BAZE DE DATE

CURS 12

Limbajul de prelucrare a datelor

- *SQL* furnizează comenzi ce permit consultarea (*SELECT*) şi actualizarea (*INSERT*, *UPDATE*, *DELETE*, *MERGE*) conţinutului 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 submulţimi de valori atât pe verticală (coloane), cât şi pe orizontală (linii) din unul sau mai multe tabele.
- Sintaxa comenzii este simplă, apropiată de limbajul natural.

Comanda *SELECT*

SELECT [*ALL | DISTINCT*] {* | *listă de atribute selectate | expr AS alias*}
FROM { [*schema.*]{*tabel* [*PARTITION* (*partition_name*)] |
(*subquery*)} [*alias_tabel*] }
[***WHERE*** *condiţie*]
[***START WITH*** *condiţie*]
[***CONNECT BY*** *condiţie*]
[***GROUP BY*** *listă de expresii* [***HAVING*** *condiţie*]]
[***ORDER BY*** {*expresie | poziţie | c_alias*} [***ASC*** | ***DESC***]] [***FOR UPDATE*** [***OF*** [*schema.*]{*table | view*}.*coloană*] [***NOWAIT***]

Comanda *SELECT*

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

Comanda *SELECT*

- Operatorii utilizaţi sunt:
  - operatori aritmetici (unari sau binari),
  - operatorul de concatenare (||),
  - operatorii de comparare (=, !=, ^=, < >, >, >=, <, <=, *IN* (echivalent cu =ANY, adică egal cu cel puţin 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 prezenţa unor instrucţiuni *SELECT* **imbricate** în oricare din clauzele *SELECT*, *WHERE*, *HAVING* sau *FROM.*
- Instrucţiunile *SELECT* care apar în clauzele respective se numesc **subcereri**.
- În cazul folosirii subcererilor, pot fi utilizaţi:
  - 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 referinţă la o coloană a cererii principale şi cererea interioară se execută pentru fiecare linie a cererii principale care o conţine);
  - fie **fără sincronizare** (încuibărit -> se execută mai întâi cererea interioară, iar rezultatul ei este transmis cererii de nivel imediat superior).

Comanda *SELECT*

Cereri mono – relaţie

- Vezi exemple și întrebări în curs

Comanda *SELECT*

- **Cereri mono – relaţie**
  - **Clauza *GROUP BY***
  - **Relaţii ierarhice**
- **Cereri multi – relaţie**
  - **Operatori pe mulţimi**
  - **Operaţii 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 lipseşte specificaţia 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 instrucţiune 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*** *condiţie*] [*clauza_returning*]

- Comanda *DELETE* nu şterge structura tabelului.
- Pentru a se putea executa instrucţiunea *DELETE*, utilizatorul care o lansează în execuţie trebuie să aibă acest privilegiu.
- În clauza *WHERE* pot fi folosite şi subcereri.
- Comanda nu poate fi folosită pentru ştergerea 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 instrucţiunea *UPDATE*, utilizatorul care o lansează în execuţie 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 dispoziţia unui număr mare de utilizatori o mulţime coerentă de date;
  - să garanteze coerenţa datelor în cazul manipulării simultane de către diferiţi utilizatori.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Coerenţa este asigurată cu ajutorul conceptului de **tranzacţie**.
- Tranzacţia este unitatea logică de lucru constând din una sau mai multe instrucţiuni *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

