BAZE DE DATE

CURS 11

SQL (partea 1)

SQL

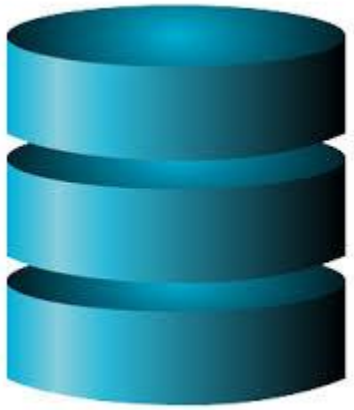
- ***Structured Query Language***
  - limbaj universal care poate fi utilizat pentru a **defini**, **interoga**, **reactualiza** şi **gestiona** baze de date relaţionale.
    - LDD + LMD + LCD + alte comenzi (de administrare etc.)
  - SQL este un limbaj **neprocedural**, adică se specifică **CE** informaţie este solicitată, dar nu modul **CUM** se obţine această informaţie.
  - SQL poate fi utilizat autonom sau prin inserarea comenzilor sale într-un limbaj de programare.
    - Cum realizați acest lucru în Java?
  - Prima versiune: 1986; ultima: 2016
  - Restricţii asupra identificatorilor (vezi curs)

SQL

- 3 familii de comenzi:
  - Comenzi pentru definirea datelor (LDD) - permit **descrierea** (definirea) **structurii obiectelor** ce modelează sistemul studiat.
  - Comenzi pentru prelucrarea datelor (LMD) - ce permit **consultarea, reactualizarea, suprimarea sau inserarea** datelor.
  - Comenzi pentru controlul datelor (LCD) - permit asigurarea **confidenţialităţii şi integrităţii** datelor, **salvarea** informaţiei, realizarea fizică a **modificărilor** în baza de date, rezolvarea unor probleme de **concurenţă**.

LDD

- O **BD** este alcătuită din **scheme**.
- O **schemă** este o mulţime de structuri logice de date, numite **obiecte**. Ea aparţine unui **utilizator al bazei de date** şi poartă numele său.
- Specificarea bazelor de date şi a obiectelor care le compun se realizează prin intermediul limbajului de definire a datelor (LDD).
- Definirea unui obiect presupune crearea, modificarea şi suprimarea sa. Limbajul de definire a datelor cuprinde instrucţiunile SQL care permit realizarea acestor operaţii (**CREATE**, **ALTER**, **DROP**).
- Instrucţiunile LDD au **efect imediat asupra bazei de date** şi înregistrează informaţia în dicţionarul datelor.
- LDD contine și instructiunile **RENAME**, **TRUNCATE** si **COMMENT**.



LDD

Tipuri de date

- Pentru memorarea datelor numerice, tipurile cele mai frecvent folosite sunt: **NUMBER**, **INTEGER**, **FLOAT**, **DECIMAL**.
- Pentru memorarea şirurilor de caractere, cele mai frecvent tipuri de date utilizate sunt: **CHAR**, **VARCHAR2** şi **LONG (**inlocuit de **CLOB)**.
- Informaţii relative la timp sau dată calendaristică se obţin utilizând tipul **DATE**.
  - secolul, anul, luna, ziua, ora, minutul, secunda
  - pentru o coloană de tip DATE sistemul rezervă 7 bytes
  - NLS_DATE_FORMAT
- Alte tipuri de date pentru date calendaristice si timp:

  TIMESTAMP (precizie_fracţiuni_secundă), **INTERVAL YEAR** (precizie_an) **TO MONTH**, **INTERVAL DAY** (precizie_zi) **TO SECOND** (prec_fracţiuni_sec)

  ➢ exemple in curs!

LDD

Modele de format

- Un model de format este un literal caracter care descrie formatul valorilor de tip DATE sau NUMBER stocate într-un şir de caractere.
- Atunci când se converteşte un şir de caractere într-o dată calendaristică sau într-un număr, modelul de format indică sistemului cum să interpreteze şirul respectiv.
- În instrucţiunile SQL se poate folosi un model de format ca argument al funcţiilor **TO_CHAR** şi **TO_DATE.**
  - Detalii in curs

LDD

Valoarea NULL

- Valoarea *null*, reprezentând lipsa datelor, nu este egală sau diferită de nici o altă valoare, inclusiv *null*!
- Excepții: există două situaţii în care sistemul Oracle consideră două valori *null* ca fiind egale:
  - la evaluarea funcţiei DECODE
  - dacă valorile *null* apar în chei (unice) compuse. Două chei compuse care conţin valori *null* sunt considerate identice dacă toate componentele diferite de *null* sunt egale.

LDD

Pseudocoloane

- O pseudocoloană se comportă ca o coloană a unui tabel, dar nu este stocată efectiv într-un tabel.
- Se pot face interogări asupra pseudocoloanelor, dar nu se pot insera, actualiza sau şterge valorile acestora.
- LEVEL returnează nivelul liniilor rezultat ale unei cereri ierarhice.
- CURRVAL şi NEXTVAL sunt pseudocoloane utile în lucrul cu secvenţe şi sunt tratate în secţiunea corespunzătoare acestora.
- ROWID returnează adresa unei linii din baza de date, furnizând modul cel mai rapid de a accesa linia respectivă.
  - conţin următoarele informaţii necesare pentru a localiza o linie: numărul obiectului, blocul de date, fişierul de date, linia în cadrul blocului de date. Valorile pseudocoloanei ROWID sunt de tipul ROWID sau UROWID.
- ROWNUM returnează numărul de ordine al liniilor rezultate în urma execuţiei unei cereri. Pseudocoloana poate fi utilizată pentru a limita numărul de linii returnate. Exemplu!

Tabele

Crearea unui tabel

- constă din generarea **structurii** sale, adică atribuirea unui nume tabelului şi definirea caracteristicilor sale (se definesc coloanele, se definesc constrângerile de integritate, se specifică parametrii de stocare etc.).
- Comanda CREATE TABLE permite crearea unui **tabel relaţional** sau a unui tabel obiect. Tabelul relaţional reprezintă **structura fundamentală pentru stocarea datelor utilizatorului**.

