

# BAZE DE DATE

CURS 12

SQL (partea 2)

# Limbajul de prelucrare a datelor

- *SQL* furnizează comenzi ce permit consultarea (*SELECT*) și actualizarea (*INSERT, UPDATE, DELETE, MERGE*) conținutului bazei de date.
- Aceste comenzi definesc limbajul de prelucrare a datelor (*LMD*).
- Comenzile limbajului *LMD* pot fi:
  - formulate direct, utilizând *SQL Developer, SQL\*PLUS* etc. ;
  - utilizate în utilitare ale sistemului *Oracle*;
  - încapsulate într-un program *PL/SQL* ;
  - încapsulate într-un program scris în limbaj gazdă.



# Comanda *SELECT*

- Una dintre cele mai importante comenzi ale limbajului de prelucrare a datelor este *SELECT*.
- Cu ajutorul ei pot fi extrase submulțimi de valori atât pe verticală (coloane), cât și pe orizontală (linii) din unul sau mai multe tabele.
- Sintaxa comenzii este simplă, apropiată de limbajul natural.

# Comanda *SELECT*

**SELECT** [ALL / DISTINCT] { \* / listă de attribute selectate / expr  
AS alias }

**FROM** { [schema.]{tabel [PARTITION (partition\_name)] /  
(subquery)} [alias\_tabel] }

[**WHERE** condiție]

[**START WITH** condiție]

[**CONNECT BY** condiție]

[**GROUP BY** listă de expresii [**HAVING** condiție]]

[**ORDER BY** {expresie / poziție / c\_alias} [**ASC** / **DESC**]] [**FOR  
UPDATE** [**OF** [schema.]{table / view}.coloană] [**NOWAIT**]



# Comanda *SELECT*

- Clauzele *START WITH* și *CONNECT BY* sunt utile pentru a construi cereri ierarhizate.
  - *START WITH* -> înregistrarea rădăcină a arborelui
  - *CONNECT BY* -> relația dintre coloanele (*părinte* și *copil*)
  - Prin folosirea operatorului *PRIOR* se poate face referință la înregistrarea *părinte*.
- Clauza *FOR UPDATE* permite blocarea coloanei (coloanelor) înainte de a actualiza sau șterge înregistrări din tabelele bazei de date. Prin folosirea clauzei *NOWAIT* se va genera o excepție și nu se va mai aștepta până la ridicarea blocajelor de pe înregistrări.

# Comanda *SELECT*

- Operatorii utilizați sunt:
  - operatori aritmetici (unari sau binari),
  - operatorul de concatenare ( || ),
  - operatorii de comparare (=, !=, ^=, < >, >, >=, <, <=, *IN* (echivalent cu =ANY, adică egal cu cel puțin una din valorile listei), *NOT IN* (echivalent cu !=ALL, adică diferit de toate elementele listei), *ALL*, [NOT] *BETWEEN* x *AND* y, [NOT] *EXISTS*, [NOT] *LIKE*, *IS* [NOT] *NULL*,
  - operatori logici (*NOT*, *AND*, *OR*).

# Comanda *SELECT*

- Limbajul permite prezența unor instrucțiuni *SELECT imbricate* în oricare din clauzele *SELECT*, *WHERE*, *HAVING* sau *FROM*.
- Instrucțiunile *SELECT* care apar în clauzele respective se numesc **subcereri**.
- În cazul folosirii subcererilor, pot fi utilizați:
  - operatorii *ALL*, *ANY*, *IN* (*=ANY*), *EXIST*, *NOT IN* (*!=ANY*), care sunt specifici cererilor ce returnează mai multe linii (*multiple-row subquery*)
  - operatorii de comparare *=*, *<*, *>*, *>=*, *<=*, *<>*, specifici cererilor care returnează o singură linie (*single-row subquery*).



