

Baze de Date

Curs introductiv

Gabriel Dragomir

Obiective

- La final vor fi dobândite următoarele deprinderi:
 1. Să implementeze modele de date necesare proiectării conceptuale a unei baze de date
 2. Să implementeze o bază de date pentru un SGBD relațional conform unui set de cerințe textuale sau specificații tehnice, să implementeze scripturi pentru exploatarea bazei de date, pe baza unui set de cerințe generale, să conceapă și să optimizeze interogări pentru o bază de date folosind sintaxa limbajelor relaționale (SQL și dialecte de ex. Transact-SQL sau PL-SQL)

Obiective (continuare)

3. Să adopte cea mai bună soluție pentru normalizarea schemei unei baze de date în vederea realizării unei proiectări optimale a unei baze de date pentru anumite clase de probleme.
4. Să utilizeze un mediu de lucru integrat evoluat pentru implementarea și programarea aplicațiilor cu baze de date la nivel BD (SQL Developer - Oracle, SQL Server Management Studio, MySQL Workbench)
5. Să utilizeze un limbaj specific pentru realizarea unei aplicații cu baze de date (aplicație PHP conectată via http la o bază de date)

Conținut

- Introducere în Baze de Date. Sisteme de Gestiune Baze de Date. Arhitectura Sistemelor de Gestiune Baze de Date.
- Structured Query Language - SQL.
- Constrângerile; Vederi.
- Aplicații cu Baze de Date.
- Modelul Relațional. Algebra Relațională.

Conținut

- Calcul Relațional.
- Elemente de proiectare Baze de Date. Forme Normale.
- Modelul “Entity – Relationship”.
- Administrare Baze de Date. Indecși. Securitate. Tranzacții.
- XML.
- NoSQL.

Bibliografie

- G.C. Dragomir-Loga, Utilizarea Bazelor de Date Relaționale, Editura UTPPRESS, 2011
- Al. Lelutiu, Perenitatea conceptelor de baze de date
- R. Ramakrishnan, J. Gerhrke, Database Management Systems, McGraw Hill, 2002
- J. Ullman, H.G. Molina, J. Widom, Database Systems, Prentice Hall, 2008
- C. J. Date, An Introduction to Database Systems, 8th edition, Pearson Education, 2004
- R. Dollinger, Baze de Date si Gestiunea Tranzactiilor, Ed. Albastra, 1998
- D. Burdescu, A.I Ionescu, L. Stanescu, Baze de Date, Editura Universitaria Craiova, 2004

Bibliografie

- R. Dollinger, Utilizarea sistemului SQL Server (SQL 7.0, SQL 2000) Ed. Albastra 2001
- Ryan K. Stephens, Ronald Plew, Bryan Morgan, Jeff Perkins - QUE - Teach Yourself SQL in 21 Days
- Philip Greenspun - SQL for Web Nerds
- <http://www.sqlzoo.net>
- R. Riordan, Designing Effective User Interface for Database Systems, Addison Wesley, 2005
- L. Welling and L. Thomson, PHP and MySQL Web Development, Pearson Education, 2005

Bibliografie

- La biblioteca UTCN, ediții mai vechi (în paranteză, cota), în special pentru Algebra Relațională și Calcul Relațional:
 - Date C. J., An Introduction to database systems (461.35511)
 - Dollinger Robert, Baze de Date (477.431)

Structura cursului

- 14 săptămâni
- 2 ore curs, 2 ore laborator + 4 ore pregătire individuală
- Examinare pe parcurs
 - Laborator
 - Prima parte – înainte de vacanța de iarnă (CPL)
 - A doua parte – după vacanța de iarnă (CFL)
- Examen final (E)
 - Teorie și probleme (materia de la curs)
 - În sesiune, online

Ponderea notelor

- Nota finală = $0,4 * N1 + 0,6 * N2$
- $N1 = 0,6 * CPL + 0,4 * CFL + [TC]$
 - CPL reprezintă nota la colocviul parțial de laborator (prima parte)
 - CFL reprezintă nota la colocviul final de laborator (a doua parte)
 - TC reprezintă bonus pentru rezolvare teme de casă optionale (teme primite la laborator din cartea: G.C. Dragomir-Loga, Utilizarea Bazelor de Date Relaționale, Editura UTPRESS, 2011)
- $N2 = 0,9 * E + PC + [TS]$
 - E reprezintă nota la examenul din sesiune (online)
 - PC reprezintă nota pentru prezența la curs (rezolvare quizuri corespunzătoare fiecărui curs)
 - TS reprezintă bonus pentru rezolvare quizuri Oracle Academy
- Conditii de participare la examenul final (E): $N1 \geq 5$
- Condiție de promovare: $E \geq 5$; Nota finală ≥ 5

Tema pentru colocviul parțial de laborator (EPL)

Se va crea schema unei baze de date folosind SGBD Oracle (obligatoriu), se va popula BD cu date relevante pentru interogările și actualizările de pe subiect.

Livrabile:

- Script SQL care pe lângă rezolvarea cerințelor conține și popularea cu date.

Tema pentru colocviul parțial de laborator (EPL)

Modul de notare:

- Creare tabele (fiecare subiect conține în jur de 4 tabele) – 1,5 p;
- Definire constrângeri de tip cheie primară, cheie străină – 1p;
- Modificarea structurii unei tabele – 0,5p;
- 2 Constrângeri la nivel atribut, tuplă – 1p;
- 2 Interogări simple (1 tabelă) – 1p;
- 2 Interogări de tip JOIN (2 sau mai multe tabele, obligatoriu cu JOIN) – 1p;
- 2 Interogări complexe (implică obligatoriu anumiți operatori cu subquery) – 1p;
- 2 Interogări cu funcții de agregare (MIN, MAX, COUNT, AVG eventual combinat cu alte clauze) – 1p;
- 3 Operații de actualizare (INSERT, UPDATE, DELETE) – 2p;
- 3 Triggere (din care unul de tip INSTEAD OF) – 3p.

TOTAL 13p

Tema pentru colocviul final de laborator

Se va implementa o aplicație web corespunzătoare subiectului „Colocviu parțial laborator BD” primit la prima lucrare de laborator.

Cerințe:

- BD implementată cu SGBD ORACLE, MySql sau MSSQL
- La nivel aplicatie (UI) implementare cu PHP (optional cu Javascript, CSS) sau cu Java sau cu C#
- Se vor implementa numai interogările (subpunctele 3-6) de pe biletul „Colocviu parțial laborator BD”. Pentru o notare cât mai bună se vor respecta cerințele de detaliu pentru interfața utilizator.
- Trebuie să se folosească una sau mai multe funcții definite de utilizator în schema BD (proceduri stocate). Deci cel puțin într-o interogare codul PHP trebuie să apeleze o procedură stocată (sau funcție utilizator „UDF” – din BD).

Tema pentru colocviul final de laborator

Interfață utilizator:

Exemplu: Presupunând că pe biletul „Colocviu Parțial de laborator BD” interogarea este:

SELECT last_name, first_name, salary FROM employees WHERE department_id = 101

Interfață cu utilizatorul în prima fază citește department_id, ca în figura de mai jos:

The figure shows a user interface element consisting of a white rectangular box with a thin black border. Inside, there is a text input field containing the text "Employee ID: [101]" in a black monospace font. Below the input field is a blue rectangular button with the text "<Cauta>" in white. The entire interface is set against a light blue background.

Tema pentru colocviul final de laborator

Iar în faza a doua interfață cu utilizatorul afișează rezultatul:

| Last name | First name | Salary |
|-----------|------------|--------|
| Higgins | John | 5000 |

<Revenire>

Se va ține cont la notă de existența unui meniu în aplicație (nu se accesează interogările una câte una ca pagini web separate, ci se integrează cele șase interogări într-o aplicație cu mai multe pagini web).

Tema pentru colocviul final de laborator

Livrabile:

- Opțional: Scripturi SQL pentru creare tabele și populare cu date (dacă implementarea BD s-a făcut cu MySql sau MSSQL);
- Script SQL pentru funcții UDF / proceduri stocate;
- Scripturi PHP (+HTML, Javascript, CSS, etc).

Modul de notare (Total 10p):

1p din oficiu, 2p utilizare funcție UDF/procedură stocată, 4p funcționare corectă interogări (d.p.d.v. PHP), 3p aspect interfață.

Nota pentru activitatea de laborator se obține prin susținerea practică, în fața calculatorului, a celor două componente (parțial și final) la momentele de timp planificate ce vor fi anunțate

Examen final teorie (EF, 90%NE)

- 10 întrebări teoretice (răspuns dat de student) – 10p;
- 2 sau 3 întrebări de tip problemă din dependențe funcționale și normalizare – 2p;
- 3 interogări pe o schemă de BD cu 3 – 4 tabele (total 15p)
 - Fiecare interogare trebuie rezolvată cu:
 - SQL - 1p per interogare
 - Algebra relațională – 1p per interogare
 - Calcul Relațional al Tuplelor – 1p per interogare
 - Calcul Relațional al Domeniilor – 1p per interogare
 - Se mai acordă 1p per interogare pentru desenarea Arborelui operator;
- Problemă care specifică în limbaj natural cerințele pentru o bază de date (total 7p)
 - Diagrama ER – 5p;
 - Document XML care reflectă diagrama ER – 1p;
 - Interogare XQUERY – 1p.

Total 34p

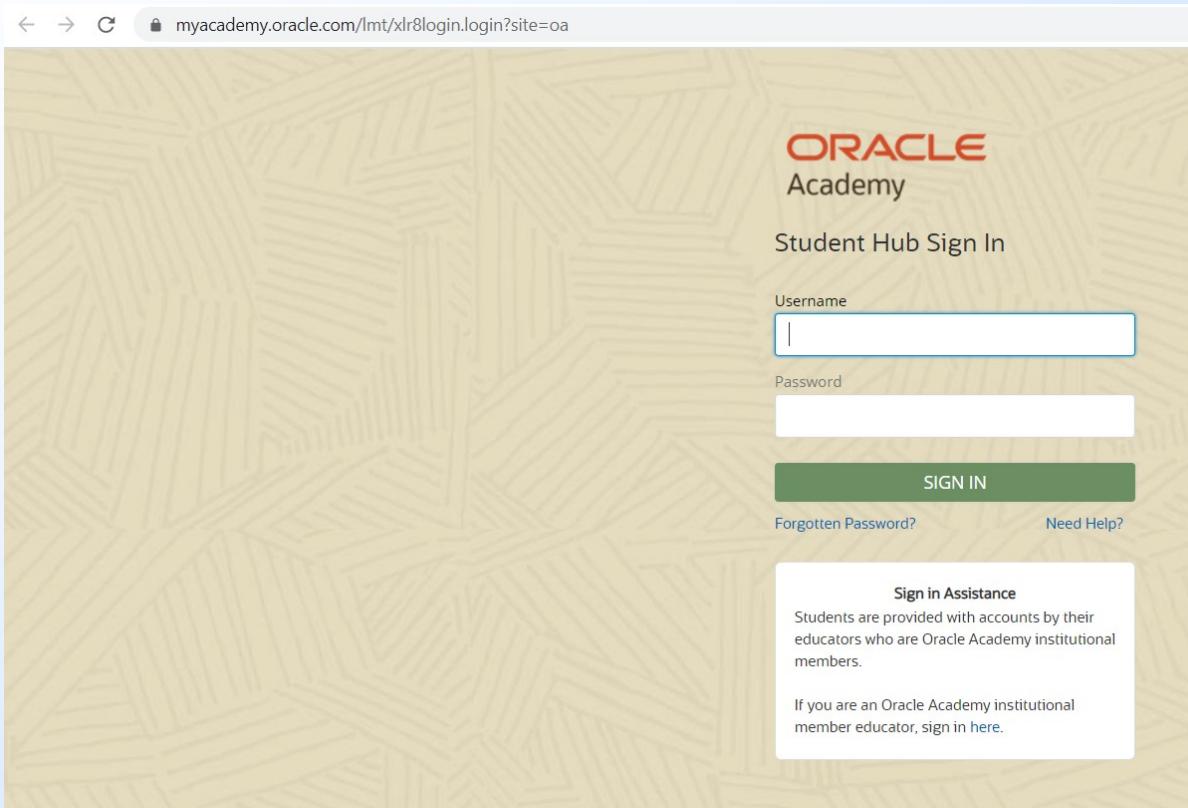
Cerințe obligatorii

- Examen parțial laborator – minim 6,5p (din 13p)
- Examen final laborator – minim 5p (din 10p)
- Examen final teorie – minim 17p (din 34p)

Lucrări de laborator

- Se lucrează pe platforma iLearning Oracle, cu acces independent pentru:
 - Materiale de studiu
 - Conțin Quiz-uri ce contează pentru TS la nota examen teorie
 - Lucrări practice SQL (APEX 5)
 - Lucrări practice PL-SQL (APEX 5)

Lucrări de laborator



- Acces la platforma Oracle iLearning pentru materialele de studiu: <https://academy.oracle.com> unde se caută SIGN IN în colțul dreapta sus și se efectuează click pe butonul SIGN IN TO STUDENT HUB
- Utilizatorul și Parola vor fi comunicate la prima întâlnire, fiecărui student

Lucrări de laborator

□ Acces platforma Oracle iLearning pentru APEX:

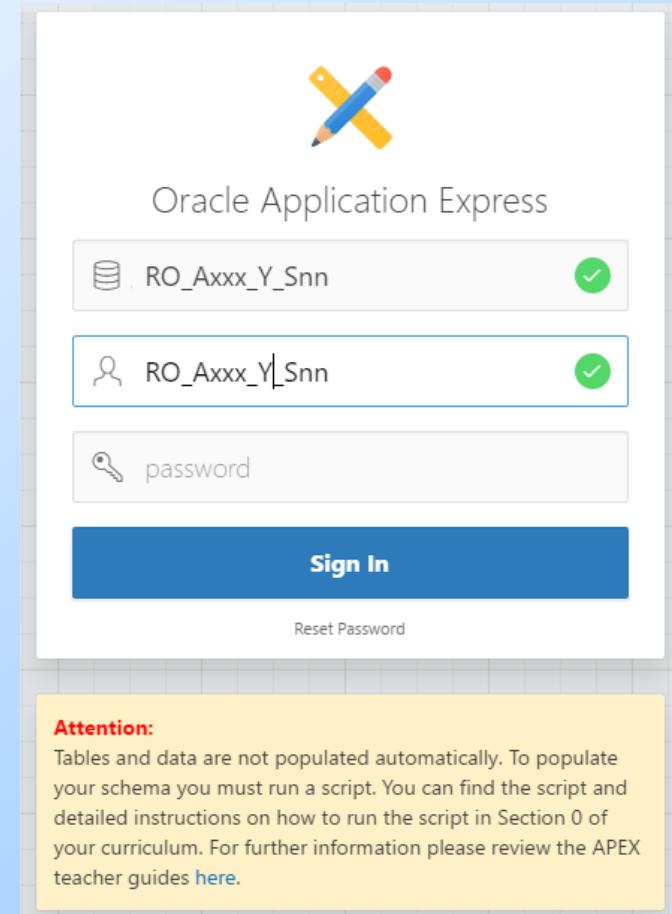
□ <https://iacademy.oracle.com>

□ xxx este un număr ce va fi comunicat ulterior

□ Y este SQL sau PLSQL

□ nn este un număr ce va fi comunicat ulterior

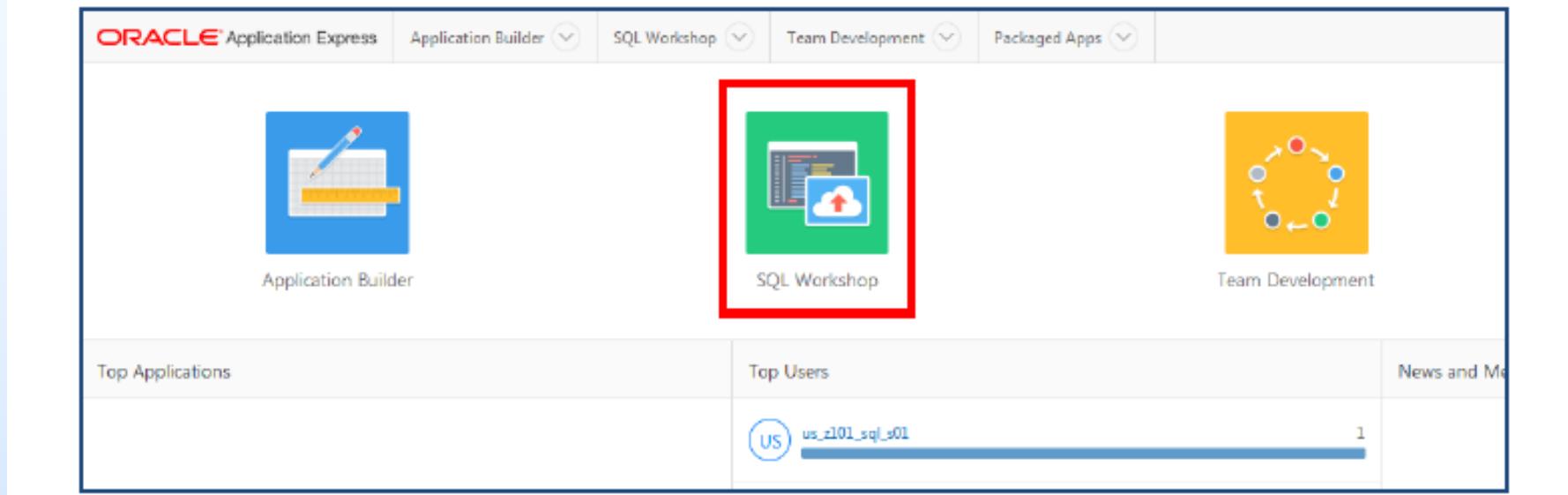
□ Parola va fi comunicată ulterior



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

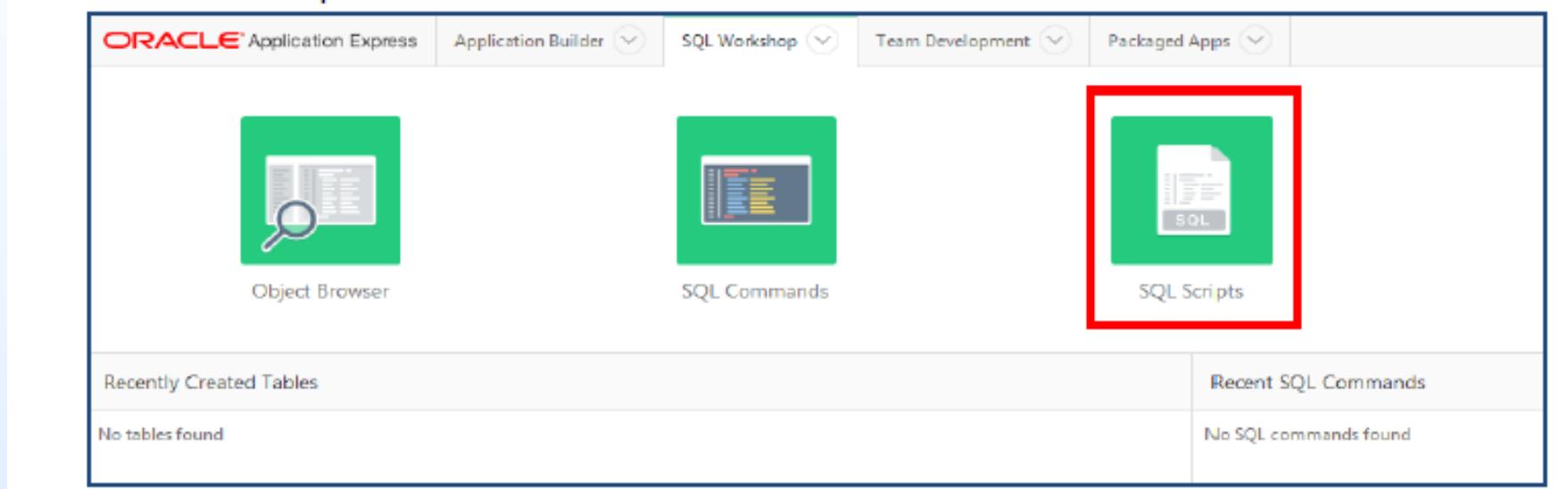
1. Download the script file from iLearning Database Programming with SQL (or PL/SQL) course. Go to Section 0, Course Resources, click “APEX Scripts and User Guides”, click “Script to Create Tables and Data for This Course” and save the script file locally on your PC.
2. Open APEX in your browser and login.
3. Select “SQL Workshop”



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

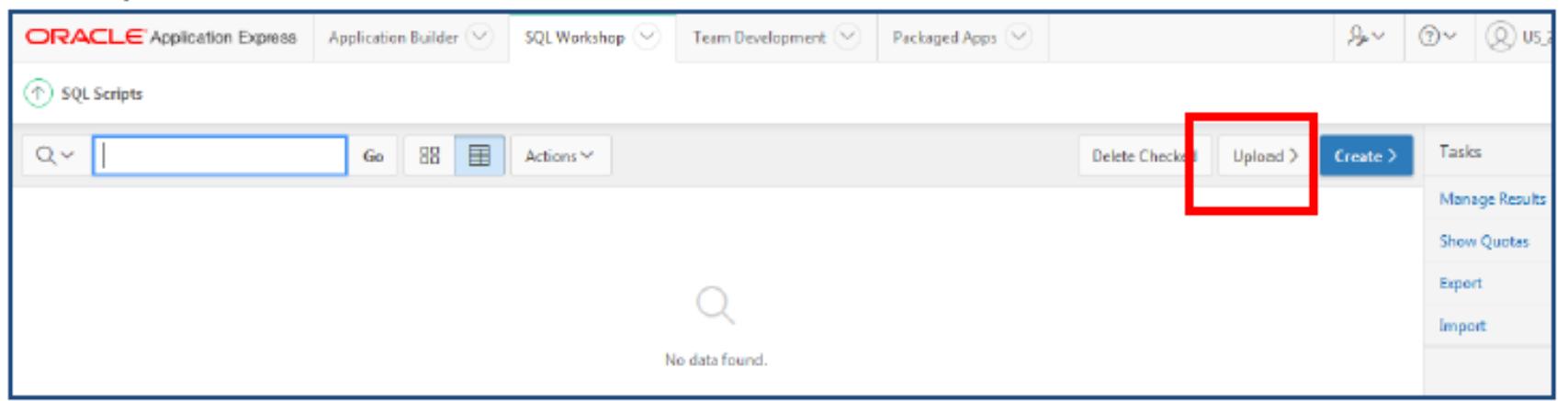
4. Select "SQL Scripts"



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

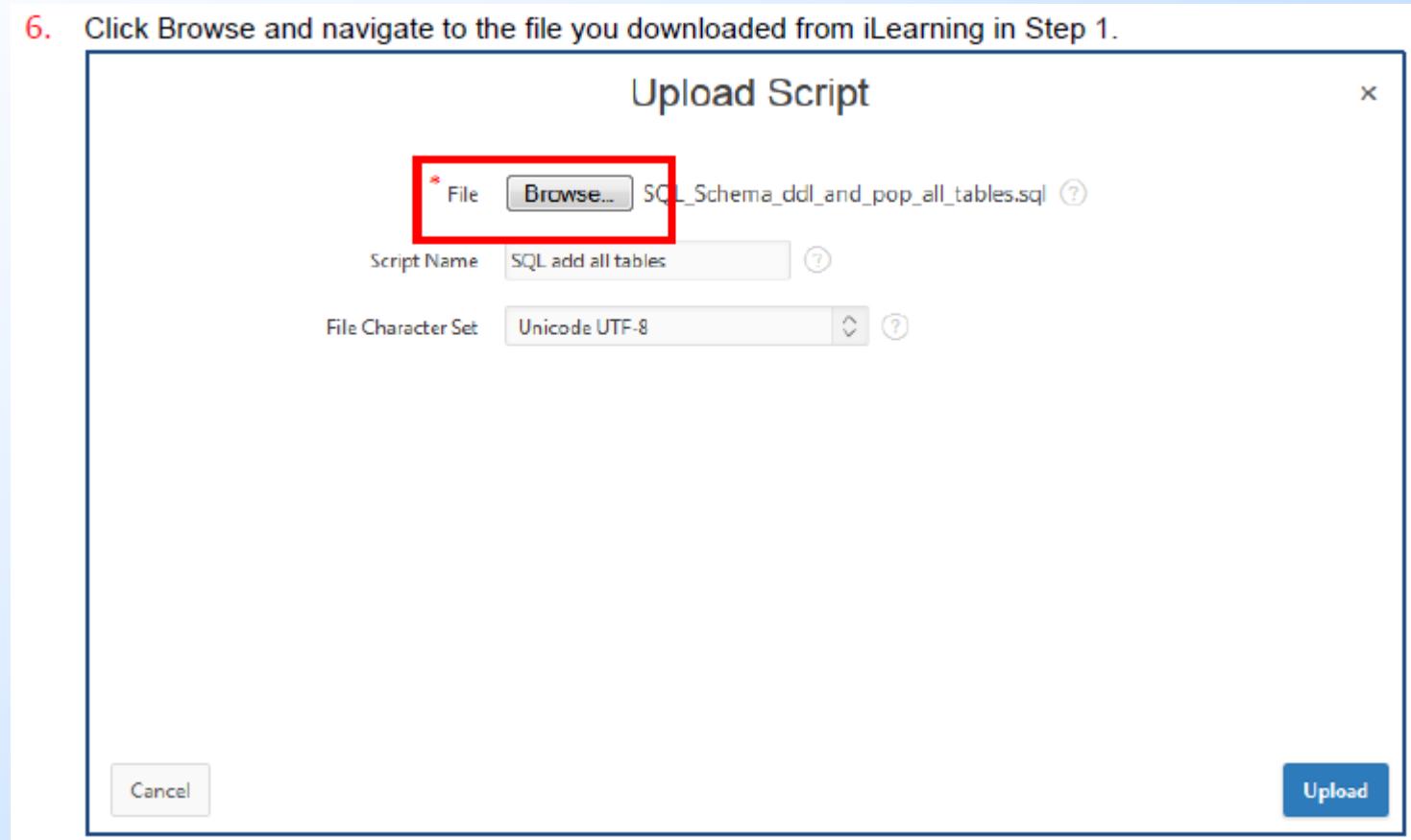
5. Click "Upload"



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

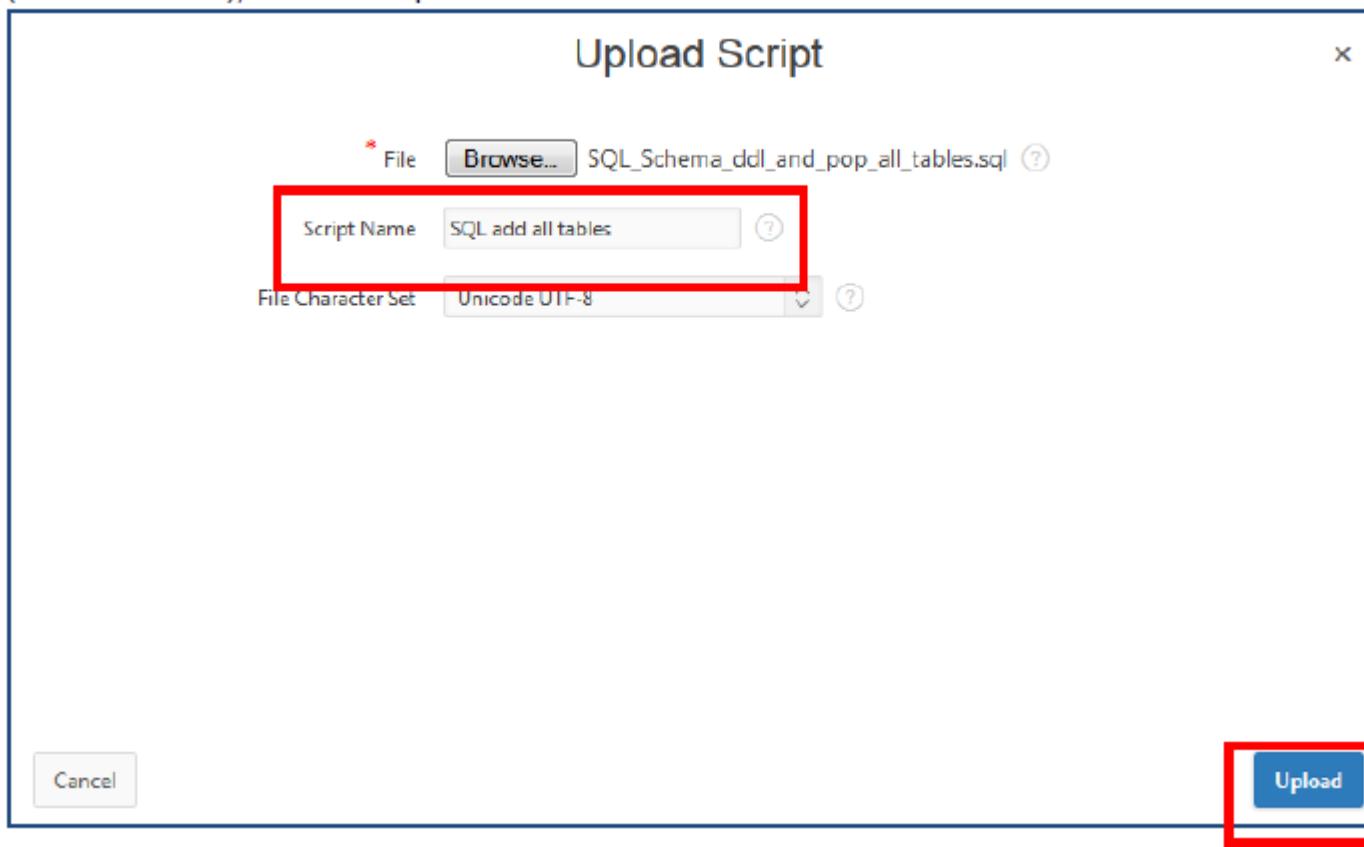
6. Click Browse and navigate to the file you downloaded from iLearning in Step 1.



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

7. Add a Script Name - "SQL add all tables" or "PLSQL add all tables", leave "File Character Set" as default (Unicode UTF-8), and click "Upload".



Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

8. You will now see the Script listed

The screenshot shows the Oracle Application Express interface with the 'SQL Workshop' tab selected. In the 'SQL Scripts' section, there is a single listed script named 'SQL add all tables'. The 'Run' button for this script is highlighted with a red box. The table below lists the script details:

| Owner | Name | Created | Updated By | Updated | Bytes | Results | Run |
|-----------------|--------------------|---------------|-----------------|---------------|---------|---------|-----|
| US_Z101_SQL_S01 | SQL add all tables | 2 seconds ago | US_Z101_SQL_S01 | 1 seconds ago | 308,307 | 0 | |

Click the Run icon.

Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

9. Click "Run Now"

The screenshot shows a confirmation dialog box titled 'Run Script'. It features a yellow exclamation mark icon and the text 'Run Script'. Below this, there is a message: 'You have requested to run the following script. Please confirm your request.' A table provides details about the script: Script Name (SQL add all tables), Created (on 06/07/2016 09:21:56 AM by US_Z101_SQL_S01), Updated (on 06/07/2016 09:21:57 AM by US_Z101_SQL_S01), Number of Statements (1540), and Script Size in Bytes (308,397). At the bottom, there are 'Cancel' and 'Run Now' buttons, with 'Run Now' being highlighted with a red box.

| Script Name | SQL add all tables |
|----------------------|--|
| Created | on 06/07/2016 09:21:56 AM by US_Z101_SQL_S01 |
| Updated | on 06/07/2016 09:21:57 AM by US_Z101_SQL_S01 |
| Number of Statements | 1540 |
| Script Size in Bytes | 308,397 |

This will take you to the Manage Script Results Page.

Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

10. Click "View Results"

The screenshot shows the Oracle Application Express interface with the SQL Workshop module selected. A table titled 'Manage Script Results' displays the execution of a script named 'PLSQL_Schema_ddl_and_pop_all_tables.sql'. The table includes columns for Script, Run By, Started, Finished, Status, Security Group Id, Statements, Bytes, and View Results. The 'View Results' button in the last column is highlighted with a red box.

| Script | Run By | Started | Finished | Ran by | Status | Security Group Id | Statements | Bytes | View Results |
|---|-------------------|---------------|------------|--------|-----------|-------------------|--------------|-------|---|
| PLSQL_Schema_ddl_and_pop_all_tables.sql | US_ZZ01_PLSQL_300 | 8 seconds ago | 06/16/2016 | 8:31 | Completed | 1083810868135457 | 1807 of 1807 | 0 | <input type="button" value="View Results"/> |

11. You can view the results, however, your first attempt to run the script will generate errors on the DROP statements due to the tables not already existing in the schema.

Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos:

12. Click the “SQL Workshop” tab

The screenshot shows the Oracle Application Express interface. The top navigation bar has tabs: ORACLE Application Express, Application Builder, SQL Workshop (which is highlighted with a red box), Team Development, and Packaged Apps. Below the tabs, there's a breadcrumb trail: SQL Scripts > Manage Script Results. A search bar and an Actions dropdown are present. The main area displays a table with one row of data:

| Script | Run By | Started | Finished | Elapsed | Status | Security Group |
|--------------------|-----------------|---------------|------------|---------|-----------|----------------|
| SQL add all tables | US_Z101_SQL_S01 | 7 seconds ago | 06/07/2016 | 6.30 | Completed | 7076210. |

13. Click “Object Browser”

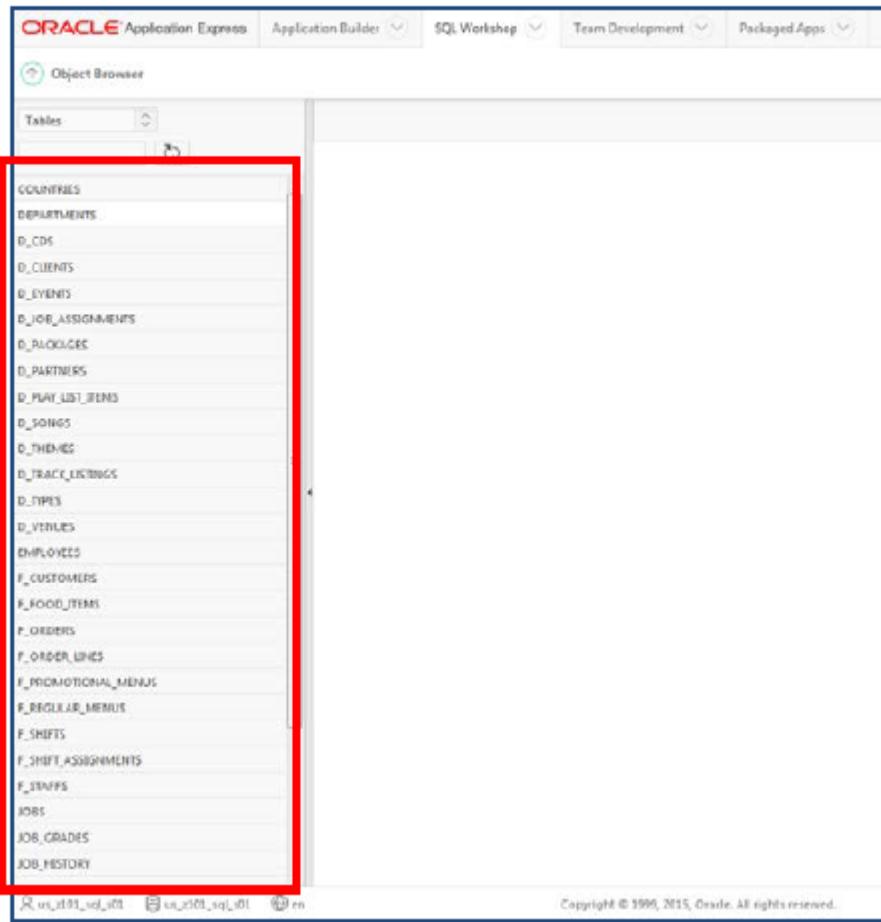
The screenshot shows the Oracle Application Express interface with the SQL Workshop tab selected. In the main area, there are three icons: Object Browser (highlighted with a red box), SQL Commands, and SQL Scripts. Below the icons, there are two sections: Recently Created Tables and Recent SQL Commands. Both sections show a message indicating no results were found.

| Recently Created Tables | Recent SQL Commands |
|-------------------------|-----------------------|
| No tables found | No SQL commands found |

Lucrări de laborator

- Pentru a adăuga tabelele necesare desfășurării activităților practice, se vor urma pașii de mai jos (ultimul pas):

14. You should now see the tables listed on the left of the Object Browser page. These are the tables (and data) that will be used in the curriculum for your course(s).



Lucrări de laborator

- Model ER. Prezentarea dictionarului datelor. Definirea schemei BD (LDD SQL).
- LMD SQL clauzele SELECT, FROM, WHERE, ORDER BY.
- Folosire funcții SQL „Single Row”.
- Operatori JOIN. Funcții de agregare. Clauzele GROUP BY, HAVING.
- Interogări imbricate. Clauzele UNION, INTERSECT, EXCEPT.

Lucrări de laborator

- LMD SQL pentru actualizarea BD.
- Constrângeri. Vederi. Triggere.
- Proceduri stocate. Cursoare.
- Prezentare mediu XAMPP, PHP Editor.
Aplicații cu BD (1). Realizare conexiune
PHP-BD Oracle. Interrogarea și afișarea
rezultatelor.
- **Colocviu parțial de laborator.**

Lucrări de laborator

- Aplicații cu BD (2). Structurare aplicație: logica de prezentare, logica aplicației, acces la BD, tratare erori.
- Aplicații cu BD (3). Actualizare date, folosire paginare în afișarea rezultatelor, apel proceduri stocate.
- **Colocviu final de laborator**

SQL?

- Explicați diferențele între:

```
SELECT b
```

```
FROM R
```

```
WHERE a<10 OR a>=10;
```

și

```
SELECT b
```

```
FROM R;
```

| a | b |
|-----|-----|
| 5 | 20 |
| 10 | 30 |
| 20 | 40 |
| ... | ... |

R

SQL?

- Respectiv, diferențele între:

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

și

```
SELECT a  
FROM R  
WHERE b IN (SELECT b FROM S);
```

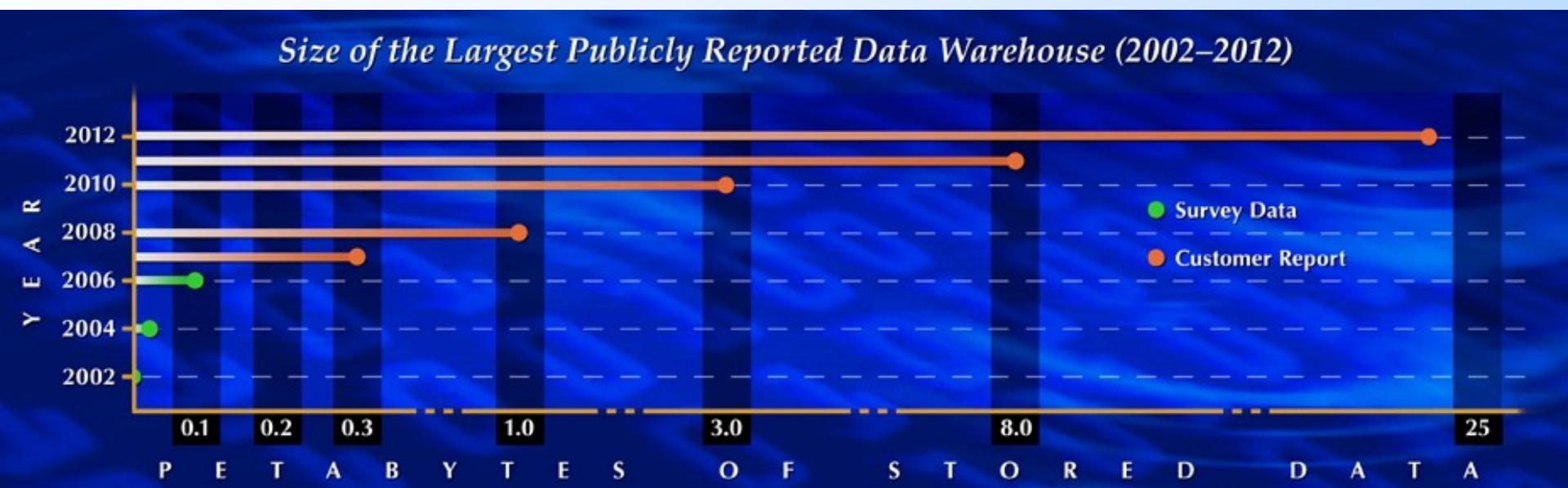
Ce este în esență o bază de date (BD)?

- În marea majoritate a cazurilor o BD conține:
 - înregistrări despre angajați,
 - înregistrări bancare,
 - etc.
- În zilele noastre, domeniul BD acoperă largi surse de date, cu multe concepte noi.
 - Căutare Web.
 - Data mining.
 - Baze de Date Științifice și medicale.
 - Integrarea informației.

Cantitatea totală a datelor de pe Terra (raport IDC)

- 2010: stivă de DVD-uri de la Pământ la Lună și înapoi (aproximativ $384.400\text{ km} \times 2$) adică 1,2 milioane petabytes, sau 1,2 zettabytes ($1,2 \times 10^{21}\text{ B}$)
- 2020: stivă de DVD-uri jumătatea distanței între Pământ și Marte (aproximativ 30 de milioane km), creștere de 40 de ori

Evoluția dimensiunii sistemelor de baze de date



44 zettabytes în 2020, de 40 de ori mai mulți bytes decât numărul stelelor în universul observabil

<https://www.visualcapitalist.com/wp-content/uploads/2019/04/data-generated-each-day-full.html>

Un lucru esențial ...

- Se poate observa că bazele de date stau în spatele aproape oricărui lucru de pe Web.
 - Căutări Google.
 - Interogări pe Amazon, eBay, etc.

Definiție

- O bază de date este o **colecție** în general mare **de date înrudite**, stocate într-un sistem de calcul astfel încât un program-calculator sau o persoană ce folosește un limbaj de interogare să o poată **consulta** pentru a răspunde la întrebări.

Programare în context BD

- Este centrată în jurul unor limbaje de programare limitate.
 - Își găsesc sens doar în aria unde se pot utiliza limbaje “non-Turing-complete”.
 - Conduce la o programare foarte succinctă.

Probleme ce apar...

1. controlul concurenței

- Multe activități (tranzacții) cu baza de date se desfășoară la același moment de timp și acționează asupra acelorași bucăți de informație.

2. optimizarea interogărilor

- Volum mare de date.

Sistem de Gestiune Baze de Date (SGBD)

- Un pachet de programe care permit crearea, utilizarea și eliminarea obiectelor ce compun baza de date.
- Scopul principal:
 - Reducerea dependenței aplicațiilor în raport cu structura datelor.
- Categorisit după modelul de date: de ex. System “R” - SGBD relațional

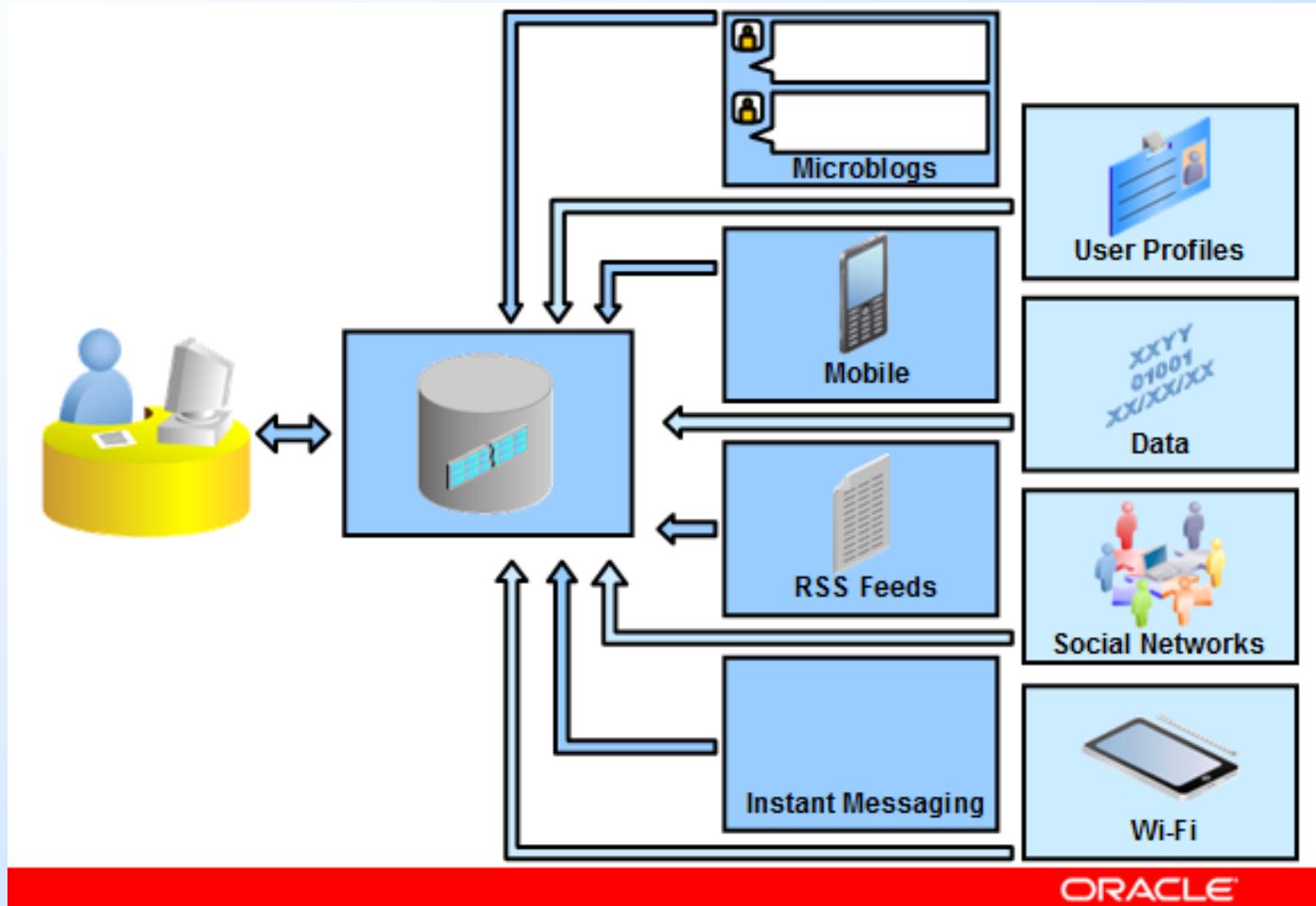
Ce este un Model de Date?

1. O reprezentare matematică a datelor.
 - Exemple:
 - modelul relațional = tabele;
 - modelul semistructurat = arbori/grafuri.
2. Operații cu datele.
3. Constrângeri.

Rolul și importanța BD

- Nu există aplicație reală fără BD
- Pe piața software există un număr mare de SGBD-uri, pentru toate tipurile de sisteme de calcul, sisteme de operare și tehnologii de acces la date
- SGBD-urile apar în top 3 a celor mai vândute produse

Scenariu de utilizare BD



Funcțiile SGBD

- Gestiunea dicționarului de date
 - Programele aplicative accesează datele prin intermediul SGBD, care caută în dicționarul de date structura datelor și a legăturilor necesare rezolvării problemei utilizatorului
- Gestiunea fișierelor de date
 - Colecția de date (BD) se materializează printr-un fișier sau colecție de fișiere de date, fișiere index, etc.

Funcțiile SGBD

□ Transformarea datelor

□ Datele introduse de utilizator nu au întotdeauna structura identică cu cea definită în baza de date

□ Gestiunea aplicațiilor

- Limbaj de descriere a datelor
- Limbaj de manipulare a datelor
- Limbaj pentru afișarea datelor (pe ecran sau la imprimantă)

Funcțiile SGBD

- Importul și exportul datelor
 - Conversia datelor pentru prelucrarea cu alt SGBD sau cu aplicații terțe (de ex. Excel)
- Controlul securității datelor
 - Utilizatorii ce au acces la date
 - La ce date are acces fiecare utilizator
 - Ce operații se pot efectua de fiecare utilizator cu datele la care are acces

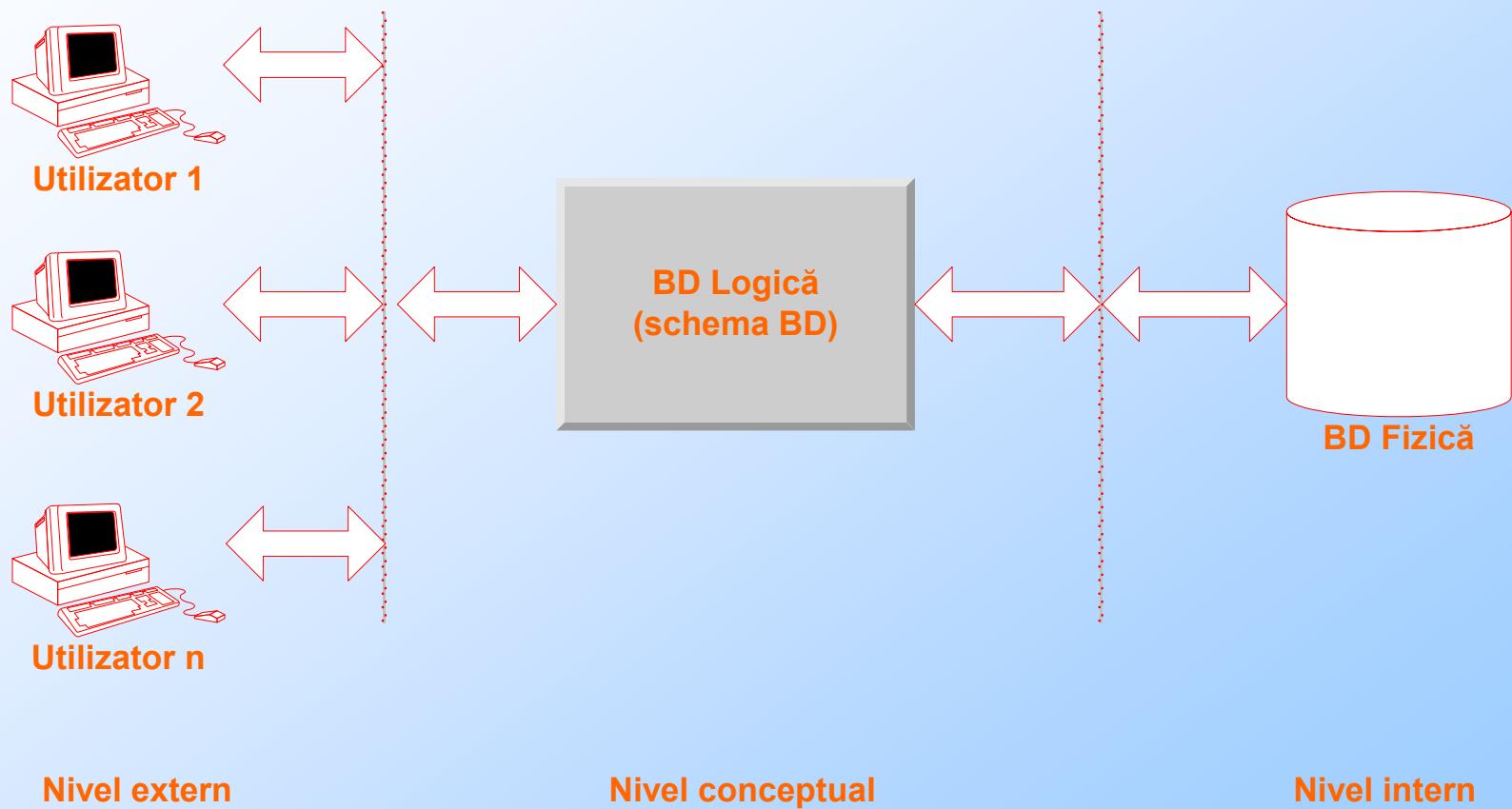
Funcțiile SGBD

- Asigurarea integrității datelor
 - Restricții de integritate, de ex. SGBD-ul poate asigura că vârsta unei persoane la introducerea în BD este cuprinsă între 18 și 40 ani
- Controlul accesului concurent la date
 - Fiecare utilizator are impresia că lucrează de unul singur (serializarea operațiilor)
- Gestiona copiilor de siguranță și a recuperării datelor în caz de dezastre

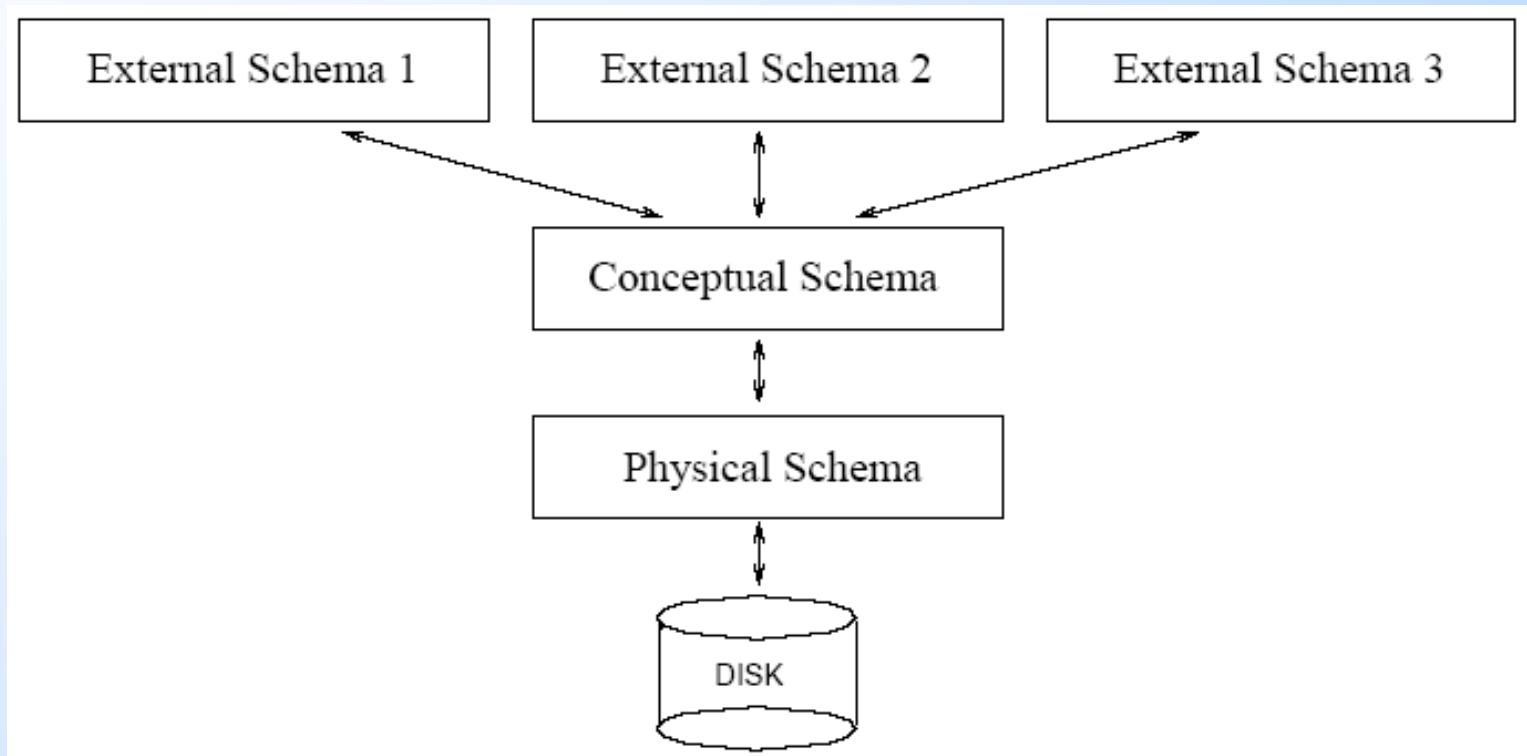
SGBD commerciale

- Oracle (Oracle 19c)
- IBM (DB2 11.5)
- Microsoft (SQL Server 2019)

Nivele de abstractizare



Nivele de abstractizare



Schema conceptuală

- sau **schemă logică**, descrie datele stocate în BD în termeni ai modelului de date al SGBD
 - Pentru un SGBD relational, schema conceptuală descrie toate tabelele (relațiile) stocate în BD
 - Limbajul folosit se numește Limbaj de Descriere a Datelor (LDD)

Problemă

- Determinați schema conceptuală pentru o universitate cu mai multe facultăți unde studiază studenți
- Răspuns:
 - Tabele ce corespund la entități: *facultăți, studenți*
 - Tabele ce corespund la relații de legătură între entități: *studenți_participă_la_cursuri*
 - Coloane ce corespund la atrbutele entităților (ce coloane are fiecare tabelă): *Student (cnp: string, nume: string, data_nășterii: date, sex_f: bool, media_sesiune: real)*

Schema fizică

- Specifică detalii suplimentare legate de stocarea datelor
- Menționează modul în care tabelele (la modelul relațional) descrise prin schema conceptuală sunt stocate pe dispozitive suport secundar, discuri sau benzi magnetice
- Descrie tipul fișierelor pentru stocare pe suport secundar (de ex. la MySQL: MyISAM sau InnoDB) și crearea unor structuri auxiliare de date numite *indecși* în scopul regăsirii mai rapide a datelor

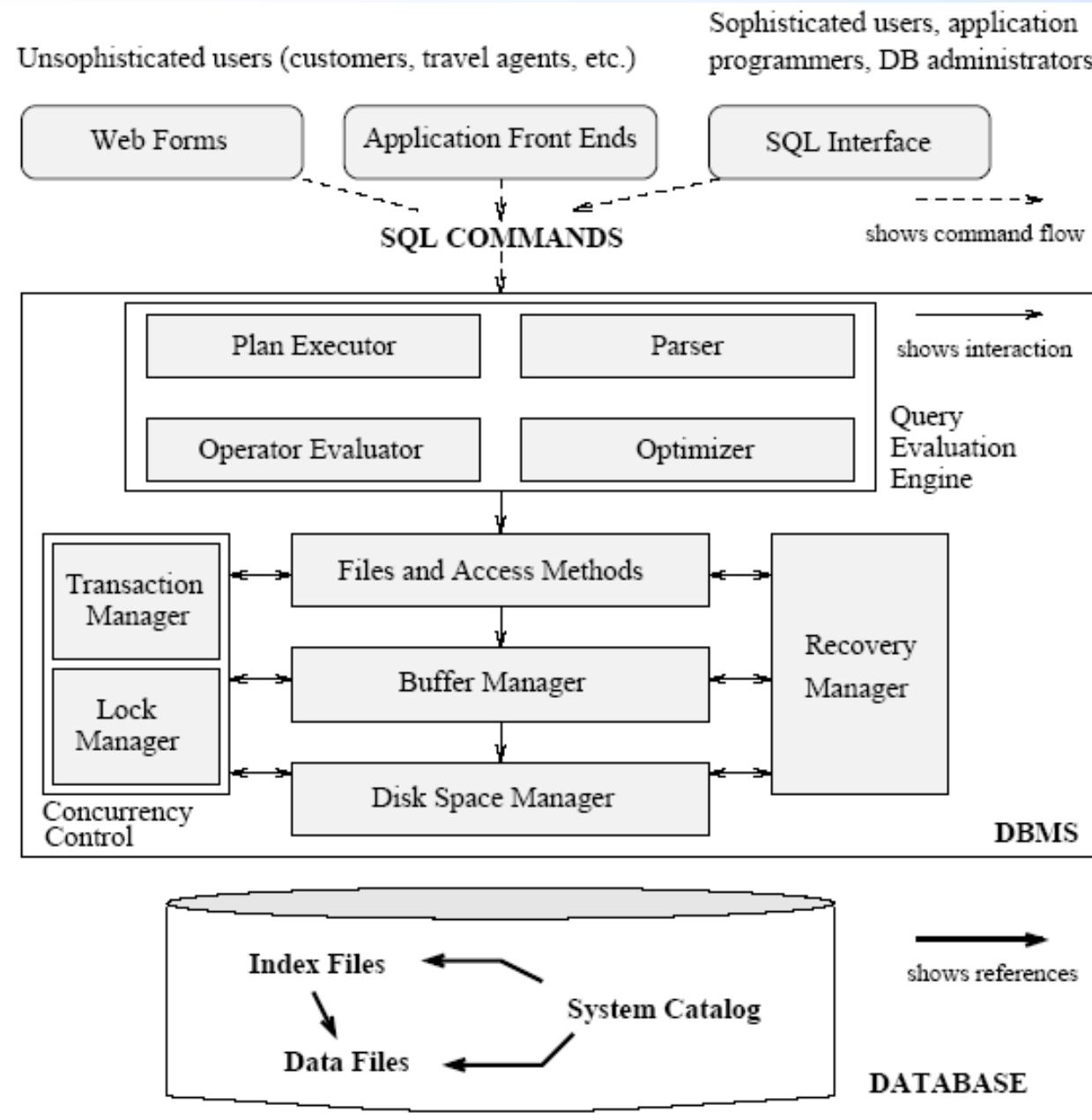
Schema externă

- Permite accesul la date să fie “croit” după nevoia grupurilor de utilizatori (de exemplu un angajat are acces la datele de salarizare proprii, iar un manager are acces la datele de salarizare ale tuturor angajaților din subordine)
- O BD are exact 1 schemă conceptuală și 1 schemă fizică
- O BD poate avea mai multe scheme externe

Independența datelor

- Aplicațiile sunt izolate față de modificările la nivel conceptual sau la nivel fizic prin cele trei nivele de abstractizare
- Independența logică a datelor
 - Vederile (**view** în modelul relațional, tabelă virtuală, schema externă) asigură posibilitatea modificării structurii datelor (schema conceptuală), acest lucru fiind ascuns aplicațiilor
- Independența fizică a datelor
 - Schema conceptuală asigură posibilitatea modificării aranjării datelor pe suport secundar sau a indecșilor, acest lucru fiind de asemenea ascuns aplicațiilor

Arhitetura Unui SGBD



Cum funcționează?

- SGBD-ul acceptă comenzi SQL generate de o varietate de interfețe utilizator
- SGBD-ul produce planuri de evaluare a interogărilor, pe care le execută asupra datelor din BD, și returnează răspunsuri

Cum funcționează?

- Un utilizator emite o interogare, aceasta este analizată și este prezentată unui *optimizator*, care folosește informația despre felul în care sunt stocate datele pentru a produce un plan de execuție eficient
- Un *plan de execuție* este o reprezentare sub formă de arbore operator (cu adnotări ce conțin informații detaliate suplimentare legate de metodele de acces, s.a.)

Cum funcționează?

- Codul ce implementează operatorii *arborelui operator* stă deasupra stratului “File and Access Methods”
- Într-un SGBD “*file*” este o colecție de pagini sau o colecție de înregistrări
- De obicei stratul “File and Access Methods” suportă “*heap file*” de pagini neordonate, sau *indexi*
- Stratul “File and Access Methods” urmărește cum sunt aranjate paginile în fișier și organizează datele în interiorul unei pagini

Cum funcționează?

- Stratul “*buffer manager*” este responsabil cu aducerea paginilor de pe disc în memoria internă
- Stratul “*disk space manager*” este cel mai de jos strat al unui SGBD și se ocupă de administrarea spațiului pe suport extern, unde sunt stocate datele
 - Straturile superioare alocă, dealocă, citesc și scriu pagini

Cum funcționează?

- Straturile “*Transaction Manager*”, “*Lock Manager*” și “*Recovery Manager*” asigură accesul concurent și recuperarea datelor în caz de incidente prin implementarea unor protocoale de blocare (“lock”), prin planificarea cu atenție a cererilor utilizatorilor și prin păstrarea într-un jurnal a tuturor modificărilor asupra BD

Cine utilizează BD?

1. Implementatori BD

- Scriu software SGBD (soft de bază)

2. Programatori de Aplicații BD

- Dezvoltă pachete de programe ce facilitează accesul la date al utilizatorilor finali
- Folosesc "host languages" sau "data languages" și unelte software
- La modul ideal aplicațiile BD lucrează prin schema externă, dar este posibil să acceseze datele și la nivelele de mai jos când este posibil să fie compromisă independența datelor

3. Utilizatori finali

- Înregistrează și interoghează BD, de obicei prin intermediul aplicațiilor
- În marea majoritate a cazurilor nu sunt specialiști în domeniul calculatoarelor (de exemplu un agent comercial)

4. Administratorul BD

- Al patrulea și cel mai important utilizator**

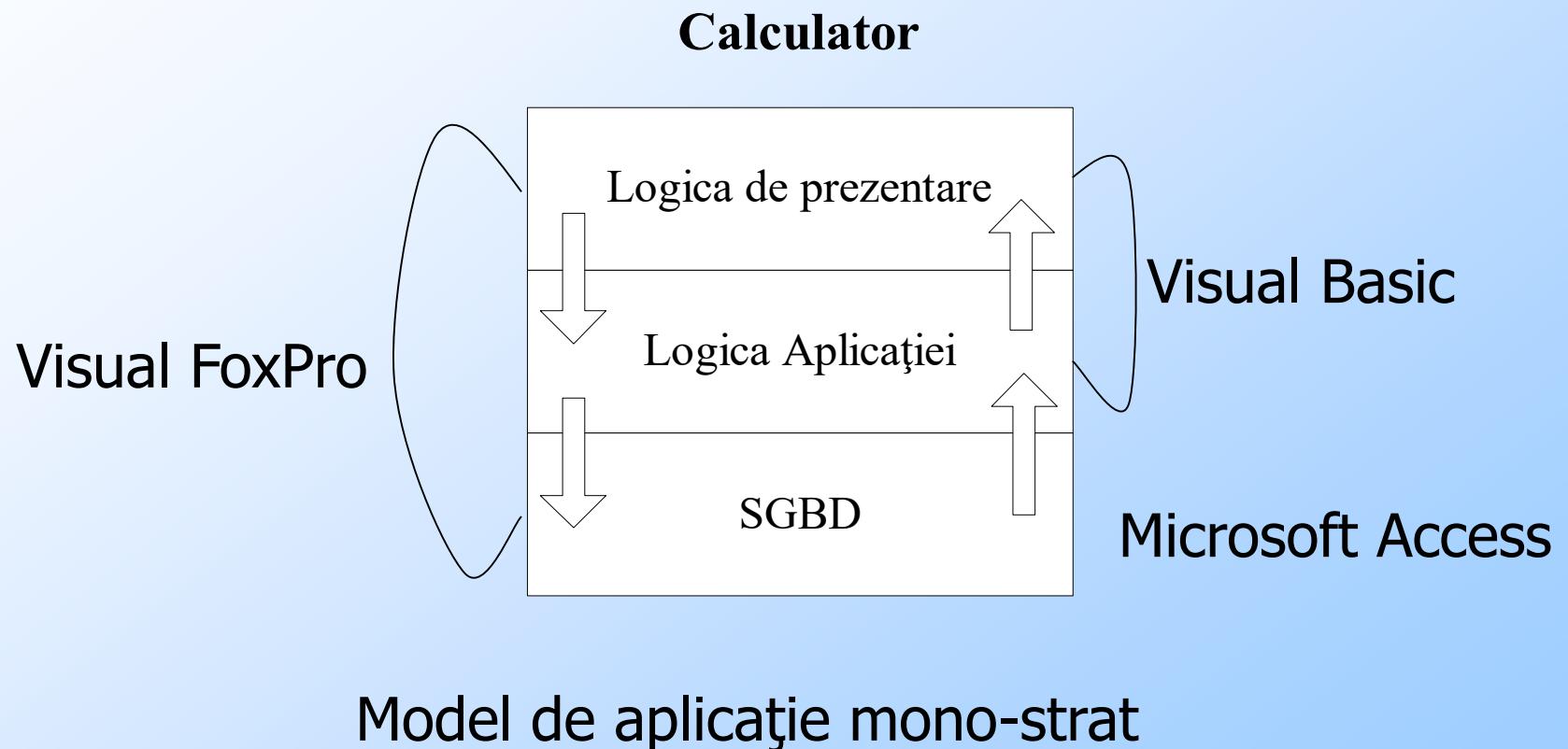
Administratorul BD

- BD de tip “corporate” sau “enterprise-wide” este activul cel mai important al companiei
- Principalele sarcini:
 - Să proiecteze schema conceptuală și schema fizică
 - Să implementeze politici de securitate și autorizare
 - Să asigure disponibilitatea datelor și să recupereze datele în cazul apariției de incidente
 - Să efectueze “database tuning” (reglarea parametrilor pentru performanță)

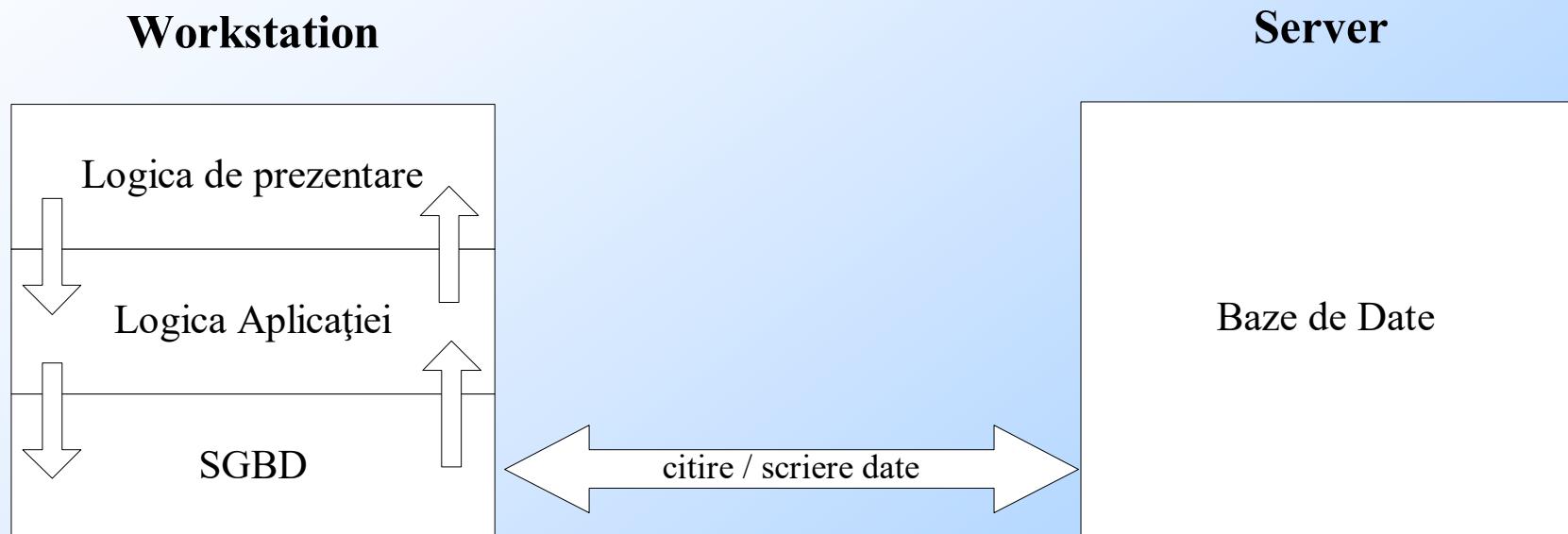
Aplicații BD

- Interfață între limbaj, tehnologie și BD
 - **SQL – Structured Query Language**
- Arhitectură
 - Mainframe
 - Client – Server
 - N – Tier
 - Web application
 - Mobile application

Arhitectura aplicației BD



Arhitectura aplicației BD



Model de aplicație cu Server de fișiere

Arhitectura aplicației BD



Functii Client:

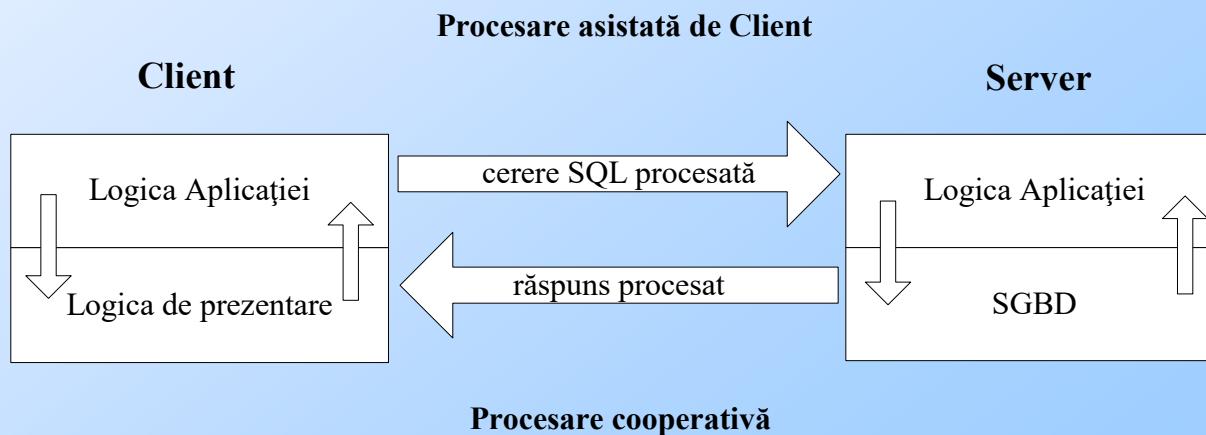
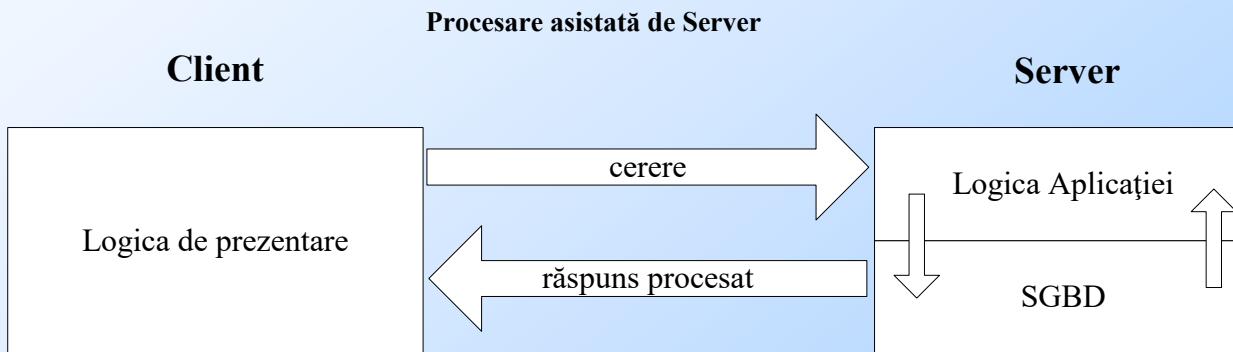
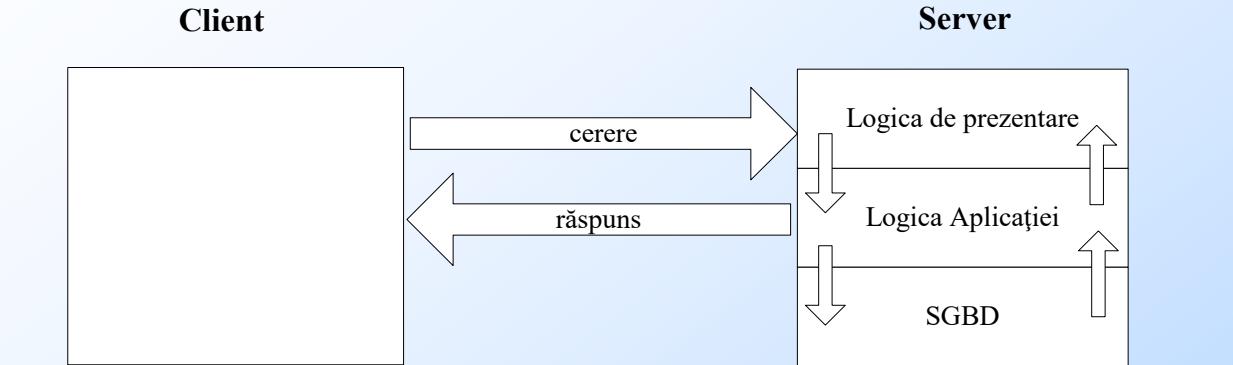
Interfață grafică
Procesare distribuită a aplicației
Aplicație locală
E-mail
Emulare terminal

Functii Server:

Server: de fișiere, de tipărire la imprimantă și de baze de date
Procesare distribuită a aplicației
E-mail
Comunicații
Administrare rețea
Administrare resurse
Administrare configurație

Componentele unui sistem „Client-Server”

Arhitectura aplicației BD



Arhitectura aplicației BD

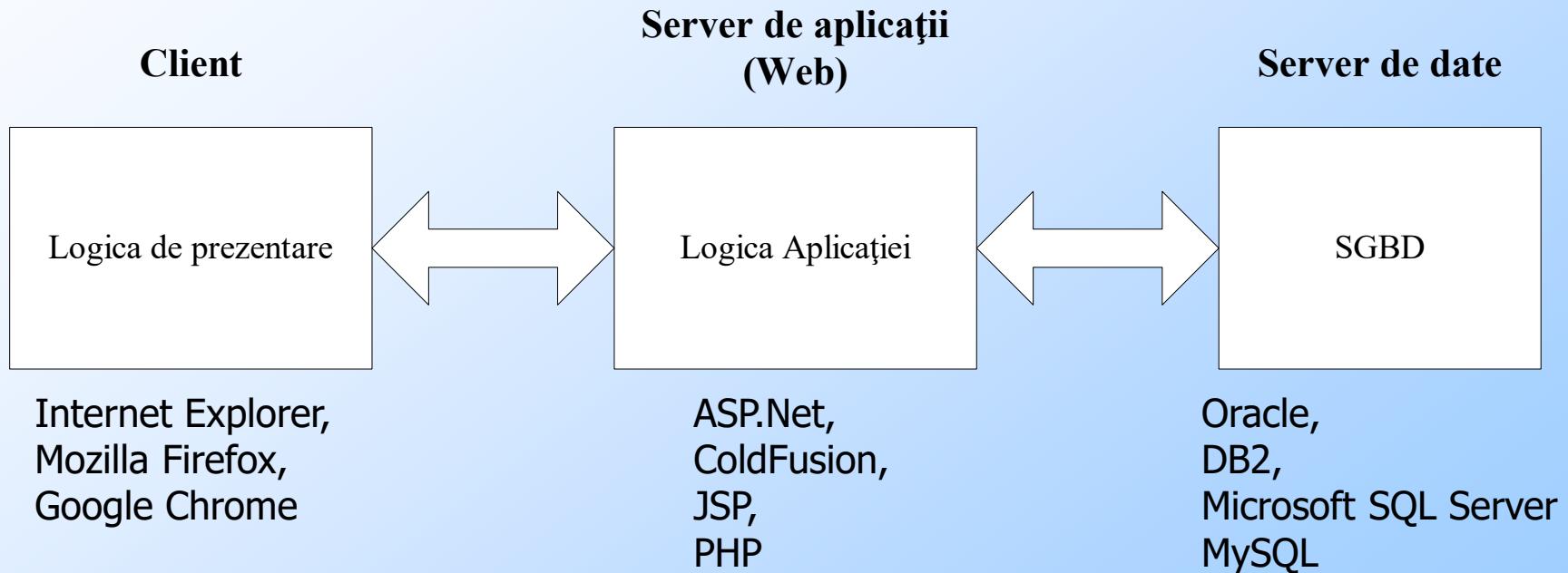
□ "thin clients"

- La client există GUI
- Serverul se ocupă de logica aplicației și de accesul la BD

□ "thick clients"

- Client mai puternic cu o parte din logica de aplicație și GUI
- Serverul are o parte din logica aplicației și accesul la BD

Arhitectura aplicației BD



Model de aplicație trei-straturi

Tehnologii pentru aplicația BD

□ Limbaje de programare

□ Python

□ Java

□ C#

Pagini Web dinamice

- Documentele clasice “hypertext” sunt documente statice.
- Scriptarea pe partea de client a apărut pentru modificarea comportamentului unei pagini web, ca răspuns la diferite evenimente cum sunt acțiunea tastaturii sau a mouse-ului.
- Scriptarea pe partea de server permite schimbarea sursei paginii furnizate între pagini. Răspunsurile serverului pot fi determinate de condiții cum ar fi datele dintr-un formular HTML, parametrii din URL, tipul browserului, trecerea timpului, starea BD.

