

## CAPITOLUL 4. CREAREA UNEI BAZE DE DATE PRIN COMENZI SQL

### 4.1. TIPURI DE UTILIZATORI AI BAZELOR DE DATE ORACLE

În funcție de volumul activităților implicate de administrarea unei baze de date Oracle, sarcinile de administrare pot fi repartizate pe mai multe categorii de utilizatori ai bazei de date.

*Administratorul bazei de date (DBA)* este, în funcție de complexitatea și mărimea unei baze de date, o persoană sau mai multe persoane, care să execute următoarele sarcini administrative:

- Instalarea și dezvoltarea sever-ului Oracle;
- Alocarea memoriei sistemului și planificarea cerințelor viitoare de memorie ale acestuia;
- Crearea bazei de date și a obiectelor acesteia (tabele, viziuni, indecși);
- Modificarea structurii bazei de date în funcție de cerințele dezvoltatorilor de aplicații;
- Definirea utilizatorilor bazei de date și întreținerea sistemului de securitate;
- Controlul și monitorizarea accesului utilizatorilor la baza de date;
- Monitorizarea și optimizarea performanțelor bazei de date;
- Definirea și asigurarea politicii de salvarea sau copiere (backup) și refacere (recovery) a bazei de date;
- Arhivarea datelor;
- Asigurarea legăturii cu firma Oracle pentru suportul tehnic și licența de utilizare a produselor Oracle.

*Dezvoltatorii de aplicații* proiectează și implementează aplicații cu baze de date Oracle, executând următoarele sarcini:

- Proiectarea și dezvoltarea unei aplicații, precum și a structurilor de date ale acesteia;
- Estimarea cerințelor de memorie pentru aplicație;
- Definirea modificărilor structurilor de date ale unei aplicații;
- Transmiterea tuturor informațiilor despre activitățile de mai sus către administratorul bazei de date;
- Stabilirea măsurilor de securitate pentru aplicație.

*Administratorul de aplicații* se ocupă cu administrarea unei aplicații;

*Utilizatorii finali ai bazei de date* au acces la baza de date prin intermediul unei aplicații sau a instrumentelor Oracle, executând în special următoarele activități:

- Adăugarea, modificarea și ștergerea datelor din baza de date în concordanță cu drepturile de acces pe care le are;
- Generarea unor rapoarte cu datele din baza de date.

*Administratorul de rețea* este responsabil cu administrarea produselor Oracle de rețea.

Pentru a putea executa sarcinile de administrare a unei baze de date o persoană trebuie să aibă privilegii (drepturi) atât la nivelul bazei de date Oracle, cât și la nivelul sistemului de operare al serverului pe care se află baza de date. Prin urmare un administrator trebuie să aibă:

- *Cont de administrator pentru sistemul de operare*, care să-i permită să execute comenzile sistemului de operare;
- *Cont de administrator Oracle* definit de două conturi de utilizator (SYS și SYSTEM cu parolele CHANGE\_OF\_INSTALL și respectiv MANAGER);
- *Rol de DBA*, care este creat automat la momentul creării unei baze de date Oracle. Acest rol conține toate privilegiile bazei de date Oracle.

Datorită faptului că un administrator execută activități pe care un utilizator obișnuit nu le poate executa este necesar ca acesta să poată fi autentificat înainte de a executa activitățile de administrare. Pentru autentificarea administratorului există două metode: autentificarea folosind sistemul de operare și autentificarea folosind fișierul de parole.

Autentificarea folosind *sistemul de operare* se face utilizând două conturi de administrator: *OSOPER* (sub care poate executa STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVELOG și RECOVER) și contul *OSDBA* (care conține toate privilegiile de sistem cu opțiunea ADMIN OPTION, precum și rolul OSOPER). Sub contul OSDBA se poate executa comanda CREATE DATABASE.

Autentificarea folosind *fișierul de parole* permite definirea parolelor de acces pentru fiecare utilizator. După stabilirea unui utilizator ca administrator, de exemplu utilizatorul SCOTT cu parola TIGER, acestuia îi va fi atribuit unul din privilegiile SYSDBA sau SYSOPER, cu comanda GRANT. După aceasta utilizatorul SCOTT se va conecta la baza de date ca SYSDBA sau SYSOPER cu comanda CONNECT.

**Exemplu:**

***GRANT SYSDBA TO scott***

**GRANT SYSOPER TO scott**  
**CONNECT scott/tiger AS SYSDBA**  
**CONNECT scott/tiger AS SYSOPER**

Administrarea fișierului cu parole include operațiile de *definire* a fișierului cu parole, *setarea* parametrului de inițializare a bazei de date `REMOTE_LOGIN_PASSWORDFILE`, *adăugarea de utilizatori* în acest fișier și *întreținerea* fișierului cu parole.

*Crearea fișierului cu parole* se execută cu utilitarul `ORAPWD`, care are trei parametri: `FILE`, `PASSWORD` și `ENTRIES`, dintre care primii doi sunt obligatorii, iar ultimul este opțional. Acești parametri definesc *numele fișierului cu parole*, *parola* pentru utilizatorul `SYS` și respectiv *numărul de utilizatori* care pot executa activități de administrator (DBA). *Setarea parametrului de inițializare* `REMOTE_LOGIN_PASSWORDFILE` cu una din valorile `NONE`, `EXCLUSIVE` și `SHARED` permite utilizarea fișierului cu parole. Valoarea `NONE` determină ca baza de date Oracle să funcționeze fără fișier de parole, valoarea `EXCLUSIVE` determină ca fișierul de parole să fie folosit exclusiv de către o singură bază de date, iar valoarea `SHARED` determină ca fișierul de parole să fie folosit de către mai multe baze de date.

Pentru a avea un grad mare de securitate pentru baza de date, va trebui ca imediat după crearea fișierului cu parole parametrul de inițializare `REMOTE_LOGIN_PASSWORDFILE` să fie setat pe valoarea `EXCLUSIVE`.

Adăugarea de utilizatori în fișierul cu parole se face la momentul atribuirii privilegiilor `SYSDBA` sau `SYSOPER` unui anumit utilizator.

**Exemplu:**

1. Se creează fișierul cu parole conform indicațiilor de mai sus;
2. Se setează parametrul de inițializare `REMOTE_LOGIN_PASSWORDFILE` cu valoarea `EXCLUSIVE`;
3. Se conectează utilizatorul `SYS`, cu parola `CHANGE_OF_INSTALL` ca `SYSDBA` folosind comanda  
**CONNECT SYS/change\_of\_install AS SYSDBA**
4. Se pornește o instanță și se creează o bază de date, dacă este necesar, sau se montează și se deschide o bază de date existentă;
5. Se creează utilizatorii care se doresc a fi administratori și care să fie adăugați în fișierul cu parole, folosind comanda  
**CREATE USER user1 IDENTIFIED BY parola1**
6. Se atribuie unul din privilegiile `SYSDBA` sau `SYSOPER` acestui utilizator cu una din comenzile:  
**GRANT SYSDBA TO user1** sau **GRANT SYSOPER TO user1**

7. Utilizatorul *USER1* este adăugat în fișierul cu parole și se poate conecta acum ca *SYSDBA* sau *SYSOPER* cu acest nume de utilizator în loc de numele *SYS*, folosind una din comenzile:

**CONNECT USER1/parola1 AS SYSDBA** sau  
**CONNECT USER1/parola1 AS SYSOPER**

Listarea membrilor fișierului cu parole se face din viziunea \$PWFIL\_Users folosind comanda

**SELECT \* FROM V\$PWFIL\_Users**

Întreținerea fișierului cu parole se referă la executarea activităților de extindere, relocare, ștergere sau schimbare a stării acestui fișier.

#### **4.2. CREAREA, PORNIREA ȘI OPRIREA UNEI BAZE DE DATE ORACLE**

Configurarea serverului Oracle presupune următoarele activități:

- Instalarea sistemului Oracle, care constă în instalarea nucleului Oracle pe server, a instrumentelor (tools-urilor) de aplicație pe stații;
- Evaluarea resurselor fizice ale calculatorului pe care se va instala serverul Oracle și baza de date;
- Definirea structurii logice a bazei de date și a strategiei de salvare (backup);
- Crearea și deschiderea bazei de date;
- Implementarea bazei de date proiectate, prin definirea segmentelor de revenire (rollback), a tabelor spațiu și a obiectelor bazei de date;
- Salvarea bazei de date și definirea utilizatorilor bazei de date, în concordanță cu licența Oracle.

Pentru a crea o bază de date Oracle trebuie să avem suficientă memorie pentru pornirea unei instanțe Oracle și pentru crearea tuturor obiectelor proiectate ale bazei de date. Dacă la momentul instalării s-a creat și o bază de date inițială atunci aceasta poate fi dezvoltată astfel încât să cuprindă, în final, toate obiectele bazei de date proiectate. De asemenea această bază de date inițială poate fi ștearsă și în locul ei să se creeze o nouă bază de date. Dacă am folosit o versiune anterioară Oracle se poate crea o bază de date nouă în întregime, dacă nu ne mai interesează vechea bază de date, altfel putem migra această bază de date la noua versiune Oracle.

Crearea unei baze de date se face în următorii pași:

- Salvarea completă a bazei de date existente;
- Crearea fișierului cu parametrii folosit la pornirea bazei de date.
- Editarea noului fișier cu parametrii, astfel încât parametrii acestuia să corespundă cerințelor noii baze de date.