- **A**tomicitate
- **C**onsistență
- **I**zolare
- **D**urabilitate

->ACID

LIMBAJUL PENTRU CONTROLUL DATELOR

- Dacă toate operaţiile ce constituie tranzacţia sunt executate şi devin efective, spunem că tranzacţia este **validată**, iar modificările aduse de tranzacţie devin definitive.
- Dacă dintr-un motiv sau altul (neverificarea condiţiilor, accesul imposibil) o operaţie a tranzacţiei nu a fost executată spunem că tranzacţia a fost **anulată**. Modificările aduse de toate operaţiile tranzacţiei anulate sunt şi ele anulate şi se revine la starea bazei de date de dinaintea tranzacţiei anulate.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Este posibil ca o tranzacţie să fie descompusă în **subtranzacţii**, astfel încât dacă este necesar să se anuleze doar parţial unele operaţii.
- Fiecare tranzacţie se poate termina:
  - "normal" (*commit*);
  - "anormal" (*rollback*).
- Controlul tranzacţiilor constă în:
  - definirea începutului şi sfârşitului unei tranzacţii,
  - validarea sau anularea acesteia,
  - eventuală descompunere în subtranzacţii.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Limbajul pentru controlul datelor (*LCD*) permite salvarea informaţiei, realizarea fizică a modificărilor în baza de date, rezolvarea unor probleme de concurenţă.
- Limbajul conţine următoarele instrucţiuni:
  - *COMMIT* - folosită pentru permanentizarea modificărilor executate asupra BD (modificările sunt înregistrate şi sunt vizibile tuturor utilizatorilor);
  - *ROLLBACK* - folosită pentru refacerea stării anterioare a BD (sunt anulate toate reactualizările efectuate de la începutul tranzacţiei);
  - *SAVEPOINT* - folosită în conjuncţie cu instrucţiunea *ROLLBACK*, pentru definirea unor puncte de salvare în fluxul programului.

LIMBAJUL PENTRU CONTROLUL DATELOR

- O tranzacţie constă:
  - dintr-o singură instrucţiune *LDD*;
  - dintr-o singură instrucţiune *LCD*;
  - din instrucţiuni *LMD* care fac schimbări consistente în date.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Tranzacţia **începe**:
  - după o comandă *COMMIT*,
  - după o comandă *ROLLBACK*,
  - după conectarea iniţială la Oracle,
  - când este executată prima instrucţiune *SQL*.
- Tranzacţia 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 tranzacţie, prima instrucţiune *SQL* executabilă va genera automat începutul unei noi tranzacţii.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Din momentul în care s-a executat instrucţiunea ***COMMIT***, BD s-a modificat (permanent) în conformitate cu instrucţiunile *SQL* executate în cadrul tranzacţiei care tocmai s-a terminat. Din acest punct începe o nouă tranzacţie.
- Dacă se foloseşte 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 tranzacţia curentă sunt anulate
- Dacă se specifică un anumit *savepoint*, atunci doar modificările de la acel *savepoint* până în momentul respectiv sunt anulate.
- Executarea unei instrucţiuni *ROLLBACK* presupune terminarea tranzacţiei curente şi începerea unei noi tranzacţii.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Punctele de salvare pot fi considerate ca nişte etichete care referă o submulţime a schimbărilor dintr-o tranzacţie, marcând efectiv un punct de salvare pentru tranzacţia curentă. Punctele de salvare NU sunt obiecte ale schemei. Prin urmare, nu sunt referite in 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 acelaşi nume cu unul creat anterior, cel definit anterior este şters 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 operaţiilor *LMD* prin interogări asupra tabelelor;
  - alţi utilizatori nu pot vizualiza rezultatele comenzilor *LMD* făcute de utilizatorul curent (***read consistency***);
  - înregistrările (liniile) afectate sunt blocate şi, prin urmare, alţi utilizatori nu pot face schimbări în datele acestor înregistrări.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Execuţia 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 şterse.
  - Starea anterioară a datelor este pierdută definitiv.
  - Toţi utilizatorii pot vizualiza rezultatele.
  - Blocările asupra liniilor afectate sunt eliberate; liniile pot fi folosite de alţi utilizatori pentru a face schimbări în date.

LIMBAJUL PENTRU CONTROLUL DATELOR

- Execuţia unei comenzi *ROLLBACK* implică anumite modificări.
  - Anulează tranzacţia în curs şi 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 tabeleModelele 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 polimorfice.
- 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 acelasi 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 poata itera rapid și să opereze modificări prin intreaga « 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 aplicatiile 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

```
[
    {
        "year" : 2013,
        "title" : "Turn It Down, Or Else!",
        "info" : {
            "directors" : [ "Alice Smith", "Bob Jones"],
            "release_date" : "2013-01-18T00:00:00Z",
            "rating" : 6.2,
            "genres" : ["Comedy", "Drama"],
            "image_url" : "http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
            "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
            "actors" : ["David Matthewman", "Jonathan G. Neff"]
        }
    },
    {
        "year": 2015,
        "title": "The Big New Movie",
        "info": {
            "plot": "Nothing happens at all.",
            "rating": 0
        }
    }
]
```

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 necesare stocarea unor volume mari de date, fără a fi necesară efectuarea de interogări complexe asupra acestora.
- Exemplele de utilizare includ stocarea preferințelor utlizatorilor sau stocarea de tip *cache*. Exemple: *Redis*, *DynanoDB*.