Scriptarea pe partea de client

- Aspectul dinamic apare la prezentare, pe calculatorul client
- Serverul web regăsește pagina și o trimitе “as is”
- Browserul web procesează codul inclus în pagină (JavaScript) și afișează pagina
- Este dependentă de browser
- Ajax (asynchronous JavaScript and XML)
- Google Maps este un exemplu de aplicație web cu Ajax

Scriptarea pe partea de server

- Browserul trimite o cerere HTTP.
- Serverul regăsește scriptul sau programul solicitat
- Serverul execută scriptul sau programul care generează o pagină HTML
 - Parametrii de intrare ai programului se obțin din "query string" sau de la un formular web
- Serverul trimite rezultatul HTML browserului client

Aplicație web BD, Middleware

□ Active Server Pages (ASP)

- Este o soluție oferită de Microsoft ce permite folosirea diferitor limbaje (VBscript sau Jscript) în interiorul unei pagini HTML
- Tehnologia este pentru sistemul de operare Windows, cu suport pentru alte platforme, dar limitat

□ ASP.NET

- Este o componentă a platformei Microsoft .NET pentru dezvoltarea de aplicații web, un amestec între ASP clasic și tehnologia .NET
- Programatorii pot crea “dynamic web sites”, aplicații web și servicii web XML

Aplicație web BD, Middleware

- ColdFusion (ColdFusion Markup Language)
 - Este un sistem comercial de scriptare (deținut din 2005 de Adobe) “cross platform” și “tag-based”
- Java Server Pages (JSP)
 - Permite incorporarea de cod Java în pagini HTML
- PHP
 - Principala caracteristică “open source” a impus această soluție bazată pe includerea codului PHP în pagini HTML ca una foarte populară

Oracle 12c

https://docs.oracle.com/database/121/nav/portal_11.htm

Unelte de administrare

□ Oracle Universal Installer

- Oracle Universal Installer (OUI) este un utilitar folosit pentru instalare software Oracle și opțiuni.
- Poate lansa automat Oracle Database Configuration Assistant ce instalează o bază de date.

Tutorial: https://apex.oracle.com/pls/apex/f?p=44785:24:0::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:6281,1

□ Oracle Database Configuration Assistant

- DBCA este un utilitar ce creează o bază de date din template-uri furnizate de Oracle, sau de la zero.
- Se poate copia cu DBCA o bază de date preconfigurată.

Unelte de administrare

□ Database Upgrade Assistant

□ DBUA este o unealtă ce ghidează administratorul pentru operația de upgrade a unei baze de date existente la o versiune nouă Oracle Database.

□ Net Configuration Assistant

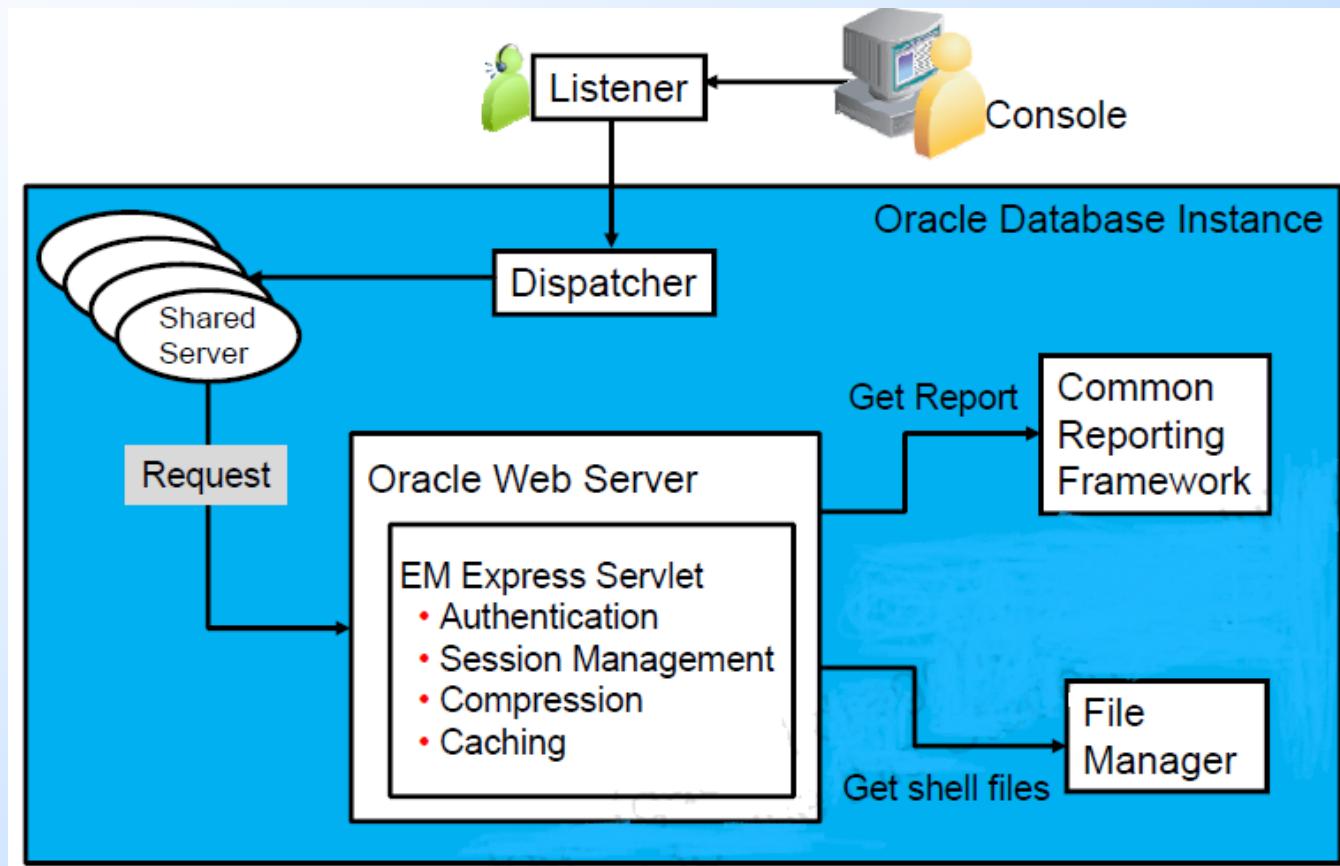
□ Este un utilitar ce permite configurare de listener și metode naming, ce sunt componente critice ale Oracle Database network.

Unelte de administrare

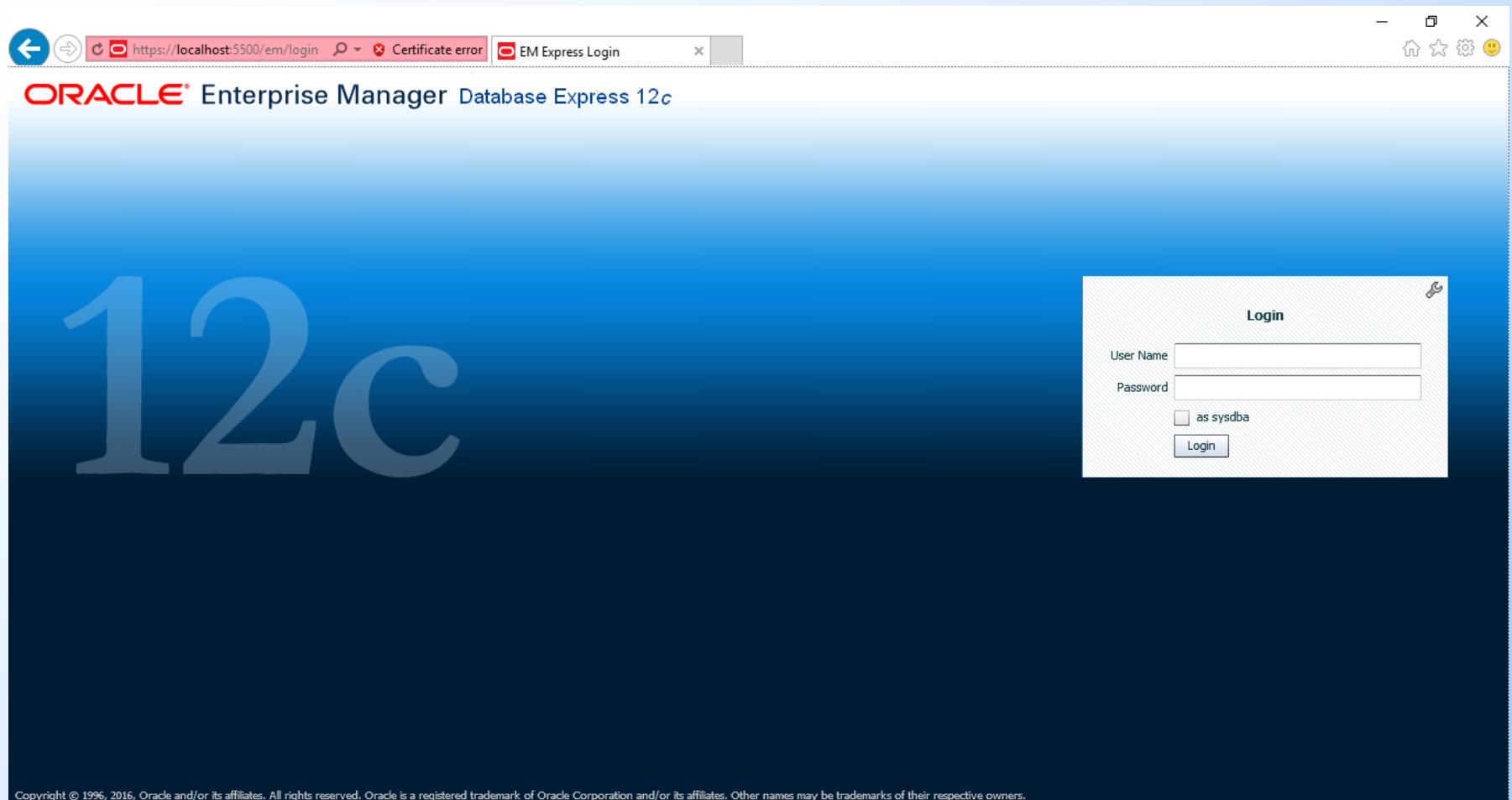
□ **Oracle Enterprise Manager Database Express**

- EM Express este cel mai important produs pentru administrarea bazei de date.
- Are o interfață Web-based.
- După instalarea Oracle Database software, crearea sau upgrade-ul unei baze de date, sau configurarea network, administratorul poate folosi EM Express pentru a administra baza de date.
- EM Express oferă și o interfață pentru performance advisors.

Arhitectura Oracle Enterprise Manager Database Express

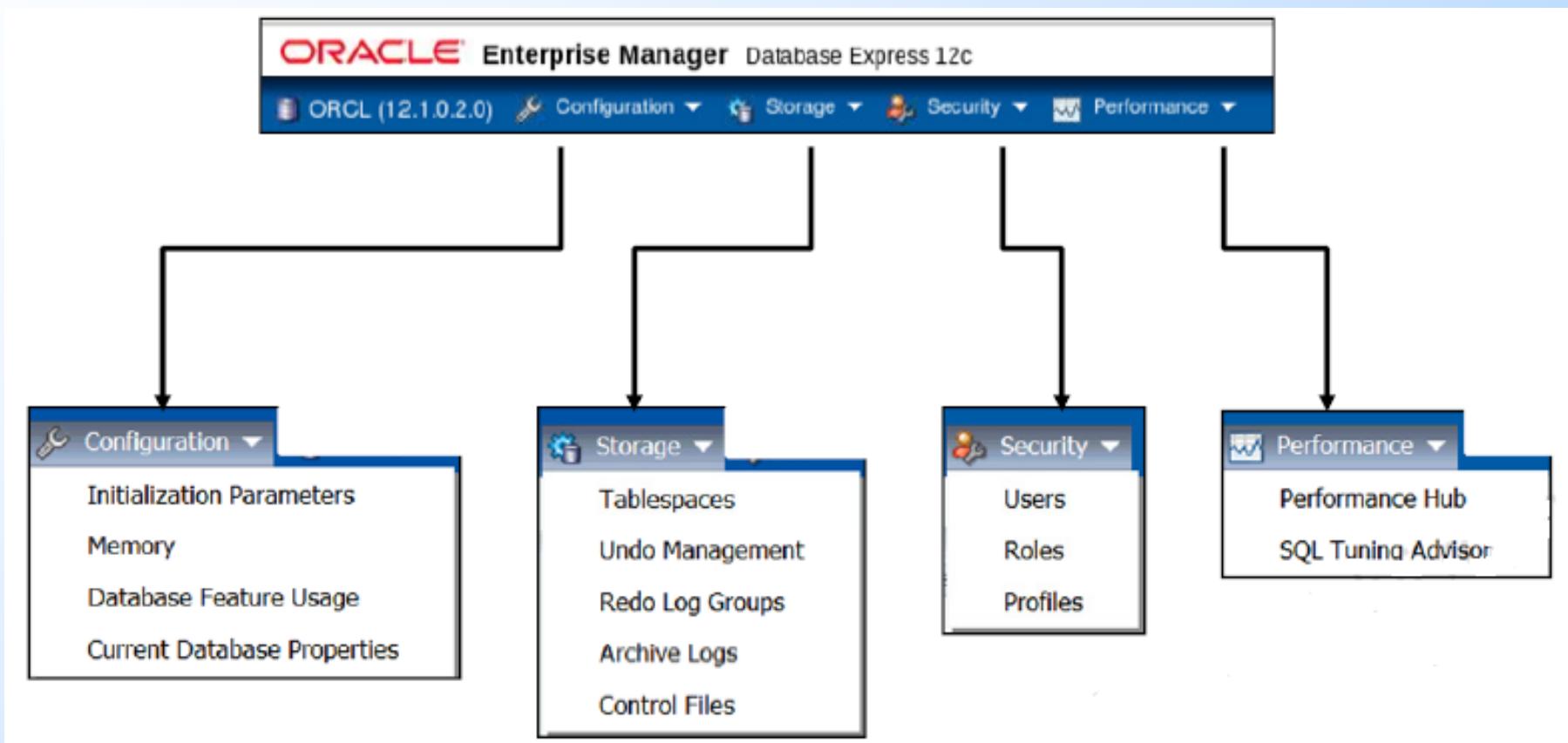


<http://localhost:5500/em>

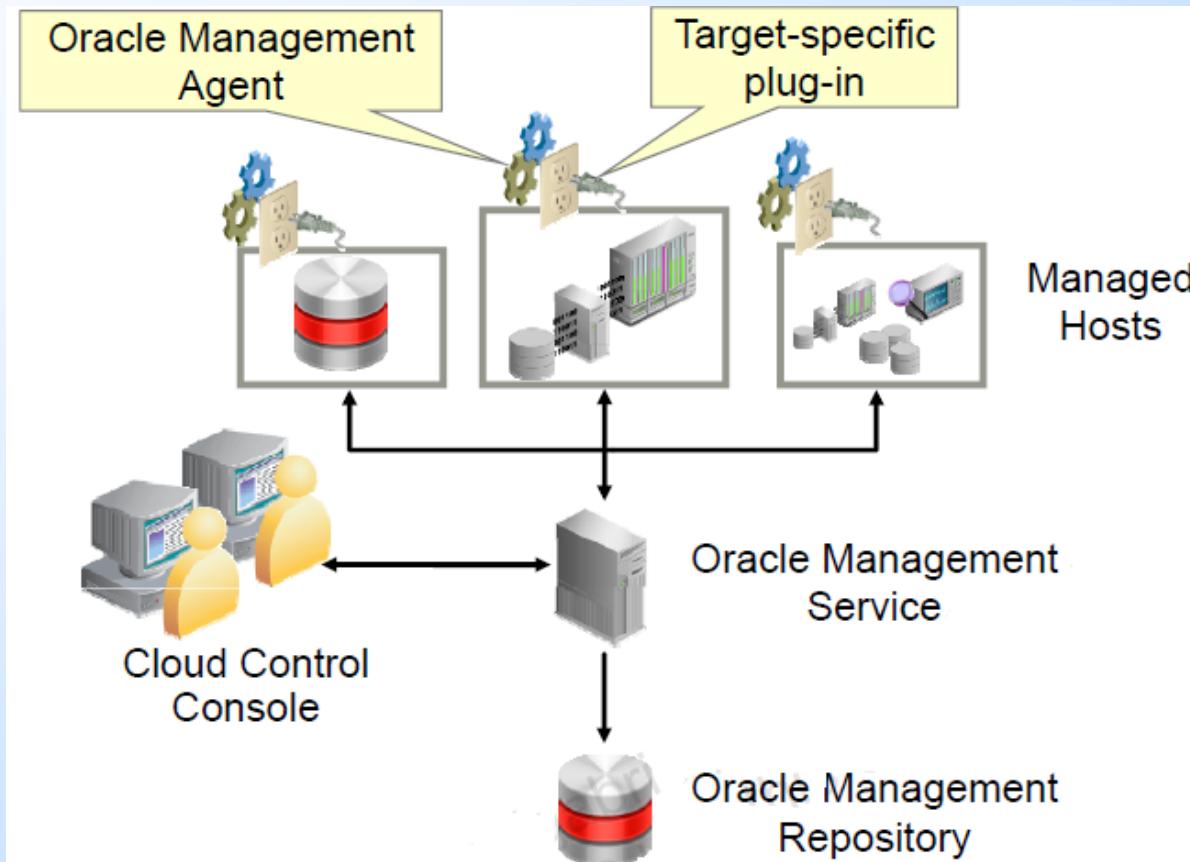


Copyright © 1996, 2016, Oracle and/or its affiliates. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

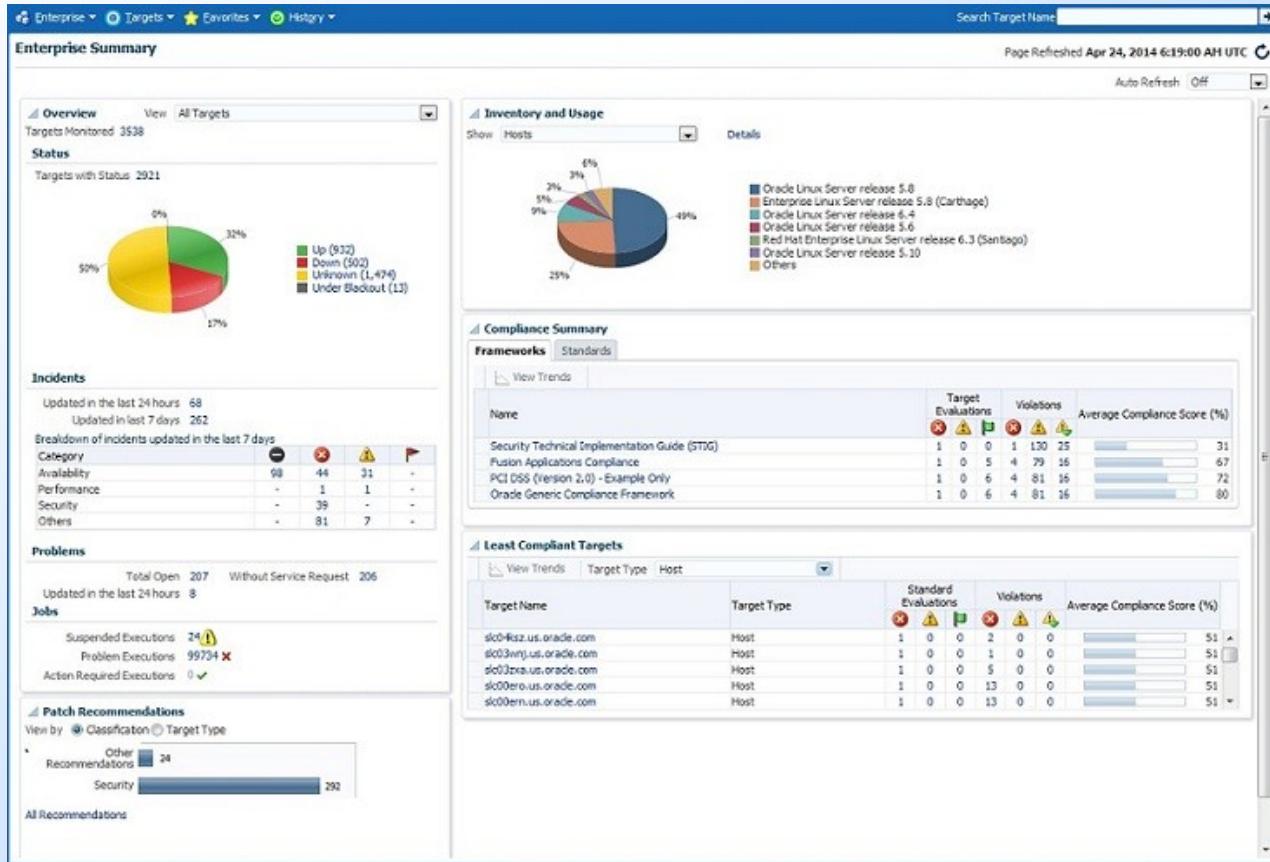
Meniuri Oracle Enterprise Manager Database Express



Componente Oracle Enterprise Manager Cloud Control



Oracle Enterprise Manager Cloud Control



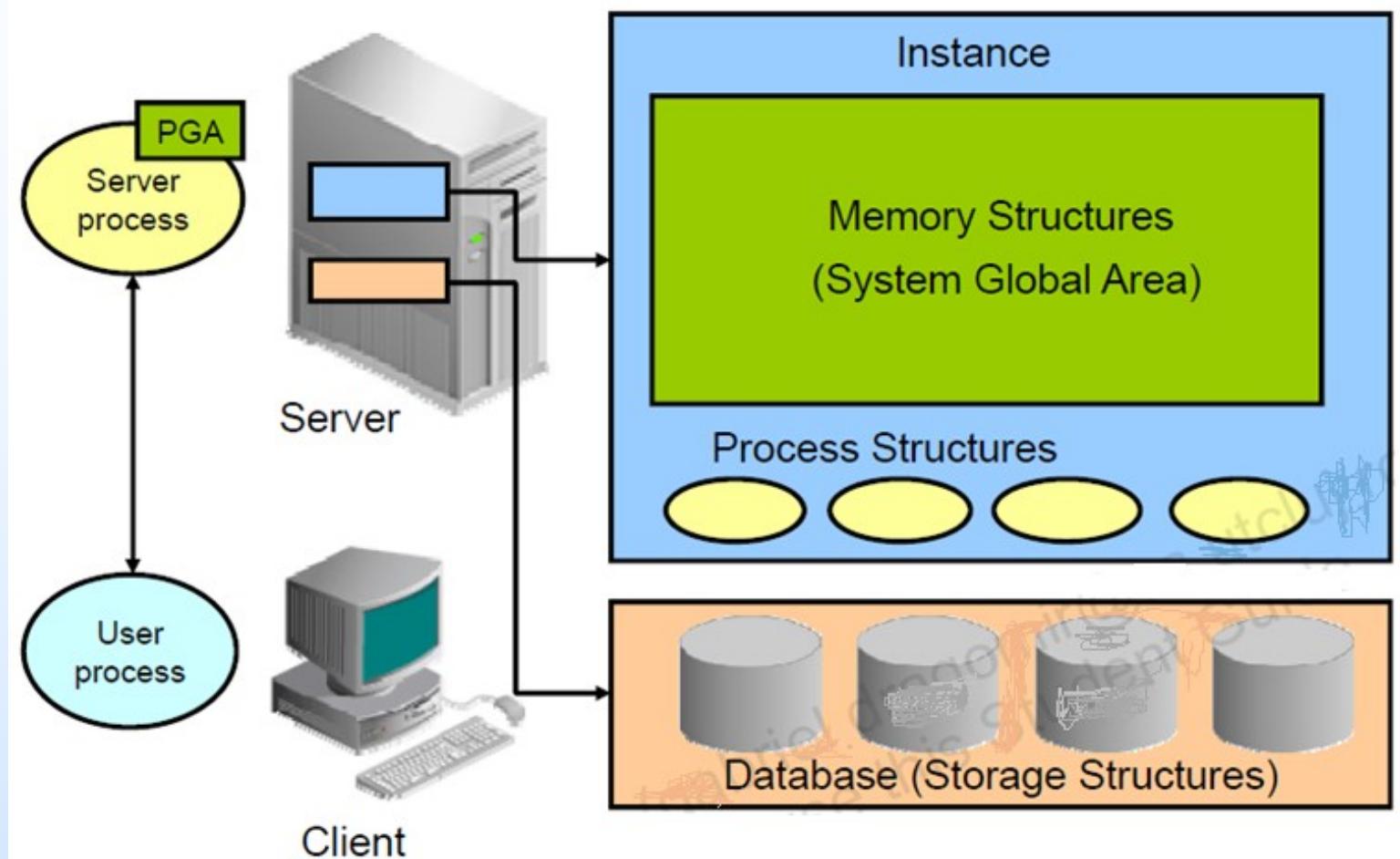
http://docs.oracle.com/cd/E24628_01/index.htm

Unelte de administrare

□ **SQL Developer**

- Este o unealtă GUI pentru accesul la o bază de date Oracle.
- SQL Developer permite editare/rulare/depanare de scripturi SQL și PL/SQL.
- SQL Developer permite browse la obiecte bază de date: tabele, vederi, trigere, proceduri stocate, etc.
- Permite crearea și rularea de rapoarte, predefinite și definite de utilizator.
- <http://docs.oracle.com/database/121/RPTUG/toc.htm>

Arhitectura Server Oracle



Arhitectura Server Oracle

□ Componente importante

- Structuri de memorie
- Procese
- Structuri de stocare

□ Instantă

- Structuri de memorie
- Procese background

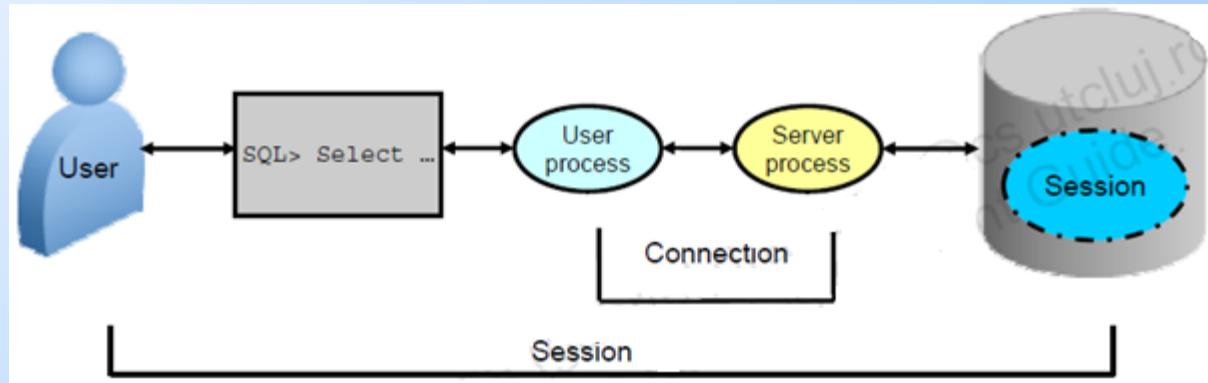
□ Baza de date

- Structuri logice
- Structuri fizice

Arhitectura Server Oracle

□ Cum funcționează?

- Se pornește o instanță de baze de date - start:
 - Este alocată o zonă de memorie partajată, SGA (System Global Area)
 - Sunt pornite procese background
- Se asociază instanța cu o bază de date – mount
- Se deschide baza de date pentru accesul utilizatorilor

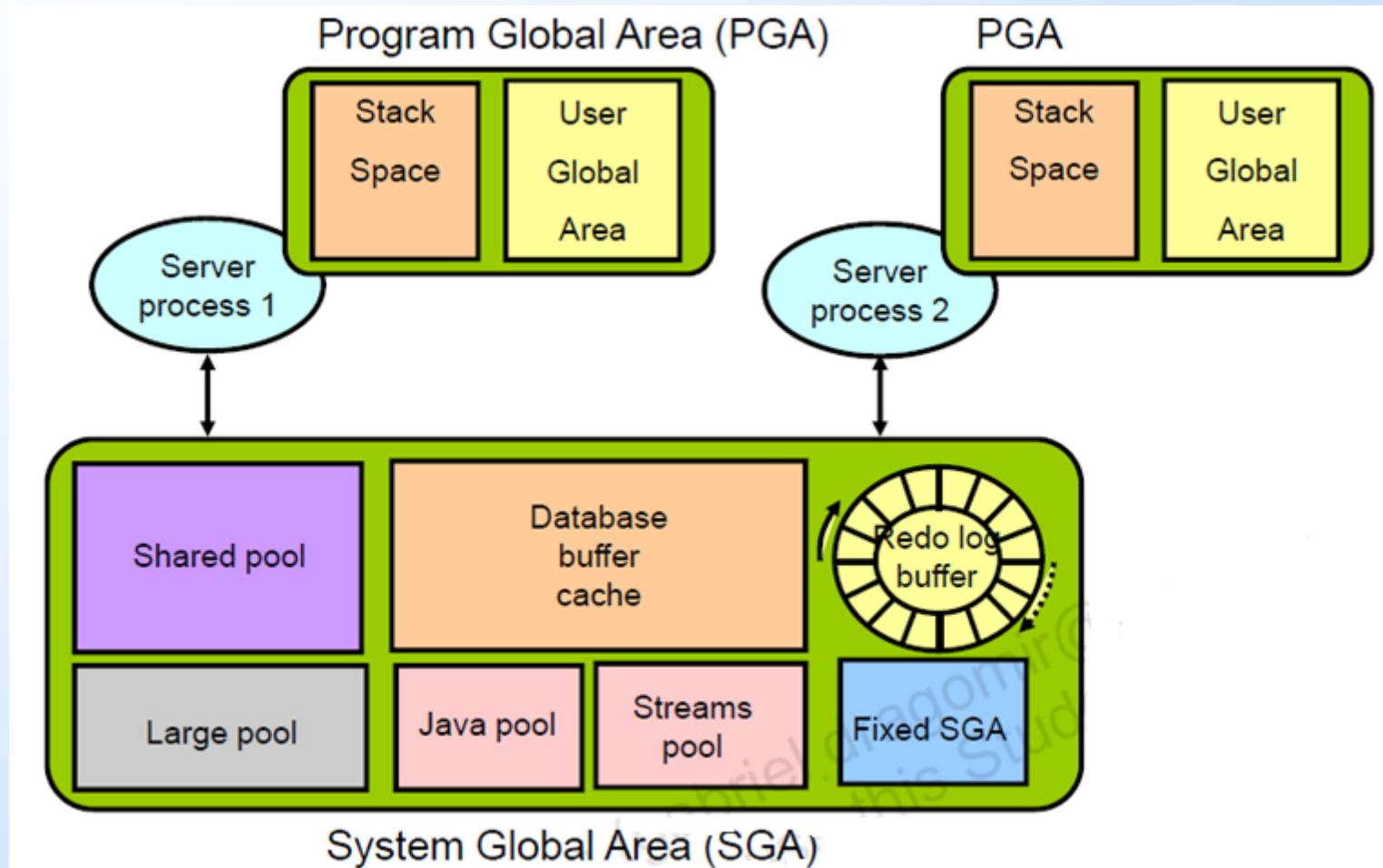


Arhitectura Server Oracle

□ Cum funcționează? (continuare)

- Un utilizator pornește SQL *Plus, furnizează nume și parolă
- Dacă numele și parola sunt valide, se stabilește o sesiune pentru acel utilizator
- Conexiunea este calea de comunicare între procesul utilizator și o instanță bază de date
- Sesiunea durează până ce utilizatorul se deconectează (în sqlplus cu exit) sau ieșe din aplicație
- Pot exista mai multe sesiuni concurente pentru același nume de utilizator

Structuri de memorie



Structuri de memorie

□ SGA

- Este o zonă de memorie partajată ce conține date și informații de control pentru 1 instanță bază de date
- Este partajată de mai multe procese server și procese background
- Conține printre altele blocuri de date ținute în cache și zone SQL partajate

□ PGA

- Este o zonă de memorie nepartajată ce conține date și informații de control pentru 1 singur proces (server sau background)

System Global Area

- Shared pool
 - Ține în cache diferite structuri ce pot fi partajate între utilizatori (biblioteci, dicționar de date, rezultat interogări, și.a.)
- Database buffer cache
 - Ține în cache blocuri de date citite de pe disc (din BD fizică)
- Redo log buffer
 - Ține în cache informații folosite pentru recuperarea instanței până când aceste informații pot fi scrise în fișierele jurnal, pe disc (de procesul log writer)

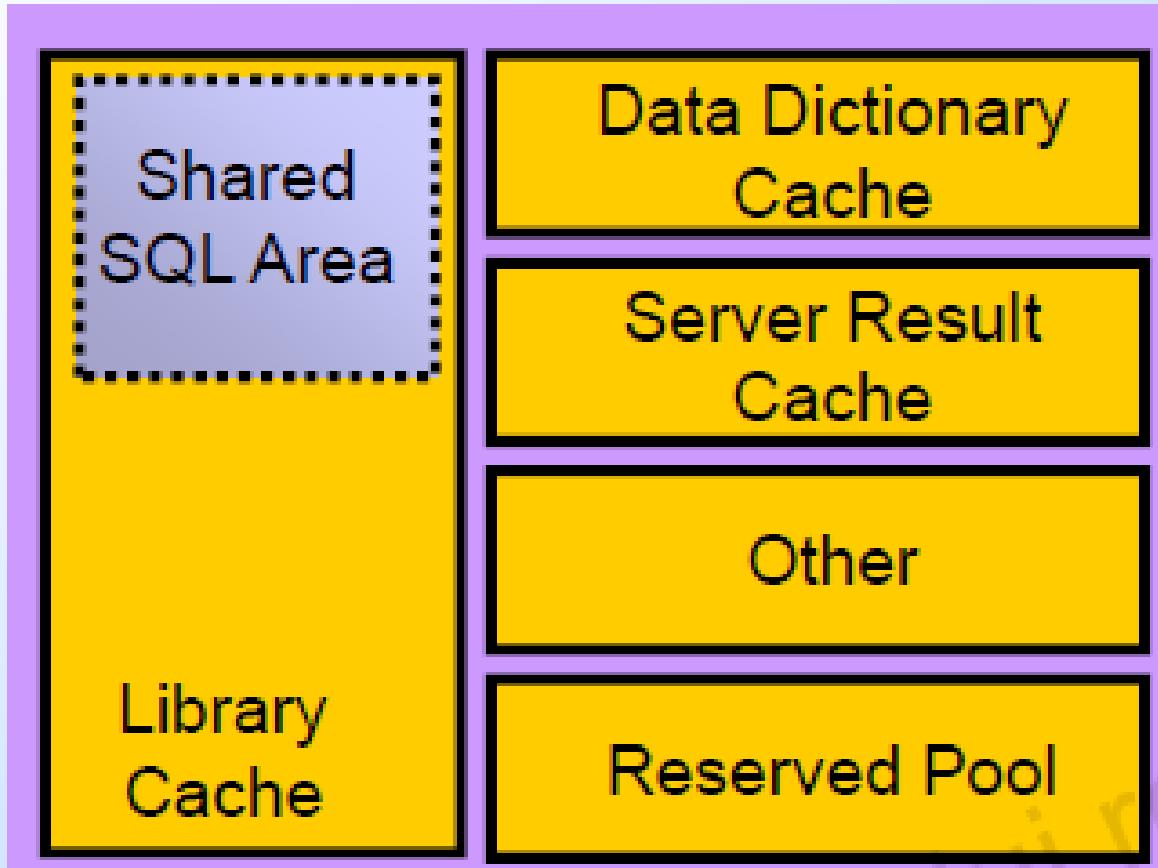
System Global Area

- Large pool
 - Este o zonă opțională folosită pentru alocări masive de memorie la procese mari cum sunt backup și recovery respectiv procese server I/O
- Java pool
 - Este o zonă folosită pentru cod și date Java specific de sesiune în Java Virtual Machine (JVM)
- Streams pool
 - Este o zonă folosită de Oracle Streams pentru capture și apply
- Fixed-SGA – informații de stare BD și instanță, informații comunicate între procese

Program Global Area

- Se alocă atunci când se pornește un proces server
 - Stivă
 - User Global Area
- Este indicat să se permită sistemului să gestioneze automat felul în care se alocă memorie
- Cu parametrii de initializare `MEMORY_TARGET` și `MEMORY_MAX_TARGET` se poate controla cel mai simplu cantitatea de memorie minimă/maximă

Shared pool

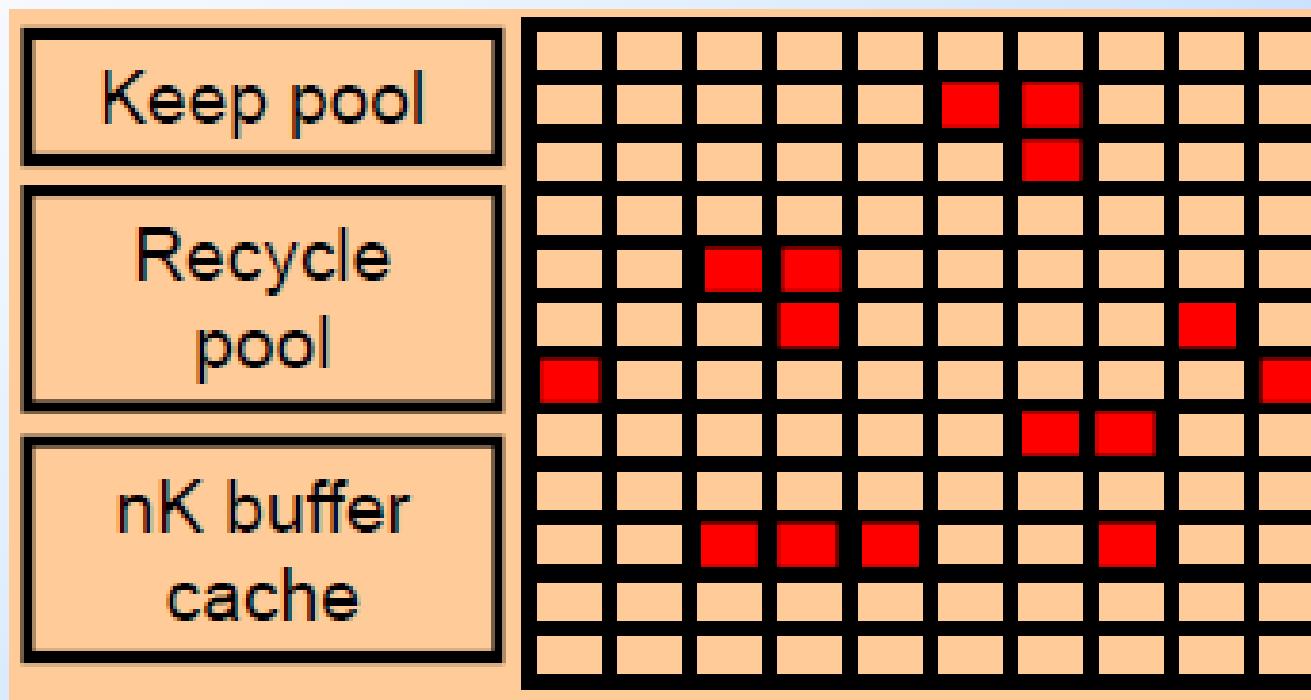


Shared pool

□ Shared SQL Area

- Dacă o instrucțiune SQL se repetă, este reutilizat planul de execuție (generat la prima folosire de optimizator)
- Dacă mai mulți utilizatori folosesc aceeași aplicație, este frecvent ca o instrucțiune SQL să se repete
- Același scenariu se întâmplă și pentru blocuri PL-SQL anonte, proceduri stocate, package-uri și trigere (numite program unit)
- În zona de memorie privată fiecărui proces server (ce corespunde unui utilizator) se țin valorile de date specifice aceluui utilizator

Database buffer cache



Database buffer cache

- La prima solicitare de către un proces server (utilizator) a unei bucăți de informație, se caută în database buffer cache
- Dacă există în cache (hit) se citește direct din memorie
- Dacă nu există în cache (miss) se copiază blocul de pe disc într-un buffer din cache
- Bufferele din cache sunt gestionate de un algoritm complex ce combină Least Recently Used (LRU) lists și touch count

Redo Log Buffer

- Este un buffer circular în SGA ce păstrează informații despre modificările asupra BD prin operații LDD, operații LMD și operații interne.
- Se folosește pentru recuperarea BD în caz de incidente.
- Cum funcționează?
 - Procesul server modifică blocurile de date din buffer cache
 - Sunt generate intrări în redo log buffer
 - Procesul background log writer scrie redo log buffer în fișierul redo log activ (sau grupul de fișiere) pe disc

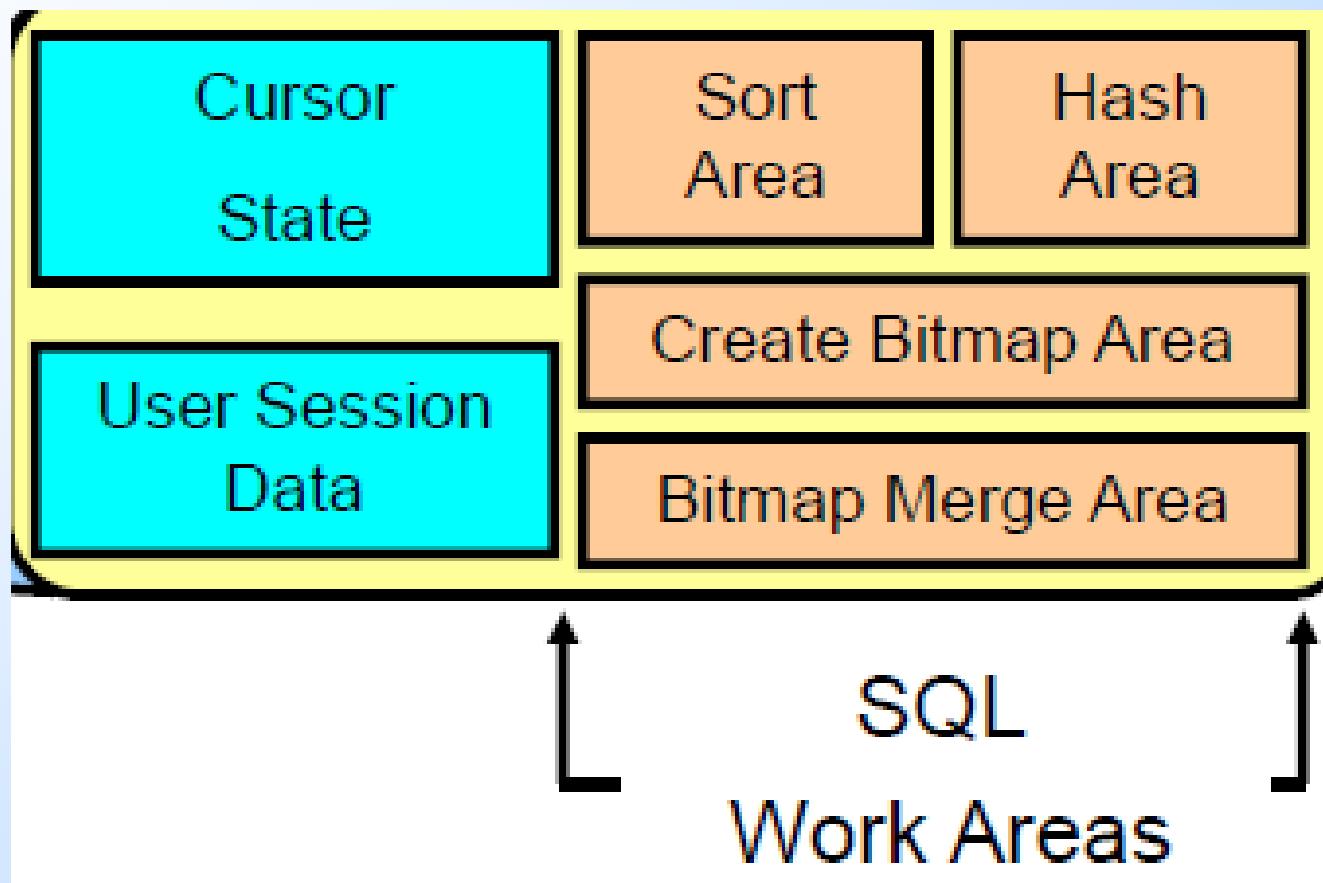
Large pool

| I/O buffer | Free memory | Parallel Query |
|----------------|---------------|------------------|
| Response queue | Request queue | Advanced Queuing |

Large pool

- Este o zonă optională de memorie, pe care administratorul BD o poate configura pentru:
 - Memorie de sesiune pentru serverul partajat și interfața Oracle XA (tranzacții ce interacționează cu mai multe servere)
 - Procese server I/O
 - Operații backup/restore
 - Operații de interogare paralele
 - Advance Queuing memory table storage
- Se recomandă folosirea large pool în loc de shared pool pentru volum mare de operații

User Global Area



User Global Area

- Este o subdiviziune a memoriei PGA (pe lângă stack space)
- Fiecare proces server (sesiune utilizator) are o astfel de zonă de memorie
- UGA are în componentă:
 - Cursor area – conține informații runtime despre cursoare
 - User session storage data area – conține informații de control despre sesiune
 - SQL working areas – conține informații pentru procesarea instrucțiunilor SQL

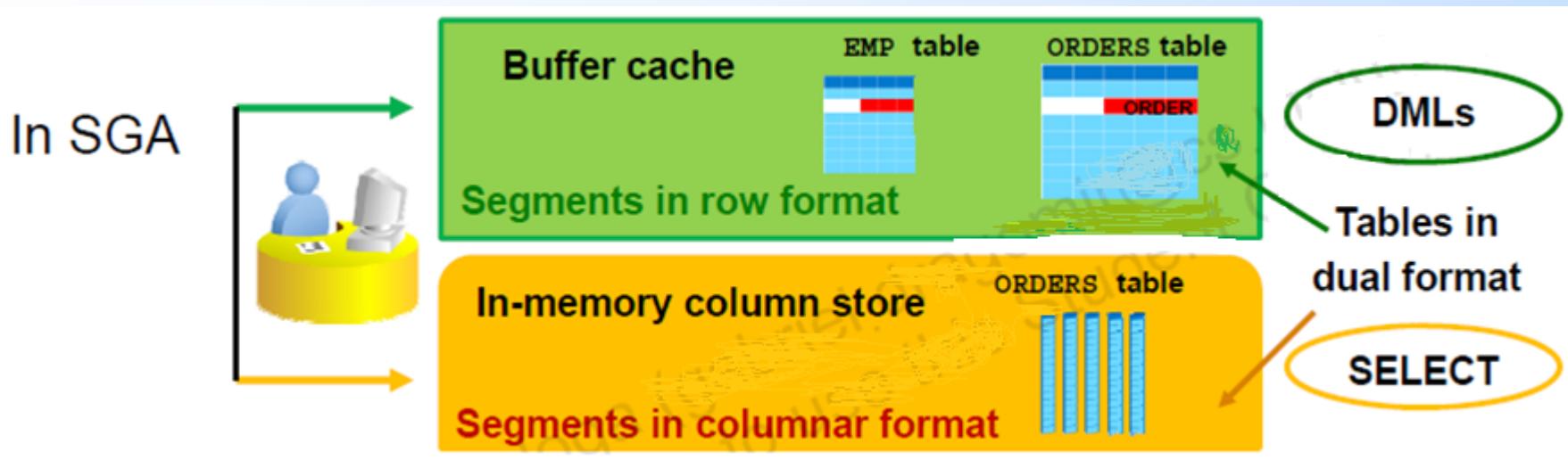
User Global Area

□ SQL working areas – conține la rândul său:

- O zonă de sortare – pentru ORDER BY și GROUP BY
- O zonă hash - pentru operații join
- O zonă bitmap - pentru indecsi în datawarehouse
- O zonă bitmap merge – folosită la plan de execuție cu bitmap index

□ În mediu server partajat, când mai mulți utilizatori partajează același proces server, UGA poate fi mutat în SGA (shared pool sau large pool) și atunci PGA conține doar stiva (stack space)

In-Memory column store



- Permite efectuarea de interogări analitice ad-hoc direct pe date tranzacționale în timp real
- Administratorul bazei de date decide ce segmente să fie ținute in-memory pe baza tipului interogărilor

In-Memory column store

□ Row format exclusiv

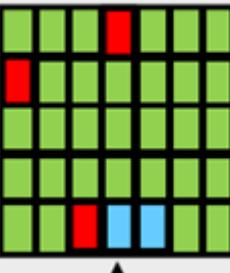
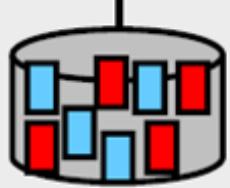
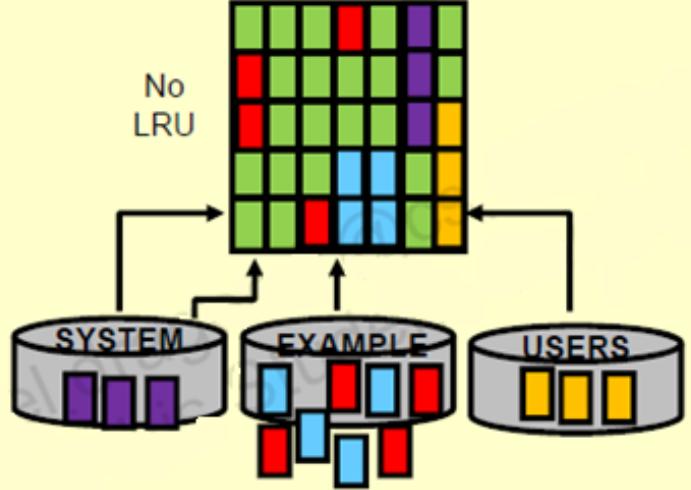
□ Segmente accesate în mod frecvent de interogări OLTP, ce operează pe puține linii și returnează multe coloane, ar trebui puse în buffer cache

□ Dual format simultan

□ Segmente accesate de interogări analitice (characteristic DSS), ce operază pe multe linii și returnează puține coloane, sunt candidate pentru column store

□ Un segment definit in-memory, dar asupra căruia se efectuează și interogări tip OLTP, se păstrează și în buffer cache, interogările „fetch-by-rowid” se adresează la buffer cache pentru date

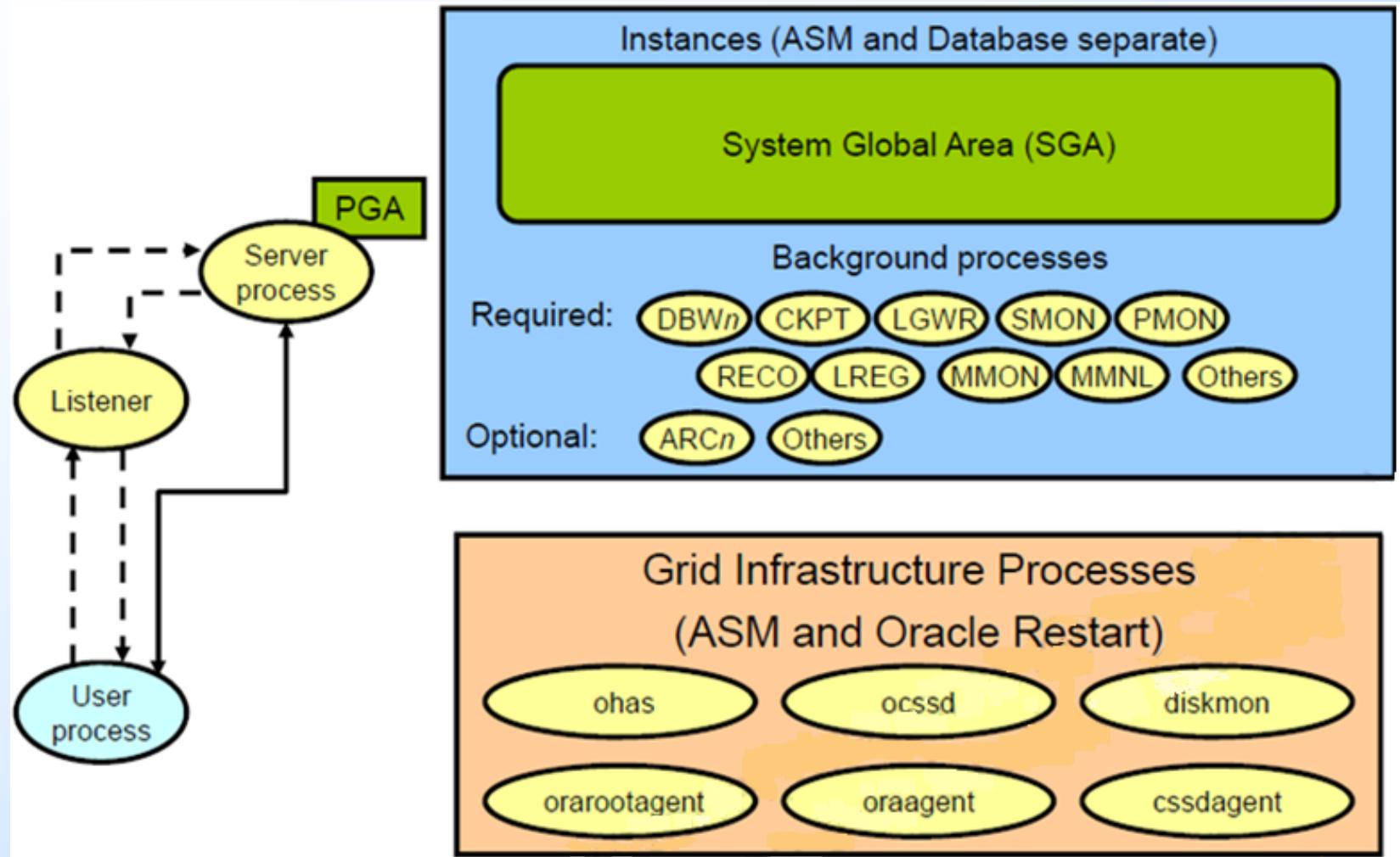
Full database In-Memory caching

| Traditional Buffer Cache Usage | Full Database In-Memory Caching |
|--|--|
| <p>DB_CACHE_SIZE= 10g</p> <p>LRU algo</p>  <p>Scans + OLTP</p> <p>Loaded in buffer cache if table size < small % of buffer cache size</p>  <p>Table HR.EMPLOYEES Table SH.SALES</p> | <p>Entire database loaded into the buffer cache:</p> <ul style="list-style-type: none">• Huge performance benefits• Two modes<ul style="list-style-type: none">– Full Database Caching– Force Full Database Caching  |

Full database In-Memory caching

- Algoritmul „table scan” încarcă o tabelă în buffer cache doar dacă dimensiunea tabelei este mai mică decât un mic procent din dimensiunea buffer cache, pentru tabele mari se folosește metoda „direct path read”, ce încarcă blocurile direct în PGA, evitând SGA pentru a nu umple buffer cache.
- Opțiunea din titlu se folosește când suma dimensiunii fișierelor de date + tablespace-ul SYSTEM + fișiere LOB cache – SYSAUX – TEMP este mai mică decât dimensiunea buffer cache

Structura proceselor



Structura proceselor

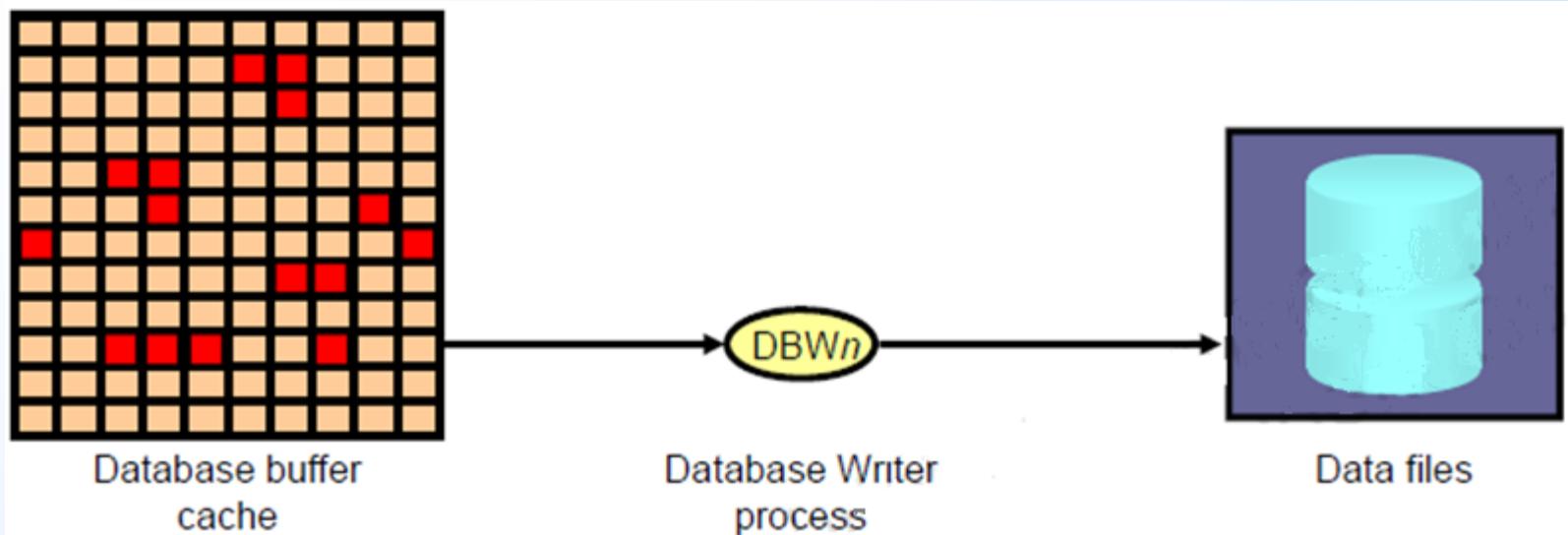
□ User process

- Este aplicația client care încearcă să se conecteze la baza de date Oracle
- Poate fi pe aceeași mașină cu baza de date sau poate fi pe o mașină la distanță, când se conectează prin rețea
- Prima dată user process comunică cu procesul Listener care creează procesul server într-un mediu dedicat
- În continuare user process comunică direct cu procesul server

Structura proceselor

- Server process
 - Analizează și execută instrucțiuni SQL
 - Citește blocuri de date din BD fizică (fișierele de pe disc) în memorie (în bufere partajate din SGA) dacă blocurile nu sunt deja în memorie
 - Returnează rezultatele instrucțiunilor SQL aplicației care a emis cererea
- Background processes
 - Database Writer Process (DBWn), Log Writer Process (LGWR), Checkpoint Process (CKPT), System Monitor Process (SMON), Process Monitor Process (PMON), Recoverer Process (RECO), Listener Registration Process (LREG), Manageability Monitor Process (MMON), Manageability Monitor Lite Process (MMNL), Job Queue Processes, Archiver Processes (ARCn), Queue Monitor Processes (QMNN)

Database Writer Process

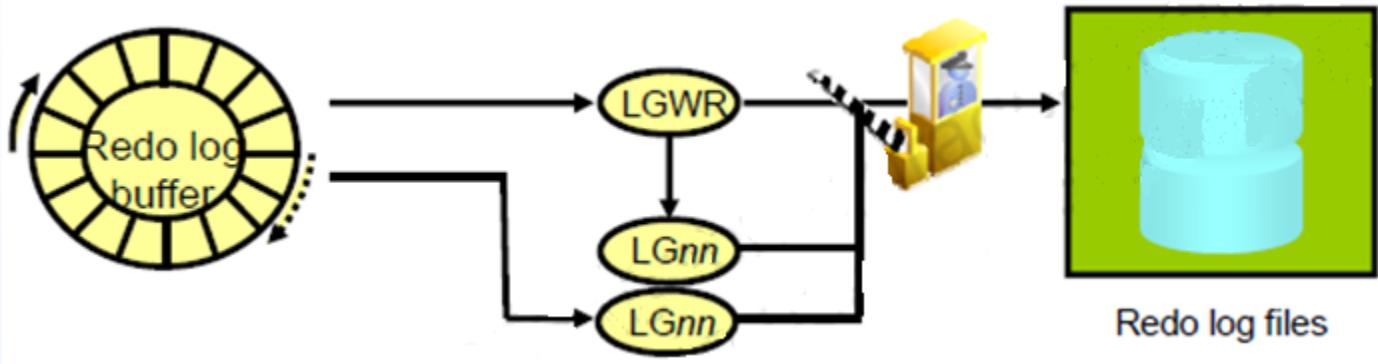


- Pot exista mai multe procese (maxim 100, `DB_WRITER_PROCESSES`).
- Când se modifică un buffer din cache, se marchează dirty, se pune în vârful cozii checkpoint (în ordinea system change number, ce corespunde cu ordinea redo log), iar când numărul buferelor disponibile din cache scade sub un prag prestabilit, scrie asincron pe disc buferele cele mai rar folosite (LRU)

Database Writer Process

- SGA are o structură de memorie ce păstrează redo byte address (RBA), poziția în stream-ul redo de unde ar trebui să înceapă operația recovery în caz de incident (cădere de sistem)
- Procesul CKPT scrie la fiecare trei secunde în fișierul de control această structură de memorie
- Procesul DBWn după ce scrie buferele dirty din lista LRU avansează pointerul redo a.î. la recovery să se evite operații I/O inutile (se numește checkpoint incremental)

Log Writer Process



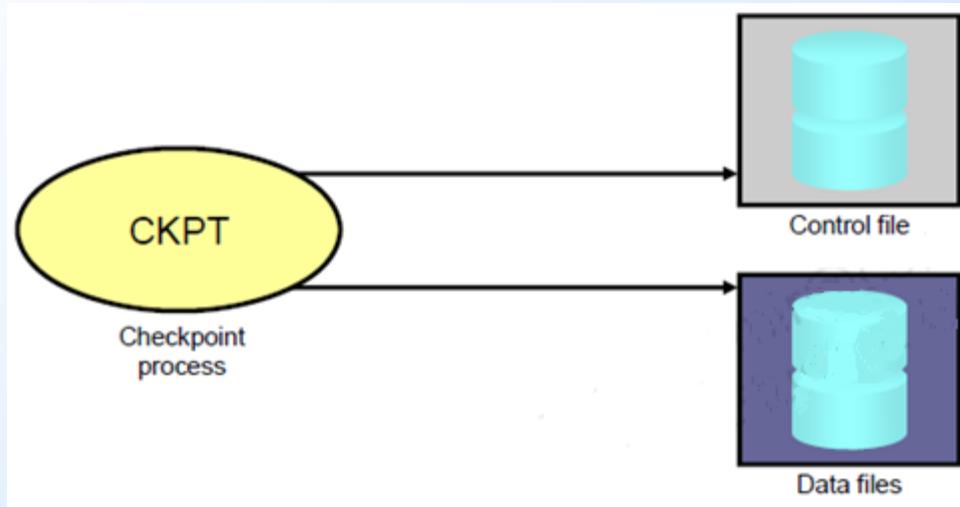
- Buferul redo log este scris pe disc când:
 - Un proces utilizator emite commit
 - A apărut online redo log switch (de exemplu când fișierul curent redo log se umple)
 - Când buferul redo log se umple la 1/3 sau 1MB
 - Înainte ca un proces DBWn să scrie datele pe disc
 - La 3 secunde de la scrierea precedentă

Log Writer Process

□ Mecanismul fast commit:

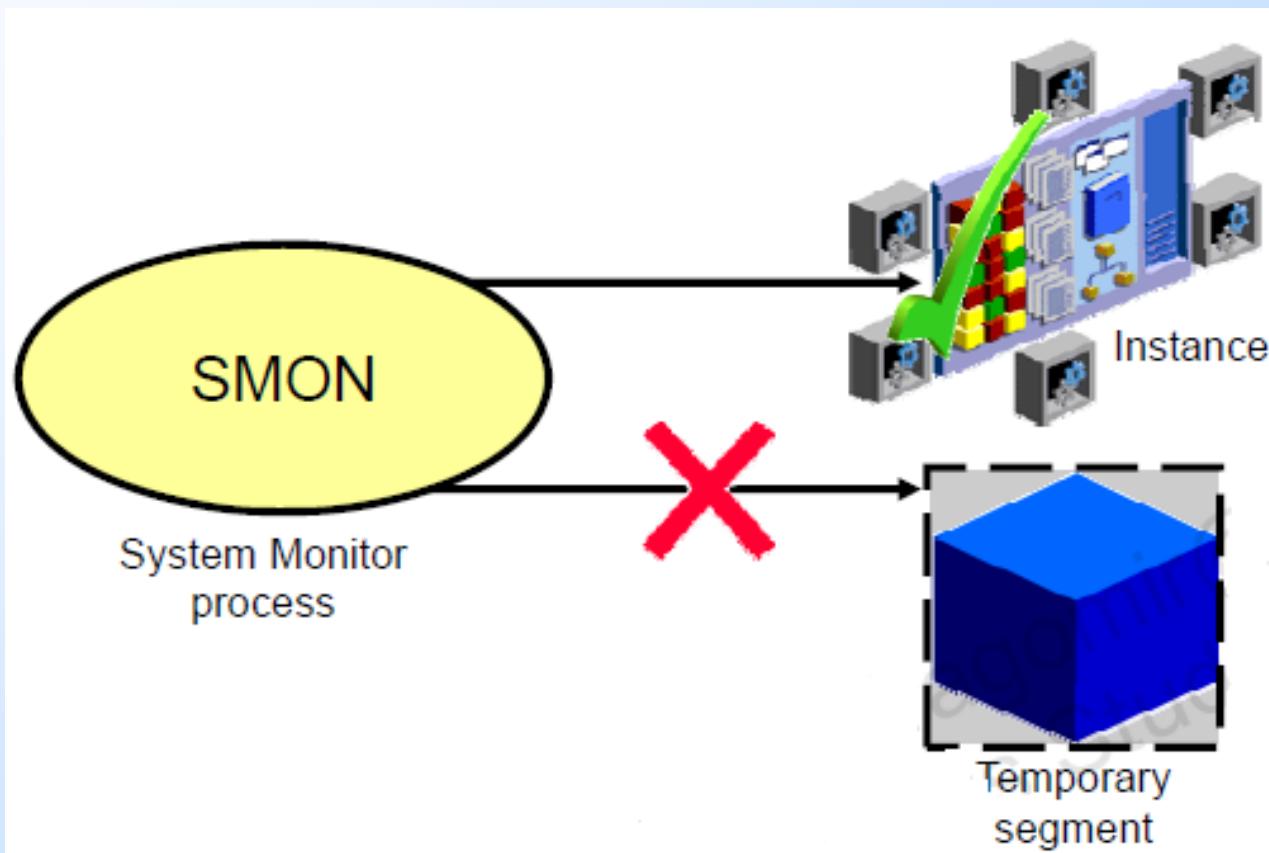
- Când un utilizator emite commit LGWR pune o înregistrare commit în redo log buffer și o scrie imediat pe disc împreună cu intrările redo ale tranzacției
- Modificările corespunzătoare ale blocurilor de date pe disc sunt întârziate până la un moment când este eficient să fie scrise
- Scrierea înregistrării commit este singurul eveniment ce spune că tranzacția s-a încheiat cu succes deși buferele cu date încă nu au fost scrise pe disc

Checkpoint Process



- Un checkpoint este o structură de date ce definește un system change number în thread-ul redo
- Checkpoint-urile sunt scrise în fișierul de control și în fiecare header al fișierelor de date
- Scrierea SCNs în headerele fișierelor de date asigură că modificările efectuate blocurilor înainte de acel SCN sunt salvate pe disc (de DBWn)

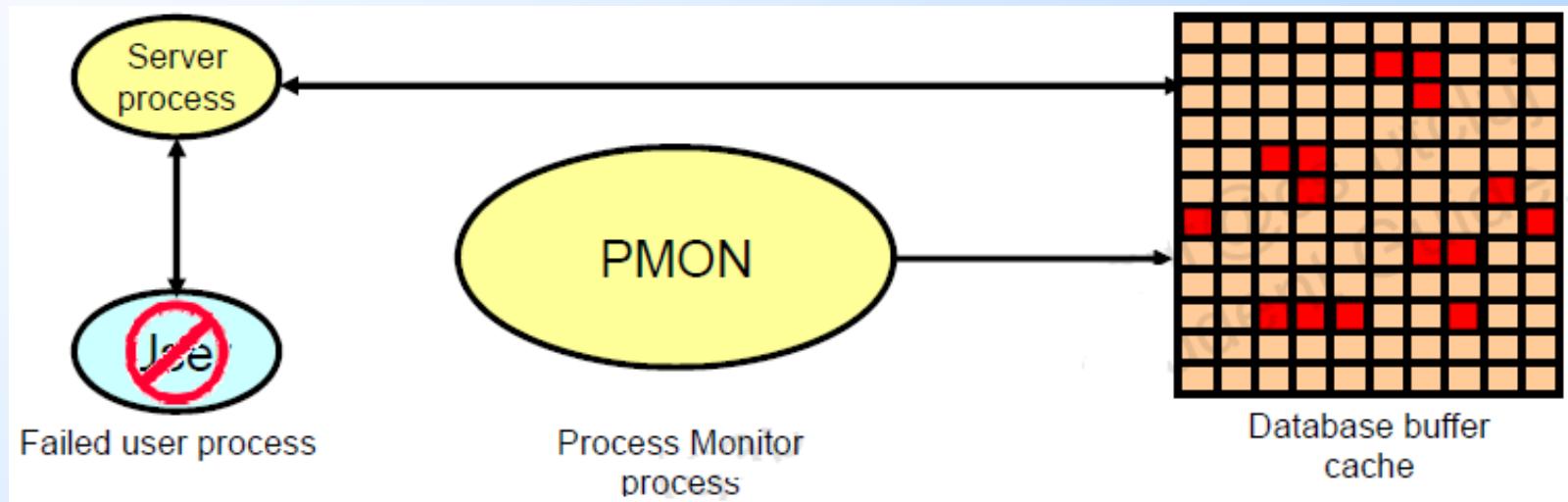
System Monitor Process



System Monitor Process

- Efectuează recovery la startup instantă, dacă este necesar
- Curăță segmentele temporare ce nu se mai folosesc
- Dacă anumite tranzacții ce au fost încheiate cu succes nu au fost prelucrate la recovery din motiv de eroare file-read sau eroare offline, SMON le recuperează când tablespace-ul sau fișierul de date este on-line din nou

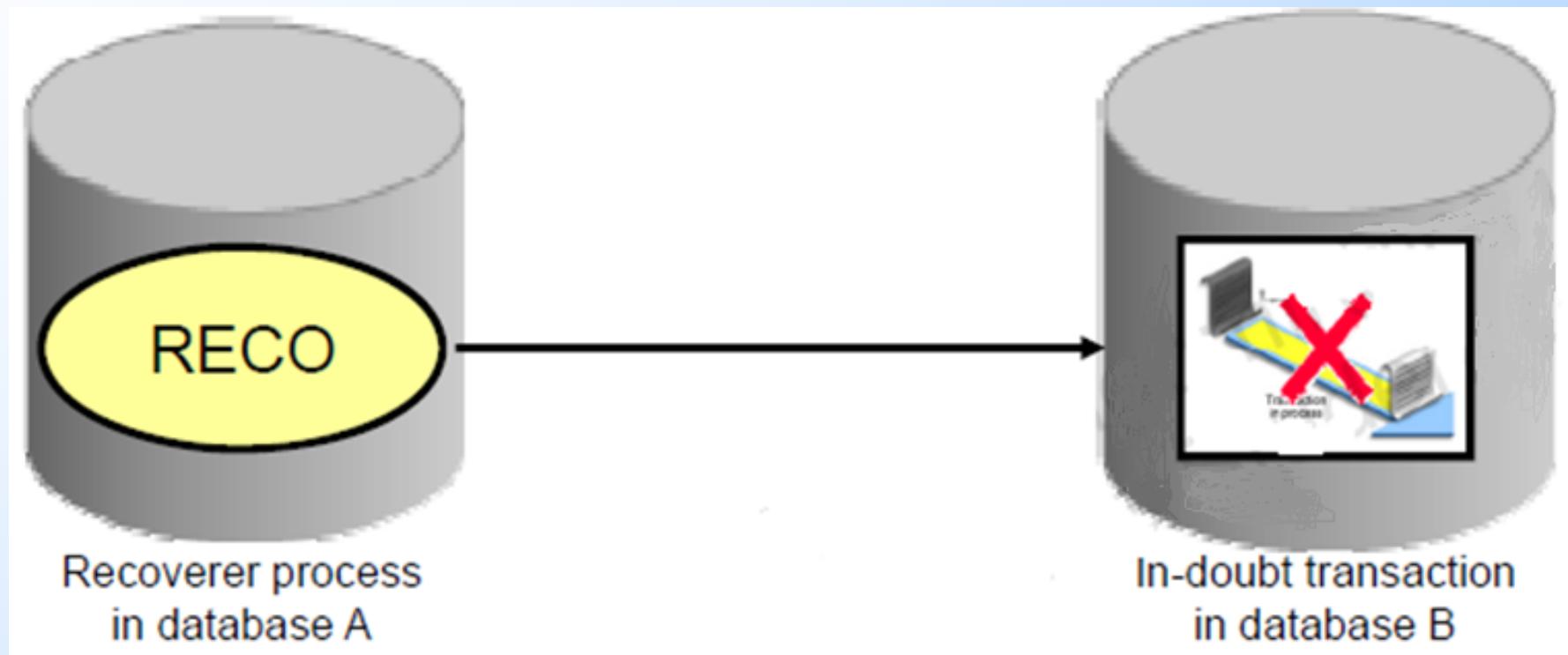
Process Monitor process



Process Monitor process

- Efectuează recovery când un proces utilizator eșuează
- Curăță buffer cache și eliberează resursele acaparate de procesul utilizator care a eșuat
 - resetează starea tabelei de tranzacții active,
 - eliberează lock-urile,
 - șterge ID-ul procesului din lista de proceze active
- PMON testează periodic starea proceselor dispecer și server și le restartează dacă sunt oprite (dar nu și dacă au fost oprite intenționat de Oracle Database)

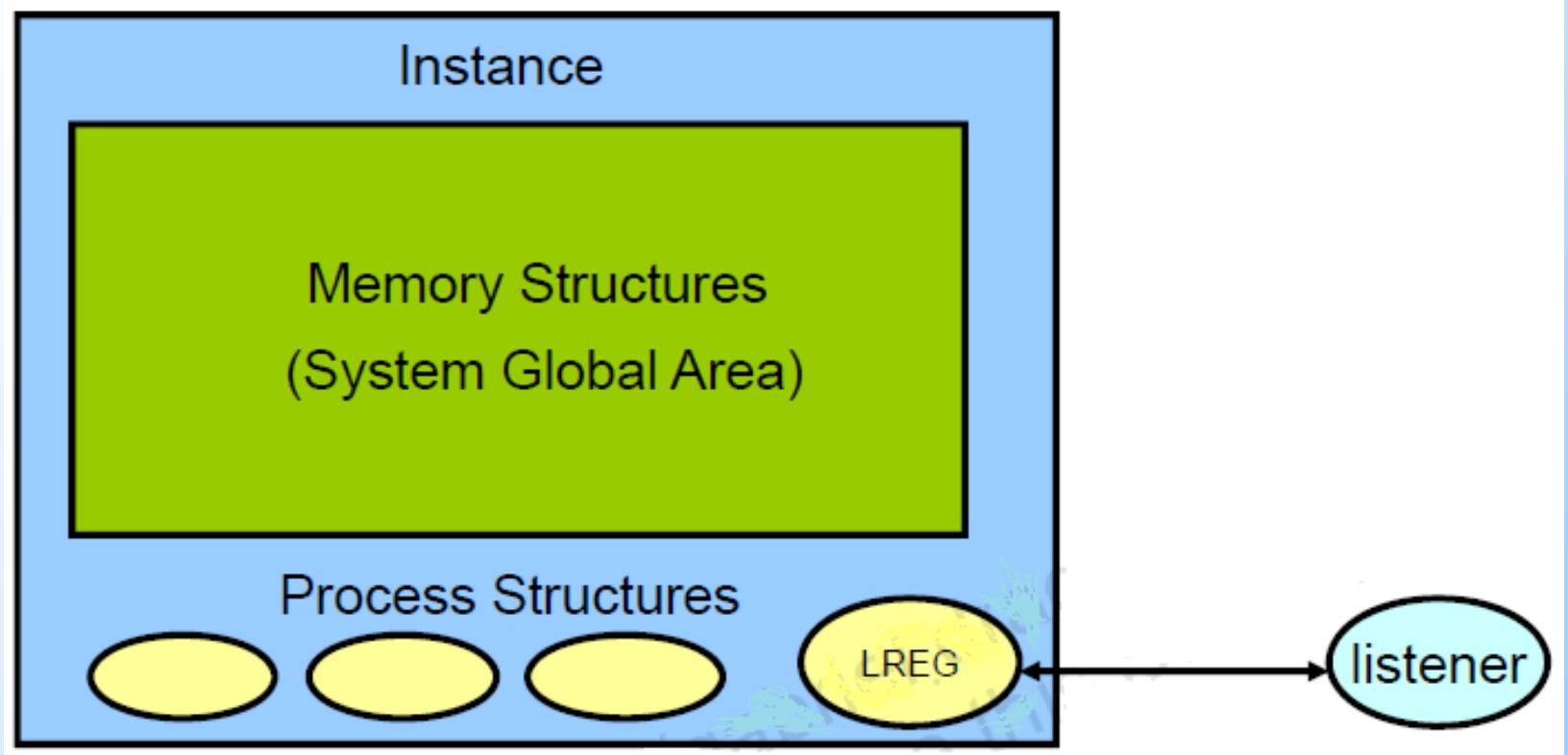
Recoverer Process



Recoverer Process

- Este un proces background
- Este folosit în mediu distribuit pentru a recupera tranzacții distribuite
- Procesul RECO se conectează automat la servere baze de date implicate într-o tranzacție distribuită „in-doubt”, rezolvă tranzacțiile, după care elimină rândurile de date corespunzătoare din tabela „pending transaction”