# Comanda *SELECT*

- Executarea subcererilor se poate face:
  - fie **cu sincronizare** (corelat -> evaluarea subcererii face referință la o coloană a cererii principale și cererea interioară se execută pentru fiecare linie a cererii principale care o conține);
  - fie **fără sincronizare** (încuibărit -> se execută mai întâi cererea interioară, iar rezultatul ei este transmis cererii de nivel imediat superior).





# Comanda *SELECT*

## Cereri mono – relație

- Vezi exemple și întrebări în curs



# Comanda *SELECT*

- **Cereri mono – relație**
  - Clauza *GROUP BY*
  - Relații ierarhice
- **Cereri multi – relație**
  - Operatori pe mulțimi
  - Operații de compunere
- **Subcereri**
- **Clauza WITH**
- **Subcereri scalare**
- **ROLLUP/CUBE/GROUPING SETS**
- **Funcții**

-> Vezi exemple și întrebări în curs

# Comanda *INSERT*

**INSERT INTO** *nume\_tabel* | *nume\_view* [(*col1*[, *col2*[,...]])]

**VALUES** (*expresia1*[, *expresia2*[,...]]) | *subcerere*;

- *expresia1*, *expresia2* – reprezintă expresii a căror evaluare este atribuită coloanelor precizate (se inserează o linie);
- *subcerere* – reprezintă o interogare (se inserează una sau mai multe linii).

# Comanda *INSERT*

- Dacă lipsește specificația coloanelor se consideră că sunt completate **toate câmpurile** tabelului sau vizualizării.
- Dacă se introduc date doar în anumite coloane, atunci aceste coloane trebuie specificate. În restul coloanelor se introduce automat **null (daca nu exista DEFAULT)**.
- Specificarea cererii din comanda *INSERT* determină **copierea unor date dintr-un tabel în altul** pe atâtea linii câte au rezultat din cerere.
- Dacă se introduc numai anumite câmpuri într-o înregistrare, atunci printre acestea trebuie să se găsească **câmpurile cheii primare**.
- Pentru a putea executa comanda *INSERT* este necesar ca utilizatorul care execută această instrucțiune să aibă **privilegiul de a insera** înregistrări în tabel sau în vizualizare.

-> Vezi exemple în curs

# Comanda *DELETE*

## *DELETE*

[*FROM*] *tablename / viewname* [*AS* *alias*] [*WHERE* *condiție*]  
[*clauza\_returning*]

- Comanda *DELETE* nu șterge **structura** tabelului.
- Pentru a se putea executa instrucțiunea *DELETE*, utilizatorul care o lansează în execuție trebuie să aibă acest **privilegiu**.
- În clauza *WHERE* pot fi folosite și **subcereri**.
- Comanda nu poate fi folosită pentru ștergerea valorilor unui câmp individual. Acest lucru se poate realiza cu ajutorul comenzii *UPDATE*.

-> Vezi exemple în curs

# Comanda *UPDATE*

**UPDATE** *tablename* | *viewname*  
**SET** (*column1*[,*column2*[,...]]) = (*subquery*) | *column* = *expr* /  
(*query*)  
[**WHERE** *condition*]

- Pentru a se putea executa instrucțiunea *UPDATE*, utilizatorul care o lansează în execuție trebuie să aibă acest **privilegiu**.
- Dacă nu este specificată clauza *WHERE* se vor modifica toate liniile.
- Cererea trebuie să furnizeze un **număr de valori** corespunzător numărului de coloane din paranteza care precede caracterul de egalitate.

-> Vezi exemple în curs



# LIMBAJUL PENTRU CONTROLUL DATELOR

- Controlul unei baze de date cu ajutorul *SQL*-ului se refera la:
  - asigurarea confidentialitatii si securitatii datelor;
  - organizarea fizica a datelor;
  - realizarea unor performante;
  - reluarea unor actiuni in cazul unei defectiuni;
  - garantarea coerentei datelor in cazul prelucrarii concurente.