Tabele

- Pentru a crea un tabel, utilizatorul trebuie să aibă acest **privilegiu** şi să dispună de **spaţiul de memorie** în care să creeze obiectul.
- La nivelul schemei sale, un utilizator are toate privilegiile.

```
CREATE TABLE [<nume_schema>.] <nume_tabel> (
        <nume_coloana_1> <tip_date> [DEFAULT <expresie>], …
        <nume_coloana_n> <tip_date> [DEFAULT <expresie>])
        [CLUSTER <nume_cluster> (<coloana_1>,…, <coloana_m>)][ENABLE |
DISABLE <clause>];
```

- Comanda poate conţine opţional clauza **TABLESPACE**, care specifică spaţiul tabel în care va fi stocat tabelul.
- De asemenea, poate conţine opţional clauza **STORAGE** care este folosită pentru setarea parametrilor de stocare prin intermediul cărora se specifică mărimea şi modul de alocare a extinderilor segmentului tabel.
- Exemple de creare de tabele in curs.

Constrângeri

- Constrângerea este un mecanism care asigură că valorile unei coloane sau ale unei mulţimi de coloane satisfac o **condiţie declarată**.
- Unei constrâgeri i se poate da un **nume unic** în cadrul schemei.
  - Dacă nu se specifică un nume explicit atunci sistemul automat îi atribuie un nume de forma **SYS_C<n>**, unde **n** reprezintă numărul constrângerii.
- Constrângerile pot fi **şterse**, pot fi **adăugate**, pot fi **activate sau dezactivate**, dar *nu pot fi modificate*.

Constrângeri

- **Constrângeri declarative**: constrângeri de domeniu, constrângerea de integritate a entităţii, constrângerea de integritate referenţială.
- **Constrângerile de domeniu** definesc valori luate de un atribut (DEFAULT, CHECK, UNIQUE, NOT NULL).
- **Constrângerea de integritate a entităţii** precizează cheia primară a unui tabel.
  - Când se creează cheia primară se generează automat un **index unic**.
  - **Valorile cheii primare sunt distincte şi diferite de valoarea *null*.**
- **Constrângerea de integritate referenţială** asigură coerenţa între cheile primare şi cheile externe corespunzătoare.
  - se verifică dacă a fost definită o cheie primară pentru tabelul referit de cheia externă;
  - dacă numărul coloanelor ce compun cheia externă corespunde numărului de coloane ale cheii primare;
  - dacă tipul şi lungimea fiecărei coloane a cheii externe corespunde cu tipul şi lungimea fiecărei coloane a cheii primare.

Constrângeri

- Definiţiile şi numele constrângerilor definite se pot fi consulta prin interogarea vizualizărilor **USER_CONSTRAINTS** şi ALL_CONSTRAINTS din dicţionarul datelor.
- Există posibilitatea ca o constrângere să fie amânată (**DEFERRABLE**).
  - În acest caz, mai multe comenzi SQL pot fi executate fără a se verifica restricţia, aceasta fiind verificată numai la sfârşitul tranzacţiei, atunci când este executată comanda COMMIT.
  - Dacă vreuna din comenzile tranzacţiei încalcă restricţia, atunci întreaga tranzacţie este derulată înapoi şi este returnată o eroare. Opţiunea implicită este **NOT DEFERRABLE**.

Partiţionarea tabelelor

- Tabelele pot fi partiționate -> exemplu în curs.

Modificarea structurii unui tabel

- Comanda care realizează modificarea structurii tabelului (la nivel de coloană sau la nivel de tabel), dar **nu modificarea conţinutului acestuia**, este **ALTER TABLE**.
- ALTER TABLE realizează **modificarea structurii** unui tabel nepartiţionat sau partiţionat, a unei partiţii sau subpartiţii dintr-un tabel.
- Comanda **nu schimbă conţinutul tabelului**.

Modificarea structurii unui tabel

- Comanda **ALTER TABLE** permite:
  - adăugarea (**ADD**) de coloane, chei (primare sau externe), constrângeri într-un tabel existent;
  - modificarea (**MODIFY**) coloanelor unui tabel, inclusiv specificarea unei valori implicite pentru o coloană existentă;
  - suprimarea unei coloane (**DROP COLUMN**);
  - adăugarea de constrângeri (**ADD CONSTRAINT**);
  - suprimarea (**DROP CONSTRAINT**) cheii primare, a cheii externe sau a altor constrângeri;
  - activarea şi dezactivarea (**ENABLE**, **DISABLE**) unor constrângeri.

Modificarea structurii unui tabel

- Comanda ALTER TABLE are următoarea sintaxă simplificată:

```
ALTER TABLE [<nume_schema>.] <nume_tabel>
[ADD(<nume_coloana> <tip_date>, <constrângere>)
| MODIFY (<nume_coloana_1>,…, <nume_coloana_n>)
|   DROP          <clauza_drop>,]   [ENABLE | DISABLE
<clause>];
```

- Exemple în curs.

Suprimarea unui tabel

- Pentru ştergerea unui tabel este utilizată comanda DROP TABLE:

```
DROP TABLE [nume_schema.]nume_tabel [CASCADE CONSTRAINTS];
```

- Clauza ***CASCADE CONSTRAINTS*** permite suprimarea tuturor constrângerilor de integritate referenţială corespunzătoare cheilor primare şi unice din tabelul supus ştergerii.
  - Altfel, sistemul returnează o eroare şi nu suprimă tabelul.
- Suprimarea unui tabel presupune:
  - suprimarea definiţiei sale în dicţionarul datelor;
  - suprimarea indecşilor asociaţi;
  - suprimarea privilegiilor conferite în legătură cu tabelul;
  - recuperarea spaţiului ocupat de tabel;
  - permanentizarea tranzactiilor in asteptare;
  - **invalidarea** (dar nu suprimarea) funcţiilor, procedurilor, vizualizărilor, secventelor, sinonimelor referitoare la tabel.

Suprimarea unui tabel

- Odată executată, instrucţiunea DROP TABLE este ireversibilă.
- Ca şi în cazul celorlalte instrucţiuni ale limbajului de definire a datelor, această comandă nu poate fi anulată (*ROLLBACK*).
- Oracle 10g a introdus o noua manieră pentru suprimarea unui tabel.
  - Când se șterge un tabel, baza de date nu eliberează imediat spațiul asociat tabelului.
  - Ea redenumește tabelul și acesta este plasat într-un *recycle bin* de unde poate fi eventual recuperat ulterior prin comanda **FLASHBACK TABLE**.

Suprimarea unui tabel

- Exemplu:

```
DROP TABLE exemplu;

SELECT …

INSERT…

FLASHBACK TABLE exemplu TO BEFORE DROP;
```

- Ștergerea unui tabel se poate face simultan cu eliberarea spațiului asociat tabelului, dacă este utilizată clauza **PURGE** în comanda **DROP TABLE**.
  - Nu este posibil un ***rollback*** pe o comanda **DROP TABLE** cu clauza **PURGE**.
- Pentru ştergerea întregului conţinut al unui tabel şi eliberarea spaţiului de memorie ocupat de acesta, sistemul Oracle oferă instrucţiunea:

```
TRUNCATE TABLE nume_tabel;
```

Suprimarea unui tabel

TRUNCATE sau DELETE?