- Controlul identicatorului instanței Oracle, care trebuie să fie identic cu numele bazei de date setat în parametrul DB\_NAME;
- Pornirea utilitarului Enterprise Manager și conectarea la Oracle ca administrator.
- Pornirea unei *instanțe Oracle* (System Global Area și procesele background) cu opțiunea Startup Nomount.
- Crearea noii bazei de date folosind comanda SQL *CREATE DATABASE*, prin intermediul căreia Oracle execută: crearea fișierelor de date (data files), fișierelor de control (control files) și a fișierelor de refacere (redo log) ale bazei de date; crearea tabelii spațiu SYSTEM și a segmentului rollback SYSTEM; crearea dicționarului de date; crearea utilizatorilor SYS și SYSTEM; specifică setul de caractere care va fi folosit la memorarea datelor în baza de date; montează și deschide baza de date pentru utilizare.
- Salvarea integrală a bazei de date.

Parametrii de inițializare a bazei de date furnizează valorile necesare pentru funcționarea acestora sub o anumită instanță Oracle. Aceștia se personalizează prin intermediul unui fișier text, numit fișierul cu parametrii de inițializare. Acesta este citit la momentul pornirii bazei de date de către serverul Oracle. Pentru a face eventuale modificări, baza de date trebuie oprită complet și repornită după ce s-au efectuat astfel de modificări.

Mulți parametrii de inițializare se folosesc pentru ajustarea și creșterea performanțelor bazei de date. Specificarea parametrilor se realizează după următoarele reguli:

- Toți parametrii sunt opționali;
- În fișierul cu parametrii se vor fi introduce numai parametrii și comentarii;
- Semnul (#) marchează începutul unui comentariu, restul liniei fiind ignorat;
- Serverul Oracle are valori asumate pentru fiecare parametru;
- Parametrii pot fi specificați în orice ordine;
- Fișierul de parametrii nu este „case-sensitive” ;
- Pentru a introduce mai mulți parametrii pe o linie aceștia se vor separa prin spațiu:

PROCESSES = 100 SAVEPOINTS = 5 OPEN\_CURSORS = 10

- Unii parametrii, ca de exemplu ROLLBACK\_SEGMENTS, acceptă valori multiple, acestea putându-se specifica între paranteze și separate prin virgulă, sau fără paranteze și virgule, ambele sintaxe fiind valide, astfel:

**ROLLBACK\_SEGMENTS = (SEG1, SEG2, SEG3, SEG4, SEG5)**

**ROLLBACK\_SEGMENTS = SEG1 SEG2 SEG3 SEG4 SEG5**

- Caracterul (/) folosește pentru marcarea întreruperii scrierii unui parametru pe o linie și continuarea lui pe linia următoare, imediat fără nici un spațiu în față, astfel:

**ROLLBACK\_SEGMENTS = (SEG1, SEG2, \**  
**SEG3, SEG4, SEG5)**

- Parametrul IFILE se poate introduce într-un fișier cu parametrii alt fișier cu parametrii, astfel:

**IFILE = COMMON1.ORA**

Se pot folosi trei niveluri de imbricare. În exemplu de mai sus fișierul COMMON1.ORA poate conține un al doilea parametru IFILE pentru fișierul COMMON2.ORA, care la rândul său poate conține un al treilea parametru IFILE pentru fișierul COMMON3.ORA. De asemenea, se pot utiliza mai mulți parametrii IFILE în același fișier, astfel:

**IFILE = DBPARMS.ORA**

**IFILE = GCPARMS.ORA**

**IFILE = LOGPARMS.ORA**

Dacă valoarea unui parametru conține caractere speciale, atunci caracterul special trebuie precedat de caracterul de comutare (\) sau întreaga valoare este inclusă între caracterele (" "), ca în exemplul de mai jos:

**DB\_DOMAIN = JAPAN.ACME#.COM**

sau

**DB\_DOMAIN = "JAPAN.ACME#.COM"**

Pentru schimbarea valorii unui parametru, aceasta se editează în fișierul cu parametrii. Când instanța Oracle este repornită aceasta va folosi noua valoare a parametrului.

Câțiva parametrii de inițializare sunt dinamici, în sensul că valorile acestora se pot modifica prin procedeul de mai sus sau în timp ce o instanță rulează, folosind comenzile SQL: ALTER SESSION, ALTER SYSTEM sau ALTER SYSTEM DEFERRED cu sintaxele:

**ALTER SESSION SET nume\_parametru = valoare**

**ALTER SYSTEM SET nume\_parametru = valoare**

**ALTER SYSTEM SET nume\_parametru = valoare DEFERRED**

Comanda ALTER SESSION schimbă valoarea unui parametru numai la nivelul sesiunii care a lansat-o, după repornirea bazei de date se va utiliza iarăși valoarea din fișierul cu parametrii.

Comanda ALTER SYSTEM modifică valoarea globală a parametrului, la nivelul întregului sistem, deci pentru toate sesiunile active, după repornirea bazei de date se va utiliza iarăși valoarea din fișierul cu parametrii.

Comanda ALTER SYSTEM DEFERRED modifică valoarea globală a parametrului nu pentru sesiunile active, ci pentru sesiunile viitoare, care vor fi active după repornirea bazei de date.

Afișarea valorilor curente ale parametrilor de inițializare ai bazei de date se face cu comanda SHOW PARAMETERS, afișarea făcându-se în ordinea alfabetică a parametrilor. Listarea la imprimantă a parametrilor afișați se face cu comanda SPOOL.

Parametrii de inițializare se pot grupa în:

- *Parametrii derivați* sunt cei ale căror valori se calculează pornind de la valorile altor parametri. Normal valorile acestora nu trebuie modificate;
- *Parametrii globali prefixați cu GC* sunt folosiți pe sistemele care suportă Oracle Parallel Server;
- *Parametrii dependenți de sistemul de operare* sunt cei ale căror valori sunt dependente de specificul sistemului de operare gazdă. Exemplu: DB\_BLOCK\_BUFFERS care indică numărul ariilor de date (data buffers) din memoria principală sau DB\_BLOCK\_SIZE care indică mărimea unui bloc de date;
- *Parametrii de tip variabilă* sunt cei ce pot lua anumite valori care să determine performanțele sistemului sau anumite limite de funcționare. Exemplu: OPEN\_CURSORS dacă i se dă valoarea 10 se vor putea deschide maxim 10 cursoare, iar DB\_BLOCK\_BUFFERS prin valorile pe care le va lua nu va impune anumite limite dar va modifica performanțele sistemului;
- *Parametrii statici* sunt cei ale căror valori nu se pot modifica în timpul unei sesiuni sau cu baza de date pornită;
- *Parametrii dinamici* sunt cei ale căror valori se pot modifica, așa cum s-a arătat mai sus cu comenzile ALTER SESSION, ALTER SYSTEM sau ALTER SYSTEM DEFERRED;

Așa cum s-a prezentat mai sus compania Oracle furnizează un fișier inițial cu parametrii cu ajutorul căruia putem să pornim o bază de date. Pentru a personaliza baza de date conform cerințelor beneficiarului, administratorul va trebui să seteze valori noi pentru anumiți parametri specificați mai jos, astfel:

- **DB\_NAME** definește numele bazei de date. Se formează dintr-un șir de maximum opt caractere. Dacă nu se furnizează, Oracle atribuie bazei de date un nume standard. Acesta se găsește în fișierul cu parametrii furnizat o dată cu software-ul Oracle. În timpul creerii bazei de date numele acesteia este scris în fișierele de date, de control și de log.

**Exemplu: DB\_NAME = BAZAI**

- **DB\_DOMAIN** este format dintr-un șir de caractere și definește domeniul din rețea căruia aparține baza de date, de obicei este definit de numele organizației căreia aparține baza de date. Dacă se utilizează numele domeniului din Internet, atunci partea adresei de e-mail care urmează după caracterul ”@” este foarte bună pentru a fi folosită ca valoare pentru acest parametru.

**Exemplu:** *DB\_DOMAIN = BUC.ORG.COM*

- **GLOBAL\_NAMES** definește numele global al bazei de date în cadrul rețelei de calculatoare și este format din numele bazei de date și numele domeniului separate prin punct. Acest parametru mai poate lua și valorile: TRUE (caz în care se forțează ca numele global al bazei de date să fie identic cu cel al bazei de date) sau FALSE (numele global nu are semnificație, nu se utilizează).

**Exemplu:** *GLOBAL\_NAMES = FALSE*

- **CONTROL\_FILES** definește numele fișierelor de control ce vor fi create pentru baza de date (se va furniza pentru fiecare fișier calea completă de acces la acesta). Este recomandat ca să se definească cel puțin două fișiere de control, care să fie plasate pe două discuri diferite.