# LIMBAJUL PENTRU CONTROLUL DATELOR

- Sistemul de gestiune trebuie:
  - să pună la dispoziția unui număr mare de utilizatori o mulțime **coerentă** de date;
  - să garanteze **coerența** datelor în cazul manipulării simultane de către diferiți utilizatori.





# LIMBAJUL PENTRU CONTROLUL DATELOR

- Coerența este asigurată cu ajutorul conceptului de **tranzacție**.
- Tranzacția este unitatea logică de lucru constând din una sau mai multe instrucțiuni *SQL*, care trebuie să fie executate **atomic** (ori se execută toate, ori nu se execută nici una!), asigurând astfel trecerea BD dintr-o stare coerentă în altă stare coerentă.



# LIMBAJUL PENTRU CONTROLUL DATELOR

## Proprietățile tranzacțiilor

- Atomicitate
- Consistență
- Izolare
- Durabilitate

->ACID

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Dacă toate operațiile ce constituie tranzacția sunt executate și devin efective, spunem că tranzacția este **validată**, iar modificările aduse de tranzacție devin **definitive**.
- Dacă dintr-un motiv sau altul (neverificarea condițiilor, accesul imposibil) o operație a tranzacției nu a fost executată spunem că tranzacția a fost **anulată**.  
Modificările aduse de toate operațiile tranzacției anulate sunt și ele **anulate** și se revine la **starea bazei de date de dinaintea tranzacției anulate**.

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Este posibil ca o tranzacție să fie descompusă în **subtranzacții**, astfel încât dacă este necesar să se anuleze doar parțial unele operații.
- Fiecare tranzacție se poate termina:
  - “normal” (*commit*);
  - “anormal” (*rollback*).
- Controlul tranzacțiilor constă în:
  - definirea începutului și sfârșitului unei tranzacții,
  - validarea sau anularea acesteia,
  - eventuală descompunere în subtranzacții.



# LIMBAJUL PENTRU CONTROLUL DATELOR

- Limbajul pentru controlul datelor (*LCD*) permite salvarea informației, realizarea fizică a modificărilor în baza de date, rezolvarea unor probleme de concurență.
- Limbajul conține următoarele instrucțiuni:
  - *COMMIT* - folosită pentru permanentizarea modificărilor executate asupra BD (modificările sunt înregistrate și sunt vizibile tuturor utilizatorilor);
  - *ROLLBACK* - folosită pentru refacerea stării anterioare a BD (sunt anulate toate reactualizările efectuate de la începutul tranzacției);
  - *SAVEPOINT* - folosită în conjuncție cu instrucțiunea *ROLLBACK*, pentru definirea unor puncte de salvare în fluxul programului.



# LIMBAJUL PENTRU CONTROLUL DATELOR

- O **tranzacție** constă:
  - dintr-o singură instrucțiune *LDD*;
  - dintr-o singură instrucțiune *LCD*;
  - din instrucțiuni *LMD* care fac schimbări consistente în date.

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Tranzacția **începe**:
  - după o comandă *COMMIT*,
  - după o comandă *ROLLBACK*,
  - după conectarea inițială la Oracle,
  - când este executată prima instrucțiune *SQL*.
- Tranzacția se **termină**:
  - dacă sistemul cade;
  - dacă utilizatorul se deconectează;
  - dacă se dau comenzile *COMMIT* sau *ROLLBACK* ;
  - dacă se execută o comandă *LDD*.
- După ce se termină o tranzacție, prima instrucțiune *SQL* executabilă va genera automat începutul unei noi tranzacții.

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Din momentul în care s-a executat instrucțiunea **COMMIT**, BD s-a modificat (permanent) în conformitate cu instrucțiunile *SQL* executate în cadrul tranzacției care tocmai s-a terminat. Din acest punct începe o nouă tranzacție.
- Dacă se folosește utilitarul *SQL\*Plus*, există posibilitatea ca după fiecare comandă *LMD* să aibă loc o permanentizare automată a datelor (un *COMMIT* implicit). Acest lucru se poate realiza folosind comanda:

*SET AUTO[COMMIT] {ON / OFF}*

- Comanda **ROLLBACK** permite restaurarea unei stări anterioare a BD.