- Fiind o instrucţiune LDD, TRUNCATE nu poate fi anulată ulterior (prin ROLLBACK).
- Ea reprezintă o alternativă a comenzii DELETE din limbajul de prelucrare a datelor.
- **DELETE nu eliberează spaţiul de memorie.**
- **TRUNCATE este mai rapidă** deoarece nu generează informaţie ROLLBACK şi nu activează declanşatorii asociaţi operaţiei de ştergere.
  - Dacă tabelul este „părintele“ unei constrângeri de integritate referenţială, el **nu poate fi trunchiat**.
  - Pentru a putea fi aplicată instrucţiunea TRUNCATE**, constrângerea trebuie să fie mai întâi dezactivată.**

Informații în DD

- În DD, informaţiile despre tabele se găsesc în vizualizarea USER_TABLES. Dintre cele mai importante coloane ale acesteia, se remarcă:

TABLE_NAME	Numele tabelului
TABLESPACE_NAME	Spaţiul tabel în care se află tabelul
CLUSTER_NAME	Numele cluster-ului din care face parte tabelul
PCT_FREE	Procentul de spaţiu păstrat liber în interiorul fiecărui bloc
PCT_USED	Procentul de spaţiu ce poate fi utilizat în fiecare bloc
INI_TRANS	Numărul iniţial de tranzacţii concurente în interiorul unui bloc
NUM_ROWS	Numărul de înregistrări din tabel
BLOCKS	Numărul de blocuri utilizate de tabel
EMPTY_BLOCKS	Numărul de blocuri ce nu conţin date
TEMPORARY	Y sau N, indică dacă tabelul este temporar (sau nu)

Informații în DD

- **Exemplu:**

```
DESCRIBE USER_TABLES;
SELECT   TABLE_NAME, NUM_ROWS, TEMPORARY
FROM     USER_TABLES;
```

- **Exemplu:**

```
SELECT 'DROP TABLE' || OBJECT_NAME || ';' FROM
USER_OBJECTS
WHERE OBJECT_TYPE = 'TABLE';
```

Tabele temporare

Sintaxa comenzii CREATE TABLE este următoarea:

```
CREATE [GLOBAL TEMPORARY] TABLE [schema.]nume_tabel          {nume_coloană_1
tip_date [DEFAULT expresie]           [constr_coloană_ref] [constr_coloană
[constr_coloană] …]           | {constr_tabel_sau_view | constr_tabel_ref} }
    [, {nume_coloană_2…} ]
     [ON COMMIT {DELETE | PRESERVE} ROWS]           [proprietăţi_fizice]
[proprietăţi_tabel];
```

- Opţiunea **GLOBAL TEMPORARY** permite **crearea unui tabel temporar**, al cărui scop este de a stoca **date specifice unei sesiuni sau unei tranzacţii**.
- **Definiţia este accesibilă tuturor sesiunilor**, dar informaţiile sunt vizibile numai sesiunii care inserează linii în acesta.
- **ON COMMIT DELETE ROWS** se utilizează pentru definirea unui tabel temporar **specific unei tranzacţii**, caz în care sistemul trunchiază tabelul, ştergând toate liniile acestuia după fiecare operaţie de permanentizare (COMMIT).
- **ON COMMIT PRESERVE ROWS** se specifică pentru a defini un tabel temporar **specific unei sesiuni**, caz în care sistemul trunchiază tabelul la terminarea sesiunii.

Tabele temporare

Exemplu:

Să se creeze un tabel temporar achizitii_azi. Sesiunea fiecărui angajat care se ocupă de achiziţii va permite stocarea în acest tabel a achiziţiilor sale de la data curentă. La sfârşitul sesiunii, aceste date vor fi şterse.

```
CREATE GLOBAL TEMPORARY TABLE achizitii_azi
    ON COMMIT PRESERVE ROWS
    AS SELECT  *
    FROM    opera
    WHERE   data_achizitiei = SYSDATE;
```

Indecşi

- Un index este un obiect al schemei unei baze de date care:
    - **creşte viteza de execuţie a cererilor;**
    - **garantează că o coloană conţine valori unice.**
- Server-ul Oracle utilizează identificatorul ROWID pentru regăsirea liniilor în structura fizică a bazei de date.
- Indexul, din punct de vedere logic, este compus **dintr-o valoare cheie şi din identificatorul adresă ROWID**.
- Cheia indexului poate fi coloana unui tabel sau concatenarea mai multor coloane (numărul maxim de coloane care pot defini cheia indexului este 32).
- Indecşii sunt utilizaţi şi întreţinuţi automat de către server-ul Oracle. Odată creat indexul, el nu necesită o acţiune directă din partea utilizatorului.

Indecşi

- Indecşii pot fi creaţi în două moduri:
  - automat, de server-ul Oracle (**PRIMARY KEY, UNIQUE KEY**);
  - manual, de către utilizator (CREATE INDEX, CREATE TABLE).

- Server-ul Oracle creează automat un **index unic** atunci când se defineşte o constrângere PRIMARY KEY sau UNIQUE asupra unei coloane sau unui grup de coloane.
- Numele indexului va fi acelaşi cu numele constrângerii.
- Indecşii, fiind obiecte ale schemei bazei, beneficiază de procesul de definire a unui obiect.

Indecşi

- Crearea unui index (care nu este obligatoriu unic) pe una sau mai multe coloane ale unui tabel se face prin comanda:

```
CREATE [UNIQUE] INDEX <nume_index> ON [<nume_schema>.]
<nume_tabel>                (<nume_col> [ASC | DESC],
<nume_col> [ASC | DESC], …)
| CLUSTER <nume_cluster>];
```

Indecşi

- **Mai mulţi indecşi asupra unui tabel nu implică întotdeuna interogări mai rapide.**
- Fiecare operaţie LMD care este permanentizată asupra unui tabel cu indecşi asociaţi presupune **actualizarea indecşilor respectivi.**
- Prin urmare, **este recomandată** crearea de indecşi numai în anumite situaţii:
  - coloana conţine o varietate mare de valori;
  - coloana conţine un număr mare de valori *null*;
  - una sau mai multe coloane sunt utilizate frecvent împreună într-o clauză WHERE sau într-o condiţie de join;
  - tabelul este mare şi este de aşteptat ca majoritatea interogărilor asupra acestuia să regăsească mai puţin de 2-4% din linii.