Exemplu: *CONTROL\_FILES = diska:cntrl1.ora, diskb:cntrl2.ora*

- **LOG\_FILES** specifică numărul maxim de grupuri de fișiere de log ce pot fi utilizate pentru o bază de date. Ia valori de la 2 la 255. Acest parametru specifică totodată și numărul minim de fișiere de log ce pot fi deschise pentru o bază de date.
- **DB\_FILE\_MULTIBLOCK\_READ\_COUNT** definește *numărul de blocuri* citite simultan pentru accesarea unei tabele a bazei de date. Este folosit pentru optimizarea parcurgerii totale a unei tabele atunci când se caută o anumită valoare a unei coloane aferentă unui rând din aceasta.
- **REMOTE\_LOGIN\_PASSWORDFILE** specifică dacă se folosește sau nu fișierul cu parole pentru identificarea utilizatorilor ce pot executa activități de administrator. Poate lua valorile: NONE nu se folosește fișierul de parole iar utilizatorul care va executa activități de administrare trebuie să fie autentificat de către sistemul de operare gazdă; EXCLUSIVE se folosește un singur fișier cu parole pentru o singură bază de date. Pe lângă utilizatorii SYS și SYSTEM și alți utilizatori pot executa sarcini de administrare; SHARED se folosește un singur fișier cu parole pentru mai multe baze de date, caz în care singurii utilizatori ce pot executa activități de administrare sunt SYS și SYSTEM. În acest caz nu se pot adăuga alți utilizatori în fișierul cu parole.



- **DB\_FILES** specifică numărul maxim de fișiere de date ce pot fi deschise de către o bază de date. De fiecare dată când se modifică acest parametru baza de date se oprește și apoi se repornește.
- **LOG\_CHECKPOINT\_INTERVAL** specifică frecvența punctelor de control (checkpoints) ;
- **LOG\_CHECKPOINT\_TIMEOUT** specifică timpul maxim dintre două puncte de control în secunde;
- **PROCESSES** specifică numărul maxim de utilizatori care se pot conecta simultan la baza de date, iar acest număr trebuie să fie mai mare decât numărul total de procese background, Job Queue și Parallel Query;
- **ROLLBACK\_SEGMENTS** definește toate segmentele rollback pe care o instanță le poate acapara la momentul pornirii. Valoarea acestui parametru se dă sub forma unei liste de valori.

**Exemplu:**

**ROLLBACK\_SEGMENTS** = (rbseg1, rbseg2, rbseg3, rbseg4)

- **LICENSE\_MAX\_SESSIONS** specifică numărul maxim de utilizatori concurențiali ce pot fi admiși în timpul unei sesiuni;
- **LICENSE\_MAX\_USERS** specifică numărul maxim de utilizatori ce pot fi creați pentru o bază de date. **LICENSE\_MAX\_SESSIONS** și **LICENSE\_MAX\_USERS** nu pot avea simultan valori diferite de zero, deci unul din acești parametri trebuie să fie setat pe zero;
- **LICENSE\_SESSIONS\_WARNING** specifică numărul de utilizatori concurențiali. Dacă se depășește, se emite un mesaj de atenționare;
- **TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT** specifică numărul tranzacțiilor concurente permise pe un segment rollback;
- **AUDIT\_TRAIL** activează sau dezactivează scrierea rândurilor în fișierul de audit. Înregistrările de audit nu se scriu dacă parametrul are valoarea NONE sau lipsește.

**Zona de memorie SGA** (System Global Area) conține următoarele structuri de memorie: *database buffer cache*, *redo log buffer* și *shared pool*. *Database Buffer Cache* este porțiunea din SGA ce conține blocurile de date citite din fișierele de date ale bazei de date. *Redo log buffer* este zona în care se păstrează informații despre modificările efectuate în baza de date. *Shared pool* este o zonă care conține la rândul său trei structuri de memorie: *library cache*, *dictionary cache* și *control structures*. Dicționarul de date al bazei de date este citit în zonele *library cache* și *dictionary cache*.

Mărimea SGA este influențată de valorile parametrilor următori:

- **DB\_BLOCK\_SIZE** definește, în bytes, mărimea unui singur bloc de date. Valorile tipice pentru acest parametru sunt 2048 și 4096;

- **DB\_BLOCK\_BUFFERS** specifică numărul de buffere ale bazei de date disponibile în zona *database buffer cache*, a cărei mărime este egală cu **DB\_BLOCK\_SIZE \* DB\_BLOCK\_BUFFERS**;
- **LOG\_BUFFER** determină mărimea în bytes a zonei *redo log buffer*
- **SHARED\_POOL\_SIZE** specifică mărimea în baiți a ariei *Shared pool*. Acest parametru poate accepta valori numerice urmate de literele "K" sau "M", unde "K" înseamnă că numărul va fi multiplicat cu 1000, iar "M" înseamnă că numărul va fi multiplicat cu 1000000.
- **OPEN\_CURSORS** specifică numărul maxim de cursoare pe care o sesiune le poate deschide simultan. Valoarea asumată este de 50. Este bine ca acest număr să fie cât mai mare pentru a nu avea probleme cu rularea unei aplicații.
- **TRANSACTIONS** specifică numărul maxim de tranzacții concurente. O valoare mare a acestui parametru determină mărirea zonei de memorie SGA.

### Exemple de creare a unei baze de date

#### 1) **CREATE DATABASE**;

În acest caz se crează o bază de date Oracle în care toți parametrii comenzii **CREATE DATABASE** iau valorile standard furnizate de firma Oracle.

#### 2) **CREATE DATABASE baza1**

```
LOGFILE GROUP 1 ('diskb:log1.log', 'diskc:log1.log') SIZE 50K,
GROUP 2 ('diskb:log2.log', 'diskc:log2.log') SIZE 50K
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
ARCHIVELOG
CHARACTER SET US7ASCII
DATAFILE 'diska:datfile1.dat' SIZE 2M
DATAFILE
'disk1:datfile2.dbf' AUTOEXTEND ON
'disk2:datfile3.dbf' AUTOEXTEND ON NEXT 10M
MAXSIZE UNLIMITED;
```

După crearea unei baze de date, instanța Oracle poate fi lăsată să ruleze, iar baza de date este deschisă și montată pentru utilizare normală. Pentru opririle și pornirile ulterioare se poate utiliza *Oracle Enterprise Manager*.

### Pornirea și oprirea bazei de date

O bază de date și instanța Oracle se poate porni cu utilitarul *Oracle Enterprise Manager* folosind fereastra de dialog Startup Database. O instanță și o bază de date asociată se pot porni în mai multe moduri.

Pornirea instanței *fără montarea* bazei de date se face atunci când dorim să creăm o bază de date. Activitatea se execută din fereastra de dialog Startup Database prin selectarea butonului radio Startup Nomount;

Pornirea instanței și *montarea bazei de date*, aceasta rămânând *închisă* se execută atunci când dorim să executăm anumite activități de întreținere: redenumirea fișierelor de date; adăugarea, ștergerea sau redenumirea fișierelor redo log; recuperarea integrală a bazei de date; Această pornire se execută din fereastra de dialog Startup Database prin selectarea butonului radio Startup Mount. Montarea bazei de date se poate execută și după pornirea unei instanțe fără bază de date montată, cu ajutorul comenzii SQL *ALTER DATABASE* cu opțiunea *MOUNT*.

**Exemplu: SQL> ALTER DATABASE baza1 MOUNT;**

Pornirea instanței, *montarea bazei de date* și *deschiderea acesteia* se face în mod *nerestricționat* (accesibilă tuturor utilizatorilor care au cu privilegiul CREATE SESSION) sau *restricționat* (accesibilă doar utilizatorilor de tip DBA, utilizatorilor cu privilegiile CREATE SESSION și RESTRICTED SESSION).

În modul de pornire restricționat se pot executa activități ca: recrearea indecșilor; exportul sau importul datelor bazei de date; încărcarea datelor cu utilitarul SQL\*Loader; blocarea temporară a accesului utilizatorilor obișnuiți la baza de date. *Pornirea în mod nerestricționat* se face din fereastra de dialog Startup Database prin selectarea butonului radio Startup Open. *Pornirea în mod restricționat* se face din fereastra de dialog Startup Database prin selectarea butonului radio Restrict

Deschiderea unei baze de date se poate face după ce instanță a fost pornită, baza de date montată dar închisă, cu ajutorul comenzii SQL *ALTER DATABASE* cu opțiunea *OPEN*.

**Exemplu: SQL> ALTER DATABASE baza1 OPEN;**

Transformarea modului de pornire normală a unei baze de date în modul restricționat se poate face și cu comanda SQL *ALTER SYSTEM* cu opțiunea *ENABLE RESTRICTED SESSION*, iar revenirea la situația inițială se face cu aceeași comandă dar cu opțiunea *DISABLE RESTRICTED SESSION*

**Exemplu:**

**SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;**

**SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;**

În anumite circumstanțe este posibil ca activitățile de pornire a bazei de date și instanței Oracle să se execute altfel decât în mod uzual. Astfel putem avea:

- *pornirea forțată a unei instanțe*, care se poate realiza atunci când instanța curentă nu poate fi oprită cu succes prin folosirea butoanelor radio Normal sau Immediate din fereastra de dialog Startup. În acest caz se poate forța pornirea unei noi instanțe Oracle, care va determina oprirea instanței anterioare aflată în situația de mai sus.
- *pornirea unei instanțe, montarea bazei de date și pornirea procesului de recuperare a bazei de date, a tabelelor spațiu sau a fișierelor de date*, care se execută atunci când știm că mediul bazei de date are nevoie de recuperare.
- *pornirea în modul exclusiv sau paralel*, care se face atunci când avem un server Oracle care permite accesul mai multor instanțe la aceeași bază de date.
- *pornirea automată a bazei de date la momentul pornirii sistemului de operare*, se face dacă dorim acest lucru.
- *pornirea unei instanțe și a unei baze de date la distanță*, se face atunci când serverul Oracle este o parte a unui sistem de baze de date distribuite.