Listener Registration Process



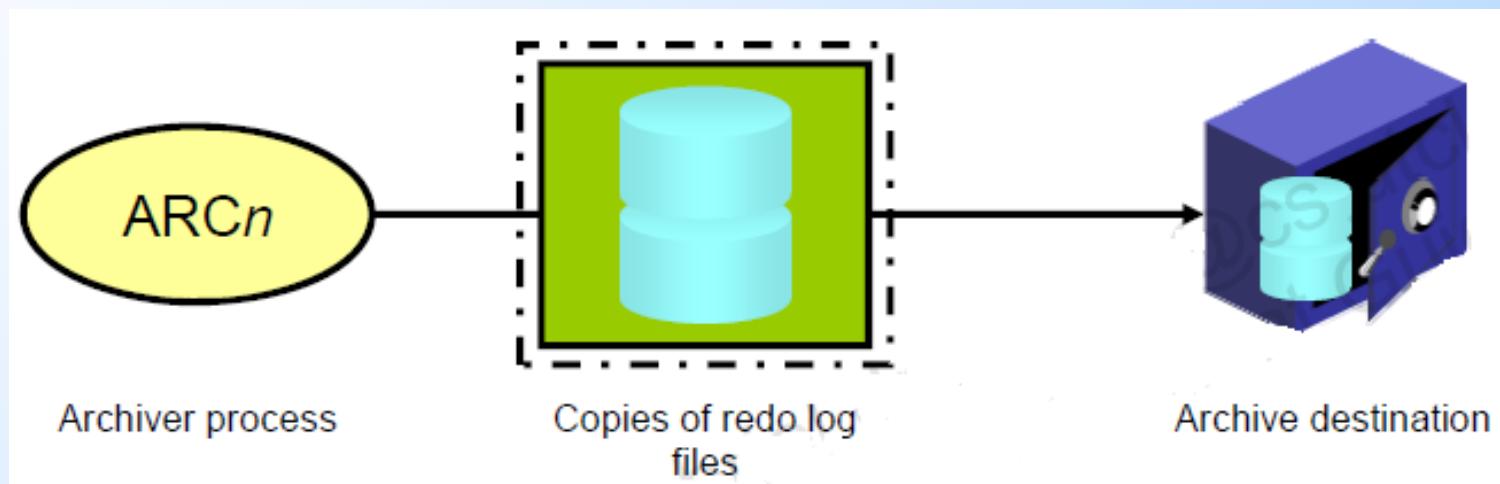
Listener Registration Process

- LREG înregistrează informații despre instanța BD și procesul dispecer cu Oracle Net Listener
 - Numele serviciilor BD
 - Numele instanței BD asociate cu serviciile împreună cu încărcarea curentă și maximă
 - Handlere de servicii (dispeceri și servere dedicate), incluzând: tip, adrese per protocol, încărcarea curentă și maximă

Listener Registration Process

- La startup instantă LREG încearcă să se conecteze la listener
 - Dacă listener funcționează, atunci îi transmite informații
 - Altfel, reîncearcă periodic să se conecteze la el (poate dura 60 sec. după pornire listener ca să se înregistreze instanța BD la el)
- ALTER SYSTEM REGISTER
 - Este comanda ce înregistrează imediat servicii la un listener

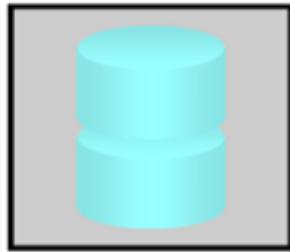
Archiver Process



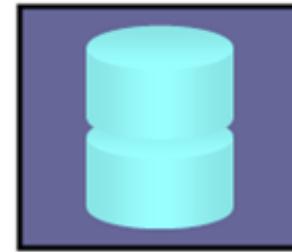
Archiver Process

- Copiază fișierele redo log pe un anumit mediu de stocare după apariția unui switch log
- Poate colecta date pentru redo tranzacție și să le transmită la destinații de sine stătătoare
- Procesele ARChLOG sunt disponibile numai dacă BD este în mod ARCHIVELOG, deși s-a setat enable arhivarea automată
- Se poate mări numărul de procese de arhivare (implicit 4)
 - De exemplu la operații de încărcare date „bulk”
- Pot exista destinații multiple pentru arhivare
 - Se recomandă 1 proces/1 destinație

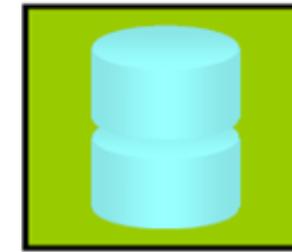
Arhitectura de stocare Oracle



Control files



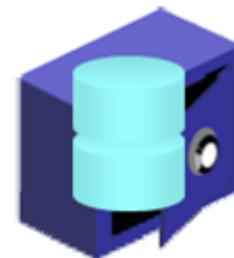
Data files



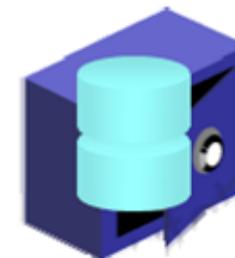
Online redo log files



Parameter file



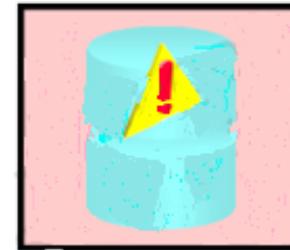
Backup files



Archived redo log
files



Password file



Alert log and trace files

Arhitectura de stocare Oracle

□ Fișiere de control

- 1 fișier de control/1 BD
- Date despre structura fizică a BD
- Se recomandă păstrarea de copii ale fișierului de control, pentru a evita pierderea de informații
- Conține și metadate pentru backup
- O BD nu poate fi deschisă fără fișierul de control

□ Fișiere de date

- Conțin date, metadate și dicționarul datelor

□ Fișiere online redo log

- Se folosesc pentru recovery BD în caz de crash instantă BD

Arhitectura de stocare Oracle

□ Fișiere adiționale

□ Fișier cu parametrii

- La startup instanță BD pentru configurare

□ Fișier cu parole

- Utilizatorii cu roluri SYSDBA, SYSOPER, SYSBACKUP, SYSDG, SYSKM, SYSASM se pot conecta de la distanță pentru a efectua task-uri administrative

□ Fișiere backup

- Imaginea BD folosită la recuperare (starea la un moment dat)

Arhitectura de stocare Oracle

□ Fișiere adiționale (continuare)

□ Fișiere redo log arhivate

- Istoricul modificărilor asupra datelor (împreună cu fișiere backup se folosesc la o recuperare completă a BD)

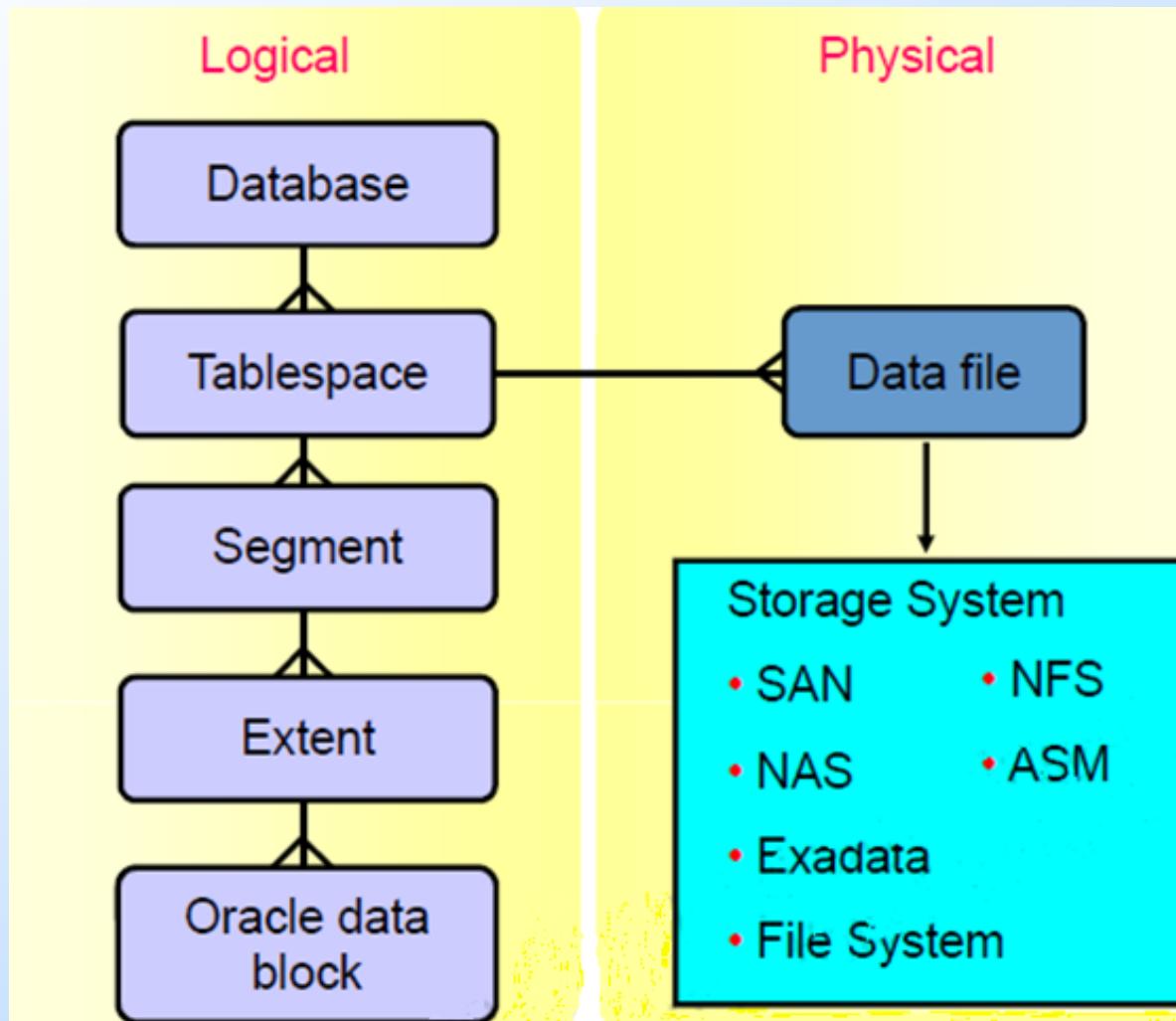
□ Fișiere trace

- Procesul server și procesele background pot scrie într-un fișier trace asociat, la apariția unei erori, informații destinate administratorului BD sau Oracle Support Services

□ Fișier alert log

- Este un jurnal cronologic cu mesajele și erorile
- Se recomandă citirea periodică a acestui fișier

Structuri BD Logice și Fizice



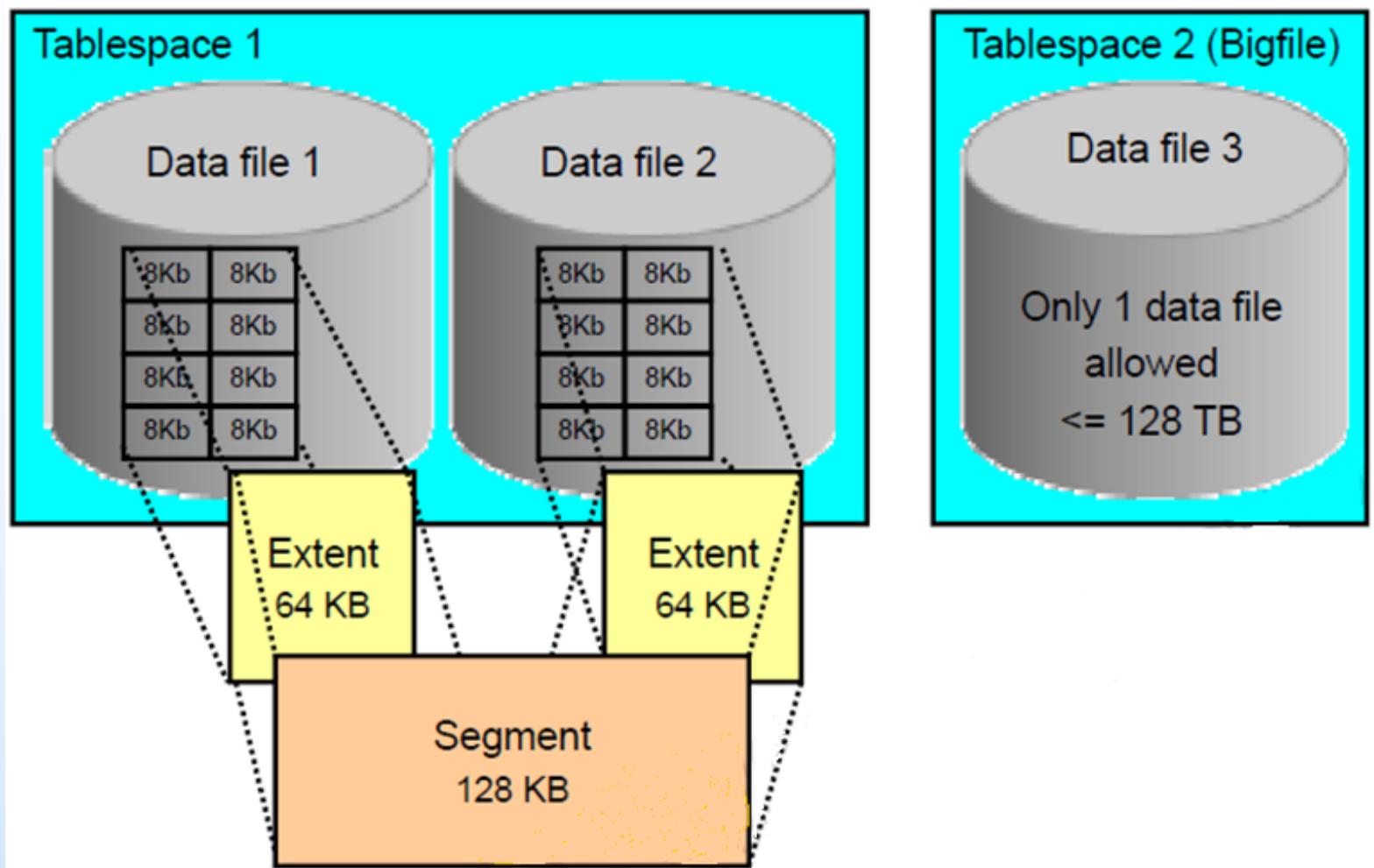
Structuri BD Logice și Fizice

- Fiecare BD fizică este împărțită în tablespace-uri
- Fiecare tablespace este stocat în unul sau mai multe fișiere pe disc
- Fiecare tablespace este constituit din segmente
- Datele sunt ținute în blocuri (numărul de octeți per bloc se specifică la creare tablespace, implicit 8KB) care formează „extent” (zonă contiguă, obținută într-o singură alocare), într-un segment sunt mai multe extent-uri

Structuri BD Logice și Fizice

- Segmente – un set de extent-uri alocat pentru o structură logică de date:
 - Data – o tabelă organizată non-clustered, non-indexed cu excepția external table, tabelă temporară globală și tabelă partitioanată are un segment de tip „Data”
 - Index – fiecare index are segmentul său
 - Undo – fiecare instanță de BD are un tablespace UNDO, un segment undo este folosit pentru „read-consistent database information”, pentru recovery și pentru anularea efectelor tranzacțiilor nefinalizate cu succes
 - Temporary – pentru zone temporare, la execuția comenzi SQL (ORDER BY, GROUP BY), la încheierea execuției comenzi pentru care a fost alocat, spațiul este eliberat

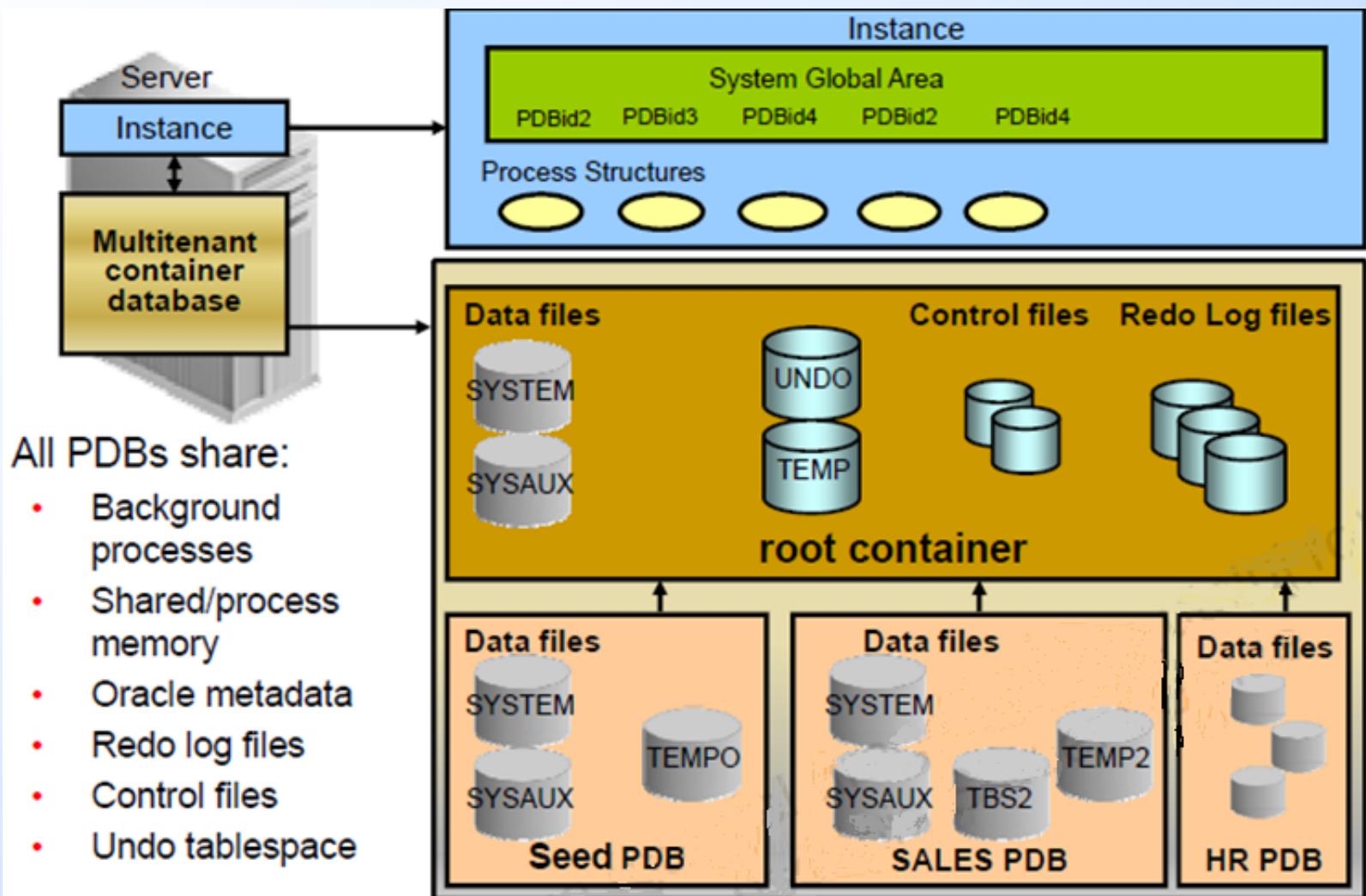
Tablespace-uri și fișiere de Date



Tablespace-urile SYSTEM și SYSAUX

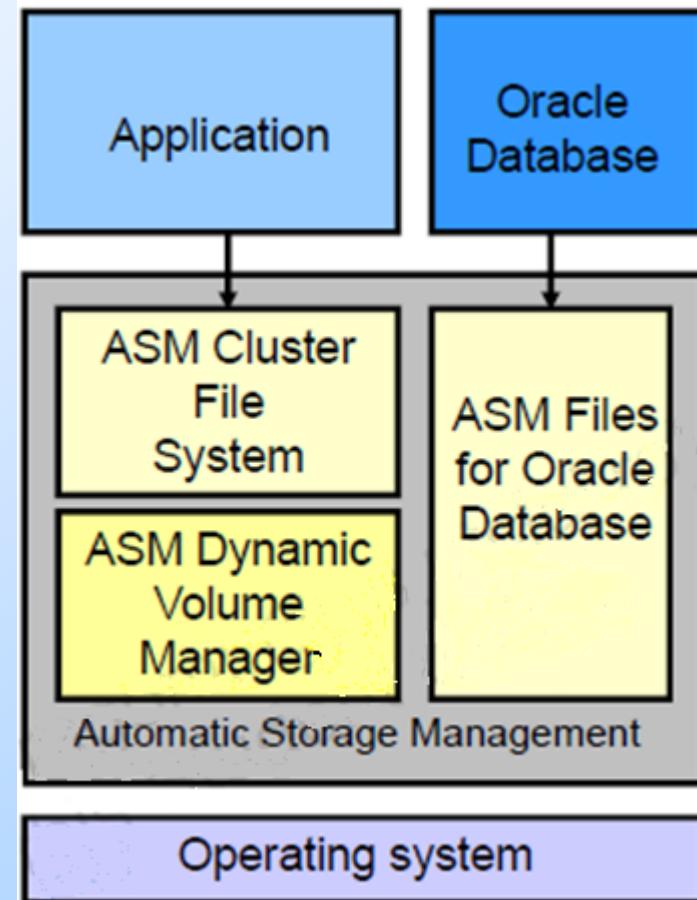
- Sunt create la crearea BD
- Sunt online când BD este „open”
- SYSTEM – conține dicționarul datelor
- SYSAUX – conține alte componente ale BD
- **NU TREBUIE FOLOSITE PENTRU DATE ALE APLICAȚIILOR!**

Arhitectura „Multitenant”



Automatic Storage Management

- Oferă integrarea pe verticală a sistemului de fișiere și gestionarului de volume pentru fișierele BD
- ASM poate distribui datele – performanță și toleranță la defecte
- Ușurează munca administratorului BD



Secvența de pornire server BD

sqlplus / as sysdba

1. SQL>startup

- Pornește instanța, asociază fișierele BD, montează și deschide BD

2. SQL>startup nomount

- Pornește instanța, BD nu este montată

3. SQL>alter database mount;

- Montează BD din starea NOMOUNT

4. SQL>alter database open;

- Deschide BD din starea MOUNT

Moduri shutdown

| Shutdown Modes | A | I | T | N |
|--------------------------------------|----|-----|-----|-----|
| Allows new connections | No | No | No | No |
| Waits until current sessions end | No | No | No | Yes |
| Waits until current transactions end | No | No | Yes | Yes |
| Forces a checkpoint and closes files | No | Yes | Yes | Yes |

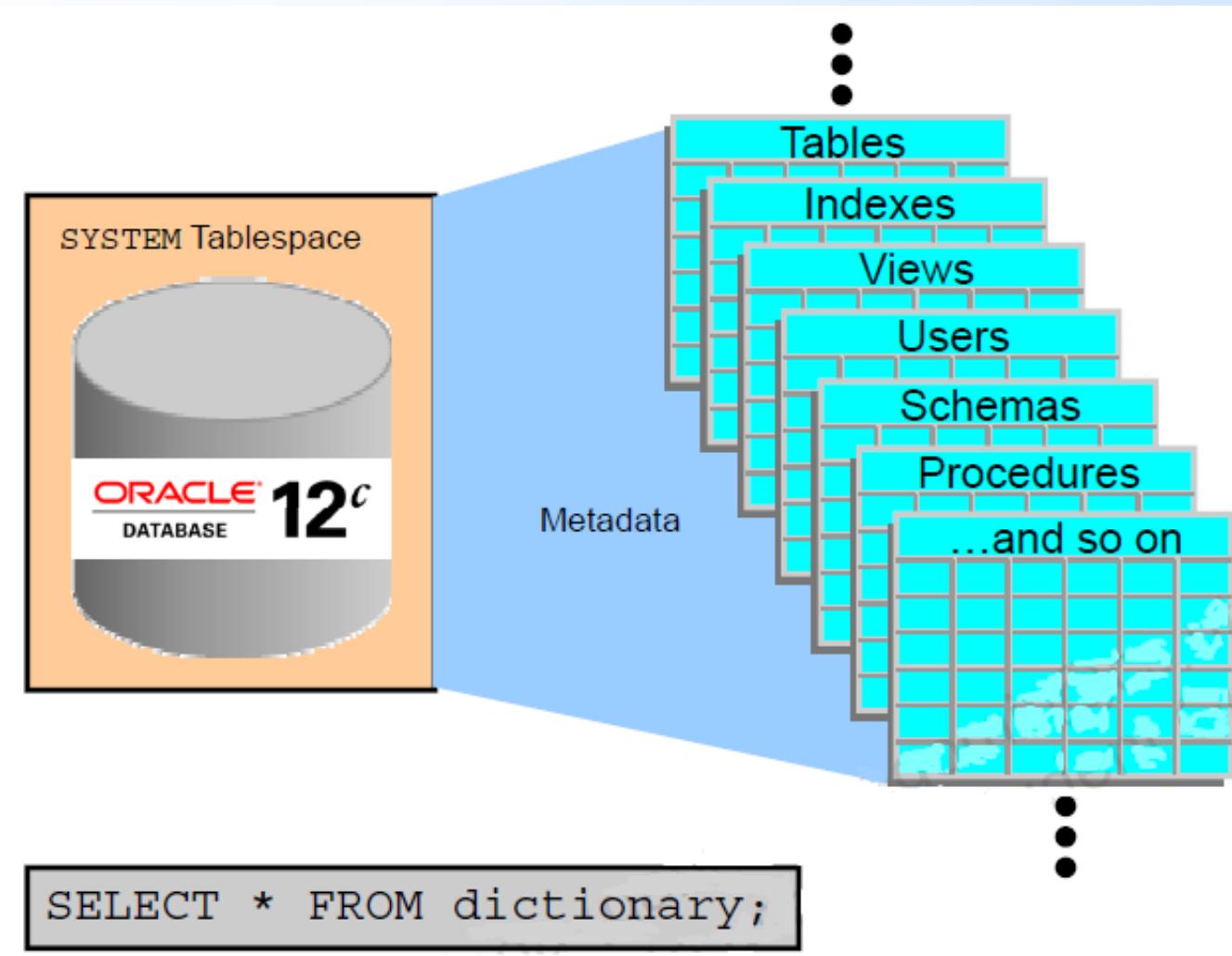
A = Abort (cere recovery la pornire)

I = Immediate (tranzacțiile curente se pierd)

T = Transactional (așteaptă să se termine tranzacțiile curente)

N = Normal (așteaptă sesiunile curente să se încheie)

Dicționarul datelor



Utilizatorul SYS

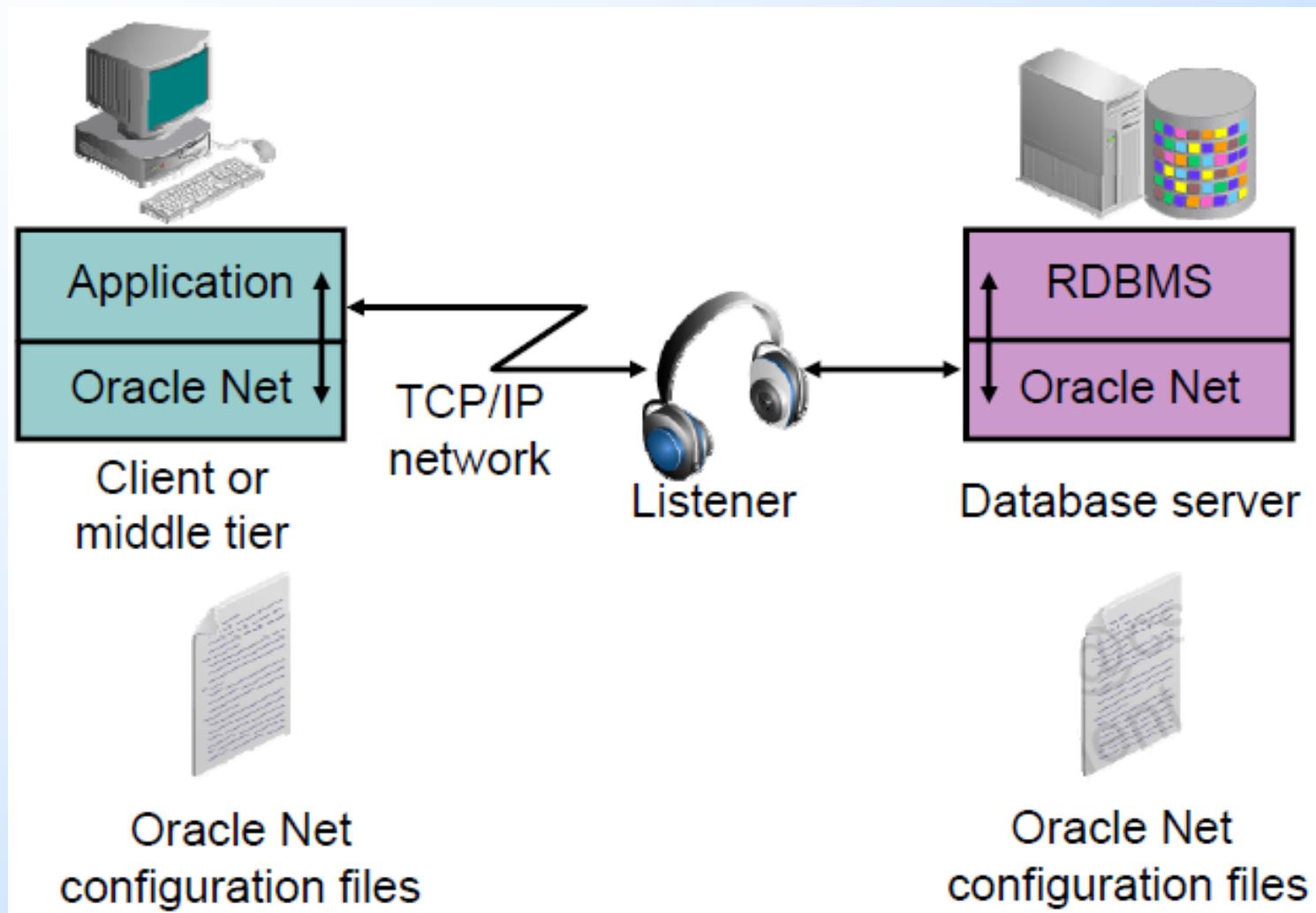
Vederi disponibile în dicționar

| Prefix | User Access | Contents | Notes |
|---------------|-------------------------|--------------------------------------|---|
| DBA_ | Database administrators | All objects | Some DBA_ views have additional columns containing information useful to the administrator. |
| ALL_ | All users | Objects to which user has privileges | Includes objects owned by user. These views obey the current set of enabled roles. |
| USER_ | All users | Objects owned by user | Views with the prefix USER_ usually exclude the column OWNER. This column is implied in the USER_ views to be the user issuing the query. |

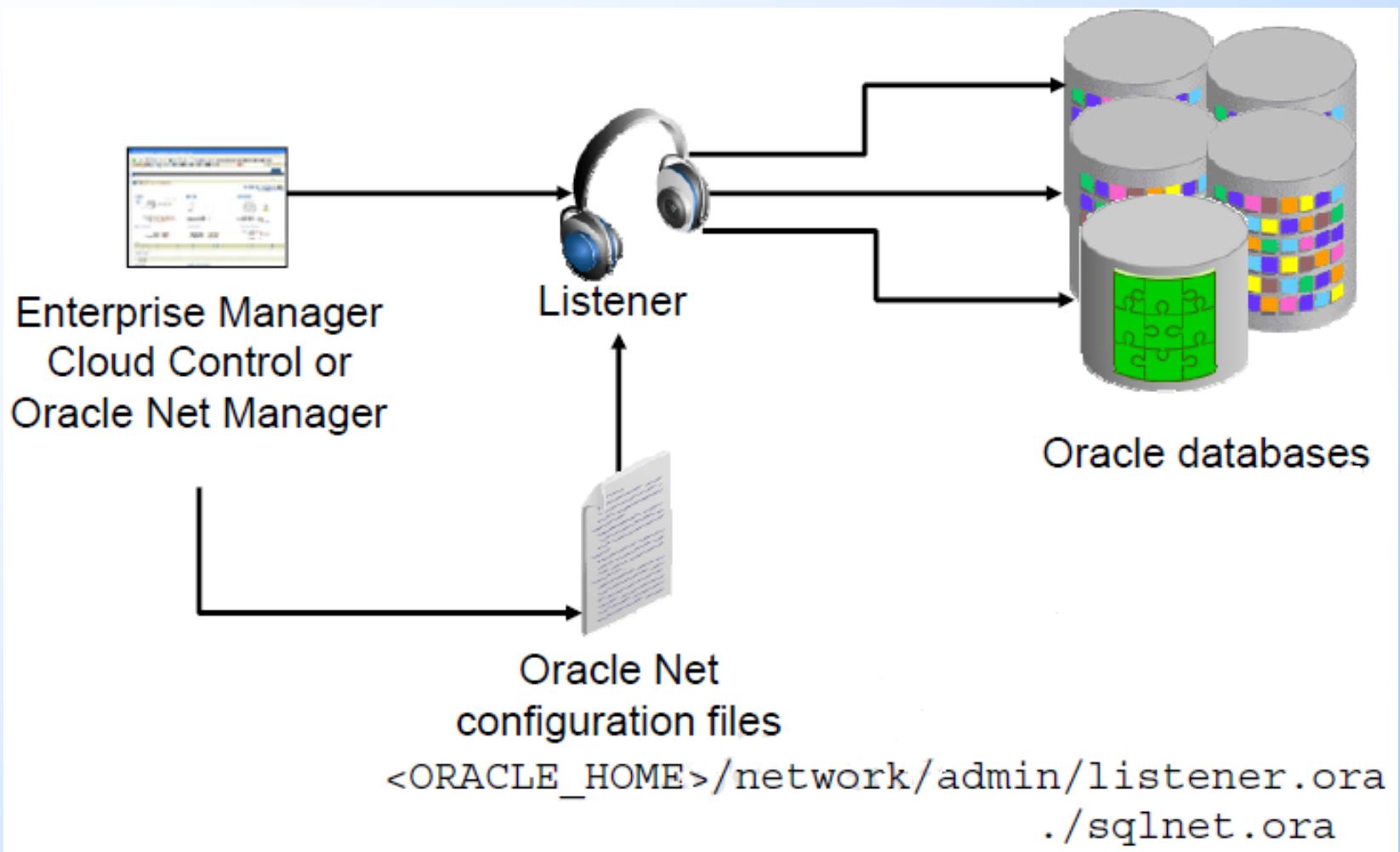
Exemple

- `SELECT table_name, tablespace_name FROM user_tables;`
- `SELECT sequence_name, min_value, max_value, increment_by FROM all_sequences WHERE sequence_owner IN ('MDSYS', 'XDB');`
- `SELECT user_name, account_status FROM dba_users WHERE account_status = 'OPEN';`
- `DESCRIBE dba_indexes;`

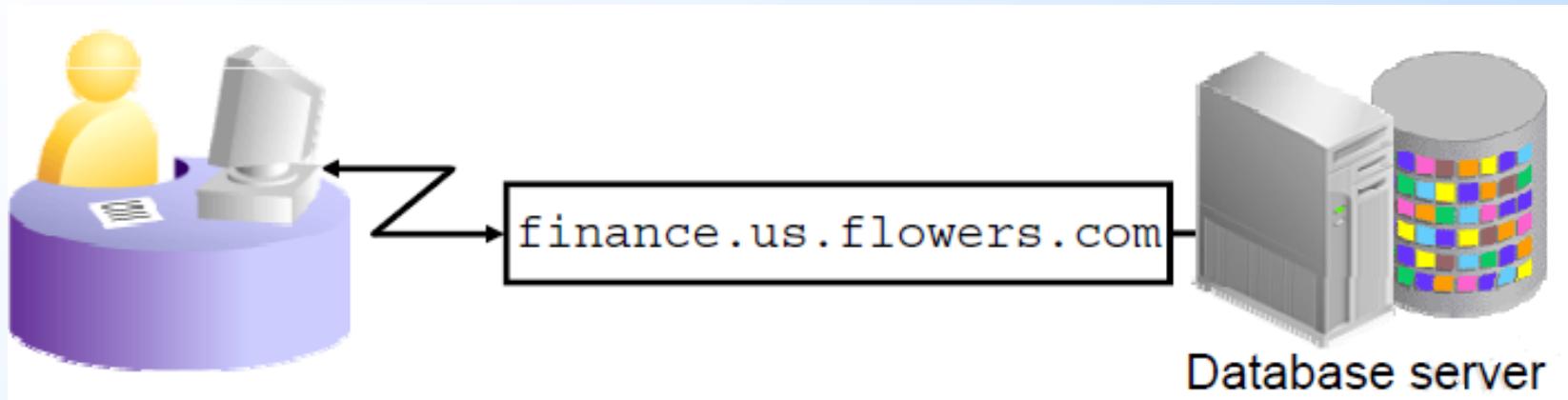
Oracle Net Services



Oracle Net Listener



Coneectarea la un server Oracle



```
(DESCRIPTION=
  (ADDRESS= (PROTOCOL=tcp) (HOST=flowers-server) (PORT=1521))
  (CONNECT_DATA=
    (SERVICE_NAME=finance.us.flowers.com) ))
```

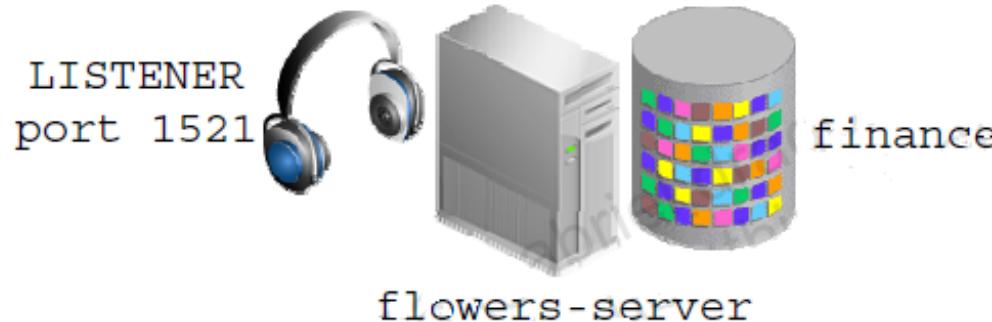
Name resolution



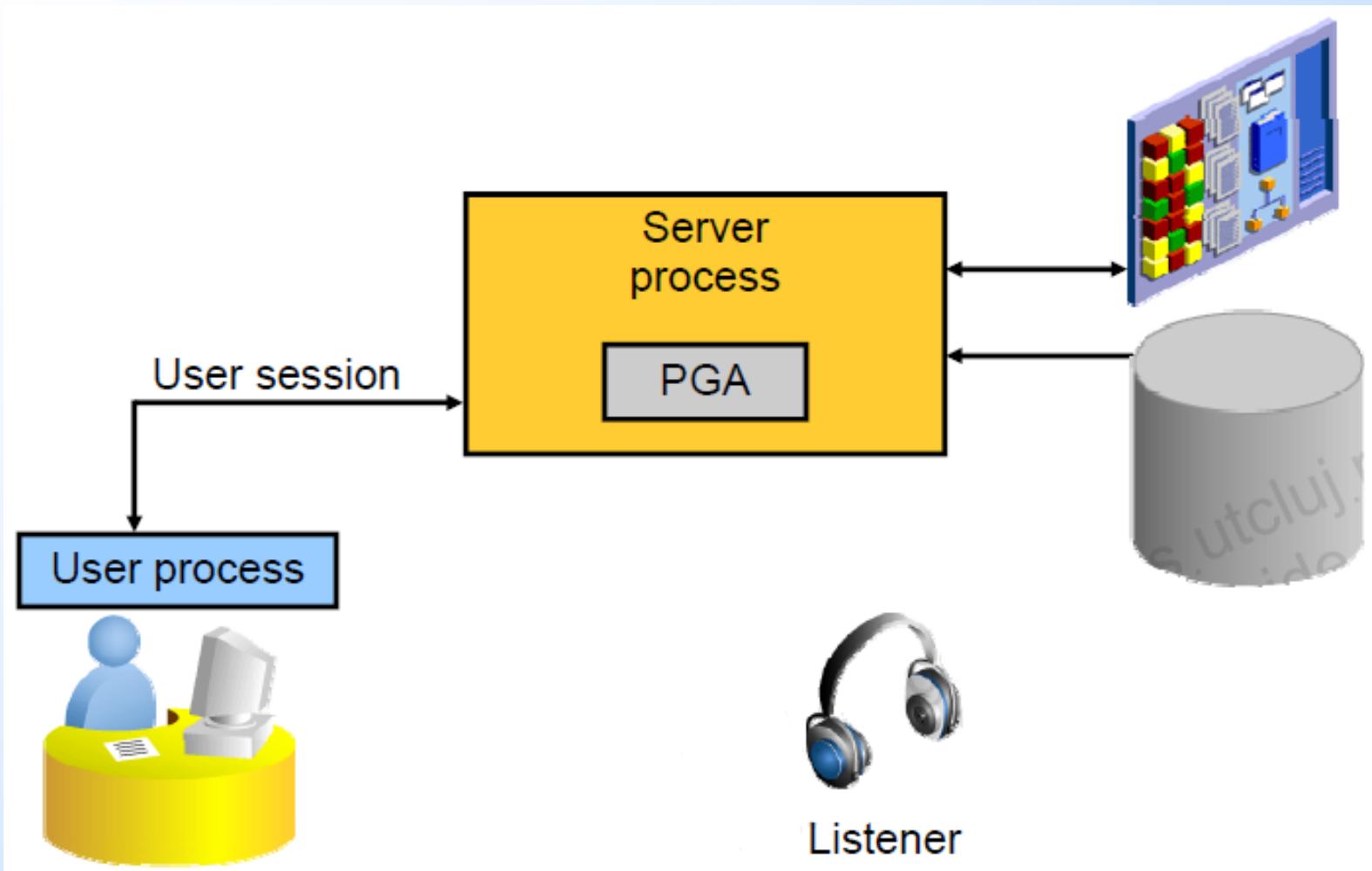
```
CONNECT jsmith/jspass@finflowers
```

Name resolution

```
finflowers = (DESCRIPTION=
  (ADDRESS= (PROTOCOL=tcp) (HOST=flowers-server) (PORT=1521) )
  (CONNECT_DATA=
    (SERVICE_NAME=finance.us.flowers.com) ))
```



Sesiuni utilizator



Metode Naming

- Easy connect:
 - se folosește un sir de conectare TCP/IP: connect nume/parola@host:port_listener/net_service_name
- Local naming:
 - trebuie să existe un fisier de configurare local, pe mașina client (tnsnames.ora): connect hr/hr@orcl
- Directory naming:
 - se folosește un directory server de tip LDAP (Lightweight Directory Access Protocol)
- External naming:
 - se folosește un sir de conectare non-Oracle, dar suportat de Oracle

Utilizatori BD

□ Cont utilizator

- Nume unic – până la 30B, începe cu literă, fără caractere speciale
- Metodă de identificare – cu parolă, globală sau externă (biometrică, cu certificat sau cu token)
- Tablespace implicit – trebuie să se acorde separat privilegii asupra obiectelor stocate în tablespace și „quota” asupra tablespace
- Tablespace temporar – aici se vor efectua sortările, nu necesită „quota”
- Profil utilizator – restricții asupra resurselor

Utilizatori BD

□ Cont utilizator (continuare)

□ Grup inițial de consumatori (de resurse, Resource Manager controlează)

□ Stare cont

- Open
- Locked
- Expired

□ Schemă

□ Colecție de obiecte BD în proprietatea unui utilizator (tabele, vederi, sevențe, proceduri stocate, indecsi)

□ Are același nume cu numele de utilizator

Utilizatori predefiniți, cu rol de administrare

- SYS („as sysdba” sau „as sysasm”)
 - Este proprietar al dicționarului datelor și Automatic Workload Repository (AWR)
 - Este folosit pentru startup/shutdown instanță BD
- SYSTEM
 - Este proprietar la tabele și vederi pentru administrare, suplimentare

Utilizatori predefiniți, cu rol de administrare

□ SYSBACKUP

- Facilitează operații de backup/recovery cu Oracle Recovery Manager (RMAN)

□ SYSDG

- Facilitează operații Oracle Data Guard

□ SYSKM

- Facilitează operații wallet Transparent Data Encryption

Recomandări

□ Least privilege

- Utilizatorii cu rol de administrator să nu folosească SYS sau SYSTEM în operații de rutină
- Se acordă privilegii potrivite altor nume de utilizatori cu scop de administrare

Introducere în SQL

Definirea Schemei
Clauzele Select-From-Where
Sortare
Interogări imbricate

SQL

- Este un limbaj “very-high-level”.
 - Spune CE trebuie făcut și nu CUM trebuie făcut.
 - Evită detaliile legate de lucrul cu datele, caracteristice limbajelor procedurale (C++ sau Java).
- SGBD-ul se îngrijește de modul de execuție a interogării.
 - Această funcțiune se numește “query optimization.”

O Relație este o Tabelă

Atribute
(cap de tabel)

Tuple
(rânduri)

| name | manf |
|-------------|----------------|
| Winterbrew | Pete's |
| Bud Lite | Anheuser-Busch |

Numele
Relației

Beers

Scheme

- *Schema Relației* = numele relației și lista de attribute.
- Optional: tipul atributelor.
- Exemplu: Beers(name, manf) sau Beers(name: string, manf: string)
- *BD* = colecție de relații.
- *Schema BD* = setul tuturor schemelor relațiilor din baza de date (BD).

Exemplu de BD

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Ceea ce este subliniat = “*key*” (tuplele nu pot avea aceeași valoare în toate atributele ce compun cheia relației).
 - Este un exemplu de constrângere.

Definirea schemei BD în SQL

- SQL este standardul de facto pentru modelul relațional.
- Este în primul rând un limbaj de interogare, pentru regăsirea informației din BD (LMD).
- Include de asemenea o componentă “*data-definition*” pentru descrierea schemei BD (LDD).

Crearea (Declararea) unei Relații

- Forma cea mai simplă:

```
CREATE TABLE <nume> (  
    <listă de elemente>  
);
```

- Pentru a elimina o relație:

```
DROP TABLE <nume>;
```

Elementele declarației “CREATE Table”

- Cel mai important element: un atribut (coloană) și tipul său.
- Tipurile obișnuite sunt:
 - INT sau INTEGER (sinonime).
 - REAL sau FLOAT (sinonime).
 - CHAR(n) = sir de caractere de lungime fixă, n caractere.
 - VARCHAR(n) = sir de caractere de lungime variabilă, maxim n caractere.

Exemplu: Create Table

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     VARCHAR(20),  
    price    REAL  
) ;
```

Valori SQL

- Intregii și realii sunt reprezentați așa cum sunt.
- Sirurile de caractere se includ între două caractere apostrof.
 - Apostrof dublat = apostrof real, de ex., 'Joe''s Bar'.
- Orice valoare poate fi NULL.

Date și Time

- DATE și TIME sunt tipuri în SQL.
- Forma unei valori “date” este:

DATE 'yyyy-mm-dd'

- **Exemplu:** DATE '2013-09-29' pentru 29 Septembrie 2013.

Valori Time

- Forma pentru valoarea “time” este:

TIME 'hh:mm:ss'

cu un punct zecimal optional și fractii de secundă în continuare.

□ **Exemplu:** TIME '15:30:02.5' = două secunde și jumătate după 3:30PM.

Declararea Cheilor

- Un atribut sau listă de attribute poate fi declarat PRIMARY KEY sau UNIQUE.
- Oricare din aceste declarații spune că nu pot exista două tuple ale relației care să corespundă întru-totul față de lista de attribute.
- Există diferențe între cele două declarații care vor fi discutate ulterior în acest curs.

Declararea Cheii formate dintr-un singur atribut

- Se completează PRIMARY KEY sau UNIQUE după declarația de tip a atributului.
- Exemplu:

```
CREATE TABLE Beers (
    name      CHAR(20) UNIQUE,
    manf      CHAR(20)
) ;
```

Declararea Cheii Compuse

- O declarație de cheie poate fi făcută ca element separat în lista de elemente a instrucțiunii CREATE TABLE.
- Această formulare este esențială atunci când cheia este compusă din mai multe atribută.
- Poate fi utilizată și la declararea cheii formate dintr-un singur atribut.

Exemplu: Cheie Compusă

- *bar* și *beer* formează împreună cheia pentru *Sells*:

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     VARCHAR(20),
    price    REAL,
    PRIMARY KEY (bar, beer)
) ;
```

PRIMARY KEY vs. UNIQUE

1. Într-o relație poate exista doar o singură cheie declarată PRIMARY KEY, și mai multe attribute UNIQUE.
2. Nici unul din attributele ce compun PRIMARY KEY nu pot fi NULL în nici o tuplă. Spre deosebire, attributele declarate UNIQUE pot avea valori NULL, și pot exista mai multe tuple cu valoarea NULL.

Clauzele Select-From-Where

SELECT listă de atrbute

FROM una sau mai multe tabele

WHERE condiție aplicată tuplelor
tabelelor

Schema BD Exemplu

- Interogările SQL prezentate în curs folosesc următoarea schemă:

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Atributele subliniate indică CHEIA fiecărei relații.

Exemplu

- Folosind **Beers(name, manf)**, care sunt mărcile de bere produse de Anheuser-Busch?

```
SELECT name  
FROM Beers  
WHERE manf = 'Anheuser-Busch';
```

Rezultatul Interrogării

| name |
|----------|
| Bud |
| Bud Lite |
| Michelob |
| ... |

Răspunsul este o relație cu un singur atribut, "name", și tuplele cu denumirea fiecărei mărci de bere produsă de Anheuser-Busch, cum este "Bud".

Cum funcționează interogarea asupra unei singure relații

- Relația este specificată în clauza FROM.
- Se aplică *selecția* precizată în clauza WHERE.
- Se aplică *proiecția extinsă* indicată prin lista de attribute din clauza SELECT.

Semanticile Operațiilor

| name | manf |
|------|----------------|
| | |
| Bud | Anheuser-Busch |
| | |

Variabila de tuplă t
parcurge toate
tuplele

Se include $t.name$
în rezultat, dacă e
adevărat

Verifică dacă
 $manf$ are
valoarea
'Anheuser-Busch'

Semanticile Operațiilor

- Se poate gândi la o *variabilă de tuplă* ce vizitează fiecare tuplă a relației menționate în clauza FROM.
- Se verifică dacă tupla “curentă” satisfacă clauza WHERE.
- Dacă se întâmplă aşa, se determină atributele sau expresiile clauzei SELECT folosind componentele acestei tuple.

* În clauza SELECT

- Atunci când există o singură relație în clauza FROM, * în clauza SELECT semnifică “toate atributele acestei relații”.
- **Exemplu:** Se folosește Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Rezultatul Interrogării:

| name | manf |
|----------|----------------|
| Bud | Anheuser-Busch |
| Bud Lite | Anheuser-Busch |
| Michelob | Anheuser-Busch |
| ... | ... |

Acum, rezultatul are fiecare atribut al relației Beers.

Redenumirea Atributelor

- Dacă se dorește ca rezultatul să aibă nume diferite de attribute, se folosește "AS <nume nou>" pentru a redenumi un atribut.
- **Exemplu:** Se folosește **Beers(name, manf):**

```
SELECT name AS beer, manf  
FROM Beers  
WHERE manf = 'Anheuser-Busch'
```

Rezultatul Interrogării:

| beer | manf |
|----------|----------------|
| Bud | Anheuser-Busch |
| Bud Lite | Anheuser-Busch |
| Michelob | Anheuser-Busch |
| ... | ... |

Expresii în clauza SELECT

□ Orice expresie ce are sens poate fi un element al clauzei SELECT.

□ **Exemplu:** Se folosește **Sells(bar, beer, price)**:

```
SELECT bar, beer,  
      price*114 AS priceInYen  
FROM Sells;
```

Rezultatul Interrogării

| bar | beer | priceInYen |
|-------|--------|------------|
| Joe's | Bud | 285 |
| Sue's | Miller | 342 |
| ... | ... | ... |

Exemplu: Constante ca Expresii

- Se folosește Likes(drinker, beer):

```
SELECT drinker,  
       'îi place Bud' AS cuiîiplaceBud  
FROM Likes  
WHERE beer = 'Bud';
```

Rezultatul Interrogării

| drinker | cui îi place Bud |
|---------|------------------|
| Sally | îi place Bud |
| Fred | îi place Bud |
| ... | ... |

Exemplu: Integrarea Informației

- Adesea se construiesc “data warehouses” din surse multiple de date.
- Să presupunem că fiecare bar are propria relație **Menu(beer, price)** .
- Pentru a contribui la **Sells(bar, beer, price)** este nevoie de interogarea fiecărui bar și de adăugarea (“insert”) denumirii barului.

Integrarea Informației

- Să spunem, că la “Joe’s Bar” emitem interogarea (query):

```
SELECT 'Joe''s Bar', beer, price  
FROM Menu;
```

Condiții Complexe În Clauza WHERE

- Operatori logici AND, OR, NOT.
- Comparații =, <>, <, >, <=, >=.
- și mulți alți operatori ce produc rezultate valoare-logică.

Exemplu: Condiție Complexă

- Se folosește **Sells(bar, beer, price)**, trebuie găsit prețul de la “Joe’s Bar” pentru “Bud”:

```
SELECT price  
FROM Sells  
WHERE bar = 'Joe''s Bar' AND  
      beer = 'Bud';
```

Formate tipice (pattern)

- O condiție poate compara un sir de caractere cu un format tipic în felul următor:
 - <Atribut> LIKE <pattern> sau <Atribut> NOT LIKE <pattern>
- *Pattern* este un sir de caractere încadrat între ghilimele cu:
 - % = “orice sir de caractere”;
 - _ = “orice caracter”.

Exemplu: LIKE

- Se folosește **Drinkers(name, addr, phone)**
Să se găsească “drinkers” cu prefixul
numărului de telefon 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_____';
```

Valori NULL

- Tuplele în relații SQL pot avea valoarea NULL pentru unul sau mai multe attribute.
- Semnificația depinde de context. Două situații obișnuite:
 - *Valoare lipsă* : de exemplu, se știe că “Joe’s Bar” are o adresă, dar aceasta nu se cunoaște.
 - *Inaplicabil* : de exemplu, valoarea atributului *sot* pentru o persoană necăsătorită.

Compararea NULL cu *Valoare*

- Condițiile au de fapt o logică 3-valori în SQL: TRUE, FALSE, UNKNOWN.
- Compararea oricărei valori (inclusiv NULL cu el însuși) cu NULL conduce la UNKNOWN.
- Pentru ca o tuplă să aparțină răspunsului la o interogare, condiția din clauza WHERE trebuie să aibă valoarea logică TRUE (nici FALSE și nici UNKNOWN).

Logica Trei-Valori

- Pentru a înțelege cum lucrează AND, OR, și NOT în logica trei-valori, să ne gândim că TRUE = 1, FALSE = 0, și UNKNOWN = $\frac{1}{2}$.
- AND = MIN; OR = MAX, $\text{NOT}(x) = 1-x$.
- Exemplu:

TRUE AND (FALSE OR NOT(UNKNOWN)) =
 $\text{MIN}(1, \text{MAX}(0, (1 - \frac{1}{2}))) =$
 $\text{MIN}(1, \text{MAX}(0, \frac{1}{2})) = \text{MIN}(1, \frac{1}{2}) = \frac{1}{2}$.

Exemplu Surprizător

- Pentru următoarea relație Sells :

| bar | beer | price |
|-----------|------|-------|
| Joe's Bar | Bud | NULL |

SELECT bar

FROM Sells

WHERE price < 2.00 OR price >= 2.00;

↔
UNKNOWN

↔
UNKNOWN

↔
UNKNOWN

Motivație: Legile 2-Valori!= Legile 3-Valori

- Anumite legi obișnuite, cum este comutativitatea lui AND, rămân valabile în logica 3-valori.
- Dar altele nu, cum este *legea excluderii mijlocului* : $p \text{ OR NOT } p = \text{TRUE}$.
 - Când $p = \text{UNKNOWN}$, membrul stâng este $\text{MAX}(\frac{1}{2}, (1 - \frac{1}{2})) = \frac{1}{2} \neq 1$.

Sortare

□ ***ORDER BY nume_atribut [ASC/DESC],...***

□ specifică ordinea tuplelor în relația rezultat

□ ordinea poate fi:

- crescătoare **ASC** ("ASCending") sau
- descrescătoare **DESC** ("DESCending")

Exemplu: ORDER BY

| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 2 |

R

| A | B |
|---|---|
| 1 | 2 |
| 5 | 2 |
| 3 | 4 |

```
SELECT *
FROM R
ORDER BY B
```

Interogări imbricate (Subqueries)

- O instrucțiune SELECT-FROM-WHERE amplasată între paranteze (*subquery*) poate fi folosită ca valoare în mai multe locuri, inclusiv în clauzele FROM și WHERE.
- **Exemplu:** În locul unei relații în clauza FROM, se poate folosi o interogare imbricată (subquery).
 - Este obligatoriu să fie utilizată o variabilă de tuplă pentru a numi tuplele rezultatului subinterrogării.

Exemplu: Subquery în FROM

- Să se găsească berea preferată de cel puțin o persoană ce frecventează “Joe’s Bar”.

```
SELECT beer
```

```
FROM Likes,
```

```
(SELECT drinker
```

```
FROM Frequent
```

```
WHERE bar = 'Joe''s Bar' ) JD
```

```
WHERE Likes.drinker = JD.drinker;
```

Persoane ce
frecventează “Joe’s Bar”



Interogări imbricate ce Returnează 1 Tuplă

- Dacă o interogare imbricată returnează 1 tuplă și numai una, atunci interogarea imbricată poate fi utilizată ca valoare.
 - De obicei tupla are o singură componentă.
 - Dacă interogarea imbricată nu returnează nici o tuplă sau mai mult de una, se semnalează o eroare run-time.

Exemplu: Interrogare imbricată cu 1 tuplă

- Se folosește **Sells(bar, beer, price)**, pentru a găsi barurile ce servesc “Miller” la prețul cu care “Joe’s Bar” vinde “Bud”.
- Răspunsul poate fi dat sub forma a două interogări:
 1. Găsește prețul cu care “Joe’s Bar” vinde “Bud”.
 2. Găsește barurile ce servesc “Miller” la acel preț.

Soluția cu interogare și interogare imbricată

```
SELECT bar
```

```
FROM Sells
```

```
WHERE beer = 'Miller' AND
```

```
    price = (SELECT price
```

```
        FROM Sells
```

```
        WHERE bar = 'Joe''s Bar'  
        AND beer = 'Bud');
```

Prețul cu
care "Joe's Bar"
vinde "Bud"

Operatorul IN

- <tuplă> IN (<subquery>) are valoarea logică **True** dacă și numai dacă tupla este membră a relației produse de interogarea imbricată.
 - La modul opus este: <tuplă> NOT IN (<subquery>).
- Expresiile “IN” pot apărea în clauze WHERE.

Exemplu: IN

- Se folosesc `Beers(name, manf)` și `Likes(drinker, beer)`, pentru a găsi numele și producătorul fiecărei beri preferată de Fred.

```
SELECT *  
FROM Beers
```

```
WHERE name IN
```

Mulțimea (set)
berilor preferate
de Fred

```
(SELECT beer  
FROM Likes  
WHERE drinker = 'Fred');
```

Operatorul EXISTS

- EXISTS(<subquery>) are valoarea logică **True** dacă și numai dacă rezultatul interogării imbricate este nevid.
- **Exemplu:** Se folosește **Beers(name, manf)** , pentru a găsi acele beri ce reprezintă unica bere a producătorului ei.

Exemplu: EXISTS

```
SELECT name  
FROM Beers b1  
  
WHERE NOT EXISTS (
```

```
SELECT *  
FROM Beers  
WHERE manf = b1.manf AND  
      name <> b1.name);
```

Multimea
berilor
ce sunt
produse
de același
producător
(cu b1), dar
au altă
denumire

De notat regula: manf se referă
la o relație din cel mai apropiat
FROM, ce conține acel atribut.

De notat
operatorul
SQL “not
equals”

Operatorul “ANY”

- $x = \text{ANY}(<\text{subquery}>)$ este o condiție booleană ce are valoarea logică **True** dacă x este “egal” cu cel puțin una din tuplele rezultatului interogării imbricate.
 - “=” poate fi oricare din operatorii de comparație.
- **Exemplu:** $x >= \text{ANY}(<\text{subquery}>)$ semnifică x nu este singura tuplă cea mai mică produsă de interogarea imbricată.
- De notat că tuplele trebuie să aibă doar o componentă.

Operatorul “ALL”

- $x <> \text{ALL}(<\text{subquery}>)$ are valoarea logică **True** dacă pentru fiecare tuplă t din relația “subquery”, x este diferit de t .
 - Cu alte cuvinte, x nu se regăsește în rezultatul interogării imbricate.
- “ $<>$ ” poate fi orice operator de comparație.
- **Exemplu:** $x \geq \text{ALL}(<\text{subquery}>)$ semnifică nu există tuplă mai mare ca x în rezultatul interogării imbricate.

Exemplu: ALL

- Se folosește **Sells(bar, beer, price)**, pentru a găsi berea(-ile) vândute la cel mai mare preț.

```
SELECT beer
```

```
FROM Sells
```

```
WHERE price >= ALL(
```

```
    SELECT price  
    FROM Sells);
```

prețul "Sells" din exterior să nu fie mai mic decât nici un preț.

SQL Partea a doua

Interogări cu mai multe Relații
Grupare/Agregare

Operatori „set” (UNION/INTERSECT/EXCEPT)
Adăugare/Modificare/Ștergere

Interogări asupra mai multor Relații

- Interogările combină adesea datele din mai multe relații.
- Se pot menționa mai multe relații într-o interogare prin introducerea lor în clauza FROM.
- Se face distincție între atributele cu același nume prin prefixare cu numele relației: “<relație>.<atribut>” .

Exemplu: Reunirea a două Relații

- Se folosesc relațiile Likes(drinker, beer) și Frequent(drinker, bar), pentru a găsi băuturile preferate de cel puțin o persoană ce frecventează "Joe's Bar".

```
SELECT beer
FROM Likes, Frequent
WHERE bar = 'Joe''s Bar' AND
      Frequent.drinker =
          Likes.drinker;
```

Semanticile Formale pentru o Interogare cu mai multe relații

- Sunt aproximativ aceleași ca și în cazul interogării cu o singură relație:
 1. Se începe cu produsul relațiilor din clauza FROM.
 2. Se aplică selecția impusă de condiția din clauza WHERE.
 3. Se aplică proiecția extinsă pe lista de atribută și expresii din clauza SELECT.

Semanticile Operaționale pentru o Interogare cu mai multe Relații

- Să ne imaginăm câte o variabilă de tuplă pentru fiecare relație din clauza FROM.
 - Aceste variabile de tuplă parcurg fiecare combinație de tuple, câte o tuplă din fiecare relație.
- Dacă variabilele de tuplă indică tuple ce satisfac condiția din clauza WHERE, aceste tuple fac parte din rezultat.

Exemplu

| drinker | bar |
|---------|-------|
| | |
| Sally | Joe's |
| | |

vt1

Frequents

| drinker | beer |
|---------|------|
| | |
| Sally | Bud |
| | |

vt2

Likes

verifică
pentru
Joe's
dacă aceste
valori sunt
egale

coloana "beer"
face parte
din rezultat

Variabile de Tuplă Explicite

- Uneori, o interogare necesită folosirea a două copii a aceleiași relații.
- Se face distincție între cele două copii prin completarea după numele relației a numelui unei variabile de tuplă, în clauza FROM.
- Oricând se poate proceda astfel, prin redenumirea relațiilor, chiar dacă nu este o condiție obligatorie.

Revenim la întrebarea din cursul introductiv:

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

prin ce diferă de:

```
SELECT a  
FROM R  
WHERE b IN (SELECT b FROM S);
```

Cu ajutorul IN se construiește un predicat pentru tuplele din R

```
SELECT a  
FROM R  
WHERE b IN
```

Valoarea 2 apare de două ori

```
(SELECT b FROM S);
```

O singură
parcursere a
tuplelor din R

| a | b |
|---|---|
| 1 | 2 |
| 3 | 4 |

R

| b | c |
|---|---|
| 2 | 5 |
| 2 | 6 |

S

(1,2) satisfac
condiția;
1 este afișat
o singură
dată.

Tuplele din R și S fac pereche prin această interogare

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

Ciclu dublu:
primul peste
tuplele din R și
al doilea peste
tuplele din S

| a | b | c |
|---|---|---|
| 1 | 2 | 2 |
| 3 | 4 | 5 |

R S

(1,2) cu (2,5) și
(1,2) cu (2,6),
ambele satisfac
condiția;
1 este afișat de
două ori.

Exemplu: Self-Join

- Folosind **Beers(name, manf)**, să se găsească toate perechile de bere produse de aceeași fabrică.
 - Nu interesează perechi cum ar fi (Bud, Bud).
 - Perechile vor fi obținute în ordine alfabetică, de ex. (Bud, Miller), nu (Miller, Bud).

```
SELECT b1.name, b2.name  
FROM Beers b1, Beers b2  
WHERE b1.manf = b2.manf AND  
      b1.name < b2.name;
```

Expresii “Join”

- SQL mai multe variante de reuniri (“bag”).
- Aceste expresii pot fi interogări de sine stătătoare (stand-alone queries) sau sunt folosite în locul relațiilor în clauza FROM.

Produs și Natural Join

□ Natural join:

R NATURAL JOIN S;

□ Produs:

R CROSS JOIN S;

□ Exemplu:

Likes NATURAL JOIN Sells;

□ Relațiile pot fi la fel de bine interogări imbricate
între paranteze.

Theta Join

- R JOIN S ON <condiție>
- Exemplu: Drinkers(name, addr) și Frequent(Frequent(drinker, bar)):

```
Drinkers JOIN Frequent ON  
name = drinker;
```

generează toate cvaduplele (d, a, d, b)
a.î. persoana d locuiește la adresa a și
frecventează barul b .

Outerjoin

- R OUTER JOIN S este elementul central al unei expresii outerjoin. Expresia este modificată de:
 1. Opțional NATURAL înaintea lui OUTER.
 2. Opțional ON <condiție> după JOIN.
 3. Opțional LEFT, RIGHT, sau FULL înainte de OUTER.
 - LEFT = se completează tuplele din R care nu au echivalent în S cu atâtea NULL-uri câte atrbute are S.
 - RIGHT = se completează tuplele din S care nu au echivalent în R cu atâtea NULL-uri câte atrbute are R.
 - FULL = se completează tuplele din R care nu au echivalent în S cu atâtea NULL-uri câte atrbute are S și se completează tuplele din S care nu au echivalent în R cu atâtea NULL-uri câte atrbute are R.
 - Este varianta implicită.

Doar una
din acestea
două

Exemplu: Outerjoin

$$R = (A \mid B)$$

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |

$$S = (B \mid C)$$

| B | C |
|---|---|
| 2 | 3 |
| 6 | 7 |

(1,2) se cuplă cu (2,3), dar celelalte două tuple sunt singulare (nu au pereche).

R OUTERJOIN S =

| A | B | C |
|------|---|------|
| 1 | 2 | 3 |
| 4 | 5 | NULL |
| NULL | 6 | 7 |

Agregare

- Funcțiile SUM, AVG, COUNT, MIN și MAX pot fi aplicate unei coloane în clauza SELECT pentru a obține un anumit nivel de agregare al coloanei.
- De asemenea există COUNT(*) ce numără tuplele.

Exemplu: Agregare

- Se folosește **Sells(bar, beer, price)** pentru a găsi prețul mediu de vânzare pentru berea “Bud”:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud' ;
```

Eliminarea Duplicatelor într-o Agregare

- Se folosește DISTINCT pentru o agregare.
- **Exemplu:** să se găsească numărul prețurilor *diferite* de vânzare pentru berea “Bud”:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud' ;
```

Valorile NULL sunt Ignorate într-o Agregare

- NULL nu contribuie la Sum, Avg sau Count și nu poate fi nici valoarea minimă (Min) nici valoarea maximă (Max) a unei coloane.
- Dar dacă nu există valori non-NULL pentru o coloană, atunci rezultatul agregării este NULL.
- **Excepție:** COUNT pentru o mulțime vidă este 0.

Exemplu: Efectul NULL

```
SELECT count(*)  
FROM Sells  
WHERE beer = 'Bud';
```

Numărul barurilor ce vând berea "Bud".

```
SELECT count(price)  
FROM Sells  
WHERE beer = 'Bud';
```

Numărul barurilor ce vând berea "Bud" la un preț cunoscut.

Gruparea valorilor de date

- Într-o instrucțiune SELECT-FROM-WHERE se poate folosi clauza GROUP BY cu o listă de attribute (la sfârșit).
- Tuplele din relația rezultat pentru blocul SELECT-FROM-WHERE sunt grupate conform valorilor atributelor prezente în clauza GROUP BY și se poate aplica orice agregare pe fiecare grup în parte (și numai pe grupurile formate).

Exemplu: Gruparea valorilor de date

- Se folosește **Sells(bar, beer, price)** pentru a găsi fiecare bere vândută și prețul mediu de vânzare:

```
SELECT beer, AVG(price)  
FROM Sells  
GROUP BY beer;
```

| beer | AVG(price) |
|------|------------|
| Bud | 2.33 |
| ... | ... |

Exemplu: Gruparea valorilor de date

- Se folosește **Sells(bar, beer, price)** și **Frequents(drinker, bar)** pentru a găsi prețul mediu dat pe berea “Bud” de fiecare persoană ce frecventează baruri:

```
SELECT drinker, AVG(price)  
FROM Frequents, Sells  
WHERE beer = 'Bud' AND  
      Frequents.bar = Sells.bar  
GROUP BY drinker;
```

Pasul doi:
grupează
după
drinker

Primul pas:
compune
triplete
 \langle drinker-bar-
price \rangle
pentru “Bud”

Restricții pentru Listele SELECT în contextul Agregării

- Dacă se folosește agregarea, atunci fiecare element din lista clauzei SELECT trebuie să fie:
 1. O Valoare Agregată, sau
 2. Un atribut al listei clauzei GROUP BY.

Exemplu de Interogare Ilegală

- Să se găsească barul ce vinde berea "Bud" la prețul cel mai ieftin:

```
SELECT bar, MIN(price)  
FROM Sells  
WHERE beer = 'Bud';
```

- Această interogare este ilegală în SQL.

Clauza HAVING

- HAVING <condiție> poate fi utilizată după clauza GROUP BY.
- Dacă apare această cluză, condiția se aplică fiecărui grup și grupurile ce nu respectă condiția sunt eliminate din rezultat.

Exemplu: HAVING

- Se folosește **Sells(bar, beer, price)** și **Beers(name, manf)** pentru a găsi prețul mediu al acelor beri ce fie sunt servite în cel puțin trei baruri, fie sunt fabricate de “Pete’s”.

Soluția

```
SELECT beer, AVG(price)  
FROM Sells  
GROUP BY beer
```

```
HAVING COUNT(bar) >= 3 OR
```

```
beer IN (SELECT name  
FROM Beers  
WHERE manf = 'Pete"s');
```

Grupurile de bere cu cel puțin 3 baruri non-NULL și de asemenea grupurile de bere au producătorul "Pete's".

Berile produse de "Pete's".

Cerințe pentru Condițiile HAVING

- Într-o interogare imbricată ("subquery") se poate folosi orice.
- În exteriorul interogărilor imbricate, condițiile pot face referire la atribut doar în următoarele circumstanțe:
 1. Un atribut proprietate de grup, sau
 2. O valoare agregată
(aceeași condiție ca și pentru clauza SELECT ce conține agregare).

Operatori „set”

Uniunea, Intersecția și Diferența

- Uniunea, intersecția și diferența relațiilor sunt exprimate prin formele următoare, fiecare implică interogări imbricate:
 - () UNION ()
 - () INTERSECT ()
 - () EXCEPT ()

Exemplu: INTERSECT

- Se folosesc Likes(drinker, beer), Sells(bar, beer, price) și Frequents(drinker, bar) pentru a găsi persoanele și mărcile de bere astfel încât:
 1. Persoana preferă berea și
 2. Persoana frecventează cel puțin un bar unde se vinde berea.

De notat trucul:
interrogarea
imbricată este
o tabelă de bază.

Soluția

```
(SELECT * FROM Likes)
```

INTERSECT

```
(SELECT drinker, beer
FROM Sells, Frequents
WHERE Frequents.bar = Sells.bar
);
```

Persoana frecventează
un bar ce vinde berea.

Semanticile “Bag”

- Deși instrucțiunea SELECT-FROM-WHERE folosește semantici “bag”, semanticile implicite pentru uniune, intersecție și diferență sunt “set”.
- Aceasta înseamnă că duplicatele sunt eliminate implicit la aplicarea unuia din cei trei operatori (UNION, INTERSECT, EXCEPT).

Motivație: Eficiență

- Atunci când se aplică proiecția extinsă, este mai ușor să se evite eliminarea duplicatelor.
 - Deoarece se poate lucra cu o singură tuplă la un moment dat.
- Pentru intersecție sau diferență, este mai eficient să se sorteze mai întâi relațiile.
 - La acel moment dat se poate la fel de bine să se eliminate duplicatele.

Eliminarea explicită a Duplicatelor

- Pentru a forța rezultatul unei interogări să fie “set”:

SELECT DISTINCT . . .

- Pentru a forța rezultatul unei interogări să fie “bag” (adică să nu fie eliminate duplicatele):

se folosește ALL,

de exemplu . . . UNION ALL . . .

Exemplu: DISTINCT

- Se folosește **Sells(bar, beer, price)**, pentru a găsi toate prețurile diferite pentru mărcile de bere vândute:

```
SELECT DISTINCT price  
FROM Sells;
```

- De notat că fără DISTINCT, fiecare preț poate să apară de mai multe ori, multiplicat după numărul de baruri/mărci de bere.

Exemplu: ALL

Interogarea:

```
(SELECT drinker FROM Frequent)  
      EXCEPT ALL
```

```
(SELECT drinker FROM Likes);
```

folosește relațiile **Frequent(drinker, bar)** și **Likes(drinker, beer)** pentru a afișa persoanele pentru care numărul de baruri frecventate este mai mare decât numărul de mărci de bere preferate și o persoană apare de atâtea ori în rezultat cât reprezintă diferența acestor numere.

Actualizarea BD

- O comandă de *actualizare* nu returnează un rezultat (aşa cum face o interogare), ci modifică baza de date într-un anumit fel.
- Există trei operaţii de actualizare:
 1. *Insert* (Adaugă) una sau mai multe tuple.
 2. *Delete* (Şterge) una sau mai multe tuple.
 3. *Update* (Modifică) valoarea(-ile) uneia sau mai multor tuple existente .

Adăugare

- Pentru a adăuga o singură tuplă:

INSERT INTO <relație>

VALUES (<listă de valori>);

- Exemplu: adaugă în Likes(drinker, beer) faptul că lui “Sally” îi place “Bud”.

INSERT INTO Likes

VALUES ('Sally' , 'Bud') ;

Specificarea Atributelor în INSERT

- Se poate adăuga la numele relației o listă de attribute.
- Există două motive:
 1. Nu mai ținem minte ordinea în care am definit attributele relației.
 2. Nu cunoaștem valorile pentru toate attributele și dorim ca sistemul să completeze componentele lipsă cu valori NULL sau valori “default” (implicite).

Exemplu: Specificare Atribute

- O altă cale pentru a adăuga faptul că lui “Sally” îi place “Bud” în Likes(drinker, beer):

```
INSERT INTO Likes (beer, drinker)  
VALUES ('Bud', 'Sally');
```

Adăugare Valori “Default”

- La instrucțiunea CREATE TABLE, se poate menționa pentru un atribut o valoare DEFAULT.
- Atunci când tupla adăugată nu are specificată valoare pentru acel atribut, va fi utilizată valoarea default.

Exemplu: Valori “Default”

```
CREATE TABLE Drinkers (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50)
        DEFAULT '123 Sesame St.',
    phone CHAR(16)
) ;
```

Exemplu: Valori “Default”

```
INSERT INTO Drinkers(name)  
VALUES ('Sally');
```

Tupla rezultată :

| name | address | phone |
|-------|---------------|-------|
| Sally | 123 Sesame St | NULL |

Adăugarea mai multor Tuple

- Se poate adăuga întregul rezultat al unei interogări într-o relație, utilizând forma:

```
INSERT INTO <relație>
( <subquery> );
```

Exemplu: Adăugarea cu “Subquery”

- Se folosește `Frequents(drinker, bar)`, pentru a introduce într-o relație nouă `PotBuddies(name)` toți prietenii potențiali ai lui “Sally” (în engleză “potential buddies”), adică acele persoane ce frecventează cel puțin un bar frecventat de “Sally”.

Altă
persoană

Soluția

```
INSERT INTO PotBuddies
```

```
(SELECT d2.drinker
```

```
FROM Frequent d1, Frequent d2
```

```
WHERE d1.drinker = 'Sally' AND
```

```
d2.drinker <> 'Sally' AND
```

```
d1.bar = d2.bar
```

```
);
```

Perechi de tuple cu persoane în care prima tuplă corespunde lui "Sally", a doua tuplă corespunde la altcineva, astfel încât barurile sunt identice.

Ştergere

- Pentru a şterge tuple, ce satisfac o condiţie, dintr-o anumită relaţie:

```
DELETE FROM <relaţie>
WHERE <condiţie>;
```

Exemplu: Ștergere

- Șterge din Likes(drinker, beer) faptul că lui “Sally” îi place “Bud”:

```
DELETE FROM Likes  
WHERE drinker = 'Sally' AND  
      beer = 'Bud';
```

Exemplu: Șterge toate Tuplele

- Golește relația Likes:

```
DELETE FROM Likes;
```

- De notat lipsa clauzei WHERE.

Exemplu: Șterge anumite Tuple

- Șterge din **Beers(name, manf)** toate berile pentru care există o altă bere a aceluiași producător.

DELETE FROM Beers b

WHERE EXISTS (

```
SELECT name FROM Beers  
WHERE manf = b.manf AND  
name <> b.name);
```



Berile aceluiași producător și cu nume diferit față de numele berii reprezentate de tupla b.

Semanticile DELETE --- (1)

- Să presupunem că “Anheuser-Busch” fabrică doar “Bud” și “Bud Lite”.
- Să presupunem că operația de stergere ajunge la tupla b prima dată pentru “Bud”.
- Interogarea imbricată (subquery) conține date, deoarece există berea “Bud Lite”, ceea ce determină eliminarea berii “Bud”.
- Atunci când operația de stergere ajunge la tupla b pentru “Bud Lite”, întrebarea este dacă se sterge această tuplă de asemenea?

Semanticile DELETE --- (2)

- Răspunsul la întrebarea de mai sus: se șterge "Bud Lite".
- Motivul este acela că ștergerea acționează în două etape:
 1. Se marchează toate tuplele pentru care condiția WHERE este satisfăcută.
 2. Se elimină tuplele marcate pentru ștergere.

UPDATE

- Pentru a modifica valorile anumitor attribute din anumite tuple ale unei relații:

UPDATE <relație>

SET <listă cu atribuirile de attribute>

WHERE <condiție asupra tuplelor>;

Exemplu: UPDATE

- Modifică numărul de telefon al lui “Fred” (persoană în tabela **Drinkers**) la valoarea 555-1212:

```
UPDATE Drinkers  
SET phone = '555-1212'  
WHERE name = 'Fred';
```

Exemplu: UPDATE pentru mai multe Tuple

- Modifică prețul maxim la bere la valoarea \$4:

```
UPDATE Sells  
SET price = 4.00  
WHERE price > 4.00;
```

Constrângeri

Chei Străine

Constrângeri Locale și Globale
“Triggers”

Constrângeri și Trigere

- O *constrângere* este o relație de legătură între elemente de date, pe care SGBD-ul este obligat să le forțeze.
 - **Exemplu:** constrângeri de tip cheie.
- "*Triggers*" sunt executate atunci când apare o condiție specifică, de exemplu, adăugarea (INSERT) unei tuple.
 - Sunt mai ușor de implementat decât constrângerile complexe.

Tipuri de Constrângeri

- Chei.
- Cheie-străină, sau integritate-referențială.
- Constrângeri la nivel Valoare.
 - Constrâng valorile unui atribut particular.
- Constrângeri la nivel Tuplă.
 - Relație de legătură între componente.
- Constrângeri Declarative ("Assertions"):
orice expresie logică SQL.

Recapitulare: Chei de tip Atribut-Singular

- Se folosește PRIMARY KEY sau UNIQUE după tip în declarația atributului.
- Exemplu:

```
CREATE TABLE Beers (
    name      CHAR(20) UNIQUE,
    manf      CHAR(20)
) ;
```

Recapitulare: Cheie Multi-atribut

- În relația vânzări (Sells) barul (bar) și berea (beer) împreună formează cheia:

```
CREATE TABLE Sells (
    bar        CHAR(20),
    beer       VARCHAR(20),
    price      REAL,
    PRIMARY KEY (bar, beer)
) ;
```

Chei străine

- Valorile ce apar în attributele unei relații trebuie să apară împreună în anumite attribute din altă relație.
- Exemplu: în **Sells(bar, beer, price)**, se așteaptă ca valoarea pentru bere (beer) să apară de asemenea în **Beers.name** .

Exprimarea Cheilor Străine

- Se folosește cuvântul cheie REFERENCES, în felul următor:
 1. După un atribut (pentru chei formate dintr-un singur atribut).
 2. Un element al schemei:
FOREIGN KEY (<listă de attribute>)
REFERENCES <relație> (<attribute>)
- Atributele referențiate trebuie să fie declarate PRIMARY KEY sau UNIQUE.

Exemplu: În cadrul definiției Atributului

```
CREATE TABLE Beers (
    name      CHAR(20) PRIMARY KEY,
    manf      CHAR(20) );
```

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer     CHAR(20) REFERENCES Beers(name),
    price    REAL );
```

Exemplu: Element al Schemei

```
CREATE TABLE Beers (  
    name    CHAR(20) PRIMARY KEY,  
    manf    CHAR(20) );
```

```
CREATE TABLE Sells (  
    bar      CHAR(20) ,  
    beer     CHAR(20) ,  
    price    REAL,  
    FOREIGN KEY(beer) REFERENCES  
        Beers(name) );
```

Fortarea Constrângerilor

Cheie Străină

- Dacă există o constrângere cheie străină de la relația R la relația S , două violări sunt posibile:
 1. La o adăugare (insert) sau modificare (update) a unei tuple în R se introduc valori ce nu se regăsesc în S .
 2. După o stergere (delete) sau modificare (update) a unei tuple în S rezultă anumite tuple inconsecvente în R (nu au pereche în S).