Indecşi

- Crearea unui index **nu este recomandată** în următoarele cazuri:
  - tabelul este mic;
  - coloanele nu sunt utilizate des în cadrul unei condiţii dintr-o cerere;
  - majoritatea interogărilor regăsesc mai mult de 2-4% din liniile tabelului;
  - tabelul este actualizat frecvent;
  - coloanele indexate sunt referite în expresii.

Indecşi

In concluzie, ce tabele sau ce coloane trebuie (sau nu) indexate?

- indexaţi tabelele pentru care interogările selectează **un număr redus de rânduri** (sub 5%);
- indexaţi tabelele care sunt interogate folosind **clauze SQL simple**;
- nu indexaţi tabelele ce conţin puţine înregistrări (accesul secvenţial este mai simplu);
- nu indexaţi tabelele care sunt frecvent actualizate, deoarece **ştergerile, inserările şi modificările sunt îngreunate de indecşi**;
- indexaţi coloanele folosite frecvent în clauza **WHERE** sau în clauza **ORDER BY**;
- nu indexaţi coloanele ce conţin date asemănătoare (puţine valori distincte).

Indecşi

Exemplu:

```
CREATE INDEX upper_nume_idx ON artist (UPPER(nume));
```

- Indexul creat prin instrucţiunea precedentă facilitează prelucrarea unor interogări precum:

```
SELECT    *   FROM      artist
WHERE     UPPER(nume) = 'GRIGORESCU';
```

- Pentru a asigura că server-ul Oracle **utilizează indexul** şi nu efectuează o căutare asupra întregului tabel, valoarea funcţiei corespunzătoare expresiei indexate **trebuie să nu fie null** în interogările ulterioare creării indexului. Următoarea instrucţiune garantează utilizarea indexului dar, în absenţa clauzei WHERE, serverul Oracle ar putea cerceta întreg tabelul.

```
SELECT    *   FROM      artist
WHERE     UPPER(nume) IS NOT NULL
ORDER BY UPPER(nume);
```

Indecşi

Exemplu:
Să se creeze tabelul *artist*, specificându-se indexul asociat cheii primare.

```
CREATE TABLE artist (cod_artist NUMBER PRIMARY KEY
                                 USING INDEX
                                    (CREATE INDEX artist_cod_idx
                                     ON artist(cod_artist)),
                             nume    VARCHAR2(30) NOT NULL,
                             prenume VARCHAR2(30),    …);
```

- **Ştergerea** unui index se face prin comanda:

```
DROP INDEX nume_index [ON [nume_schema.] nume_tabel]
```

  - Pentru a suprima indexul trebuie ca acesta să se găsească în schema personală sau să avem privilegiul de sistem DROP ANY INDEX.

- Pentru a **reconstrui un index** se pot folosi două metode:
  - se şterge indexul (DROP INDEX) şi se recreează (CREATE INDEX);
  - se utilizează comanda ALTER INDEX cu opţiunea REBUILD.

Indecşi

Tipuri de indecşi:

- index de tip arbore B* – creat la executarea unei comenzi standard CREATE INDEX;
- index partiţionat – folosit în cazul tabelelor mari pentru a stoca valorile coloanei indexate în mai multe segmente;
- index de cluster – bazat pe coloanele comune ale unui cluster;
- index cu cheie inversă – sunt B* arbori, dar care stochează datele în mod invers;
- index de tip bitmap – nu se stochează valorile efective ale coloanei indexate, ci un bitmap format pe baza acestor valori.

Indecşi

Informații în DD:

- Informaţii referitoare la indecşi şi la coloanele care îi definesc pot fi regăsite în vizualizările USER_INDEXES, respectiv USER_IND_COLUMNS din dicţionarul datelor.

Exemplu:
Să se obţină informaţii referitoare la indecşii tabelului *carte*.

```
SELECT a.index_name, a.column_name,   a.column_position poz,
b.uniqueness
FROM  user_indexes b, user_ind_columns a
WHERE a.index_name = b.index_name
AND  a.table_name = 'carte';
```

Secvenţe

- O secvenţă este un obiect în baza de date care serveşte pentru a genera **întregi unici** în sistemele multi-utilizator, evitând apariţia conflictelor şi a blocării.
- Secvenţele sunt memorate şi generate **independent de tabele**; aceeaşi secvenţă poate fi utilizată pentru mai multe tabele.
- O secvenţă poate fi creată de un utilizator şi poate fi partajată de mai mulţi utilizatori.
- Crearea unei secvenţe se face cu ajutorul comenzii:

```
CREATE SEQUENCE [<nume_schema>.]<nume_secventa>   [INCREMENT BY n]  [START
WITH m]   [{MAXVALUE n | NOMAXVALUE}]  [{MINVALUE n | NOMINVALUE}]
[{CACHE k | NOCACHE}]
```

  - CACHE k | NOCACHE specifică numărul de valori alocate de server-ul Oracle pe care le va păstra în memoria *cache* pentru a oferi utilizatorilor un acces rapid (implicit sunt alocate 20 de valori);

Secvenţe

- O secvenţă este referită într-o comandă SQL cu ajutorul pseudo-coloanelor:
  - NEXTVAL – referă valoarea următoare a secvenţei;
  - CURRVAL – referă valoarea curentă a secvenţei.

- NEXTVAL şi CURRVAL pot fi folosite în:
  - clauza VALUES a unei comenzi INSERT;
  - clauza SET a unei comenzi UPDATE;
  - lista SELECT a unei subcereri dintr-o comanda INSERT;
  - lista unei comenzi SELECT.

Secvenţe

- NEXTVAL şi CURRVAL nu pot fi folosite în:
  - subinterogare în SELECT, DELETE sau UPDATE;
  - interogarea unei vizualizări;
  - comandă SELECT cu operatorul DISTINCT;
  - comandă SELECT cu clauza GROUP BY, HAVING sau ORDER BY;
  - clauza WHERE a unei comenzi SELECT;
  - condiţia unei constrângeri CHECK;
  - valoarea DEFAULT a unei coloane într-o comandă CREATE TABLE sau ALTER TABLE;
  - comandă SELECT care este combinată cu altă comandă SELECT printr-un operator pe mulţimi (UNION, INTERSECT, MINUS).

Secvenţe

- Din dicţionarul datelor pot fi obţinute informaţii despre secvenţe folosind vizualizarea USER_SEQUENCES.