*ROLLBACK [TO [SAVEPOINT] savepoint];*





# LIMBAJUL PENTRU CONTROLUL DATELOR

- Dacă nu se specifică nici un *savepoint*, **toate modificările** făcute în tranzacția curentă sunt anulate
- Dacă se specifică un anumit *savepoint*, atunci doar modificările **de la acel *savepoint*** până în momentul respectiv sunt anulate.
- Executarea unei instrucțiuni *ROLLBACK* presupune terminarea tranzacției curente și începerea unei noi tranzacții.

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Punctele de salvare pot fi considerate ca niște etichete care referă o submulțime a schimbărilor dintr-o tranzacție, marcând efectiv un punct de salvare pentru tranzacția curentă. Punctele de salvare **NU sunt obiecte ale schemei**. Prin urmare, nu sunt referite în DD.
- *Server-ul Oracle* implementează **un punct de salvare implicit** pe care îl mută automat după ultima comandă *LMD* executată.
- Dacă este creat un punct de salvare având același nume cu unul creat anterior, cel definit anterior este șters automat.

*SAVEPOINT savepoint;*

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Starea datelor înainte de *COMMIT* sau *ROLLBACK* este următoarea:
  - starea anterioară a datelor poate fi **recuperată**;
  - utilizatorul curent **poate vizualiza** rezultatele operațiilor *LMD* prin interogări asupra tabelelor;
  - alți utilizatori **nu pot vizualiza** rezultatele comenzilor *LMD* făcute de utilizatorul curent (***read consistency***);
  - înregistrările (liniile) afectate sunt **blocate** și, prin urmare, alți utilizatori nu pot face schimbări în datele acestor înregistrări.

# LIMBAJUL PENTRU CONTROLUL DATELOR

- Execuția unei comenzi *COMMIT* implică anumite modificări.
  - Toate schimbările (*INSERT*, *DELETE*, *UPDATE*) din baza de date făcute după anterioara comandă *COMMIT* sau *ROLLBACK* sunt **definitive**. Comanda se referă numai la schimbările făcute de utilizatorul care dă comanda *COMMIT*.
  - Toate punctele de salvare vor fi **șterse**.
  - Starea anterioară a datelor este **pierdută definitiv**.
  - Toți utilizatorii **pot vizualiza** rezultatele.
  - Blocările asupra liniilor afectate sunt **eliberate**; liniile pot fi folosite de alți utilizatori pentru a face schimbări în date.



# LIMBAJUL PENTRU CONTROLUL DATELOR

- Execuția unei comenzi *ROLLBACK* implică anumite modificări.
  - **Anulează** tranzacția în curs și toate modificările de date făcute după ultima comandă *COMMIT*.
  - Sunt **eliberate** blocările liniilor implicate.

**Consistența la citire** -> vezi curs

# NoSQL

- Bazele de date *NoSQL* (*Not only SQL*) au apărut ca răspuns la necesitatea de a oferi **scalabilitate** și **răspuns rapid la interogări**, permițând totodată efectuarea de **modificări** frecvente la nivelul aplicației.
- Prin termenul de bază de date *NoSQL* se face referire la orice tip de bază de date nerelațională.
- Bazele de date relaționale sunt, la ora actuală, cunoscute ca baze de date de tip *SQL*, datorită denumirii limbajului standard de operare în cadrul acestor baze de date.
  - Aceste baze de date au fost dezvoltate centrat pe ideea de reducere a duplicării datelor, lucru explicabil în contextul costului mare de stocare de la momentul respectiv.