**Oprirea unei baze de date** se poate face în două moduri: normal sau forțat.

*Modul normal*, în care oprirea bazei de date se face ca revers al operației de pornire normală, sens în care se execută închiderea bazei de date, demontarea bazei de date și oprirea instanței Oracle. Activitatea se execută din fereastra de dialog Shutdown Database prin selectarea butonului radio Normal. Oprirea unei baze de date în condiții normale presupune executarea de către Oracle a următoarelor activități: oprirea conexiunilor la baza de date; deconectarea tuturor utilizatorilor; la următoarea pornire a bazei de date nu se pornesc procedurile de recuperare.

*Modul forțată* se poate execută în două moduri: imediat sau prin anularea instanței.

**Oprirea imediată a bazei de date** se execută în cazul unui incident iminent. În cadrul acestei opriri Oracle execută instrucțiunea SQL aflată în lucru și orice altă tranzacție nefinalizată este anulată prin procesul de rollback; toți utilizatorii conectați sunt deconectați imediat.

Oprirea se face din fereastra de dialog Shutdown Database prin selectarea butonului Immediate.

**Oprirea prin anularea instanței** se execută dacă baza de date sau una din aplicațiile sale funcționează anormal și nici una din metodele de oprire anterioare nu funcționează. Această oprire se execută din fereastra de dialog Shutdown prin setarea butonului radio Abort.

În timpul acestei opriri Oracle finalizează instrucțiunile SQL aflate în lucru, tranzacțiile nefinalizate nu mai sunt aduse la starea anterioară momentului începerii acestora (nu mai sunt anulate prin procesul de roll back), iar toți utilizatorii sunt deconectați imediat.

### 4.3. CREAREA ȘI ACTUALIZAREA TABELELOR

Comanda de creare a unei tabele, **CREATE TABLE**, realizează fie definirea unei tabele, urmând ca introducerea de date să se efectueze ulterior, fie definirea și încărcarea cu date a unei tabele, chiar în momentul creerii ei, folosind sintaxele:

**CREATE TABLE** nume-tabelă  
 (spec-col [NOT NULL],...)  
 SPACE definire-spațiu [PCTFREE n] |  
 CLUSTER nume-cluster (nume-col,...)];

**CREATE TABLE** nume-tabelă  
 [(nume-col [NOT NULL],...)]  
 [SPACE definire-spatiu [PCTFREE n]  
 CLUSTER nume-cluster (nume-col,...)]  
 [AS cerere]

Unde:

**spec-col** cuprinde **nume-col**, format, mărime.

**nume-col** este numele coloanei din tabelă.

**format** reprezintă formatul coloanei din tabelă. Formatele acceptate sunt specificate în tabelul următor:

Tipul datelor	Formatul	Explicații
Char	CHAR( mărime)	Date alfanumerice. Marimea maximă a coloanei este de 240 caractere.
Date	DATE	Permit introducerea câmpurilor de tip dată. Exemplu: January 1, 4712 BC to December 31, 4712 AD

Long	LONG	Date caracter de mărime variabilă, într-o tabelă trebuie să fie definită doar o coloană de tip LONG.
	LONG VARCHAR	La fel ca LONG.
	LONG ROW	Date binare RAW.
Numerice	NUMBER	Date numerice de mărime implicită.
	NUMBER (mărime)	Date numerice de mărime specificată.
	NUMBER (întregi zecimali)	Date numerice în baza zece.
	NUMBER(*)	La fel ca datele de tip NUMBER.
	DECIMAL	La fel ca datele de tip NUMBER.
	FLOAT	Nu acceptă descrierea mărimii coloanei.
	INTEGER	La fel ca datele de tip NUMBER.
	INTEGER (mărime)	Nu acceptă descrierea mărimii zecimalelor.
	SMALLINT	La fel ca date de tip INTEGER
RAW	RAW(mărimea)	Date binare RAW. Mărimea maximă este 240 octeți.
	LONGRAW	Date de tip LONG.

În cazul celei de-a doua sintaxe, tipul și mărimea coloanelor vor fi copiate din rezultatul obținut în urma specificării **AS** cerere. În ambele sintaxe se utilizează alte cuvinte cheie sau parametrii, care au următoarea semnificație:

**NULL** sau **NOT NULL** se referă la câmpurile din coloane care pot avea sau nu valori nule. Clauzele nume-col [**NOT NULL**] se pot repeta pentru maximum 254 de coloane.

**SPACES** desemnează modelul de alocare a spațiului pentru tabela ce se crează. Modelul este creat în prealabil cu **CREATE SPACES**. Dacă parametrul **SPACE** este omis, pentru toți parametrii de spațiu vor fi utilizate valorile implicite din comanda **CREATE SPACE**.

**PCTFREE** permite specificarea unei alte valori decât cea implicită de 20% pentru spațiul lăsat liber. Are aceeași funcție ca și în comanda **CREATE SPACE**.

**CLUSTER** este parametrul care indică faptul că tabela va fi inclusă în clusterul cu numele nume-cluster. Introducerea datelor în tabela va declanșa memorarea lor în ordinea indicelui de cluster.

**Nume-col** se referă la numele coloanelor din tabelă care corespund coloanelor din cluster.

**Exemplu:**

Să se creeze tabelele: PRODUSE, CLIENȚI, PRETURI SALARIATI, DEPOZITE, COMENZI ce intră în componența bazei de date COMBAZA.

```
SQL> CREATE TABLE PRODUSE  
2 (Codd NUMBER(6) NOT NULL,  
3 Codd NUMBER(5) NOT NULL,  
4 DENP CHAR(11),  
5 STOC NUMBER(6),  
6 DATACRT DATE,  
7 UM CHAR(3));
```

*Table created.*

```
SQL> CREATE TABLE CLIENȚI  
2 (CODC NUMBER(6) NOT NULL,  
3 DENC CHAR(11),  
4 LOC CHAR(11),  
5 STR CHAR(16),  
6 NRCHARC3),  
7 CONTNUMBER(11),  
8 TEL NUMBER(8),  
9 TFAX NUMBER(8))
```

*Table created.*

```
SQL> CREATE TABLE PRETURI  
2 Codd NUMBER(5) NOT NULL,  
3 PRETMAX NUMBEB(9,2),  
4 PRETMIN NUMBER(9,2), 6 DATAI DATE,  
6 DATASF DATE);
```

*Table created.*

```
SQL> CREATE TABLE SALARIATI  
2 (MARCA NUMBER(4) NOT NULL,  
3 NUME CHAR(16),  
4 FUNCT CHAR(7),  
6 Codd NUMBER(G) NOT NULL, 6 SALA NUMBER(9,2)  
7 VENS NUMBER(9.2),  
8 CODS NUMBER(4) NOT NULL);
```

*Table created.*

```
SQL> CREATE TABLE DEPOZITE  
2 (Codd NUMBER(6) NOT NULL,  
3 DEND NUMBER(6) NOT NULL,  
4 NRSAL NUMBER(2));
```

*Table created.*

```
SQL> CREATE TABLE COMENZI  
2 (NRCOM NUMBER(6) NOT NULL,
```

**3 CODP NUMBER(5) NOT NULL,**  
**4 CODC NUMBEIK6) NOT NULL,**  
**5 DATAL DATE,**  
**6 CANT NUMBER(7),**  
**7 PREȚ NUMBER(9-2));**  
**Table created.**

Modificarea unei tabele presupune adăugarea de noi coloane la sfârșitul acesteia sau modificarea tipurilor unor coloane deja existente. Comanda care realizează aceste operații este ALTER TABLE. Sintaxa ei este:

**ALTER TABLE table {ADD | MODIFY}**  
**(spec-col [NULL | NOT NULL,...])**

unde:

**ADD** adaugă la sfârșitul tabelii noi coloane care inițial au valori nule; ulterior, prin actualizare, aceste coloane se vor completa cu date.

**MODIFY** schimbă definirea unei coloane existente. Modificarea tipului sau mărimii unei coloane presupune ca aceasta să conțină numai valori nule.

spec-col reprezintă numele, formatul și mărimea coloanei ce se va adăuga sau se va modifica.

**NULL** se referă la faptul că valorile din coloană ce se adaugă/modifică sunt nule.

**NOT NULL** specifică faptul că o coloană nu poate avea valori nule (**NOT NULL**). Unei coloane deja existente i se poate atașa parametrul **NOT NULL** numai dacă aceasta conține valori nenule.

### **Exemple:**

1) Să se adauge la tabela DEPOZITE o nouă coloană CANTDEP ce nu a fost prevăzută inițial.

**SQL> ALTER TABLE DEPOZITE ADD**  
**2 (CANTDEP NUMBER(7));**

**Table altered.**

2) Să se modifice atributele coloanei CANTDEP (respectiv mărirea acesteia cu două unități).

**SQL> ALTER TABLE DEPOZITE MODIFY**  
**2 (CANTDEP NUMBER(9));**

**Table altered.**



Pentru ștergerea unei tabele într-o bază de date se utilizează comanda **DROP TABLE**. Sintaxa ei este:

### **DROP TABLE nume-tabelă:**

În general, nu se pot șterge tabelele create de alți utilizatori. La ștergerea unei tabele, se șterg automat indecșii corespunzători (create fie de către proprietarul tablei, fie de către alți utilizatori) și privilegiile conferite în legătură cu ea. Rămân, însă, viziunile și sinonimele referitoare la tabela ștersă, care vor deveni invalide. Va trebui fie ca acestea să fie șterse, fie să se definească sau să se redefinăscă tabela în așa fel încât viziunile și sinonimele să devină valide.