Exemplu:

- Să se creeze o secvenţă *domeniuseq* care să fie utilizată pentru a insera noi domenii în tabelul domeniu şi să se insereze un nou domeniu.
- Să se afişeze informaţiile referitoare la secvenţa *domeniuseq*.

```
CREATE SEQUENCE domeniuseq
 START WITH 1
 INCREMENT BY 1;

 INSERT INTO domeniu  VALUES
 (domeniuseq.NEXTVAL,'Informatica');

 SELECT INCREMENT, START, MAXVALUE, MINVALUE
 FROM   USER_SEQUENCES
 WHERE  SEQUENCE_NAME = 'domeniuseq';
```

Vizualizări

- Vizualizarea (view) este un **tabel logic (virtual)** relativ la date din una sau mai multe tabele sau vizualizări.
- Vizualizarea este definită plecând de la o **cerere** a limbajului de interogare a datelor, moştenind caracteristicile obiectelor la care se referă.
- Vizualizarea, fiind virtuală, **nu solicită o alocare de memorie** pentru date.
- Textul cererii (SELECT) care defineşte vizualizarea este salvat în DD.
- Oracle transformă cererea referitoare la o vizualizare într-o cerere relativă la tabelele de bază.
  - Nucleul Oracle determină **fuzionarea** cererii relative la vizualizare cu comanda de definire a vizualizării, analizează rezultatul fuziunii în zona partajată şi execută cererea.
  - Dacă sunt utilizate clauzele UNION, GROUP BY şi CONNECT BY, atunci Oracle nu determină fuzionarea, el va rezolva vizualizarea şi apoi va aplica cererea rezultatului obţinut.

Vizualizări

- O vizualizare reflectă la orice moment **conţinutul** exact al tabelelor de bază.
- Orice modificare efectuată asupra tabelelor se repercutează instantaneu asupra vizualizării.
- Ştergerea unui tabel implică **invalidarea vizualizărilor asociate** tabelului şi nu ştergerea acestora.
- Vizualizările sunt definite pentru:
  - furnizarea unui nivel mai înalt de **securizare a bazei** (restricţionarea acesului la date);
  - simplificarea formulării unei cereri;
  - mascarea complexităţii datelor;
  - afişarea datelor într-o altă reprezentare decât cea a tabelelor de bază;
  - asigurarea confidenţialităţii anumitor informaţii;
  - definirea constrângerilor de integritate.

Vizualizări

- Vizualizările pot fi **simple** şi **complexe**.
- O vizualizare **simplă**:
  - extrage date dintr-un singur tabel,
  - nu conţine funcţii sau grupări de date şi
  - asupra ei pot fi efectuate operaţii LMD.
- O vizualizare este considerată **complexă** dacă:
  - extrage date din mai multe tabele,
  - conţine funcţii sau grupări de date şi
  - nu permite întotdeauna (prin intermediul său) operaţii LMD asupra tabelelor de bază.

Vizualizări

- Operaţiile LMD asupra vizualizărilor complexe sunt restricţionate de următoarele **reguli**:
  - nu se poate insera, actualiza sau şterge o linie dintr-o vizualizare dacă aceasta conţine **funcţii grup, clauza GROUP BY, cuvântul cheie DISTINCT sau pseudocoloana ROWNUM**;
  - nu se poate adăuga sau modifica o linie dintr-o vizualizare, dacă aceasta conţine **coloane definite prin expresii**;
- Nu pot fi adăugate linii printr-o vizualizare, dacă tabelul de bază conţine coloane care au **constrângerea NOT NULL şi nu apar în lista SELECT a vizualizării**
  - generalizare și pentru celelelate tipuri de constrângeri

Vizualizări

- Pentru a obţine informaţii referitoare la vizualizările definite, se pot interoga vizualizările **USER_VIEWS** şi **ALL_VIEWS** din dicţionarul datelor.
- Textul instrucţiunii SELECT care defineşte o vizualizare este stocat într-o coloană de tip LONG, numită **TEXT**.
- Atunci când datele sunt accesate prin intermediul unei vizualizări, server-ul Oracle efectuează următoarele operaţii:
  - recuperează definiţia acesteia din USER_VIEWS;
  - verifică privilegiile de acces la tabelele ei de bază;
  - converteşte cererea într-o operaţie echivalentă asupra tabelelor de bază.

Vizualizări

- Crearea unei vizualizări se realizează cu ajutorul comenzii:

```
CREATE [OR REPLACE][FORCE | NOFORCE] VIEW   [<nume_schema>.]<nume_view>
[(<alias>[,<alias>]…)]     AS <cerere_SELECT>
[WITH {CHECK OPTION [CONSTRAINT <nume_constrangere>] |READ ONLY }];
```

– **OR REPLACE** recreează vizualizarea dacă aceasta deja există.
– **FORCE** creează vizualizarea chiar dacă tabelul de bază nu există sau chiar dacă vizualizarea face referinţă la obiecte care încă nu sunt create. Deşi vizualizarea va fi creată, utilizatorul nu poate să o folosească.
– **NO FORCE** este implicită şi se referă la faptul că vizualizarea este creată numai dacă tabelele de bază există.
– Cererea este o **comandă SELECT** care poate să conţină alias pentru coloane.
– **WITH CHECK OPTION** specifică faptul că reactualizarea datelor din tabele (inserare sau modificare) se poate face numai asupra datelor selectate de vizualizare (care apar în clauza WHERE).
– **WITH READ ONLY** asigură că nici o operaţie LMD nu poate fi executată asupra vizualizării.

Vizualizări

Exemplu:
Să se genereze o vizualizare care conţine informaţii referitoare la împrumutul cărţilor şi în care să fie implementată constrângerea că orice carte, care există într-un singur exemplar, poate fi împrumutată maximum 15 zile.

```
CREATE VIEW imprumutare
AS SELECT * FROM  imprumuta
WHERE cod_carte NOT IN
       (SELECT cod   FROM carte  WHERE nrex = 1)
OR  datares - dataim < 15
WITH CHECK OPTION;
```

Vizualizări

- Interogarea unei vizualizări este similară celei unui tabel.
- Numărul coloanelor specificate în definiţia vizualizării trebuie să fie egal cu cel din lista asociată comenzii SELECT.

Vizualizări

- Asupra cererii care defineşte vizualizarea se impun următoarele **restricţii**:
  - **nu pot fi selectate pseudocoloanele CURRVAL şi NEXTVAL** ale unei secvenţe;
  - dacă sunt selectate pseudocoloanele **ROWID, ROWNUM sau LEVEL**, acestora trebuie să li se specifice alias-uri (-> în general, pentru coloanele calculate);
  - dacă cererea selectează toate coloanele unui tabel, utilizând simbolul „*“, iar ulterior se adaugă coloane noi tabelului, **vizualizarea nu va conţine acele coloane** până la recrearea sa printr-o instrucţiune CREATE OR REPLACE VIEW.

