



Sisteme de baze de date

Curs 7 – Organizarea logică a bazei de date (Indecși, alte obiecte, dictionarul datelor)

Sorina Predut

sorina.predut@unibuc.ro

Universitatea din București



Indecși

- Un index este o **structură opțională a bazei de date care permite accesarea directă a unui rând dintr-un tabel.**
- Indecșii pot fi creați pentru una sau mai multe coloane a unui tabel, în acest ultim caz folosindu-se denumirea de indecși compuși sau indecși concatenați.
- Un index este **utilizat** de către baza de date **pentru a găsi rapid valori pentru coloana sau coloanele pentru care a fost creat indexul**, în acest mod furnizând o cale de acces directă la liniile asociate acestora **fără a mai fi necesară investigarea fiecărui rând din tabel.**



Tipuri de indecși

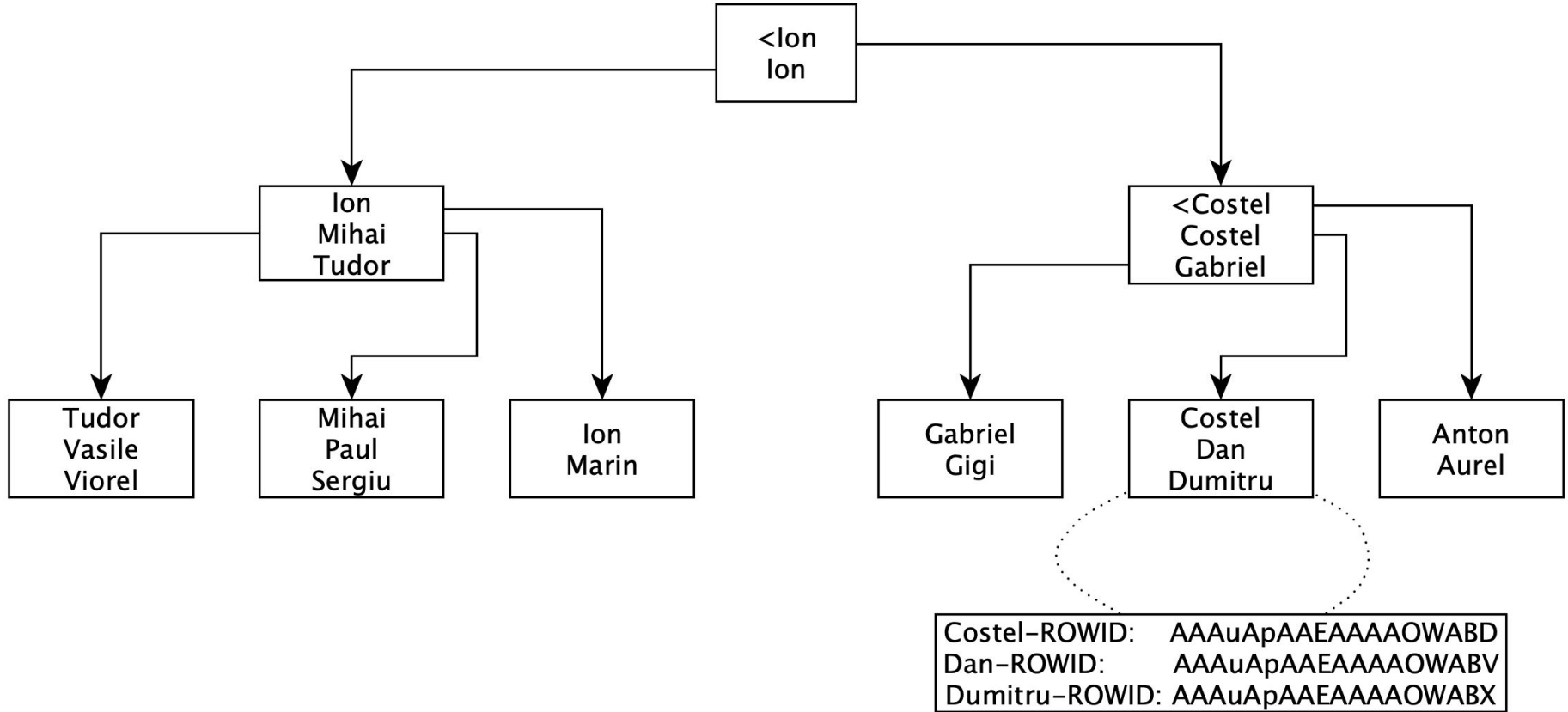
Index de tip arbore B*

- Cel mai întâlnit tip de index folosit de Oracle este indexul de tip arbore B* (**B*-tree index sau balanced tree index**).

Acesta este tipul de index creat la executarea unei comenzi standard CREATE INDEX și tipul de index la care ne-am referit anterior.

Un arbore B*, este un arbore în care pentru găsirea oricărei valori din arbore e necesar același număr de pași, indiferent de valoarea căutată.

Figura următoare ilustrează structura unui index de tip arbore B*.





Tipuri de indecși - cont.

- Pentru indecși neunici, dacă există mai multe ROWID pentru aceeași valoare a datelor indexate, atunci și acestea sunt sortate, deci indecșii neunici sunt sortați întâi după valoarea datelor și apoi după ROWID. Valorile nule nu sunt indexate, cu excepția indecșilor de cluster.



Tipuri de indecși - cont.

Index partiționat

- O noutate adusă de versiunea Oracle8 este posibilitatea de partiționare a unui index. Așa cum un tabel poate fi partiționat, la rândul lui un index de tip arbore B^* poate fi și el partiționat. Indecșii partiționați sunt folosiți în cazul tabelelor mari pentru a stoca valorile coloanei indexate în mai multe segmente. Practic, pentru fiecare partiție a tabelului poate fi creată o partiție a unui index. Prin urmare, partiționarea permite stocarea unui index în mai multe spații tabel. Unul din avantajele partiționării este descreșterea domeniului de valori în care indexul caută o anumită valoare.



Tipuri de indecși - cont.

- Pentru a crea un index partiționat se folosește comanda CREATE index cu clauza PARTITION.

Un index partiționat, la fel ca și un tabel partiționat permite definirea unor spații table și parametrii de stocare diferiți pentru fiecare parte.

∃ **2 moduri de a defini un index partiționat: local și global.**

- **local:** partițiile indexului sunt similare cu partițiile tabelului; în acest caz indexul este partiționat în funcție de aceeași coloană, pe același interval de valori ca și tabelul la care se referă; partițiile trebuie enunțate în aceeași ordine ca și partițiile tabelului la care se referă.

Aceasta este metoda cel mai des folosită deoarece partiționarea este dirijată de tabelul de bază.



Tipuri de indecși - cont.

- Sintaxa pentru crearea unui index partiționat local este următoarea:

```
CREATE INDEX nume_index ON tabel (coloana [,coloana] ...)
LOCAL
    (PARTITION nume_partiție
    [PCTFREE întreg] [PCTUSED întreg]
    [TABLESPACE spațiu_tabel][STORAGE parametrii_de_stocare]
    [,PARTITION nume_partiție
    VALUES[LESS|GREATER]THAN(listă_valori)
    [PCTFREE întreg] [PCTUSED întreg]
    [TABLESPACE spațiu_tabel][STORAGE parametrii_de_stocare]]...)
```




Tipuri de indecși - cont.

- Următorul exemplu creează un index partiționat local pe baza tabelului partiționat salariat_part.

```
CREATE INDEX nume_sal_part_ind ON salariat_part(nume)
LOCAL
    (PARTITION salariu_mic
     TABLESPACE ts_ind_alfa STORAGE (initial 10K next 10K),
     PARTITION salariu_mediu
     TABLESPACE ts_ind_beta STORAGE (initial 20K next 20K),
     PARTITION salariu_mare
     TABLESPACE ts_ind_alfa STORAGE (initial 10K next 10K));
```



Tipuri de indecși - cont.

- **global:** partițiile indexului sunt definite de utilizator și nu sunt similare cu partițiile tabelului la care se referă indexul.

Sintaxa pentru crearea unui index partiționat global este următoarea:

```
CREATE INDEX nume_index ON tabel (coloana [,coloana] ...)
GLOBAL PARTITION BY RANGE (listă_coloane)
    (PARTITION nume_partiție VALUES [LESS|GREATER] THAN
      (listă_valori) [PCTFREE întreg] [PCTUSED întreg]
      [TABLESPACE spațiu_tabel][STORAGE parametrii_de_stocare]
      [,PARTITION nume_partiție VALUES [LESS|GREATER] THAN
        (listă_valori)
        [PCTFREE întreg] [PCTUSED întreg]
        [TABLESPACE spațiu_tabel][STORAGE parametrii_de_stocare]]...)
```



Tipuri de indecși - cont.