# NoSQL



- Bazele de date relaționale (*SQL*) au structuri mai rigide, iar scalabilitatea schemelor este costisitoare.
- În bazele de date *NoSQL* datele sunt stocate în **structuri diferite de tabelele relaționale**. Din acest punct de vedere, bazele de date NoSQL pot fi de următoarele tipuri:
  - Document
  - Cheie-valoare
  - Coloane dinamice
  - Graf



# NoSQL



- Schemele obținute sunt **flexibile** și **scalabile** chiar și în condițiile unui volum mare de date și utilizatori.
- Modelarea nerelațională a datelor corelate este mai simplă decât în bazele de date *SQL* deoarece **datele conectate prin relații nu mai trebuie să fie distribuite în diferite tabele.**
- Modelele de date *NoSQL* permit stocarea datelor corelate în cadrul unei singure structuri de date.
- Bazele de date *NoSQL* au apărut atunci când **costul stocării datelor a scăzut semnificativ**, iar reducerea duplicării datelor nu a mai fost la fel de importantă precum în trecut.





# NoSQL



- Odată cu reducerea costului stocării, a crescut rapid **cantitatea de date** pe care aplicațiile trebuia să le stocheze și să le interogheze.
- Aceste date au diferite formate și dimensiuni, putând fi **structurate, semi-structurate sau polimorfe**.
- Definirea preliminară a unei scheme care să găzduiască astfel de date este, de multe ori, **imposibilă**. Bazele de date *NoSQL* permit stocarea unor cantități uriașe de date nestructurate, oferindu-le flexibilitate.

# NoSQL

- În același timp cu apariția bazelor de date *NoSQL*, a crescut în popularitate *manifestul Agile*, care a determinat *reconsiderarea modului în care sunt dezvoltate aplicațiile software*, ca urmare a necesității de adaptare rapidă a acestora la *modificarea frecventă a cerințelor*.
  - Astfel, a fost necesar ca dezvoltatorii să poată itera rapid și să opereze modificări prin întreaga « stivă » *software*, până la modelul bazei de date. Bazele de date *NoSQL* au fost cele care au oferit această flexibilitate.
- De asemenea, *Cloud computing* a devenit tot mai popular, iar dezvoltatorii au început să utilizeze *cloud-urile publice* pentru găzduirea aplicațiilor și a datelor.
  - Astfel, a apărut necesitatea de a distribui datele pe mai multe servere, din diferite regiuni, pentru ca aplicațiile să fie fiabile și datele să poată fi stocate în funcție de *geolocalizare*.

# NoSQL

## Baze de date de tip document

- Aceste baze de date stochează datele în documente similare obiectelor JSON (*JavaScript Object Notation*).
- Fiecare document conține perechi de câmpuri și valori asociate.
- Valorile pot fi de diferite tipuri (șiruri de caractere, numere, valori boolene, vectori, obiecte), iar structurile sunt, de regulă, similare obiectelor cu care dezvoltatorii lucrează în aplicație.
- Datorită diversității tipurilor de date ale valorilor și a existenței unor limbaje puternice de interogare, bazele de date de tip document sunt potrivite în cele mai multe cazuri de utilizare. Exemplu: *MongoDB*.

# NoSQL

JSON

```
1  [
2    {
3      "year" : 2013,
4      "title" : "Turn It Down, Or Else!",
5      "info" : {
6        "directors" : [ "Alice Smith", "Bob Jones"],
7        "release_date" : "2013-01-18T00:00:00Z",
8        "rating" : 6.2,
9        "genres" : ["Comedy", "Drama"],
10       "image_url" : "http://ia.media-imdb.com/images/N/09ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
11       "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
12       "actors" : ["David Matthewman", "Jonathan G. Neff"]
13     }
14   },
15   {
16     "year": 2015,
17     "title": "The Big New Movie",
18     "info": {
19       "plot": "Nothing happens at all.",
20       "rating": 0
21     }
22   }
23 ]
```

<https://aws.amazon.com/nosql/document/>

# NoSQL

## Baze de date de tip cheie-valoare

- Sunt baze de date mai simple, în care fiecare element (*item*) conține chei și valori.
- O valoare va putea fi regăsită prin intermediul cheii sale, deci modul de interogare este simplu.
- Bazele de date cheie-valoare sunt potrivite cazurilor de utilizare în care este necesară stocarea unor volume mari de date, fără a fi necesară efectuarea de interogări complexe asupra acestora.
- Exemplele de utilizare includ stocarea preferințelor utilizatorilor sau stocarea de tip *cache*. Exemple: *Redis*, *DynamoDB*.