#### ***Exemplu:***

Să se steargă tabela **DEPOZITE**.

**SQL> DROP TABLE DEPOZITE;**

**Table dropped.**

Se pot modifica numele atribuite tabelelor, sau sinonimelor utilizând comanda **RENAME**. Sintaxa ei este:

### **RENAME nume-vechi TO nume-nou;**

unde:

**nume-vechi** și **nume-nou** sunt constante de tip șir de caractere (nu se scriu între ghilimele sau apostrofuri).

## **4.4. CREAREA ȘI ACTUALIZAREA INDECȘILOR**

În vederea obținerii de performanțe superioare privind accesul la tabelele unei baze de date se pot construi indecși folosind comanda **CREATE INDEX**.

Comanda are ca efect crearea unui index la o tabelă, index care va conține câte o intrare pentru fiecare valoare care apare în coloana specificată pentru tabela respectivă. Această coloană se numește coloana de index.

Sintaxa comenzii **CREATE INDEX** este:

**CREATE [UNIQUE] INDEX nume-index  
ON nume-tabela (nume-col [ASC | DESC], nume-col [ASC | DESC],...)  
[COMPRESS | NOCOMPRESS]  
[SYSSORT | NOSYSSORT]  
[ROWS =n][PCTFREE = {20 | n}];**

unde:

**UNIQUE** se referă la faptul că tabela nu va conține două rânduri cu aceleași valori în coloanele index (deci nu vor exista dubluri în coloana index). Dacă se omite, tabela poate conține oricâte asemenea rânduri care să aibă în coloana index aceeași valoare.

**ASC|DESC** precizează ordinea în care va fi păstrat indexul: ascendentă sau descendentă.

**COMPRESS** indică faptul că indecșii pot fi comprimați, reducându-se în acest mod spațiul necesar memorării acestora și mărindu-se viteza operațiilor care folosesc indecșii. Comprimarea nu permite găsirea într-un index a valorii unui anumit câmp fără să acceseze tabela. **NOCOMPRESS** suprimă comprimarea indecșilor.

**SYSSORT** specifică faptul că procedura standard ORACLE de sortare este folosită pentru a crea un index. **NOSYSSORT** este folosit pentru depanare.

**ROWS** specifică numărul aproximativ de rânduri ce urmează a fi indexate. Acest număr este utilizat pentru a optimiza sortarea în **SYSSORT**. Clauza nu are efect în **SQL\*Plus**.

**PCTFREE** specifică, în momentul creării indexu-lui, procentul din spațiul pentru index ce trebuie să rămână liber pentru actualizări. Nu are efect asupra extensiilor alocate după crearea indexului.

Indexul creat permite creșterea vitezei de acces la datele unei tabele prin intermediul coloanei index datorită faptului că se vor căuta rândurile corespunzătoare valorilor coloanei index, fără a citi întreaga tabelă. Deoarece indexul și tabela trebuie actualizate, introducerile, ștergerile și modificările în câmpul de valori ale coloanei index necesită un timp mai mare. De aceea, se recomandă crearea de maxim 3 indecși pentru o tabelă. În funcție de necesități, se pot crea și șterge în mod dinamic indecși.

Indecșii creați pentru o singură tabelă trebuie să aibă nume distincte. Pentru tabele diferite pot fi creați indecși cu același nume. În momentul ștergerii acestor indecși trebuie specificată tabela din care fac parte.

Comanda de validare a unui index este **VALIDATE INDEX**. Această comandă verifică integritatea indexului specificat pentru o anumită tabelă. Dacă pe tabela respectivă a fost creat un singur index, al cărui nume va fi specificat în comandă, atunci numele tablei poate fi omis. Comanda **VALIDATE INDEX** are sintaxa:

**VALIDATE INDEX nume-index**  
**[ON nume-tabelă] [WITH LIST];**

unde:

**WITH LIST** salvează informațiile referitoare la index într-un fișier. nume-index este numele indexului ce urmează a fi validat. Dacă indexul este valid, comanda va afișa un mesaj corespunzător (Index validated). Obținerea oricărui alt mesaj va determina ștergerea indexului și recrearea lui. Ștergerea unui index din baza de date se realizează cu ajutorul comenzii **DROP INDEX**, cu sintaxa:

**DROP INDEX nume-index [ON nume-tabelă];**

Dacă pe o anumită tabelă a fost creat un singur index, numele tabelii respective poate fi omis în linia de comandă. Este posibil ca doi indecși creați pe tabele diferite să aibă același nume. În acest caz, în linia de comandă vor apărea atât numele indexului cât și al tabelii pentru care a fost creat. Se recomandă numai ștergerea indecșilor proprii.

**Exemple:**

1) Să se creeze un index cu numele SUMAR pentru coloana FUNCT din tabela SALARIAȚI.

**SQL> CREATE INDEX SUMAR**

**2 ON SALARIAȚI(FUNCT);**

**Index created**

2) Să se creeze un index cu numele MARCA pentru coloana MARCA din tabela SALARIAȚI.

**SQL> CREATE INDEX MARCA**

**2 ON SALARIAȚI(MARCA);**

**Index created.**

3) Să se creeze un index cu numele MARCA din pentru coloana MARCA clia tabela SALARIAȚI cu suprimarea comprimării lui.

**SQL> CREATE INDEX MARCA**

**2 ON SALARIAȚI(MARCA) NOCOMPRESS;**

**Index created.**

4) Să se verifice integritatea indexului SUMAR.

**SQL> VALIDATE INDEX SUMAR;**

**Index validated.**

6) Să se verifice integritatea indexului MARCA pentru tabela SALARIAȚI.

**SQL> VALIDATE INDEX MARCA ON SALARIAȚI;**

**Index validated.**

7) Să se șteargă indexul SUMAR.

**SQL> DROP INDEX SUMAR;**

**Index dropped.**

8) Să se șteargă indexul MARCA din tabela SALARIAȚI.

**SQL> DROP INDEX MARCA ON SALARIAȚI;**

*Index dropped.*

#### **4.5. CREAREA ȘI ACTUALIZAREA TABELELOR ȘI INDECȘILOR PARTIȚIONAȚI**

**Partiționarea** este procesul de împărțire a tabelelor și indecșilor foarte mari în mai multe tabele, și respectiv indecși, mai mici numite **partiții** pentru a le putea manipula mai ușor.

O dată definite partițiile, instrucțiunile SQL pot accesa și manipula aceste partiții în loc de a manipula întreaga tabelă sau întregul index din care au provenit.

Toate partițiile au aceleași atribute logice ca și tabela sau indexul din care au provenit. De exemplu toate partițiile unei tabele au aceleași coloane cu aceleași constrângeri de integritate ca și tabela, iar partițiile unui index sunt construite după aceeași coloană ca și indexul din care au provenit.

Fiecare partiție este memorată într-un segment propriu aflat în aceeași tabelă spațiu sau în tabele spațiu diferite. Plasarea partițiilor în tabele spațiu diferită prezintă următoarele avantaje semnificative:

- Riscul pierderii datelor poate fi diminuat;
- Salvarea și restaurarea partițiilor se poate face independent unele față de altele;
- Se poate realiza o echilibrare a operațiilor de I/O prin maparea partițiilor pe discuri diferite.

O partiție este definită de următoarele:

- *Numele partiției* identifică în mod unic partiția în schema de obiecte a unui utilizator;
- *Referirea unei partiții* se face în context obligatoriu cu numele tablei sau indexului din care provine. Numele unei partiții poate să apară în instrucțiuni DDL și DML, precum și în utilitarele Import/Export și SQL\*Loader.

**Exemplu:**

***ALTER TABLE tab10 DROP PARTITION part1;***

- *Aria de cuprindere a partiției* este formată din rândurile tablei sau indexului, care aparțin acesteia bazat pe valorile coloanelor ce definesc partiția. Aria de cuprindere este definită prin clauzele:

**PARTITION BY RANGE ( coloana1, colana2, ...)**

și

**VALUES LESS THAN (valoare1, valoare2, ...)**

unde *coloana1*, *coloana2*, ... formează lista coloanelor. În funcție de valorile acestora se stabilesc rândurile din tabelă sau index care aparțin partiției. Coloanele după care se face partiționarea nu trebuie să conțină valori de tip NULL, iar numărul lor nu poate fi mai mare de cât 16; *valoare1*, *valoare2*, ... formează lista valorilor coloanelor. În funcție ele se selectează rândurile pentru a fi incluse în partiție. Aceste valori formează limita superioară a partiției. *Limitele partiției* definesc mulțimea rândurilor tabelii sau indexului ce vor fi incluse în aceasta. Orice partiție are două limite, *limita inferioară* a cărei valoare este definită de valoarea LESS THAN din definirea partiției precedente și **care este inclusă în partiție**, și *limita superioară este* definită de valoarea LESS THAN din definirea partiției curente, valoare **care nu este inclusă în partiție**. De la această regulă face excepție prima partiție care nu are limită inferioară. Partiționarea nu se poate face după pseudocoloanele LEVEL, ROWID sau MISLABEL. Ca valoare în clauza LESS THAN a ultimei partiții se poate specifica și MAXVALUE, care reprezintă o valoare virtuală egală cu infinit. *Cheia de partiționare* este un set de valori format din valorile coloanelor de partiționare aferente unui rând al partiției. Specificarea unei valori mai alta decât MAXVALUE pentru parametrul LESS THAN *impune o constrângere implicită* de tip CHECK la nivel de tabelă, cu această valoare.