- Următoarea comandă creează indexul global nume_sal_ind pentru tabela salariat, index ce va avea două partiții:

```
CREATE INDEX nume_sal_ind ON salariat(nume)
GLOBAL PARTITION BY RANGE (nume)
(PARTITION VALUES LESS THAN ('N')
TABLESPACE ts_alfa_ind,
PARTITION VALUES LESS THAN (MAXVALUE)
TABLESPACE ts_beta_ind);
```



Tipuri de indecși - cont.

Index de cluster

- Un index de cluster (index de grup) este un index bazat pe coloanele comune ale unui cluster.

Nu se pot efectua nici un fel de comenzi DML asupra unui cluster până când nu a fost creat un index de cluster.

Gruparea tabelor în cluster nu afectează crearea de indecși suplimentari pentru tabele individuale; aceștia pot fi creați sau distruși ca de obicei.



Tipuri de indecși - cont.

Index cu cheie inversă

- Un index cu cheie inversă (**reverse-key index**) reprezintă o nouă metodă (adusă de versiunea Oracle8) de a îmbunătăți anumite tipuri de căutări.
Așa cum s-a discutat până acum, pentru sporirea eficienței căutărilor unor valori în baza de date se folosesc de obicei arborii B*.
Există totuși unele cazuri în care aceștia îngreunează accesul la date.
De exemplu, presupunem că avem o coloană indexată ce conține prenumele a mii de persoane.
Să presupunem de asemenea că în această coloană există mii de prenume ce încep cu litera 'S'.



Tipuri de indecși - cont.

- În momentul în care se dorește inserarea mai multor înregistrări ce încep cu litera 'S' **pot apărea șteangulări ale operațiilor de citire/scriere** deoarece modificările în index-ul asociat coloanei vor apărea în același nod al arborelui.

Indecșii cu cheie inversă sunt folosiți tocmai în această situație deoarece ei stochează datele în mod invers.

Prin urmare, prenumele 'SANDU' va fi stocat tot într-un arbore B^* ca 'UDNAS'.

Acest tip de index este folositor numai în cazul căutării în arbore a unor valori exacte.



Tipuri de indecși - cont.

- Un index cu cheie inversă se creează folosind comanda CREATE INDEX cu opțiunea REVERSE, de exemplu:

```
CREATE INDEX sal_prenume_ind ON salariat(prenume) REVERSE;
```

- Un index obișnuit se poate transforma în index cu cheie inversă folosind comanda ALTER INDEX... REBUILD cu opțiunea REVERSE.

De exemplu:

```
ALTER INDEX sal_prenume_ind REBUILD REVERSE;
```



Tipuri de indecși - cont.

Index de tip bitmap

- Un alt tip de index, introdus în Oracle8, este indexul de tip bitmap.
Într-un astfel de index, **în loc de a se stoca valorile propriu-zise ale coloanei indexate, indexul stochează un bitmap format pe baza acestor valori.**
Cu alte cuvinte, indexul ține un bitmap pentru fiecare rând, bitmap care conține un bit pentru fiecare rând din tabel.
Bitul este 1 dacă valoarea respectivă este conținută în acel rând și 0 dacă nu este.
De exemplu, să presupunem că avem un index de tip bitmap creat pe baza coloanei culoare dintr-un tabel masina:



Tipuri de indecși - cont.

Tabel masina

Nr_masina	Marca	Culoare
1	Ford Mondeo	Alb
2	Dacia Nova	negru
3	Daewoo Cielo	alb
4	Daewoo Tico	verde
5	Ford Mondeo	roșu
6	Ford Mondeo	albastru
7	Dacia Nova	alb
8	Dacia Nova	negru
9	Daewoo Cielo	verde
10	Ford Mondeo	verde
11	Dacia Nova	verde
12	Daewoo Tico	verde

13	Ford Mondeo	albastru
14	Dacia Nova	verde
15	Daewoo Tico	albastru
16	Dacia Nova	verde
17	Ford Mondeo	alb
18	Daewoo Tico	negru
19	Dacia Nova	verde
29	Ford Mondeo	verde

Index de tip bitmap pentru coloana culoare

Culoare = alb	Culoare = negru	Culoare = verde	Culoare = albastru	Culoare = roșu
1	0	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	1	0	0
0	0	0	0	1
0	0	0	1	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	1	0	0
0	0	0	1	0
0	0	1	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0



Tipuri de indecși - cont.

- În acest exemplu, coloana culoare poate avea numai cinci valori: alb, negru, roșu, verde, albastru, iar în tabel sunt 20 de rânduri.

Bitmap-ul corespunzător fiecărei culori va avea 20 de biți, fiecare dintre aceștia având valoarea 1 când mașina va avea culoarea respectivă și 0 în caz contrar.

Deci bitmap-ul corespunzător culorii alb va avea 1 pe pozițiile 1, 3, 7, 17, cel corespunzător culorii negru va avea 1 pe pozițiile 2, 8, 18, etc.

De exemplu, pentru aflarea nr. de mașini albe se va executa interogarea:

```
SELECT COUNT(*)  
FROM masina  
WHERE culoare = 'alb';
```



Tipuri de indecși - cont.

- Un index de tip bitmap poate procesa foarte eficient o astfel de interogare prin simpla numărare a valorilor 1 din bitmap-ul corespunzător valorii alb.
- Un index de tip bitmap se creează folosind comanda CREATE INDEX cu opțiunea BITMAP, de exemplu:

```
CREATE BITMAP INDEX culoare_ind ON masina(culoare);
```



Tipuri de indecși - cont.

- Spre deosebire de indecșii tradiționali de tip arbore B*, folosirea indecșilor de tip bitmap se recomandă atunci când:
 - nr. de valori distincte ale coloanei indexate este relativ mic (coloana are cardinalitate mică); de exemplu, în cazul unei coloane ce conține starea civilă sau sexul unei persoane.
 - majoritatea interogărilor conțin combinații multiple de condiții WHERE ce implică operatorul OR.
- În acest caz, indecșii de tip bitmap pot avea o dimensiune mult mai mică decât indecșii de tip arbore B*.

Pe de altă parte însă, indecșii de tip bitmap sunt ineficienți în cazul unor coloane cu număr mare de valori.



Tabele organizate pe bază de index

- Tabelele organizate pe bază de index (**index-organised tables**) sunt o noutate adusă de versiunea Oracle 8.
Un tabel organizat pe bază de index diferă față de un tabel obișnuit prin faptul că datele tabelului sunt stocate în indexul asociat.
Ori de câte ori se vor face modificări asupra datelor din tabel, precum adăugarea de noi rânduri, modificarea sau ștergerea rândurilor existente, se va modifica doar indexul.



Tabele organizate pe bază de index

- Mai exact, un tabel organizat pe bază de index este ca un tabel obișnuit având un index de tip arbore B* pe una sau mai multe coloane, dar în loc de a folosi spații separate de stocare pentru tabel și index, **Oracle folosește doar un singur index de tip B*, care conține atât valorile coloanelor indexate, cât și valorile celorlalte coloane pentru rândul corespunzător.**

Deci, în loc ca fiecare intrare a indexului să conțină valoarea coloanei sau coloanelor indexate și valoarea ROWID pentru rândul corespunzător, ea conține întreg rândul.

Coloana sau coloanele după care se face indexarea sunt cele care constituie cheia primară a tabelului.

Din această cauză, tabelele organizate pe index sunt eficiente pentru accesarea datelor prin intermediul cheii primare sau un prefix valid al acesteia.



Tabele organizate pe bază de index

- Un tabel organizat pe bază de index poate fi manipulat de către aplicații la fel ca un tabel obișnuit, folosind comenzi SQL.
Diferența constă în faptul că în cazul tabelului organizat pe bază de index, toate operațiile sunt efectuate numai asupra indexului.
- Următorul tabel rezumă diferențele cele mai importante dintre un tabel obișnuit și un tabel organizat pe bază de index.

Tabel Obișnuit	Tabel organizat pe bază de index
<p>ROWID identifică în mod unic un rând; specificarea cheii primare este opțională. Are coloana implicită ROWID. Accesul la date se face prin intermediul ROWID.</p> <p>Permite definirea de indecși pentru coloane care nu fac parte din cheia primară. Permite definirea constrângerii UNIQUE și a triggerelor. Poate fi stocat într-un cluster conținând alte tabele. Poate fi partiționat.</p>	<p>Cheia primară identifică în mod unic un rând; specificarea cheii primare este obligatorie. Nu are coloana implicită ROWID. Accesul la date se face prin intermediul cheii primare.</p> <p>Nu permite definirea de indecși pentru coloane care nu fac parte din cheia primară. Nu permite definirea constrângerii UNIQUE, dar permite definirea triggerelor. Nu poate fi stocat într-un cluster conținând alte tabele. Nu poate fi partiționat.</p>



Tabele organizate pe bază de index

- Principalele avantaje ale tabelelor organizate pe baza de index sunt următoarele:
 - Se reduce timpul de acces la date prin interogări care folosesc ca termen de comparație coloanele indexate,
 - Se reduce spațiul de stocare datorită faptului că nu mai este creat un index suplimentar pentru cheia primară a tabelului.