NoSQL

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

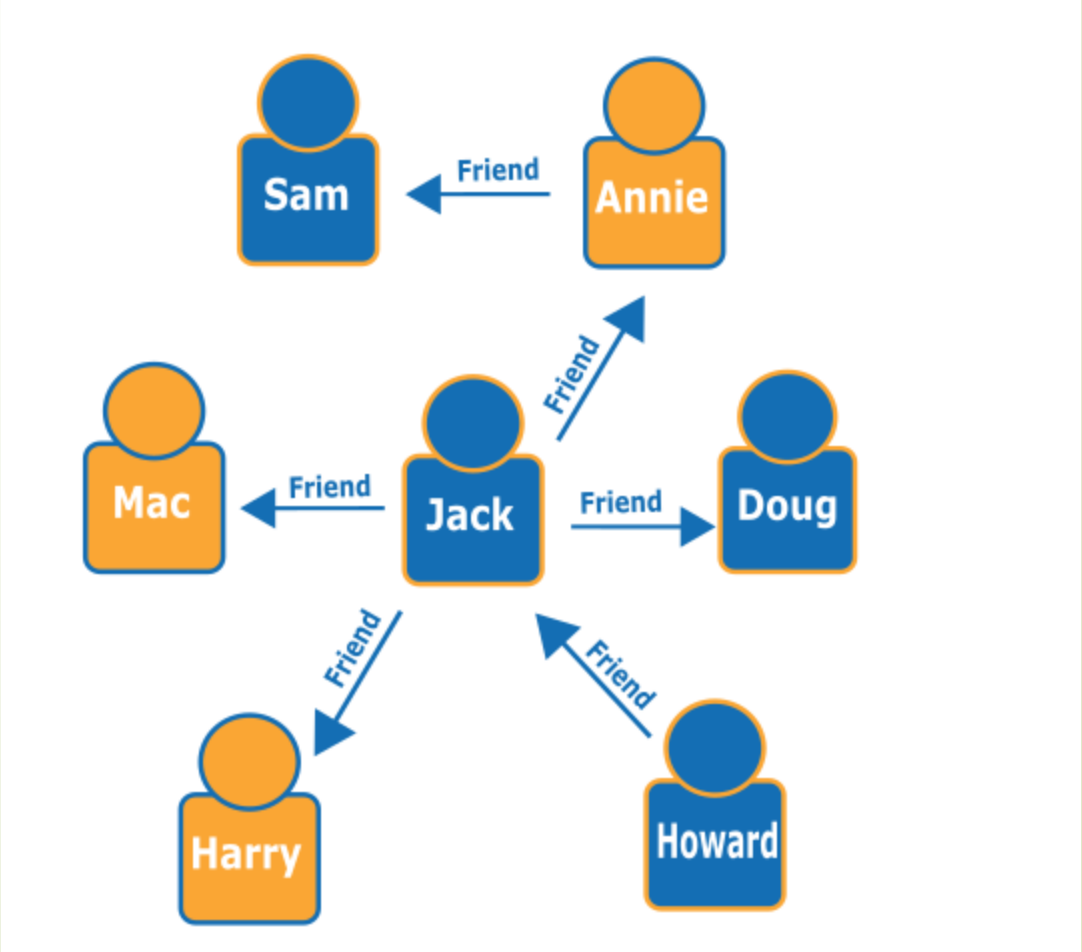
- Aceste baze de date stochează datele în tabele, rânduri și coloane dinamice.
- Stocarea de acest tip este mai flexibilă decât cea oferită de bazele de date relaționale, deoarece nu este necesar ca fiecare linie să aibă aceleași coloane.
- Aceste baze de date pot fi privite ca baze de date cheie-valoare 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/