Exemplu:

- Presupunem $R = \text{Sells}$, $S = \text{Beers}$.
- O adăugare (insert) sau modificare (update) în **Sells** ce introduce o bere (beer) inexistentă trebuie respinsă.
- O stergere (delete) sau modificare (update) în **Beers** ce elimină o valoare “beer” ce se regăsește în anumite tuple din **Sells** poate fi gestionată în una din trei modalități:

Forțarea Constrângerilor Cheie Străină

1. *Implicit*: Modificarea este respinsă.
2. "*Cascade*": Se aplică aceleasi modificări în Sells.
 - "Delete beer": se elimină tuple în Sells.
 - "Update beer": se modifică valori în Sells.
3. "*Set NULL*": se completează valoarea NULL pentru "beer".

Exemplu: “Cascade”

- Se șterge tupla “Bud” din Beers:
 - Urmarea este că se șterg toate tuplele din Sells ce au “beer = ‘Bud’”.
- Se modifică tupla “Bud” completând ‘Budweiser’ în locul lui ‘Bud’ în Beers:
 - Urmarea este că se modifică toate tuplele din Sells cu “beer = ‘Bud’” la “beer = ‘Budweiser’”.

Exemplu: “Set NULL”

- Se șterge tupla “Bud” din Beers:
 - Se modifică toate tuplele din Sells cu “beer = ‘Bud’” la “beer = NULL”.
- Se modifică tupla “Bud” completând ‘Budweiser’ în locul lui ‘Bud’ în Beers :
 - Se modifică toate tuplele din Sells cu “beer = ‘Bud’” la “beer = NULL”.

Alegerea unei Politici de aplicare modificări

- La declararea cheii străine, se poate preciza politica SET NULL sau CASCADE independent pentru “delete” față de “update”.
- Se completează declarația cheii străine cu:
ON [UPDATE, DELETE][SET NULL CASCADE]
- Pot fi folosite două astfel de clauze.
- În caz că această clauză lipsește se aplică politica “rejected”.

Exemplu: Stabilire Politică

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     CHAR(20),
    price    REAL,
    FOREIGN KEY(beer)
        REFERENCES Beers(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ;
```

Constrângeri la nivel Valoare (Atribut)

- Se adaugă CHECK(<condiție>) la declarația pentru un atribut.
- Condiția poate folosi numele atributului, dar orice altă relație sau atribut ce apar trebuie folosite într-o interogare imbricată (subquery).

Exemplu: Check la nivel atribut

```
CREATE TABLE Sells (
    bar      CHAR(20) ,
    beer     CHAR(20)      CHECK ( beer IN
                                (SELECT name FROM Beers) ) ,
    price    REAL      CHECK ( price <= 5.00 )
);
```

Când se aplică “Check” la nivel atribut?

- Constrângerile la nivel valoare de atribut (check) sunt aplicate în momentul când se efectuează o adăugare (insert) sau modificare (update).
 - **Exemplu:** CHECK (price <= 5.00) verifică orice preț nou și respinge modificarea (acelei tuple) dacă prețul este mai mare de 5 \$.
 - **Exemplu:** CHECK (beer IN (SELECT name FROM Beers)) nu se verifică atunci când o bere (beer) este ştearsă din Beers (aşa cum se verifică la constrângerea de tip cheie străină).

Constrângeri la nivel Tuplă

- CHECK (<condiție>) poate fi adăugată ca element al schemei de relație.
- Condiția poate face referire la orice atribut al relației.
 - Alte attribute sau relații necesită o interogare imbricată (subquery).
- Se verifică doar la INSERT sau UPDATE.

Exemplu: “Check” la nivel tuplă

- Doar “Joe’s Bar” poate vinde bere mai scumpă de 5 (\$) sau altă monedă):

```
CREATE TABLE Sells (
    bar        CHAR(20),
    beer       CHAR(20),
    price      REAL,
    CHECK (bar = 'Joe''s Bar' OR
           price <= 5.00)
) ;
```

Constrângeri Declarative ("Assertions")

- Sunt elemente ale schemei bazei de date, cum sunt relațiile sau vederile.
- Se definesc în felul următor:

```
CREATE ASSERTION <nume>
    CHECK (<condiție>);
```
- Condiția se poate referi la orice relație sau atribut din schema bazei de date.

Exemplu: Constrângere “Assertion”

- În **Sells(bar, beer, price)**, nici un bar nu poate vinde la un preț mediu mai mare de 5 \$.

```
CREATE ASSERTION NoRipoffBars CHECK (
    NOT EXISTS (
        SELECT bar FROM Sells
        GROUP BY bar
        HAVING 5.00 < AVG(price)
    ));

```

Baruri cu
Preț mediu
sub 5 \$

Exemplu: Constrângere “Assertion”

- În `Drinkers(name, addr, phone)` și în `Bars(name, addr, license)`, nu pot exista mai multe baruri decât persoane.

```
CREATE ASSERTION FewBar CHECK (
    (SELECT COUNT(*) FROM Bars) <=
    (SELECT COUNT(*) FROM Drinkers)
) ;
```

Când se aplică Constrângerea “Assertion”?

- În principiu, fiecare constrângere declarativă (assertion) se verifică după orice modificare a oricărei relații din baza de date.
- Un sistem intelligent ar observa doar acele modificări ce pot cauza violarea unei constrângerii declarative.
 - **Exemplu:** O modificare (update) în Beers nu afectează “FewBar”. De asemenea nici o adăugare în Drinkers.

“Triggers”: Motivație

- Constrângerile declarative (assertion) reprezintă o unealtă puternică, dar SGBD-ul nu știe cu exactitate când trebuie verificate.
- Constrângerile la nivel Atribut și Tuplă sunt verificate la momente de timp bine determinate, dar nu sunt unelte la fel de puternice.
- Trigerele permit utilizatorului să decidă când să se verifice orice fel de condiție.

Reguli “Event-Condition-Action”

- O altă denumire pentru “trigger” este “*ECA rule*”, sau “*event-condition-action* rule”.
- *Event* : de obicei este un tip de modificare bază de date, de exemplu, adăugare în Sells.
- *Condition* : Orice expresie logică SQL.
- *Action* : orice secvență de instrucțiuni SQL.

Exemplu: Trigger

- În loc să se folosească o constrângere cheie străină și să se respingă adăugările în **Sells(bar, beer, price)** pentru mărci de bere necunoscute, un “trigger” poate adăuga acea marcă de bere în Beers, cu valoarea NULL pentru producător (manf).

Exemplu: Definiția Trigerului

```
CREATE TRIGGER BeerTrig
    AFTER INSERT ON Sells
        REFERENCING NEW ROW AS NewTuple
        FOR EACH ROW
            WHEN (NewTuple.beer NOT IN
                (SELECT name FROM Beers))
                INSERT INTO Beers(name)
                    VALUES(NewTuple.beer);
```

Eveniment

Condiție

Acțiune

Opțiuni: Evenimentul

- AFTER poate fi BEFORE.
 - Dacă relația este o vedere (view), atunci poate fi INSTEAD OF.
 - Actualizările prin intermediul vederilor pot fi efectuate prin definiția de trigere ce translatează actualizările în operații asupra tabelelor de bază.
- INSERT poate fi DELETE sau UPDATE.
 - UPDATE poate fi UPDATE . . . pentru un atribut particular.

Opțiuni: FOR EACH ROW

- Trigerele sunt “row-level” sau “statement-level”.
- FOR EACH ROW indică “row-level”; dacă lipsește indică “statement-level”.
- *Row level triggers* : se execută o singură dată pentru fiecare tuplă modificată.
- *Statement-level triggers* : se execută o singură dată pentru o instrucțiune SQL, indiferent de numărul tuplelor modificate.

Optiuni: REFERENCING

- Instructiunile INSERT implică o tuplă nouă (pentru “row-level”) sau o tabelă nouă (pentru “statement-level”).
 - “tabela nouă” este mulțimea tuplelor inserate.
- DELETE implică o tuplă sau o tabelă “old”.
- UPDATE implică ambele.
- Se face referire la aceste lucruri prin [NEW OLD][TUPLE TABLE] AS <nume>

Caz particular MS SQL Server

- Trigerele sunt “statement-level”.
- Pentru referirea la valorile vechi există tabela (temporară) “deleted”.
- Pentru referirea la valorile noi există tabela (temporară) “inserted”.

Optiuni: Conditia

- Orice conditie ce are valoare logică.
- Este evaluată pentru starea bazei de date înainte sau după evenimentul ce a declanșat trigerul, depinde de ce cuvânt cheie a fost folosit în definiția trigerului (BEFORE sau AFTER).
 - Evaluarea se face totdeauna înainte ca modificările să aibă efect.
- Tupla/Tabela “new”/“old” este accesată prin intermediul numelor folosite în clauza REFERENCING.

Optiuni: Actiunea

- Pot exista mai multe instrucțiuni SQL ca și acțiune.
 - Se folosesc BEGIN . . . END atunci când există mai multe instrucțiuni SQL.
- Interogările nu au sens ca și acțiune, prin urmare există limitare doar la actualizări.

Caz particular MS SQL Server

- Acțiunea unui trigger poate conține orice instrucțiune validă Transact SQL:

```
IF OBJECT_ID ('Purchasing.LowCredit', 'TR') IS NOT NULL
    DROP TRIGGER Purchasing.LowCredit
GO
CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT
AS
DECLARE @creditrating tinyint, @vendorid int
SELECT @creditrating = v.CreditRating, @vendorid = p.VendorID
FROM Purchasing.PurchaseOrderHeader p INNER JOIN inserted i ON
    p.PurchaseOrderID = i.PurchaseOrderID JOIN Purchasing.Vendor v on
    v.VendorID = i.VendorID
IF @creditrating = 5
BEGIN
    RAISERROR ('This vendor''s credit rating is too low to accept new
               purchase orders.', 16, 1)
ROLLBACK TRANSACTION
END
```

Exemplu

- Se folosesc relațiile **Sells(bar, beer, price)** și **RipoffBars(bar)** (are un singur atribut), pentru a întreține lista barurilor ce cresc prețul oricărei mărci de bere cu mai mult de 1 \$.

Trigerul

CREATE TRIGGER PriceTrig

AFTER UPDATE OF price ON Sells

REFERENCING

OLD ROW AS ooo

NEW ROW AS nnn

FOR EACH ROW

WHEN(nnn.price > ooo.price + 1.00)

INSERT INTO RipoffBars

VALUES(nnn.bar);

Evenimentul –
doar modificări
ale prețului

Update ne permite
să discutăm despre
tuple “old” și “new”

Se ia în considerare
fiecare preț modificat

Condiția:
creșterea
prețului
mai mult
de 1 \$

La creșterea prețului
suficient de mult, se
adaugă barul în
RipoffBars

Vederi

“View”

- O vedere (*view*) este o relație definită în termeni de tabele stocate (numite *tabele de bază*) și alte vederi.
- Există două categorii de vederi:
 1. *Virtuale* = datele afișate nu sunt stocate în baza de date; reprezintă doar o interogare pentru construirea relației.
 2. *Materializate* = sunt stocate.

Declararea Vederilor

- Se declară cu instrucțiunea:

```
CREATE [MATERIALIZED] VIEW  
    <nume> AS <interrogare>;
```

- Implicit o vedere este virtuală.

Exemplu

- CanDrink(drinker, beer) este o vedere ce “conține” perechi drinker-beer astfel încât persoana frecventează cel puțin un bar ce oferă berea la vânzare:

```
CREATE VIEW CanDrink AS
    SELECT drinker, beer
    FROM Frequent, Sells
    WHERE Frequent.bar = Sells.bar;
```

Exemplu: Accesul la o Vedere

- Interogarea unei vederi se face ca și în cazul tabelei de bază.
 - Actualizarea unei vederi este limitată la o singură tabelă de bază.
- Exemplu de interogare:

```
SELECT beer FROM CanDrink  
WHERE drinker = 'Sally';
```

Trigere definite pentru Vederi

- În general, este imposibil de actualizat o vedere virtuală, pentru că ea nu există fizic.
- Există însă trigerul INSTEAD OF, ce permite interpretarea actualizărilor produse vederii astfel încât actualizările să aibă sens.
- **Exemplu:** Vederea Synergy are triplete (drinker, beer, bar) cu semnificația barul servește berea, persoana frecventează barul și persoanei îi place berea.

Exemplu: Vederea Synergy

CREATE VIEW Synergy AS

SELECT Likes.drinker, Likes.beer, Sells.bar

FROM Likes, Sells, Frequent

WHERE Likes.drinker = Frequent.drinker

AND Likes.beer = Sells.beer

AND Sells.bar = Frequent.bar;

Câte o copie a fiecărui atribut

Natural join între Likes,
Sells și Frequent

Interpretarea Adăugării Într-o Vedere

- Nu se poate adăuga în Synergy --- este o vedere virtuală definită pe trei tabele de bază.
- Se poate în schimb folosi un triger INSTEAD OF pentru a transforma adăugarea tripletului **(drinker, beer, bar)** în vederea Synergy, în adăugări în toate cele trei tabele de bază, câte o adăugare pentru fiecare: Likes, Sells și Frequent.
- Sells.price va trebui completat cu NULL.

Trigerul

```
CREATE TRIGGER ViewTrig
```

```
INSTEAD OF INSERT ON Synergy
```

```
REFERENCING NEW ROW AS n
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO LIKES VALUES(n.drinker, n.beer);
```

```
    INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
```

```
    INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
```

```
END;
```

Vederi Materializate

- **Problemă:** de fiecare dată când una din tabelele de bază este actualizată, vederea materializată este posibil să trebuiască să fie actualizată.
 - Nu ne putem permite să recalcăm vederea la fiecare actualizare.
- **Soluție:** Periodic se reconstruiește vederea materializată, în caz contrar vederea este “out of date”.

Exemplu: Listă e-mail

- Lista de e-mail cu studenții unei grupe se presupune o vedere materializată.
- Dacă este introdus un nou student în baza de date, el nu apare în vedere până la împrospătarea vederii.
 - De aceea studentul respectiv nu va primi e-mailuri primite de colegii săi până la împrospătarea vederii.

Exemplu: Data Warehouse

- Wal-Mart (lanț de magazine) stochează fiecare vânzare la fiecare magazin într-o bază de date.
- În cursul nopții, vânzările din ziua respectivă sunt folosite pentru a actualiza *data warehouse* = vederi materializate cu vânzările.
- Datele din data warehouse sunt analizate de analiști ce prezic tendințe și mută bunurile la magazinele unde sunt vânzările cele mai bune.

Aplicații cu BD

Persistent Stored Modules (PSM)
Embedded SQL

Utilizare SQL în aplicații

1. Codul scris într-un limbaj specializat este stocat în BD (de exemplu PSM, PL/SQL – Oracle sau Transact-SQL – Microsoft SQL Server).
2. Instrucțiunile SQL sunt incluse într-un *limbaj gazdă* (cum este C).
3. Se folosesc unelte de conectare pentru a permite unui limbaj convențional să acceseze o bază de date (de exemplu CLI, JDBC, PHP/DB).

Proceduri Stocate

- PSM, sau “*persistent stored modules*,” permit stocare de proceduri ca elemente ale schemei BD.
- PSM = un amalgam de instrucțiuni convenționale (if, while, etc.) și SQL.
- Oferă posibilități altfel existente în SQL.

Formatul de bază PSM

```
CREATE PROCEDURE <nume> (  
    <listă de parametri> )  
<declarații locale optionale>  
<corp>;
```

□ Alternativa este Funcția:

```
CREATE FUNCTION <nume> (  
    < listă de parametri > ) RETURNS <tip>
```

Parametri în PSM

- Spre deosebire de alte limbaje cum este C unde există perechi nume-tip, PSM folosesc triplete *mod-nume-tip*, unde *mod* poate fi:
 - IN = procedura folosește valoarea, dar nu o poate modifica.
 - OUT = procedura modifică valoarea, nu o utilizează.
 - INOUT = ambele.

Exemplu: Procedură Stocată

- Vom scrie o procedură ce primește două argumente b și p , iar acțiunea constă în adăugarea unei tuple la relația **Sells(bar, beer, price)** ce are "bar = 'Joe's Bar'", "beer = b " și "price = p ".
- Este utilizată de Joe pentru a-și alcătui mai ușor meniul.

Procedura

```
CREATE PROCEDURE JoeMenu (
```

```
    IN b      CHAR(20),  
    IN p      REAL
```

Parametrii sunt amândoi
read-only, nu pot fi modificați

```
)
```

```
INSERT INTO Sells  
VALUES('Joe''s Bar', b, p);
```

Corpul:
o singură adăugare

Apelul Procedurilor

- Se folosește instrucțiunea EXECUTE (CALL) urmată de numele procedurii și de argumente.

- Exemplu:

```
EXECUTE JoeMenu ('Mooseadrool', 5.00);
```

- Funcțiile sunt folosite în expresii SQL oriunde se potrivește valoarea returnată (ca tip de dată).

Instrucțiuni specifice PSM

- RETURN <expresie> setează valoarea returnată de funcție.
 - Spre deosebire de C, etc., RETURN *NU* termină execuția funcției.
- DECLARE <nume> <tip> se folosește pentru a declara variabile locale.
- BEGIN . . . END grupează instrucțiuni.
 - Instrucțiunile se separă cu “;”.

Instrucțiuni specifice PSM

□ Instrucțiuni de atribuire:

SET <variabilă> = <expresie>;

□ Exemplu: SET b = 'Bud' ;

□ Etichete: se prefixează numele cu ":".

Instrucțiuni IF

- Forma simplificată:

```
IF <condiție> THEN  
    <instrucțiune(-i)>  
END IF;
```

- Se adaugă ELSE <instrucțiune(-i)>,
 IF . . . THEN . . . ELSE . . . END IF;
- Se poate adăuga suplimentar ELSEIF
 <instrucțiune(-i)>: IF ... THEN ... ELSEIF ...
 THEN ... ELSEIF ... THEN ... ELSE ... END IF;

Exemplu: IF

- Să se noteze barurile după numărul de clienți. Se folosește **Frequents(drinker,bar)**.
 - Clienți <100 : 'nepopular'.
 - Clienți 100-199 : 'mediu'.
 - Clienți ≥ 200 : 'popular'.
- Funcția **Rate(b)** acordă notă barului b.

Exemplu: IF (continuare)

```
CREATE FUNCTION Rate (IN b CHAR(20) )
```

```
    RETURNS CHAR(10)
```

```
    DECLARE cust INTEGER;
```

```
BEGIN
```

```
    SET cust = (SELECT COUNT(*) FROM Frequent  
                WHERE bar = b);
```

```
    IF cust < 100 THEN RETURN 'nepopular'
```

```
    ELSEIF cust < 200 THEN RETURN 'mediu'
```

```
    ELSE RETURN 'popular'
```

```
    END IF;
```

```
END;
```

Numărul
clientilor
barului b

Instrucțiune
IF imbricată
13

Return apare în acest loc, nu unde
se utilizează instrucțiunile RETURN

Bucle

- Forma de bază:

<nume buclă>: LOOP <instrucțiuni>
END LOOP;

- Ieșirea dintr-o buclă:

LEAVE <nume buclă>

Exemplu: Ieșirea din Buclă

bucla1: LOOP

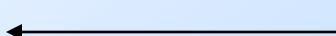
...

LEAVE bucla1;

Dacă este executată această instrucțiune . . .

...

END LOOP;



Controlul trece la acest punct (după END LOOP)

Alte Forme de Bucle

- WHILE <condiție>
 DO <instrucțiuni>
END WHILE;

- REPEAT <instrucțiuni>
 UNTIL <condiție>
END REPEAT;

Interogări

- În general interogările SELECT-FROM-WHERE *NU* sunt permise în PSM.
- Există trei moduri pentru a obține efectul unei interogări:
 1. Interogările ce produc o singură valoare pot fi utilizate ca expresie într-o atribuire.
 2. SELECT . . . INTO având rezultat 1 tuplă.
 3. Cursoare.

Exemplu: Atribuire/Interrogare

- Se folosește variabila locală p și **Sells(bar, beer, price)**, pentru a obține prețul la care "Joe" vinde "Bud":

```
SET p = (SELECT price FROM Sells  
WHERE bar = 'Joe''s Bar' AND  
beer = 'Bud') ;
```

SELECT . . . INTO

- O altă cale pentru a obține valoarea unei interogări ce returnează 1 tuplă este să se folosească **INTO <variabilă>** după clauza **SELECT**.

- **Exemplu:**

```
SELECT price INTO p FROM Sells  
WHERE bar = 'Joe''s Bar' AND  
beer = 'Bud';
```

Cursoare

- Un *cursor* este în esență o variabilă de tuplă ce parcurge toate tuplele din rezultatul unei anumite interogări.
- Se declară cursorul *c* în felul următor:
DECLARE c CURSOR FOR <interogare>;

Deschiderea și Închiderea de Cursoare

- Pentru a folosi cursorul c , trebuie emisă comanda:
OPEN c;
- Interogarea din definiția c este evaluată și c este setat să facă referire la prima tuplă a rezultatului.
- La terminarea lucrului cu c , este emisă comanda:
CLOSE c;

Extragerea Tuplelor dintr-un Cursor

- Pentru a obține următoarea tuplă din cursorul c , este emisă comanda:
 $\text{FETCH FROM } c \text{ INTO } x_1, x_2, \dots, x_n;$
- x -urile sunt o listă de variabile, câte una pentru fiecare componentă a tuplei referite de către c .
- c se deplasează automat la următoarea tuplă.

Ieșirea din bucla unui Cursor

- În mod obișnuit un cursor este folosit într-o buclă creată cu instrucțiunea FETCH, pentru fiecare tuplă extrasă se execută o anumită acțiune.
- Se pune întrebarea cum se ieșe din buclă atunci când nu mai există tuple de prelucrat?

Ieșirea din bucla unui Cursor

- Fiecare operație SQL returnează o *stare*, ce este un sir de caractere format din 5 cifre.
 - De exemplu, 00000 = “Este în regulă,” și 02000 = “Regăsirea unei tuple a eșuat.”
- În PSM, se poate obține valoarea stării într-o variabilă numită SQLSTATE.

Ieșirea din bucla unui Cursor

- Se poate declara o *condiție*, care să fie o variabilă logică ce este “true” numai dacă SQLSTATE are o valoare particulară.
- **Exemplu:** Se poate declara condiția NeGasit pentru a reprezenta 02000 :

```
DECLARE NeGasit CONDITION FOR  
SQLSTATE '02000' ;
```

Ieșirea din bucla unui Cursor

- Structura buclei unui cursor arată în felul următor:

Buclacursor: LOOP

...

 FETCH c INTO ... ;

 IF NeGasit THEN LEAVE Buclacursor;

 END IF;

...

END LOOP;

Exemplu: Cursor

- Presupunem că dorim să scriem o procedură care să examineze **Sells(bar, beer, price)** și să mărească cu 1 (\$) prețul berilor vândute de “Joe’s Bar” și care au prețul mai mic decât 3 (\$).
 - Obs. Se putea rezolva cu un simplu UPDATE, dar scopul este de a vedea soluția folosind un cursor.

Declarațiile necesare

```
CREATE PROCEDURE JoeGouge()
```

```
    DECLARE Berea CHAR(20);
```

```
    DECLARE Pretul REAL;
```

```
    DECLARE NeGasit CONDITION FOR
```

```
        SQLSTATE '02000';
```

```
    DECLARE c CURSOR FOR
```

```
        (SELECT beer, price FROM Sells  
         WHERE bar = 'Joe''s Bar');
```

Se vor folosi pentru a păstra perechi bere-preț la extragerea datelor folosind cursorul c

Se returnează meniul la "Joe's Bar"

Corpul Procedurii

BEGIN

OPEN c;

Bucla_menui: LOOP

 FETCH c INTO Berea, Pretul;

 IF NeGasit THEN LEAVE Bucla_menui END IF;

 IF Pretul < 3.00 THEN

 UPDATE Sells SET price = Pretul + 1.00

 WHERE bar = 'Joe''s Bar' AND beer = Berea;

 END IF;

END LOOP;

CLOSE c;

END;

Verifică dacă cel mai recent FETCH nu a regăsit nici o tuplă

Dacă "Joe's Bar" are un preț mai mic de 3 (\$) pentru bere, mărește prețul acelei mărci de bere la barul "Joe's Bar" cu 1 (\$).

PL/SQL

- Oracle folosește o variantă de SQL/PSM numită PL/SQL.
- PL/SQL permite crearea de proceduri stocate și funcții, ce pot fi executate din “*generic query interface*” (sqlplus), ca orice instrucțiune SQL.
- În sqlplus se poate scrie o instrucțiune PL/SQL asemănător cu corpul unei proceduri, dar aceasta este executată o singură dată.

Instrucțiuni PL/SQL

DECLARE

 <declarații>

BEGIN

 <instrucțiuni>

END;

.

run

Secțiunea DECLARE este optională.

Procedura în PL/SQL

CREATE OR REPLACE PROCEDURE

<nume> (<argumente>) **AS**

De notat
prezența "AS"

<declarații optionale>

BEGIN

<instrucțiuni PL/SQL>

END;

.
run

Efectul este stocarea
procedurii în BD;
nu execuția ei.

Exemplu:JoeMenu

- Se va defini în PL/SQL procedura **JoeMenu(b,p)** ce adaugă berea *b* cu prețul *p* la lista berilor vândute de barul “Joe’s Bar” (în relația Sells).

Procedura “JoeMenu” în PL/SQL

```
CREATE OR REPLACE PROCEDURE JoeMenu (
    b IN Sells.beer%TYPE,
    p IN Sells.price%TYPE
) AS
BEGIN
    INSERT INTO Sells
        VALUES ('Joe''s Bar', b, p);
END;
```

De notat aceste
precizări
generice de tip.

run

Procedură stocată în Microsoft SQL Server

- Următoarea procedură returnează un cursor ce conține descrierea valutelor:

```
USE AdventureWorks;
GO
IF OBJECT_ID ( 'dbo.uspCurrencyCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.uspCurrencyCursor;
GO
CREATE PROCEDURE dbo.uspCurrencyCursor
    @CurrencyCursor CURSOR VARYING OUTPUT
AS
    SET @CurrencyCursor = CURSOR
        FORWARD_ONLY STATIC FOR
            SELECT CurrencyCode, Name
            FROM Sales.Currency;
    OPEN @CurrencyCursor;
GO
```

Procedură stocată în Microsoft SQL Server

- Următoarea secvență de instrucțiuni prezintă utilizarea cursorului creat de procedura stocată `uspCurrencyCursor` într-un script SQL:

```
USE AdventureWorks;
GO
DECLARE @MyCursor CURSOR;
EXEC dbo.uspCurrencyCursor @CurrencyCursor = @MyCursor OUTPUT;
WHILE (@@FETCH_STATUS = 0)
BEGIN;
    FETCH NEXT FROM @MyCursor;
END;
CLOSE @MyCursor;
DEALLOCATE @MyCursor;
GO
```

Embedded SQL

- Ideea de bază: Un preprocesor translatează instrucțiunile SQL în apeluri de procedură ce se încadrează în codul limbajului gazdă.
- Toate instrucțiunile “embedded SQL” încep cu EXEC SQL, în aşa fel încât preprocesorul să le descopere cu ușurință.

Variabile Partajate

- Pentru a “închega” SQL cu programul în limbaj gazdă, cele două părți trebuie să partajeze anumite variabile.
- Declarațiile variabilelor partajate sunt încadrate de:

→ EXEC SQL BEGIN DECLARE SECTION;

Sunt
necesare <declarații limbaj gazdă>

→ EXEC SQL END DECLARE SECTION;

Utilizarea Variabilelor Partajate

- În SQL, variabilele partajate trebuie să fie precedate de “:.”
 - Ele pot fi folosite ca și constante furnizate de programul limbaj gazdă.
 - Ele pot primi valori prin instrucțiuni SQL și pot transfera aceste valori programului limbaj gazdă.
- În limbajul gazdă, variabilele partajate se comportă ca orice altă variabilă.

Exemplu: Căutarea Prețului

- Se va utiliza limbajul “C” cu “embedded SQL” pentru a schița părțile importante ale unei funcții de căutare a prețului cunoscându-se denumirea berii și denumirea barului.
- Se folosește relația **Sells(bar, beer, price)**.

Exemplu: C și SQL

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
    char Barul[21], Berea[21];
```

```
    float Pretul;
```

```
EXEC SQL END DECLARE SECTION;
```

```
/* obținerea valorilor Barul și Berea */
```

```
EXEC SQL SELECT price INTO :Pretul
```

```
    FROM Sells
```

```
    WHERE bar = :Barul AND beer = :Berea;
```

```
/* anumite operații cu Pretul */
```

De notat, tablourile
au 21 caractere
pentru 20
caractere +
endmarker

SELECT-INTO
asemănător
cu PSM

Embedded Queries

- “Embedded SQL” are aceleasi limitări ca și PSM cu privire la interogări:
 - SELECT-INTO pentru o interogare trebuie să garanteze că produce o singură tuplă.
 - În caz contrar, trebuie folosit un cursor.
 - Există mici diferențe sintactice, dar ideea rămâne aceeași.

Instrucțiuni Cursor

- Declararea unui cursor *c* se face cu:

EXEC SQL DECLARE *c* CURSOR FOR <interrogare>;

- Se deschide și se închide cursorul *c* cu:

EXEC SQL OPEN CURSOR *c*;

EXEC SQL CLOSE CURSOR *c*;

- Datele se extrag din cursorul *c* cu:

EXEC SQL FETCH *c* INTO <variabilă(-e)>;

- Macroul NOT FOUND este true dacă și numai dacă FETCH eșuează în a mai găsi o tuplă.

Exemplu: Tipărirea meniului “Joe’s Bar”

- Vom scrie C + SQL pentru a tipări meniul “Joe’s Bar” – lista perechilor bere-preț pe care o regăsim din relația **Sells(bar, beer, price)** cu condiția “bar = Joe’s Bar”.
- Un cursor va vizita fiecare tuplă Sells ce are bar = “Joe’s Bar”.

Exemplu: Declarațiile

```
EXEC SQL BEGIN DECLARE SECTION;  
    char Berea[21]; float Pretul;  
EXEC SQL END DECLARE SECTION;  
  
EXEC SQL DECLARE c CURSOR FOR  
    SELECT beer, price FROM Sells  
    WHERE bar = 'Joe''s Bar';
```

Cursorul se declară în exteriorul secțiunii de declarații

Exemplu: Partea Executabilă

```
EXEC SQL OPEN CURSOR c;  
while(1) {  
    EXEC SQL FETCH c  
        INTO :Berea, :Pretul;  
    if (NOT FOUND) break;  
    /* se formatează și se tipărește Berea și Pretul */  
}  
EXEC SQL CLOSE CURSOR c;
```

Stilul C pentru terminarea repetiției

Necesitatea SQL Dinamic

- Majoritatea aplicațiilor folosesc interogări specifice și instrucțiuni de actualizare ce interacționează cu BD.
 - SGBD-ul compilează instrucțiuni EXEC SQL ... înapeluri procedură specifice și produce un program obișnuit în limbaj gazdă ce folosește o bibliotecă.
- Sqlplus nu știe ceea ce are nevoie să facă până la execuție.

SQL Dinamic

- Pregătirea (“prepare”) unei interogări:
EXEC SQL PREPARE <nume-interogare>
 FROM <textul interogării>;
- Execuția unei interogări:
EXEC SQL EXECUTE <nume-interogare>;
- “Prepare” = optimizare interogare.
- Se pregătește o dată, se execută de mai multe ori.

Exemplu: O Interfață Generică

```
EXEC SQL BEGIN DECLARE SECTION;
    char interogare[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
    /* se afișează prompterul SQL> */
    /* se citește interogarea utilizator în tabloul
interogare */
    EXEC SQL PREPARE q FROM :interogare;
    EXEC SQL EXECUTE q;
}
```

q este o variabilă SQL ce reprezintă forma optimizată a unei instrucțiuni ce este ținută în :*interogare*

Execute-Immediate

- Dacă urmează să executăm doar o singură dată interogarea, se pot combina pașii PREPARE și EXECUTE în unul singur:

```
EXEC SQL EXECUTE IMMEDIATE <text>;
```

Exemplu: Interfață Generică

```
EXEC SQL BEGIN DECLARE SECTION;
    char interogare[MAX_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {
    /* se afișează prompter SQL> */
    /* se citește interogarea utilizator în tabloul
interogare */
    EXEC SQL EXECUTE IMMEDIATE :interogare;
}
```

Aplicații cu BD

Call-Level Interface
Java Database Connectivity
PHP

Intercalare de SQL

- Interogările SQL sunt adeseori construite de către programe.
- Aceste interogări primesc *constante* introduse de utilizator.
- Atunci când codul nu este tratat cu atenție, se pot întâmpla lucruri neașteptate.

Exemplu: Intercalare SQL

- Presupunem că există relația
Utilizatori(nume, parolă, cont)
- Se construiește o interfață Web : se citește numele și parola utilizatorului, în sirurile *n* și *p*, se emite interogarea, se afișează numărul contului.

```
SELECT cont FROM Utilizatori  
WHERE nume = :n AND parola = :p
```

Ecranul utilizator:

Nume:

Comentariu
în Oracle sau
MS SQL Server

Parola:

Numărul contului este 1234-567

Interogarea la Execuție

```
SELECT cont FROM Utilizatori
```

```
WHERE nume = 'Misu' -- ' AND  
parola = 'interesant?'
```

Sunt tratate ca și comentariu

Limbaj Gazdă/ Interfețe SQL

Via Biblioteci

- A treia abordare pentru conectarea la BD a limbajelor convenționale este apelul prin intermediu bibliotecilor.
 1. C + CLI
 2. Java + JDBC
 3. PHP + PEAR/DB

Arhitectura “Three-Tier”

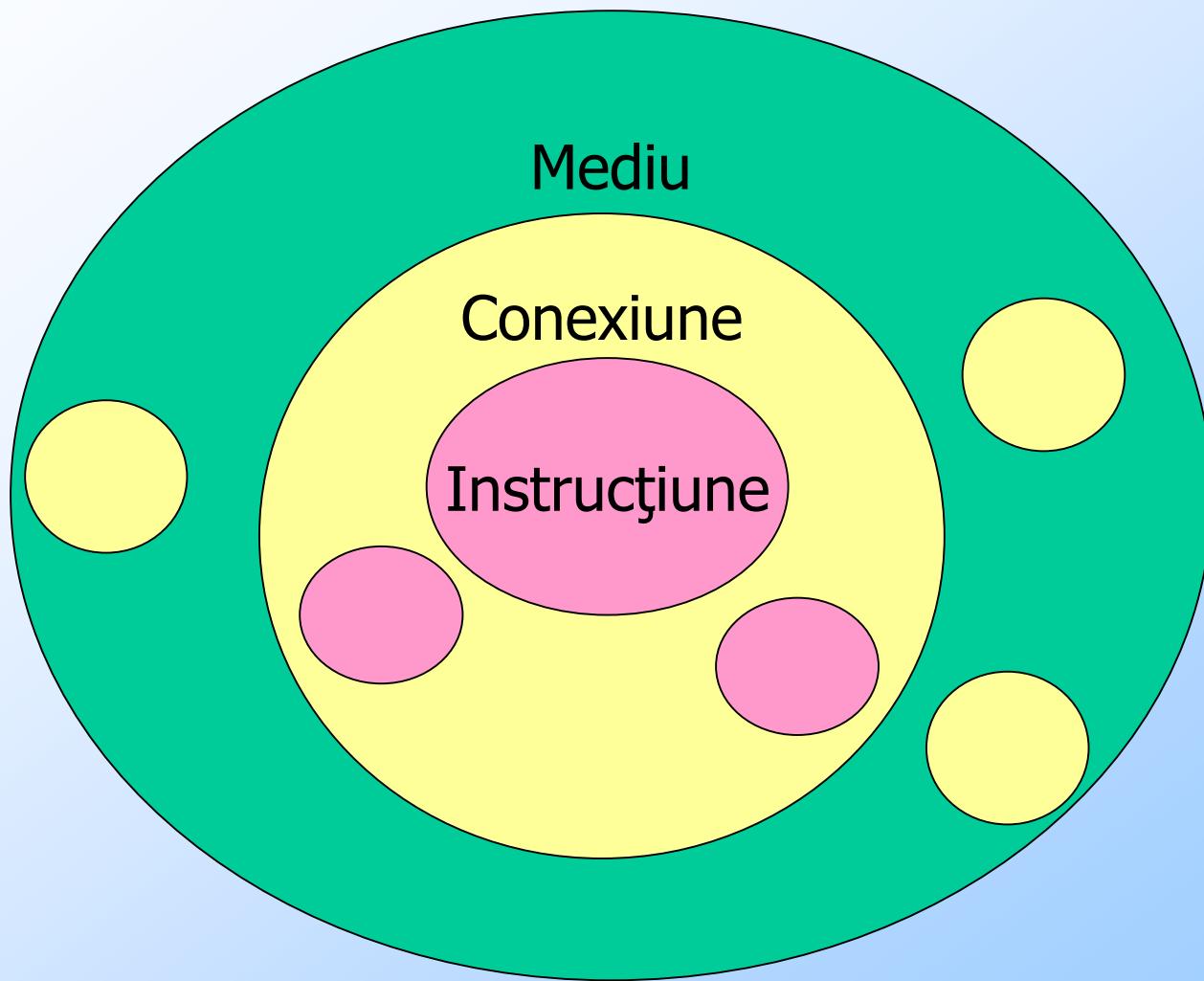
- Un mediu obișnuit de utilizare a BD are trei straturi de procesoare:
 1. *Servere Web* --- tratează cu utilizatorul.
 2. *Servere de Aplicație* --- execută “business logic”.
 3. *Servere BD* --- extrag din BD ceea ce este necesar pentru serverele de aplicație.

Exemplu: Amazon

- BD păstrează informația despre produse, clienți, etc.
- “Business logic” include printre altele “ce trebuie făcut după ce este selectat ‘checkout?’”
 - **Răspuns:** Să se afișeze ecranul “cum veți achita?”.

Medii, Conexiuni, Interogări

- BD este, pentru majoritatea limbajelor de acces la baze de date, un *mediu* ("environment").
- Serverele BD întrețin un număr de *conexiuni*, astfel încât serverele de aplicații pot lansa interogări sau pot efectua actualizări.
- Serverul de aplicație emite de obicei *instrucțiuni* : interogări și actualizări.



SQL/CLI

- În loc să se folosească un preprocesor (embedded SQL), se poate folosi o bibliotecă cu funcții.
- Biblioteca pentru C este numită SQL/CLI = “*Call-Level Interface*”
- Preprocesorul pentru embedded SQL translatează instrucțiunile EXEC SQL ... în CLI sau apeluri similare.

Structuri de Date

- C se conectează la BD cu structuri de tipul următor:
 1. *Medii* : reprezintă SGBD-uri instalate.
 2. *Conexiuni* : conexiuni (login) la BD.
 3. *Instructiuni* : instrucțiuni SQL transmise printr-o conexiune.
 4. *Descrieri* : înregistrări despre tuplele unei interogări, sau parametrii unei instrucțiuni.

Handle-uri

- Funcția **SQLAllocHandle(T,I,O)** este folosită pentru a crea aceste structuri, ce se numesc *handle-uri* de mediu, conexiune și instrucțiune.
 - T = tip, de exemplu, SQL_HANDLE_STMT.
 - I = handle de intrare = structură la un nivel superior (instrucțiune < conexiune < mediu).
 - O = (adresa unui) handle de ieșire.

Exemplu: SQLAllocHandle

```
SQLAllocHandle (SQL_HANDLE_STMT,  
    con_Mea, &inst_Mea);
```

- Con_Mea este un handle de conexiune creat anterior.
- inst_Mea este numele unui handle de instrucțiune ce va fi creat.

Pregătirea și Executarea

- **SQLPrepare(H, S, L)** cauzează interpretarea sirului S , de lungime L , ca fiind o instrucțiune SQL și optimizarea ei;
 - instrucțiunea executabilă este plasată în handle-ul de instrucțiune H .
- **SQLExecute(H)** cauzează execuția instrucțiunii SQL reprezentată de handle-ul de instrucțiune H .

Exemplu: Pregătire și Execuție

```
SQLPrepare(inst_Mea, "SELECT  
    beer, price FROM Sells  
    WHERE bar = 'Joe''s Bar'",  
    SQL_NTS);
```

```
SQLExecute(inst_Mea);
```

Această constantă precizează că argumentul al doilea este “null-terminated string”; adică, lungimea este determinată prin numărarea caracterelor.

Execuția Directă

- Dacă o instrucțiune S va fi executată doar o singură dată, se poate combina PREPARE și EXECUTE cu:

SQLExecuteDirect(H,S,L);

- Unde, H este un handle de instrucțiune și L este lungimea sirului S .

Extragerea Tuplelor

- La execuția instrucțiunii SQL, dacă aceasta este o interogare, este nevoie să se extragă tuplele rezultat.
 - Un cursor este implicit prin faptul că s-a executat o interogare, cursorul nu trebuie declarat explicit.
- **SQLFetch(H)** obține următoarea tuplă din rezultatul instrucțiunii cu handle-ul *H*.

Accesarea Rezultatului Interrogării

- La extragerea unei tuple, este nevoie să se preia componentele tuplei undeva.
- Fiecare componentă este legată la o variabilă prin funcția **SQLBindCol**.
 - Această funcție are 6 argumente, din care vor fi prezentate doar componentele 1, 2 și 4:
 - 1 = handle-ul instrucțiunii interogare.
 - 2 = numărul coloanei.
 - 4 = adresa variabilei.

Exemplu: “Binding”

- Presupunem că s-a executat apelul SQLExecute(inst_Mea), unde inst_Mea este handle-ul interogării:

```
SELECT beer, price FROM Sells  
WHERE bar = 'Joe''s Bar'
```

- Legarea rezultatului la Berea și Pretul:
SQLBindCol(inst_Mea , 1, , &Berea, ,);
SQLBindCol(inst_Mea , 2, , &Pretul, ,);

Exemplu: “Fetching”

- În acest moment se pot extrage toate tuplele răspunsului:

```
while ( SQLFetch(inst_Mea) != SQL_NO_DATA )
{
    /* aici se utilizează Berea și Pretul */
}
```

Macrou CLI ce reprezintă
SQLSTATE = 02000 = “failed
to find a tuple.”

JDBC

- *Java Database Connectivity* (JDBC) este o bibliotecă similară cu SQL/CLI, dar cu Java ca limbaj gazdă.
- Asemănător CLI, dar cu puține diferențe.

Specificarea unei Conexiuni

```
import java.sql.*;  
Class.forName(com.mysql.jdbc.Driver);  
Connection con_Mea =  
    DriverManager.getConnection(...);
```

Cel încărcat cu `forName`

Clasele JDBC

Driver pentru mySql

URL-ul BD, numele și parola

Instrucțiuni

- JDBC oferă două clase:
 1. *Statement* = un obiect ce poate accepta un string ce este o instrucțiune SQL și poate executa un astfel de string.
 2. *PreparedStatement* = un obiect ce are o instrucțiune SQL asociată, ce este gata de execuție.

Crearea Instrucțiunilor

- Clasa “Connection” are metode pentru a crea “Statements” și “PreparedStatement”.

```
Statement inst1 = con_Mea.createStatement();  
PreparedStatement inst2 =  
    con_Mea.createStatement()  
        "SELECT beer, price FROM Sells " +  
        "WHERE bar = 'Joe's Bar'"  
);
```

createStatement fără argumente returnează un obiect “Statement”; iar cu un argument returnează un obiect “PreparedStatement”.

Execuția instrucțiunilor SQL

- JDBC face distincție între interogări și actualizări, pe care le numește “updates.”
- “Statement” și “PreparedStatement” au fiecare metodele `executeQuery` și `executeUpdate`.
 - Pentru “Statements”: un argument - interogarea sau actualizarea ce trebuie executată.
 - Pentru “PreparedStatements”: nici un argument.

Exemplu: Actualizare

- inst1 este un obiect “Statement”.
- El se poate utiliza pentru a adăuga o tuplă:

```
inst1.executeUpdate(  
    "INSERT INTO Sells " +  
    "VALUES ('Brass Rail', 'Bud', 3.00)"  
) ;
```

Exemplu: Interogare

- inst2 este un obiect "PreparedStatement" ce păstrează interogarea:
"SELECT beer, price FROM Sells WHERE bar = 'Joe's Bar'"
- **executeQuery** returnează un obiect din clasa ResultSet, ce va fi examinat ulterior.
- Interogarea:

```
ResultSet meniu = inst2.executeQuery();
```

Accesul la “ResultSet”

- Un obiect de tipul ResultSet este ceva asemănător unui cursor.
- Metoda `next()` avansează “cursorul” la următoarea tuplă.
 - Prima dată când se aplică `next()`, se regăsește prima tuplă.
 - Dacă nu mai există tuple, `next()` returnează valoarea “`false`”.

Accesul la Componentele Tuplelor

- Atunci când “ResultSet” face referire la o tuplă, se pot obține componentele tuplei aplicând anumite metode pentru “ResultSet”.
 - Metoda $\text{get}X(i)$, unde X este un anumit tip, și i este numărul componentei, returnează valoarea componentei.
 - Valoarea trebuie să aibă tipul X .

Exemplu: Accesarea Componentelor

- Meniu = “ResultSet” pentru interogarea
“SELECT beer, price FROM Sells WHERE bar = ‘Joe’ ’s Bar”
- Sunt accesate “beer” și “price” pentru fiecare tuplă:

```
while ( meniu.next() ) {  
    Berea = Menu.getString(1);  
    Pretul = Menu.getFloat(2);  
    /*anumite operații cu Berea și Pretul*/  
}
```

PHP

- Un limbaj ce este folosit pentru acțiuni într-un text HTML.
- Este indicat de <? PHP *cod*?>.
- Există biblioteca DB în **PEAR** (PHP Extension and Application Repository).
 - Este inclusă cu include(DB.php).

Variabile în PHP

- Încep cu \$.
- Este permis să nu se declare tipul unei variabile.
- Se atribuie unei variable o valoare ce aparține unei “clase”, caz în care sunt accesibile metode ale acelei clase.

Valori “String”

- PHP rezolvă o problemă foarte importantă pentru limbajele ce construiesc sirurile de caractere ca și valori:
 - Cum se spune că un subșir trebuie interpretat ca variabilă și să fie înlocuit (subșirul) cu valoarea sa?
- Soluția PHP:
 - Ghilimele semnifică “se înlocuiește”;
 - Apostrofuri semnifică “nu se înlocuiește”.

Exemplu: Se înlocuiește sau Nu?

`$100` = "una sută dolari";

`$sue` = 'Îmi datorezi \$100.';

`$joe` = "Îmi datorezi \$100.";

□ Valoarea variabilei `$sue` este

'Îmi datorezi \$100',

□ În timp ce valoarea variabilei `$joe` este

'Îmi datorezi una sută dolari'.

Tablouri PHP

- Există două categorii: *numerice* și *asociative*.
- Tablourile numerice sunt cele obișnuite, indexate 0,1,...
 - **Exemplu:** \$a = array("Paul", "George", "John", "Ringo");
 - Atunci \$a[0] este "Paul", \$a[1] este "George", și.a.m.d.

Tablouri Asociative

- Elementele unui tablou asociativ \$a sunt perechi $x \Rightarrow y$, unde x este un sir de caractere cheie si y este orice valoare.
- Dacă $x \Rightarrow y$ este un element al lui \$a, atunci \$a[x] este y .

Exemplu: Tablouri Asociative

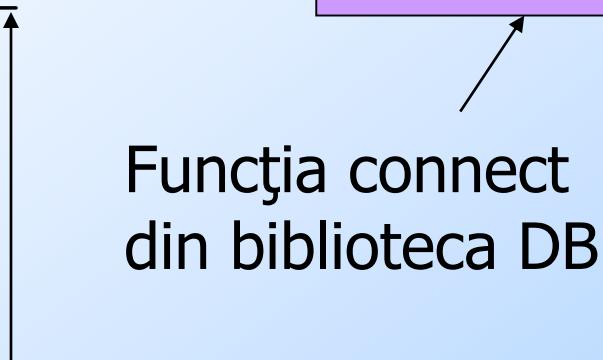
- Un mediu poate fi exprimat ca un tablou asociativ:

```
$med_Meu = array(  
    "phptype" => "mySql",  
    "hostspec" => "localhost",  
    "database" => "scoala",  
    "username" => "student",  
    "password" => "nuSeStie");
```

Efectuarea unei Conexiuni

- Se importă biblioteca DB și se folosește tabloul \$med_Meu:

```
$con_Mea = DB::connect($med_Meu);
```



Clasa variabilei este Connection
din cauză că este returnată
de DB::connect().

Execuția instrucțiunilor SQL

- Se aplică metoda **query** unui obiect Connection.
- Această metodă primește un argument de tipul string și returnează un rezultat.
 - Ce poate fi un cod de eroare sau relația returnată de o interogare.

Exemplu: Execuția unei Interrogări

- Să se găsească barurile ce vând o bere dată de variabila \$bere.

```
$bere = 'Bud';
```

Concatenare
în PHP

```
$rezultat = $con_Mea->query(  
    "SELECT bar FROM Sells"  
    ."")  
    ." WHERE beer = $bere ;");
```

De reținut că
variabila se înlocuiește
cu valoarea sa.

Aplicarea
metodei

Cursor în PHP

- Rezultatul unei interogări *este* reprezentat de tuplele returnate.
- Metoda **fetchRow** se aplică rezultatului și returnează următoarea tuplă, sau FALSE dacă nu mai există tuple.

Exemplu: Cursor

```
while ($bar =  
       $result->fetchRow()) {  
    // diferite actiuni cu $bar  
}
```

Exemplu: testora.php

```
<?php  
  
$c = oci_connect('student1', 'student', '//serverora/ordl');  
$s = oci_parse($c, 'select city from locations');  
oci_execute($s);  
  
while ($res = oci_fetch_array($s, OCI_ASSOC))  
{  
    echo $res['CITY'] . "<br>";  
}  
?>
```

Variabile BIND

- Aplicația re-execută instrucțiuni cu valori diferite
- Se îmbunătășește database throughput
- Ajută la prevenirea atacurilor de tip “SQL Injection”

Variabile BIND

```
$s = oci_parse($c, "select last_name from employees  
                      where employee_id = :eidbv");  
  
$myeid = 101;  
oci_bind_by_name($s, ":EIDBV", $myeid);  
oci_execute($s);  
oci_fetch_all($s, $res);  
echo "Numele: ". $res['LAST_NAME'][0] ."  
\n";  
  
$myeid = 102;  
oci_execute($s); // Nu necesită "re-parse"  
oci_fetch_all($s, $res);  
echo "Last name is: ". $res['LAST_NAME'][0] ."  
\n";
```

PL/SQL și PHP

```
create or replace procedure
myproc(a in varchar2, b in number) as
begin
insert into mytable (mydata, myid) values (a, b);
end;
/
<?php
$c = oci_connect('student1', 'student', '//serverora/orcl');
$s = oci_parse($c, "call myproc('mydata', 123)");
oci_execute($s);
?>
```

Exemplu: Tranzacție

```
function do_transactional_insert($c, $a)
{
    $s = oci_parse($c, 'insert into ptab (pdata) values
        (:bv)');
    oci_bind_by_name($s, ':bv', $v, 20, SQLT_CHR);
    foreach ($a as $v)
        $r = oci_execute($s, OCI_DEFAULT); // Nefinalizat
    oci_commit($c); // Finalizat
}
```

Modelul Relațional

Teoria relațională
Algebra Relațională
Calcul relațional

Modelul Relațional

□ E.F. Codd, 1970 (1972)

□ System R, 1977 (IBM)

□ Oracle, 1979

Modelul Relațional

- **Relație:** Fiind dată o colecție de mulțimi D_1, D_2, \dots, D_n (nu neapărat distințe), se spune că R este o relație pe aceste mulțimi dacă este o mulțime de n -tuple (d_1, d_2, \dots, d_n) astfel încât d_i aparține D_i , $i=1..n$
- Multimile D_1, D_2, \dots, D_n sunt domeniile relației R .
- n este **gradul** sau **aritatea** relației R .
- Numărul de n -tuple reprezintă **cardinalitatea** relației R .

Modelul Relațional

□ **Relație** (a doua definiție): Se definește produsul cartezian $D_1 \times D_2 \times \dots \times D_n$ al mulțimilor D_1, D_2, \dots, D_n multimea tuturor n-tuplelor ordonate (d_1, d_2, \dots, d_n) astfel încât d_1 aparține D_1 , d_2 aparține D_2, \dots, d_n aparține D_n .

O relație R pe mulțimile D_1, D_2, \dots, D_n este o submulțime a produsului cartezian $D_1 \times D_2 \times \dots \times D_n$.

Modelul Relațional - Paradigme

Relația este o mulțime (set) de n-tuple:

1. Nu există două elemente (n-tuple) identice.
2. Ordinea elementelor este indiferentă.

Modelul Relațional

- **Domeniu:** Ansamblul de valori admisibile pentru o componentă a unei relații.
- **Exemple:**
 - Domeniul numelor de persoane
 - Domeniul numelor de orașe
 - Domeniul notelor (mulțimea {1, 2, 3, 4, 5, 6, 7, 8, 9, 10})
- **Domenii compatibile:** Multimile de valori care le definesc sunt comparabile d.p.d.v. semantic.
- **Atribut:** Un domeniu cu nume, adică utilizarea unui domeniu sub un nume oarecare (într-o relație).

Modelul Relațional

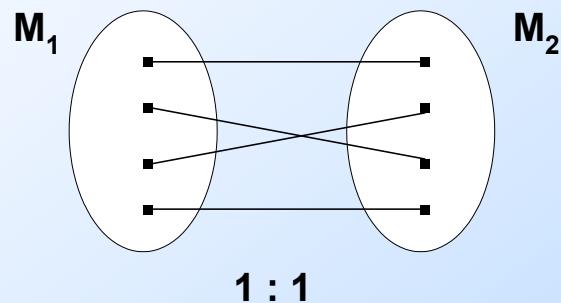
Cheie: Se numește cheie a unei relații R, un subset K al atributelor relației R ce satisface proprietățile:

1. Identificare unică, fiecare tuplă a relației R este identificată în mod unic de valorile atributelor care compun cheia K
2. Neredundanță, subsetul K este minimal în sensul că eliminarea oricărui atribut din K duce la pierderea proprietății 1.

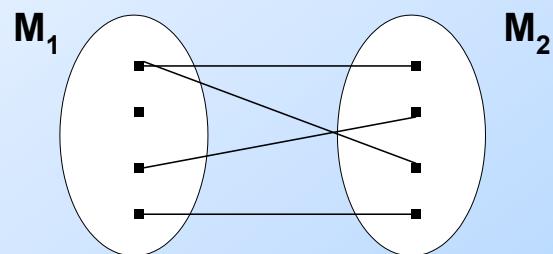
Modelul Relațional

- Tipuri de chei:
 - Primară
 - Candidată
 - Străină
- **Atribut prim**: Atribut constituent al unei chei.
 - O cheie poate fi *simplă* când are un singur atribut sau *compusă* când este formată din mai multe attribute.

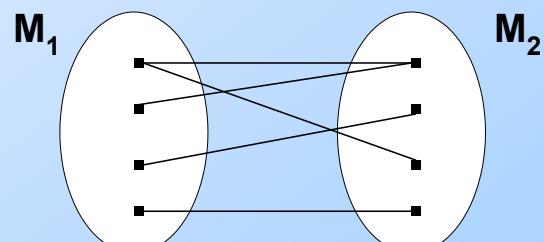
Tipuri de legături



$1 : 1$

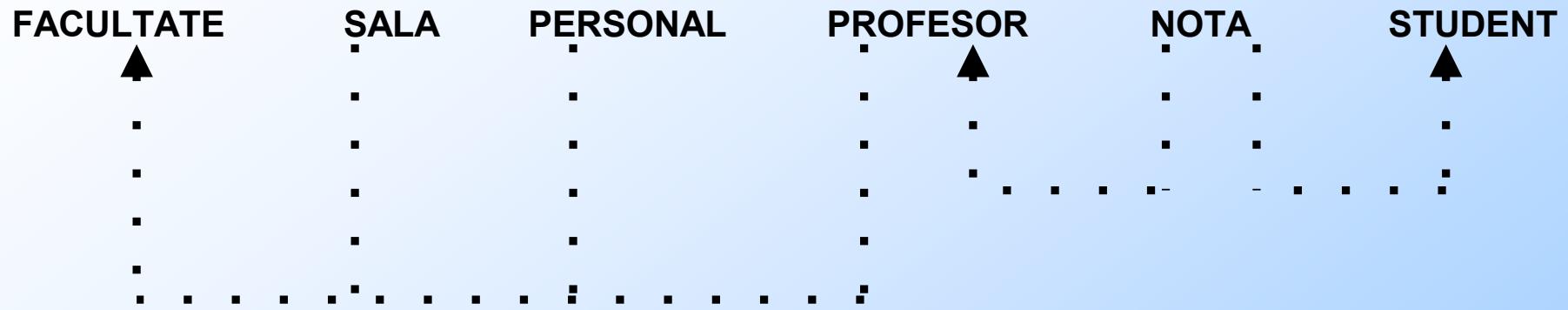


$1 : N$



$M : N$

Arborele de structură a datelor (modelul relațional)



Schema BD relaționale cuprinde relații și legături relaționale realizate prin **valoare** (chei străine) și NU prin pointeri ca în modelele ierarhic sau rețea, de aceea structura de date este de tip NIVEL, toate nodurile sunt pe același nivel, fiecare nod reprezintă o relație. Legăturile M:N se implementează prin introducerea unei RELAȚII DE LEGĂTURĂ (NOTA).

SGBD total relational

- Principiul integrității domeniului
- Principiul integrității relației
- Principiul integrității referinței
- LMD cel puțin echivalent cu algebra relatională

“Algebră”

- Sistem matematic ce constă din:
 - *Operanzi* --- variabile sau valori din care se construiesc valori noi.
 - *Operatori* --- simboluri ce denotă procedurile ce construiesc valorile noi din valori existente.

Algebra Relațională

- Este o algebră ai cărei operanzi sunt relații sau variabile ce reprezintă relații.
- Operatorii sunt concepuți astfel încât să fie efectuate operațiile dorite cu relațiile din BD.
- Rezultatul este o algebră ce poate fi utilizată ca un *limbaj de interogare* pentru relații.

Esența Algebrei Relaționale

- Uniune, intersecție și diferență.
 - Operațiile obișnuite pe mulțimi, dar *ambii operanzi trebuie să aibă aceeași schemă de relație.*
- Selectie: alege anumite rânduri.
- Proiecție: alege anumite coloane.
- Produs și join: compun din relații.
- Redenumire relații și attribute.

Operatori Primitive (cinci)

- Reuniunea
- Diferența
- Produsul cartezian
- Selectia
- Proiectia

Selectie

- $R1 := \sigma_C(R2)$
- C este o condiție (asemănător cu instrucțiunea "if") ce face referire la attributele din $R2$.
- $R1$ conține acele tuple din $R2$ ce satisfac C .

Exemplu: Selectie

Relația Sells:

| bar | beer | price |
|-------|--------|-------|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |
| Sue's | Bud | 2.50 |
| Sue's | Miller | 3.00 |

JoeMenu := $\sigma_{\text{bar}=\text{"Joe's"}}(\text{Sells})$:

| bar | beer | price |
|-------|--------|-------|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |

Proiecție

□ $R1 := \pi_L(R2)$

□ L este o listă de atribute din schema relației $R2$.

□ $R1$ este construită în felul următor:

- Mai întâi pentru fiecare tuplă din $R2$ se extrag atributele din lista L , în ordinea specificată.
- Se elimină tuplele dupicate, dacă există.

Exemplu: Proiecție

Relația Sells:

| bar | beer | price |
|-------|--------|-------|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |
| Sue's | Bud | 2.50 |
| Sue's | Miller | 3.00 |

Prices := $\Pi_{\text{beer}, \text{price}}(\text{Sells})$:

| beer | price |
|--------|-------|
| Bud | 2.50 |
| Miller | 2.75 |
| Miller | 3.00 |

Proiecție Extinsă

- Se folosește același operator Π_L , dar se permit în lista L expresii arbitrare ce implică attribute:
 1. Expresii aritmetice cu attribute, de exemplu: $A+B->C$.
 2. Duplicarea atributelor (un atribut să apară de mai multe ori).

Exemplu: Proiecție Extinsă

$$R = (\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array})$$

$$\pi_{A+B \rightarrow C, A, A}(R) =$$

| C | A1 | A2 |
|---|----|----|
| 3 | 1 | 1 |
| 7 | 3 | 3 |

Produs

□ $R3 := R1 \times R2$

- Face pereche între fiecare tuplă $t1$ din $R1$ și fiecare tuplă $t2$ din $R2$.
- O tuplă din $R3$ se obține prin concatenarea $t1t2$.
- Schema relației $R3$ este constituită din atributele din $R1$ și apoi din $R2$, în ordine.
- Atenție la atributele cu același nume în $R1$ și $R2$, de exemplu A : se folosește exprimarea $R1.A$ și $R2.A$.

Exemplu: $R3 := R1 \times R2$

$R1($

| A, | B |
|----|---|
| 1 | 2 |
| 3 | 4 |

$R2($

| B, | C |
|----|----|
| 5 | 6 |
| 7 | 8 |
| 9 | 10 |

$R3($

| A, | R1.B, | R2.B, | C |
|----|-------|-------|----|
| 1 | 2 | 5 | 6 |
| 1 | 2 | 7 | 8 |
| 1 | 2 | 9 | 10 |
| 3 | 4 | 5 | 6 |
| 3 | 4 | 7 | 8 |
| 3 | 4 | 9 | 10 |

Theta-Join

- $R3 := R1 \bowtie_C R2$
 - Se face produsul $R1 \times R2$.
 - Apoi se aplică σ_C rezultatului.
- Ca și pentru σ , C poate fi orice condiție cu valoare booleană.
 - Versiuni istorice ale acestui operator permitteau doar $A \theta B$, unde θ este $=, <$, etc.; de unde și numele “theta-join.”

Exemplu: Theta Join

Sells(

| | bar, | beer, | price |) |
|-------|--------|-------|-------|---|
| Joe's | Bud | 2.50 | | |
| Joe's | Miller | 2.75 | | |
| Sue's | Bud | 2.50 | | |
| Sue's | Coors | 3.00 | | |

Bars(

| | name, | addr |) |
|-------|-------|-----------|---|
| Joe's | | Maple St. | |
| Sue's | | River Rd. | |

BarInfo := Sells $\bowtie_{Sells.bar = Bars.name}$ Bars

BarInfo(

| | bar, | beer, | price, | name, | addr |) |
|-------|--------|-------|--------|-------|-----------|---|
| Joe's | Bud | 2.50 | | Joe's | Maple St. | |
| Joe's | Miller | 2.75 | | Joe's | Maple St. | |
| Sue's | Bud | 2.50 | | Sue's | River Rd. | |
| Sue's | Coors | 3.00 | | Sue's | River Rd. | |

Natural Join

- Este o variantă folositoare (join *natural*) conectează două relații prin:
 - Egalizarea atributelor cu același nume și
 - Proiecția unei singure copii a fiecărui atribut pereche (unul din attributele egalizate).
- Notație $R3 := R1 \bowtie R2$.

Exemplu: Natural Join

Sells(

| bar, | beer, | price |
|-------|--------|-------|
| Joe's | Bud | 2.50 |
| Joe's | Miller | 2.75 |
| Sue's | Bud | 2.50 |
| Sue's | Coors | 3.00 |

)

Bars(

| bar, | addr |
|-------|-----------|
| Joe's | Maple St. |
| Sue's | River Rd. |

)

BarInfo := Sells \bowtie Bars

Notă: Bars.name a devenit Bars.bar pentru a face posibil natural join.

BarInfo(

| bar, | beer, | price, | addr |
|-------|--------|--------|-----------|
| Joe's | Bud | 2.50 | Maple St. |
| Joe's | Miller | 2.75 | Maple St. |
| Sue's | Bud | 2.50 | River Rd. |
| Sue's | Coors | 3.00 | River Rd. |

)

Redenumire

- Operatorul ρ redefineste schema unei relatiilor.
- $R_1 := \rho_{R_1(A_1, \dots, A_n)}(R_2)$ produce R_1 , o relatie cu atributele A_1, \dots, A_n si aceleasi tuple ca si R_2 .
- Notatia simplificata : $R_1(A_1, \dots, A_n) := R_2$.

Exemplu: Redenumire

Bars(

| | |
|-------|-----------|
| name, | addr |
| Joe's | Maple St. |
| Sue's | River Rd. |

)

$R(\text{bar}, \text{addr}) := \text{Bars}$

$R($

| | |
|-------|-----------|
| bar, | addr |
| Joe's | Maple St. |
| Sue's | River Rd. |

 $)$

Construirea de Expresii Complexe

- Se combină operatorii folosind paranteze și reguli de precedență.
- Există trei notații, ca și la expresiile aritmetice:
 1. Secvențe de instrucțiuni de atribuire.
 2. Expresii cu mai mulți operatori.
 3. Arbori expresie.

Secvențe de Atribuiriri

- Sunt create nume de relații temporare.
- Redenumirea poate fi implicată de acordarea relațiilor a unei liste de attribute.
- **Exemplu:** $R3 := R1 \bowtie_C R2$ poate fi rescrisă:

$R4 := R1 \times R2$

$R3 := \sigma_C(R4)$

Expresii Într-o Singură Atribuire

- **Exemplu:** operația theta-join $R3 := R1 \bowtie_C R2$ poate fi rescrisă: $R3 := \sigma_C(R1 \times R2)$
- Precedența operatorilor relaționali:
 1. $[\sigma, \pi, \rho]$ (cei mai prioritari).
 2. $[x, \bowtie]$.
 3. \cap .
 4. $[\cup, -]$

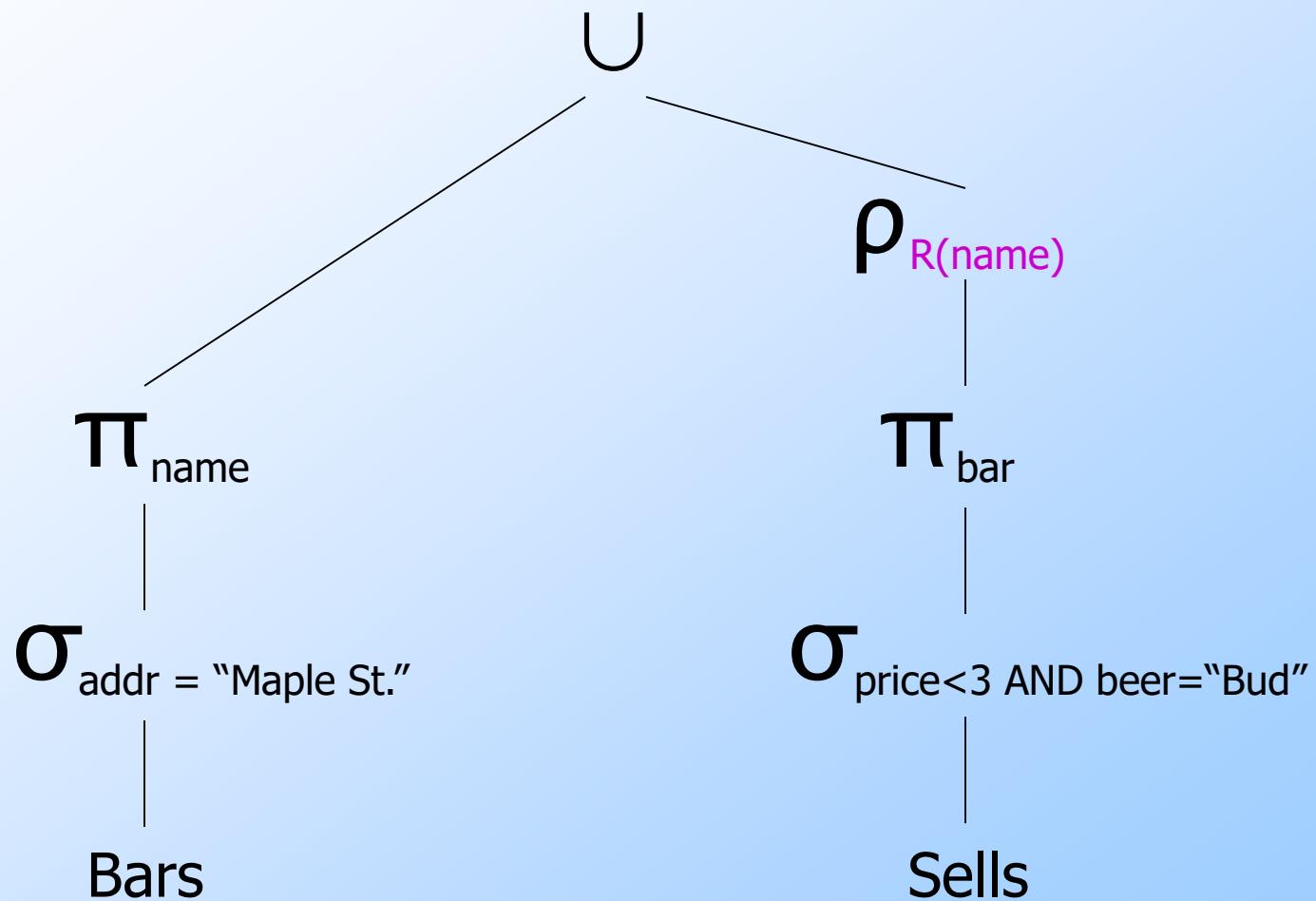
Arbore Expresie

- Frunzele sunt operanți:
 - variabile ce precizează relații;
 - în particular, relații constante.
- Nodurile interioare sunt operatori, aplicații nodurilor fiu.

Exemplu: Arborele unei Interrogări

- Se folosesc relațiile `Bars(name, addr)` și `Sells(bar, beer, price)`, pentru a găsi numele barurilor ce fie se găsesc pe “Maple St.” fie vând “Bud” mai ieftin de 3 (\$).

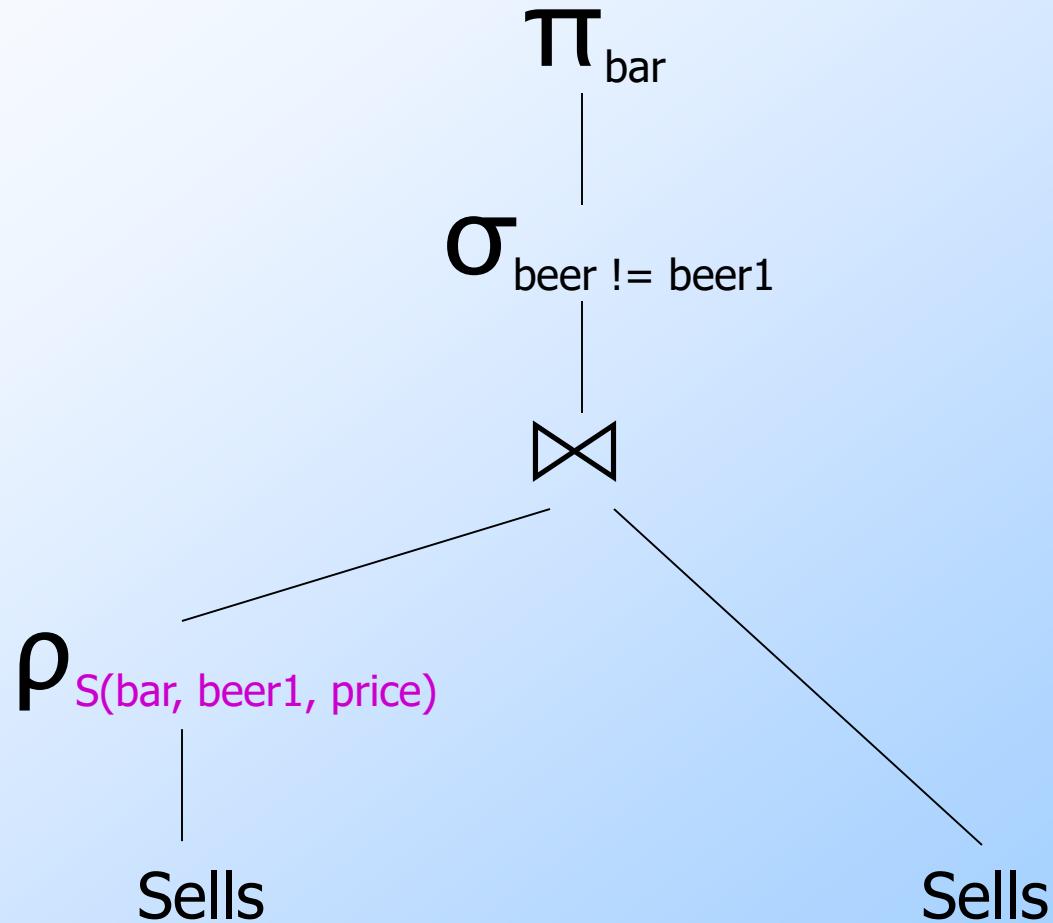
Arborele:



Exemplu: Self-Join

- Se folosește **Sells(bar, beer, price)**, pentru a găsi barurile ce vând două beri diferite la același preț.
- **Strategia**: prin redenumire, se definește o copie a relației Sells, numită **S(bar, beer1, price)**. Join natural între Sells și S constă din cvadruplele (bar, beer, beer1, price) astfel încât barul vinde ambele beri ("beer" și "beer1") la prețul "price".