Crearea tabelelor organizate pe bază de index

- Pentru a crea un tabel organizat pe bază de index, se folosește comanda SQL CREATE TABLE cu specificația ORGANIZATION INDEX.
La crearea unui tabel organizat pe bază de index, trebuie neapărat specificată cheia primară a tabelului.
În plus, se pot folosi clauzele OVERFLOW, THRESHOLD și INCLUDING prezentate în continuare, care sunt opționale.



Crearea tabelelor organizate pe bază de index

- O intrare dintr-un index de tip arbore B^* este de obicei destul de mică, constând dintr-o pereche (valoare indexată, ROWID).
Pe de altă parte însă, într-un tabel organizat pe bază de index, intrările din indexul arborelui B^* corespunzător pot fi foarte mari deoarece ele constau dintr-o pereche (cheie_primară, coloane_secundare), adică un rând întreg.
Dacă acestea sunt foarte mari, se poate ajunge la situația în care fiecare nod frunză să conțină doar un singur rând sau o porțiune de rând, distrugându-se astfel densitatea indexului de tip arbore B^* .



Crearea tabelelor organizate pe bază de index

- Pentru a evita această problemă, la crearea unui tabel organizat pe index se poate folosi clauza `OVERFLOW`, care specifică un spațiu tabel de depășire (`OVERFLOW TABLESPACE`).

Alături de această clauză se mai poate specifica și o valoare de prag (`PCTTHRESHOLD`).

Valoarea de prag este specificată ca procent din mărimea unui bloc.

Dacă dimensiunea rândului depășește valoarea de prag specificată, atunci coloanele care nu fac parte din cheia primară pentru rândurile care depășesc valoarea de prag vor fi stocate în spațiul tabel de depășire specificat.



Crearea tabelelor organizate pe bază de index


- În acest caz, intrările din index vor conține perechea (cheie_primară, capăt_rînd), unde capăt_rînd conține partea de început a restului de coloane; acesta este exact ca o porțiune normală de rînd, cu excepția faptului că face referință la porțiunea rămasă de rînd, care este stocată în spațiul tabel de depășire.
Dacă nu este specificată clauza OVERFLOW, atunci toate liniile care depășesc valoarea de prag sunt eliminate complet și nu mai sunt inserate în tabela organizată pe index.
- Clauza INCLUDING, dacă este specificată, determină coloana care va împărți tabelul în porțiunea de index și porțiunea de depășire.
Fiecare coloană după coloana INCLUDING este stocată în porțiunea de depășire a tabelului organizat pe index.



Crearea tabelelor organizate pe bază de index

- Deci, sintaxa comenzii CREATE TABLE pentru un tabel organizat pe index este următoarea:

```
CREATE TABLE nume_tabel  
(nume_coloana tip_data ... PRIMARY KEY ...)  
ORGANIZATION INDEX  
TABLESPACE spațiu_tabel  
[PCTTHRESHOLD întreg]  
[INCLUDING nume_coloana]  
[OVERFLOW TABLESPACE spațiu_tabel_depășire]
```



```
CREATE TABLE carte
(
    serie VARCHAR2(5),
    numar NUMBER(7),
    titlu VARCHAR2(20),
    cod_autor VARCHAR2(10),
    editura VARCHAR2(10),
    descriere VARCHAR2(200)
    CONSTRAINT pk_carte PRIMARY KEY (serie, numar)
)
ORGANIZATION INDEX TABLESPACE ts_alfa
PCTTHRESHOLD 40
INCLUDING editura
OVERFLOW TABLESPACE ts_beta;
```




Vederi

- O vedere este un “**tabel logic**”, fiind de asemenea organizată în rânduri și coloane.
- Ea preia rezultatul unei interogări și îl tratează ca pe un tabel, de unde și numele de tabel logic.
- De exemplu, dacă din tabelul salariat se dorește vizualizarea doar a 5 coloane (cod_salariat, nume, prenume, data_nastere, cod_dept) și numai a rândurilor pentru care cod_tara = 40, se poate crea o vedere care conține numai aceste linii și coloane:

Vederi - cont.

tabel de bază salariat

cod_ salariat	nume	prenume	data_ naștere	salariu	manager	Cod_ Dept	cod_tara
101	Popescu	Ion	11-DEC-77	5000		1	44
102	Vasilescu	Vasile	12-JAN-77	3000	101	1	40
103	Georgescu	Ilie	01-MAY-78	3000	101	1	44
104	Enescu	Gică	11-JUN-66	2000	102	1	40
105	Georgescu	Viorel	02-APR-77	2000	104	2	40

vedere salariat_40

cod_ salariat	nume	prenume	data_ naștere	cod_ dept
102	Vasilescu	Vasile	12-JAN-77	1
104	Enescu	Gică	11-JUN-66	1
105	Georgescu	Viorel	02-APR-77	2



Vederi - cont.

- O vedere poate fi construită din una sau mai multe tabele sau chiar alte vederi și permite ca datele din mai multe tabele să fie rearanjate, reunite logic sau ca noi date să fie calculate pe baza acestora.
- Dpdv al aplicației, vederile au același comportament ca și tabelele: vederile pot și interogate și, **cu anumite excepții** care vor fi menționate mai târziu, **asupra vederilor se pot efectua operații DML** (INSERT, DELETE, UPDATE).
- Vederea este un instrument foarte puternic pentru dezvoltatorul de aplicații. Ea poate fi bazată pe mai multe tabele sau vederi care se pot găsi pe mașini diferite sau pot aparține unor utilizatori diferiți, acestea fiind prezentate ca și cum ar fi un singur tabel logic.



Vederi - cont.

- Spre deosebire de tabel, **vederea nu stochează date și nici nu are alocat vreun spațiu de stocare**; vederea doar extrage sau derivă datele din tabelele la care aceasta se referă. Aceste tabele poartă numele de **tabele de bază ale vederii**. Acestea pot fi tabele sau pot fi ele însele vederi.
- Oracle stochează definiția vederii în dicționarul de date sub forma textului interogării care definește vederea, de aceea **o vedere poate fi gândită ca o “interogare stocată”**.
- Pentru vizualizarea definițiilor vederilor se poate folosi coloana TEXT a vederilor ALL_VIEWS, DBA_VIEW și USER_VIEW din dicționarul de date.



Vederi - cont.

- Vederile pot fi interogate exact la fel ca tabelele, folosind comanda SELECT.
- Când o interogare SQL se referă la o vedere, Oracle combină această interogare cu interogarea care definește vederea.
- În general, vederile sunt create pentru următoarele scopuri:
 - Asigurarea unui nivel mai mare de securitate a bazei de date prin limitarea accesului la un nr. mai restrâns de linii și coloane ale unui tabel.
 - Simplificarea interogărilor SQL, permițând vizualizarea unor date care în mod normal necesită interogări SQL destul de complicate.

De exemplu, o vedere poate permite utilizatorilor vizualizarea datelor din mai multe tabele fără ca aceștia să fie obligați să folosească un SELECT pe mai multe tabele.



Vederi - cont.

- Prezentarea diferită a datelor față de cea din tabelele de bază.
De exemplu, coloana unei vederi poate avea alt nume decât coloana corespunzătoare din tabelul de bază, acest lucru neafectând în nici un fel tabelul de bază.
- Efectuarea unor interogări care nu ar putea fi efectuate fără existența unei vederi. De exemplu, este posibilă definirea unei vederi care realizează un join între o vedere care include clauza GROUP BY și un alt tabel; acest lucru nu poate fi făcut într-o singură interogare.



Vederi - cont.

- Pentru a menține calcule mai complicate.
Interogarea care definește vederea poate efectua calcule complicate asupra datelor dintr-un tabel; prin menținerea acestei interogări ca o vedere, calculele pot fi efectuate de fiecare dată când se face referire la vedere.
- Asigurarea transparentă a datelor pentru anumiți utilizatori și aplicații.
O vedere poate conține date din mai multe tabele, care pot fi proprietatea mai multor utilizatori.



Crearea vederilor

- O vedere este creată folosind comanda SQL CREATE VIEW.

De exemplu:

```
CREATE VIEW salariat_40
AS SELECT cod_salariat, nume, prenume, salariu, cod_dept
FROM salariat
WHERE cod_tara = 40;
```




Crearea vederilor - cont.