# NoSQL

## Baze de date de tip coloane dinamice (*wide column*)

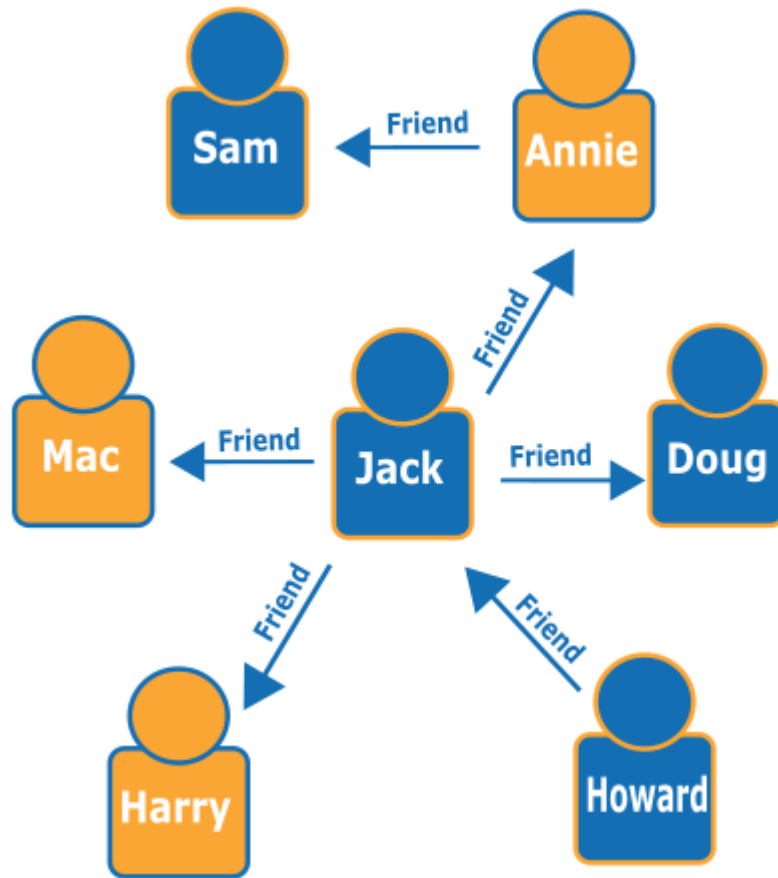
- Aceste baze de date stochează datele în tabele, rânduri și coloane dinamice.
- Stocarea de acest tip este mai flexibilă decât cea oferită de bazele de date relaționale, deoarece **nu este necesar ca fiecare linie să aibă aceleași coloane**.
- Aceste baze de date pot fi privite ca baze de date cheie-valoare bidimensionale.
- Bazele de date *wide-column* sunt adecvate atunci când se stochează volume mari de date și se poate face o predicție asupra tipurilor de interogări.
- Exemplele de utilizare includ stocarea datelor IoT și a profilurilor de utilizatori. Exemple: *Cassandra*, *Hbase*.

# NoSQL

## Bazele de date de tip graf

- Aceste baze de date stochează datele în noduri și muchii.
- Nodurile stochează informații precum cele despre persoane, locuri, obiecte, evenimente, iar muchiile stochează informații despre relațiile dintre noduri.
- Bazele de date de acest tip sunt adecvate în cazurile în care este necesară **traversarea relațiilor pentru a determina *pattern*-uri**.
- Exemplele de utilizare includ rețele sociale, detectarea fraudelor și motoarele de recomandări.
- Exemple: *Neo4J*, *JanusGraph*.

# NoSQL



<https://aws.amazon.com/nosql/graph/>