Arborele



Scheme pentru Rezultate

- **Uniune, intersecție și diferență:** schemele celor doi operanzi trebuie să fie aceleași, aşa că se folosește acea schemă pentru rezultat.
- **Selectie:** schema rezultatului este aceeași cu schema operandului.
- **Proiecție:** lista atributelor dă schema.

Scheme pentru Rezultate

- **Produs**: schema este constituită din atributele ambelor relații.
 - Se folosește R.A, etc., pentru a distinge două attribute numite A .
- **Theta-join**: asemănător cu produs.
- **Natural join**: uniunea atributelor celor două relații.
- **Redenumire**: operatorul dă schema.

Algebra Relațională pe “Bags”

- Un *bag* (sau *multiset*) este asemănător unui set, dar un element poate apărea de mai multe ori.
- Exemplu: {1,2,1,3} este un bag.
- Exemplu: {1,2,3} este un bag ce se întâmplă să fie și un set.

De ce “Bags”?

- SQL, limbajul de interogare cel mai important pentru BD relaționale, este de fapt un limbaj pe “bag”.
- Anumite operații, cum este proiecția, sunt mult mai eficiente pe bag-uri decât set-uri.

Operații pe Bag-uri

- **Selectia** se aplică fiecărei tuple, astfel încât efectul pe bag-uri este același cu efectul pe set-uri.
- **Proiecția** se aplică de asemenea fiecărei tuple, dar ca operator “bag”, nu se elimină duplicatele.
- **Produsul și join-urile** sunt efectuate pe fiecare pereche de tuple, deci duplicatele din bag-uri nu au efect asupra modului de operare.

Exemplu: Selecție “Bag”

$R($

| A, | B |
|----|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

$$\sigma_{A+B < 5} (R) =$$

| A | B |
|---|---|
| 1 | 2 |
| 1 | 2 |

Exemplu: Proiecție “Bag”

$R($

| A, | B |
|----|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

)

$$\pi_A(R) =$$

| A |
|---|
| 1 |
| 5 |
| 1 |

Exemplu: Produs "Bag"

$R($

| | |
|----|---|
| A, | B |
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

 $)$

| | |
|----|---|
| A, | B |
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

$S($

| | |
|----|---|
| B, | C |
| 3 | 4 |
| 7 | 8 |

 $)$

| | |
|----|---|
| B, | C |
| 3 | 4 |
| 7 | 8 |

$R \times S =$

| A | R.B | S.B | C |
|---|-----|-----|---|
| 1 | 2 | 3 | 4 |
| 1 | 2 | 7 | 8 |
| 5 | 6 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 7 | 8 |

Exemplu: Theta-Join “Bag”

$R($

| A, | B |
|----|---|
| 1 | 2 |
| 5 | 6 |
| 1 | 2 |

$S($

| B, | C |
|----|---|
| 3 | 4 |
| 7 | 8 |

$R \bowtie_{R.B < S.B} S =$

| A | R.B | S.B | C |
|---|-----|-----|---|
| 1 | 2 | 3 | 4 |
| 1 | 2 | 7 | 8 |
| 5 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 7 | 8 |

Uniune “Bag”

- Un element apare în uniunea a două bag-uri suma numărului de apariții din fiecare bag.
- Exemplu: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Intersecție “Bag”

- Un element apare în intersecția a două bag-uri de minimul numărului de apariții în cele două bag-uri.
- Exemplu: $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$.

Diferența “Bag”

- Un element apare în diferența $A - B$ a două bag-uri de atâtea ori cât apare în A , minus numărul de apariții în B .
 - Dar niciodată mai puțin de 0 ori.
- Exemplu: $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$.

Atenție: Legile Bag!= Legile Set

- Anumite reguli, dar *nu toate* regulile algebrice ce sunt valabile pentru set-uri sunt de asemenea valabile pentru bag-uri.
- **Exemplu:** comutativitatea uniunii ($R \cup S = S \cup R$) se păstrează pentru bag-uri.
 - Deoarece adunarea este comutativă, prin adăugarea numărului de apariții ale lui x în R și S nu depinde de ordinea lui R și S .

Exemplu: Regulă ce nu se păstrează

- Uniunea set-urilor este *idempotentă*, ceea ce înseamnă că $S \cup S = S$.
- Pentru bag-uri, dacă x apare de n ori în S , atunci el apare de $2n$ ori în $S \cup S$.
- Astfel $S \cup S \neq S$ în general.
 - adică, $\{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

Semijoin

- Se utilizează de către SGBD-uri pe baze de date distribuite și se notează:

$$A \bowtie B$$

- Se presupune că cele două relații A și B ce trebuie cuplate se găsesc la locații diferite.
- Pentru optimizarea traficului în rețea este mai convenabil să se transporte doar multimea de valori corespunzătoare atributului de join de la un site la altul, se efectuează join și la urmă probabil la o a treia locație se transportă fractiunile din cele două relații pentru a obține rezultatul final.

Semijoin

A \bowtie B

- După notația de mai sus se transportă mai întâi de la locația de reședință a relației B la locația de reședință a relației A.
- De exemplu A este `Student(nr_matr, nume, gen, coda)` și B este `Nota(nr_matr, code, nota)`. Se trasportă doar `nr_matr` din `Nota` la locația unde se găsește `Student`.

Diviziunea

- **Definiție:** Diviziunea relației A de grad m prin relația B de grad n, notată $A \div B$, este o relație R de grad $m-n$, formată din mulțimea tuplelor r cu proprietatea că pentru orice tuplă b din B există o tupla a în A egală cu rezultatul concatenării tuplelor r și b .
- Multimea atributelor relației B trebuie să fie o submulțime a mulțimii atributelor relației A.
- Relația R conține doar acele attribute din A ce nu apar în B.
- O tuplă din relația A este reținută pentru rezultat dacă și numai dacă este legată de fiecare tuplă din relația B.

Exemplu: Diviziunea

A

| Nr_matr | Nume | Prenume | Data_n | Nota | Disciplina |
|---------|--------|-----------|------------|------|--------------------|
| 1 | Pop | Victor | 31.12.1980 | 8 | Baze de Date |
| 1 | Pop | Victor | 31.12.1980 | 9 | Programare |
| 2 | Albert | Alexandru | 04.01.1979 | 7 | Baze de Date |
| 2 | Albert | Alexandru | 04.01.1979 | 9 | Programare |
| 2 | Albert | Alexandru | 04.01.1979 | 8 | Sisteme de Operare |
| 3 | Pop | Mariana | 12.03.1982 | 6 | Baze de Date |
| 3 | Pop | Mariana | 12.03.1982 | 8 | Programare |
| 3 | Pop | Mariana | 12.03.1982 | 10 | Sisteme de Operare |
| 3 | Pop | Mariana | 12.03.1982 | 7 | Analiza Matematica |
| 4 | Marcu | Ioana | 07.04.1983 | 8 | Baze de Date |
| 4 | Marcu | Ioana | 07.04.1983 | 9 | Programare |
| 5 | Vasile | Mircea | 27.03.1981 | 7 | Baze de Date |
| 5 | Vasile | Mircea | 27.03.1981 | 8 | Programare |
| 5 | Vasile | Mircea | 27.03.1981 | 9 | Sisteme de Operare |

B

| Nota | Disciplina |
|------|--------------|
| 8 | Baze de Date |
| 9 | Programare |

$$R = A \div B$$

R

| Nr_matr | Nume | Prenume | Data_n |
|---------|-------|---------|------------|
| 1 | Pop | Victor | 31.12.1980 |
| 4 | Marcu | Ioana | 07.04.1983 |

$$A \div B = \pi_X(A) - \pi_X(((\pi_X(A)) \times B) - A)$$

| A | B |
|--------|----|
| X | Z |
| Pop | 8 |
| Pop | 7 |
| Pop | 10 |
| Man | 9 |
| Man | 9 |
| Man | 9 |
| Man | 7 |
| Vesa | 8 |
| Vesa | 9 |
| Vesa | 10 |
| Mircea | 7 |
| Mircea | 8 |
| Mircea | 10 |

$$A \div B = \pi_X(A) - \pi_X(((\pi_X(A)) \times B) - A)$$

| $\pi_X(A)$ |
|------------|
| X |
| Pop |
| Man |
| Vesa |
| Mircea |

| $(\pi_X(A)) \times B$ | |
|-----------------------|----|
| X | Z |
| Pop | 8 |
| Pop | 7 |
| Pop | 10 |
| Man | 8 |
| Man | 7 |
| Man | 10 |
| Vesa | 8 |
| Vesa | 7 |
| Vesa | 10 |
| Mircea | 8 |
| Mircea | 7 |
| Mircea | 10 |

| $((\pi_X(A)) \times B) - A$ | |
|-----------------------------|------|
| X | Y(Z) |
| Man | 8 |
| Man | 10 |
| Vesa | 7 |

$$A \div B = \pi_X(A) - \pi_X(((\pi_X(A)) \times B) - A)$$

$$\pi_X(((\pi_X(A)) \times B) - A)$$

| |
|------|
| X |
| Man |
| Vesa |

$$\pi_X(A) - \pi_X(((\pi_X(A)) \times B) - A)$$

| |
|--------|
| X |
| Pop |
| Mircea |

Limitări ale Algebrei Relaționale

□ Tratarea recursivității

- De exemplu o tuplă în relația **Traseu(plecare, sosire, nr_tren)** conține date despre două stații adiacente din mersul trenurilor
- Pentru a răspunde la întrebarea “**Care este traseul între două localități pentru un tren cu *nr_tren* dat?**” este nevoie de o succesiune nedeterminabilă inițial de pași
- La un pas se cuplează după predicatul **plecare = sosire \wedge nr_tren = valoare** relația **Traseu** cu ea însăși

Limitari ale Algebrei Relationale

TRASEU

| Plecare | Sosire | Nr_tren |
|-----------|-------------|---------|
| Bucuresti | Ploiesti | 311 |
| Ploiesti | Sinaia | 311 |
| Sinaia | Predeal | 311 |
| Predeal | Brasov | 311 |
| Brasov | Cluj-Napoca | 311 |

R1

| Plecare | Sosire |
|-----------|----------|
| Bucuresti | Ploiesti |

R2

| Plecare1 | Sosire1 | Sosire2 |
|-----------|----------|---------|
| Bucuresti | Ploiesti | Sinaia |

R3

| Plecare1 | Sosire1 | Sosire2 | Sosire3 |
|-----------|----------|---------|---------|
| Bucuresti | Ploiesti | Sinaia | Predeal |

R4

| Plecare1 | Sosire1 | Sosire2 | Sosire3 | Sosire4 |
|-----------|----------|---------|---------|---------|
| Bucuresti | Ploiesti | Sinaia | Predeal | Brasov |

R5

| Plecare1 | Sosire1 | Sosire2 | Sosire3 | Sosire4 | Sosire5 |
|-----------|----------|---------|---------|---------|-------------|
| Bucuresti | Ploiesti | Sinaia | Predeal | Brasov | Cluj-Napoca |

Calcul Relațional - Caracteristici

- neprocedural sau *declarativ*
- descrie răspunsul la interogare fără a da detalii explicite despre cum va fi executată interogarea

Calcul Relațional - Variante

- Calcul Relațional al Tuplelor (CRT)
 - Variabilele din CRT sunt variabile de tuplă.
 - Din acest punct de vedere SQL se aseamănă cu CRT.

- Calcul Relațional al Domeniilor (CRD)
 - Variabilele din CRD sunt variabile de domeniu.
 - Din acest punct de vedere QBE se aseamănă cu CRD.

CRT

- *Variabila de tuplă* este o variabilă ale cărei valori sunt tuple ale unei scheme particulare de relație.
- Fiecare valoare atribuită unei variabile de tuplă are același număr și același tip de attribute.

Interogare CRT

- Are forma $\{T \mid p(T)\}$, unde T este o variabilă de tuplă, iar $p(T)$ denotă o *formulă* ce descrie T .
 - T este singura variabilă liberă din formula p
- Rezultatul interogării este setul tuplelor t pentru care formula $p(T)$ este evaluată “true” prin înlocuirea $T = t$.
- Limbajul ce descrie formulele $p(T)$ este esența CRT și este un subset simplu al “*first-order logic*”.

Exemplu: Interrogare CRT

- Folosind **Beers(name, manf)**, care sunt mărcile de bere produse de Anheuser-Busch?

$$\{ B.\text{name} \mid B \in \text{Beers} \wedge B.\text{manf} = \text{'Anheuser-Busch'} \}$$

- Răspunsul conține componenta “name” pentru acele instanțe ale lui B ce trec testul.

CRT – Sintaxa Interogării

- Rel - nume relație
- $R \text{ și } S$ - variabile de tuplă
 - a - un atribut din R
 - b - un atribut din S
- op - un operator din mulțimea $\{<, \leq, \neq, \geq, >\}$

Formulă atomică

- $R \in Rel$
- $R.a$ op $S.b$
- $R.a$ op *constantă*
- *constantă* op $R.a$

Formulă CRT

- Se definește recursiv ca fiind una din următoarele, unde p și q sunt formule și $p(R)$ denotă o formulă în care apare variabila R :
 - orice formulă atomică
 - $\neg p, p \vee q, p \wedge q, p \Rightarrow q$
 - $\exists R(p(R))$, unde R este o variabilă de tuplă
 - $\forall R(p(R))$, unde R este o variabilă de tuplă

Formulă CRT

- În ultimele două clauze, cuantificatorii \forall și \exists se spune că “*leagă*” variabila R .
- O variabilă se spune că este “*liberă*” într-o formulă sau subformulă (o formulă conținută într-o formulă mai mare) dacă (sub)formula nu conține un cuantificator care să o lege.

Formulă CRT

- Se observă că fiecare variabilă într-o formulă CRT apare într-o subformulă ce este atomică, și fiecare schemă de relație specifică un domeniu pentru fiecare atribut.
- Asigură că fiecare variabilă într-o formulă CRT are un domeniu bine definit ("well-defined") din care sunt luate valorile variabilei.
 - Fiecare variabilă are un *tip* bine definit ("well-defined").

Exemplu: Interrogare CRT

- Se folosesc relațiile Likes(drinker, beer) și Frequent(drinker, bar), pentru a găsi băuturile preferate de cel puțin o persoană ce frecventează "Joe's Bar".

```
{ P | ∃ L ∈ Likes ∧ ∃ F ∈ Frequent  
(F.bar = 'Joe''s Bar' ∧ L.drinker  
= F.drinker ∧ P.beer = L.beer) }
```

CRD

- O variabilă de domeniu ia valori din domeniul unui anumit atribut.
- O interogare CRD este de forma $\{<x_1, x_2, \dots, x_n> \mid p(<x_1, x_2, \dots, x_n>)\}$.
- Unde fiecare x_i este sau o variabilă de domeniu sau o constantă și $p(<x_1, x_2, \dots, x_n>)$ denotă o formulă CRD cu unicele variabile libere x_i , $1 \leq i \leq n$.
- Rezultatul interogării este mulțimea tuplelor $\{<x_1, x_2, \dots, x_n>$ pentru care formula este evaluată “true”.

Formulă CRD

- Este definită asemănător cu formula CRT.
- Diferența constă în faptul că variabilele sunt variabile de domeniu.
- op - un operator din mulțimea $\{<, \leq, \neq, \geq, >\}$
- X și Y sunt variabile de domeniu

Formulă atomică CRD

- $\langle x_1, x_2, \dots, x_n \rangle \in Rel$
 - unde Rel este o relație cu n attribute
 - fiecare x_i , $1 \leq i \leq n$ este fie o variabilă fie o constantă
- X op Y
- X op *constantă*
- *constantă* op X

Formulă CRD

- Se definește recursiv una din următoarele, unde p și q sunt formule și $p(X)$ denotă o formulă în care apare variabila X .
 - Orice formulă atomică
 - $\neg p, p \vee q, p \wedge q, p \Rightarrow q$
 - $\exists R(p(R))$, unde R este o variabilă de domeniu
 - $\forall R(p(R))$, unde R este o variabilă de domeniu

Exemplu: Interogare CRD

- Folosind **Beers(name, manf)**, care sunt mărcile de bere produse de Anheuser-Busch?

$$\{ \langle N \rangle \mid \langle N, M \rangle \in \text{Beers} \wedge M = \text{'Anheuser-Busch'} \}$$

Interogări nesigure

- Se consideră interogarea:

$$\{S \mid \neg(S \in Sells)\}$$

- Sintactic este corectă.
- Multimea tuplelor S este evident infinită, în contextul domeniilor infinite.
- Ideea este de a restriona calculul relațional pentru a nu permite interogări nesigure.

Formule independente de domeniu

- $\text{DOM}(F)$ unde F este o formulă din calculul relațional, este reuniunea dintre mulțimea constantelor ce apar în F și mulțimea tuturor valorilor de atrbute ce apar în relațiile specificate ca parametri în F .
- Fie $F(X_1, X_2, \dots, X_n)$ o formulă și D o mulțime de valori astfel încât $\text{DOM}(F) \subseteq D$

Relația corespunzătoare formulei F în raport cu mulțimea de valori D este mulțimea tuplelor

$$(x_1, x_2, \dots, x_n) \in D \times D \times \dots \times D$$

astfel încât prin substituirea fiecărei variabile de domeniu X_i cu valoarea corespunzătoare x_i , $F(X_1, X_2, \dots, X_n)$ devine adevarată.

Formule independente de domeniu

- Formula F este independentă de domeniu dacă relația corespunzătoare acesteia în raport cu orice $D \supseteq \text{DOM}(F)$ nu depinde de mulțimea de valori D .
- Exemplu: Fie $F(X,Y) = \neg R(X,Y)$ și fie $R(X,Y)=\{(a,a),(b,b)\} \rightarrow \text{DOM}(F)=\{a,b\}$

Fie $D_1=\{a,b\}$ și $D_2=\{a,b,c\}$

$R_1(X,Y)=\{(a,b),(b,a)\}$

$R_2(X,Y)=\{(a,b),(b,a),(a,c),(c,a),(b,c),(c,b)\}$

$R_1 \neq R_2 \Rightarrow F(X,Y) = \neg R(X,Y)$ este dependentă de domeniu

Formule sigure

- Independența de domeniu a unei formule F este o problemă indecidabilă.
- Formule sigure \subseteq Formule independente de domeniu.
- Proprietăți:
 - Orice formulă sigură este formulă independentă de domeniu.
 - Fiind dată o formulă F, se poate spune dacă este sau nu sigură.
 - Orice interogare din algebra relațională poate fi exprimată cu ajutorul formulelor sigure.

Formule sigure

□ Restricții:

□ formula NU conține \forall (transformarea de echivalență
 $(\forall X)F \equiv \neg(\exists X)\neg F$)

□ $F_1 \vee F_2 \Rightarrow F_1(X_1, X_2, \dots, X_n) \wedge F_2(X_1, X_2, \dots, X_n)$

unde (X_1, X_2, \dots, X_n) sunt variabile libere

□ orice variabilă liberă din $F_1 \wedge F_2 \wedge \dots \wedge F_m$ trebuie să fie variabilă limitată:

- variabilă liberă în cel puțin o F ce nu este comparație aritmetică sau negație
- F este de forma $X = a$ sau $a = X$, unde a este o constantă
- F este de forma $X = Y$ sau $Y = X$, Y este limitată

□ În $F_1 \wedge F_2 \wedge \dots \wedge F_m$ cel puțin o F_i trebuie să fie pozitivă

Elemente de proiectare BD (1)

Dependențe Funcționale
Decompoziții
Forme Normale

Problema proiectării bazelor de date relaționale

- Alegerea schemelor de relație și modul de grupare al atributelor în relații pentru a reprezenta tipuri de entități sau legături între tipuri de entități.
- Există mai multe alegeri posibile, unele pot fi mai convenabile decât altele.
- Dependența datelor → Ideea centrală (influențează proprietățile relațiilor în raport cu operațiile curente)

Dependențe Funcționale

- $X \rightarrow Y$ este o declarație pentru relația R astfel încât oricând două tuple din R au aceleași valori pentru toate atributele ce formează X , atunci valorile din cele două tuple pentru toate atributele din setul Y trebuie să coincidă.
 - Se spune "Aserțiunea $X \rightarrow Y$ este valabilă în R ."
 - X, Y, Z reprezintă seturi de atribute
 - A, B, C reprezintă atribută singulare

Divizarea membrului dreapta al unei DF

- $X \rightarrow A_1 A_2 \dots A_n$ este valabilă pentru R dacă și numai dacă $X \rightarrow A_1$, $X \rightarrow A_2, \dots$, $X \rightarrow A_n$ sunt valabile pentru R .
- Exemplu: $A \rightarrow BC$ este echivalent cu $A \rightarrow B$ și $A \rightarrow C$.
- Nu există regulă de divizare a membrului stânga al unei DF.
- În general DF au în partea dreapta un singur element.

Exemplu: DF

Drinkers(name, addr, beersLiked, manf, favBeer)

- Se poate gândi că:
 1. name → addr favBeer
 - Notă această FD este identică cu name → addr și name → favBeer.
 2. beersLiked → manf

Exemplu: Date Posibile

| name | addr | beersLiked | manf | favBeer |
|---------|------------|------------|--------|-----------|
| Janeway | Voyager | Bud | A.B. | WickedAle |
| Janeway | Voyager | WickedAle | Pete's | WickedAle |
| Spock | Enterprise | Bud | A.B. | Bud |

Deoarece $\text{name} \rightarrow \text{addr}$

Deoarece $\text{name} \rightarrow \text{favBeer}$

Deoarece $\text{beersLiked} \rightarrow \text{manf}$

Chei ale Relațiilor

- K este o *supercheie* pentru relația R dacă K determină funcțional toate atributele din R .
- K este o *cheie* pentru R dacă K este o supercheie, și nici un subset al lui K nu este supercheie.

Exemplu: Supercheie

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} este o supercheie deoarece aceste atribute determină împreună toate celelalte atribute.
 - name → addr favBeer
 - beersLiked → manf

Exemplu: Cheie

- $\{name, beersLiked\}$ este o cheie
deoarece nici $\{name\}$ nici $\{beersLiked\}$
nu este o supercheie.
 - $name$ NU \rightarrow manf; $beersLiked$ NU \rightarrow addr.
- Nu există alte chei, dar există mai multe
superchei.
 - Orice superset al $\{name, beersLiked\}$.

Deducerea DF

- Fiind date DF $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$, presupunem că dorim să verificăm dacă o DF $Y \rightarrow B$ este valabilă în una din relațiile ce satisfac DF date.
- Exemplu: Dacă $A \rightarrow B$ și $B \rightarrow C$ este valabilă, atunci $A \rightarrow C$ este valabilă.
- Este important pentru o bună proiectare a schemelor de relație.

Testul pentru Inferență

- Pentru a testa dacă $Y \rightarrow B$, se pornește de la presupunerea că două tuple au aceleași valori pentru toate attributele ce formează Y .

← →

Y

0000000. . . 0

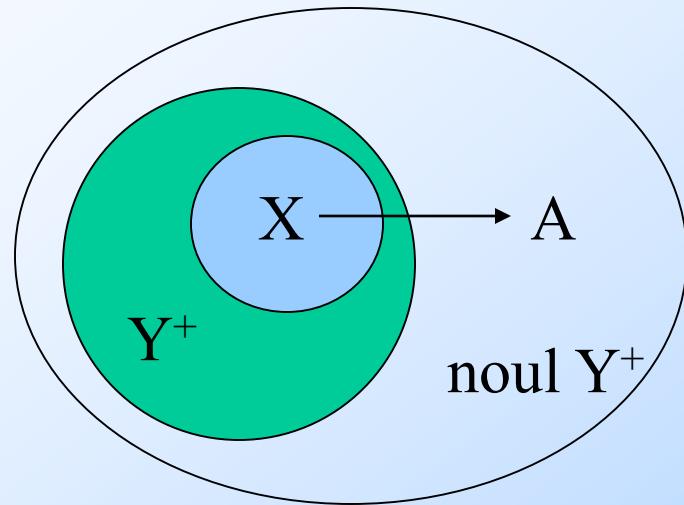
00000?? . . . ?

Testul pentru Inferență

- Se folosesc DF date pentru a infera că aceste tuple trebuie să aibă aceleasi valori corespunzătoare altor attribute.
 - Dacă B este unul din aceste attribute, atunci $Y \rightarrow B$ este adevarat.
 - În caz contrar, cele două tuple, formează o relație formată din două tuple ce dovedește că $Y \rightarrow B$ nu este deductibilă din DF date.

Testul de Închidere Tranzitivă

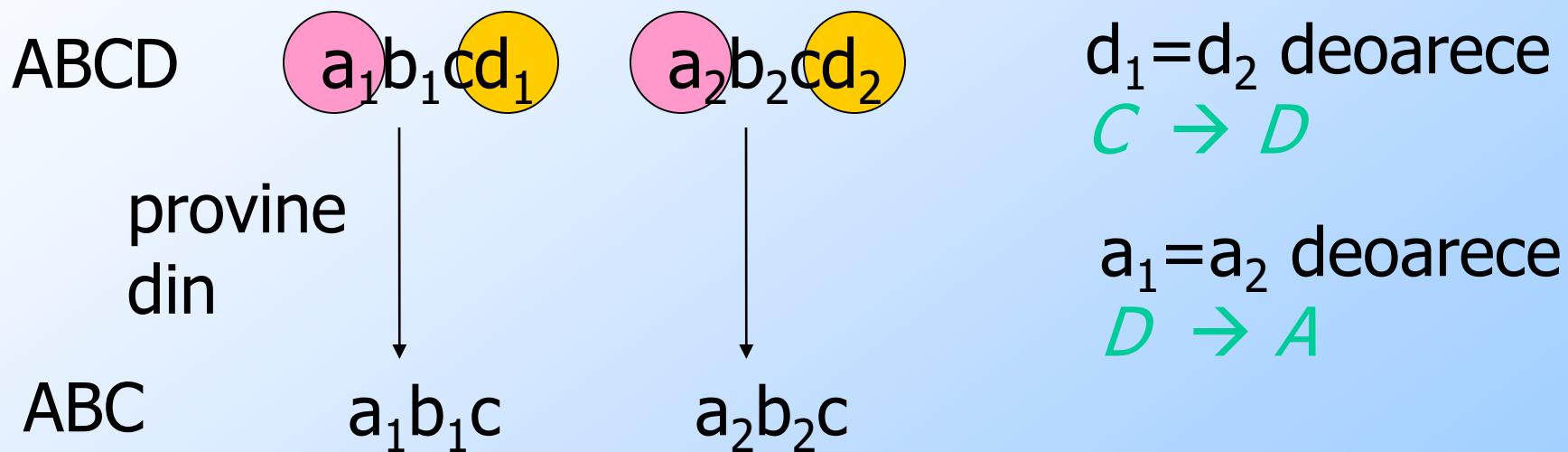
- O cale mai simplă de a testa inferența este calculul *închiderii* lui Y , notate Y^+ .
- Ipoteza: $Y^+ = Y$.
- Inducția: Se caută o DF ce are în stânga X ce este un subset al Y^+ curent. Dacă DF este $X \rightarrow A$, se adaugă A la Y^+ .



Găsirea tuturor DF-le

- Motivație: “normalizarea,” procesul prin care se “sperate” schema unei relații în două sau mai multe scheme.
- Exemplu: $ABCD$ cu DF-le $AB \rightarrow C$, $C \rightarrow D$, și $D \rightarrow A$.
 - Se decompune în ABC , AD . Ce DF-le sunt valabile în ABC ?
 - Nu numai $AB \rightarrow C$, ci de asemenea $C \rightarrow A$!

Explicație



Așadar, tuplele din proiecție cu același C au același A;
 $C \rightarrow A$.

Regula de Bază

1. Se începe cu DF date și se caută toate DF *netriviale* ce pot fi deduse din DF date.
 - Netrivial = partea dreapta nu este conținută în partea stânga.
2. Se păstrează doar acele DF ce implică attributele schemei după proiecție.

Algoritm

1. Pentru fiecare set de atribute X , se calculează X^+ .
2. Se adaugă $X \rightarrow A$ pentru toate A în $X^+ - X$.
3. Se elimină $XY \rightarrow A$ dacă $X \rightarrow A$.
 - Deoarece $XY \rightarrow A$ se deduce din $X \rightarrow A$ prin orice proiecție.
4. În final, se folosesc doar DF ce implică atributele proiecției.

Trucuri

- Nu e nevoie să se calculeze înciderea tranzitivă pentru setul vid sau pentru setul format din toate atributele.
- Dacă $X^+ = \text{toate atrbutele}$, este înciderea tranzitivă a oricărui superset al lui X .

Exemplu: Deducerea DF

- ABC cu DF $A \rightarrow B$ și $B \rightarrow C$. Se deduce DF AC .
 - $A^+ = ABC$; conduce la $A \rightarrow B$, $A \rightarrow C$.
 - Nu este nevoie să se calculeze AB^+ sau AC^+ .
 - $B^+ = BC$; conduce la $B \rightarrow C$.
 - $C^+ = C$; nu conduce la nimic.
 - $BC^+ = BC$; nu conduce la nimic.

Axiomele lui Armstrong

□ Reflexivitate (trivială)

Dacă $Y \subseteq X$ atunci $X \rightarrow Y$

Exemplu: (Nume, Adresa) \rightarrow Nume

□ Augmentare (trivială)

Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y \cup Z$

Exemplu: (Nume,Produs) \rightarrow (Adresa,Produs)

□ Tranzitivitate

Dacă $X \rightarrow Y$ și $Y \rightarrow Z$ atunci $X \rightarrow Z$

Proiectarea Schemei Relaționale

- Obiectivul proiectării schemei relaționale este evitarea anomaliiilor și a redundanței.
 - *Anomalia la Adăugare*: nu se poate adăuga un fapt valid până ce nu se adaugă o tuplă
 - *Anomalia la Modificare*: una din aparițiile unui fapt se modifică, dar nu toate aparițiile
 - *Anomalia la Ștergere*: un fapt valid se pierde când se șterge o tuplă

Exemplu de Proiectare proastă

Drinkers(name, addr, beersLiked, manf, favBeer)

| name | addr | beersLiked | manf | favBeer |
|---------|------------|------------|--------|-----------|
| Janeway | Voyager | Bud | A.B. | WickedAle |
| Janeway | ??? | WickedAle | Pete's | ??? |
| Spock | Enterprise | Bud | ??? | Bud |

Datele sunt redundante, deoarece fiecare poziție **???** poate fi exprimată folosind DF $\text{name} \rightarrow \text{addr}$ favBeer și $\text{beersLiked} \rightarrow \text{manf}$.

Anomaliile la proiectare greșită

| name | addr | beersLiked | manf | favBeer |
|---------|------------|------------|--------|-----------|
| Janeway | Voyager | Bud | A.B. | WickedAle |
| Janeway | Voyager | WickedAle | Pete's | WickedAle |
| Spock | Enterprise | Bud | A.B. | Bud |

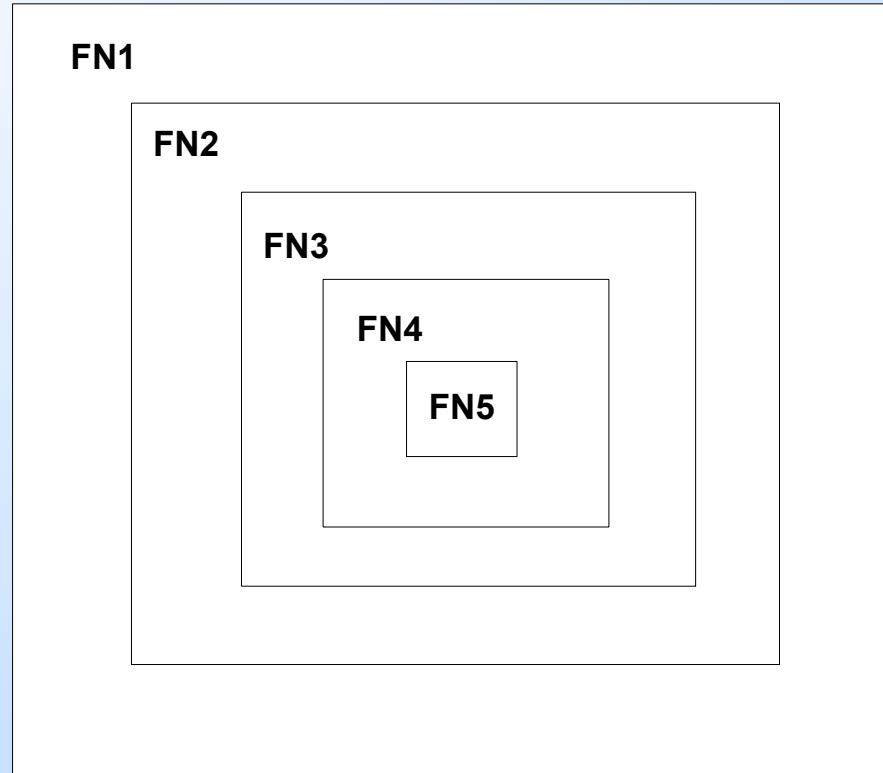
Anomalia la Adăugare : nu se poate menționa existența unei mărci noi de bere până ce nu se cunoaște o persoană căreia să îi placă berea.

Anomalia la Modificare : dacă Janeway își schimbă adresa la *Intrepid*, se va ține minte să se modifice fiecare tuplă unde apare ea?

Anomalia la Ștergere: dacă nimănuți nu-i place Bud, se pierde Faptul că Anheuser-Busch fabrică Bud.

Forme Normale

Relatii nenormalizate



FN1

Definitie: O relație R este în **FN1** dacă și numai dacă toate atributele sale iau valori atomice.

- În modelul relațional, prin definiție, toate atributele unei relații R sunt definite pe domenii ce conțin elemente atomice.
- Toate tuplele unei relații au același număr de câmpuri și aceeași dimensiune.

FN1

Drinkers(name, addr, beersLiked, manf,
favBeer)

- Relația **Drinkers** este în FN1, ar fi nenormalizată dacă s-ar permite ca atributul **addr** să înregistreze (Localitate, Strada, Numar)

FN2

Definiție: O relație R este în **FN2** dacă este în FN1 și orice atribut neprim este total dependent față de orice cheie a relației.

- FN2 rezolvă problemele cauzate de dependențele între atribute prime și cele neprime.

Drinkers(name, addr, beersLiked, manf, favBeer)

cheia este {name, beersLiked}

DF-le: name → addr favBeer, beersLiked → manf

Drinkers NU este în FN2.

FN3

Definiție: O relație R este în **FN3** dacă este în FN2 și nici un atribut neprim nu este dependent față de un alt atribut neprim.

- Elimină anomaliile la adăugare, modificare, ștergere.

Beers(name, manf, addr_manf)

Beers este în FN2, dar NU este în FN3 (name este cheie, există DF $\text{manf} \rightarrow \text{addr_manf}$).

Forma Normală Boyce-Codd

Definiție: Se spune că relația R este în **FNBC** dacă totdeauna când $X \rightarrow Y$ este o DF netrivială valabilă pentru R , X este o supercheie.

- Ne reamintim: *netrivial* înseamnă Y nu este conținut în X .
- Ne reamintim, o *supercheie* este orice superset al unei chei (nu neapărat unul anume).

Exemplu 1

Drinkers(name, addr, beersLiked, manf, favBeer)

DF-le: $\text{name} \rightarrow \text{addr favBeer}$, $\text{beersLiked} \rightarrow \text{manf}$

- Singura cheie este {name, beersLiked}.
- În fiecare DF, partea stânga *nu este* o supercheie.
- Oricare din DF-le demonstrează că *Drinkers* nu este în FNBC

Exemplu 2

Beers(name, manf, manfAddr)

DF-le: name → manf, manf → manfAddr

- Singura cheie este {name} .
- Name → manf nu violează FNBC, dar manf → manfAddr da.

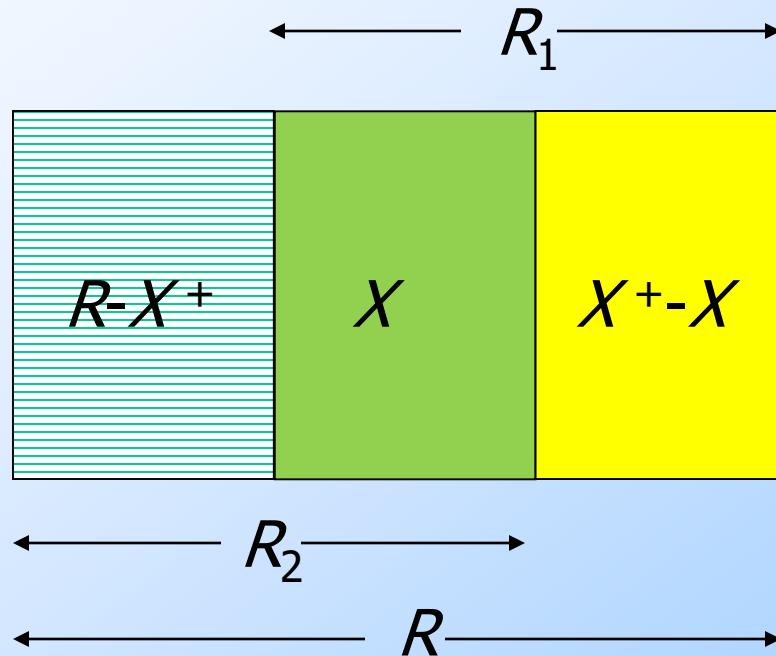
Descompunerea în BCNF

- Se dă: relația R și DF-le F .
- Se caută DF-le $X \rightarrow Y$ ce violează FNBC.
 - Dacă există DF în F ce violează FNBC, atunci R necesită descompunere.
- Se calculează X^+ .
 - Nu trebuie să conțină toate atributele, altfel X este o supercheie.

Descompunerea lui R Folosind $X \rightarrow Y$

- Se înlocuiește R cu relațiile ce au schemele:
 1. $R_1 = X^+$.
 2. $R_2 = R - (X^+ - X)$.
- *Se deduc* DF-le date F pentru cele două noi relații.

Descompunerea lui R



Exemplu: Descompunere FNBC

Drinkers(name, addr, beersLiked, manf, favBeer)

$F = \text{name} \rightarrow \text{addr}, \quad \text{name} \rightarrow \text{favBeer},$
 $\text{beersLiked} \rightarrow \text{manf}$

- Se alege una din DF ce violează FNBC
 $\text{name} \rightarrow \text{addr}$.
- Se obține închiderea tranzitivă a părții stânga:
 $\{\text{name}\}^+ = \{\text{name}, \text{addr}, \text{favBeer}\}$.
- Se descompune relația în două relații:
 1. Drinkers1(name, addr, favBeer)
 2. Drinkers2(name, beersLiked, manf)

Exemplu: Descompunere FNBC

- Nu am terminat; trebuie verificat dacă Drinkers1 și Drinkers2 respectă FNBC.
- Deducerea DF-le este simplă.
- Pentru **Drinkers1(name, addr, favBeer)**, DF-le relevante sunt **name → addr** și **name → favBeer**.
- Așadar, **{name}** este singura cheie și Drinkers1 este în FNBC.

Exemplu: Descompunere FNBC

- Pentru $\text{Drinkers2}(\underline{\text{name}}, \underline{\text{beersLiked}}, \text{manf})$, singura DF este $\text{beersLiked} \rightarrow \text{manf}$, și singura cheie este $\{\text{name}, \text{beersLiked}\}$.
 - Ce violează FNBC.
- $\text{beersLiked}^+ = \{\text{beersLiked}, \text{manf}\}$, aşa că se descompune Drinkers2 în:
 1. $\text{Drinkers3}(\underline{\text{beersLiked}}, \text{manf})$
 2. $\text{Drinkers4}(\underline{\text{name}}, \underline{\text{beersLiked}})$

Exemplu: Descompunere FNBC

- Descompunerea rezultată pentru *Drinkers*:
 1. *Drinkers1*(name, addr, favBeer)
 2. *Drinkers3*(beersLiked, manf)
 3. *Drinkers4*(name, beersLiked)
- De notat: *Drinkers1* dă informații despre persoane, *Drinkers3* dă informații despre mărci de bere, iar *Drinkers4* dă informații despre relația de legătură între persoane și mărcile de bere cu semnificația “îi place”.

FN3 - Motivație

- Există o structură de DF-le ce cauzează probleme la descompunere.
 - $AB \rightarrow C$ și în același timp $C \rightarrow B$.
 - Exemplu: $A = \text{adresa_nr}$, $B = \text{adresa_strada}$, $C = \text{cod_postal}$.
 - Există două chei, $\{A,B\}$ și $\{A,C\}$.
 - $C \rightarrow B$ este o violare FNBC, deci trebuie descompusă în AC , BC .

DF-le Nu pot fi Forțate

- Problema este că dacă se folosesc AC și BC pentru shema BD, DF $AB \rightarrow C$ nu poate fi forțată din DF-le din aceste relații rezultate în urma descompunerii.
- Exemplu cu $A = \text{street}$, $B = \text{city}$, și $C = \text{zip}$.

DF ce Nu poate fi Forțată

| street | zip |
|--------------|-------|
| 545 Tech Sq. | 02138 |
| 545 Tech Sq. | 02139 |

| city | zip |
|-----------|-------|
| Cambridge | 02138 |
| Cambridge | 02139 |

Se face join pentru tuplele cu zip egal.

| street | city | zip |
|--------------|-----------|-------|
| 545 Tech Sq. | Cambridge | 02138 |
| 545 Tech Sq. | Cambridge | 02139 |

Deși nici una din DF-le nu a fost violată în relațiile după descompunere, DF $\text{street city} \rightarrow \text{zip}$ este violată de BD ca un întreg.

FN3 Evită Această Problemă

- A treia Formă Normală (FN3) acționează asupra condiției FNBC astfel ca descompunerea să nu fie necesară când se iese problema de mai sus.
- Un atribut este *prim* dacă este membru al unei chei.
- $X \rightarrow A$ violează FN3 dacă și numai dacă X nu este o supercheie, și A nu este prim.

Exemplu: FN3

- La exemplul precedent DF-le sunt $AB \rightarrow C$ și $C \rightarrow B$, iar cheile sunt AB și AC .
- Prin urmare A , B , și C sunt fiecare prime.
- Deși $C \rightarrow B$ violează BCNF, nu violează FN3.

Proprietățile unei Descompuneri

1. *Cuplarea fără pierdere de informații* : relațiile originale trebuie să poată fi obținute din schema descompusă, adică să se reconstruiască originalul.
2. *Conservarea DF-le* : relațiile obținute prin descompunere trebuie să satisfacă toate DF-le inițiale.

Proprietățile unei Descompuneri

- Se poate obține (1) la o descompunere FNBC.
- Se pot obține ambele (1) și (2) la o descompunere FN3.
- Nu totdeauna este posibil să se obțină ambele (1) și (2) la o descompunere FNBC.
 - Exemplul street-city-zip demonstrează acest fapt.

Testul “Pierdere de Informații”

- Dacă se aplică proiecția lui R în R_1, R_2, \dots, R_k , se poate obține R prin join?
- Orice tuplă din R poate fi obținută din fragmentele proiecției.
- Singura întrebare este: la join, este posibil să obținem ceva ce nu exista în original?

Testul “Pierdere de Informații”

- Presupunem că tupla t apare în urma join.
- Deci t este rezultatul join al unor tuple din proiecțiile lui R , câte una din fiecare R_i ce formează descompunerea.
- Se pot folosi DF-le date pentru a arăta că una din aceste tuple trebuie să fie t ?

Testul “Pierdere de Informații”

- Se începe prin a presupune $t = abc\dots$.
- Pentru fiecare i , există o tuplă s_i din R ce are a, b, c, \dots printre atributele lui R_i .
- s_i poate avea orice valori pentru alte atributе.
- Se folosește aceeași literă ca în t , dar cu un indice al componenteи.

Exemplu: Testul

- Fie $R = ABCD$, și descompunerea AB , BC , și CD .
- Fie DF-ile $C \rightarrow D$ și $B \rightarrow A$.
- Presupunem că tupla $t = abcd$ este join-ul tuplelor proiecțiilor AB , BC , CD .

Tuplele
din R pro-
iectate pe
AB, BC, CD.

Tabloul

| A | B | C | D |
|--------------------|---|-------|------------------|
| a | b | c_1 | d_1 |
| \cancel{a}_2 a | b | c | \cancel{d}_2 d |

Din $B \rightarrow A$ Din $C \rightarrow D$

S-a demonstrat că tupla
a doua trebuie să fie t .

Testul “Pierdere de Informații”

Concluzia

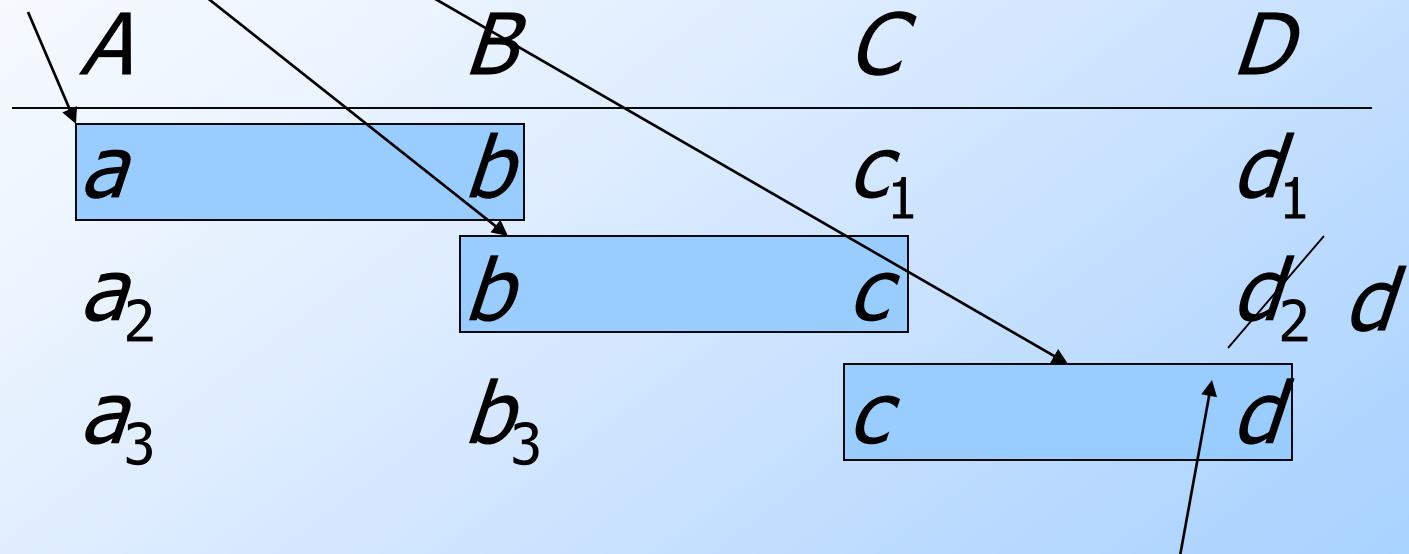
1. Dacă două tuple au aceeași parte stânga a unei DF, atunci pentru partea dreaptă se face să fie de asemenea egale.
2. Totdeauna se înlocuiește un simbol cu indice prin simbolul corespondent fără indice, dacă este posibil.
3. Atunci când se ajunge la o tuplă fără indice, se știe că orice tuplă din project-join este în original (join fără pierdere de informații).
4. În caz contrar, tabloul final este exemplu negativ.

Exemplu: Join cu Pierdere

- Aceeași relație $R = ABCD$ și aceeași descompunere.
- Considerăm singura DF $C \rightarrow D$.

Acstea proiecții
se cuplază pentru
a forma $abcd$.

Tabloul



Acstea trei tuple sunt un exemplu
 R ce arată ce înseamnă join cu
pierdere de informații. $abcd$
nu există în R , dar prin proiecție
și apoi prin cuplare se obține $abcd$.

Din $C \rightarrow D$

Descompunere FN3

Concluzii

- Se poate construi totdeauna o decompoziție formată din relații FN3 fără pierdere de informații și conservarea DF-le inițiale.
- Este nevoie de *condiții minime* pentru DF-le:
 1. Părțile dreapta să fie un singur atribut.
 2. Nici o DF nu poate fi eliminată.
 3. Nici un atribut nu poate fi eliminat din partea stânga.

Descompunere FN3

Concluzii

1. Se “sparg” părțile dreapta.
2. Se încearcă să se eliminate în mod repetat câte o DF și se verifică dacă DF-le rămase sunt echivalente cu originalul.
3. Se încearcă să se eliminate în mod repetat câte un atribut din partea stânga și se verifică dacă DF-le rezultate sunt echivalente cu originalul.

Descompunere FN3

Concluzii

- O relație pentru fiecare DF în condițiile minime.
- Schema este uniunea părților stânga și dreapta.
- Dacă nici o cheie nu este conținută într-o DF, atunci se adaugă o relație a cărei schemă este o cheie.

Exemplu: Descompunere FN3

- Relația $R = ABCD$.
- DF-le $A \rightarrow B$ și $A \rightarrow C$.
- Descompunerea: AB și AC din DF-le, plus AD pentru o cheie.

Dependențe multivalorice

Fie schema de relație $R(X, Y, Z)$, unde X , Y și Z sunt attribute simple sau compuse. Se notează x , y și z valori ale atributelor X , Y și Z (eventual și cu indici).

Definiție: Se spune că există o dependență multivalorică a atributului Z față de atributul Y sau că Y multidetermină pe Z și se notează $Y \rightarrow\rightarrow Z$, dacă pentru orice valori x_1, x_2, y, z_1, z_2 , $x_1 \neq x_2, z_1 \neq z_2$ astfel încât dacă tuplele (x_1, y, z_1) , (x_2, y, z_2) fac parte din R , atunci și tuplele (x_2, y, z_1) , (x_1, y, z_2) fac parte din R .

Dependențe multivalorice

- Din simetria definiției rezultă că dependența $Y \rightarrow \rightarrow Z$ implică existența dependenței multivalorice $Y \rightarrow \rightarrow X$.
- O consecință este că între X și Z nu poate exista o dependență funcțională.
- O dependență funcțională este și o dependență multivalorică (dependențele multivalorice generalizează dependențele funcționale).

Dependențe multivalorice

Exemplu:

AGENTII(Nume_A, Nume_F, Nume_B)

Nume_A →→ Nume_F

Nume_A →→ Nume_B

dacă și numai dacă numele oricărei femei din evidența unei agenții este asociat cu numele oricărui bărbat din evidența aceleiași agenții.

Dependențe multivalorice

Dependențele multivalorice depind de context:

APROVIZIONARE(Beneficiar, Furnizor, Produs, Pret)

Dacă nu ar exista Pret, atunci

Beneficiar →→ Furnizor și Beneficiar →→ Produs

Dar în prezența acestuia dependențele depind de legăturile Pret cu celelalte attribute.

□ Prețul unui produs este același la toți furnizorii

Beneficiar →→ Furnizor și Beneficiar →→ (Produs, Pret)

□ Prețul unui produs diferă de la furnizor la furnizor, deci există dependență funcțională (Furnizor, Produs) → Pret

Beneficiar →→ (Furnizor, Produs, Pret)

(irelevantă pentru problema modelată)

FN4

Este o generalizare a FNBC.

Definiție: O relație R este în **FN4** dacă oricare ar fi dependența multivalorică $X \rightarrow\!\!\! \rightarrow Y$, unde Y nu este un subset a lui X și $X \cup Y$ nu contine toate atributele lui R, atunci atributul X (simplu sau compus) este o cheie sau conține o cheie a lui R.

Ca și în cazul trecerii de la FN3 la FNBC, descompunerea în relații FN4 se poate face fără pierdere de informații dar nu totdeauna se conservă dependențele multivalorice și/sau funcționale.

FN4

Exemplu:

CURSURI(Profesor, Disciplina, Limba)

Se presupune că un profesor poate susține cursuri la mai multe discipline și cunoaște mai multe limbi, prin urmare un curs poate fi susținut în toate limbile cunoscute de un anumit profesor.

Profesor →→ Disciplina

Profesor →→ Limba

FN4

PROFESORI

| Profesor | Disciplina | Limba |
|----------|--------------------|----------|
| Pop | Programare | Engleza |
| Costea | Baze de Date | Engleza |
| Costea | Baze de Date | Franceza |
| Costea | Baze de Date | Germana |
| Costea | Sisteme de Operare | Engleza |
| Costea | Sisteme de Operare | Franceza |
| Costea | Sisteme de Operare | Germana |
| Popovici | Programare | Franceza |
| Popovici | Programare | Germana |

PD(Profesor, Disciplina)

PL(Profesor, Limba)

FN4

Cheia este **{Profesor, Disciplina, Limba}**.

Nu există dependențe funcționale, deci PROFESORI este în FNBC.

Deoarece nu este în FN4 există următoarele anomalii:

- Adăugare: dacă profesorul Costea acceptă să țină cursul “Programare”, trebuie introdusă câte o tuplă pentru fiecare limbă cunoscută de acesta
- Stergere: dacă se sterge singura disciplină predată de un profesor se pierd informațiile referitoare la limbile cunoscute de acesta
- Modificare: trebuie modificate mai multe tuple

FNPJ (FN5)

Dacă relația R este în FNBC și toate cheile sunt simple atunci poate fi garantată FN5. Prima condiție poate fi relaxată la FN3 (în loc de FNBC).

Demonstrație:

- dacă relația R este în FN3 și toate cheile sunt simple atunci relația R este în FNBC
- dacă relația R nu este în FN5, atunci este o DJ (dependență join)

$\{X_1, \dots, X_m\}$ a schemei ce nu este o consecință logică a cheilor candidate

FNPJ

SP

| S# | P# |
|----|----|
| S1 | P1 |
| S1 | P2 |
| S2 | P1 |

PJ

| P# | J# |
|----|----|
| P1 | J2 |
| P2 | J1 |
| P1 | J1 |

JS

| J# | S# |
|----|----|
| J2 | S1 |
| J1 | S1 |
| J1 | S2 |

SP * PJ

| S# | P# | J# |
|----|----|----|
| S1 | P1 | J2 |
| S1 | P1 | J1 |
| S1 | P2 | J1 |
| S2 | P1 | J2 |
| S2 | P1 | J1 |

S = Furnizor
P = Produs
J = Proiect

Dacă proiectul J folosește produsul P, atunci toți furnizorii produsului P îl furnizează proiectului J

SPJ

| S# | P# | J# |
|----|----|----|
| S1 | P1 | J2 |
| S1 | P2 | J1 |
| S2 | P1 | J1 |
| S1 | P1 | J1 |

*

Dependență join: (S1, P1) în SP, (P1, J1) în PJ, (J1, S1) în JS implică (S1, P1, J1) în SPJ.

Modelul Entitate-Relație

Diagramme E/R
Entitate “Weak”

Obținerea Schemei Relaționale din Diagramme E/R

Modelul E/R

- Scop: permite schițarea elementelor ce vor fi păstrate în BD (ajută la definirea schemei BD indiferent de modelul de date ales).
 - Include constrângeri asupra datelor, dar nu include operații asupra datelor.
- Ce se obține: desen numit *diagramă entitate-relație*.
(mai exact relație de legătură)
- Ulterior: se convertește diagrama E/R într-o schemă de BD.

Modelarea E/R

- Proiectarea BD face parte din cadrul mai general de proiectare a unui sistem informatic destinat unei companii.
- Există persoane din conducerea companiei care au cunoștințe despre afacerea respectivă, dar nu pot spune ce să conțină BD.
- Schițarea componentelor cheie reprezintă o modalitate eficientă pentru dezvoltarea unei BD funcționale.

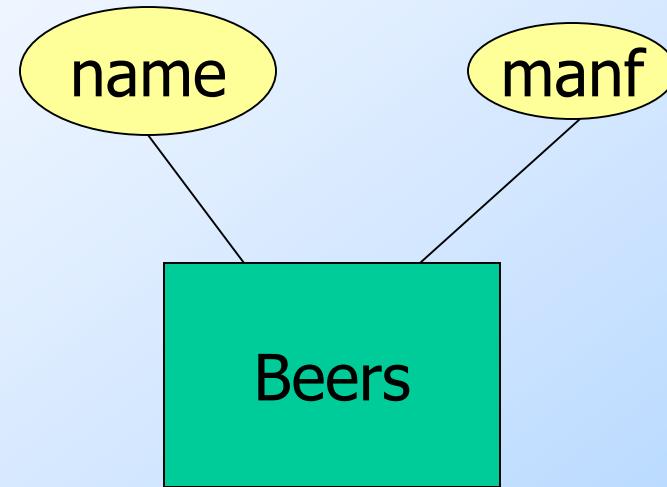
Seturi de Entități

- *Entitate* = un “lucru” sau obiect din lumea reală.
- *Set de Entități* = colecție de entități similare.
 - Asemănător cu o clasă din programarea orientată obiect.
- *Atribut* = proprietate a unui set de entități.
 - Atributele sunt valori simple, ca de exemplu întregi sau siruri de caractere;
 - NU sunt valori compuse ca de exemplu structuri, seturi, etc.

Diagrame E/R

- Într-o diagramă entitate-relație :
 - Entitate = dreptunghi.
 - Atribut = oval, cu o linie către un dreptunghi ce reprezintă setul entitate corespunzător.

Exemplu:

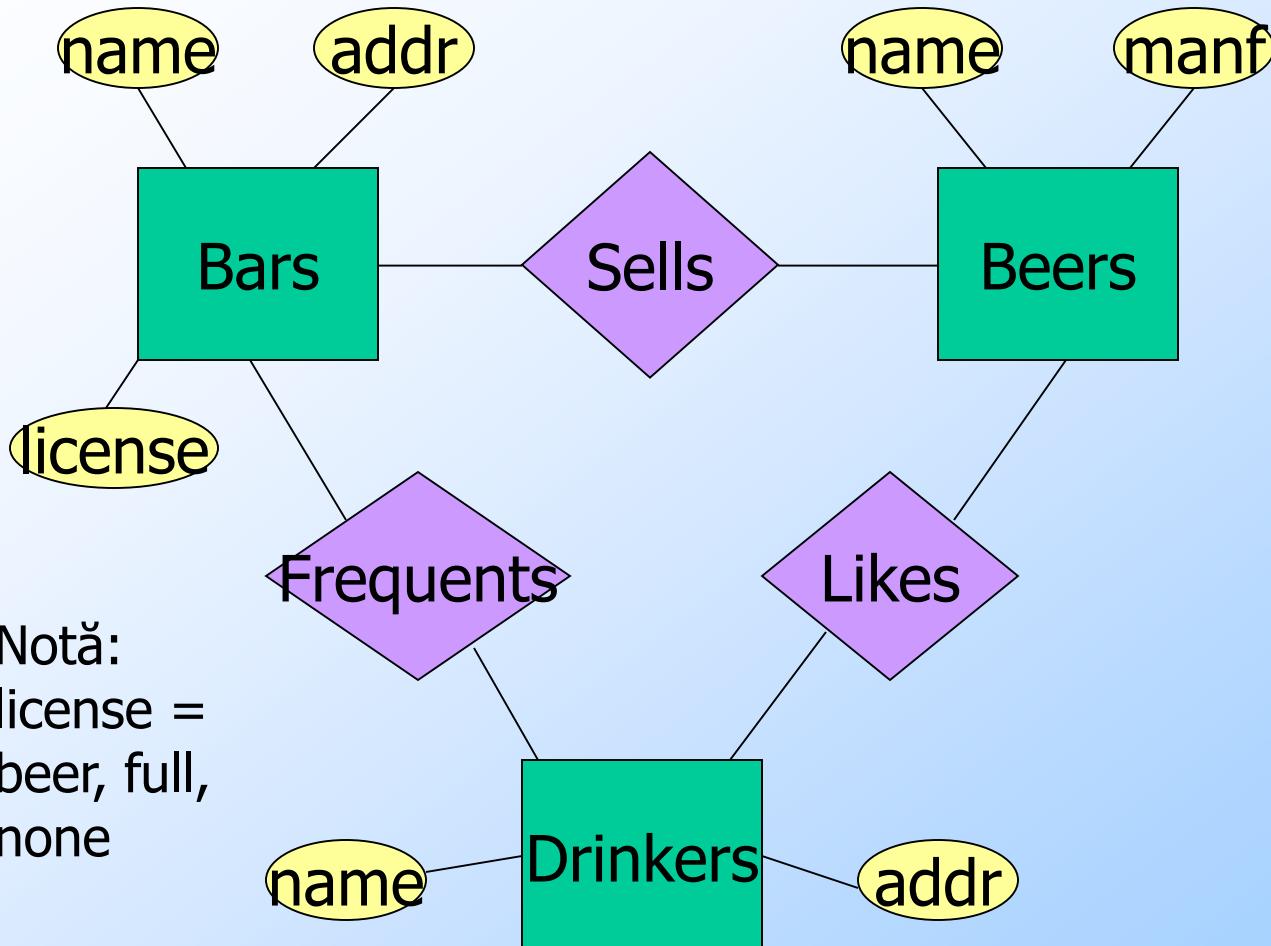


- Setul entitate **Beers** are două atribută, **name** și **manf** ("manufacturer").
- Fiecare entitate **Beers** are valori pentru aceste două atribută, de exemplu (Bud, Anheuser-Busch)

Relații de legătură

- O relație de legătură conectează două sau mai multe seturi de entități.
- Este reprezentată printr-un romb, cu linii la fiecare din seturile de entități implicate.

Exemplu: Relații de legătură



"Bars Sell some Beers".

"Drinkers Like some Beers".

"Drinkers Frequent some Bars".

Notă:
license =
beer, full,
none

Setul Relație de legătură

- “Valoarea” curentă a unui *set entitate* reprezintă setul entităților aparținătoare.
 - Exemplu: setul tuturor barurilor din BD.
- “Valoarea” unei relații de legătură este un *set relație de legătură*, un set de tuple cu o componentă din fiecare set entitate ce intră în legătură.

Exemplu: Set Relație de legătură

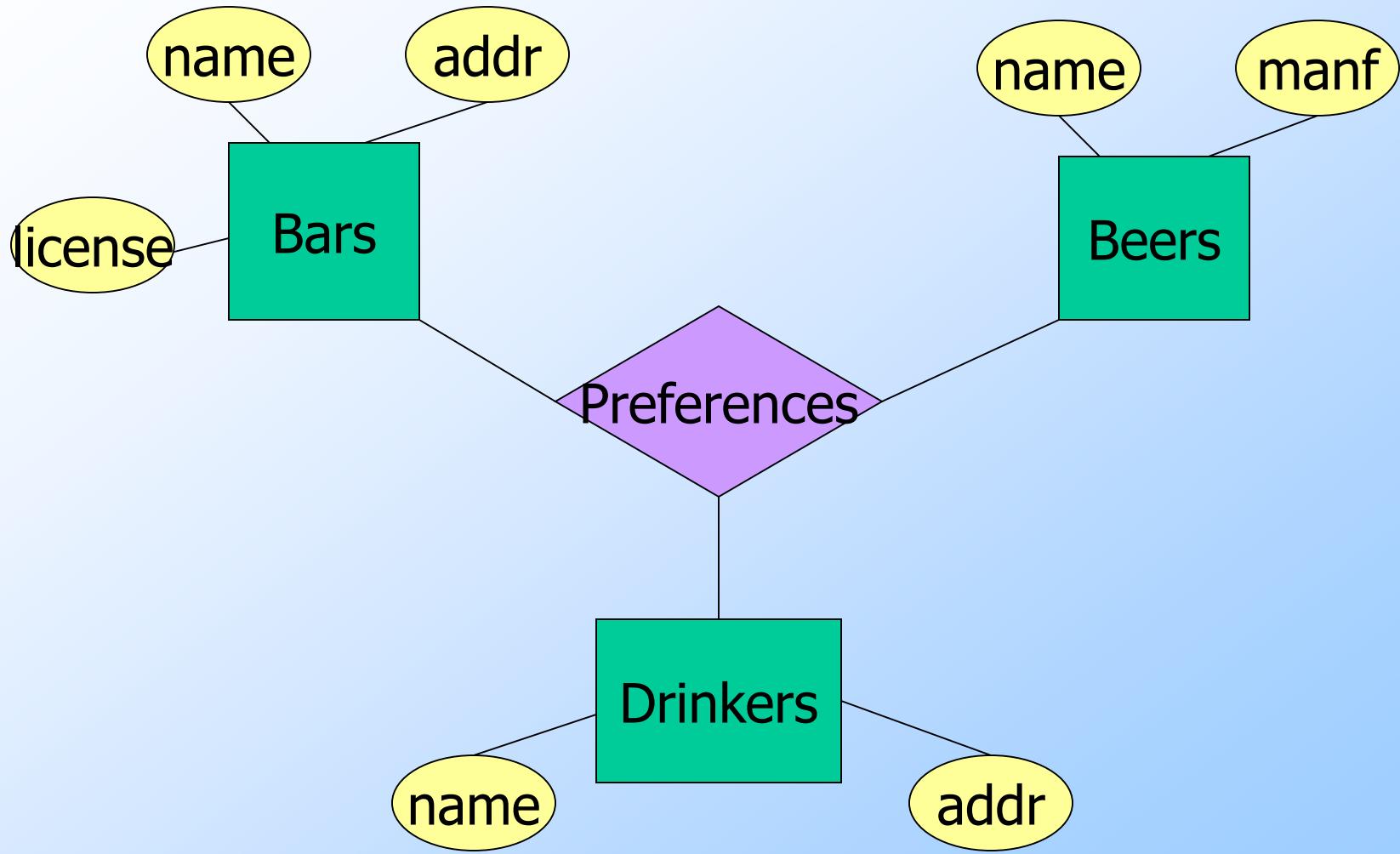
□ Pentru relația de legătură **Sells**, se poate imagina setul relație de legătură următor:

| Bar | Beer |
|-----------|------------|
| Joe's Bar | Bud |
| Joe's Bar | Miller |
| Sue's Bar | Bud |
| Sue's Bar | Pete's Ale |
| Sue's Bar | Bud Lite |

Relații de legătură Multiple

- Uneori este nevoie ca o relație de legătură să conecteze mai mult de două seturi entitate.
- Să presupunem că persoanele (drinkers) vor consuma anumite beri (beers) la anumite baruri (bars).
 - Cele trei relații de legătură binare **Likes**, **Sells** și **Frequents** nu permit să se facă această distincție.
 - Este nevoie de o relație de legătură cu 3 capete.

Exemplu: Relație de legătură Multiplă



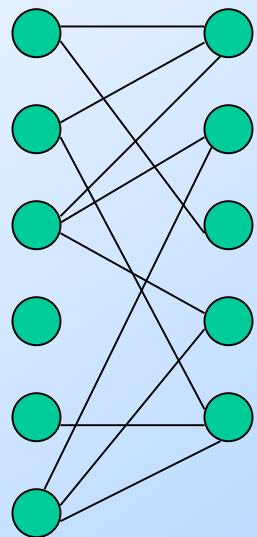
Un Set Tipic Relație de Legătură Multiplă

| Bar | Drinker | Beer |
|-----------|---------|------------|
| Joe's Bar | Ann | Miller |
| Sue's Bar | Ann | Bud |
| Sue's Bar | Ann | Pete's Ale |
| Joe's Bar | Bob | Bud |
| Joe's Bar | Bob | Miller |
| Joe's Bar | Cal | Miller |
| Sue's Bar | Cal | Bud Lite |

Relații de Legătură M:N

- Tinta: relații de legătură **binare**, cum este **Sells** între **Bars** și **Beers**.
- Într-o relație de legătură M:N (*many-many*), o entitate din oricare set poate fi conectată cu mai multe entități din celălalt set.
 - De exemplu, un bar vinde mai multe sortimente de bere; o marcă de bere este vândută în mai multe baruri.

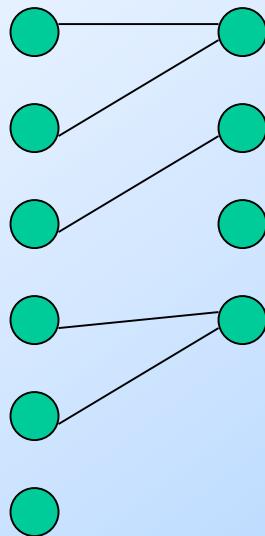
M:N



Relații de Legătură N:1

- Unele relații de legătură binare sunt **N:1** (*many-one*).
- Fiecare entitate din primul set se conectează cu cel mult o entitate din setul al doilea.
- O entitate din setul al doilea se poate conecta cu zero, una, sau mai multe entități din primul set.

N:1



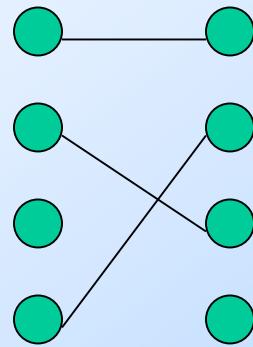
Exemplu: Relație de Legătură N:1

- **Favorite**, de la Drinkers la Beers este “many-one”.
- O persoană are cel mult 1 marcă de bere favorită.
- Dar o marcă de bere poate fi berea favorită a mai multor persoane, inclusiv zero.

Relații de Legătură 1:1

- Într-o **relație de legătură 1:1 (one-one)**, fiecare entitate a unuia din seturile entități este pus în relație cu cel mult o entitate a celuilalt set.
- **Exemplu:** Relația de legătură **Best-seller** între seturile entități **Manfs** ("manufacturer") și **Beers**.
 - O marcă de bere nu poate fi fabricată de mai mult de un fabricant, și nici un fabricant nu poate avea mai mult de o marcă de bere cel mai bine vândută (presupunând că nu sunt două sau mai multe la egalitate pe primul loc).

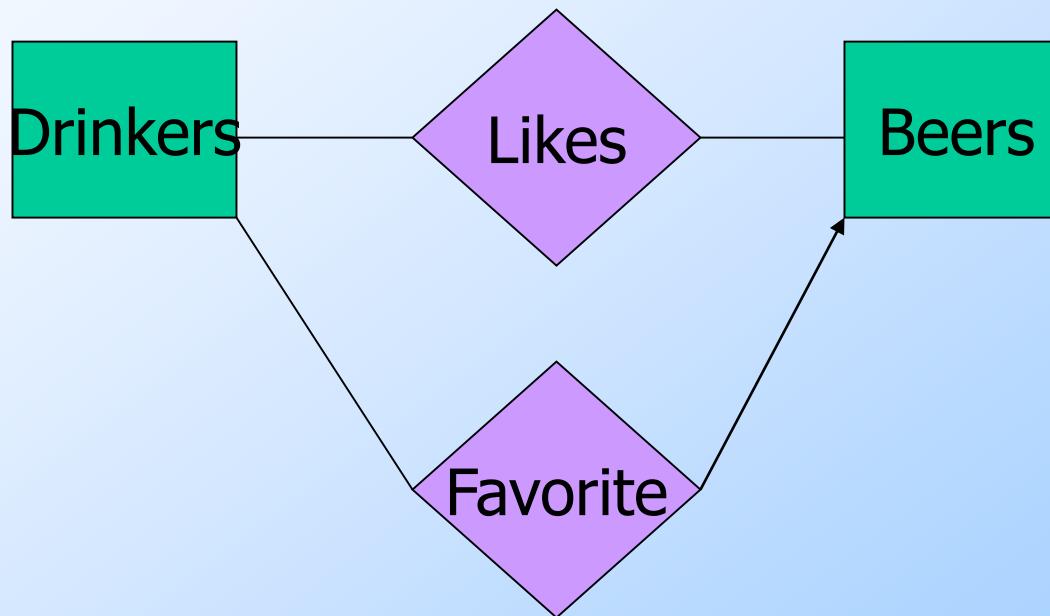
1:1



Reprezentarea capătului relației de legătură

- Se folosește săgeata cu vârf spre capătul “1” al relației de legătură “many-one”.
 - De ținut minte: Seamănă cu dependența funcțională.
- O relație de legătură 1:1 se reprezintă cu săgeți la ambele capete (spre ambele seturi entitate).
- **Săgeată rotunjită** = “exact 1,” de exemplu, când fiecare entitate din primul set este pusă în relație cu exact 1 entitate din setul al doilea.

Exemplu: Relație de Legătură N:1



Notă: două relații de legătură conectează aceleasi seturi entitate, dar sunt diferite.

Exemplu: Relație de Legătură 1:1

- Se consideră **Best-seller** între **Manfs** și **Beers**.
- Anumite mărci de bere nu sunt cele mai bine vândute mărci de bere ale nici unui fabricant, prin urmare vârful săgeții spre **Manfs** nu poate fi rotunjit.
- Dar un fabricant de bere obligatoriu are 1 marcă de bere cel mai bine vândută.

Diagrama E/R



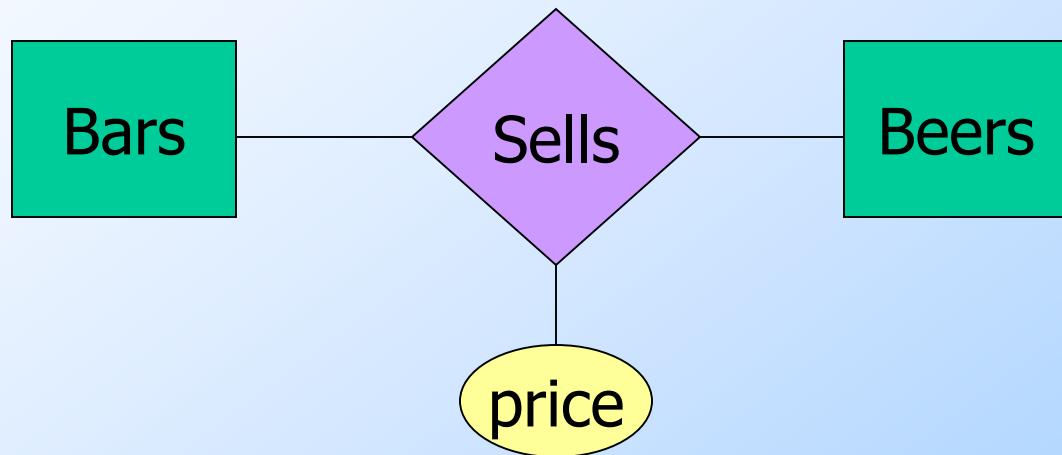
O bere este “best-seller” pentru 0 sau 1 “manufacturer”.

Un “manufacturer” are exact 1 “best-seller”.

Atribute pentru Relații de Legătură

- Uneori este folositor să se atașeze un atribut unei relații de legătură.
- Trebuie gândit la acest atribut ca la o proprietate a tuplelor din setul relație de legătură.

Exemplu: Atribut pentru o Relație de Legătură

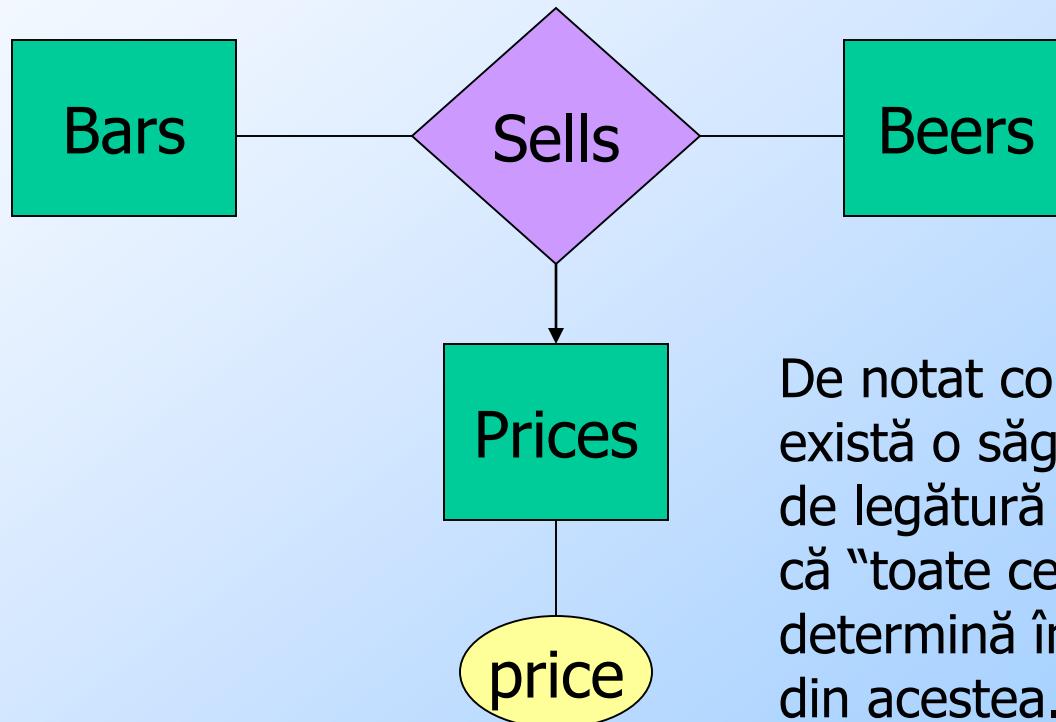


Preț este o funcție ce depinde atât de bar cât și de bere, nu de una din ele.

Diagrame Echivalente fără Atribute pentru Relațiile de Legătură

- Se crează un set entitate pentru a reprezenta valorile atributului.
- Acel set entitate participă în relația de legătură.

Exemplu: Eliminarea unui Atribut de la o Relație de Legătură



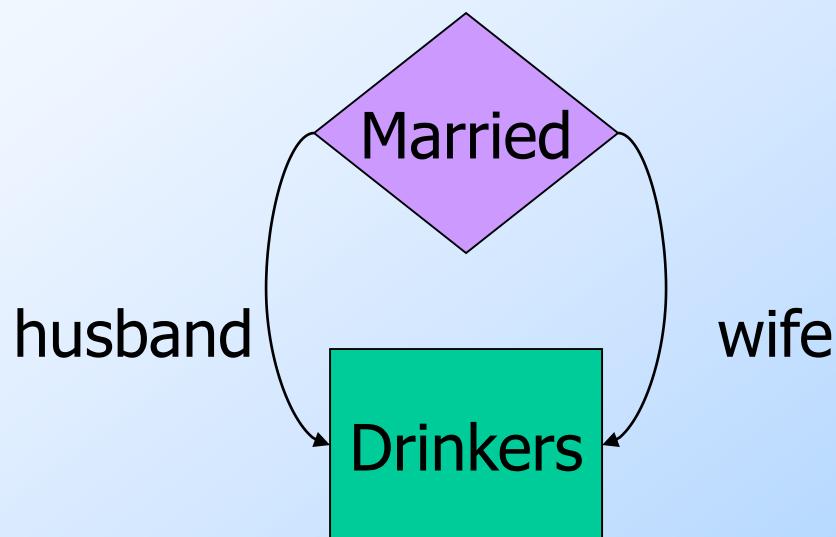
De notat convenția: dacă există o săgeată de la relația de legătură multiplă, înseamnă că “toate celelalte seturi entitate determină împreună unul singur din acestea.”