- O sintaxă simplificată a comenzii CREATE VIEW este următoarea:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW nume_vedere  
[(alias [, alias]...)]  
AS subinterogare  
[WITH READ ONLY]  
[WITH CHECK OPTION [CONSTRAINT nume_cosntrangere]]  
unde
```



Crearea vederilor - cont.

- **OR REPLACE** recreează vederea dacă ea există deja.

Această opțiune poate fi folosită pentru a schimba definiția unei vederi existente fără a o distruge în prealabil.

Avantajul recreării vederii prin opțiunea REPLACE este că în acest caz se păstrează toate privilegiile acordate asupra acestei vederi.

De exemplu, să presupunem că după crearea unei vederi, au fost acordate privilegiile asupra vederii pentru anumite roluri sau pentru anumiți utilizatori.

Dacă după aceea vederea este distrusă și recreată, atunci toate privilegiile asupra vederii au fost pierdute și trebuie acordate din nou.

Dacă vederea este însă recreată folosind opțiunea OR REPLACE, atunci privilegiile acordate sunt păstrate și nu mai este necesară acordarea lor încă o dată.



Crearea vederilor - cont.

- **FORCE** este o opțiune care permite crearea vederii indiferent dacă tabelele de bază și coloanele la care se face referire există sau nu, sau dacă utilizatorul posedă sau nu privilegiile corespunzătoare în legătură cu tabelele respective.
Opțiunea opusă, **NOFORCE**, creează vederea numai dacă tabelele de bază există și dacă utilizatorul posedă privilegiile corespunzătoare în legătură cu tabelele respective; **NOFORCE** este opțiunea implicită.
Dacă se folosește opțiunea **FORCE** și un tabel de bază nu există sau una dintre coloane nu este validă, atunci Oracle va crea vederea cu erori de compilare.
Dacă mai târziu tabelul în cauză este creat sau coloana este corectată, atunci vederea poate fi folosită, Oracle recompilând-o dinamic înainte de folosire.



Crearea vederilor - cont.

- **alias** specifică numele expresiilor selectate de interogarea vederii.
Numărul alias-urilor trebuie să fie același cu numărul de expresii selectate de către interogarea vederii.
Un alias trebuie să fie unic în cadrul unei interogări.
Dacă sunt omise alias-urile, Oracle va folosi denumirile coloanelor din interogare.
Atunci când interogarea vederii conține și expresii, nu doar simple coloane, trebuie folosite alias-uri.
- **AS** indică interogarea vederii.
Aceasta poate fi orice instrucțiune SELECT care nu conține clauzele ORDER BY și FOR UPDATE.



Crearea vederilor - cont.

- opțiunea **WITH READ ONLY** asigură că nici o operație DML (inserare, ștergere, modificare) nu va fi asigurată asupra vizualizării.
- **WITH CHECK OPTION** este o constrângere care arată că toate actualizările efectuate prin intermediul vederii vor afecta tabelele de bază numai dacă actualizările respective vor avea ca rezultat numai rânduri care pot fi vizualizate prin intermediul vederii. **CONSTRAINT** furnizează un nume pentru constrângerea **CHECK OPTION**.
Asupra acestor opțiuni vom reveni puțin mai târziu.
- Exemplul următor ilustrează crearea unei vederi care conține codul, numele, prenumele, salariul și sporul salarial pentru toți salariații din departamentul 1 și țara cu codul 40.



Crearea vederilor - cont.

```
CREATE OR REPLACE VIEW salariat_1
(cod, nume, prenume, salariu, spor_salariu)
AS SELECT cod_salariat, nume, prenume, salariu, salariu*0.1
FROM salariat
WHERE cod_dept = 1 AND cod_tara = 40;
```

- Pentru datele din tabelul salariat vederea salariat_1 va conține următoarele date:

salariat_1

cod	nume	prenume	salariu	spor_salariu
102	Vasilescu	Vasile	3000	300
104	Enescu	Gică	3000	300



Operații DML asupra vederilor

- În momentul în care în tabelele de bază sunt adăugate noi date sau sunt actualizate sau șterse cele existente, aceste modificări se reflectă corespunzător în vederile bazate pe aceste tabele.
Acest lucru este adevărat și viceversa, cu singura mențiune că există anumite restricții la inserarea, actualizarea sau ștergerea datelor dintr-o vedere.
Acele restricții sunt redată pe scurt în continuare:



Operații DML asupra vederilor - cont.

- Nu pot fi inserate, șterse sau actualizate datele din vederi care conțin una dintre următoarele:
 - operatorul DISTINCT (pentru eliminarea duplicatelor);
 - clauzele GROUP BY, HAVING, START WITH, CONNECT BY;
 - pseudo-coloana ROWNUM (această pseudo-coloană conține un număr ce indică ordinea în care Oracle selectează înregistrările dintr-un tabel);
 - funcțiile de grup (COUNT, SUM, MAX, MIN, AVG, STDDEV, VARIANCE, GLB);
 - operatorii de mulțimi (UNION, UNION ALL, INTERSECT, MINUS).



Operații DML asupra vederilor - cont.

- Nu pot fi inserate sau actualizate valorile coloanelor care rezultă prin calcul, de exemplu coloana `spor_salariu` de mai înainte.
De asemenea nu se pot efectua operații DML asupra valorilor coloanelor care au fost calculate folosind funcția `DECODE`.
- Nu pot fi inserate sau actualizate date care ar încălca constrângerile din tabele de bază. De exemplu, dacă în tabela `salariat` coloana `nume` este definită ca `NOT NULL`, atunci în orice vedere bazată pe acest tabel care nu conține coloana `nume` (de exemplu `salariat_exemplu` definită ulterior) nu va fi posibilă inserarea de date deoarece aceasta ar duce la încălcarea constrângerii `NOT NULL`.



Operații DML asupra vederilor - cont.

```
CREATE VIEW salariat_exemplu  
AS SELECT cod_salariat, prenume, data_nastere  
FROM salariat;
```

- În Oracle pot fi inserate, șterse sau actualizate datele din vederi bazate pe mai multe tabele, însă cu anumite excepții, care vor fi discutate mai târziu.



Operații DML asupra vederilor - cont.

- În plus față de regulile de mai înainte, la crearea unei vederi se poate utiliza clauza WITH CHECK OPTION care impune ca singurele date care pot fi inserate sau actualizate prin intermediul vederii să fie numai acelea care pot fi vizualizate de aceasta. Pentru a clarifica acest aspect, să considerăm vederea salariat_2000 definită mai jos și posibilitățile existente de a insera rânduri în această vedere.

```
CREATE VIEW salariat_2000
AS SELECT cod_salariat, nume, prenume, data_nastere, salariu
FROM salariat
WHERE salariu > 2000;
```



Operații DML asupra vederilor - cont.

- Prima comandă SQL de mai jos va duce la inserarea unui rând în tabela de bază salariat, care poate fi vizualizat și prin intermediul vederii salariat_2000, deoarece valoarea corespunzătoare coloanei salariu este mai mare decât 2000.
Pe de altă parte, a doua comandă SQL va duce la inserarea unui rând în tabelul de bază care nu poate fi însă vizualizat prin intermediul vederii.

```
INSERT INTO salariat_2000
      (cod_salariat, nume, prenume, data_nastere, salariu)
VALUES (106, 'Ionescu', 'Vasile', '11-JUL-60' , 3000);
```

```
INSERT INTO salariat_2000
      (cod_salariat, nume, prenume, data_nastere, salariu)
VALUES (107, 'Popescu', 'Viorel', '22-JAN-69' , 1000);
```



Operații DML asupra vederilor - cont.

- De exemplu, dacă înaintea executării acestor comenzi în tabelul salariat există datele menționate anterior, atunci după executarea acestor comenzi, datele din tabelul salariat și vederea salariat_2000 vor fi următoarele:

tabel de bază salariat

cod_ salariat	Nume	prenume	data_ nastere	salariu	manager	cod_ dept	cod_tara
101	Popescu	Ion	11-DEC-77	5000		1	44
102	Vasilescu	Vasile	12-JAN-77	3000	101	1	40
103	Georgescu	Ilie	01-MAY-78	3000	101	1	44
104	Enescu	Gică	11-JUN-66	2000	102	1	40
105	Georgescu	Viorel	02-APR-77	2000	104	2	40
106	Ionescu	Vasile	11-JUL-60	3000			
107	Popescu	Viorel	22-JAN-69	1000			

vedere salariat_2000

Cod_ Salariat	nume	prenume	data_ nastere	salariu
101	Popescu	Ion	11-DEC-77	5000
102	Vasilescu	Vasile	12-JAN-77	3000
103	Georgescu	Ilie	01-MAY-78	3000
106	Ionescu	Vasile	11-JUL-60	3000



Operații DML asupra vederilor - cont.

- Să presupunem acum că aceeași vedere, salariat_2000, este creată folosind clauza WITH CHECK OPTION:

```
CREATE VIEW salariat_2000
AS SELECT cod_salariat, nume, prenume, data_nastere, salariu
FROM salariat
WHERE salariu > 2000
WITH CHECK OPTION;
```



Operații DML asupra vederilor - cont.

- Și în acest caz, inserarea în vedere a unui rând pentru care valoarea coloanei salariu este mai mare decât 2000 se face fără probleme.
Pe de altă parte însă, orice încercare de a insera în vedere rânduri pentru care salariul este mai mic sau egal cu 2000 va produce o eroare indicând încălcarea constrângerii WITH CHECK OPTION, comanda nemodificând tabelul de bază.
De exemplu, executarea celor două comenzi de INSERT de mai devreme va avea în acest caz ca rezultat următoarele date din salariat și salariat_2000.

tabel de bază salariat

cod_ salariat	nume	prenume	data_ nastere	salariu	manager	cod_ dept	cod_tara
101	Popescu	Ion	11-DEC-77	5000		1	44
102	Vasilescu	Vasile	12-JAN-77	3000	101	1	40
103	Georgescu	Ilie	01-MAY-78	3000	101	1	44
104	Enescu	Gică	11-JUN-66	2000	102	1	40
105	Georgescu	Viorel	02-APR-77	2000	104	2	40
106	Ionescu	Vasile	11-JUL-60	3000			

vedere salariat_2000

cod_ salariat	nume	prenume	data_ nastere	salariu
101	Popescu	Ion	11-DEC-77	5000
102	Vasilescu	Vasile	12-JAN-77	3000
103	Georgescu	Ilie	01-MAY-78	3000
106	Ionescu	Vasile	11-JUL-60	3000



Operații DML asupra vederilor - cont.

- În cazul folosirii constrângerii CHECK OPTION, acestea i se poate atribui un nume folosind opțiunea CONSTRAINT din cadrul comenzii CREATE VIEW. De exemplu:

```
CREATE VIEW salariat_2000
AS SELECT cod_salariat, nume, prenume, data_nastere, salariu
FROM salariat
WHERE salariu > 2000
WITH CHECK OPTION CONSTRAINT salariu_2000;
```

- Dacă opțiunea CONSTRAINT este omisă, Oracle atribuie în mod automat constrângerii CHECK OPTION un nume de forma "SYS_Cn" unde n este un întreg care face ca numele constrângerii să fie unic în baza de date.



Operații DML asupra vederilor bazate pe mai multe tabele (Join-Views)

- Așa cum am menționat mai înainte, în Oracle este posibilă inserarea, actualizarea sau ștergerea datelor dintr-o vedere bazată pe mai multe tabele, cu anumite restricții însă. Alături de restricțiile generale, aplicabile tuturor vederilor, prezentate anterior, există și restricții specifice numai vederilor bazate pe mai multe tabele. Acestea sunt redată de următoarele reguli:
- **Regula generală:** Orice operație de INSERT, UPDATE sau DELETE pe o vedere bazată pe mai multe vederi poate modifica datele doar din unul dintre tabelele de bază.



Join-Views - cont.

- Înainte de a enunța regulile specifice pentru fiecare dintre operațiile INSERT, UPDATE sau DELETE este necesară definirea conceptului de **tabel protejat de cheie (key-preserved table)**.
- Dată fiind o vedere bazată pe mai multe tabele, un tabel de bază al vederii este protejat prin cheie **dacă orice cheie selectată a tabelului este de asemenea și cheie a vederii**.
Deci, un tabel protejat prin cheie este un tabel ale cărui chei se păstrează și la nivel de vedere.
Trebuie reținut că, pentru a fi protejat prin cheie, nu este necesar ca un tabel să aibă toate cheile selectate în vedere.



Join-Views - cont.

- Este suficient ca, atunci când cheia tabelului este selectată, aceasta să fie și cheie a vederii. Proprietatea unui tabel de a fi protejat prin cheie nu este o proprietate a datelor din tabel, ci o proprietate a schemei.
- În exemplul următor, dacă pentru fiecare combinație (cod_tara, cod_dept) ar exista un singur salariat, atunci combinația (cod_tara, cod_dept) din tabelul departament ar fi unică pentru datele din vederea rezultată, dar tabelul departament tot nu ar fi protejat prin cheie. Pentru a ilustra această noțiune cât și regulile următoare, să considerăm o definiție simplificată a tabelelor departament și salariat în care păstrăm doar constrângerile de cheie primară și integritate referențială.

```
CREATE TABLE departament(  
  cod_dept NUMBER(10),  
  cod_tara NUMBER(10),  
  nume_dept VARCHAR2(10),  
  PRIMARY KEY(cod_dept, cod_tara));
```

```
CREATE TABLE salariat(  
  cod_salariat NUMBER(10) PRIMARY KEY,  
  nume VARCHAR2(10),  
  prenume VARCHAR2(10),  
  data_nastere DATE,  
  manager NUMBER(10) REFERENCES salariat(cod_salariat),  
  salariu NUMBER(10),  
  cod_dept NUMBER(10),  
  cod_tara NUMBER(10),  
  FOREIGN KEY(cod_dept, cod_tara) REFERENCES departament(cod_dept,  
  cod_tara));
```



Join-Views - cont.

- Să mai considerăm și următoarea vedere bazată pe aceste două tabele:

```
CREATE VIEW sal_dept  
  (cod_salariat, nume, prenume, salariu, cod_dept, cod_tara, nume_dept)  
AS SELECT s.cod_salariat, s.numa, s.prenume, s.salariu, s.cod_dept,  
  s.cod_tara, d.nume_dept  
FROM salariat s, departament d  
WHERE s.cod_dept = d.cod_dept  
AND s.cod_tara = d.cod_tara;
```

În exemplul de mai sus, tabelul salariat este protejat prin cheie. Regulile specifice pentru fiecare dintre operațiile DML (INSERT, UPDATE sau DELETE) sunt redade în continuare:



Join-Views - cont.

➤ **Reguli de actualizare (UPDATE):**

1. Toate coloanele care pot fi actualizate printr-o vedere trebuie să corespundă coloanelor dintr-un tabel protejat prin cheie.

Dacă o coloană provine dintr-o tabelă neprotejată prin cheie, atunci Oracle nu va putea identifica în mod unic înregistrarea care va trebui actualizată.

De exemplu, comanda SQL de mai jos se va executa cu succes.

```
UPDATE sal_dept  
SET salariu = salariu + 100  
WHERE cod_tara = 40;
```




Join-Views - cont.

Pe de altă parte, comanda următoare va eșua:

```
UPDATE sal_dept  
SET nume_dept = 'IT'  
WHERE cod_dept = 1 AND cod_tara = 40;
```

Comanda de mai sus va eșua pentru că ea încearcă să actualizeze o coloană din tabelul departament, tabel care nu este protejat prin cheie.



Join-Views - cont.

2. Dacă vederea este definită folosind clauza WITH CHECK OPTION, atunci toate coloanele de joncțiune și toate coloanele tabelor repetate nu pot fi modificate.
De exemplu, dacă vederea sal_dept ar fi fost definită folosind clauza WITH CHECK OPTION, atunci comanda următoare va eșua deoarece încearcă modificarea unei coloane de joncțiune.

```
UPDATE sal_dept  
SET cod_dept = 2  
WHERE cod_salariat = 101 AND cod_tara = 40;
```



Join-Views - cont.

➤ **Reguli de inserare (INSERT):**

3. O comandă INSERT nu poate să se refere în mod explicit sau implicit la coloane dintr-un tabel care nu este protejat prin cheie.

De exemplu, dacă în tabelul departament există o linie cu `cod_dept = 3` și `cod_tara = 40`, atunci următoarea comandă SQL va fi executată cu succes:

```
INSERT INTO sal_dept (cod_salariat, nume, cod_dept, cod_tara)
VALUES (110, 'Marinescu', 3, 40);
```

În caz contrar, comanda va eșua, fiind încălcată constrângerea de integritate referențială.

Pe de altă parte, comanda următoare va eșua deoarece ea încearcă inserarea de date în mai multe tabele.

```
INSERT INTO sal_dept (cod_salariat, nume, nume_dept)
VALUES (111, 'Georgescu', 'IT');
```



Join-Views - cont.

4. Dacă o vedere este definită folosind clauza WITH CHECK OPTION, atunci nu se pot executa comenzi INSERT în acea vedere.
- **Reguli de ștergere (DELETE):**
5. Rândurile dintr-o vedere pot fi șterse numai dacă în joncțiune există un tabel protejat prin cheie și numai unul.

Dacă ar exista mai multe tabele, Oracle nu ar ști din care tabel să șteargă rândul.

De exemplu, comanda SQL de mai jos se va executa cu succes deoarece ea poate fi tradusă într-o operație de ștergere pe tabelul salariat:

```
DELETE FROM sal_dept  
WHERE nume = 'Popescu';
```



Join-Views - cont.

Pe de altă parte, dacă se încearcă executarea unei comenzi DELETE pe vederea de mai jos, ea va eșua deoarece ambele tabele de bază, s1 și s2, sunt protejate prin cheie:

```
CREATE VIEW emp_emp AS  
SELECT s1.nume, s2.prenume  
FROM salariat s1, salariat s2  
WHERE s1.cod_salariat = s2.cod_salariat;
```



Join-Views - cont.

6. Dacă vederea este definită folosind clauza WITH CHECK OPTION, atunci nu pot fi șterse rânduri din vedere.

De exemplu, nu se poate executa o instrucțiune DELETE pe vederea de mai jos deoarece ea este definită ca auto-joncțiune a unui tabel protejat prin cheie.