Vizualizări

Exemplu:

- Să se creeze o vizualizare care conţine numele şi prenumele artistului, numărul operelor sale şi valoarea medie a acestora.

```
CREATE VIEW artist_nr_val(nume, numar_opere, val_medie) AS
SELECT   a.nume || ' ' || a .prenume "Nume si prenume", COUNT(o. cod_opera) numar,
         AVG(o.valoare) medie
FROM     opera o, artist a
WHERE    o.cod_artist = a.cod_artist
GROUP BY o.cod_artist, a.nume, a.prenume;
```

- Să se creeze vizualizarea sculptura ce va conţine codul operei, data achiziţiei, codul artistului şi stilul operelor al căror tip este „sculptura“.

```
CREATE OR REPLACE VIEW sculptura (cod_sculptura, informatii, cod_sculptor, stil) AS
SELECT   cod_opera, 'Sculptura ' || titlu || ' a fost achizitionata la data ' ||
data_achizitiei, cod_artist, stil
FROM     opera
WHERE    tip = 'sculptura';
```

Vizualizări

- **Reactualizarea tabelelor implică reactualizarea corespunzătoare a vizualizărilor!!! Reactualizarea vizualizărilor implică reactualizarea tabelelor de bază? NU! Există restricţii care trebuie respectate!!!**
- Nu pot fi modificate date din vizualizare sau adaugate date prin vizualizare, dacă aceasta conține **coloane definite prin expresii**.
- Nu pot fi inserate, şterse sau actualizate date din vizualizări ce **conţin**:
  - operatorul DISTINCT;
  - clauzele GROUP BY, HAVING, START WITH, CONNECT BY;
  - pseudo-coloana ROWNUM;
  - funcţii grup;
  - operatori de mulţimi.
- Nu pot fi inserate sau actualizate date care ar încălca **constrângerile din tabelele de bază**.

Vizualizări

- Nu pot fi inserate sau actualizate **valorile coloanelor care rezultă prin calcul**.
- Nu se pot face operaţii LMD asupra coloanelor calculate cu DECODE.

Alături de restricţiile prezentate anterior, aplicabile tuturor vizualizărilor, există restricţii specifice, aplicabile vizualizărilor bazate pe mai multe tabele.

- **Regula fundamentală este că orice operaţie INSERT, UPDATE sau DELETE pe o vizualizare bazată pe mai multe tabele poate modifica datele doar din unul din tabelele de bază. In care???**
- Un tabel de bază al unei vizualizări este **protejat prin cheie (key preserved table)** dacă orice cheie selectată a tabelului este de asemenea şi cheie a vizualizării.
  - Deci, un tabel protejat prin cheie este un tabel ale cărui chei se păstrează şi la nivel de vizualizare.

Vizualizări

- Asupra unui *join view* pot fi aplicate instrucţiunile INSERT, UPDATE sau DELETE, doar dacă sunt îndeplinite următoarele condiţii:
  - instrucţiunea LMD afectează numai unul dintre tabelele de bază;
  - în cazul instrucţiunii **UPDATE**, toate coloanele care pot fi reactualizate trebuie să corespundă coloanelor dintr-un tabel protejat prin cheie (în caz contrar, Oracle nu va putea identifica unic înregistrarea care trebuie reactualizată);
  - în cazul instrucţiunii **DELETE**, rândurile unei vizualizări pot fi şterse numai dacă există un tabel în join protejat prin cheie şi numai unul (în caz contrar, Oracle nu ar şti din care tabel să şteargă);
  - în cazul instrucţiunii **INSERT**, toate coloanele în care sunt inserate valori trebuie să provină dintr-un tabel protejat prin cheie.

Vizualizări

- ALL_UPDATABLE_COLUMNS, DBA_UPDATABLE_COLUMNS şi USER_UPDATABLE_COLUMNS sunt vizualizări din DD ce conţin informaţii referitoare la coloanele vizualizărilor existente, care pot fi reactualizate

Exemplu:

1. Să se creeze un view ce conţine câmpurile nume, prenume, job din tabelul salariat.

2. Să se insereze, să se actualizeze şi să se şteargă o înregistrare în acest view. Ce efect vor avea aceste acţiuni asupra tabelului de bază?

```
CREATE VIEW viz2 AS
SELECT  nume, prenume, job FROM   salariat;
```

- View-ul nu conţine cheia primară!

```
INSERT INTO viz2 VALUES  ('Popescu','Valentin','grafician');
```

va genera eroarea: ORA-01400: cannot insert NULL into ("SCOTT"."SALARIAT"."COD_SALARIAT")

Vizualizări

- Actualizarea job-ului salariatului având numele "Popescu":

```
UPDATE  viz2 SET  job = 'programator' WHERE nume = 'Popescu'; SELECT
nume, prenume, job  FROM  salariat;
```

- Ştergerea înregistrării referitoare la salariatul având numele "Popescu":

```
DELETE FROM viz2 WHERE nume = 'Popescu';
```

- Operaţiile care se realizează asupra view-ului se realizează şi în tabelul salariat.
- Pentru un caz mai general, când view-ul conţine cheia externă a tabelului de bază, sunt permise modificări ale view-ului, dacă acestea nu afectează cheia externă.

Vizualizări

Exemplu:

Să se creeze un view care conţine câmpurile nume, prenume, job din tabelul salariat. Să se introducă în view doar persoanele care sunt graficieni.

```
CREATE VIEW viz21
AS SELECT  nume, prenume, job
   FROM   salariat
   WHERE  job = 'grafician'
   WITH CHECK OPTION;
```

Vizualizări

Exemplu:
Să se creeze o vizualizare care să conţină cod_salariat, nume, prenume din tabelul salariat şi coloana tip din tabelul grafician. Apoi să se insereze, să se actualizeze şi să se şteargă o înregistrare din acest view (vizualizarea conţine cheia primară cod_salariat din tabelele salariat şi grafician).

```
CREATE VIEW viz4
AS SELECT s.cod_salariat,nume,prenume,tip
FROM   salariat s, grafician g
WHERE  s.cod_salariat=g.cod_salariat;
```

- În cazul inserării unei înregistrări pentru care se specifică toate câmpurile:

```
INSERT INTO   viz4
VALUES (30,'Popescu','Valentin','artist plastic');
```

va apărea următoarea eroare: *ORA-01776: cannot modify more than one base TABLE through a join view*

Vizualizări

(continuare exemplu)

- Pot fi inserate date doar într-un tabel de bază (în oricare, dar în unul singur) prin intermediul view-ului, astfel:

```
INSERT INTO viz4 (cod_salariat, nume)
VALUES  (30, 'Popescu');
```

- Comanda pentru ştergerea unei înregistrări:

```
DELETE  viz4 WHERE  cod_salariat = 3;
```

va genera următoarea eroare: *ORA-01752: cannot delete from view without exactly one key-preserved TABLE*.