### Reguli de partiționare a tabelelor și a indecșilor

*Partiționarea tabelelor* se face respectând câteva reguli esențiale, astfel:

- O tabelă *poate fi* partiționată dacă *nu* este inclusă într-un grup de tabele sau *nu* conține tipurile de date LOB, LONG, LONG RAW sau obiect;
- O tabelă partiționată sau nepartiționată *poate avea* indecși și partiționați și/sau nepartiționați;
- *Atributele fizice* ale unei partiții pot fi specificate inițial prin comanda CREATE TABLE pentru crearea partiției *implicit* sau *explicit*. *Specificarea implicită* se face prin furnizarea atributelor fizice pentru tabelă și nespecificarea acestora pentru partiție în clauza PARTITION. Atributele fizice implicite pot fi ulterior modificate cu comanda **ALTER TABLE MODIFY DEFAULT ATTRIBUTES**. *Specificarea explicită* se face prin furnizarea acestora în clauza PARTITION, caz în care valorile atributelor fizice ale partiției vor suprascrie valorile atributelor fizice implicite ale tabelii. Atributele fizice explicite pot fi ulterior modificate cu comenzile

**ALTER TABLE MODIFY PARTITION**

sau

## **ALTER TABLE MOVE PARTITION**

*Partiționarea indecșilor se face respectând câteva reguli:*

- Un index *poate fi* partiționat dacă *nu* este inclus într-un grup de indecși sau nu este definit pentru o tabelă inclusă într-un grup de tabele;
- *Indecșii pot fi de patru categorii: locali prefixați, locali neprefixați, globali prefixați și globali neprefixați.* Indecșii globali pot fi partiționați sau nepartiționați. *Un index este local* dacă cheile dintr-o partiție oarecare a acestuia referă rânduri aflate într-o singură partiție a tabelului pentru care a fost creat. Indexul *local este prefixat* dacă este definit pe un set de coloane dintre care cea mai din stânga este coloana după care a fost partiționat atât indexul local, cât și tabela pe care a fost creat acesta. De exemplu, presupunem că avem tabela TAB10 și indexul său INDEX10, care au fost partiționați după coloanele COL1 și COL2. Dacă indexul INDEX10 este definit pe coloanele (COL1, COL2, COL3), atunci el este *local prefixat* pentru că în partea cea mai din stânga(prefix) a listei coloanelor de indexare se află coloanele COL1 și COL2 după care s-a făcut partajarea tabelului și a indexului său. Dacă indexul INDEX10 ar fi definit pe coloanele (COL1, COL3) atunci indexul este *local neprefixat*. Indecșii locali nu pot fi partiționați cu clauza BY RANGE, această clauză se aplică numai pentru indecșii globali. *Un index este global* dacă cheile dintr-o partiție oarecare a acestuia referă rânduri aflate în mai multe partiții ale tabelului pentru care a fost creat. La fel ca și în cazul indexului local putem avea *index global prefixat* și *index global neprefixat*. Indexul global arată faptul că *partiționarea acestuia este definită de utilizator și nu trebuie să fie echivalentă* cu partiționarea tabelului pentru care se creează.
- Definirea atributelor fizice ale indecșilor comportă aceleași reguli ca la definirea atributelor fizice ale partițiilor unei tabele, cu deosebirea că aici se folosesc pentru *crearea unui index* comenzile:

### **CREATE INDEX**

sau

### **ALTER TABLE SPLIT PARTITION,**

iar pentru *modificarea* atributelor fizice comenzile SQL

### **ALTER INDEX MODIFY PARTITION**

sau

### **ALTER INDEX REBUILD PARTITION.**

### **Crearea partițiilor unei tabeli și ale unui index**

Crearea partițiilor este similară cu cea a creării tabelilor sau indecșilor.

*Crearea partițiilor unei tabeli* se face cu comanda **CREATE TABLE** folosind și clauza **PARTITION**.

#### ***Exemplu:***

Avem tabela **FACTURI** ce cuprinde documente din anii 1999 - 2002. Tabela are printre alte coloane și coloanele **AN**, **LUNA**, **ZI** și dorim să partiționăm tabela în trei partiții, astfel încât partiția 1 să conțină date din anii 1999 și 2000, partiția 2 date din anul 2001, iar partiția 3 date din anul 2001. Cu comanda de mai jos vom crea cele 3 partiții, astfel:

```
CREATE TABLE facturi  
(numar_factura NUMBER(12),  
nume_furnizor VARCHAR(25),  
an NUMBER(4),  
luna NUMBER(2),  
zi NUMBER(2))  
STORAGE (INITIAL 100K NEXT 50K) LOGGING  
PARTITION BY RANGE (an, luna, zi)  
(PARTITION p1_1999_2000 VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_1 NOLOGGING,  
PARTITION p1_2001 VALUES LESS THAN (2002, 13, 32)  
TABLESPACE tabsp_2 NOLOGGING,  
PARTITION p1_2002 VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_3 NOLOGGING);
```

*Crearea partițiilor unui index* se face cu comanda **CREATE INDEX** folosind și clauza **PARTITION**. Activitatea comportă unele diferențe de la un tip de index la altul, conform exemplurilor de mai jos:

#### ***Exemple:***

- 1) Crearea unui *index local prefixat* pentru tabela de mai sus *cu specificarea partițiilor*:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi,  
număr_factură)  
LOCAL (PARTITION p1 TABLESPACE tabsp1,  
PARTITION p2 TABLESPACE tabsp1,  
PARTITION p3 TABLESPACE tabsp1);
```

- 2) Crearea unui *index local prefixat* pentru tabela de mai sus *fără specificarea partițiilor*:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi,  
număr_factură) LOCAL;
```

- 3) Crearea unui *index local neprefixat* pentru tabela de mai sus *cu specificarea* partițiilor:

```
CREATE INDEX index_loc_prefix ON facturi  
LOCAL (PARTITION p1 TABLESPACE tabsp1,  
PARTITION p2 TABLESPACE tabsp1,  
PARTITION p3 TABLESPACE tabsp1);
```

- 4) Crearea unui *index local neprefixat* pentru tabela de mai sus *fără specificarea* partițiilor:

```
CREATE INDEX index_loc_prefix ON facturi (an, lună, zi)  
LOCAL;
```

- 5) Crearea unui *index global prefixat* pentru tabela de mai sus *cu specificarea* numelui partițiilor și a parametrului GLOBAL:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi,  
număr_factură)  
GLOBAL PARTITION BY RANGE (an, luna, zi)  
(PARTITION p1_1999_2000 VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_1 NOLOGGING,  
PARTITION p1_2001 VALUES LESS THAN (2002, 13, 32)  
TABLESPACE tabsp_2 NOLOGGING,  
PARTITION p1_2002 VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_3 NOLOGGING);
```

- 6) Crearea unui *index global prefixat* pentru tabela de mai sus *fără specificarea* numelui partițiilor și a parametrului GLOBAL:

```
CREATE INDEX index_loc_prefix ON facturi (an, luna, zi,  
număr_factură)  
PARTITION BY RANGE (an, luna, zi)  
(PARTITION VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_1 NOLOGGING,  
PARTITION VALUES LESS THAN (2002, 13, 32)  
TABLESPACE tabsp_2 NOLOGGING,  
PARTITION VALUES LESS THAN (2001, 13, 32)  
TABLESPACE tabsp_3 NOLOGGING);
```

- 7) Crearea unui *index global prefixat* pentru tabela de mai sus *cu specificarea* numelui partițiilor și a parametrului GLOBAL, cu un număr de partiții diferit de cel al partițiilor tabelii pe care se crează și cu alte coloane de partiționare decât cele după care s-a partiționat tabela:

```
CREATE INDEX index_loc_prefix ON facturi (an,lună,  
număr_factură)  
GLOBAL PARTITION BY RANGE (an, lună)  
(PARTITION p1_1999_2001 VALUES LESS THAN (2002, 13)  
TABLESPACE tabsp_1 NOLOGGING,
```



***PARTITION p1\_2001 VALUES LESS THAN (2003, 13)  
TABLESPACE tabsp\_2 NOLOGGING);***

### **Întreținerea partițiilor**

*Întreținere a partițiilor se realizează prin executarea activităților de modificare, mutare, adăugare, distrugere, trunchiere, splitare, reunire a acestora, precum și schimbarea partițiilor și reconstruirea partițiilor index.*

- **Modificarea** unei partiții a unei tabelă se face cu comanda **ALTER TABLE** cu clauza **MODIFY PARTITION**. Cu această comandă se pot modifica *atributele fizice* ale partiției sau ale partiției indexului aferent.
- **Mutarea partițiilor** unei tabelă sau index se face pentru a schimba tabela spațiu în care rezidă acestea din diverse considerente, dintre care cele de obținere a unor performanțe în exploatarea sunt cele mai frecvente. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **MOVE PARTITION**.

**Exemplu:**

***ALTER TABLE tab10 MOVE PARTITION part1 TABLESPACE  
tabsp10 NOLOGGING;***

Mutarea partiției unei tabelă care conține date, determină necesitatea recreării tuturor indecșilor locali sau globali atașați acesteia.

- **Adăugarea unor noi partiții** se poate executa doar pentru o tabelă partiționată sau un index local. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **ADD PARTITION**.

**Exemplu:**

***ALTER TABLE tab100 ADD PARTITION part1 VALUES LESS  
THAN (935);***

O nouă partiție poate fi adăugată doar după partiția cu limita superioară cea mai mare. Dacă dorim să adăugăm o partiție la început, între partițiile existente sau după ultima partiție care are limita superioară egală cu valoarea MAXVALUE, atunci acest lucru se poate realiza prin splitarea unei partiții, în cazul de față prima, una adiacentă cu locul de inserare a noii partiții, și respectiv ultima partiție. Dacă pentru tabela partiționată căreia îi adăugăm o nouă partiție există un index local, atunci Oracle creează automat o partiție de index pentru noua partiție adăugată.

- **Distrugerea partițiilor unei tabelă** se face după anumite reguli funcție de situațiile în care se află partiția, cu ajutorul comenzii **ALTER TABLE** cu opțiunea **DROP PARTITION**. *Distrugerea partiției unei tabelă care conține date și a indexului global se poate face lăsând*

nealterați indecșii globali în timpul distrugerii partiției, după care aceștia vor fi recreați sau ștergând toate rândurile partiției cu comanda **DELETE** după care distrugem partiția. Această comandă actualizează indecșii globali. *Distrugerea partiției unei tabele care conține date și a constrângerilor referențiale de integritate* se poate face dezactivând constrângerile de integritate, distrugând partiția și apoi reactivând constrângerile de integritate. Ștergerea rândurilor partiției cu comanda **DELETE**, apoi distrugem partiția.

- **Distrugerea partițiilor unui index** se face după anumite reguli funcție de situațiile în care se află partiția, cu ajutorul comenzii **ALTER INDEX** cu opțiunea **DROP PARTITION**. O partiție a unui index local nu poate fi distrusă, ea se distruge implicit atunci când se distruge partiția tabelului căreia îi corespunde. O partiție fără date a unui index global poate fi distrusă. Dacă o partiție a unui index global conține date, atunci prin distrugerea acesteia partiția următoare devine invalidă și trebuie recreată;
- **Trunchierea partițiilor unei tabele** se face cu comanda **ALTER TABLE** cu opțiunea **TRUNCATE PARTITION** și are ca efect ștergerea tuturor rândurilor de date din această partiție și a partiției corespunzătoare a indexului local asociat, dacă există. Partițiile indecșilor nu pot fi trunchiate, singură trunchiere posibilă este cea specificată mai sus. Trunchierea unei partiții a unei tabele și a indecșilor globali asociați și/sau a constrângerilor de integritate referențială se face după aceleași reguli ca și operația de distrugere.
- **Splitarea partițiilor** unei tabele sau ale unui index se face cu comanda **ALTER TABLE/INDEX** cu opțiunea **SPLIT PARTITION**. O partiție a unei tabele ce conține date, prin splitare toți indecșii asociați devin inutilizabili și ca atare aceștia trebuie recreați. Numai partiția fără date nu invalidează indecșii. Partiția unui index poate fi splitată numai dacă indexul este global și nu conține date, căci partiția unui index local se splitază automat atunci când se splitază partiția tabelului căreia îi corespunde.
- **Fuzionarea sau reunirea partițiilor** unei tabele sau ale unui index se face cu ajutorul comenzilor **ALTER TABLE/INDEX** și cu una din opțiunile **DROP PARTITION** sau **EXCHANGE PARTITION**, pentru că o opțiune explicită de fuzionare nu există. O partiție a unei tabele poate fi reunită cu altă partiție numai dacă nu are indecși globali sau constrângeri de integritate referențiale asociate. Fuzionarea unei partiții a unei tabele se face totdeauna cu partiția imediat superioară. Presupunem că avem tabela TAB10 cu partițiile P1, P2, P3 și P4 și vrem

să reunim partiția P2 cu P3, se vor exporta datele din tabele P2, se execută comanda **ALTER TABLE tab10 DROP PARTITION p2** , după care importăm datele exportate în partiția P3.

- **Fuzionarea sau reunirea partițiilor unui index** local se face implicit când se reunesc partițiile corespunzătoare ale acestora, iar fuzionarea a două partiții P2 și P3, care conțin date, ale unui index global, se poate face astfel:

**ALTER INDEX index\_global DROP PARTITION p2;**

**ALTER INDEX index\_global REBUILD PARTITION p3;**

**Schimbarea partițiilor** realizează transformarea unei partiții a unei tabele într-o tabelă nepartiționată sau o tabelă nepartiționată într-o partiție a unei tabele partiționate. Operația se execută cu comanda **ALTER TABLE** cu opțiunea **EXCHANGE PARTITION**.

#### **4.6. CREAREA ȘI ACTUALIZAREA VEDERILOR**

O *viziune (vedere sau tabelă virtuală)* este o formă de prezentare a datelor din una sau mai multe tabele sau viziuni pentru un utilizator, obținută prin executarea unei cereri. O viziune este tratată ca o tabelă și se mai numește și *tabelă virtuală*.

Utilizatorul care creează viziunea trebuie să aibă privilegiul **CREATE VIEW** sau **CREATE ANY VIEW**. Proprietarul viziunii trebuie să aibă, în mod explicit, acordate privilegiile de acces la toate obiectele referite de către viziune, privilegiile ce nu pot fi obținute prin intermediul rolului. De asemenea, funcționalitatea unei viziuni depinde de privilegiile proprietarului acesteia. De exemplu, dacă proprietarul viziunii are privilegiul **SELECT** pentru tabela din care s-a creat viziunea (numită și tabelă de bază), atunci acesta poate executa prin intermediul viziunii doar operații de **SELECT** din tabelă.

Operația se execută cu comanda SQL **CREATE VIEW**, astfel:

- 1) **CREATE VIEW v10 AS**  
**SELECT col1, col2, col4**  
**FROM tab2**  
**WHERE col1 = 20;**

Crearea unei viziuni *v10*, cu coloanele *col1*, *col2* și *col4*, ca un subset de date din tabela *tab2*, care are coloanele *col1,col2,col3,col4,col5*

- 2) **CREATE VIEW v11 AS**  
**SELECT col1, col2, col4, col7**  
**FROM tab2, tab3**  
**WHERE tab2.col1 = tab3.col1;**

Crearea viziunii *v11* ca reuniune a unor date din tabelele *tab2 (col1,col2,col3,col4,col5)* și *tab3 (col1, col8, col9)*:

Dacă în timpul creării unei viziuni Oracle detectează anumite erori acestea sunt semnalate, iar dacă se folosește opțiunea **FORCE** viziunea este totuși creată cu starea **INVALID**. Cu această opțiune o vedere poate fi creată chiar dacă tabela sau tabelele de bază nu există. Viziunea astfel creată va fi validă, deci va putea fi utilizată, abia după ce se va crea tabela de bază, iar proprietarul viziunii va primi drepturile necesare de utilizare a acestora.

**Vederea de tip reuniune (vedere join)** este definită ca vederea care cumulează rânduri din mai multe tabele. Modificarea unei astfel de vederi se face respectându-se condiția de *cheie rezervată*. O tabelă se numește *tabelă cu cheie rezervată* dacă orice cheie a acesteia poate fi cheie în vederea tip reuniune a cărei tabelă de bază este. Altfel spus, o tabelă cu cheie rezervată are cheile rezervate în cadrul vederii join. Prin intermediul unei astfel de vederi *se pot actualiza* date (**UPDATE**, **DELETE** sau **INSERT**) *numai* în tabela de baza care conține cheia sau cheile rezervate, cu condiția *obligatorie* ca opțiunea **SELECT** de creare a vederii să *nu conțină* una din *clauzele* **DISTINCT**, **GROUP BY**, **START WITH**, **CONNECT BY**, **ROWNUM** și *nici o operație* de setare de tip **UNION**, **UNION ALL**, **INTERSECT** sau **MINUS**. Deci prin intermediul unei vederi de tip reuniune se pot modifica date numai asupra coloanelor care se mapează pe tabela de bază care conține cheia sau cheile rezervate. Cu ajutorul vederilor **ALL\_UPDATABLE\_COLUMNS**, **DBA\_UPDATABLE\_COLUMNS** și **USER\_UPDATABLE\_COLUMNS** din dicționarul de date se pot obține informații despre coloanele vederii de tip reuniune ce pot fi modificate.

**Înlocuirea unei vederi** este operația de recrearea acesteia și se execută prin distrugerea vederii și recrearea acesteia și redefinirea vederii cu clauza **OR REPLACE**.

**Exemplu:**

```
CREATE OR REPLACE VIEW v10 AS  
SELECT col1, col2 FROM tab2  
WHERE col1 = 30;
```

Înainte de a înlocui o vedere, trebuie avute în vedere următoarele efecte:

- Înlocuirea unei vederi determină înlocuirea definiției acesteia din dicționarul de date;
- Dacă în vedere înlocuită a existat clauza **CHECK OPTION**, iar în definiția noii vederi nu mai este inclusă, această clauză este distrusă;
- Toate vederile și programele PL/SQL dependente de vedere înlocuită devin invalide.

**Distrugerea vederilor** se execută cu comanda **DROP VIEW** .

**Vederea partiționată** împarte o tabelă foarte mare în bucăți mai mici numite partiții și le reunește pe acestea pentru a se obține performanțe în administrare și regăsirea datelor. Cererile care folosesc anumite intervale de valori conforme cu cele folosite la crearea partițiilor vor regăsi date numai din partițiile aferente acestora. O vedere partiționată se creează folosind constrângerea de integritate CHECK sau clauza WHERE. Considerăm tabelele t1, t2 și t3 cu aceleași coloane, deci pot fi considerate partiții ale unei tabele care la însumează.

**Exemplu:**