```
CREATE VIEW emp_manag AS
SELECT s1.nume, s2.nume nume_manager
FROM salariat s1, salariat s2
WHERE s1.manager = s2.cod_salariat
WITH CHECK OPTION;
```



Join-Views - cont.

- Vederile `ALL_UPDATABLE_COLUMNS`, `DBA_UPDATABLE_COLUMNS` și `USER_UPDATABLE_COLUMNS` ale dicționarului de date conțin informații care arată care dintre coloanele vederilor existente pot fi actualizate.
Vederile care nu pot fi actualizate direct pot fi actualizate folosind triggeri `INSTEAD OF` (vor fi discutate mai târziu).



Recompilarea vederilor

- Recompilarea unei vederi permite detectarea eventualelor erori referitoare la vederea respectivă înaintea executării vederii.

După fiecare modificare a tabelor de bază este recomandabil ca vederea să se recompileze.

Acest lucru se poate face folosind comanda SQL ALTER VIEW ... COMPILE:

```
ALTER VIEW salariat_2000 COMPILE;
```




Distrugerea vederilor

- Pentru distrugerea unei vederi se folosește comanda DROP VIEW:

```
DROP VIEW salariat_2000;
```

- Utilizarea unei vederi este câteodată o sabie cu două tăișuri.

Deși vederile constituie un instrument extrem de convenabil pentru dezvoltatorul de aplicații, ele pot avea un impact negativ asupra performanței acestora.

Dacă pentru vederi simple (de exemplu simple copii ale unui tabel) impactul asupra performanței nu este sesizabil, pentru vederi complexe, care cuprind interogări pe mai multe tabele, acest impact poate fi foarte sever.

De aceea folosirea vederilor trebuie planificată cu grijă, pentru a se vedea dacă avantajul creat de simplitatea în manipulare a acestora compensează impactul negativ asupra performanței.



Secvențe

- De multe ori este necesară **crearea unei secvențe de numere pentru a fi folosite ca valori ale cheii unui tabel**, de exemplu coloana cod_salariat din tabelul salariat.

În loc de a genera aceste numere manual, **Oracle oferă o facilitare pentru generarea unei astfel de secvențe în mod automat.**

Pentru a crea manual o secvență de numere, ar fi necesară blocarea rândului care conține ultima valoare a secvenței (pentru a evita preluarea acestei valori de mai multe ori), generarea noii valori și apoi deblocarea rândului.

Blocarea acestor rânduri poate fi evitată prin folosirea generatorului de secvențe furnizat de Oracle.

Acesta poate genera secvențe de numere de până la 38 de digiți, fără a fi necesară blocarea manuală a rândurilor.



Secvențe - cont.

- Secvențele sunt memorate și generate indiferent de tabele.
Prin urmare, aceeași secvență poate fi utilizată pentru mai multe tabele.
- **O secvență este practic un obiect al bazei de date care servește la generarea unor numere întregi unice care poate fi folosită simultan de mai mulți utilizatori, evitând apariția conflictelor și a blocării.**
- Ca orice alt obiect al bazei de date, o secvență poate fi creată, modificată sau distrusă.
- Oracle stochează definițiile secvențelor în dicționarul de date.



Crearea, modificarea și distrugerea secvențelor

- Pentru a crea o secvență se folosește comanda SQL CREATE SEQUENCE, având următoarea sintaxă:

```
CREATE SEQUENCE nume_secvență  
[INCREMENT BY întreg]  
[START WITH întreg]  
[MAXVALUE întreg | NOMAXVALUE]  
[MINVALUE întreg | NOMINVALUE]  
[CYCLE | NOCYCLE]  
[CACHE întreg | NOCACHE]  
[ORDER | NOORDER]
```

unde



Secvențe - cont.

- **INCREMENT BY** specifică intervalul dintre 2 numere ale secvenței.
Aceasta poate fi orice valoare pozitivă sau negativă, dar nu poate fi 0.
Dacă valoarea este negativă atunci secvența este în ordine descrescătoare, dacă este pozitivă, atunci secvența este în ordine crescătoare.
Valoarea implicită (default) este 1.
- **START WITH** specifică prima valoare generată de secvență.
Această opțiune poate fi folosită pentru a începe o secvență crescătoare cu o valoare mai mare decât valoarea sa minimă sau pentru a începe o secvență descrescătoare cu o valoare mai mică decât valoarea sa maximă.
Pentru secvențe crescătoare valoarea implicită este valoarea minimă a secvenței, iar pentru secvențe descrescătoare valoarea implicită este valoarea maximă a secvenței.



Secvențe - cont.

- **MINVALUE** specifică valoarea minimă a secvenței.
NOMINVALUE specifică o valoare minimă de 1 pentru o secvență crescătoare sau -10 pentru o secvență descrescătoare.
NOMINVALUE este valoarea implicită.
- **MAXVALUE** specifică valoarea maximă a secvenței.
NOMAXVALUE specifică o valoare maximă de 10 pentru o secvență descrescătoare și 1027 pentru o secvență crescătoare.
NOMAXVALUE este valoarea implicită.



Secvențe - cont.

- **CYCLE** arată că secvența continuă să genereze valori și după ce a atins valoarea maximă (în cazul secvențelor crescătoare) sau valoarea minimă (în cazul secvențelor descrescătoare). În acest caz, după ce o secvență crescătoare a atins valoarea maximă, ea va genera valoarea sa minimă; după ce o secvență descrescătoare a atins valoarea minimă, ea va genera valoarea sa maximă.
NOCYCLE arată că secvența nu mai poate genera valori după ce a atins valoarea maximă (în cazul secvențelor crescătoare) sau valoarea minimă (în cazul secvențelor descrescătoare). Valoarea implicită este **NOCYCLE**.



Secvențe - cont.

- **CACHE** arată câte valori ale secvenței sunt prealocate de către Oracle și ținute în memorie pentru acces mai rapid.
Valoarea minimă a acestui parametru este 2.
NOCACHE arată ca nu există valori prealocate ale secvenței.
Dacă se omit amândouă opțiunile, CACHE și NOCACHE, Oracle prealocă implicit 20 de valori.



Secvențe - cont.

- Următorul exemplu creează secvența sal_seq:

```
CREATE SEQUENCE sal_seq  
INCREMENT BY 1  
START WITH 1  
NOMAXVALUE  
NOCYCLE  
CACHE 10;
```



Secvențe - cont.

- Fiecare dintre parametrii specificați la crearea unei secvențe pot fi modificați prin executarea comenzii SQL ALTER SEQUENCE:

```
ALTER SEQUENCE sal_seq  
MAXVALUE 100000  
CYCLE  
CACHE 20;
```

- O secvență poate fi distrusă folosind comanda SQL DROP SEQUENCE:

```
DROP SEQUENCE sal_seq;
```



Utilizarea secvențelor

- După definirea unei secvențe, aceasta poate fi utilizată și incrementată de mai mulți utilizatori.
Oracle nu așteaptă încheierea unei tranzacții care accesează secvența pentru a permite utilizarea ei de către un alt utilizator.
- O secvență poate fi referită într-o comandă SQL cu ajutorul pseudo-coloanelor **NEXTVAL** și **CURRVAL**, sub forma **nume_secventa.NEXTVAL** și respectiv **nume_secventa.CURRVAL**.
Valoarea următoare este generată de pseudo-coloana NEXTVAL, în timp ce valoarea curentă este repetată la folosirea pseudo-coloanei CURRVAL.



Utilizarea secvențelor - cont.

- În exemplul următor, secvența sal_seq este folosită pentru a insera o noua linie în tabelul salariat:

```
INSERT INTO salariat (cod_salariat, nume, prenume, data_nastere)
VALUES (sal_seq.NEXTVAL, 'Georgescu', 'Vasile', '11-MAY-69');
```

- La generarea unui nou număr din secvență, acesta este disponibil numai sesiunii care l-a creat.

Orice altă sesiune care folosește aceeași secvență va obține o nouă valoare din secvență - evident, tot prin intermediul pseudo-coloanei NEXTVAL.



Utilizarea secvențelor - cont.

- Pentru a utiliza valoarea curentă a secvenței în sesiunea curentă de lucru se folosește pseudo-coloana CURRVAL.

Valoarea CURRVAL este disponibilă numai dacă NEXTVAL a fost folosită în sesiunea curentă - în caz contrar încercarea de a folosi pseudo-coloana CURRVAL va produce o eroare.

De exemplu, să presupunem că și generarea de valori pentru coloana cod_dept din tabelul departament se face prin intermediul unei secvențe dept_seq.

Atunci, următoarele comenzi SQL vor insera 2 departamente și câte 2 salariați pentru fiecare departament:



Utilizarea secvențelor - cont.