- Modificarea unei înregistrări se face prin secvenţa care urmează. Toate actualizările care se fac în view se fac şi în tabelele de bază.

```
UPDATE  viz4 SET   tip = 'designer'
WHERE  cod_salariat = 3;
```

Vizualizări

Exemplu:

Care dintre coloanele unei vizualizări sunt actualizabile?

```
SELECT column_name, updatable
FROM  user_updatable_columns
WHERE table_name = 'viz4';
```

Exemplu:

1. Să se creeze un view (viz3) care să conţină, pentru fiecare categorie de salariat, salariile medii şi numărul de angajaţi din tabelul salariat.
2. Să se insereze, să se actualizeze şi să se şteargă o înregistrare în view.

```
CREATE VIEW viz3 (nr, job, salmed)
AS SELECT COUNT(*), job, AVG(salariu)
FROM   salariat GROUP BY  job;
```

- Nu se pot face inserări, actualizări sau ştergeri într-un view ce conţine **funcţii grup**. După oricare din aceste operaţii apare acelaşi mesaj:

ORA-01732: data manipulation operation not legal on this view

Vizualizări

Exemplu:

Să se creeze o vizualizare care să conţină coloanele cod_contractant, adresa, telefon din tabelul contractant şi coloanele nr_contract, tip_contract, data_incheiere din tabelul contract. Să se insereze o înregistrare în vizualizare.

```
CREATE VIEW viz44
AS SELECT c.cod_contractant, adresa, telefon,    co.nr_contract, tip_contract,
data_incheiere
FROM   contractant c, contract co
WHERE  c.cod_contractant=co.cod_contractant;
```

- La inserarea unei înregistrări căreia i se specifică valorile tuturor câmpurilor din ambele tabele:

```
INSERT INTO viz44(cod_contractant, adresa, nr_contract, data_incheiere)
VALUES  (200, 'Str. Marmurei, 14', '6235', TO_DATE('January 03,2002','Month
dd,yyyy'));
```

se obţine eroarea: *ORA-01779: cannot modify a column which maps to a non key-preserved TABLE*

Vizualizări

- Cererea din definiţia vizualizării poate fi restricţionată prin clauzele WITH READ ONLY şi WITH CHECK OPTION.
- Opţiunea WITH READ ONLY asigură că nu pot fi efectuate operaţii LMD asupra vizualizării.
- Constrângerea WITH CHECK OPTION garantează faptul că va fi permisă, prin intermediul vizualizării, numai inserarea sau actualizarea de linii accesibile acesteia (care sunt selectate de cerere).
- **Prin urmare, această opţiune asigură constrângeri de integritate şi verificări asupra validităţii datelor inserate sau actualizate.**
- Opţiunea WITH CHECK OPTION nu poate funcţiona dacă:
  - există o cerere imbricată în cadrul subcererii vizualizării sau în vreuna dintre vizualizările de bază;
  - operaţiile de inserare, ştergere şi modificare se fac prin intermediul declanşatorilor INSTEAD OF.
- Cuvântul cheie CONSTRAINT permite numirea constrângerii WITH CHECK OPTION.

Vizualizări

Exemplu:
Să se creeze o vizualizare ce conţine artiştii de naţionalitate română, care au opere expuse în muzeu. Definiţia vizualizării nu va permite modificarea naţionalităţii unui artist sau inserarea unui artist având altă naţionalitate decât cea română.

```
CREATE VIEW artist_roman
AS SELECT   *
FROM     artist
WHERE    nationalitate = 'romana'
WITH CHECK OPTION CONSTRAINT artist_roman_ck;

UPDATE   artist_roman SET      nationalitate = 'engleza'
WHERE    cod_artist = 25;
```

- Încercarea de actualizare a unei linii prin instrucţiunea anterioară va genera eroarea *„ORA-01402: view WITH CHECK OPTION where-clause violation*“.

Vizualizări

Exemplu:
Să se creeze o vizualizare asupra tabelului galerie care să nu permită efectuarea nici unei operaţii LMD.

```
CREATE VIEW viz_galerie
AS SELECT   cod_galerie, nume_galerie
FROM     galerie
WITH READ ONLY;

DELETE FROM viz_galerie
WHERE  cod_galerie = 10;
```

- Dacă se încearcă modificarea sau inserarea unei linii prin intermediul unei vizualizări asupra căreia a fost definită o constrângere WITH READ ONLY, server-ul Oracle generează eroarea „ORA-01733: virtual column not allowed here“.

Vizualizări

Exemplu:

Să se creeze o vizualizare care conţine codul şi titlul operelor de artă, codul şi numele artiştilor care le-au creat, precum şi codul galeriilor unde sunt expuse. Să se afle dacă este posibilă adăugarea unei noi înregistrări prin intermediul acestei vizualizări.

```
CREATE VIEW opera_artist
AS SELECT o.cod_opera, o.titlu, o.cod_galerie,          a.cod_artist, a.nume
FROM   opera o, artist a
WHERE  o.cod_artist = a.cod_artist;
```

Instrucţiunea următoare afişează numele coloanelor şi valorile YES/NO, după cum aceste coloane sunt, sau nu, modificabile.

```
SELECT COLUMN_NAME, UPDATABLE
FROM   USER_UPDATABLE_COLUMNS
WHERE  TABLE_NAME = 'OPERA_ARTIST';
```

Se va obţine că doar primele trei coloane ale vizualizării sunt modificabile.

Vizualizări

Constrângeri asupra vizualizărilor

- Se pot defini constrângeri la nivel de vizualizare, respectiv la nivel de coloană sau atribut.
- Constrângerile asupra vizualizărilor constituie o submulţime a constrângerilor specifice tabelelor.
- Pot fi specificate explicit numai constrângerile UNIQUE, PRIMARY KEY şi FOREIGN KEY.
- Constrângerea de tip CHECK poate fi realizată prin precizarea clauzei WITH CHECK OPTION în comanda care defineşte vizualizarea.
- Constrângerile asupra vizualizărilor pot fi definite numai în modul **DISABLE NOVALIDATE**. Aceste cuvinte cheie trebuie specificate la declararea constrângerii, nefiind permisă precizarea altor stări.

