemplul 5.4

Comanda următoare determină crearea unui index *bitmap join* care referă coloane din mai multe tabele.

```
CREATE BITMAP INDEX ind_vanz_tip_prod_categ
ON vanzari(tip, categorie_1)
FROM   vanzari v, clienti c, produse p
WHERE  c.id_client = v.client_id
AND    p.id_produs = v.produs_id;
```

❖ Rezultatele *join*-ului trebuie stocate și, de aceea, indecșii *bitmap join* au următoarele restricții:

- doar operațiile *LMD* paralele sunt permise în mod curent pe tabelele de fapte; operațiile *LMD* paralele pe unul dintre tabelele dimensiune participante vor determina ca indexul să fie inutilizabil;
- atunci când sunt utilizați indecși *Bitmap* doar o singură tabelă poate fi actualizată în mod concurent de tranzacții diferite;
- nicio tabelă nu poate apărea de două ori în *join*;
- nu se poate crea un index *bitmap join* pe o tabelă de tip *index-organized* sau pe o tabelă temporară;
- coloanele din index trebuie să fie doar coloane din tabelele dimensiune;
- coloanele care intră în *join* dintr-o tabelă dimensiune trebuie să facă parte din cheia primară sau să aibă definită o constrângere de unicitate (constrângerile respective trebuie definite cu opțiunea *validate*);
- dacă o tabelă dimensiune are cheie primară compusă, atunci fiecare coloană care face parte din cheia primară trebuie să facă parte din *join*.

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