```
INSERT INTO departament (cod_dept, cod_tara, nume_dept)
VALUES (dept_seq.NEXTVAL, 40, 'Proiectare');
```

```
INSERT INTO salariat (cod_salariat, nume, prenume, data_nastere, cod_dept,
cod_tara)
VALUES (sal_seq.NEXTVAL, 'Vasilescu', 'Costel', '17-MAY-70',
dept_seq.CURRVAL, 40);
```

```
INSERT INTO salariat (cod_salariat, nume, prenume, data_nastere, cod_dept,
cod_tara)
VALUES (sal_seq.NEXTVAL, 'Popescu', 'Vasile', '17-JUN-72', dept_seq.CURRVAL,
40);
```



Utilizarea secvențelor - cont.

```
INSERT INTO departament (cod_dept, cod_tara, nume_dept)
VALUES (dept_seq.NEXTVAL, 40, 'Vanzari');
```

```
INSERT INTO salariat (cod_salariat, nume, prenume, data_nastere, cod_dept,
cod_tara)
VALUES (sal_seq.NEXTVAL, 'Ionescu', 'Gheorghe', '12-MAY-71',
dept_seq.CURRVAL, 40);
```

```
INSERT INTO salariat (cod_salariat, nume, prenume, data_nastere, cod_dept,
cod_tara)
values (sal_seq.NEXTVAL, 'Diaconescu', 'Marian', '15-MAY-57',
dept_seq.CURRVAL, 40);
```



Utilizarea secvențelor - cont.

- CURRVAL și NEXTVAL pot fi folosite în următoarele locuri:
 - clauza VALUES a unei comenzi INSERT, ca în exemplul de mai înainte.
 - clauza SET a unei comenzi UPDATE, ca în exemplul de mai jos:

```
UPDATE salariat  
SET cod_salariat = sal_seq.NEXTVAL  
WHERE cod_salariat = 7;
```
 - lista unei comenzi SELECT, ca în exemplul de mai jos:

```
SELECT sal_seq.NEXTVAL  
FROM dual;
```




Utilizarea secvențelor - cont.

- CURRVAL și NEXTVAL nu pot fi folosite în următoarele locuri:
 - o subinterogare;
 - interogarea unei vederi sau a unui instantaneu;
 - o comandă SELECT cu operatorul DISTINCT;
 - o comandă SELECT cu clauza GROUP BY sau ORDER BY;
 - o comandă SELECT care este combinată cu altă comandă SELECT printr-un operator de mulțime (UNION, INTERSECT, MINUS);
 - clauza WHERE a unei comenzi SELECT;
 - valoarea DEFAULT a unei coloane într-o comandă CREATE TABLE sau ALTER TABLE;
 - condiția unei constrângeri CHECK.



Utilizarea secvențelor - cont.

- Atunci când un nr. este generat de către o secvență, aceasta este incrementată indiferent dacă tranzacția a fost actualizată permanent (committed) sau derulată înapoi (rolled back). Dacă 2 sau mai mulți utilizatori incrementează aceeași secvență în mod concurent, numerele generate la cererea unuia dintre ei pot să nu fie consecutive deoarece între timp au putut fi generate alte valori la cererea celorlalți utilizatori. Un utilizator nu poate avea niciodată acces la un număr generat de o secvență la cererea unui alt utilizator. În plus, un utilizator nu poate ști dacă un alt utilizator generează numere folosind aceeași secvență.



Utilizarea secvențelor - cont.

- Dacă numerele generate de către o secvență sunt folosite la popularea unui câmp, este posibil ca valorile din acel câmp să nu fie numere consecutive – în cazul când o tranzacție a fost derulată înapoi.
De exemplu, dacă valoarea CURRVAL este 10 în cazul secvenței sal_seq de mai înainte, acest lucru nu implică neapărat faptul că tabelul salariat conține 10 înregistrări.



Sinonime

- Un sinonim este **un obiect care face referire la un alt obiect din baza de date, cu alte cuvinte un alias al acestuia.**
- Prin urmare, Oracle oferă posibilitatea de a atribui mai multe nume aceluiași obiect.
- În Oracle se pot crea sinonime pentru tabele, vederi, instantanee, secvențe, funcții, proceduri sau pachete din baza de date sau chiar pentru alte sinonime din baza de date. Deoarece un sinonim este doar un alias, el nu necesită nici un spațiu de stocare, în afara de definiția sa din dicționarul bazei de date.



Sinonime - cont.

- Sinonimele sunt folosite de obicei pentru a permite unui utilizator să folosească obiecte din schema altui utilizator, fără a specifica proprietarul acestora.
Când un utilizator folosește un sinonim, acesta trebuie să cunoască doar numele sinonimului, nu și proprietarul sau numele obiectului referit de sinonim.
Sinonimele sunt mecanisme puternice prin intermediul cărora se realizează independența unui obiect al bazei de date față de utilizatorul care este proprietarul său.



Sinonime - cont.

- Un sinonim este de 2 feluri: **public sau privat**.
- Un sinonim public este proprietatea grupului special de utilizatori numit PUBLIC și poate fi folosit de către toți utilizatorii bazei de date.
- Un sinonim privat este proprietatea numai a unui utilizator, putând fi accesat la fel ca orice obiect din schema acestuia.

Pe de altă parte, trebuie reținut că definirea unui sinonim, fie el public sau privat, nu implică accesul la obiectul referit de acesta de către alți utilizatori.

Pentru a accesa respectivul obiect, utilizatorul trebuie să posede privilegiile adecvate pentru accesarea obiectului.



Crearea, modificarea și distrugerea sinonimelor

- Pentru a crea un sinonim se folosește comanda SQL CREATE SYNONYM, având sintaxa:
`CREATE [PUBLIC] SYNONYM [schema.]nume_sinonim
FOR [schema.]obiect;`
- Dacă se specifică opțiunea PUBLIC, atunci sinonimul creat este public, altfel sinonimul este privat.
- Pentru a înțelege cum funcționează sinonimele, să considerăm, de exemplu că tabelul salariat aparține schemei utilizatorului costica și că un al utilizator, mitica, creează un sinonim public, sal1, și un sinonim privat, sal2, pentru acesta:
`CREATE PUBLIC SYNONYM sal1 FOR costica.salariat;
CREATE SYNONYM sal2 FOR costica.salariat;`



Sinonime - cont.

- Atunci, pentru a vizualiza toate datele din tabela salariat, orice utilizator poate folosi următoarea comandă SQL:

```
SELECT * FROM sal1;
```

În schimb, doar utilizatorul mitica poate folosi comanda SQL de mai jos pentru a vizualiza datele din tabelul salariat - aceasta deoarece sal2 face parte din schema acestui utilizator.

```
SELECT * FROM sal2;
```

În plus, așa cum s-a menționat mai devreme, pentru a putea executa comenzile de mai sus, utilizatorii respectivi trebuie să posede privilegiul SELECT pe tabelul de bază salariat.

- Numele atribuite sinonimelor pot fi aceleași cu ale obiectelor de bază (bineînțeles, cu excepția situației când sinonimul este privat și aparține aceleiași scheme ca și obiectul de bază).



Sinonime - cont.

- Pentru a distruge un sinonim din baza de date se folosește comanda:

```
DROP [PUBLIC] SYNONYM [schema.]nume_sinonim;
```

De exemplu:

```
DROP PUBLIC SYNONYM sal1;
```

```
DROP SYNONYM sal2;
```



Dicționarul de Date

- Dicționarul de date este o componentă esențială a unei baze de date Oracle.
- Dicționarul de date reprezintă o mulțime de tabele care pot fi doar vizualizate (read-only) și care furnizează informații despre baza de date.
- El este creat automat la crearea bazei de date și conține:
 - definițiile tuturor obiectelor de schemă din baza de date (tabele, vederi, indecși, sinonime, secvențe, proceduri, funcții, pachete, trigger, etc.);
 - cât spațiu a fost alocat și cât spațiu este utilizat în prezent de obiectele de schemă;
 - valori implicite (default) pentru coloane;
 - informații despre constrângeri de integritate;
 - numele utilizatorilor bazei de date;
 - privilegiile și rolurile acordate fiecărui utilizator;
 - alte informații generale despre baza de date;



Dicționarul de Date - cont.

- Dicționarul de date este structurat în tabele și vederi, exact ca oricare alte date ale bazei de date.
- Toate tabelele și vederile dicționarului de date pentru o anumită bază de date sunt **stocate în spațiul tabel SYSTEM** al bazei de date respective și sunt **proprietatea utilizatorului SYS**. Din această cauză, **asupra obiectelor din schema SYS nu trebuie niciodată efectuate operații de UPDATE, DELETE sau INSERT**, deoarece acestea pot avea efecte distrugătoare asupra bazei de date.



Dicționarul de Date - cont.

- Dicționarul de date este un element esențial pentru fiecare bază de date Oracle, dar este în același timp și un mijloc foarte important care poate fi utilizat de dezvoltatori de aplicații sau administratorul bazei de date pentru a afla informații despre aceasta.

Deci, asupra obiectelor dicționarului de date există **2 moduri de acces**:

- **de către utilizatorii bazei de date:** tabelele sau vederile dicționarului de date pot fi interogate de utilizatorii bazei de date prin comenzi SQL SELECT pentru a afla informații.
Utilizatorii bazei de date nu pot modifica obiectele dicționarului de date, ci le pot doar vizualiza.
- **de către Oracle Server:** acesta modifică dicționarul de date pentru a reflecta schimbările din structura bazei de date.

De asemenea, în timpul operațiilor bazei de date, Serverul Oracle citește informații din dicționarul de date pentru a verifica dacă obiectele bazei de date există și dacă utilizatorii au acces la ele.



Dicționarul de Date - cont.

- Oracle creează sinonime publice pentru multe din vederile dicționarului de date pentru a permite un acces convenabil la acestea utilizatorilor bazei de date.
- Pentru dezvoltatorii de aplicații care se referă la obiectele dicționarului de date, se recomandă ca aceștia să folosească sinonimele publice în loc de obiectele propriu-zise: este mai puțin probabil ca numele sinonimelor să se schimbe de la o versiune la alta.



Dicționarul de Date - cont.

- **Obiectele** dicționarului de date se împart în următoarele **2 categorii**:
 - **tabele de bază**: acestea stochează informații despre baza de date asociată.
În general, aceste tabele sunt folosite doar de Oracle, utilizatorii le accesează direct foarte rar deoarece informațiile conținute în acestea sunt greu de înțeles.
 - **vederi** accesibile utilizatorilor: acestea prezintă informațiile din tabelele de bază într-un format ușor de înțeles pentru utilizatori.
Acestea sunt folosite de utilizatorii bazei de date pentru a accesa informațiile din dicționarul de date.



Dicționarul de Date - cont.

- **Vederile** dicționarului de date au **nume** care reflectă tipul de utilizare pentru care sunt destinate.

Vederile sunt clasificate în 3 grupe care se disting între ele prin prefixele **USER**, **ALL** și **DBA**:

- **USER_xxxxx:**

Acestea conțin informații despre schema utilizatorului curent, incluzând obiectele schemei, privilegiile acordate de către utilizator, etc.

De exemplu, următoarea interogare întoarce numele tuturor tabelelor conținute în schema curentă:

```
SELECT table_name  
FROM user_tables;
```



Dicționarul de Date - cont.

➤ **ALL_XXXXX:**

Acestea conțin informații despre obiectele care pot fi accesate de către utilizatorul curent, adică obiectele din propria schemă plus obiectele pentru care are acordate privilegii de acces.

De exemplu, următoarea interogare întoarce proprietarul și numele tuturor tabelor care pot fi accesate de utilizatorul curent:

```
SELECT owner, table_name  
FROM all_tables;
```




Dicționarul de Date - cont.

➤ **DBA_XXXXX:**

Acestea conțin informații despre toate obiectele bazei de date, deci ele sunt destinate pentru a fi folosite de către administratorul bazei de date; aceste vederi pot fi folosite doar de utilizatori care au acordat privilegiul de sistem SELECT ANY TABLE sau rolul de DBA.

Fiind destinate exclusiv administratorul bazei de date, pentru aceste vederi nu sunt create sinonime, deci numele lor trebuie prefixat cu cel al proprietarului, SYS, de exemplu:

```
SELECT owner, table_name  
FROM dba_tables;
```



Dicționarul de Date - cont.

- Ca regulă generală, vederile al căror nume diferă doar prin prefix (de exemplu USER_TABLES, ALL_TABLES și DBA_TABLES) au coloane identice și prin urmare conțin informații similare, cu excepția faptului că vederile cu prefix USER nu conțin coloana OWNER.
- O listă a vederilor dicționarului de date se găsește la https://docs.oracle.com/cd/B28359_01/nav/catalog_views.htm
- Următorul tabel rezumă principalele vederi ale dicționarului de date care conțin informații despre fișierele de date și structurile logice de stocare, utilizatorii, privilegiile și rolurile bazei de date precum și despre obiectele schemei.
Sunt listate doar vederile cu prefixul DBA, însă în majoritatea cazurilor există și vederi similare cu prefixele USER și ALL.

Fișiere de date, spații tabel, segmente, extinderi	DBA_TABLESPACES, DBA_DATA_FILES, DBA_SEGMENTS, DBA_EXTENTS
Utilizatori, roluri și privilegii	DBA_USERS, DBA_ROLES, DBA_ROLE_PRIVS, DBA_COL_PRIVS, DBA_ROLE_PRIVS, DBA_SYS_PRIVS, DBA_TAB_PRIVS
Toate obiectele bazei de date	DBA_OBJECTS, DBA_OBJECT_SIZE
Tabele	DBA_TABLES
Constrângeri	DBA_CONSTRAINTS
Vederi	DBA_VIEWS
Indecși	DBA_INDEXES, DBA_IND_COLUMNS
Clustere și clustere hash	DBA_CLUSTERS, DBA_CLU_COLUMNS, DBA_CLUSTER_HASH_EXPRESION
Secvențe	DBA_SEQUENCES
Sinonime	DBA_SYNONYMS
Pachete, Proceduri și Funcții	DBA_SOURCE, DBA_ERRORS,
Triggere	DBA_TRIGGERS, DBA_TRIGGER_COLS
Instantanee	DBA_REGISTERED_SNAPSHOTS, DBA_REGISTERED_SNAPSHOT_GROU, DBA_SNAPSHOTS, DBA_SNAPSHOT_LOGS,



	DBA_SNAPSHOT_LOG_FILTER_COLS, DBA_SNAPSHOT_REFRESH_TIMES
Legături ale bazei de date	DBA_DB_LINKS
Tipuri de date	DBA_TYPES, DBA_TYPE_ATTRS, DBA_DEPENDENCIES, DBA_TYPE_METHODS, DBA_OBJECT_TABLES, DBA_METHOD_PARAM, DBA_METHOD_RESULTS



Dicționarul de Date - cont.

- Alături de vederile menționate mai sus, Oracle mai păstrează niște **vederi care conțin informații despre activitatea curentă a bazei de date**.
Acestea au prefixul **V_\$** și sunt numite **tabele de performanță dinamică** (dynamic performance tables).
De exemplu V_\$DATAFILE conține informații despre fișierele de date ale bazei de date. Aceste vederi nu trebuie în general accesate decât de administratorul bazei de date.
Și pentru aceste vederi sunt create sinonime publice, acestea fiind prefixate cu V\$ (deci sinonimul public pentru V_\$DATAFILE se va numi V\$DATAFILE).



Dicționarul de Date - cont.

- În dicționarul de date **există și vederi** al căror nume nu folosesc prefixele menționate mai înainte.

Printre acestea menționăm:

- **DICTIONARY:** Acesta conține toate tabelele, vederile și sinonimele din dicționarul de date.
- **DICT_COLUMNS:** Acesta conține toate coloanele din obiectele dicționarului de date. De exemplu, dacă ne interesează să obținem informații referitoare la vederile ce conțin date despre tabelele bazei de date vom utiliza următoarea comandă:

```
SELECT *  
FROM dictionary  
WHERE table_name LIKE '%TABLE%';
```



Dicționarul de Date - cont.

- **DUAL:** Acesta este un mic tabel, având o singură coloana numită DUMMY de tip VARCHAR2(1) și un singur rând conținând valoarea 'X'.
El este folosit în interogări pentru a returna un rezultat și aceasta datorită faptului că într-o instrucțiune SELECT trebuie specificată neapărat și clauza FROM.
Tabelul DUAL este pur și simplu un tabel standard Oracle care este utilizat ca un tabel de test.
De exemplu, pentru a afișa data curentă se folosește următoarea interogare:

```
SELECT sysdate FROM dual;
```
- Și pentru aceste vederi sunt create sinonime publice.
De exemplu, vederea DICTIONARY are sinonimul public DICT.



Dicționarul de Date - cont.

- Pentru obiectele unei baze de date există posibilitatea de a face anumite **comentarii** asupra lor prin inserarea unui text în dicționarul de date.

Comentariul este creat prin comanda:

```
COMMENT ON {TABLE nume_obiect | COLUMN nume_obiect.nume_coloana}  
nume_obiect IS 'text comentariu';
```

unde nume_obiect reprezintă numele unui tabel, vederi sau instantaneu.

Comentariul se poate referi la tabele, vederi, instantanee sau coloane.



Bibliografie

F. Ipate, M. Popescu, Dezvoltarea aplicațiilor de baze de date în Oracle 8 și Oracle Forms 6, Editura ALL, 2000.