Vizualizări

Exemplu:

Să se creeze o vizualizare care conţine codurile, numele şi adresele galeriilor. Se va impune unicitatea valorilor coloanei adresa şi constrângerea de cheie primară pentru coloana corespunzătoare codului galeriei.

```
CREATE VIEW viz_galerie(
      cod_gal, nume,
      adresa UNIQUE DISABLE NOVALIDATE,
      CONSTRAINT cp_viz PRIMARY KEY (cod_gal)
                  DISABLE NOVALIDATE)
AS SELECT cod_galerie, nume_galerie, adresa
   FROM galerie;
```

Grupări (clustere)

- Cluster-ul este o regrupare fizică a două sau mai multe tabele, relativ la una sau mai multe coloane, **cu scopul măririi performanţelor**.
- Coloanele comune definesc cheia cluster-ului.
- Un cluster este un obiect al bazei care necesită:
  - un nume unic la nivelul schemei,
  - specificarea coloanelor care compun cheia cluster-ului,
  - specificarea spaţiului de stocare (opţional),
  - un index (relativ la cheia cluster-ului).
- Un cluster trebuie să aibă cel puţin un index. Acest index trebuie creat înaintea oricărei comenzi LMD care va acţiona asupra tabelelor cluster-ului.
- Un index al cluster-ului se deosebeşte de un index al tabelului (de exemplu, absenţa indexului afectează utilizatorul – datele cluster-ului nu sunt accesibile).

Grupări (clustere)

- Coloanele comune definite pentru cluster, reprezintă cheia cluster-ului şi criteriul de regrupare.
- Liniile diferitelor tabele sunt regrupate **în interiorul aceluiaşi bloc** urmărind cheia cluster-ului.
- Dacă liniile asociate unei aceiaşi valori a cheii cluster-ului necesită un spaţiu de mai multe blocuri, atunci **blocurile sunt înlănţuite**.
- Crearea unui cluster presupune:
  - crearea structurii cluster-ului;
  - crearea indexului cluster-ului;
  - crearea tabelelor care vor compune cluster-ul.

Grupări (clustere)

- Crearea unui cluster se realizeaza prin comanda:

```
CREATE CLUSTER nume_cluster (nume_coloana tip_data
[,nume_coloana tip_data] …) [SIZE n]
```

- Există două modalităţi pentru introducerea unui tabel într-un cluster.
- O primă variantă presupune că cluster-ul este creat pentru un tabel care deja există.
- A doua variantă presupune că introducerea tabelului în cluster se face în momentul creării structurii tabelului (comanda CREATE TABLE).

Grupări (clustere)

Exerciţiu:

Să se obţină un cluster referitor la lista cărţilor din fiecare domeniu.

- Varianta 1

```
CREATE CLUSTER cdoml(cdom  CHAR(1));
CREATE INDEX indcom ON CLUSTER cdoml;
CREATE TABLE domeniu2   CLUSTER cdoml(coded)   AS SELECT *
FROM domeniu;
DROP TABLE domeniu;
RENAME domeniu2 TO domeniu;
ALTER TABLE carte MODIFY  coded NOT NULL;
CREATE TABLE carte2 CLUSTER cdoml(coded)   AS SELECT * FROM
carte;
DROP TABLE carte;
RENAME carte2 TO carte;
```

Grupări (clustere)

- Varianta 2

```
CREATE CLUSTER cdoml(cdom  CHAR(1));
CREATE INDEX indcom ON CLUSTER cdoml; -- crearea spatiului
CREATE TABLE domeniu (cod CHAR(1) NOT NULL,   intdom
CHAR() ... )
CLUSTER  cdoml(cod);

CREATE TABLE carte (cod CHAR(5) NOT NULL,    …
cod_dom CHAR(1) NOT NULL)
CLUSTER  cdoml(cod_dom);
```

Grupări (clustere)

- Pentru a scoate un tabel dintr-un cluster sunt parcurse următoarele etape:
  - se creează un nou tabel, în afara cluster-ului, prin duplicarea celui vechi;
  - se distruge tabelul din cluster;
  - se suprimă cluster-ul.

```
CREATE TABLE alfa  AS SELECT * FROM domeniu;

DROP TABLE domeniu;

RENAME alfa TO domeniu;

CREATE TABLE beta  AS SELECT * FROM carte;

DROP TABLE carte;

RENAME beta TO carte;

DROP CLUSTER cdoml;
```

Informaţii despre obiectele bazei de date

- Pot fi obţinute consultând DD.
- Dintre ele se remarcă:
  - definiţiile tuturor obiectelor din baza de date;
  - spaţiul alocat şi spaţiul utilizat în prezent de obiectele schemei;
  - constrângerile de integritate;
  - numele utilizatorilor bazei;
  - privilegiile şi rolurile acordate fiecărui rol;
  - alte informaţii generale despre baza de date.
- Tabelul USER_CATALOG conţine informaţii despre tabelele şi vizualizările definite de un utilizator particular. Acest tabel poate fi referit şi prin sinonimul său public CAT.
- Tabelul USER_OBJECTS conţine informaţii despre toate obiectele definite de utilizatorul curent. Tabelul are următoarea schemă relaţională: *USER_OBJECTS (object_name, object_id, object_type, created, last_ddl_time, timestamp, status)*

Informaţii despre obiectele bazei de date

Vizualizările cele mai importante ale dicţionarului datelor conţin:

- descrierea tabelelor definite de utilizatori (USER_ALL_TABLES),
- informaţii despre constrângerile definite de utilizator (USER_CONSTRAINTS),
- informaţii despre legăturile bazei de date (USER_DB_LINKS),
- erorile curente ale obiectelor depozitate (USER_ERRORS),
- informaţii despre indecşii creaţi de utilizator (USER_INDEXES),
- informaţii despre tabelele utilizatorului (USER_TABLES) etc.

Informaţii despre obiectele bazei de date

Vizualizările din dicţionarul datelor referitoare la tabele conţin:

- USER_TAB_COLUMNS|COLS – informaţii despre coloanele tabelelor,
- USER_CONS_COLUMNS – informaţii despre constrângeri la nivel coloană,
- USER_TAB_COMMENTS – informaţii despre comentarii la nivel tabel,
- USER_COL_COMMENTS – informaţii despre comentarii la nivel coloană,
- USER_TAB_PARTITIONS – informaţii despre partiţiile tabelelor.

Alte tipuri de obiecte ale schemei BD

- Sinonime
- Comentarii
- Legaturi intre baze de date (database link)
- Vizualizari materializate