Roluri

- Uneori un set entitate apare de mai multe ori într-o relație de legătură.
- Se etichetează capătele relației de legătură către setul entitate cu nume ce semnifică *roluri*.

Exemplu: Roluri

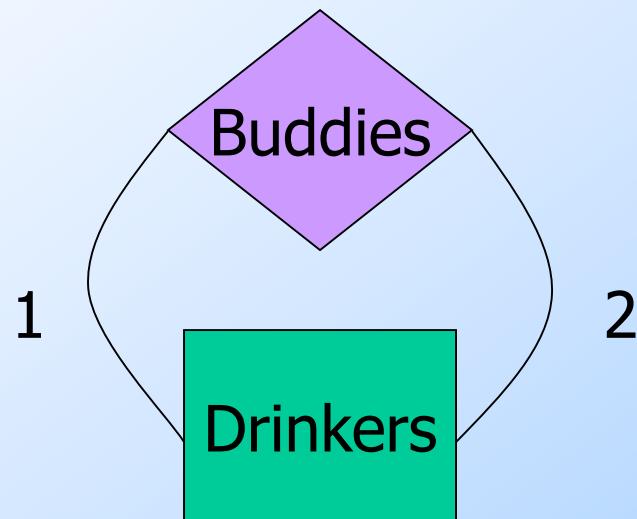
Setul Relație de Legătură



| Husband | Wife |
|---------|------|
| Bob | Ann |
| Joe | Sue |
| ... | ... |

Exemplu: Roluri

Setul Relație de Legătură



| Buddy1 | Buddy2 |
|--------|--------|
| Bob | Ann |
| Joe | Sue |
| Ann | Bob |
| Joe | Moe |
| ... | ... |

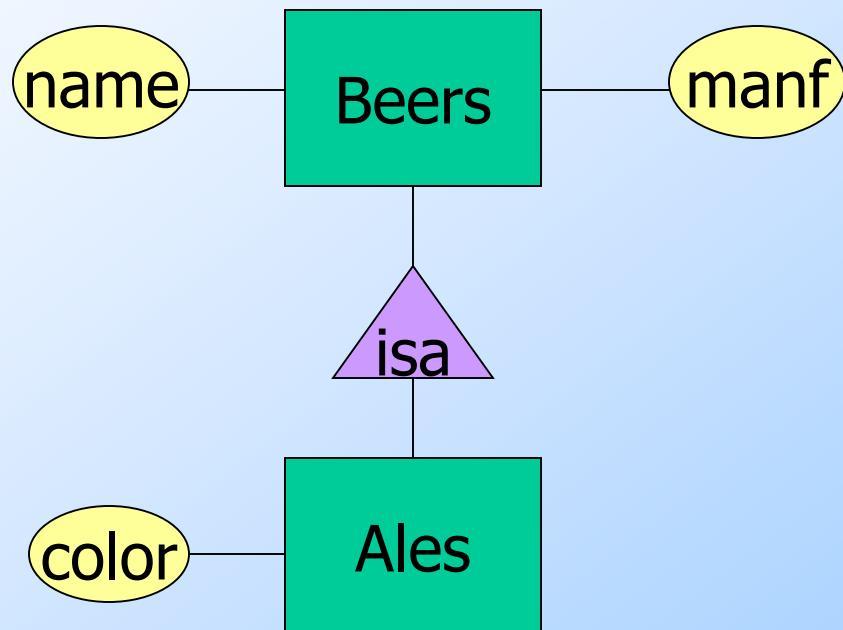
Subclase

- **Subclasă** = caz special = mai puține entități = mai multe proprietăți.
- **Exemplu:** “Ale” reprezintă un sortiment de bere.
 - Nu fiecare bere este un “ale”, dar unele sunt.
 - Să presupunem că în plus față de toate **proprietățile** (attribute și relații de legătură) lui “beers”, “ales” au atributul “color”.

Subclase în Diagrame E/R

- Să presupunem că subclasele formează un arbore.
 - Adică nu există moștenire multiplă.
- Triunghiuri “Isa” indică relația de legătură subclasă.
 - Pointează spre superclasă.

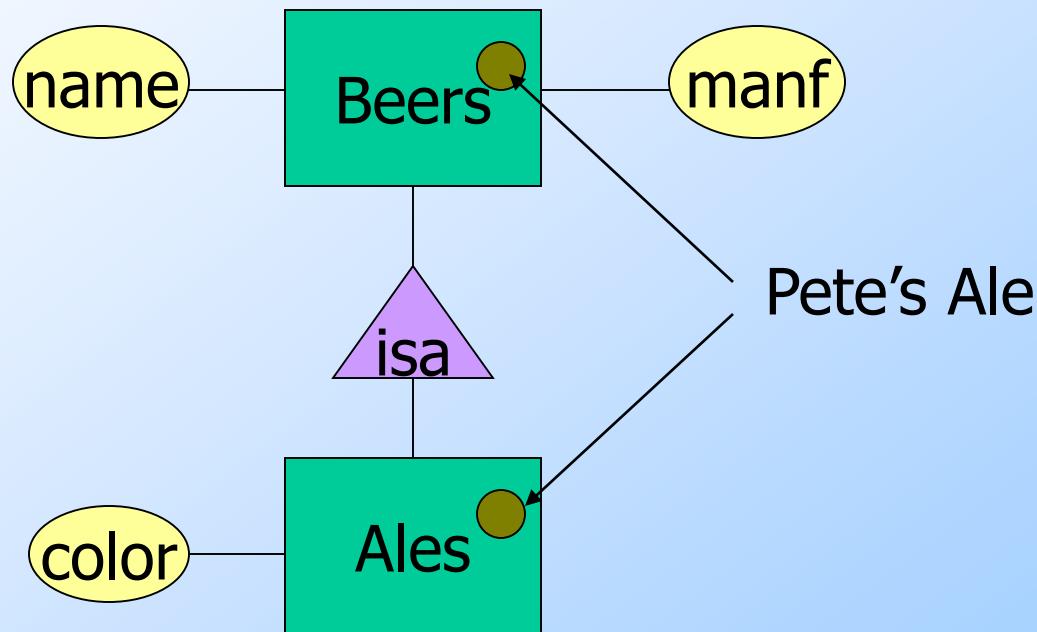
Exemplu: Subclase



E/R Vs. Subclase Orientate-Obiect

- În OO, obiectele sunt într-o clasă și numai una.
 - Subclasele moștenesc de la superclase.
- În contrast, entitățile E/R au *reprezentante* în toate subclasele la care aparțin.
 - **Regulă:** dacă entitatea E este reprezentată într-o subclasă, atunci E este reprezentată în superclasă (și recursiv înapoi în arbore).

Exemplu: Reprezentante ale Entităților



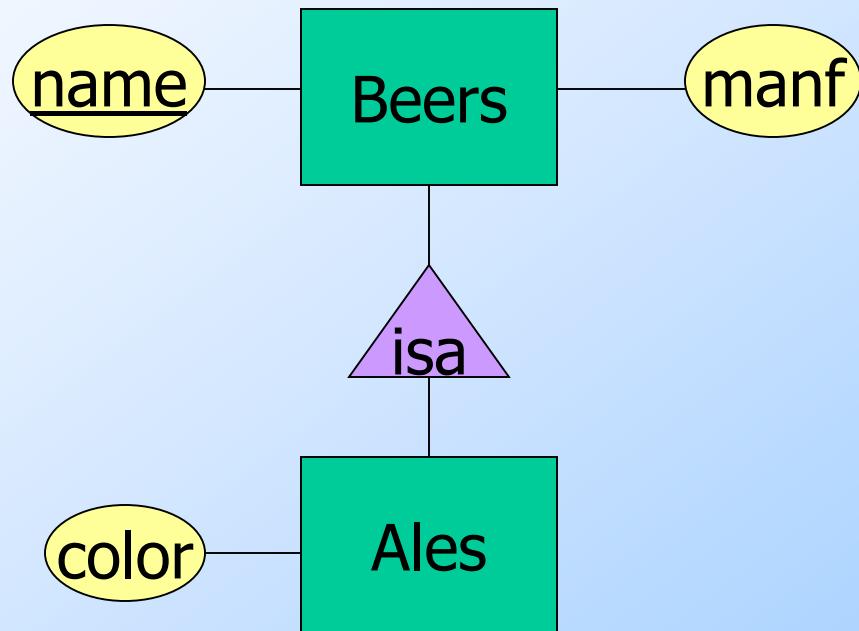
Chei

- O *cheie* reprezintă o mulțime de attribute pentru un set entitate astfel încât două entități din set nu pot avea aceleași valori pentru toate attributele cheii.
 - Se poate ca două entități să aibă anumite valori egale, pentru attributele cheii, dar nu toate.
 - Trebuie desemnată o cheie pentru fiecare set entitate.

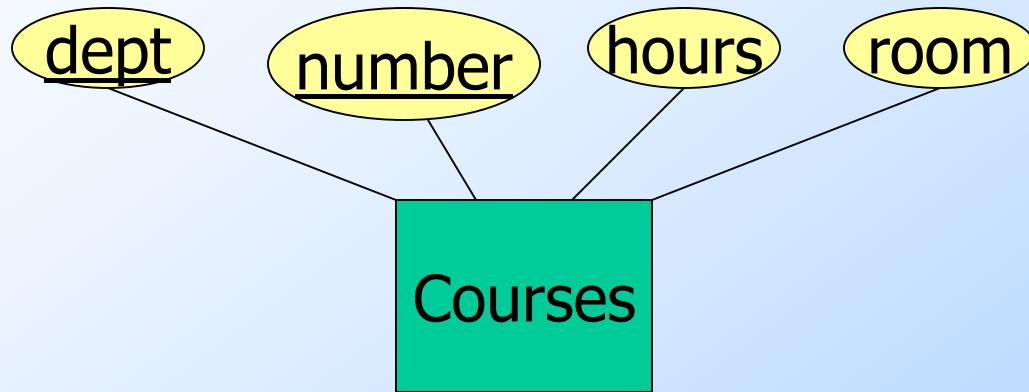
Chei În Diagrame E/R

- Se subliniază atributul(-ele) cheie.
- Într-o ierarhie “Isa”, doar setul entitate rădăcină are cheie, și aceasta servește ca și cheie pentru toate entitățile din ierarhie.

Exemplu: name este Cheie pentru Beers



Exemplu: Cheie Multi-atribut



- De notat că **hours** și **room** pot fi alese pentru cheie, dar este nevoie de o singură cheie.

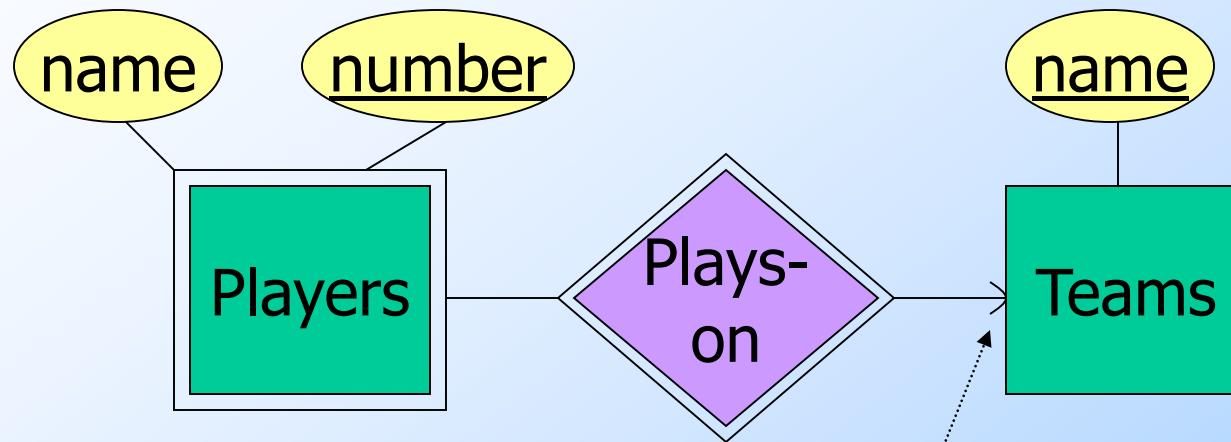
Seturi Entitate “Weak”

- Ocazional, entitățile unui set entitate necesită “ajutor” pentru a le identifica în mod unic.
- Setul entitate E se spune că este “*weak*” dacă pentru a identifica unic entitățile din E , este nevoie de a urmări relații de legătură “many-one” plecând de la E și să se includă cheia entităților din seturile entitate conectate.

Exemplu: Set Entitate “Weak”

- “**name**” este aproape o cheie pentru jucători de fotbal, dar pot exista doi jucători cu același “name”.
- “**number**” este un lucru ce nu poate fi cheie, deoarece jucători din două echipe pot avea același “number” (numărul de pe tricou).
- Dar numărul de pe tricou, împreună cu numele echipei pus în legătură de “**Plays-on**” ar trebui să fie unic.

Diagrama E/R Set Entitate “Weak”



Notă: trebuie rotunjit deoarece fiecare jucător are nevoie de o echipă pentru cheie.

Linie dublă pentru rombul relației de legătură N:1 *susținătoare*.

Linie dublă pentru dreptunghiul setului entitate “weak”.

Reguli pentru Setul Entitate “Weak”

- Un set entitate “weak” are una sau mai multe relații de legătură N:1 către alte seturi entitate (susținătoare).
 - Nu fiecare relație de legătură N:1 ce pleacă de la un set entitate “weak” este “susținător”.
 - Relațiile de legătură susținătoare trebuie să aibă vârful de săgeată rotunjit (entitatea de la capătul “one” este garantată).

Reguli pentru Setul Entitate “Weak”

- Cheia pentru un set entitate weak este format din attributele subliniate și din cheile seturilor entitate susținătoare.
 - De exemplu **number** (player) și **name** (team) este o cheie pentru **Players** în exemplul precedent.

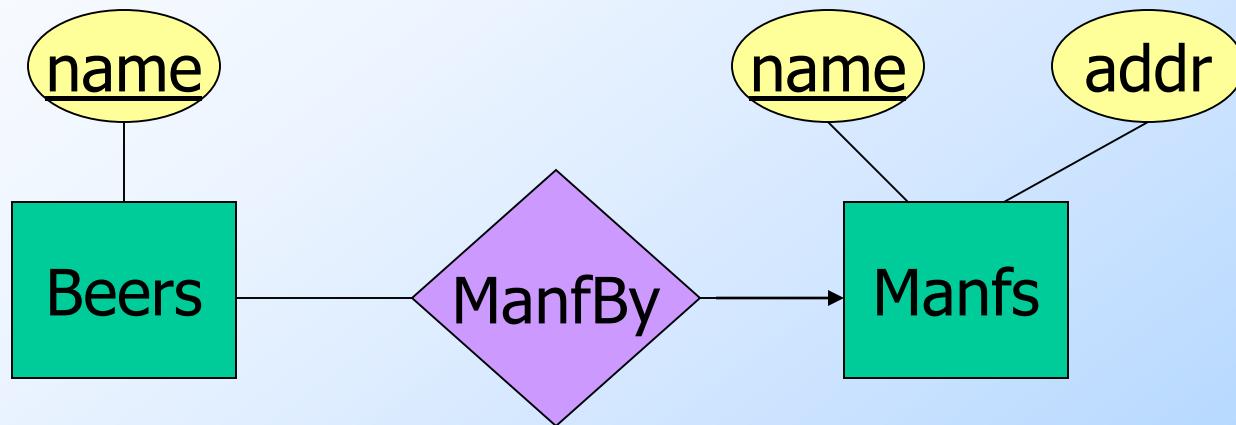
Tehnici de Proiectare

1. A se evita redundanta.
2. A se limita utilizarea de seturi entitate “weak”.
3. Să nu se folosească un set entitate dacă se poate folosi un atribut.

Evitarea Redundanței

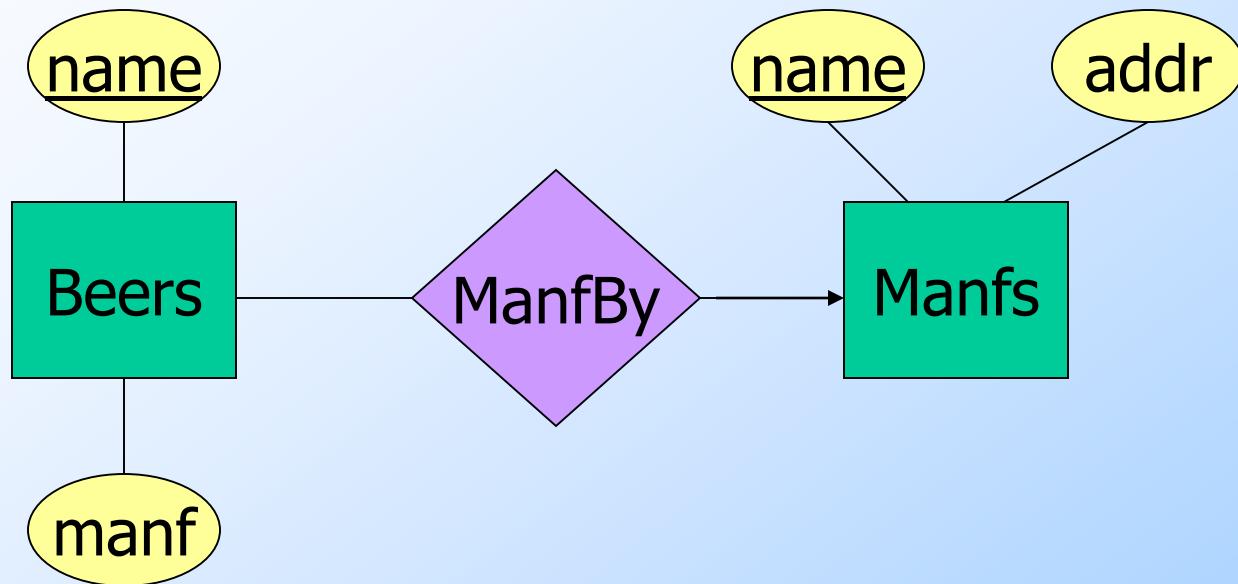
- *Redundanță* = specificarea aceluiași lucru în două (sau mai multe) moduri diferite.
- Se îrosește spațiu și (mai important) se încurajează inconsistenta.
 - Două reprezentări ale aceluiași fapt devin inconsistente dacă se modifică una și se lasă cealaltă nemodificată.
 - De reamintit anomaliiile de la DF-le.

Exemplu: Bun



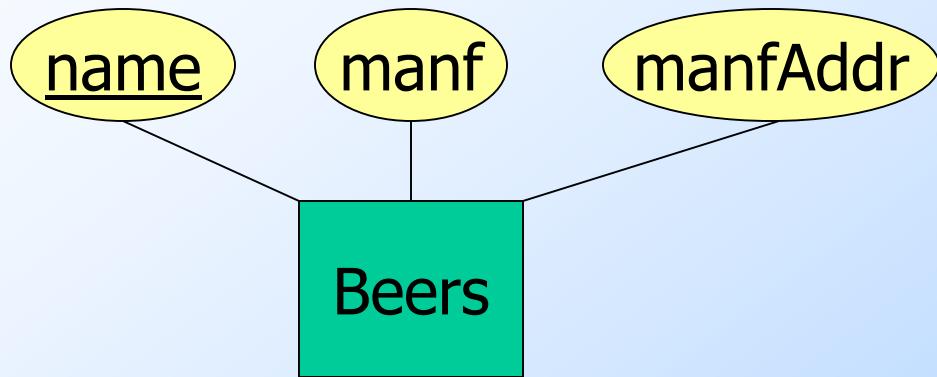
Această proiectare specifică adresa fiecărui fabricant exact o singură dată.

Exemplu: Rău



Această proiectare precizează fabricantul unei mărci de bere de două ori: prima dată ca un atribut și a doua oară ca o entitate pusă în legătură.

Exemplu: Rău



Această proiectare repetă adresa fabricantului pentru fiecare bere și adresa se pierde dacă temporar nu există mărci de bere pentru un fabricant.

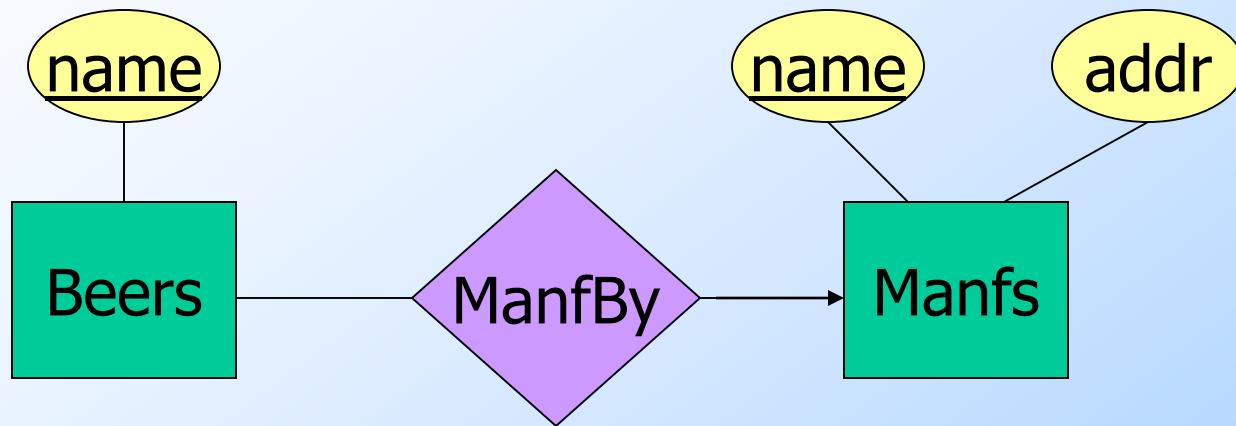
Seturi Entitate Versus Atribute

- Un set entitate ar trebui să satisfacă cel puțin una din următoarele condiții:
 - Reprezintă mai mult decât numele unui lucru; are cel puțin un atribut ce nu este cheie.

sau

- Este capătul “many” într-o relație de legătură “many-one” sau “many-many”.

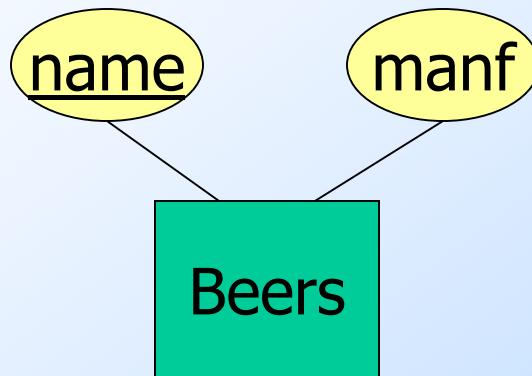
Exemplu: Bun



Manfs merită să fie un set entitate deoarece are atributul ce nu este cheie **addr**.

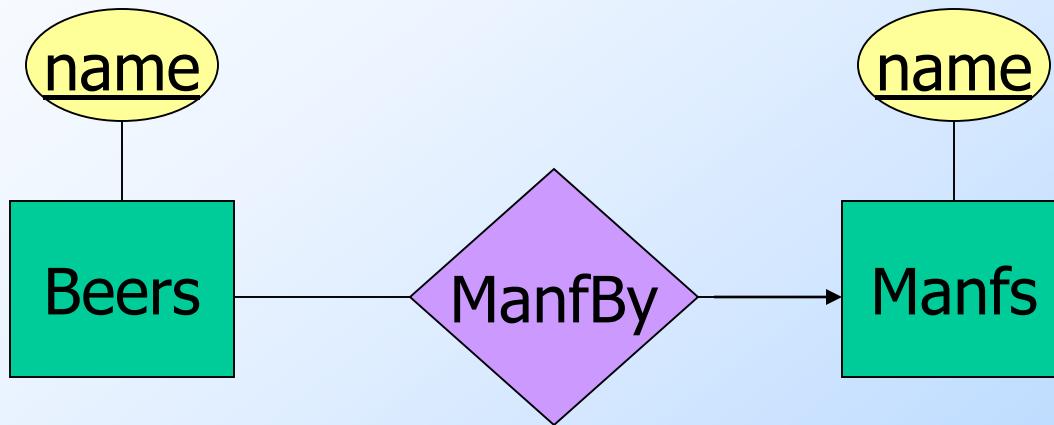
Beers merită să fie un set entitate deoarece este la capătul "many" al relației de legătură "many-one" **ManfBy**.

Exemplu: Bun



Nu este nevoie să se definească “manufacturer” ca un set entitate, deoarece nu se înregistrează nimic despre fabricanți în afara numelui lor.

Exemplu: Rău



Deoarece “manufacturer” nu este altceva decât un nume, și nu este la capătul “many” al nici unei relații de legătură, nu ar trebui să fie set entitate.

Să nu se exagereze cu folosirea Seturilor Entitate “Weak”

- Proiectanții de BD începători au adesea îndoieri în a alege cheia.
 - Aceștia marchează seturile entitate ca fiind “weak”, susținute de toate seturile entitate încunjurătoare cu care sunt legate.
- În realitate, se crează ID-uri unice pentru seturi entitate.
 - Exemple: CNP pentru o persoană, număr serial pentru un calculator etc.

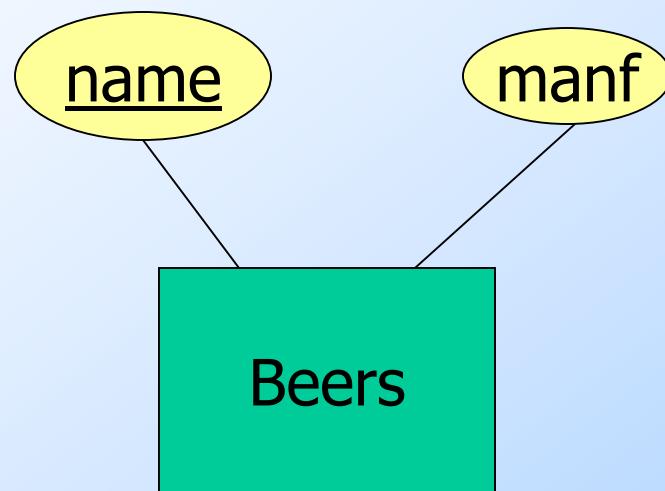
Când este nevoie de Seturi Entitate “Weak”?

- Motivul obișnuit este acela că nu există o autoritate globală capabilă să creeze ID-uri unice.
- **Exemplu:** este puțin probabil ca să existe o înțelegere între echipele de fotbal din întregă lume pentru a atribui un număr unic fiecărui jucător.

Obținerea Schemei Relaționale din Diagrame E/R

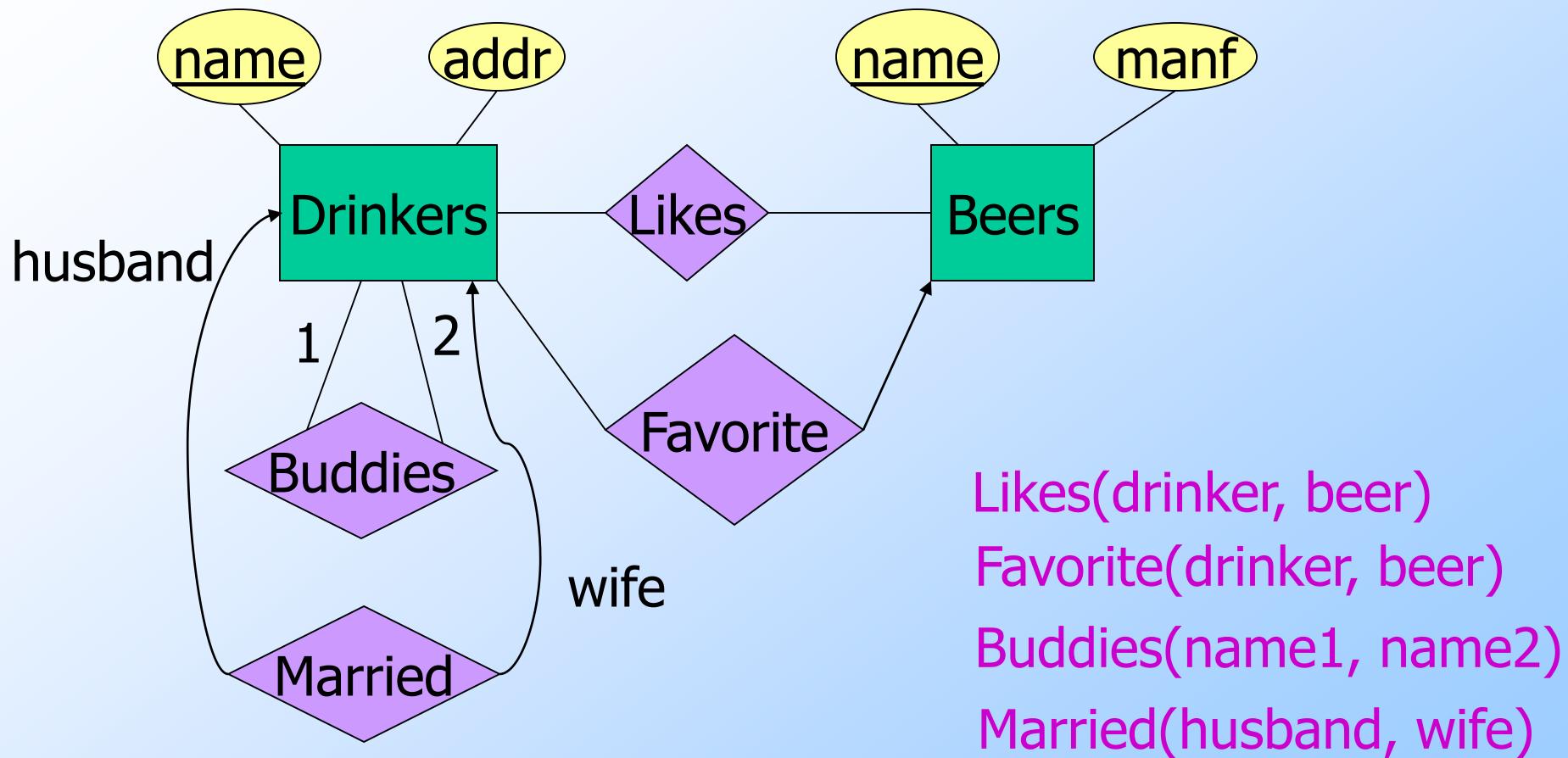
- Set entitate -> relație.
 - Atribute -> atrbute.
- Relații de legătură -> relații cu următoarele atribut:
 - Cheile seturilor entitate conectate.
 - Atributele relației de legătură (dacă există).

Set Entitate -> Relație



Relația: Beers(name, manf)

Relație de Legătură -> Relație



Combinarea Relațiilor

- Se poate combina într-o relație:
 1. Relația pentru un set entitate E
 2. Relațiile pentru relațiile de legătură “many-one” pentru care E este capătul “many.”
- Exemplu: `Drinkers(name, addr)` și `Favorite(drinker, beer)` se combină pentru a forma `Drinker1(name, addr, favBeer)`.

Risc implicat de Relații de Legătură “Many-Many”

- Combinarea Drinkers cu Likes este greșit. Conduce la redundanță:

| name | addr | beer |
|-------|-----------|--------|
| Sally | 123 Maple | Bud |
| Sally | 123 Maple | Miller |

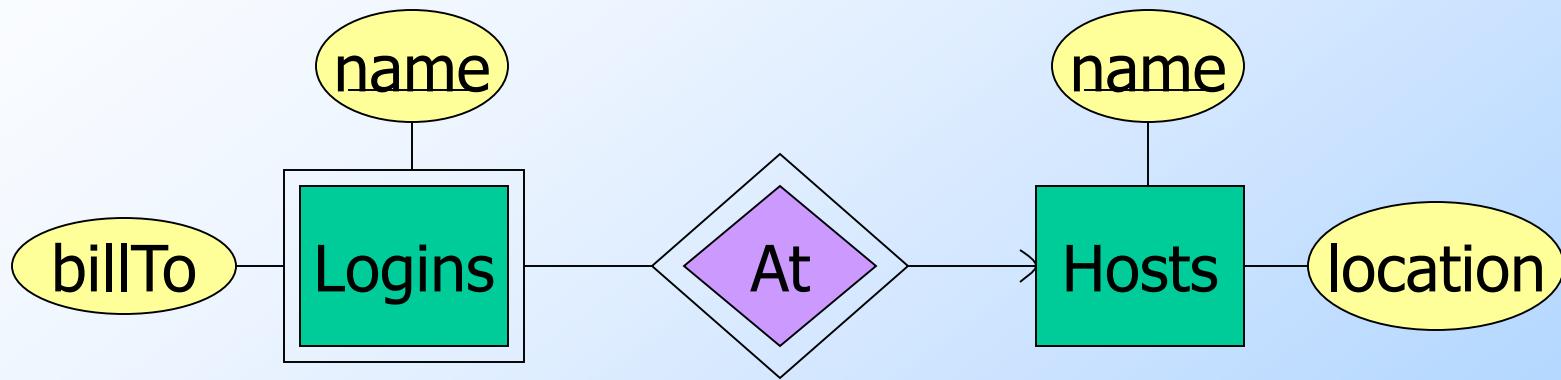
Redundanță



Seturi Entitate “Weak”

- Relația pentru un set entitate “weak” trebuie să includă atributele ce compun cheia (inclusiv cele ce aparțin altor seturi entitate), plus propriile attribute noncheie.
- O relație de legătură susținătoare este redundantă și nu conduce la nici o relație (cu excepție când are attribute proprii).

Exemplu: Set Entitate "Weak" -> Relație



Hosts(hostName, location)

Logins(loginName, hostName, billTo)

~~At(loginName, hostName, hostName2)~~

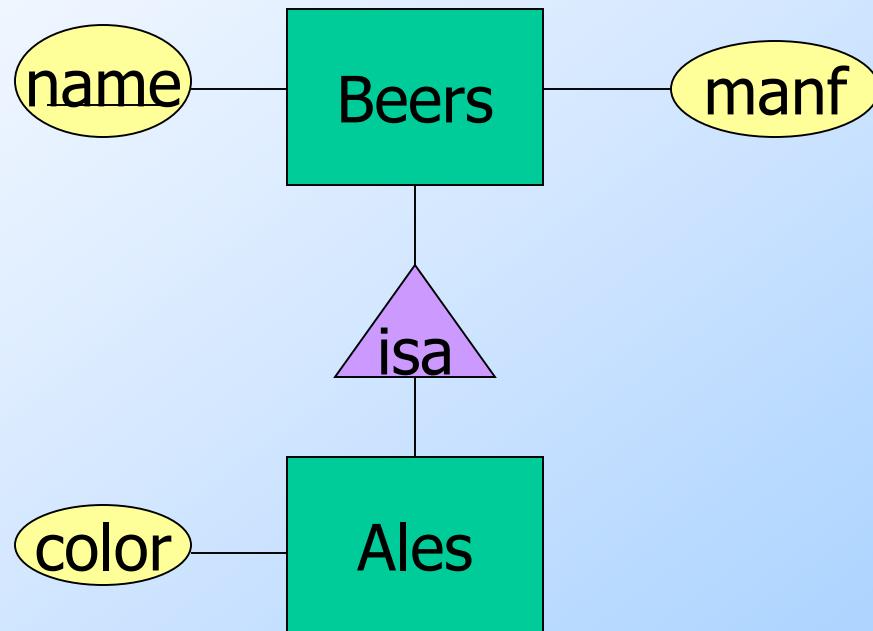
At devine parte
a Logins

Trebuie să fie același

Subclase: Trei Abordări

1. *Orientat-Obiect*: O relație per subset de subclase, cu toate atributele relevante.
2. *Folosire null-uri*: O relație; entitățile au NULL în atributele ce nu le aparțin.
3. *Stil E/R*: O relație pentru fiecare subclasă:
 - Atribut (-e) cheie.
 - Atribute ale acelei subclase.

Exemplu: Subclasă -> Relații



Orientat-Obiect

| name | manf |
|-------|----------------|
| Bud | Anheuser-Busch |
| Beers | |

| name | manf | color |
|------------|--------|-------|
| Summerbrew | Pete's | dark |
| Ales | | |

Prinde bine la interogări de genul:

Găsește culoarea “ales” produsă de “Pete’s”.

Stil E/R

| name | manf |
|------------|----------------|
| Bud | Anheuser-Busch |
| Summerbrew | Pete's |

Beers

| name | color |
|------------|-------|
| Summerbrew | dark |

Ales

Prinde bine la interogări de genul:

Găsește toate "beers" (inclusiv "ales") produse de "Pete's".

Folosire Null-uri

| name | manf | color |
|-------------------|--------------------------|--------------|
| Bud Summerbrew | Anheuser-Busch Pete's | NULL dark |
| Beers | | |

Salvează spațiu dacă NU sunt multe atribute ce conțin NULL.

Administrare BD

Stocare (BD fizică Oracle 12c)
Backup/Recovery

Administrarea spațiului BD fizică

- Serverul de baze de date Oracle gestionează automat spațiul pe disc
- Eventualele probleme sunt semnalate sub formă de alerte și sunt recomandate soluții posibile

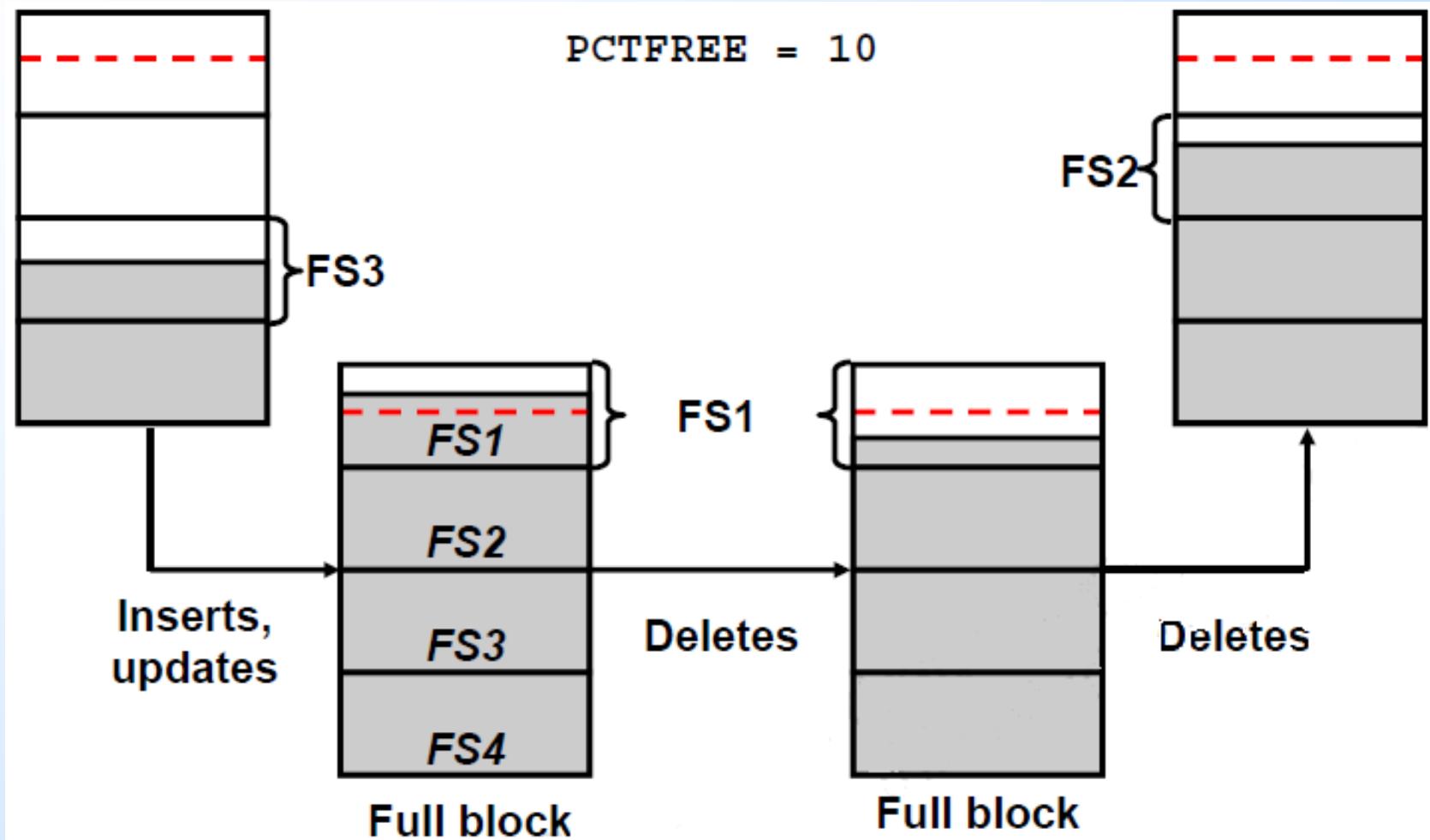
Administrarea spațiului

- Oracle Managed Files (OMF)
 - Operațiile se specifică în termeni obiecte BD în loc de fișiere pe disc
- Pentru un tablespace Oracle folosește bitmaps la gestionarea spațiului (locally managed) – Automatic Segment Space Management
- Se dă posibilitatea ca fișierele să-și mărească dimensiunea automat în funcție de gradul de umplere al tablespace-ului

Administrarea spațiului

- Gestiune proactivă a spațiului
 - Când spațiul liber scade sub un prag specificat, se declanșează alertă
- Planificarea spațiului (capacity planning)
 - Serverul BD estimează spațiul pe baza structurii tabelelor, a cardinalității tabelelor și a tendinței de creștere pe baza istoricului, raport stocat în Automatic Workload Repository (AWR)

Block Space Management



Block Space Management

- Automatic Segment Space Management
împarte fiecare bloc de date în 4 secțiuni FS_i,
 $i=1$ (75% - 100% spațiu liber) .. 4 (0% -
25% spațiu liber)
- În funcție de lungimea unei tuple inserate se
poate cunoaște dacă un bloc satisface tupla
inserată
- „full” reprezintă starea în care un bloc nu mai
este disponibil pentru operații insert

Block Space Management

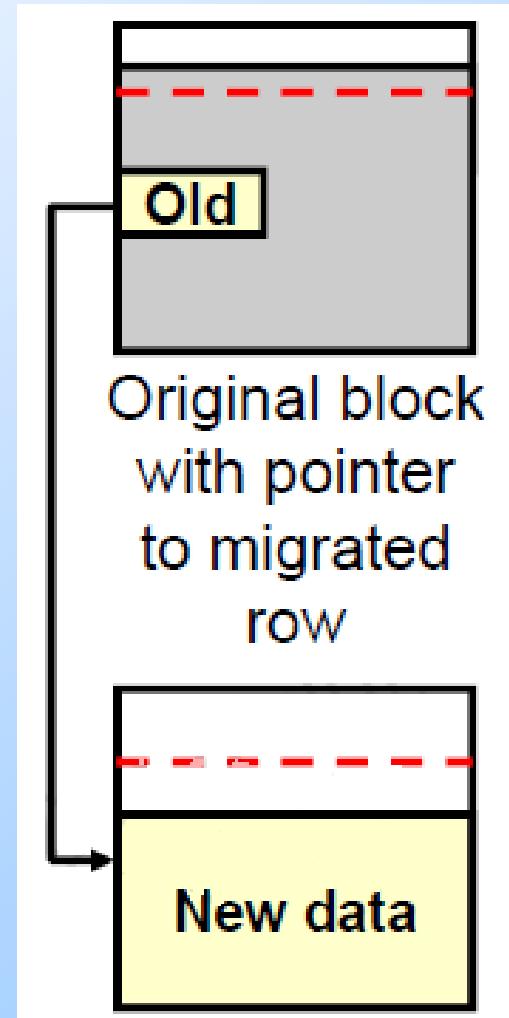
- Coloanele de tip LOB (Large Object): BLOB, CLOB, NCLOB și BFILE nu folosesc parametrul PCTFREE
- Blocurile folosite la sistemele OLTP și blocurile necomprime au parametrul PCTFREE implicit 10
- Blocurile comprimate au parametrul PCTFREE implicit 0

Block Space Management

- Oracle recomandă Oracle Database block de dimensiuni 2 KB sau 4 KB pentru Online Transaction Processing (OLTP) sau „mixed workload environments” și dimensiuni mai mari, de: 8 KB, 16 KB, sau 32 KB pentru Decision Support System (DSS).

Row chaining and migration

- Să presupunem o tabelă care are o coloană de tip VARCHAR, de exemplu Nume Persoană
- La INSERT se alocă spațiu într-un bloc oarecare
- La UPDATE dimensiunea tuplei poate să crească (initial erau 3 caractere de exemplu „Pop” și se modifică la noua valoare „Popescu”, +4 octeți) și să depășească spațiul liber din bloc
- Se alocă spațiu într-un alt bloc din același segment
- ROWID se păstrează



Row chaining and migration

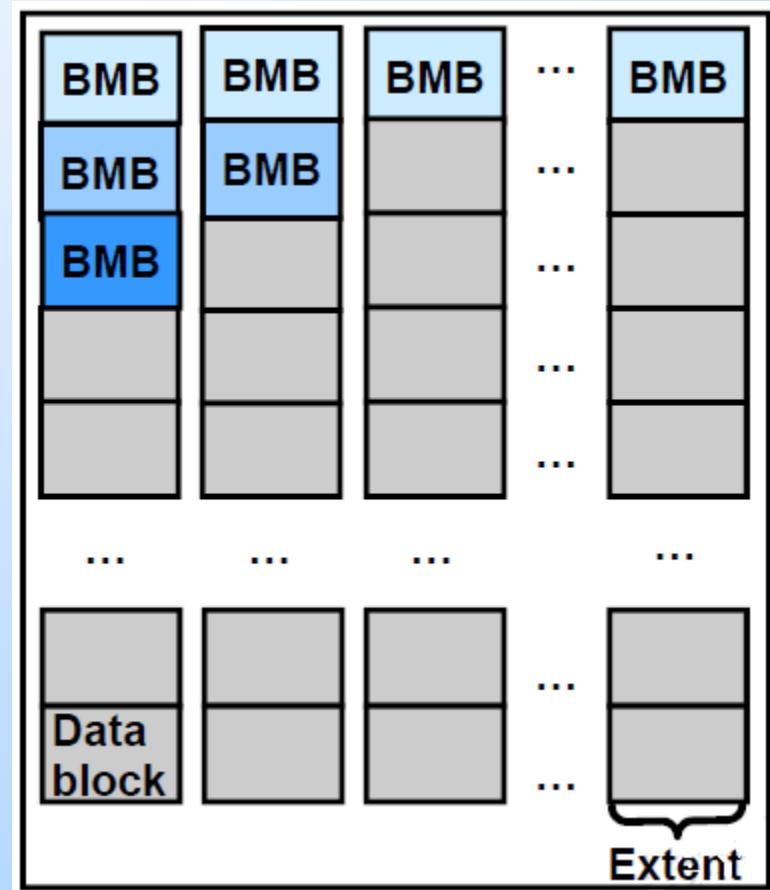
- Înlănțuirea tuplelor nu poate fi evitată la tabele cu coloane de tip LONG sau LONG RAW, în al doilea caz scenariul de bază la UPDATE este ca tupla să fie migrată în întregime într-un nou bloc dacă spațiul liber din blocul curent este ocupat, ROWID se păstrează
- Înlănțuirea tuplelor conduce la scăderea performanței, deoarece serverul BD trebuie să citească două sau mai multe blocuri

Row chaining and migration

- Segmentele ce conțin tuple înlăntuite pot fi descoperite cu Segment Advisor
- Spațiul liber fragmentat din interiorul unui bloc este fuzionat automat atunci când:
 - La INSERT sau UPDATE se încearcă să se folosească un bloc cu suficient spațiu liber
 - Spațiul liber este fragmentat a.î. datele nu pot fi scrise într-o zonă contiguă a unui bloc

Gestiunea spațiului liber din segmente

- Este urmărit prin bitmaps (Automatic Segment Space Management la creare tablespace)
- În zona de început a segmentului există blocuri bitmap (BMB) ce descriu utilizarea spațiului pentru blocurile de date



BMB sunt organizate arborescent cu maxim 3 nivele

Tipuri de segmente

- Set de extent-uri alocate pentru o structură logică de tip:
 - Tabelă și cluster
 - Index
 - Undo
 - Temporar
- Alocarea se face dinamic de serverul BD

Tipuri de segmente

□ Segment de tip Tabelă

- Fiecare tabel ne-clusterizat are datele stocate în extent-uri ale unui segment tabelă
- O tabelă partitioanată are câte un segment pentru fiecare partitie

□ Cluster

- Fiecare cluster are un segment de date, unde sunt stocate datele fiecărei tabele din cluster

Tipuri de segmente

□ Segment Index

- Fiecare index are un astfel de segment
- Pentru index partitioanat, fiecare partitie are un segment index

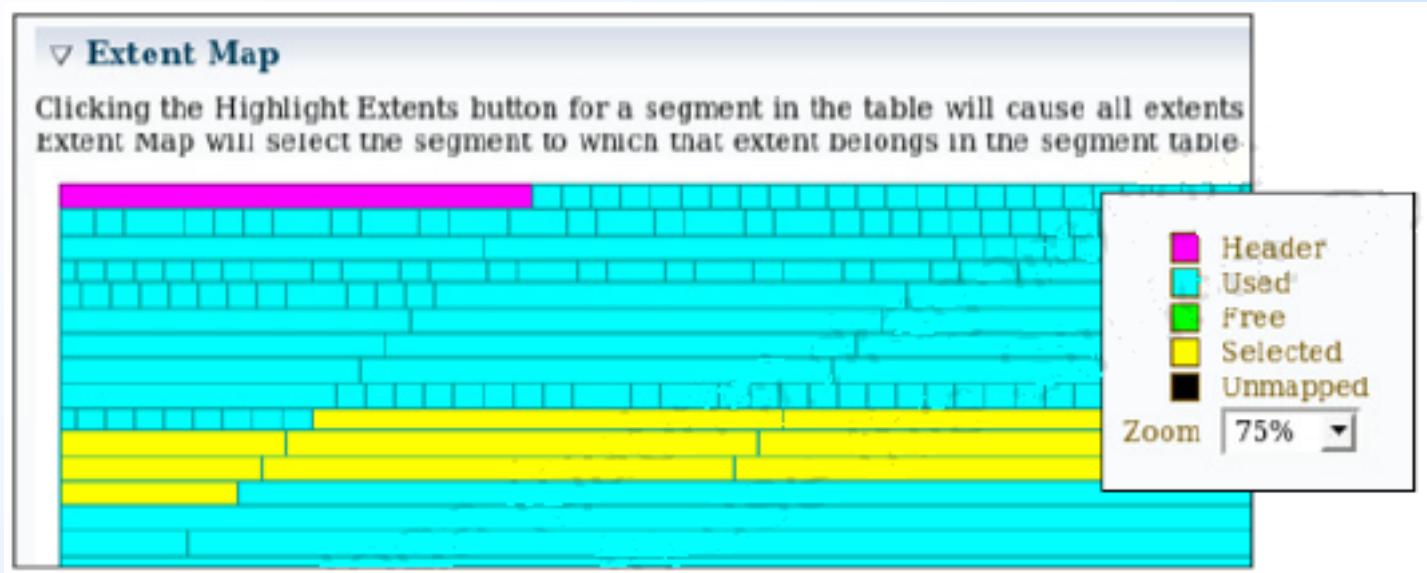
□ Segment Undo

- Se folosește pentru a reface starea datelor (rollback)

□ Segment Temporar

- Este creat atunci când o instructiune SQL are nevoie de spatiu temporar, la terminarea instructiunii spatiul (extent-urile) este redat sistemului

Alocare extent-uri



- Serverul BD determină fișierul de date al tablespace-ului
- Se caută în bitmap-urile fișierului de date blocuri adiacente libere
- Dacă nu există blocuri libere, se caută în alt fișier de date

DEFERRED_SEGMENT_CREATION

- ❑ Implicit este TRUE
- ❑ Când este creată o nouă tabelă, definiția tablei se salvează în dicționarul datelor, dar nu este alocat spațiu pentru date
- ❑ La primul INSERT în tabelă se alocă spațiu
- ❑ Avantaje:
 - ❑ Se salvează spațiu la instalarea unei aplicații cu sute de tabele, dar care nu sunt populate cu date
 - ❑ Durata de instalare de reduce mult

DEFERRED_SEGMENT_CREATION

□ Exemplu

- SQL> SHOW PARAMETERS deferred_segment_creation;
- SQL>CREATE TABLE seg_test(c NUMBER, d VARCHAR2(500));
- SQL>SELECT segment_name FROM user_segments;
 - Obs. Nu afișează nimic
- SQL>INSERT INTO seg_test VALUES(1, 'aaaaaaaaaa')
- SQL>SELECT segment_name FROM user_segments;
 - Obs. Afisează SEG_TEST
- Alte vederi ce pot fi interogate pentru coloana SEGMENT_CREATED: USER_TABLES, USER_INDEXES, USER_LOBS (arată YES când s-a alocat spațiu)
- Tabela SYS.SEG\$ spune ce parametri s-au folosit la creare tabelă

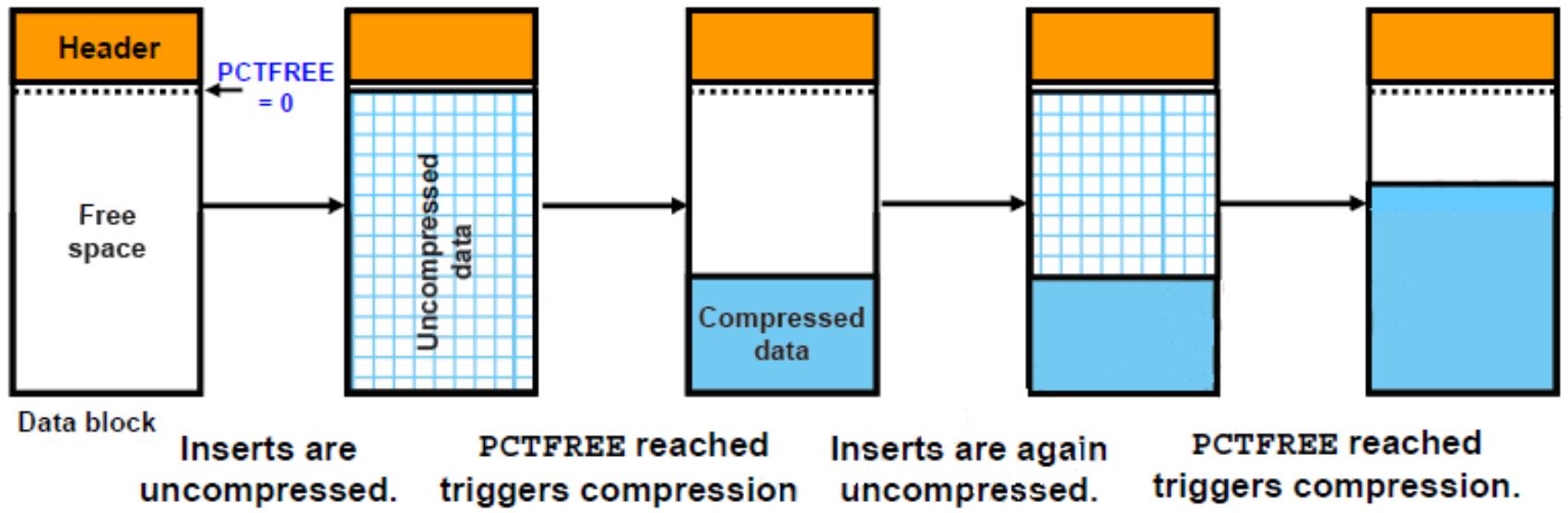
Controlul DEFERRED SEGMENT CREATION

- Cu parametrul DEFERRED_SEGMENT_CREATION ce poate fi setat:
 - În fișierul de initializare
 - Cu ALTER SESSION
 - SQL>ALTER SESSION SET DEFERRED_SEGMENT_CREATION = TRUE;
 - Cu ALTER SYSTEM
- Cu clauza SEGMENT CREATION la CREATE TABLE
 - IMMEDIATE
 - DEFERRED (implicit)
- Indecșii moștenesc de la tabela de bază

Comprimarea spațiului

- Trei metode:
 - Basic table compression
 - Advanced row compression
 - Hybrid columnar compression (cu Exadata)
- Cuvântul cheie COMPRESS
- Basic: Operații bulk (de exemplu
CREATE TABLE as SELECT ..)
- Advanced: la orice operație DML

Comprimare pentru operații INSERT Direct-Path



- CREATE TABLE .. COMPRESS BASIC .. ;
- Se folosește la încărcare date bulk în datawarehouse
- Maximizează spațiul liber contiguu în blocuri

Advanced row compression

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | Y | | Y | | Y |
| G | | Y | | G | |
| | G | | Y | Y | G |

Bloc necomprimat

| | | | | | |
|---|---|---|---|---|---|
| G | Y | | | | |
| | Y | | Y | | Y |
| G | | Y | | G | |
| | G | | Y | Y | G |

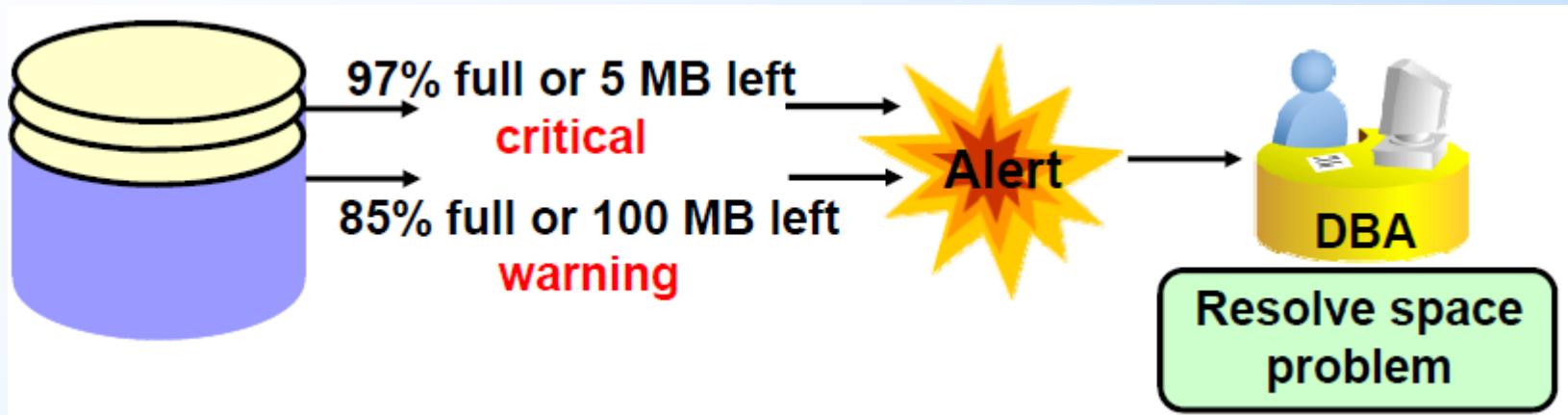
Compresie OLTP cu tabelă de simboluri la începutul blocului

- CREATE TABLE .. ROW STORE COMPRESS ADVANCED ..;
- Se recomandă în medii OLTP active
- Valorile duplicate sunt înlocuite cu o referință la tabela de simboluri

Folosirea Compression Advisor

- Compression Advisor analizează obiectele BD și determină ratele de compresie estimate
- Ajută administratorul BD la determinarea nivelelor potrivite de compresie
- Din Enterprise Manager determină compresia OLTP

Folosirea Pragurilor

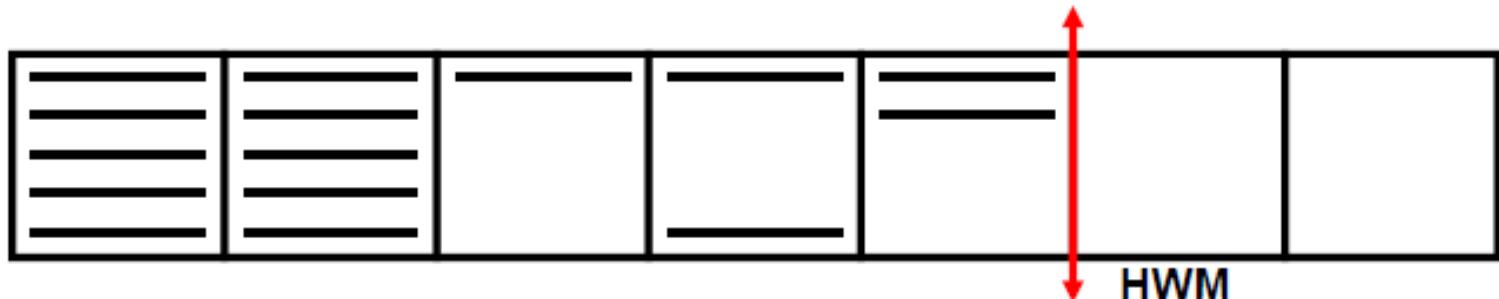


- Se pot fixa praguri fie pentru cât de ocupat este un tablespace, fie pentru spațiul rămas liber, exprimat în procente
 - Critical
 - Warning
- DBMS_SERVER_ALERT este package-ul folosit pentru set și get valori praguri

Rezolvare probleme de spațiu

- Se adaugă fișier, sau se redimensionează fișier existent
- Se fixează AUTOEXTEND ON (atenție la spațiu liber pe disc))
- Se efectuează „shrink” la obiecte dispersate
- Se reduce UNDO_RETENTION
- Se verifică interogările cu durată lungă, ce folosesc tablespace-uri temporare

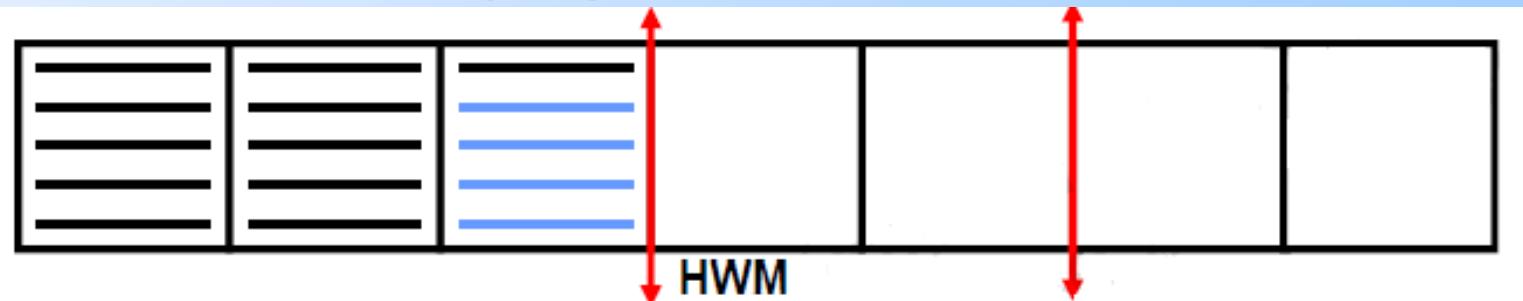
Efectuare „shrink”



`ALTER TABLE employees SHRINK SPACE COMPACT;`



`ALTER TABLE employees SHRINK SPACE;`



Efectuare „shrink”

- În prima fază se deplasează rândurile spre stânga (fără a afecta prin „lock” operații DML)
- În faza a doua „Higher-Water Mark” (HVM) este ajustat și spațiul nefolosit este eliberat
- Clauza COMPACT este folositoare la interogări de lungă durată, pentru a nu fi afectate
 - Progresul operației „shrink” este ținut în blocurile bitmap ale segmentului
 - Faza a doua se poate efectua fără COMPACT la ore când BD e mai puțin accesată

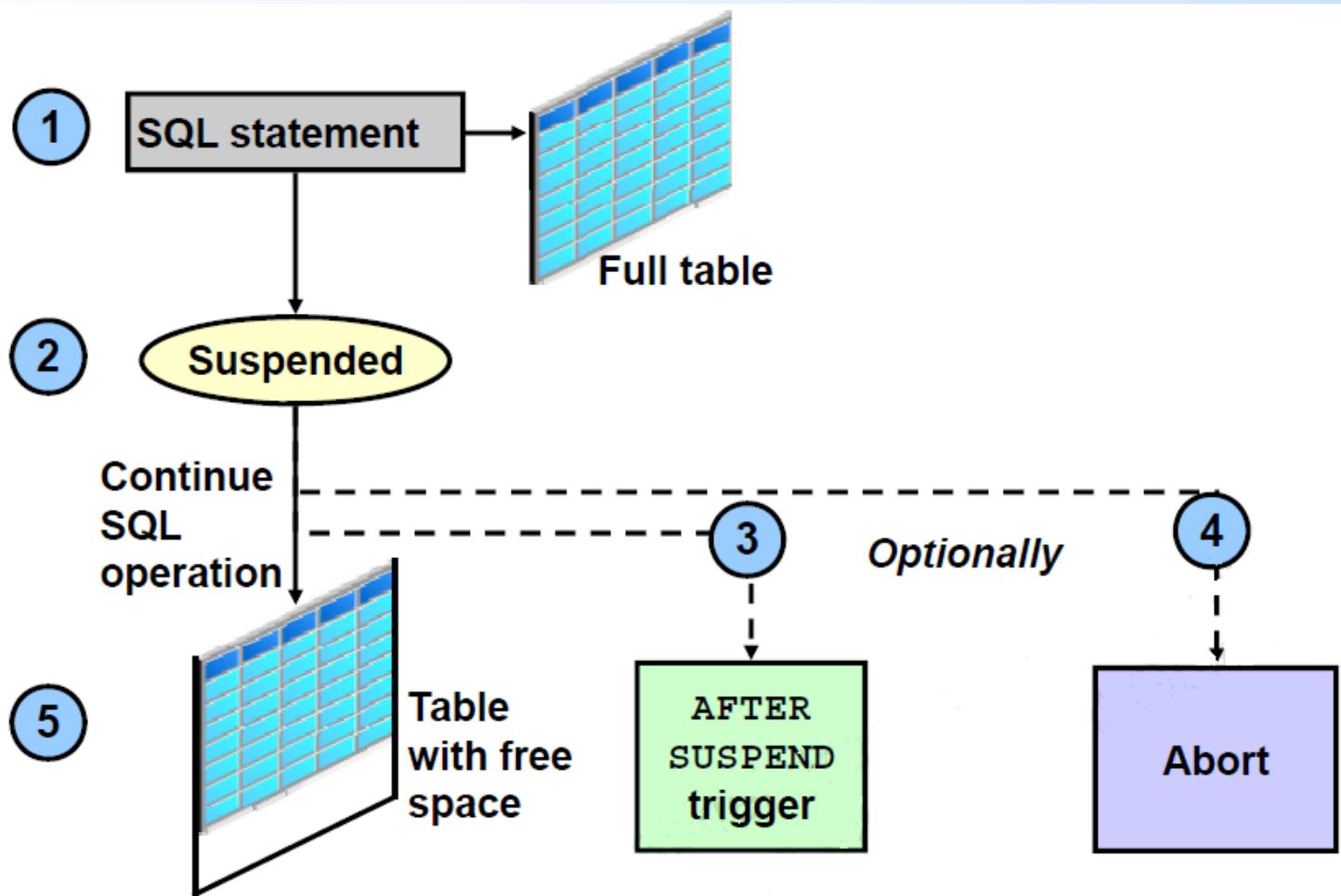
„Automatic Segment Advisor”

- Există un „job” planificat să ruleze în ferestrele:
 - Zile lucrătoare: 22:00 – 02:00
 - Sâmbăta și duminica: începe la 06:00 și durează 20 ore
- Examinează statistici BD, extrage mostre de date de segment și selectează următoarele obiecte pentru evaluare:
 - Tablespace-uri ce au depășit praguri critice
 - Segmente cu activitate intensă sau cu rată de creștere mare

Instrucțiune care poate fi reluată

- Permite oprirea de operații ce consumă mult spațiu înainte de a primi un mesaj de eroare
- Se poate corecta problema în timp ce instrucțiunea este suspendată
- Condiții de suspendare:
 - Lipsă spațiu
 - S-a atins numărul maxim de extent-uri
 - S-a depășit „quota” pentru spațiu (tablespace gestionat de dicționar)
- Poate fi suspendată și reluată de mai multe ori

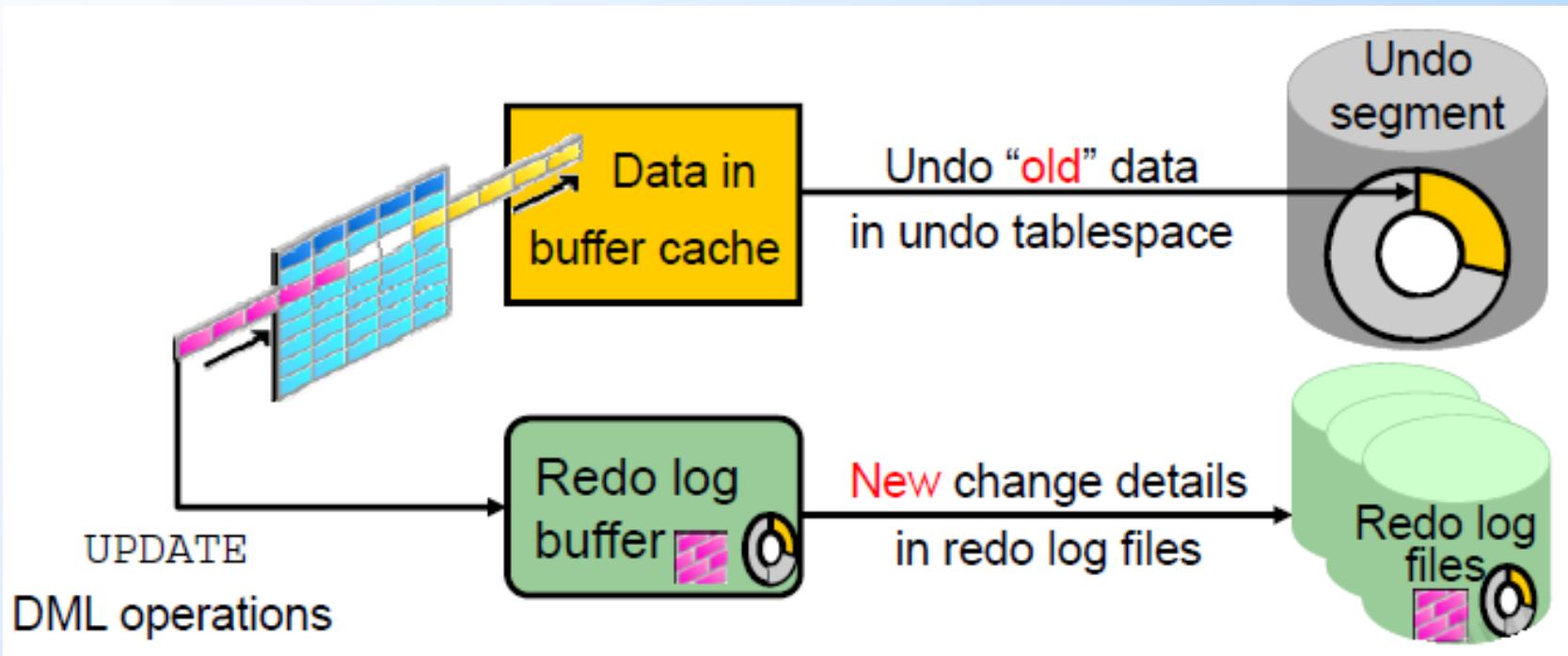
Reluarea instrucțiunilor suspendate



Undo data

- Este o înregistrare corespunzătoare acțiunii unei tranzacții
- Este capturată pentru fiecare tranzacție ce modifică date
- Se păstrează cel puțin până la încheierea tranzacției
- Oferă suport pentru:
 - Acțiuni rollback
 - Interogări consistente la read
 - Oracle Flashback Query, Oracle Flashback Transaction, Oracle Flashback Table
 - Recuperarea de tranzacții eșuate

Tranzacții și Undo Data



Tranzacții și Undo Data

- La începutul unei tranzacții îi este atribuit un segment undo
- Pe durata de viață a tranzacției, înainte de modificarea datelor, datele originale sunt copiate în segmentul undo
- Vederea V\$TRANSACTION spune ce tranzacții sunt asignate la care segmente undo
- Operații paralele și DDL pot cauza folosirea mai multor segmente undo de către o tranzacție

Tranzacții și Undo Data

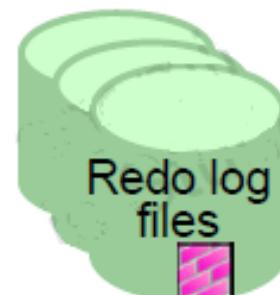
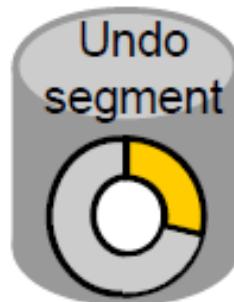
- Segmentele undo
 - Sunt create automat de serverul BD pentru a susține tranzacțiile
 - Sunt constituite din extent-uri care sunt la rândul lor constituite din blocuri de date
 - Acționează ca un buffer circular, crescând sau micșorându-se automat
- Tranzacția scrie într-un extent al segmentului undo până la terminarea tranzacției sau umplerea spațiului extent-ului, caz în care folosește spațiu din extent-ul următor; când toate extent-urile sunt ocupate se reia cu primul extent sau se solicită alocarea unui nou extent pentru segmentul undo

Tranzacții și Undo Data

- Segmentele undo există într-un tablespace specializat numit “undo tablespace”
- Pentru sistemele OLTP cu multe tranzacții concurente de scurtă durată, pot apărea conflicte pe headerul de fișier; soluția este folosirea mai multor fișiere pentru undo tablespace
- Deși o BD poate avea mai multe undo tablespace-uri unul singur poate fi folosit în mod curent pentru oricare instanță a BD
- Fiecare segment undo are cel puțin 2 extent-uri, numărul maxim depinde de dimensiunea blocului (la bloc de 8 kB sunt 32765 blocuri)

Comparație între Undo Data și Redo Data

| | Undo | Redo |
|-----------|---------------------------------------|----------------------------------|
| Record of | How to undo a change | How to reproduce a change |
| Used for | Rollback, read consistency, flashback | Rolling forward database changes |
| Stored in | Undo segments | Redo log files |



Backup și Recovery

- Administratorul BD este responsabil să asigure că BD este disponibilă utilizatorilor:
 - Anticipează și acționează pentru a evita cauzele principale ce cauzează eșuări
 - Încearcă să crească timpul mediu între eșuări (MTBF)
 - Protejează componentele critice prin redundanță
 - Real Application Clusters (RAC)
 - Oracle Data Guard
 - Încearcă să reducă timpul mediu de recuperare (MTTR)
 - Încearcă să minimizeze pierderile de date
 - Fișiere jurnal (Archive log)
 - Tehnologia flashback
 - Standby DBs și Oracle Data Guard

Tipuri de eșuări

- Eșuare instrucțiune
- Eșuare proces utilizator
- Eșuare rețea
- Eroare utilizator
- Eșuare instanță
- Eșuare mediu de stocare

Eșuare instrucțiune

| Probleme tipice | Soluții posibile |
|--|---|
| Încercare de a introduce date invalide într-o tabelă | Se lucrează cu utilizatorii pentru a valida și corecta datele |
| Încercare de a efectua operații cu privilegii insuficiente | Se oferă privilegiile potrivite la nivel de sistem sau obiect |
| Încercare de a aloca spațiu, ce eșuează | <ul style="list-style-type: none">Permite alocare de spațiu pentru reluare instrucțiuneMărește quota owneruluiAdaugă spațiu la tablespace |
| Erori logice în aplicații | Se lucrează cu dezvoltatorii pentru a corecta erori de program |

Eșuare proces utilizator

| Probleme tipice | Soluții posibile |
|---|---|
| Utilizatorul efectuează o deconectare anormală | Nu este nevoie de acțiunea administratorului BD la rezolvarea de eșuare proces utilizator. |
| Sesiunea unui utilizator se termină anormal | |
| Utilizatorul întâlnește o eroare în program ce termină sesiunea | Există procese background la nivel de instantă care efectuează rollback la modificările uncommitted și eliberează lock-urile. |

Eșuare rețea

| Probleme tipice | Soluții posibile |
|-------------------------------------|---|
| Eșuare Listener | Se configurează Listener de backup și connect-time failover |
| Eșuare Network Interface Card (NIC) | Se configurează mai multe plăci de rețea |
| Eșuare conexiune la rețea | Se configurează o conexiune rețea de rezervă |

Eroare utilizator

| Probleme tipice | Soluții posibile |
|--|--|
| Utilizatorul modifică (UPDATE) sau șterge (DELETE) date din greșeală | Se efectuează rollback la tranzacție și la tranzacțiile dependente sau se reface tabela la starea inițială |
| Utilizatorul efectuează DROP TABLE | Se recuperează tabela din recycle bin. Se recuperează tabela dintr-un backup. |

Eroare utilizator

- Se poate utiliza Oracle LogMiner pentru a interoga online redo logs folosind Enterprise Manager sau interfața SQL.
- Datele despre tranzacții sunt persistente în online redo logs mai mult decât în segmentele undo, dacă a fost configurată arhivarea informației redo, atunci există până ce se sterg fișierele arhivă.
- Dacă s- șters din greșeală o tabelă cu DROP, atunci cu tehnologia flashback se poate încerca recuperarea din recycle bin.