```
ALTER TABLE t1 ADD CONSTRAINT c1 CHECK (col1 between  
0 and 1000);
```

```
ALTER TABLE t2 ADD CONSTRAINT c1 CHECK (col1 between  
1000 and 10000);
```

```
ALTER TABLE t3 ADD CONSTRAINT c1 CHECK (col1 between  
10000 and 100000);
```

```
CREATE VIEW v10 AS
```

```
SELECT * FROM t1 UNION ALL
```

```
SELECT * FROM t2 UNION ALL
```

```
SELECT * FROM t3;
```

```
CREATE VIEW v10 AS
```

```
SELECT * FROM t1 WHERE col1 between 0  
and 1000 UNION ALL
```

```
SELECT * FROM t2 WHERE col1 between 1000  
and 10000 UNION ALL
```

```
SELECT * FROM t3 WHERE col1 between 10000 and 100000);
```

#### **4.7. CREAREA ȘI ACTUALIZAREA SECVENȚELOR**

**Secvențele** sunt numere unice de identificare a coloanelor unei tabele și pot fi utilizate la efectuarea diferitelor operații într-o aplicație.

**Crearea** unei secvențe se execută cu comanda **CREATE SEQUENCE**, astfel:

```
CREATE SEQUENCE secv_1 INCREMENT BY 1
```

```
START WITH 1
```

```
NOMAXVALUE
```

```
CACHE 10;
```

unde:

INCREMENT BY arată valoare cu care se incrementează o secvență curentă pentru a se obține secvența următoare,

START WITH este valoarea de pornire a secvenței (prima valoare),  
NOMAXVALUE arată că nu avem o limită superioară până unde se pot genera secvențe,

CACHE definește numărul de secvențe viitoare care se păstrează anticipat în memorie pentru obținerea unor performanțe superioare. Când această valoare se epuizează Oracle încarcă în memorie următorul set de secvențe.

**Modificarea** unei secvențe se face cu comanda SQL **ALTER SEQUENCE** și poate opera asupra parametrilor inițiali ai comenzii de creare a secvenței.

Parametrul de inițializare **SEQUENCE\_CACHE\_ENTRIES** determină numărul secvențelor care pot fi ținute în memorie de Oracle.

**Distrugerea** secvențelor se face cu comanda SQL **DROP SEQUENCE**.

**Referirea** secvențelor se face cu pseudocoloanele **NEXTVAL** și **CURRVAL**, în care:

**NEXTVAL** generează următoarea valoare a secvenței. Referirea se face prin *nume\_secvență.NEXTVAL*.

**Exemple:**

1) **CREATE SEQUENCE** *secv1*;  
**INSERT INTO** *tab1* (*COL1*, *COL2*) **VALUES** (*secv1.NEXTVAL*,  
*300*);

*CURRVAL* definește valoarea curentă a secvenței și se referă prin *nume\_*

2) **INSERT INTO** *tab10* (*COL4*, *COL5*) **VALUES**  
(*secv1.CURRVAL*, *1300*);  
**INSERT INTO** *tab10* (*COL4*, *COL5*) **VALUES**  
(*secv1.CURRVAL*, *2 300*);

Valoarea **NEXTVAL** poate fi referită o singură dată, iar valoarea **CURRVAL** de mai multe ori, cu condiția ca valoarea **NEXTVAL** să fi fost referită, deci secvența curentă să fi fost creată.

#### **4.8. CREAREA ȘI ACTUALIZAREA SINONIMELOR**

**Sinonimul** este un alt nume (alias) pentru o tabelă, o vedere, secvență, procedură, funcție sau pachet. Sinonimul poate fi *public* sau *privat*. Sinonimul public este inclus în schema unui grup de utilizatori numit **PUBLIC** și este accesibil tuturor utilizatorilor, iar cel privat aparține numai unui anumit utilizator.

**Crearea** sinonimului se face cu comanda **CREATE SYNONYM** și se distruge cu comanda **DROP SYNONYM**.

**Exemple:**

Crearea unui sinonim privat:

***CREATE SYNONYM sin1 FOR tab10;***

Crearea unui sinonim public:

***CREATE PUBLIC SYNONYM sin10 FOR tab10;***

Distrugerea unui sinonim:

***DROP SYNONYM sin1;***

***DROP PUBLIC SYNONYM sin10;***

#### **4.9. CREAREA ȘI ACTUALIZAREA GRUPURILOR DE TABELE ȘI A GRUPURILOR DE INDECȘI**

**Grupul de tabele (cluster)** este o metodă de memorare comprimată a unor tabele de date care au coloane comune și care sunt foarte des folosite împreună.

**Cheia grupului (key cluster)** este coloana sau grupul de coloane pe care tabelele grupate le au comune. Cheia grupului se va specifica atunci când se creează acesta și atunci când se creează tabelele ce vor fi incluse în grup. Fiecare valoare a cheii de grup se va memora o singură în cadrul grupului de tabele sau al grupului de indecși indiferent de câte ori apare aceasta în tabelele grupului.

**Coloanele care se aleg pentru a fi definite cheile de grup** sunt cele folosite cel mai mult pentru a reuni tabelele atunci când se execută o anumită cerere. Cea mai bună cheie de grup este aceea care are suficiente valori unice, astfel încât rândurile care se grupează după aceasta să poată fi incluse într-un singur bloc de date. Astfel dacă avem *prea puține* rânduri pe o cheie de grup obținem o pierdere a spațiului de memorie și performanțe neglijabile, iar dacă avem *prea multe* timpul de căutare suplimentar, căutare în mai multe blocuri de date, va duce la degradarea performanțelor.

**Setarea parametrilor de memorie** PCTFREE și PCTUSED trebuie făcută cu mare grijă, astfel încât să nu afectăm spațiul utilizat pentru inserarea rândurilor și nici pe cel ce va fi folosit pentru actualizarea datelor aferente rândurilor inserate în bloc. Valorile acestor parametrii folosite la definirea grupului sunt automat utilizate și pentru tabelele ce vor fi grupate. Chiar dacă vom specifica acești parametrii la crearea unei tabele în cadrul grupului ei vor fi ignorați.

**Specificarea spațiului** necesar pentru memorarea rândurilor aferente unei chei de grup se face prin intermediul clauzei *SIZE* a comenzii SQL *CREATE CLUSTER*. Valoarea acestui parametru este specificată în bytes și reprezintă spațiul ce trebuie rezervat în cadrul blocului de date aferent grupului pentru memorarea valorii sau valorilor cheii de grup. Prin intermediul acestuia, Oracle determină numărul rândurilor de date ce încap

într-un bloc de date al grupului. Dacă SIZE este specificat astfel încât într-un bloc de date să încapă două chei de grup atunci spațiul acestui bloc este folosit de ambele chei, iar dacă un bloc de date nu poate cuprinde toate rândurile aferente unei chei de grup, cheia de grup se memorează o singură dată, iar blocurile de date se înlanțuiesc de blocul în care se află cheia de grup. Dacă într-un bloc de date încap mai multe chei de grup, atunci acesta poate să aparțină mai multor lanțuri de date.

**Specificarea locului (tabelei spațiu)** în care să fie plasat grupul de tabele și grupul index asociat este obligatorie la crearea acestor grupuri.

**Crearea grupului de tabele (cluster)** se face cu comandă SQL *CREATE CLUSTER*. Se va crea întâi grupul de tabele și apoi tabelele ce vor face parte din grup. Atributele fizice se furnizează o singură dată, doar pentru grup nu și pentru tabele.

*Exemplu:*

```
CREATE CLUSTER grup1 (col1 NUMBER (5))  
PCTUSED 75 PCTFREE 10  
SIZE 600  
TABLESPACE tabsp1  
STORAGE (INITIAL 200k  
NEXT 290k  
MINEXTENTS 3  
MAXEXTENTS 25  
PCTINCREASE 30);
```

```
CREATE TABLE tab1 (col1 NUMBER(5) PRIMARY KEY, col2  
NUMBER (10), ...) CLUSTER grup1 (col1);  
CREATE TABLE tab2 (col3 NUMBER(5) PRIMARY KEY, col4  
NUMBER (10), ...,  
col1 NUMBER (5) REFERENCE tab1(col1)) CLUSTER grup1  
(col1);
```

unde *col1* este cheia grupului.

**Crearea grupului de indecși (cluster)** se face cu comandă SQL *CREATE INDEX* cu clauza *ON CLUSTER*. Se va crea întâi grupul de tabele și apoi grupul de indecși asociat.

*Exemplu:*

```
CREATE INDEX index1  
ON CLUSTER grup1  
INITRANS 2  
MAXTRANS 5  
TABLESPACE tabsp2GR
```



***PCTFREE 10  
STORAGE (INITIAL 50k  
NEXT 50k  
MINEXTENTS 3  
MAXEXTENTS 25  
PCTINCREASE 30);***

**Modificarea grupurilor** de tabele sau de indecși se face cu comanda SQL *ALTER CLUSTER*. Elementele ce pot face obiectul modificării sunt atributele fizice ale grupului.