Tehnologia flash back

Pentru analiza erorilor

Oracle Flashback Query

Oracle Flashback Versions Query

Oracle Flashback Transaction Query

Pentru recuperare în caz de eroare

Oracle Flashback Transaction Backout

Oracle Flashback Table

Oracle Flashback Drop

Oracle Flashback Database

Tehnologia flash back

- Pentru analiza erorilor:
 - Flashback Query permite a se vizualiza date salvate aşa cum existau ele la un moment dat în trecut: SELECT cu clauza AS OF și o marcă de timp sau System Change Number (SCN)
 - Flashback Version Query permite a se vizualiza date istorice pentru un interval de timp specific: SELECT cu clauza VERSIONS BETWEEN
 - Flashback Transaction Query permite a se vizualiza toate modificările efectuate bazei de date la nivel de tranzacție

Tehnologia flash back

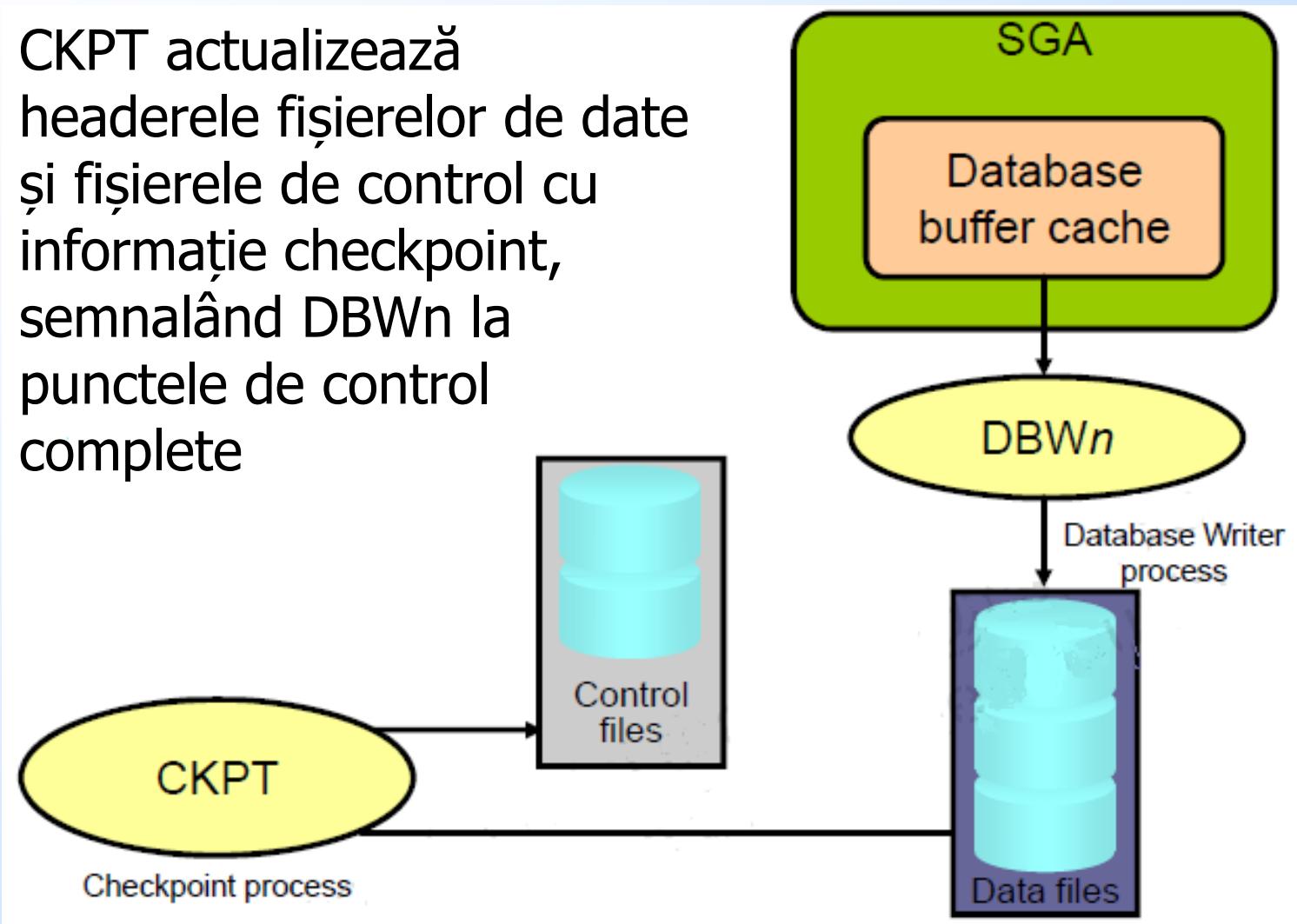
- Pentru recuperare în caz de eroare:
 - Flashback Transaction Backout efectuează rollback la o tranzacție specifică și de asemenea la tranzacțiile dependente
 - Flashback Table reface una sau mai multe tabele la conținutul lor la un moment de timp anterior, fără a afecta alte obiecte
 - Flashback Drop reface efectele unei operații DROP TABLE preluând tabela din recycle bin împreună cu obiectele dependente (indeski și trigere)
 - Flashback Database reface baza de date la un moment dat sau SCN

Eșuare instanță

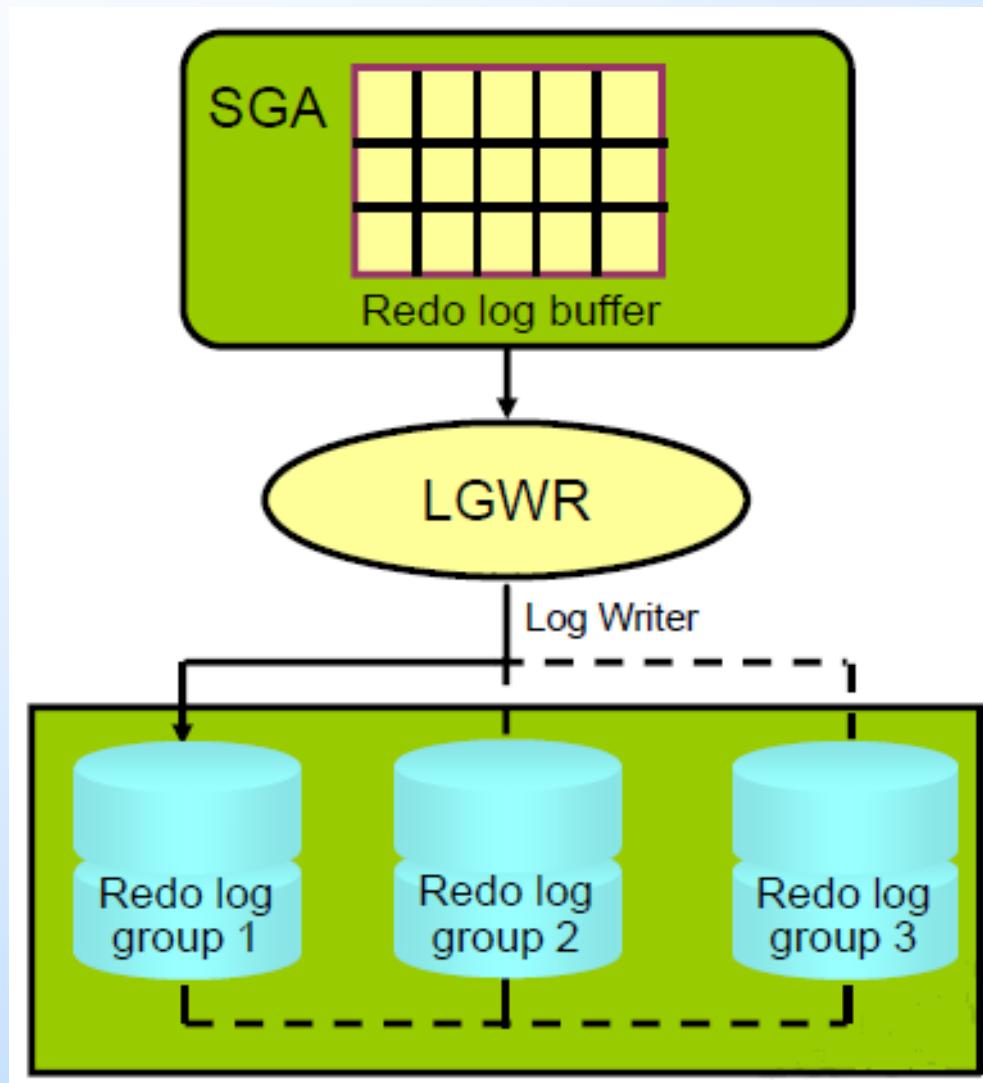
| Probleme tipice | Soluții posibile |
|--|---|
| Întrerupere de energie electrică | Se repornește instanța cu STARTUP. Recuperarea din eșuarea instanței este un proces automat, ce include rularea înainte a modificărilor utilizând redo logs și la sfârșit rularea înapoi a tranzacțiilor ce nu au fost încheiate cu succes (prin COMMIT). |
| Eșuare hardware | |
| Eșuare a unui proces background critic | |
| Proceduri shutdown de urgență | Se investighează cauzele eșecului prin consultarea alert log, fișiere trace și Enterprise Manager. |

Procesul Checkpoint (CKPT)

CKPT actualizează
headerele fișierelor de date
și fișierele de control cu
informație checkpoint,
semnalând DBWn la
punctele de control
complete



Fișiere Redo Log și Log Writer



Fișiere Redo Log și Log Writer

❑ Fișierele redo log

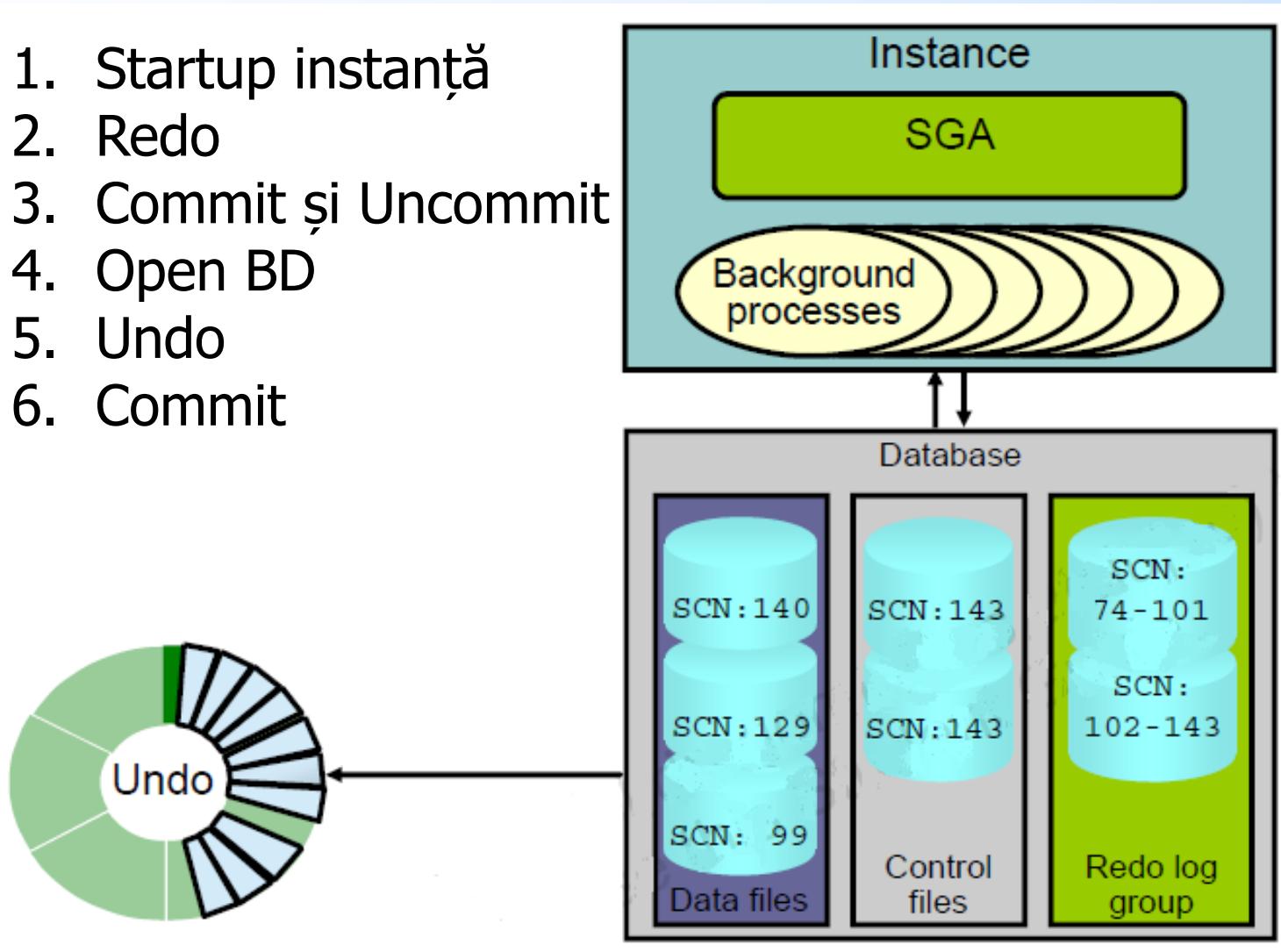
- ❑ Înregistrează modificările efectuate bazei de date
- ❑ Ar trebui să fie multiplexate pentru a proteja împotriva pierderilor de informație

❑ Log Writer (LGWR) scrie:

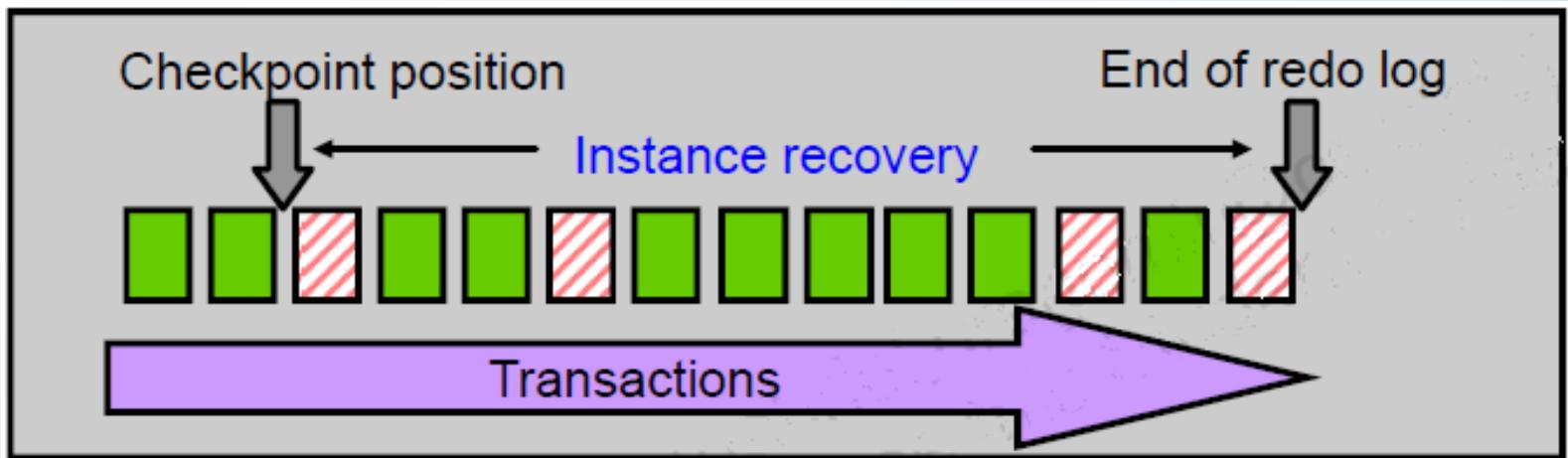
- ❑ La COMMIT
- ❑ Când se umple 1/3 (redo log buffer)
- ❑ La fiecare 3 secunde
- ❑ Înainte să scrie DBWn
- ❑ Înainte de shutdown normal

Faze ale recuperării instanței

1. Startup instanță
2. Redo
3. Commit și Uncommit
4. Open BD
5. Undo
6. Commit



Reglarea recuperării instanței

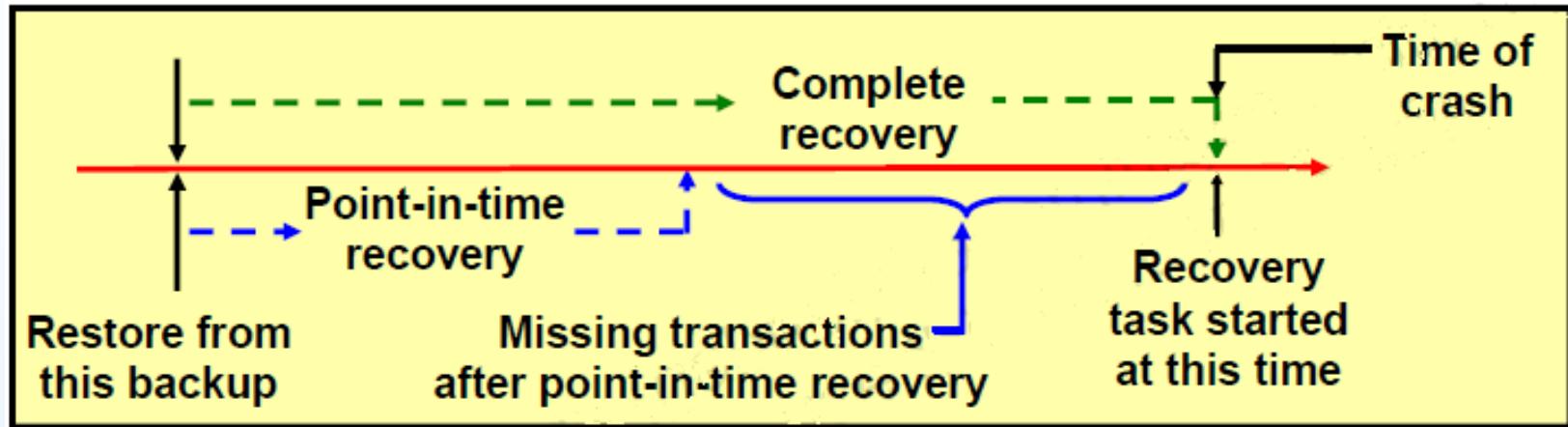


- În timpul recuperării instanței, tranzacțiile între poziția checkpoint și sfârșitul redo log trebuie aplicate fișierelor de date
- Reglarea se face prin controlul diferenței între poziția checkpoint și sfârșitul redo log

Eșuare mediu de stocare

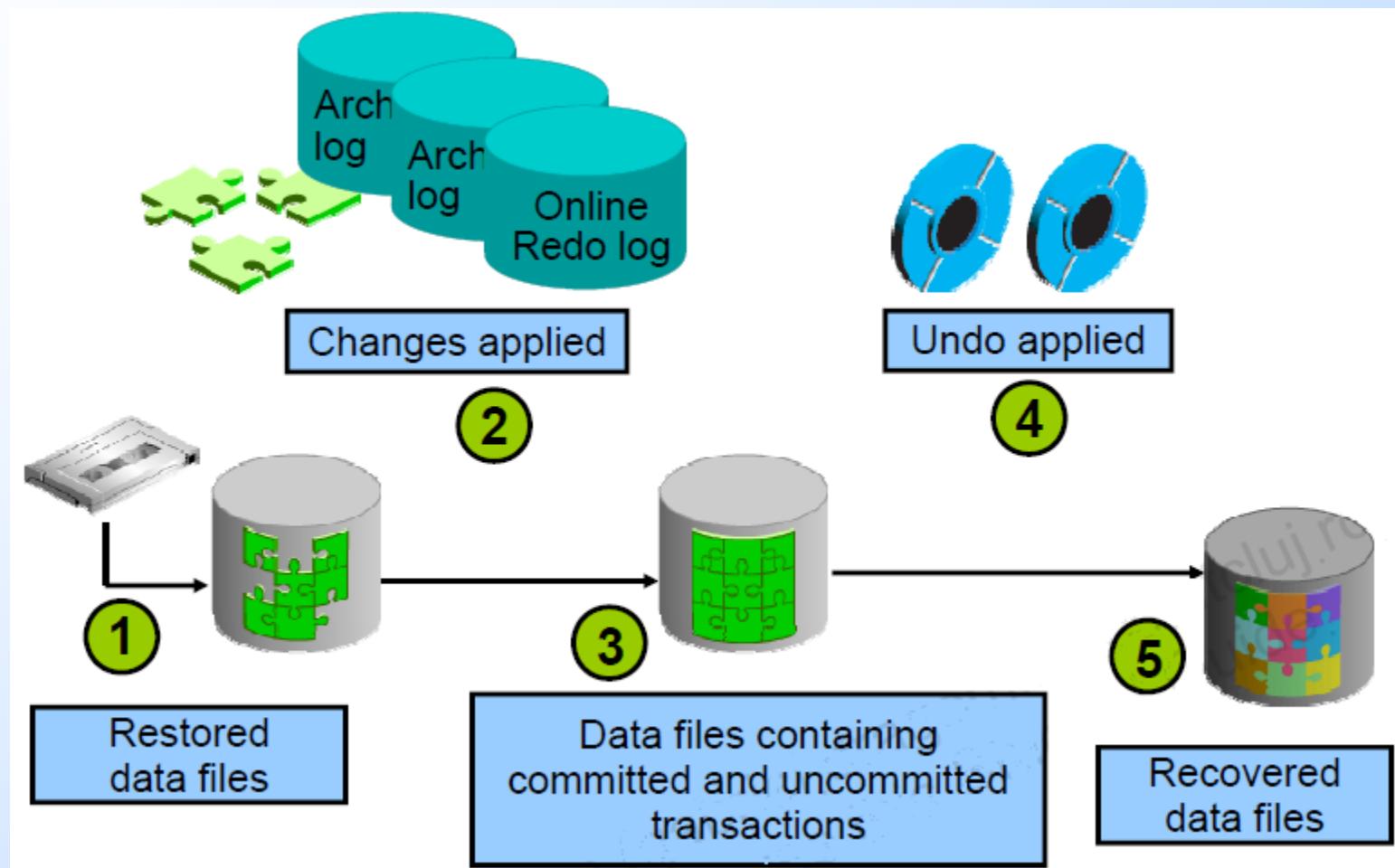
| Probleme tipice | Soluții posibile |
|---|---|
| Eșuare disk drive | <ol style="list-style-type: none">1. Se restaurează fișierul afectat din backup |
| Eșuare disk controller | <ol style="list-style-type: none">2. Se informează baza de date cu privire la noua locație a fișierului (dacă este cazul) |
| Ștergerea sau coruperea unui fișier de care are nevoie o operație pe baza de date | <ol style="list-style-type: none">3. Se recuperează fișierul prin aplicarea de informație redo (dacă este cazul) |

Comparație între recuperare completă și incompletă



- Recuperare completă: aduce BD sau tablespace la zi incluzând toate modificările de date commit efectuate până la momentul de timp solicitat
- Recuperare incompletă (PITR): aduce BD sau tablespace la un moment de timp specificat, în trecut (point in time recovery)

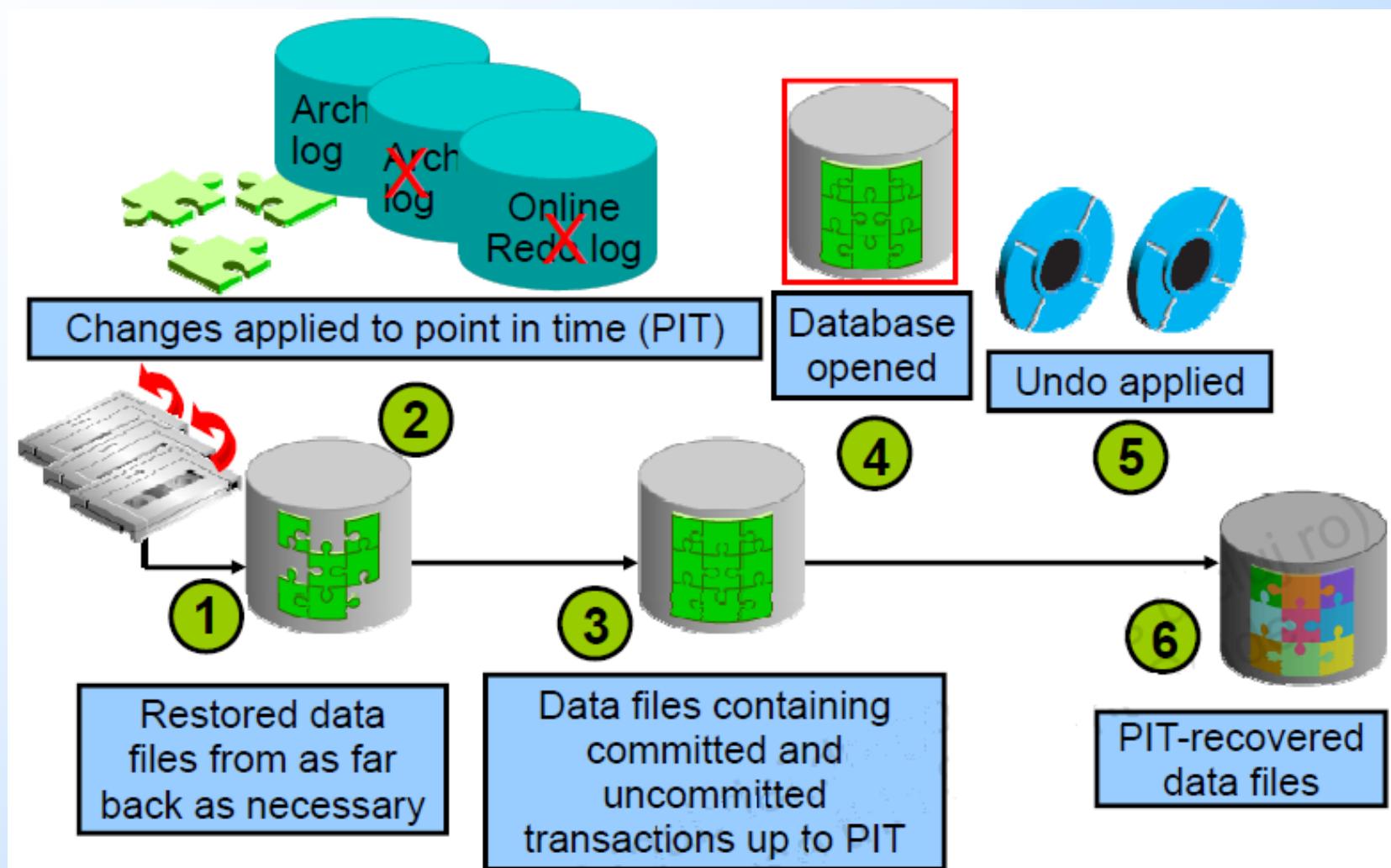
Procesul de recuperare completă



Procesul de recuperare completă

1. Fișierele stricate sau lipsă sunt restaurate din backup
2. Sunt aplicate modificările din backup-uri incrementale, fișiere redo log arhivate și fișiere redo log online
3. Fișierele de date restaurate vor conține acum modificări committed și uncommitted
4. Blocurile undo sunt folosite pentru rollback a modificărilor uncommitted (transaction recovery)
5. Fișierele de date sunt recuperate și consistente

Procesul de recuperare Point-in-Time



Procesul de recuperare Point-in-Time

1. Se restaurează fișierele de date din backup (fie prin copiere fizică a fișierelor la nivel SO, fie cu comanda RMAN RESTORE)
2. Se folosește comanda RECOVERER – se aplică redo din fișiere redo log arhivate
3. Stare pe parcursul recuperării: fișierele de date conțin unele tranzacții încheiate cu succes și altele nefinalizate
4. Se folosește comanda ALTER DATABASE OPEN – BD se deschide înainte de aplicare undo pentru disponibilitate maximă
5. Se aplică undo data (pentru tranzacții nefinalizate)
6. Procesul se încheie

Configurare pentru recuperare

- Se programează backupuri frecvente
- Se multiplexează fișierele de control
- Se multiplexează grupuri redo log
- Se păstrează copii arhivate ale jurnalelor redo

Configurare Fast Recovery Area

- Este spațiul pe disc unde se păstrează arhivele jurnalelor, backup-uri, jurnale flashback, fișiere de control multiplexate, jurnale redo multiplexate
- Se recomandă să fie pe alt disc față de fișierele BD și fișierele jurnal și de control primare
- Dimensiunea spațiului ar trebui să fie dublu față de dimensiunea BD

Configurare Fast Recovery Area

- Spațiul este gestionat prin politici de retenție
- Serverul BD Oracle gestionează automat spațiul luând decizia când să șteargă fișiere backup de care nu mai este nevoie
- Spațiul este monitorizat continuu cu Enterprise Manager pentru a preîntâmpina umplerea sa
- Periodic se copiază fișierele pe bandă magnetică

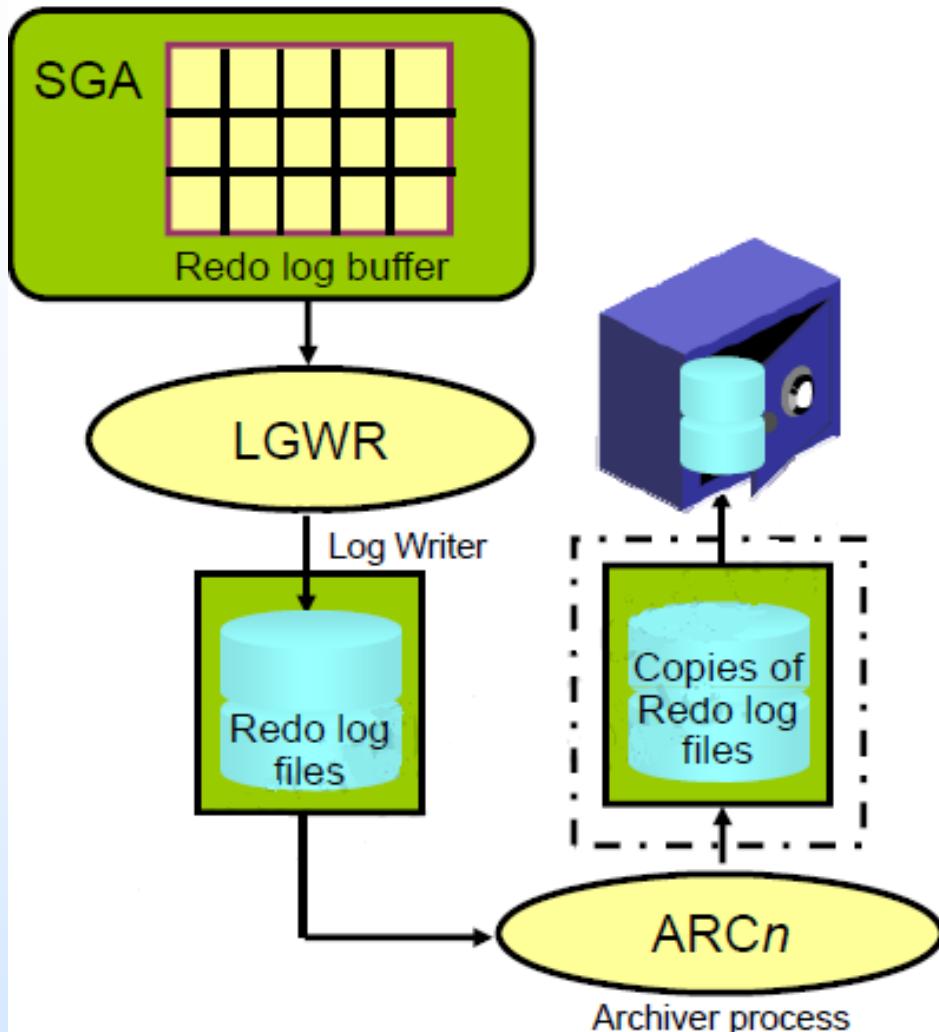
Multiplexarea fișierelor de control

| | Stocare ASM | Stocare sistem fișier |
|---|---|--|
| Best practice | O copie pe fiecare grup de disc (cum ar fi +DATA și +FRA) | Cel puțin două copii, fiecare pe alt disc (cel puțin unul pe controler de disc separat) |
| Pași pentru a crea fișiere de control suplimentare | Nu sunt necesare copii suplimentare ale fișierului de control | <ol style="list-style-type: none">1. Se modifică SPFILE cu comanda ALTER SYSTEM SET fișiere_control2. Se oprește BD3. Se copiază fișierul de control în altă parte4. Se pornește BD și se verifică adăugarea noului fișier de control |

Fișiere jurnal redo

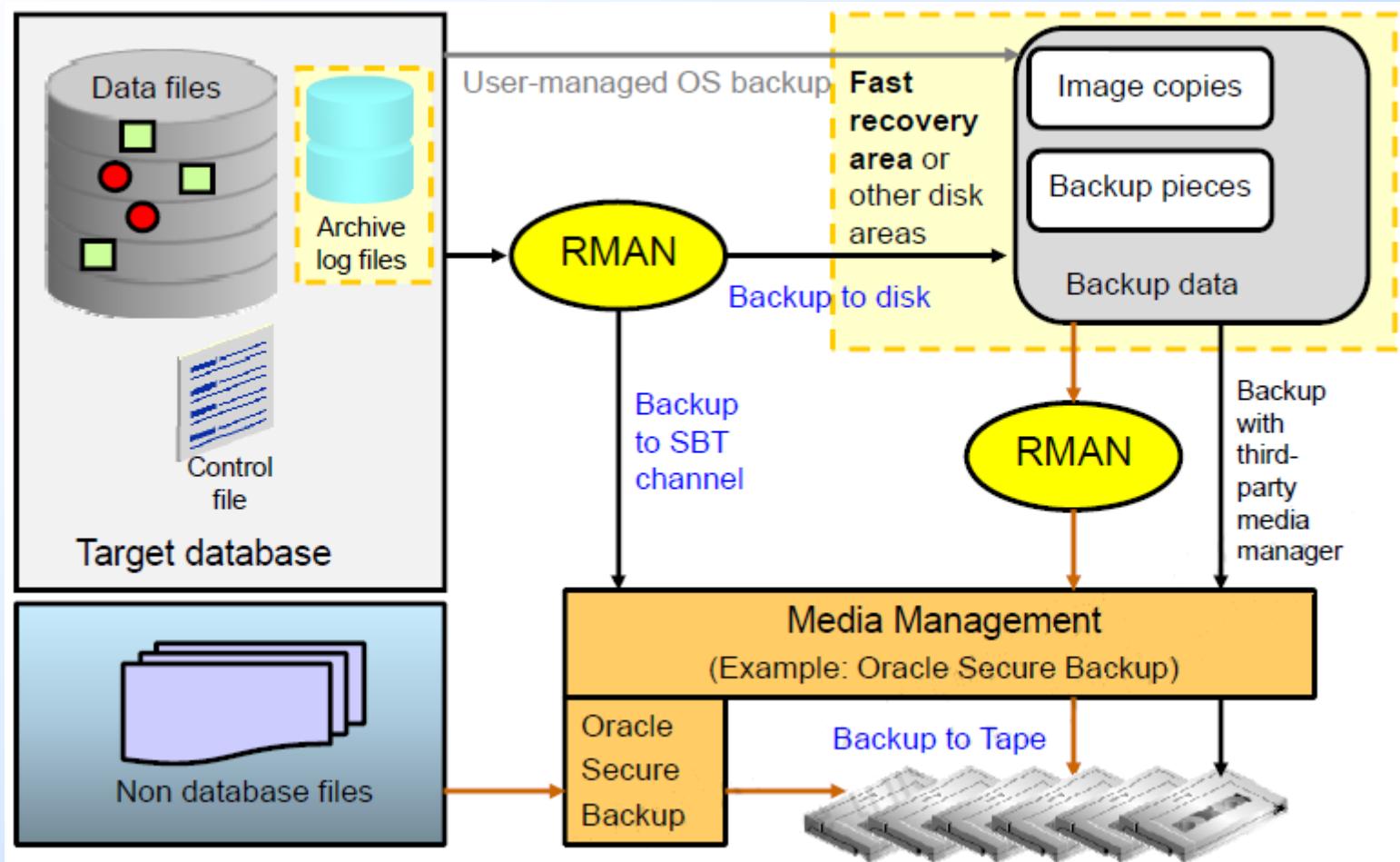
- Oracle recomandă:
 - Fiecare grup să aibă minim două fișiere
 - Fiecare fișier să fie pe un disc separat când se utilizează sistem fișier sau pe un grup de discuri separat când se folosește ASM (+DATA, +FRA)
- Atenție la performanță, deoarece un commit se finalizează doar după ce informația despre tranzacție este scrisă în toate fișierele jurnal

Procesul Archiver (ARCn)



- Este un proces background optional
- Arhivează fisierile jurnal redo online când BD este în mod ARCHIVELOG
- Conservă modificările asupra BD

Soluții Backup



Oracle Secure Backup

- Oracle Secure Backup și RMAN oferă o soluție end-to-end:
 - Gestiunea centralizată backup pe bandă magnetică
 - Administrare media pentru RMAN
 - Backup date în rețea – servere, clienți, Network Attached Storage (NAS) și alte dispozitive de stocare

Backup manual

- Folosește scripturi scrise de utilizator
- Necesită ca fișierele BD să fie în starea potrivită pentru backup
- Se bazează pe comenzi sistem de operare pentru backup fișiere

Actiuni ale scriptului de backup

- Se interoghează V\$DATAFILE pentru a determina ce fișiere cu date se salvează
- Se interoghează V\$LOGFILE pentru a identifica fișierele online redo log
- Se interoghează V\$CONTROLFILE pentru a identifica fișierele de control
- Se pune fiecare tablespace în modul online backup
- Se interoghează V\$BACKUP pentru a determina ce fișiere sunt atașate unui tablespace aflat în mod online backup
- Se emit comenzi SO de copiere fișiere la locația backup
- Se scoate fiecare tablespace din modul online backup

Terminologie backup

□ Strategie backup:

- Întreagă – toată BD
- Parțială – porțiune din BD

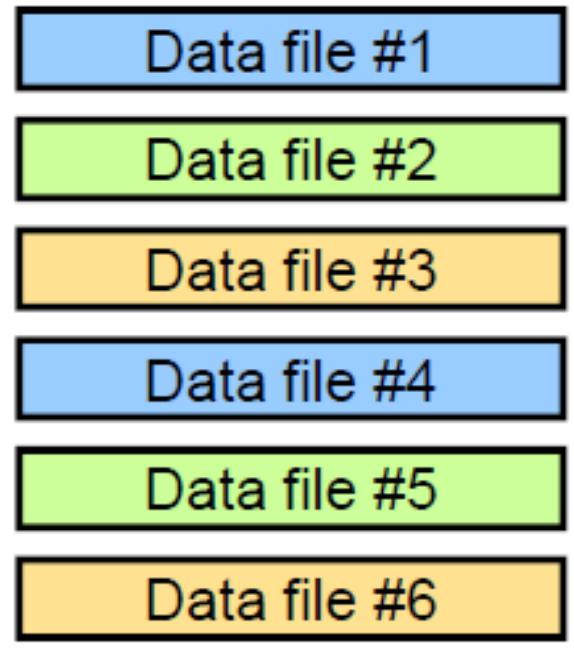
□ Tip backup:

- Complet – toate blocurile de date
- Incremental – doar informația modificată de la un backup precedent
 - Cumulativ – modificări de la început
 - Diferențial – modificări de la incrementul precedent

□ Mod backup:

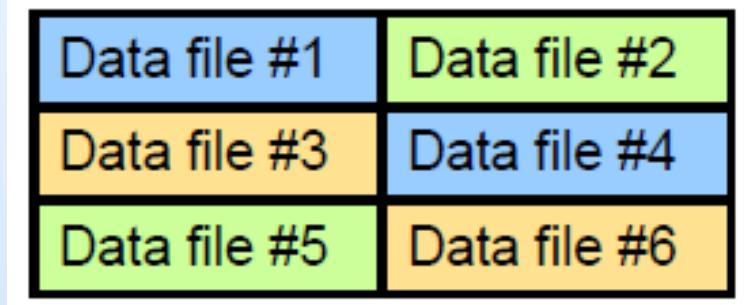
- Offline – consistent
- Online - inconsistent

Tipuri de backup



Copii imagine

(duplicate ale fișierelor de date
și fișierelor jurnal la nivel SO)



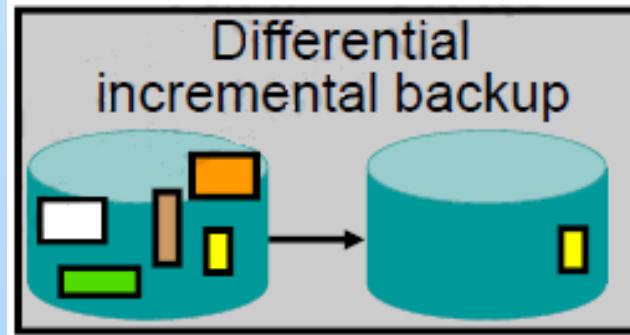
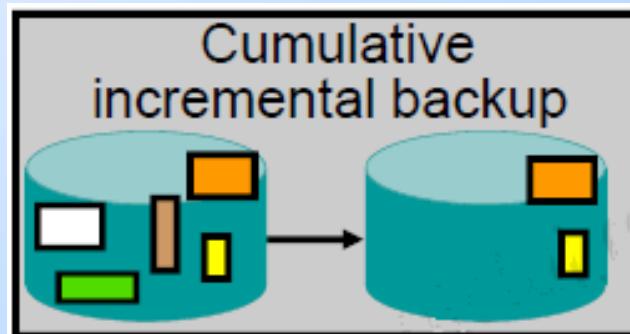
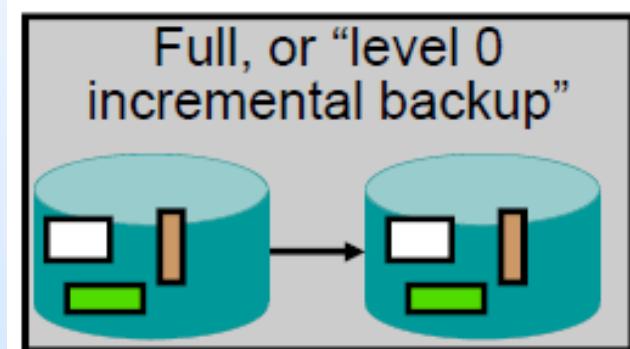
Set backup

(fișiere binare comprimate în
format proprietar Oracle)

Blocurile goale nu sunt salvate, de
obicei 20% din blocuri sunt goale
→ mai avantajos

Tipuri backup RMAN

- Full backup conține toate blocurile de date, este echivalent cu Incremental backup level 0
- Backup incremental cumulativ level 1 conține doar blocurile modificate de la precedentul backup incremental level 0
- Backup incremental diferențial level 1 conține doar blocurile modificate de la ultimul backup incremental



Recomandări Oracle

Full backup
+ daily incremental
= new “full” backup
+ daily archived logs for recovery



- Strategia Oracle pentru backup folosește backup incremental și actualizare incrementală
- Atunci când BD este în modul ARCHIVELOG poate fi restaurată la ultima tranzacție finalizată cu succes
RMAN>BACKUP DATABASE PLUS ARCHIVELOG;

Pornirea BD

1. Starea NOMOUNT: instanța citește fișierul cu parametri de initializare
2. Starea MOUNT: instanța verifică dacă toate fișierele de control menționate în fișierul cu parametri de initializare sunt prezente și sincronizate, dacă un fișier lipsește sau este corrupt instanța rămâne în starea NOMOUNT
3. Starea OPEN:
 - Instanța verifică dacă grupurile de fișiere redo log au cel puțin un membru prezent (membri lipsă sunt notați în alert log)
 - Instanța verifică dacă toate fișierele de date sunt prezente (fișierele offline nu sunt verificate) dacă nu există vreun fișier instanța rămâne în starea MOUNT

Pornirea BD

(continuare)

- Instanța verifică dacă toate fișierele de date (care nu sunt offline sau read-only) sunt sincronizate cu fișierul de control, dacă este necesar este executată automat recuperarea, dacă nu se poate recupera din jurnalele redo online este necesară media recovery (de către administratorul BD) și instanța rămâne în starea MOUNT (v\$recoverer_file oferă informații asupra fișierelor ce necesită atenție)
- După ce BD este OPEN, instanța poate eșua dacă se pierde:
 - Un fișier de control
 - Un fișier ce aparține tablespace-urilor SYSTEM sau UNDO
 - Un întreg grup redo log (dacă cel puțin un fișier redo log este disponibil, instanța rămâne OPEN)

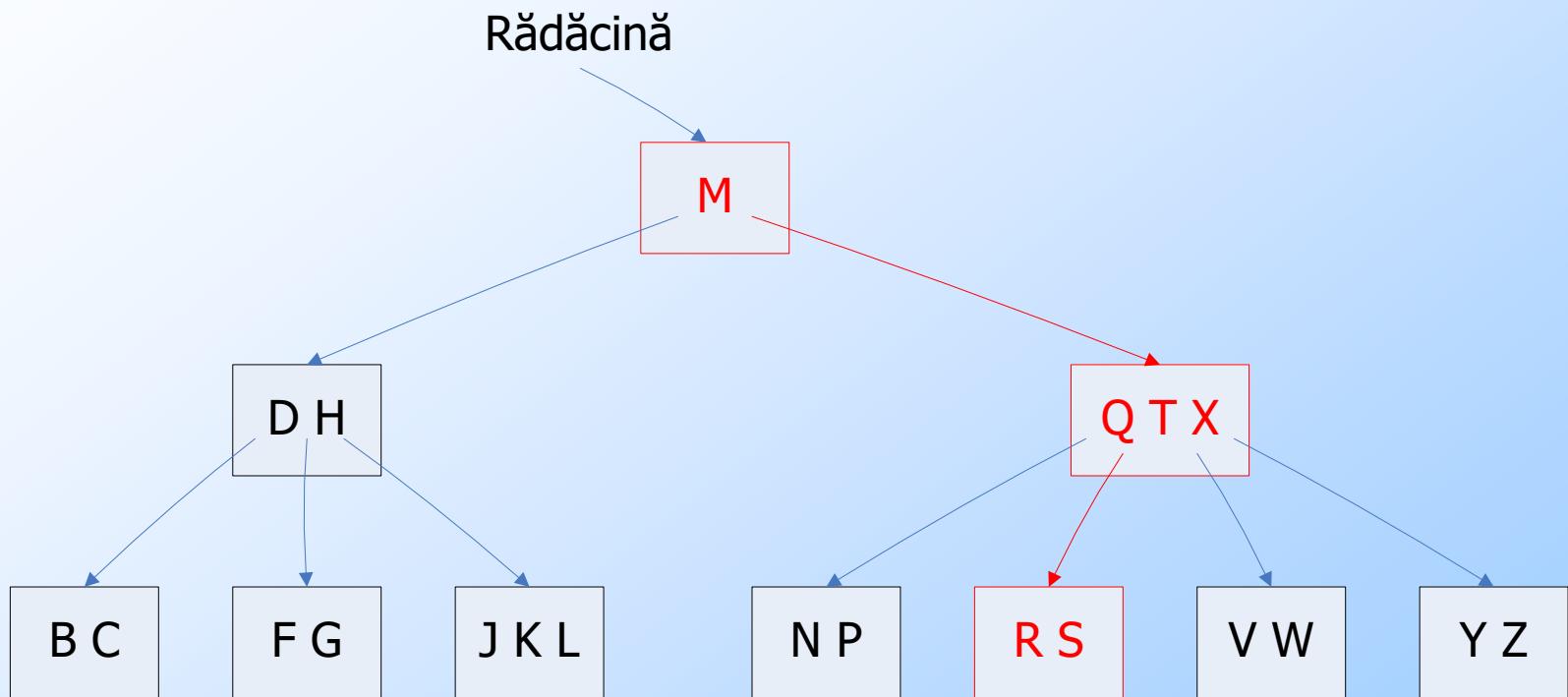
Administrare BD

Indecsi
Autorizare
Tranzactii

Indecși

- *Index* = structură de date folosită (în scopul performanței) pentru a mări viteza de acces la tuplele unei relații, definită pe valorile unuia sau mai multor attribute.
- Poate fi o tabelă de dispersie (hash), dar de obicei SGBD-urile folosesc arbore de căutare echilibrat cu noduri gigant (o pagină disc completă) numit *B-tree*.

B-Tree



Un arbore “B-Tree”, cu culoare roșie este marcat traseul de căutare a literei R.

Declarare Indecși

- Nu există standard!

- Sintaxa:

```
CREATE INDEX BeerInd ON  
Beers(manf);
```

```
CREATE INDEX SellInd ON  
Sells(bar, beer);
```

Folosire Indecși

- Fiind dată valoarea v , indexul conduce doar la acele tuple ce au v în atributul(-ele) indexului.
- Exemplu: se folosește BeerInd și SellInd pentru a găsi prețurile berilor fabricate de “Pete’s” și vândute de “Joe”.

Folosire Indecși

```
SELECT price FROM Beers, Sells  
WHERE manf = 'Pete''s' AND  
Beers.name = Sells.beer AND  
bar = 'Joe''s Bar';
```

1. Se folosește BeerInd pentru a obține toate berile fabricate de "Pete's".
2. Apoi se folosește SellInd pentru a obține prețurile acelor beri, ce au bar = "Joe's Bar"

“Database Tuning”

- Decizia ce indecși să fie creați reprezintă o problemă foarte importantă.
- **Pro:** Un index mărește viteza de execuție a interogărilor (ce fac uz de indexul respectiv).
- **Con:** Un index încetinește toate actualizările relației de definiție a indexului deoarece trebuie modificat și indexul.

Exemplu: Tuning

- Presupunem că singurele lucruri făcute cu BD “beers” au fost:
 1. S-au adăugat fapte noi într-o relație (10%).
 2. S-a căutat prețul unei beri la un bar dat (90%).
- În acest caz **SellInd** pentru **Sells(bar, beer)** este necesar, dar **BeerInd** pentru **Beers(manf)** nu este necesar.

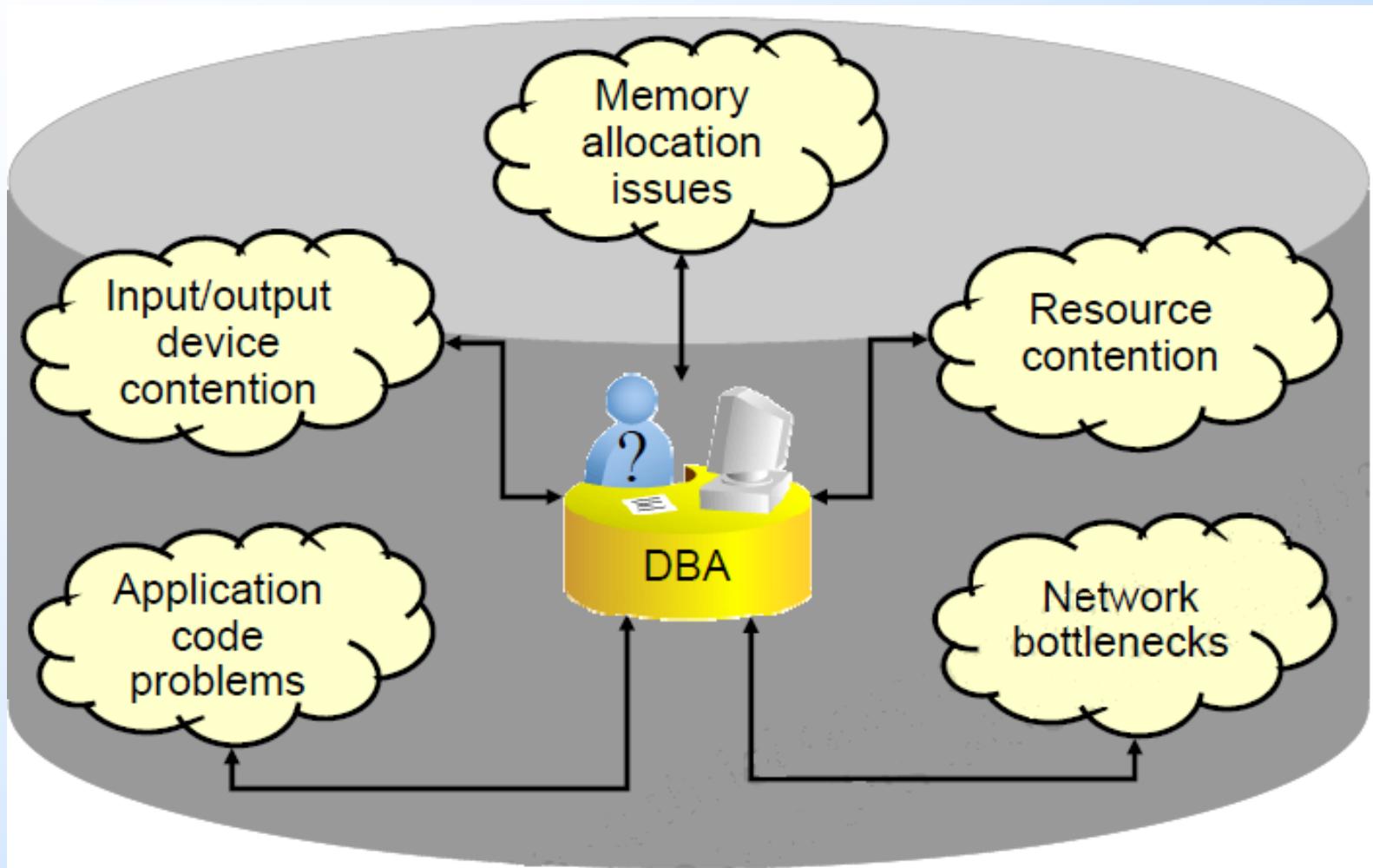
Sfaturi pentru Tuning

- Să se efectueze o cercetare laborioasă.
 - Reglarea manuală pentru performanță este dificilă.
- Sfatul este să se urmărească *încărcarea interogărilor*, de exemplu:
 1. Se aleg aleator interogări din istoricul interogărilor pe BD, sau
 2. Proiectantul oferă un eșantion al încărcării.

Sfaturi pentru Tuning

- Sfatul este ca în continuare să se genereze îndecsi candidați și să se evalueze fiecare index pentru Încărcare .
 - Fiecare eşantion de interogare se transmite optimizatorului de interogări, presupunându-se că doar indexul curent este disponibil.
 - Se măsoară îmbunătățirea/degradarea duratei medii de rulare a interogărilor.

Gestiunea performanței - Oracle



Gestiunea performanței - Oracle

- Oracle Enterprise Manager Database Express oferă mai multe posibilități pentru a monitoriza performanța instanței BD
- Pe pagina principală există grafice care arată:
 - Funcționarea CPU
 - Sesiunile active
 - Folosirea memoriei
 - Folosirea stocării de date
- Pe pagina principală sunt afișate alerte
- Detaliile sunt disponibile pe pagina Performance Hub

Gestiunea performanței - Oracle

- Graficele afișate de Oracle Enterprise Manager sunt construite din vederi de performanță, ce pot fi accesate și cu comenzi SQL
- Activități pentru tuning:
 - Planificare performanță – procesul prin care se stabilește mediul hardware, software, SO, infrastructură rețea, ș.a.
 - Tuning instanță – ajustarea parametrilor BD și SO
 - Tuning SQL – la nivel aplicație, componentele să nu concureze pentru acapararea resurselor în mod nenecesar

Planificare performanță

- Opțiuni de investiție – de exemplu numărul de discuri
- Arhitectura sistemului
- Scalabilitate – gestiunea creșterii numărului de utilizatori, clienți, sesiuni, tranzacții (serializarea operațiilor duce la gâtuirea sistemului – felul cum sunt scrise comenzi SQL afectează performanța)
- Principii de design aplicație – simplitate în design, folosire vederi și indecsi, modelarea datelor
- Testarea încărcării (workload), modelarea și construirea
- Lansarea (deploying) de aplicații noi – înainte de lansarea în producție orice aplicație trebuie testată cu un volum de date și încărcare sistem reprezentative

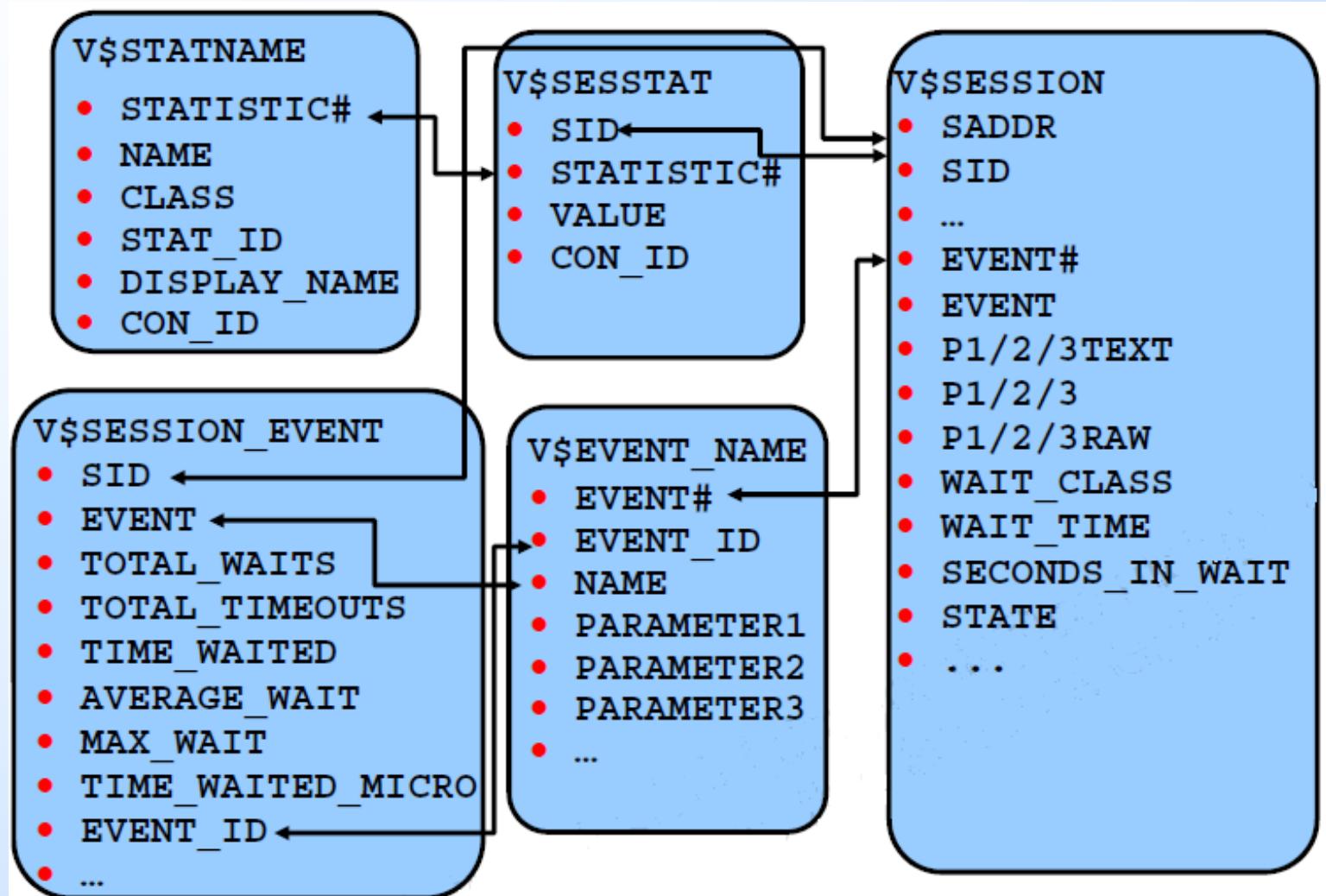
Tuning instanță

- Se stabilesc obiective specifice, de exemplu „Se procesează 500 tranzacții per minut (aplicația vânzări)“
- Se alocă memorie în mod corespunzător pentru aplicație, prea puțină memorie alocată pentru anumite părți ale serverului BD poate conduce procesele background să lucreze ineficient, de aceea trebuie efectuată analiză
- De obicei gâtuirea BD vine de la operațiile I/O cu discul
- Performanța mai este influențată de configurația SO

Date folosite pentru Tuning

- Statistici cumulative:
 - Evenimente wait cu informație de timp (cele mai importante sunt buffer busy waits)
 - Model de timp (procent de timp BD prin comparație cu nivel de referință)
- Metrici – contorizări statistice per unitate (timp – secunde, tranzacție, sau sesiune)
 - Se pot stabili praguri ce cauzează alerte (de exemplu atunci când numărul de citiri per milisecundă depășesc un prag, sau zona archive log este 95% plină)
- Statistici eșantion: istoric sesiune activă
 - Statistici per sesiune
 - Statistici per SQL
 - Statistici per serviciu
 - Alte dimensiuni

Statisticci per sesiune



Statistici per sesiune

□ V\$SESSION

- Se poate determina dacă sesiunea curentă este de tip utilizator sau este de tip background

□ V\$SESSION sau V\$SESSION_WAIT

- Se pot determina resursele sau evenimentele pentru care așteaptă sesiunea curentă

□ V\$SESSTAT

- statistici sesiune utilizator

□ V\$SESSION_EVENT

- Informații despre waits pentru evenimente de către sesiune

Statistici per sesiune

□ V\$SESSTAT și V\$SYSSTAT

- Valori cumulate per instanță, care se resetează când instanța este oprită

□ V\$MYSTAT

- Statistici pentru sesiunea curentă

□ V\$SESSMETRIC

- Metrici de performanță pentru toate sesiunile active
- Folosire CPU, număr citiri fizice (la nivel bloc de date), număr hard parses și rata de citiri logice (la nivel SQL)

Evenimente wait (peste 800)

- V\$EVENT_NAME
- Sunt statistici ce sunt incrementate de un proces sau thread server pentru a indica că trebuie să aștepte după un eveniment pentru a-și continua acțiunea
- Indică probleme:
 - Conflict de blocare
 - Conflict buffer
 - Conflict I/O
- Clase evenimente wait:
 - Administrative, Application, Cluster, Concurrency, Configuration, Idle, Network, Other, Scheduler, System I/O și User I/O

Gestiunea memoriei

- Automatic Memory Management (AAM) permite să se specifică memoria totală alocată instanței (SGA și PGA) – **recomandat să se folosească**
- Automatic Shared Memory Management (ASMM) permite:
 - Să se specifică memoria totală SGA – un parametru de initializare
 - Serverului Oracle să administreze cantitatea de memorie alocată pentru shared pool, Java pool, buffer cache, streams pool și large pool
- Administrare manuală a memoriei partajate
 - Dimensionează componentele cu parametri de initializare
 - Se folosește Memory Advisor pentru recomandări

SQL Tuning

- Se identifică instrucțiuni SQL ce au nevoie de tuning
- Se regleză instrucțiune cu instrucțiune
- Se regleză aplicația per ansamblu
- Există SQL Advisors pentru a identifica și regla instrucțiuni SQL

Optimizatorul Oracle

- Determină planul de execuție eficient (poate fi consultat prin Enterprise Manager, EXPLAIN PLAN sau din linia de comandă sqlplus, AUTOTRACE)
- Evaluează expresii și condiții
- Folosește statistici sistem (I/O, CPU, etc.) și la nivel de obiect (număr de tuple, index, etc.)
- Decide cum să se acceseze datele
- Decide ce tipuri de indecsi să se folosească la implementarea operației join
- Determină calea (path) cea mai eficientă

Statistică Optimizator

- Reprezintă un snapshot la un moment dat de timp
- Sunt persistente față de repornirea instanței
- Sunt colectate automat
- **Exemplu:**

```
SQL>SELECT COUNT(*) FROM hr.employees;
```

Rezultat: 214

```
SQL>SELECT num_rows FROM dba_tables WHERE  
owner = 'HR' AND table_name = 'EMPLOYEES';
```

Rezultat: 107

Statistici Optimizator

- Statisticile pentru tabele și indecsi sunt stocate în dicționarul datelor și nu sunt timp-real
- Statisticile includ: dimensiunea tabelei sau indexului în blocuri de date, număr de tuple, dimensiunea medie a unei tuple și numărul de șenile (chain) pentru tabele, înălțimea și numărul de tuple șterse de tip frunză pentru indecsi
- Deoarece impactul asupra performanței al menținerii statisticilor în timp real este prohibitiv, statisticile se colectează periodic (în ferestre de timp de menenanță, automat, prin job specific)

Vederi pentru Statistici

V\$SYSSTAT

- STATISTIC#
- NAME
- CLASS
- VALUE
- STAT_ID

V\$SYSTEM_WAIT_CLASS

- WAIT_CLASS_ID
- WAIT_CLASS#
- WAIT_CLASS
- TOTAL_WAITS
- TIME_WAITED

V\$SGASTAT

- POOL
- NAME
- BYTES

V\$EVENT_NAME

- EVENT_NUMBER
- EVENT_ID
- NAME
- PARAMETER1
- PARAMETER2
- PARAMETER3
- WAIT_CLASS

V\$SYSTEM_EVENT

- EVENT
- TOTAL_WAITS
- TOTAL_TIMEOUTS
- TIME_WAITED
- AVERAGE_WAIT
- TIME_WAITED_MICRO

Vederi pentru Statistici

- Statisticile la nivel instanță sunt dinamice și se resetează la restart instanță
- Statistici wait events
 - Vederea V\$EVENT_NAME conține toate evenimentele wait posibile
 - Vederea V\$SYSTEM_EVENT conține statistici cumulative pentru toate sesiunile (de la pornirea instanței)
- Statistici la nivel sistem
 - Vederea V\$STATNAME
 - Sunt peste 400 statistici
 - Vederea V\$SYSSTAT conține totaluri de la pornirea instanței

Vederi pentru Statistici

- Statisticile systemwide sunt clasificate după: activitate la nivel general de instanță, activitate redo log buffer, locking, activitate buffer cache BD
- Fiecare statistică sistem poate apartine la mai multe clase
- Statistici globale SGA
 - Există vederea V\$SGASTAT
 - Afisează informații cumulative referitoare la utilizarea în detaliu a SGA, de la pornirea instanței
 - Informații de timp nu se colectează pentru evenimente wait când parametrul de configurare STATISTICS_LEVEL are valoarea BASIC (TIME_STATISTICS are valoarea FALSE)

Directive Plan SQL

- O directivă plan SQL reprezintă o informație suplimentară pe care optimizatorul o poate folosi pentru:
 - A colecta statistici lipsă
 - A crea statistici grup de coloane
 - A efectua eșantionare dinamică
- Directivele pot fi folosite pentru mai multe instrucțiuni
 - Sunt colectate pentru expresii de interogare
- Sunt persistente pe disc în tablespace-ul SYSAUX
- Sunt întreținute automat

Planuri de execuție adaptive

- Se permite optimizatorului să adapteze „din mers” planul de execuție al interogării ca urmare a unor estimări inadecvate
- Se bazează pe statistici colectate în timpul execuției
- Există două tehnici:
 - Planuri dinamice – alege între sub-planuri în timpul execuției instrucțiunii
 - Re-optimizare – se modifică planul de execuție după încheierea instrucțiunii curente

Folosire SQL Advisors

- AWR identifică și înregistrează statistici despre instrucțiunile SQL recente de tip high-load
- SQL Access Advisor – ia în considerare modificări efectuate unui set de instrucțiuni SQL și câștigul net asupra performanței
 - Set ipotecic de instrucțiuni
 - Set istoric de instrucțiuni
 - Set de instrucțiuni creat manual
- SQL Tuning Advisor – analizează una sau mai multe instrucțiuni SQL, examinează statistici, profile SQL, indecsi, vederi materializate și SQL restructurat

Folosire SQL Advisors

□ SQL Repair Advisor

- Rulat din Support Workbench când o instrucțiune SQL eșuează cu o eroare critică (ce produce un incident)
- Caută să recomande un SQL patch, pe care dacă nu îl găsește, utilizatorul va apela la Suport Oracle

□ SQL Performance Analyzer

- Prezice și previne potențiale probleme de performanță datorate unei schimbări în mediul BD ce poate afecta structura planurilor de execuție

SQL Tuning Advisor

Comprehensive SQL tuning



SQL Tuning
Advisor

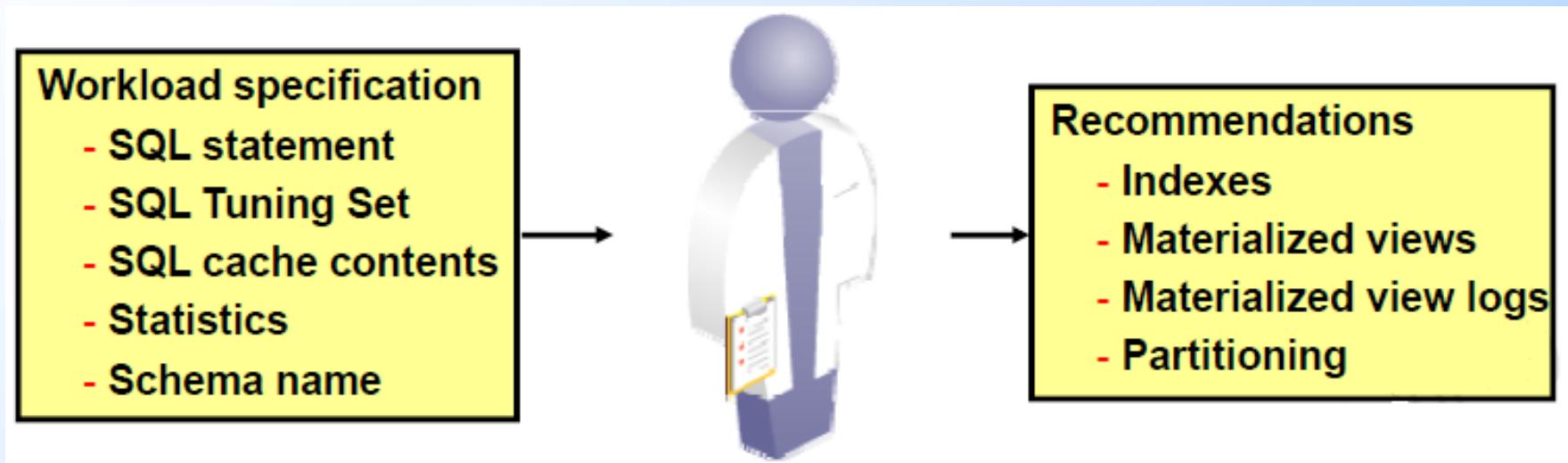
Detect stale or missing
statistics

Tune SQL plan
(SQL profile)

Add missing index

Restructure SQL

SQL Access Advisor



SQL Performance Advisor



1. Capture SQL workload on production.
2. Transport the SQL workload to a test system.
3. Build “before-change” performance data.
4. Make changes.
5. Build “after-change” performance data.
6. Compare results from steps 3 and 5.
7. Tune regressed SQL.

Autorizare

Autorizare

- Un sistem fișier identifică anumite privilegii asupra obiectelor (fișierelor) pe care le gestionează.
 - De obicei read, write, execute.
- Un sistem fișier identifică anumiți参入者 cărora l-i se acordă (grant) privilegiile.
 - De obicei pentru owner, un grup, toți utilizatorii.

Privilegii

- SQL identifică un set mult mai detaliat de privilegii asupra obiectelor (relațiilor).
- Există nouă privilegii în total, unele din ele pot fi restricționate la nivel de coloană, altele la nivel de tabelă.

Privilegii

- Cele mai importante privilegii pentru o relație:
 1. **SELECT** = dreptul de a interoga relația.
 2. **INSERT** = dreptul de a adăuga tuple.
 - Se poate aplica unui singur atribut.
 3. **DELETE** = dreptul de a șterge tuple.
 4. **UPDATE** = dreptul de a modifica tuple.
 - Se poate aplica unui singur atribut.

Exemplu: Privilegii

- Pentru instrucțiunea următoare:

```
INSERT INTO Beers(name)
```

```
    SELECT beer FROM Sells
```

```
WHERE NOT EXISTS  
      (SELECT * FROM Beers  
       WHERE name = beer);
```

berile ce nu apar
în Beers, se adaugă
în Beers cu NULL
pentru manufacturer.

- Este nevoie de privilegii SELECT asupra Sells și Beers, și INSERT asupra Beers sau Beers.name.

Obiecte BD

- Obiectele pentru care există privilegii includ tabele de bază și vederi.
- Alte privilegii sunt dreptul de a crea obiecte de un anumit tip, de exemplu, trigger-e.
- Vederile formează o unealtă importantă pentru controlul accesului.

Exemplu: Vederile și Controlul Accesului

- Se poate să nu se dorească să se acorde privilegiul SELECT asupra **Emps(name, addr, salary)**.
- Mai sigur este să se acorde SELECT asupra:

```
CREATE VIEW SafeEmps AS  
SELECT name, addr FROM Emps;
```

- Interogările pe SafeEmps nu necesită SELECT asupra Emps, ci doar asupra SafeEmps.

ID-uri de Autorizare

- Un utilizator este referit printr-un *ID de autorizare*, de obicei “login name”.
- Există un ID de autorizare PUBLIC.
 - Acordarea unui privilegiu PUBLIC îl face disponibil oricărui ID de autorizare.

Acordarea Privilegiilor

- Pentru obiectele (de exemplu relațiile) create de un utilizator, acesta are toate privilegiile posibile.
- Se pot acorda privilegii altor utilizatori (ID-uri de autorizare), inclusiv PUBLIC.
- Se pot acorda privilegii cu clauza WITH GRANT OPTION, ce permite celui autorizat să acorde mai departe privilegii.

Instrucțiunea GRANT

- Pentru a acorda privilegii:
GRANT <listă de privilegii>
ON <relație sau alt obiect>
TO <listă ID-uri de autorizare>;
- Dacă se dorește ca primitorul să fie capabil să transmită privilegiul(-ile) altora, se adaugă:
WITH GRANT OPTION

Exemplu: GRANT

- Presupunem că utilizatorul curent este proprietar (owner) al Sells.

```
GRANT SELECT, UPDATE(price)  
ON Sells  
TO sally;
```

- După aceasta, Sally are dreptul să emită orice interogare asupra relației Sells și poate modifica prețul.

Exemplu: Opțiunea GRANT

- Se presupune următoarea instrucțiune:

```
GRANT UPDATE ON Sells TO sally  
WITH GRANT OPTION;
```

- În urma acestei instrucțiuni, Sally are dreptul să modifice valoarea oricărui atribut al relației “Sells”, și poate acorda altora privilegiul “UPDATE ON Sells”.
 - De asemenea, Sally poate acorda privilegii mai specifice: UPDATE (price) ON Sells.

Revocarea Privilegiilor

```
REVOKE <listă de privilegii>
ON <relație sau alt obiect>
FROM <listă de ID-uri de autorizare>;
```

- Privilegiile specificate, acordate de utilizatorul curent nu mai pot fi folosite de utilizatorii specificați pentru a justifica utilizarea privilegiului.
- Dar utilizatorii specificați pot avea privilegiul deoarece l-au primit de la altcineva.

Opțiuni REVOKE

- La instrucțiunea REVOKE se atașează una din următoarele:
 1. **CASCADE**. Orice acordare mai departe a privilegiului se revocă pe toată lungimea căii de acordare.
 2. **RESTRICT**. Dacă privilegiul a fost transmis altora, REVOKE eșuează, este o atenționare că trebuie făcute alte acțiuni pentru a îngrădi privilegiul.

Diagrame Grant

- Noduri = utilizator/privilegiu/opțiune
“grant”?/este “owner”?
 - UPDATE ON R, UPDATE(a) on R și UPDATE(b) ON R aparțin la noduri diferite.
 - SELECT ON R și SELECT ON R WITH GRANT OPTION aparțin la noduri diferite.
- Arcul $X \rightarrow Y$ semnifică nodul X a fost folosit pentru “grant” Y .

Notație pentru Noduri

- Se folosește AP pentru nodul ce reprezintă ID-ul de autorizare A cu privilegiul P .
 - P^* = privilegiul P cu opțiunea “grant”.
 - P^{**} = sursa privilegiului P .
 - Adică, A este “owner” pentru obiectul pentru care P este privilegiu.
 - Notă ** implică opțiunea “grant”.

Trasarea Arcelor

- Atunci când A acordă privilegiul P lui B , se trasează un arc de la AP^* la AP^{**} pentru BP .
 - Sau pentru BP^* dacă acordarea privilegiului este cu opțiunea “grant”.
- Dacă A acordă un subprivilegiu Q al lui P (de exemplu UPDATE(a) pentru R dacă P este UPDATE ON R) atunci arcul merge la BQ sau BQ^* .

Trasarea Arcelor

- **Regula fundamentală:** Utilizatorul C are privilegiul Q atât timp cât există o cale de la XP^{**} la CQ , CQ^* , sau CQ^{**} și P este un superprivilegiu al lui Q .
- P poate fi Q și X poate fi C .

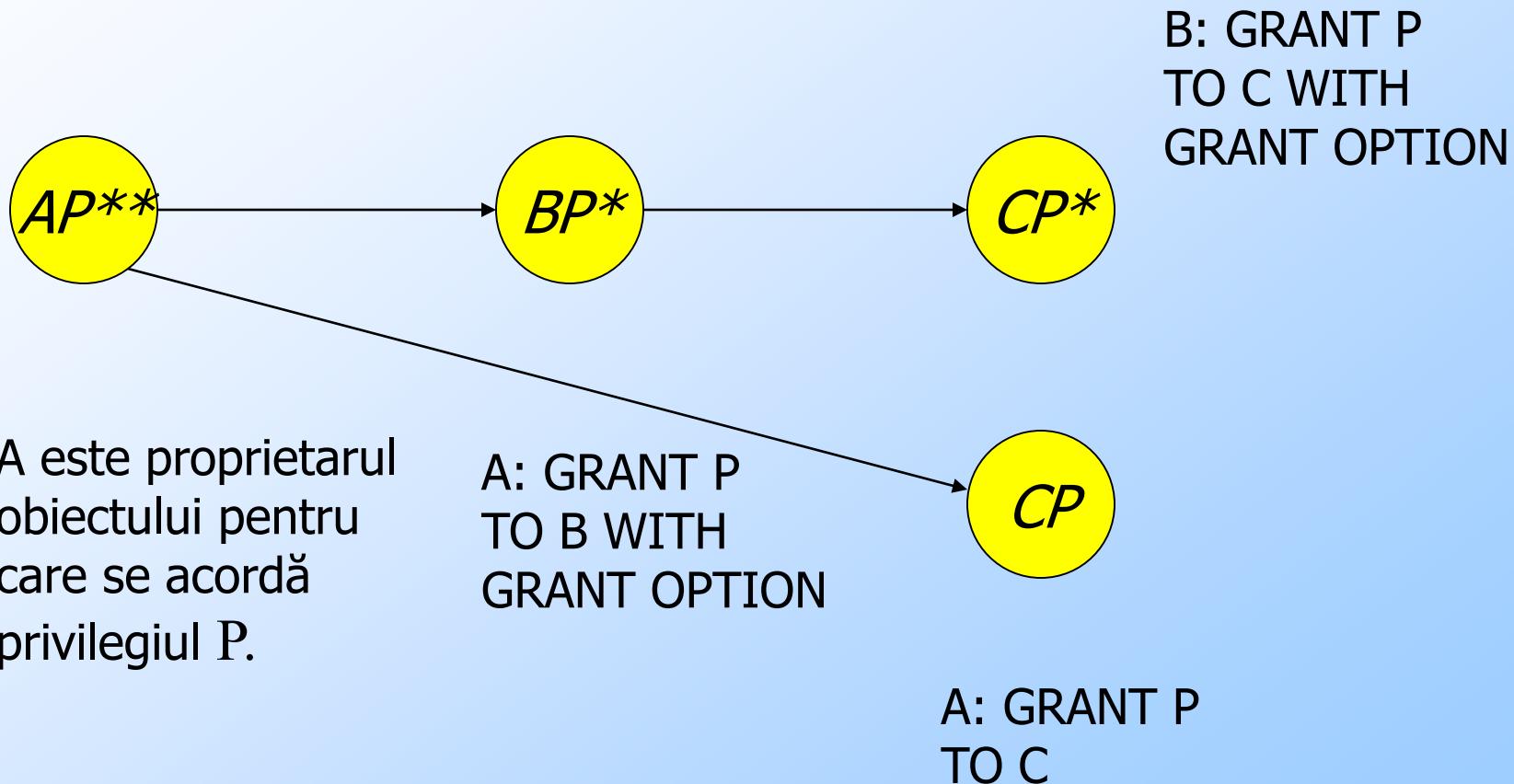
Trasarea Arcelor

- Dacă A revocă P pentru B cu opțiunea CASCADE, se șterge arcul de la AP la BP .
- Dacă A folosește în schimb, RESTRICT și există un arc de la BP în altă parte, atunci se respinge revocarea și nu se produce nici o modificare a grafului.

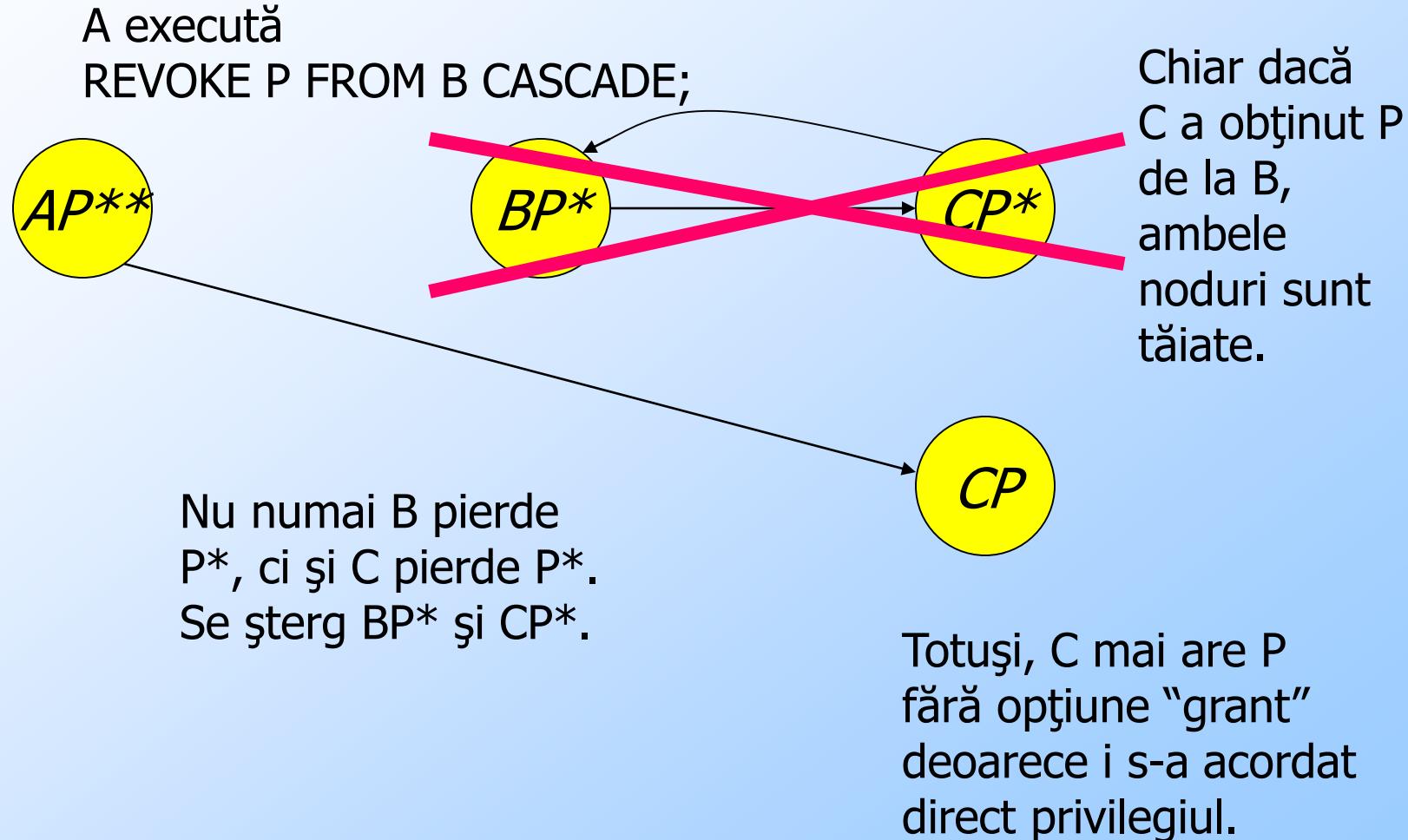
Trasarea Arcelor

- La verificarea arcelor, se testează ca fiecare nod să aibă o cale de la un anume nod **, ce reprezintă proprietarul.
- Orice nod ce nu are o astfel de cale reprezintă un privilegiu revocat și se sterge din diagramă.

Exemplu: Diagramă “Grant”



Exemplu: Diagramă “Grant”



Utilizatori BD Oracle

- Fiecare cont utilizator BD are:
 - Nume de utilizator unic (până la 30 car.)
 - Metodă de autentificare (prin parolă, metode globale și externe cum ar fi metode biometrice, cu certificat sau cu token)
 - Tablespace temporar
 - Profil utilizator
 - Grup inițial de consumatori
 - Stare cont utilizator

Utilizatori BD Oracle

□ Schema

- Colectie de obiecte BD aflate în proprietatea unui utilizator
- Are același nume cu numele utilizatorului
- Pentru a accesa baza de date utilizatorul trebuie să specifice un cont utilizator valid și să se autentifice conform metodei de autentificare
- Pentru tablespace utilizatorul trebuie să aibă privilegiu de acces și quota de spațiu (atribuite separat)

Utilizatori BD Oracle

- Tablespace-ul temporar este folosit pentru sortări și tabele temporare, nu are nevoie de quota
- Profil utilizator înseamnă set de resurse și restricții de parolă atribuite utilizatorului
- Grupul inițial de consumatori este folosit de Resource Manager
- Starea unui cont:
 - Open – doar acești utilizatori pot accesa BD
 - Locked
 - Expired

Utilizatori BD Oracle

- O schemă reprezintă o colecție de obiecte de tip structuri logice ce fac referire la date ținute în BD:
 - Table
 - View
 - Sequence
 - Stored procedure
 - Synonym
 - Index
 - Cluster
 - Database link
- Un utilizator nu înseamnă neapărat o persoană, se practică să fie creat utilizator pentru un grup de obiecte gestionate de o aplicație (de exemplu HR)

Utilizatori BD Oracle

- Utilizatori predefiniți:
 - SYS – proprietarul dictionarului datelor și Automatic Workload Repository (AWR), folosit la startup și shutdown instanță BD
 - SYSTEM – are în proprietate tabele și vederi suplimentare cu caracter de administrare
 - SYSBACKUP – facilitează operații backup și recovery cu Oracle Recovery Manager (RMAN)
 - SYSDG – facilitează operații Oracle Data Guard
 - SYSKM – facilitează operații Transparent Data Encryption wallet
- Recomandări – principiul privilegiilor cele mai puține, administratorul BD să aibă un cont separat, cu privilegii corespunzătoare

Utilizatori BD Oracle

□ Privilegii

- SYSDBA – operații startup, shutdown; creare fișier parametru server SPFILE; modificare mod ARCHIVELOG; permite beneficiarului să vizualizeze datele utilizator
- SYSOPER - operații startup, shutdown; creare fișier parametru server SPFILE; modificare mod ARCHIVELOG
- SYSBACKUP – operații backup și recovery folosind RMAN sau splplus
- SYSDG – operații data guard folosind Data Guard Broker sau interfață linie de comandă DGMGRL
- SYSKM – gestionează operații wallet Transparent Data Encryption (gestiunea cheilor)

Utilizatori BD Oracle

□ Protectia utilizatorilor privilegiati

- Conectare locală – protecția este asigurată de SO prin apartenența la un grup de utilizatori privilegiati SO
- Conectare la distanță – protecția este asigurată printr-un fișier cu parole (case sensitive) sau cu opțiunea Advanced Security dacă este nevoie de protecție mai puternică

□ Autentificarea utilizatorilor

- Cu parolă – la log in se solicită parola
- Externă – metoda de autentificare este externă BD (SO, Kerberos, Radius)
- Globală – identificare utilizatori printr-un serviciu director LDAP

Utilizatori BD Oracle

- Privilegiu – dreptul de a executa un tip de instrucțiune SQL sau de a accesa un obiect al altui utilizator
- Privilegii sistem
 - Permit efectuarea unei operații specifice cu BD (exemplu – creare tablespace)
 - Acordate de administrator sau de cineva care are permisiunea de a administra privilegiul
 - Există peste 170 privilegii sistem
- Privilegii obiect
 - Pot fi acordate de proprietarul obiectului, de administrator sau de cineva care deține privilegiul, cu GRANT OPTION

Utilizatori BD Oracle

□ Rol

- Este folosit pentru a grupa împreună privilegii și roluri
- Facilitează acordarea de privilegii în grup, utilizatorilor

□ Beneficii ale utilizării Rol

- Ușurează administrarea privilegiilor – în loc să se acorde aceleasi privilegii de mai multe ori utilizatorilor, se creează rol, se acordă privilegii rolului și se acordă utilizatorilor grant asupra rolului
- Administrare dinamică a privilegiilor – dacă un privilegiu asociat cu un rol se modifică, toți utilizatorii cu grant asupra rolului obțin modificarea
- Disponibilitate selectivă asupra privilegiilor – rolurile pot fi enable sau disable astfel încât privilegiile să fie on sau off temporar

Utilizatori BD Oracle

- Roluri predefinite și privilegii incluse:
 - **CONNECT** – privilegiu de a crea sesiune
 - **DBA** – majoritatea privilegiilor sistem și alte roluri
 - **RESOURCE** – creare cluster, creare indextype, creare operator, creare procedură, creare secvență, creare tabelă, creare trigger, creare type
 - **SCHEDULER_ADMIN** – CREATE ANY JOB, CREATE EXTERNAL JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
 - **SELECT_CATALOG_ROLE** – privilegii SELECT pe obiecte din dicționarul datelor

Utilizatori BD Oracle

□ Profiluri

- Controlează consumul resurselor (RESOURCE_LIMIT trebuie să aibă valoarea TRUE)
- Administrează starea contului și expirarea parolei
- Un utilizator poate avea un singur profil la un moment dat
- Limitările pot fi precizate în CPU/sesiune (în sutimi de secundă), CPU/apel sau pot face referire la profilul DEFAULT
- Exemplu: CPU/sesiune = 1000 înseamnă că o sesiune cu acest profil dacă folosește mai mult de 10 secunde CPU recepționează un mesaj de eroare și se efectuează log off

Utilizatori BD Oracle

□ Limitări de rețea/memorie

- Connect Time (în minute)
- Idle Time (în minute pentru procesul server)
- Concurrent Sessions (număr sesiuni ce folosesc același cont)
- Private SGA (spațiul consumat în SGA de sortări, etc. doar în cazul în care sesiunea folosește server partajat)

□ Limitări disc I/O

- Se limitează cantitatea de date pe care un utilizator poate să o citească la nivel de sesiune sau la nivel apel
- Se referă atât la memoria internă cât și la memoria externă

□ Există și limitări compuse (combinații din cele de mai sus)

Utilizatori BD Oracle

□ Atribuirি quota la utilizatori

- Utilizatorii care nu au privilegiul sistem UNLIMITED TABLESPACE au nevoie să li se acorde quota pentru tablespace pentru a fi posibil să creeze obiecte stocate în acel tablespace (se referă la owner)

□ Tipuri quota

- O valoare specificată în MB sau KB
- Nelimitată

□ Nu se accordă quota tablespace-urilor SYSTEM sau SYSAUX (doar utilizatorii SYS și SYSTEM au dreptul de a crea obiecte în aceste două tablespace-uri)

□ Nu se accordă quota tablespace-urilor temporare sau undo

Utilizatori BD Oracle

- Serverul BD verifică quota când un utilizator creează sau extinde un segment
- Activitățile ce au nevoie de quota sunt cele care folosesc spațiu în tablespace
 - Creare de vederi sau activități ce folosesc tablespace temporar NU au nevoie de quota
- Quota este revizuită atunci când obiectele deținute de utilizator sunt eliminate din schemă (drop) cu clauza PURGE sau dacă obiectele se află în recycle bin și sunt eliminate de acolo

Utilizatori BD Oracle

□ Aplicarea principiului cele mai puține privilegii

- Se protejează dicționarul datelor

O7_DICTIONARY_ACCESIBILITY = FALSE

- Se revocă privilegii neneceare de la PUBLIC
- Se folosesc liste de control acces (ACL) pentru a controla accesul la rețea
- Se restricționează directoarele accesibile utilizatorilor
- Se limitează utilizatorii cu privilegii de administrare
- Se restricționează autentificarea de la distanță la BD

REMOTE_OS_AUTHENT = FALSE

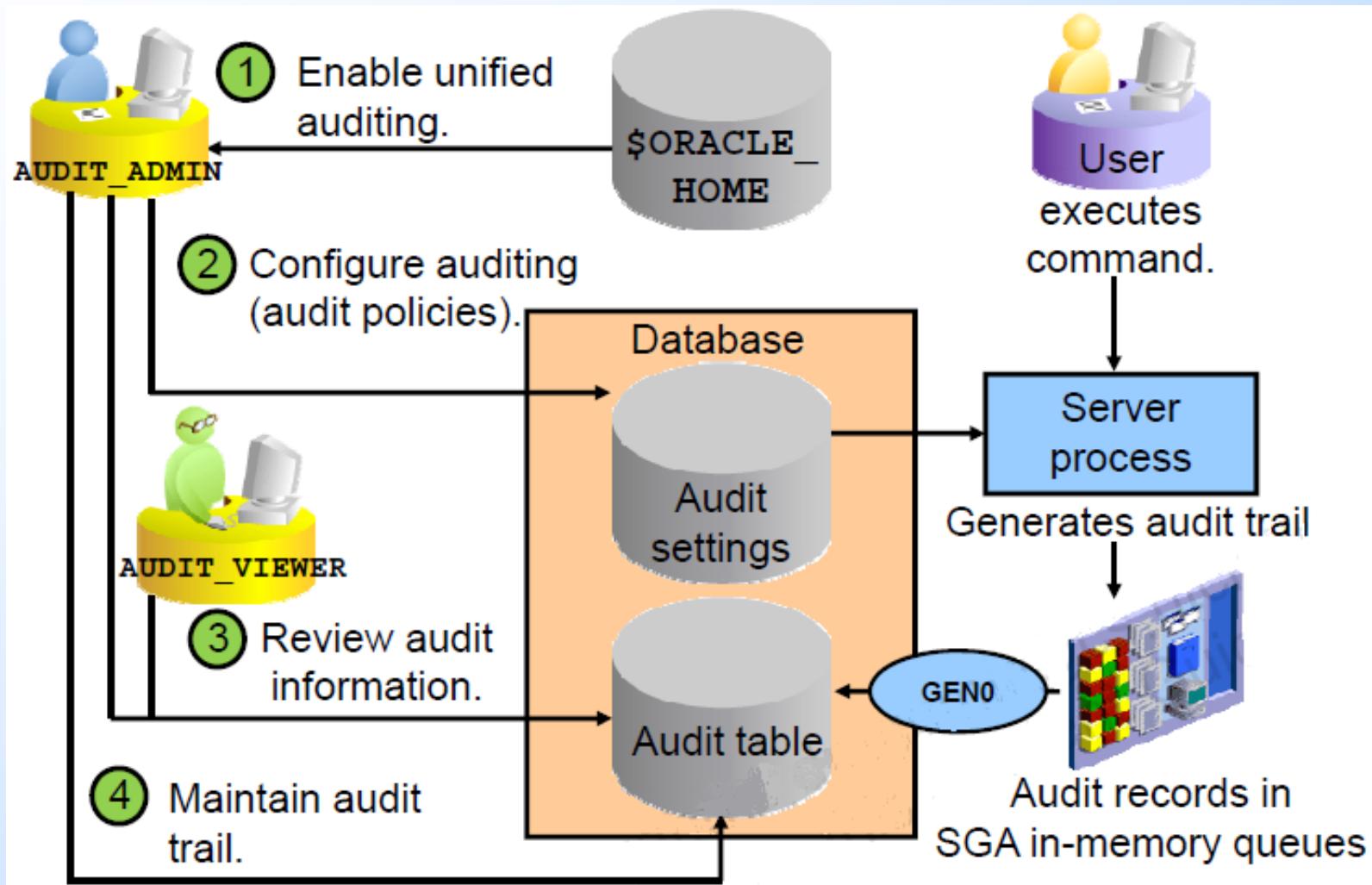
Oracle, implementare auditare

- Auditare = capturare și stocare informație despre ceea ce se întâmplă în sistem
 - Audit obligatoriu – auditare acțiuni indiferent de opțiuni de audit sau parametrii (de exemplu conectări ale utilizatorilor privilegiați)
 - Audit BD standard – sunt selectate obiectele și privilegiile ce se dorește a fi auditate și se creează politicile de audit potrivite
 - Auditare bazată pe valoare – extinde auditarea standard prin capturare pe lângă eveniment, a valorilor actualizate (se implementează cu trigere)
 - Fine-grained auditing (FGA) - extinde auditarea standard prin capturare instrucțiune SQL

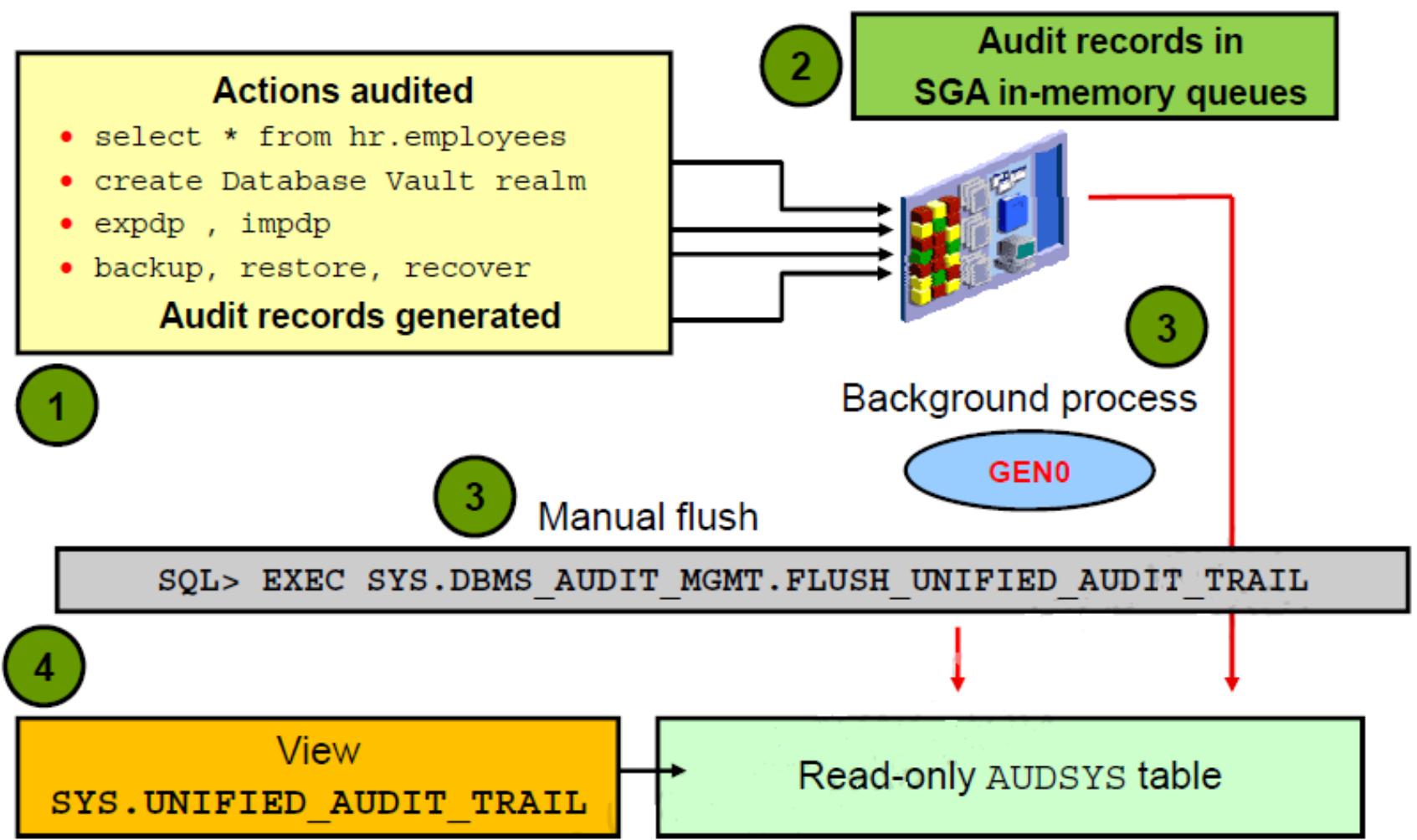
Oracle, implementare auditare

- Activități auditate
 - Conturi utilizator, roluri și privilegii
 - Acțiuni cu obiecte
 - Valori de context aplicație
 - Oracle Data Pump
 - Oracle Database Real Application Security
 - Oracle Database Vault
 - Oracle Label Security
 - Oracle Recovery Manager
 - Evenimente direct path Oracle SQL*Loader

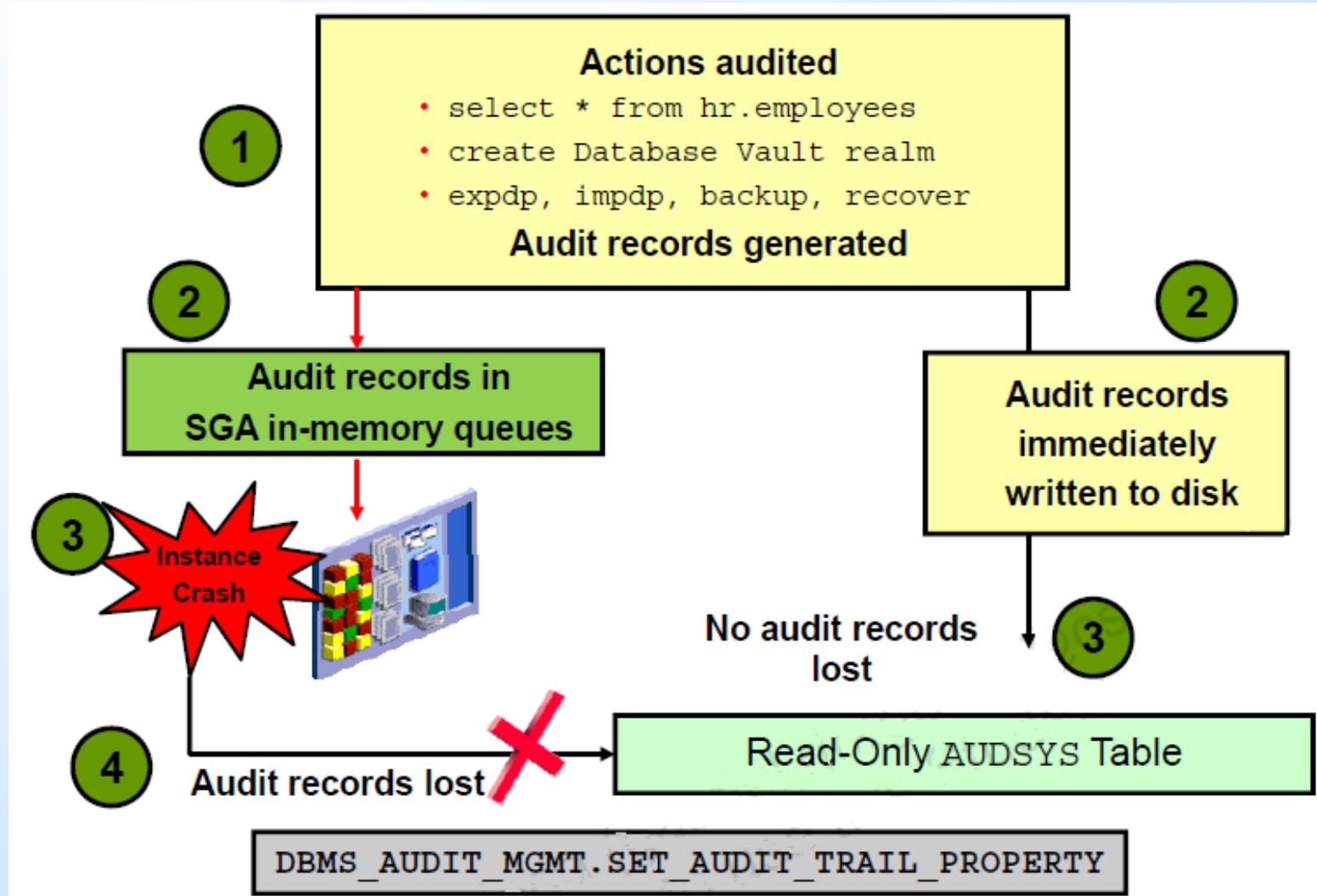
Auditare BD



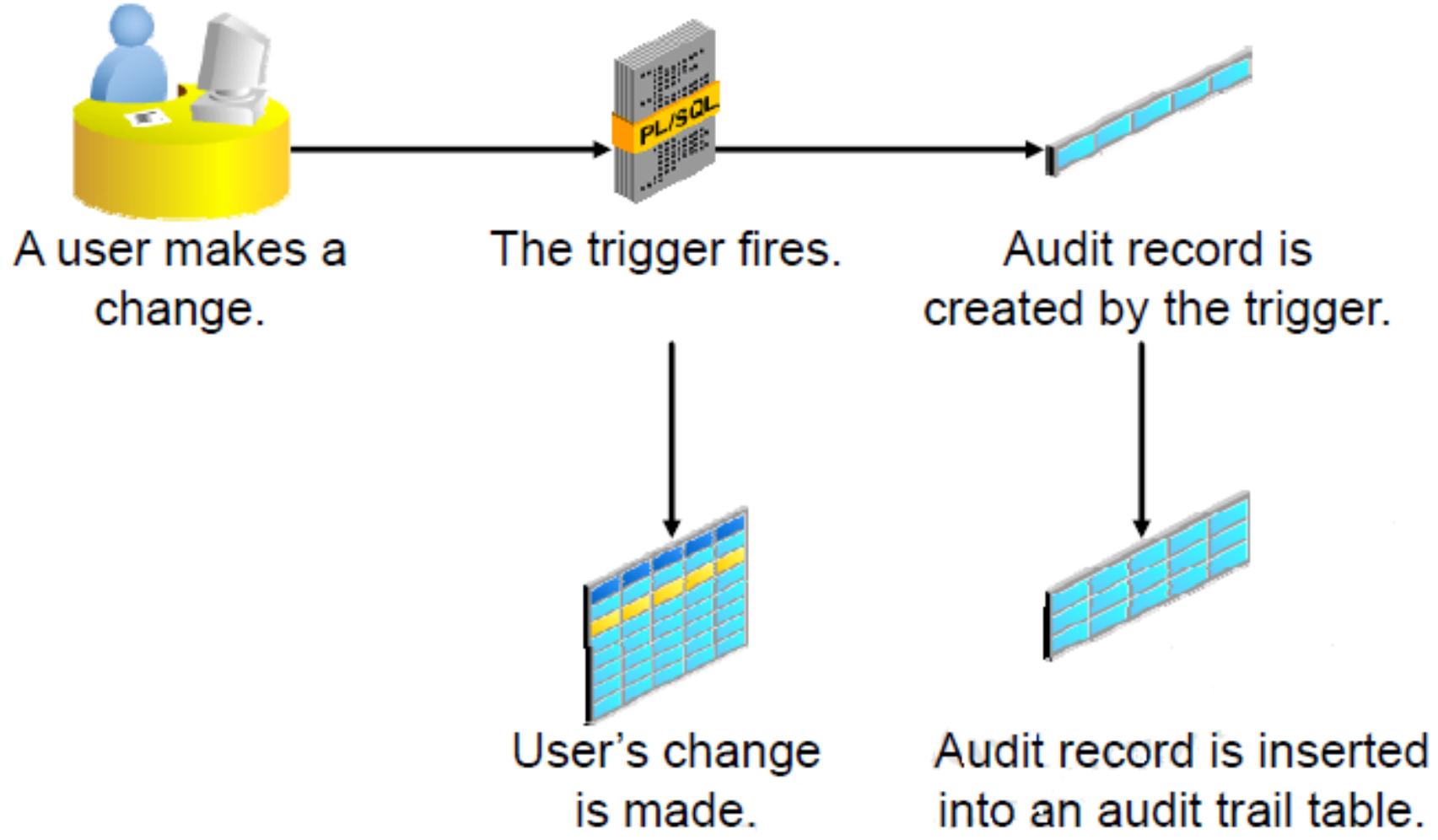
Arhitectura de audit



Setare mod de scriere pentru înregistrările de audit



Auditare bazată pe valoare



Tranzacții

Tranzacții - Motivație

- Sistemele de BD sunt accesate în mod normal de mai mulți utilizatori sau procese la același moment de timp.
 - Atât interogări cât și actualizări.
- Spre deosebire de sistemele de operare, ce *susțin* interacțiunea proceselor, un SGBD are nevoie de supervizare a proceselor contra interacțiunilor ce cauzează probleme.

Exemplu: Interacțiune cu probleme

- Tatăl și fiul posedă carduri bancare pentru același cont din bancă.
- Fiecare scoate de la ATM-uri diferite 100 LEI, în același timp.
 - SGBD-ul trebuie să asigure să nu se piardă nici una din operațiile asupra contului.
- **Comparație:** Un SO permite ca două persoane să editeze un document în același timp. Dacă ambele scriu, modificările efectuate de una din persoane se pierde.

Tranzacții

- *Tranzacție* = un proces ce implică interogări și/sau actualizări ale BD.
- În mod normal există proprietăți puternice cu privire la concurență.
- În SQL este formată din instrucțiuni singulare sau control explicit de programare.

Tranzacții ACID

- *Tranzacțiile ACID* au proprietățile:
 - *Atomicitate* : “Totul sau nimic”.
 - *Consistență* : Constrângerile BD să fie respectate.
 - *Izolare* : Utilizatorul vede ca și cum la un moment dat de timp se execută un singur proces.
 - *Durabilitate* : Efectele unui proces “supraviețuiesc” unei căderi a sistemului.
- **Optional:** deseori sunt susținute forme mai slabe de tranzacții.

COMMIT

- Instrucțiunea SQL COMMIT cauzează încheierea cu succes a unei tranzacții.
- Modificările asupra BD devin permanente.

ROLLBACK

- Instrucțiunea SQL ROLLBACK cauzează terminarea tranzacției, dar cu *abandonare*.
 - Nu există efecte în BD.
- Eșuările ca de exemplu împărțirea la 0 sau violarea unei constrângeri pot cauza rollback, chiar dacă programatorul nu a apelat-o.

Exemplu: Procese ce Interacționează

- Presupunem relația **Sells(bar,beer,price)** și faptul că “Joe’s Bar” vinde numai “Bud” la 2.50 (\$) și “Miller” la 3.00 (\$).
- Sally interrogează **Sells** pentru a afla prețul cel mai mare și cel mai mic cu care vinde Joe.
- Joe decide să nu mai vândă Bud și Miller, numai Heineken la 3.50 (\$).

Program lui Sally

- Sally execută următoarele două instrucțiuni SQL numite **(min)** și **(max)**:
 - (max)** SELECT MAX(price) FROM Sells
 WHERE bar = 'Joe''s Bar';
 - (min)** SELECT MIN(price) FROM Sells
 WHERE bar = 'Joe''s Bar';

Programul lui Joe

- Aproximativ în același timp, Joe execută următorii pași: **(del)** și **(ins)**.

(del) DELETE FROM Sells

WHERE bar = 'Joe''s Bar';

(ins) INSERT INTO Sells

VALUES('Joe''s Bar', 'Heineken', 3.50);

Intercalarea Instrucțiunilor

- Deși (max) trebuie să apară înaintea lui (min), și (del) trebuie să apară înaintea lui (ins), nu există alte constrângeri asupra ordinii acestor instrucțiuni, doar dacă se grupează instrucțiunile lui Sally și/sau instrucțiunile lui Joe în tranzacții.

Exemplu: Intercalare

- Presupunem pașii următori de execuție:
(max) (del) (ins) (min).

Prețurile lui Joe: {2.50,3.00}{2.50,3.00} {3.50}

Instructiune: (max) (del) (ins) (min)

Rezultat: 3.00 3.50

- Sally observă MAX < MIN!

Rezolvarea Problemei prin Utilizare de Tranzacții

- Dacă se grupează instrucțiunile lui Sally **(max)(min)** într-o tranzacție, ea nu poate vedea această inconsistență.
- Ea vede prețurile lui Joe la un moment fix de timp.
 - Fie înainte, fie după modificarea prețurilor, sau la mijloc, dar MAX și MIN sunt calculate pe aceleași prețuri.

O altă Problemă: Rollback

- Presupunem că Joe execută **(del)(ins)**, nu ca o tranzacție și după execuția acestor instrucțiuni, se gândește să execute instrucțiunea ROLLBACK.
- Dacă Sally execută instrucțiunile proprii după **(ins)** dar înainte de rollback, ea vede o valoare, 3.50, ce nu a existat niciodată în BD.

Soluția

- Dacă Joe execută **(del)(ins)** ca o tranzacție, efectele acesteia nu pot fi văzute de alții până ce tranzacția execută COMMIT.
- Dacă tranzacția execută ROLLBACK, atunci efectele nu pot fi *niciodată* văzute.

Nivele de Izolare

- SQL definește patru *nivele de izolare* = opțiuni în legătură cu ce interacțiuni le sunt permise tranzacțiilor ce se execută în același timp.
- Doar un singur nivel ("serializable") = tranzacții ACID.
- Fiecare SGBD implementează tranzacțiile în mod propriu.

Alegerea Nivelului de Izolare

- În cadrul unei tranzacții, se poate preciza:

SET TRANSACTION ISOLATION LEVEL X

unde X =

1. SERIALIZABLE
2. REPEATABLE READ
3. READ COMMITTED
4. READ UNCOMMITTED

Tranzacții Serializabile

- Dacă Sally = **(max)(min)** și Joe = **(del)(ins)** sunt fiecare tranzacții, și Sally execută cu nivelul de izolare SERIALIZABLE, atunci ea va vedea BD fie înainte, fie după ce Joe execută, dar nu la mijloc.

Nivelul de Izolare este o Alegere Personală

- Optiunea aleasă de un utilizator, de exemplu “run serializable”, afectează doar felul în care *acel utilizator* vede BD, nu felul în care o văd alții.
- **Exemplu:** Dacă Joe execută “serializable”, dar Sally nu, atunci Sally se poate să nu vadă prețuri pentru “Joe’s Bar”.
 - adică, Sally observă ca și cum ar fi executat la mijlocul tranzacției lui Joe.

Tranzacții Read-Committed

- Dacă Sally execută cu nivelul de izolare READ COMMITTED, atunci ea vede doar datele “committed”, dar nu neapărat aceleși date de fiecare dată.
- **Exemplu:** Cu READ COMMITTED, intercalarea **(max)(del)(ins)(min)** este permisă atât timp cât Joe face “commit”.
 - Sally vede MAX < MIN.

Tranzacții Repeatable-Read

- Cerința este ca și la read-committed, plus: dacă datele sunt citite din nou, atunci tot ceea ce a fost văzut prima dată este văzut a doua oară.
 - Dar a doua oară și citiri ulterioare pot vedea *mai multe* tuple.

Exemplu: Repeatable Read

- Presupunem că Sally execută cu REPEATABLE READ, și ordinea de execuție este **(max)(del)(ins)(min)**.
 - **(max)** vede prețurile 2.50 și 3.00.
 - **(min)** poate vedea 3.50, dar de asemenea vede 2.50 și 3.00, deoarece aceste prețuri au fost văzute la prima citire de **(max)**.

Read Uncommitted

- O tranzacție ce se execută cu READ UNCOMMITTED poate vedea datele din BD, chiar dacă nu au fost scrise de o tranzacție ce nu a efectuat committed (și poate niciodată).
- **Exemplu:** Dacă Sally execută cu READ UNCOMMITTED, ea poate vedea prețul 3.50 chiar dacă Joe va abandona mai târziu.

XML

“Document Type Definition”
(DTD)

Schemă XML

Document XML “Well-Formed”

- *XML Well-Formed* permite folosirea tagurilor proprii.
- *XML Valid* se conformează unui anumit DTD.

XML “Well-Formed”

- Documentul începe cu o *declarație*, delimitată de <?xml ... ?> .

- Exemplu de declarație:

```
<?xml version = "1.0"  
standalone = "yes" ?>
```

- “standalone” = “fără DTD”

- Un document are un *tag root* ce înconjoară tag-uri imbricate.

Tag-uri

- Tag-urile sunt perechi de etichete:

<FOO> ... </FOO>

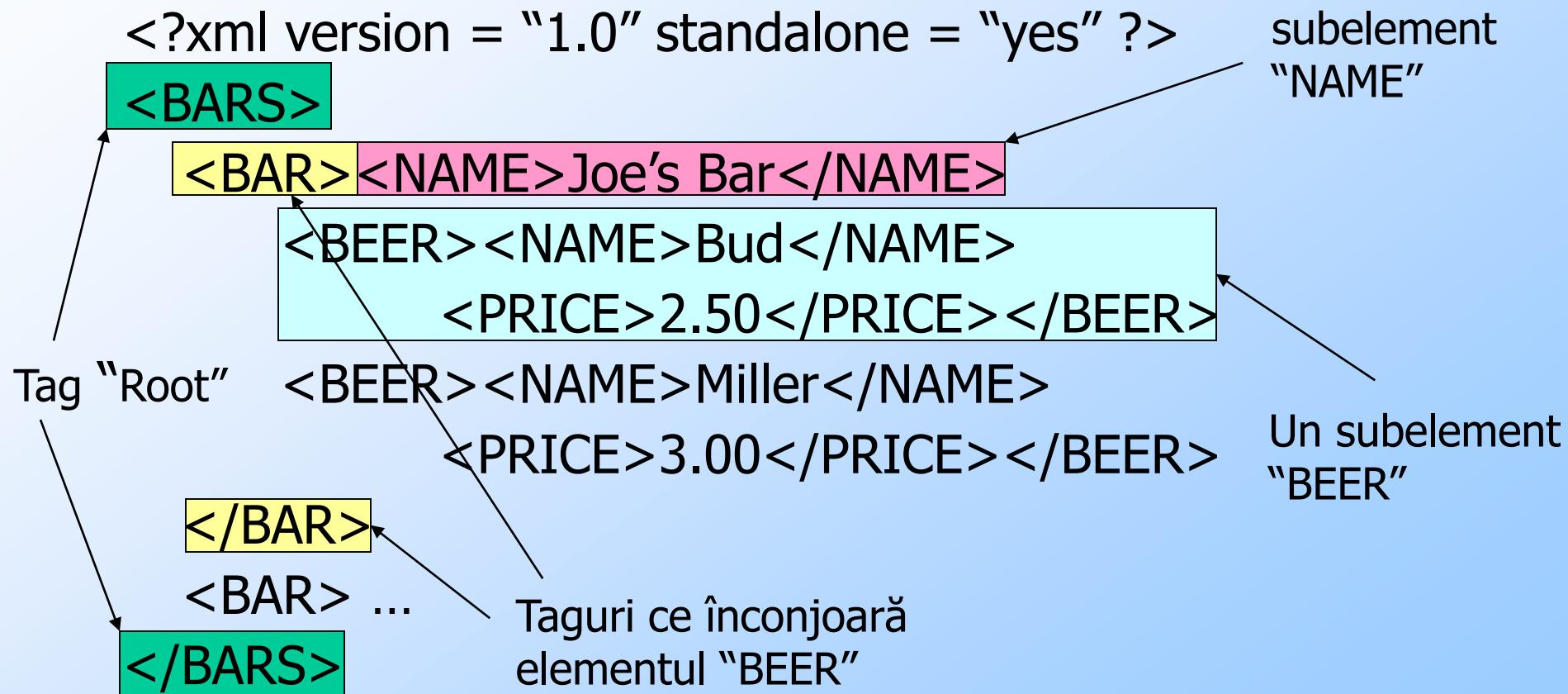
- Sunt permise tag-uri singulare:

<FOO/>

- Tag-urile pot fi imbricate în mod arbitrar.

- Tag-urile XML sunt “case-sensitive”.

Exemplu: XML “Well-Formed”



Structura DTD

```
<!DOCTYPE <tag root> [
    <!ELEMENT <nume> (<componente>) >
    . . . alte elemente . . .
] >
```

Elemente DTD

- Descrierea unui element constă din nume (tag) și o descriere între paranteze a tag-urilor imbricate.
 - Include succesiunea subtag-urilor și multiplicitatea lor.
- Frunzele (elemente text) au #PCDATA (*Parsed Character DATA*) în loc de tag-uri imbricate.

Exemplu: DTD

```
<!DOCTYPE BARS [  
    <!ELEMENT BARS (BAR*)>  
    <!ELEMENT BAR (NAME, BEER+)>  
    <!ELEMENT NAME (#PCDATA)>  
    <!ELEMENT BEER (NAME, PRICE)>  
    <!ELEMENT PRICE (#PCDATA)>  
]>
```

Un obiect "BARS" are zero sau mai multe BAR-uri imbricate.

Un BAR are un NAME și unul sau mai multe subobiecte BEER.

BEER are NAME și PRICE.

NAME și PRICE sunt text.

Descrieri Element

- Subtag-urile trebuie să apară în ordinea prezentată.
- Un tag poate fi urmat de un simbol ce indică multiplicitatea.
 - * = zero sau mai multe.
 - + = unu sau mai multe.
 - ? = zero sau unu.
- Simbolul | poate conecta secvențe alternative de tag-uri.

Exemplu: Descriere Element

- Un nume are un titlu optional (de exemplu, "Prof."), un prenume și numele de familie, în această ordine, sau poate fi o adresă IP:

```
<!ELEMENT NUME (  
    TITLU?, PRENUME, NUME) | ADRIIP  
>
```

Folosirea DTD-uri

1. Se specifică standalone = “no”.
2. și fie:
 - a) Se include DTD ca prefață a documentului XML, sau
 - b) În continuarea DOCTYPE și a <tag-ului root> se specifică SYSTEM și calea (“path”) către fișierul ce conține DTD.

Exemplu: (a)

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE BARS [  
    <!ELEMENT BARS (BAR*)>  
    <!ELEMENT BAR (NAME, BEER+)>  
    <!ELEMENT NAME (#PCDATA)>  
    <!ELEMENT BEER (NAME, PRICE)>  
    <!ELEMENT PRICE (#PCDATA)>  
>]
```

DTD

```
<BARS>  
    <BAR><NAME>Joe's Bar</NAME>  
        <BEER><NAME>Bud</NAME> <PRICE>2.50</PRICE></BEER>  
        <BEER><NAME>Miller</NAME> <PRICE>3.00</PRICE></BEER>  
    </BAR>  
    <BAR> ...  
</BARS>
```

Documentul XML

Exemplu: (b)

- Presupunem că DTD pentru BARS se găsește în fișierul bar.dtd.

```
<?xml version = "1.0" standalone = "no" ?>
<!DOCTYPE BARS SYSTEM "bar.dtd">
<BARS>
    <BAR><NAME>Joe's Bar</NAME>
        <BEER><NAME>Bud</NAME>
            <PRICE>2.50</PRICE></BEER>
        <BEER><NAME>Miller</NAME>
            <PRICE>3.00</PRICE></BEER>
    </BAR>
    <BAR> ...
</BARS>
```

De aici se știe că DTD se găsește în fișierul bar.dtd

Atribute

- Tag-urile XML pot avea *atribute*.
- Într-un DTD,

```
<!ATTLIST E . . . >
```

declară attributele elementului *E*,
împreună cu tipurile de date.

Exemplu: Atribute

- Bar-urile pot avea un atribut "kind", un sir de caractere ce descrie barul.

```
<!ELEMENT BAR (NAME BEER*)>
```

```
<!ATTLIST BAR kind CDATA  
      #IMPLIED>
```

Atributul este optional
opus: #REQUIRED

tipul sir de
caractere;
fară tag-uri

Exemplu: Folosire Atribut

- Într-un document ce acceptă tag-uri “BAR”, poate exista:

```
<BAR kind = "sushi">  
    <NAME>Homma's</NAME>  
    <BEER><NAME>Sapporo</NAME>  
        <PRICE>5.00</PRICE></BEER>  
    ...  
</BAR>
```

ID-uri și IDREF-uri

- Atributele pot fi pointeri de la un obiect la altul.
 - Comparabil cu HTML: NAME = "foo" și HREF = "#foo".
- Permite structurii unui document XML să fie un graf, în loc de arbore.

Crearea ID-urilor

- Presupunem un element E cu un atribut A de tip ID.
- Atunci când se folosește tag-ul $\langle E \rangle$ într-un document XML, atributul A primește o valoare unică.
- Exemplu:

```
<E A = "xyz">
```

Crearea IDREF-urilor

- Pentru a permite elementelor de tip F să facă referire la un alt element cu un atribut ID, F primește un atribut de tip IDREF.
- Sau, atributul are tipul IDREFS, astfel încât elementul F poate face referire la oricâte alte elemente.

Exemplu: ID-uri și IDREF-uri

- Un DTD nou pentru BARS include atât subelemente BAR cât și BEER.
- BARS și BEERS au atribut ID name.
- BARS au subelemente SELLS, ce constau din un număr (prețul unei beri) și IDREF-ul theBeer ce conduce la acea marcă de bere.
- BEERS au atributul soldBy, ce este un IDREFS ce conduce la toate barurile ce vând berea respectivă.

DTD

```
<!DOCTYPE BARS [  
    <!ELEMENT BARS (BAR*, BEER*)>  
    <!ELEMENT BAR (SELLS+)>  
        <!ATTLIST BAR name ID #REQUIRED>  
    <!ELEMENT SELLs (#PCDATA)>  
        <!ATTLIST SELLs theBeer IDREF #REQUIRED>  
    <!ELEMENT BEER EMPTY>  
        <!ATTLIST BEER name ID #REQUIRED>  
        <!ATTLIST BEER soldBy IDREFS #IMPLIED>  
]>
```

Se va explica în continuare

Elementele BAR au numele ca un atribut ID și au unul sau mai multe subelemente SELLs.

Elementele SELLs au un număr (prețul) și o referință la beer.

Elementele BEER au un atribut ID denumit name, și un atribut soldBy ce este un set de nume de BAR-uri.

Exemplu: Document

```
<BARS>
  <BAR name = "JoesBar">
    <SELLS theBeer = "Bud">2.50</SELLS>
    <SELLS theBeer = "Miller">3.00</SELLS>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ..." /> ...
</BARS>
```

Elemente “Empty”

- Un element poate fi descris prin attribute.
 - În exemplul anterior BEER.
- Un alt exemplu: elementele SSELLS ar fi putut avea atributul price în locul valorii ce reprezintă “price”.

Exemplu: Element “Empty”

- În DTD se declară:

```
<!ELEMENT SELLS EMPTY>
```

```
<!ATTLIST SELLS theBeer IDREF #REQUIRED>
```

```
<!ATTLIST SELLS price CDATA #REQUIRED>
```

- Exemplu de utilizare:

```
<SELLS theBeer = "Bud" price = "2.50" />
```

De notat excepția la
regula “tag-uri pereche”

Schemă XML

- Oferă o cale mai puternică pentru a descrie structura documentelor XML.
- Declarațiile Schemă-XML sunt ele însese documente XML .
 - Ele descriu “elemente” iar descrierea se face de asemenea cu “elemente”.

Structura unui Document Schemă-XML

```
<? xml version = ... ?>  
<xs:schema xmlns:xs =  
    "http://www.w3.org/2001/XMLSchema">  
    . . .  
</xs:schema>
```

Folosirea "xs" la definirea elementelor schemei se referă la tag-uri din acest namespace.

Se definește "xs" ca fiind *namespace* și este descris prin URL-ul respectiv. Orice sir de caractere poate lua locul "xs".

Elementul xs:element

- Are atributele:
 - **name** = numele tag-ului elementului definit.
 - **type** = tipul elementului.
 - Poate fi un tip Schemă-XML , de exemplu, xs:string.
 - Sau numele tipului definit în document.

Exemplu: xs:element

```
<xs:element name = "NUME"  
           type = "xs:string" />
```

□ Descrie elemente ca de exemplu:

```
<NUME>Pop Ion</NUME>
```

Tipuri Complexe

- Pentru a descrie elemente ce constau din subelemente, se folosește **xs:complexType**.
 - Atributul **name** precizează numele tipului.
- Un subelement tipic al unui tip complex este **xs:sequence**, ce conține o secvență **xs:element** de subelemente.
 - Se folosesc attributele **minOccurs** și **maxOccurs** pentru a controla numărul aparițiilor unui **xs:element**.

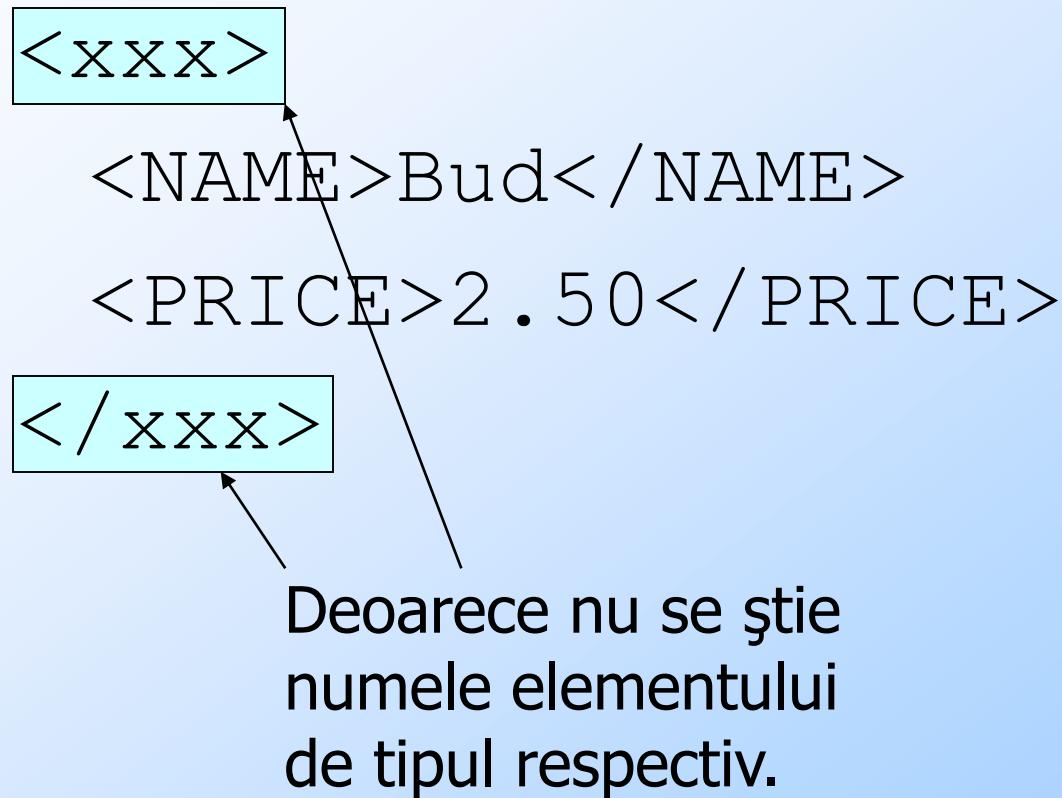
Exemplu: Tip pentru Beers

```
<xs:complexType name = "beerType">
  <xs:sequence>
    <xs:element name = "NAME"
      type = "xs:string"
      minOccurs = "1" maxOccurs = "1" />
    <xs:element name = "PRICE"
      type = "xs:float"
      minOccurs = "0" maxOccurs = "1" />
  </xs:sequence>
</xs:complexType>
```

Exact 1
apariție

Asemănător
cu ? la DTD

Un Element de Tip “beerType”



Exemplu: un Tip pentru Bars

```
<xs:complexType name = "barType">
  <xs:sequence>
    <xs:element name = "NAME"
      type = "xs:string"
      minOccurs = "1" maxOccurs = "1" />
    <xs:element name = "BEER"
      type = "beerType"
      minOccurs = "0" maxOccurs =
      "unbounded" />
  </xs:sequence>
</xs:complexType>
```

Asemănător cu * la DTD

xs:attribute

- Elementele **xs:attribute** pot fi folosite în cadrul unui tip complex pentru a indica atributele elementelor tipului respectiv.
- Atribute **xs:attribute**:
 - name și type ca și la **xs:element**.
 - use = "required" sau "optional".

Exemplu: xs:attribute

```
<xs:complexType name = "beerType">
  <xs:attribute name = "name"
    type = "xs:string"
    use = "required" />
  <xs:attribute name = "price"
    type = "xs:float"
    use = "optional" />
</xs:complexType>
```

Un Element de Tipul beerType

```
<xxx name = "Bud"  
      price = "2.50"
```

Pentru că nu se cunoaște numele elementului.

```
/>
```

Elementul este “empty”, deoarece nu sunt declarate subelemente.

Tipuri Simple Restrictionate

- **xs:simpleType** poate descrie enumerări și tipuri de bază 'plaje de valori'.
- **name** este un atribut
- **xs:restriction** este un subelement.

Restrictii

- Atributul **base** specifică tipul simplu ce va fi restrictionat, de exemplu, **xs:integer**.
- **xs:{min, max}{Inclusive, Exclusive}** sunt patru atribute ce specifică limita inferioară și superioară a unei plaje de valori numerice.
- **xs:enumeration** este un subelement cu atributul **value** ce permite tipuri enumerare.

Exemplu: Attributul **license** pentru BAR

```
<xs:simpleType name = "license">
  <xs:restriction base = "xs:string">
    <xs:enumeration value = "Full" />
    <xs:enumeration value = "Beer only" />
    <xs:enumeration value = "Sushi" />
  </xs:restriction>
</xs:simpleType>
```

Exemplu: Prețuri în Plaja [1,5)

```
<xs:simpleType name = "price">  
  <xs:restriction  
    base = "xs:float"  
    minInclusive = "1.00"  
    maxExclusive = "5.00" />  
</xs:simpleType>
```

Chei în Schema XML

- Un xs:element poate avea un subelement xs:key.
- Semnificația: pentru acest element, toate subelementele la care se ajunge cu un anumit *selector* de cale ("path") vor avea valori unice pentru o anumită combinație de *câmpuri*.
- Exemplu: pentru un element BAR, atributul name al elementului BEER este unic.

Exemplu: Cheie

```
<xs:element name = "BAR" ... >  
  . . .  
<xs:key name = "barKey">  
  <xs:selector xpath = "BEER" />  
  <xs:field xpath = "@name" />  
</xs:key>  
  . . .  
</xs:element>
```

@ indică un atribut și nu un tag.

XPath este un limbaj de interogare pentru XML. Singurul lucru ce este nevoie să fie cunoscut pentru moment este că o cale ("path") este o secvență de tag-uri separate de /.

Chei Străine

- Un subelement **xs:keyref** pentru un **xs:element** precizează că pentru acest element, anumite valori (definite prin un selector și câmp(uri), ca și la chei) trebuie să apară ca și valori ale unei chei.

Exemplu: Cheie Străină

- Presupunem că s-a declarat că subelementul NAME al BAR este o cheie pentru BARS.
 - Numele cheii este barKey.
- Se dorește să se declare că elementele DRINKER au subelementele FREQ. Un atribut **bar** al FREQ este cheie străină, ce referă către NAME al BAR.

Exemplu: Cheie Străină în Schema XML

```
<xs:element name = "DRINKERS"
    . . .
<xs:keyref name = "barRef"
    refers = "barKey"
    <xs:selector xpath =
        "DRINKER/FREQ" />
    <xs:field xpath = "@bar" />
</xs:keyref>
</xs:element>
```

Limbaje de interogare pentru XML

XPath

XQuery

XSLT

Modelul de Date

XPath/XQuery

- Corespondentul “relației” din modelul relational este: *secvența de articole*.
- Un *articol* este unul din următoarele:
 1. O valoare primitivă, de exemplu: integer sau string.
 2. Un *nod*.

Principalele Tipuri de Noduri

1. *Noduri Document* ce reprezintă documente în întregime.
2. *Elemente* ce sunt bucăți de document ce constau din câteva tag-uri de început și de sfârșit, și tot ceea ce există între ele.
3. *Atribute* ce sunt nume ce primesc valori în interiorul tag-urilor de început.

Noduri Document

- Formulate prin `doc(URL)` sau `document(URL)`.
- **Exemplu:** `doc(/usr/class/cs145/bars.xml)`
- Interogările XPath (și XQuery) fac referire la un nod document fie explicit, fie implicit.
 - **Exemplu:** definițiile cheilor în Schema XML folosesc expresii Xpath ce fac referire la documentul descris de către schemă.

DTD pentru Exemplu

```
<!DOCTYPE BARS [  
    <!ELEMENT BARS (BAR*, BEER*)>  
    <!ELEMENT BAR (PRICE+)>  
        <!ATTLIST BAR name ID #REQUIRED>  
    <!ELEMENT PRICE (#PCDATA)>  
        <!ATTLIST PRICE theBeer IDREF #REQUIRED>  
    <!ELEMENT BEER EMPTY>  
        <!ATTLIST BEER name ID #REQUIRED>  
        <!ATTLIST BEER soldBy IDREFS #IMPLIED>  
]>
```

Exemplu Document

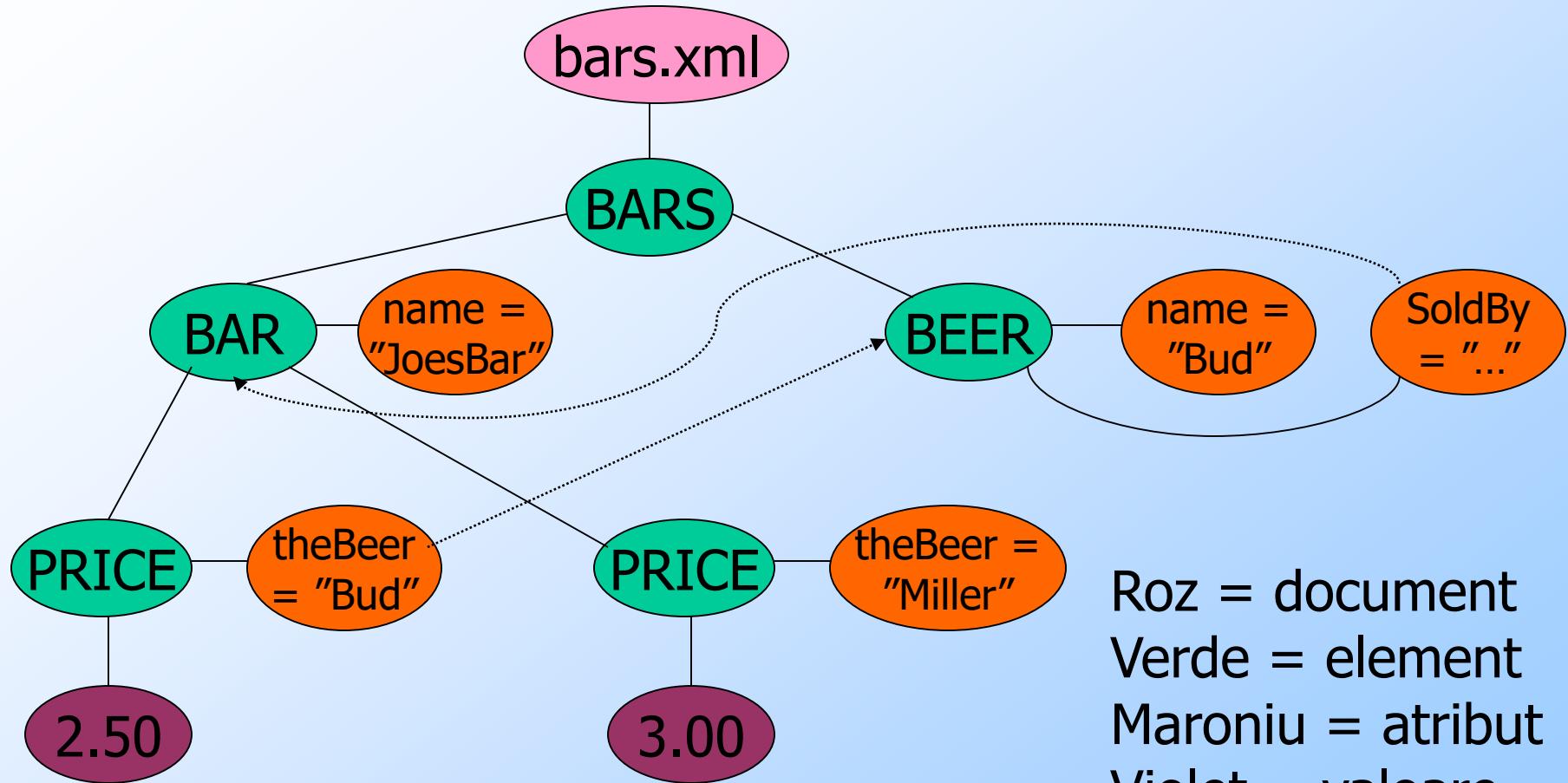
```
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ... "/> ...
</BARS>
```

Un nod element

Un nod atribut

Nodul document este totul de mai sus,
plus header-ul (<? xml version...).

Noduri, Date Semistructurate



Roz = document
Verde = element
Maroniu = atribut
Violet = valoare primitivă

Căi În Documente XML

- XPath este un limbaj cu ajutorul căruia se descriu *căi* în documente XML.
- Rezultatul căii descrise este o secvență de articole.

Expresii cale

- Expresii cale simple sunt secvențe de slash-uri (/) și tag-uri, ce încep cu /.
 - **Exemplu:** /BARS/BAR/PRICE
- Construirea rezultatului se face pornind de la nodul document, prin procesarea fiecărui tag de la stânga.

Evaluarea unei Expresii Cale

- Se presupune că primul tag este rădăcina.
 - Procesarea nodului document prin acest tag conduce la o secvență ce constă doar din elementul rădăcină.
- Să presupunem că avem o secvență de articole, și că tag-ul următor este X .
 - Pentru fiecare articol ce este un nod element, se înlocuiește elementul cu subelementele ce au tag-ul X .

Exemplu: /BARS

```
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ... "> ...
</BARS>
```

Un articol,
elementul BARS

Exemplu: /BARS/BAR

```
<BARS>
```

```
  <BAR name = "JoesBar">
    <PRICE theBeer ="Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
```

```
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ..." /> ...
</BARS>
```

Acest element BAR urmat de toate celelalte elemente BAR

Exemplu: /BARS/BAR/PRICE

```
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer ="Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ..."/> ...
</BARS>
```

The diagram shows two `<PRICE>` elements highlighted with a green rectangular background. One arrow points from the first highlighted element to a text block at the bottom right. Another arrow points from the second highlighted element to the same text block.

Elementele PRICE afișate,
următe de elementele PRICE
ale celorlalte baruri.

Atribute În Căi

- În loc să se menționeze subelemente cu un anumit tag, se poate menționa un atribut al elementelor.
- Un atribut este indicat cu prefixul @ înaintea numelui.

Exemplu:

/BARS/BAR/PRICE/@theBeer

```
<BARS>
```

```
  <BAR name = "JoesBar">
```

```
    <PRICE theBeer = "Bud">2.50</PRICE>
```

```
    <PRICE theBeer = "Miller">3.00</PRICE>
```

```
  </BAR> ...
```

```
  <BEER name = "Bud" soldBy = "JoesBar"
```

```
    SuesBar ..."/> ...
```

```
</BARS>
```

Aceste atribute, "Bud" "Miller", participă la rezultat, urmate de alte valori theBeer.

Recapitulare: Secvențe de Articole

- Până acum, toate secvențele de articole au fost secvențe de elemente.
- Atunci când o expresie cale se termină cu un atribut, rezultatul este de obicei o secvență de valori de tip primitivă, cum au fost sirurile de caractere din exemplul precedent.

Căi ce încep oriunde

- Atunci când calea pornește de la nodul document și începe cu $//X$, atunci primul pas poate începe cu rădăcina sau cu orice subelement al rădăcinii, atât timp cât tag-ul este X .

Exemplu: //PRICE

```
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer ="Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
  <BEER name = "Bud" soldBy = "JoesBar
    SuesBar ..."/> ...
</BARS>
```

The diagram shows two `<PRICE>` elements highlighted with a green rectangular background. Two arrows point from a text annotation at the bottom right towards these highlighted elements.

Elementele PRICE indicate și orice alte elemente PRICE în întregul document

Wild-Card *

- Asterix (*) în locul unui tag reprezintă orice tag.
- Exemplu: /*/*PRICE reprezintă toate obiectele price la al treilea nivel de imbricare.

Exemplu: /BARS/*

Acest element BAR, toate celelalte elemente BAR, elementul BEER, toate celelalte elemente BEER

```
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
<BEER name = "Bud" soldBy = "JoesBar
  SuesBar ... "/> ...
</BARS>
```

Condiții de Selectie

- Unui tag îi poate urma o condiție între [...].
- Dacă apare, atunci numai acele căi sunt incluse în rezultat, ce au acel tag și satisfac condiția.

Exemplu: Condiție de Selectie

□ /BARS/BAR/PRICE[ < 2.75]

Elementul
current.

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

Condiția PRICE < 2.75 (\$) determină acest preț participă la rezultat, dar prețul berii Miller nu participă la rezultat.

Exemplu: Atribut în Selectie

□ /BARS/BAR/PRICE[@theBeer = "Miller"]

<BARS>

 <BAR name = "JoesBar">

 <PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

 </BAR> ...



De această dată, acest element PRICE este selectat, împreună cu orice alte prețuri pentru Miller.

Axe

- În general, expresiile cale permit să se pornească de la rădăcină și să se execute pași în încercarea de a găsi o secvență de noduri la fiecare pas.
- La fiecare pas, se poate urma una din mai multe *axe*.
- Axa implicită este **child:: ---** merge la toți fiilor setului curent de noduri.

Exemplu: Axe

- /BARS/BEER este de fapt o scurtătură pentru /BARS/child::BEER .
- @ este de fapt o scurtătură pentru axa attribute::.
- Astfel, /BARS/BEER[@name = "Bud"] este scurtătura pentru /BARS/BEER[attribute::name = "Bud"]

Mai multe Axe

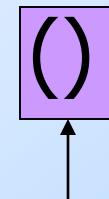
- Alte axe folositoare sunt:
 1. **parent::** = părintele(-ții) nodului(-lor) curent(-e).
 2. **descendant-or-self::** = nodul(-rile) curent(-e) și toți descendenții.
 - Notă: // este de fapt scurtătura pentru această axă.
 3. **ancestor::**, **ancestor-or-self**, etc.
 4. **self** (punctul).

XQuery

- XQuery extinde XPath spre un limbaj de interogare similar cu SQL.
- Folosește modelul de date secvență-de-articole.
- XQuery este un limbaj expresie.
 - Asemănător algebrei relaționale --- orice expresie XQuery poate reprezenta argumentul altrei expresii XQuery.

Secvențe de Articole

- XQuery va formula uneori secvențe de secvențe.
- Secvențele sunt pe nivel.
- Exemplu: $(1\ 2\ (3\ 4)) = (1\ 2\ 3\ 4)$.



↑
secvența
vidă

Expresii FLWR

1. Una sau mai multe clauze **for** și/sau **let**.
2. În continuare o cluză optională **where**.
3. O cluză **return**.

Semanticile Expresiilor FLWR

- Fiecare **for** crează o buclă.
 - **let** produce o definiție locală.
- La fiecare iterație a buclelor imbricate, dacă există, se evaluează clauza **where**.
- Dacă **where** returnează TRUE, se invocă clauza **return**, și se adaugă valoarea sa la ieșire.

Clauze FOR

for <variabilă> in <expresie>, . . .

- Variabilele încep cu \$.
- O variabilă **for** primește fiecare articol din secvența denotată prin expresie.
- Tot ceea ce urmează lui **for** se execută o dată pentru fiecare valoare a variabilei.

Documentul exemplu BARS

```
for $beer in  
    document("bars.xml")/BARS/BEER/@name  
return
```

```
<BEERNAME> {$beer} </BEERNAME>
```

- \$beer ia valori din plaja atributelor “name” pentru toate mărcile de bere din documentul exemplu.
- Rezultatul este o secvență de elemente BEERNAME : <BEERNAME>Bud</BEERNAME>
<BEERNAME>Miller</BEERNAME> . . .

Exemplu: FOR

“Expandează șirul inclus prin înlocuirea variabilelor și a expresiilor cale cu valorile lor.”

Folosirea Marcatorilor

- Atunci când un nume variabil, de exemplu \$x, sau o expresie, ar putea să fie text, este nevoie să folosim marcatori de jur împrejur pentru a evita interpretarea literală a ei.
 - **Exemplu:** <A>\$x este un element A cu valoarea "\$x", asemănător cu elementul A: <A>foo ce are valoarea "foo".

Folosirea Marcatorilor

- return \$x este lipsit de ambiguitate.
- Pentru a returna sirul de caractere, se include intre ghilimele: return "\$x".

Clauze LET

let <variabilă> := <expresie>, . . .

- Valoarea variabilei devine *secvența* articolelor definite prin expresie.
- Notă **let** nu cauzează iterație, aşa cum face **for**.

Exemplu: LET

```
let $d := document("bars.xml")
let $beers := $d/BARS/BEER/@name
return
    <BEERNAMES> {$beers} </BEERNAMES>
```

- Returnează un element cu toate numele mărcilor de bere, în felul următor:

```
<BEERNAMES>Bud Miller ...</BEERNAMES>
```

Clauze Order-By

- FLWR este în realitate FLWOR: o clauză order-by poate precede return.
- Forma: order by <expresie>
 - Optional poate exista **ascending** sau **descending**.
- Expresia este evaluată pentru fiecare atribuire a variabilelor.
- Determină poziționarea în secvența de ieșire.

Exemplu: Order-By

- Lista prețurilor pentru berea Bud, începând cu cel mai mic preț.

```
let $d := document("bars.xml")
```

```
for $p in  
    $d/BARS/BAR/PRICE[@theBeer="Bud"]
```

```
order by $p
```

Ordonează legăturile după valorile din interiorul elementelor.

Generează legături pentru \$p la elemente PRICE.

```
return $p
```

Fiecare legătură este evaluată pentru ieșire.

Rezultatul este o secvență de elemente PRICE.

Comparație: SQL ORDER BY

- SQL funcționează în același fel; ceea ce se ordonează este determinat de clauzele FROM și WHERE, nu ieșirea (clauza SELECT).

- Exemplu: Fie $R(a,b)$,

```
SELECT b FROM R  
WHERE b > 10
```

```
ORDER BY a;
```

Valorile "b" sunt extrase din tuplele relației R și sunt afișate în ordinea specificată.

Tuplele relației R cu $b > 10$ sunt ordonate după valorile "a".

Predicate

- În mod normal, condițiile implică existența cuantificării.
- Exemplu: /BARS/BAR[@name] semnifică “toate barurile ce au un nume”.
- Exemplu: /BARS/BEER[@soldAt = “JoesBar”] dă setul mărcilor de bere ce sunt vândute la “Joe’s Bar”.

Exemplu: Comparații

- Să încercăm să obținem elementele PRICE (din toate barurile) pentru toate mărcile de bere ce sunt vândute de "Joe's Bar".
- Ieșirea va fi formată din elemente BBP cu numele barului și marca de bere ca atribute și elementul preț ca subelement.

Strategia

1. Se crează o buclă *for* triplă, cu variabile ce parcurg toate elementele BEER, toate elementele BAR, și toate elementele PRICE în interiorul elementelor BAR.
2. Se verifică marca de bere să fie vândută la "Joe's Bar" și numele mărcii de bere să corespundă cu **theBeer** din elementul PRICE.
3. Se construiește elementul de ieșire.

Interogarea

```
let $bars = doc("bars.xml") / BARS
for $beer in $bars/BEER
for $bar in $bars/BAR
for $price in $bar/PRICE
where $beer/@soldAt = "JoesBar" and
      $price/@theBeer = $beer/@name
return <BBP bar = "{$bar/@name}" beer
      = "{$beer/@name}">{$price}</BBP>
```

Adevărat dacă
"JoesBar" apare
oriunde în secvență



Comparații Stricte

- Pentru a pretinde că lucrurile comparate sunt secvențe doar a unui singur element, se folosesc operatori comparație Fortran:
 - eq, ne, lt, le, gt, ge.
- **Exemplu:** \$beer/@soldAt eq "JoesBar" este adevărat doar dacă "Joe's" este singurul bar ce vinde berea respectivă.

Comparația Elementelor și a Valorilor

- Atunci când un element este comparat cu o valoare primitivă, elementul este tratat ca valoarea sa, dacă acea valoare este atomică.

□ Exemplu:

```
/BARS/BAR [@name="JoesBar"] /  
PRICE [@theBeer="Bud"] eq "2.50"  
este adevărat dacă Joe vinde Bud cu  
2.50 ($).
```

Comparația a două Elemente

- Nu este suficient ca două elemente să fie asemănătoare.
- **Exemplu:**

```
/BARS/BAR [@name="JoesBar"] /
```

```
PRICE [@theBeer="Bud"] eq
```

```
/BARS/BAR [@name="SuesBar"] /
```

```
PRICE [@theBeer="Bud"]
```

este fals, chiar dacă Joe și Sue au același preț pentru Bud.

Comparația a două Elemente

- Pentru ca elementele să fie egale, ele trebuie să fie identice, fizic, în documentul respectiv.
- **Subtilitate:** elementele sunt în realitate pointeri la secțiuni de documente particulare, nu sirurile de text ce apar în secțiune.

Obținerea Datelor din Elemente

- Presupunem că se dorește să se compare valorile elementelor, în loc de locația lor în documente.
- Pentru a extrage doar valoarea (de exemplu, prețul *în sine*) dintr-un element E , se folosește $\text{data}(E)$.

Exemplu: data()

- Presupunem că se dorește modificarea a ceea ce returnează “găsiți prețurile mărcilor de bere la baruri ce vând o marcă de bere vândută de asemenea de Joe” pentru a obține un element BBP “empty” (vid, fără valoare) cu prețul ca atribut.

```
return <BBP bar = "{$bar/@name}" beer  
= "{$beer/@name}" price =  
{data($price)} />
```

Eliminarea Duplicatelor

- Se folosește funcția distinct-values aplicată unei secvențe.
- **Subtilitate:** această funcție elimină tagurile din jurul elementelor și compară valorile sir de caractere.
 - Dar nu reface tag-urile în rezultat.

Exemplu: Toate Prețurile Distincte

```
return distinct-values(  
    let $bars = doc("bars.xml")  
    return $bars/BARS/BAR/PRICE  
)
```

De reamintit: XQuery este un limbaj expresie. O interogare poate apărea în orice loc unde poate apărea o valoare.

Valori Booleene Efective

□ *Valoarea booleană efectivă* (EBV) a unei expresii este:

1. Valoarea existentă dacă expresia este de tip boolean.
2. FALSE dacă expresia evaluează la 0, "" [șirul vid], sau () [secvența vidă].
3. TRUE în celelalte cazuri.

Exemple EBV

1. `@name="JoesBar"` are EBV TRUE sau FALSE, depinzând de faptul că atributul name este "JoesBar".
2. `/BARS/BAR[@name="GoldenRail"]` are EBV TRUE dacă un anumit bar este denumit Golden Rail, și FALSE dacă nu există un astfel de bar.

Operatori Booleeni

- E_1 and E_2 , E_1 or E_2 , not(E), se aplică oricărora expresii.
- Mai întâi se iau EBV-urile expresiilor.
- **Exemplu:** not(3 eq 5 or 0) are valoarea TRUE.
- De asemenea: true() și false() sunt funcții ce returnează valorile TRUE și FALSE.

Expresii Ramificație

- if (E_1) then E_2 else E_3 este evaluată:
 - Se calculează EBV-ul lui E_1 .
 - Dacă este adevărat, atunci rezultatul este E_2 ; altfel rezultatul este E_3 .

- Exemplu: subelementele PRICE ale \$bar, spun că este barul lui Joe.

```
if ($bar/@name eq "JoesBar")
then $bar/PRICE else ()
```

Secvența vidă.

Expresii de Cuantificare

some $\$x$ in E_1 satisfies E_2

1. Se evaluatează secvența E_1 .
2. Fie $\$x$ (orice variabilă) fiecare articol din secvență, și se evaluatează E_2 .
3. Se returnează TRUE dacă E_2 are EBV-ul TRUE pentru cel puțin un $\$x$.

În mod analog:

every $\$x$ in E_1 satisfies E_2

Exemplu: Some

- Barurile ce vând cel puțin o marcă de bere mai ieftin de 2 (\$).

```
for $bar in  
    doc("bars.xml") /BARS/BAR  
where some $p in $bar/PRICE  
    satisfies $p < 2.00  
  
return $bar/@name
```

Notă: where \$bar/PRICE < 2.00 ar fi funcționat de asemenea.

Exemplu: Every

- Barurile ce nu vând nici o marcă de bere mai scumpă de 5 (\$).

```
for $bar in
    doc("bars.xml")/BARS/BAR
where every $p in $bar/PRICE
    satisfies $p <= 5.00
return $bar/@name
```

Ordinea În Document

- Comparație după ordinea în document: << și >>.
- **Exemplu:** \$d/BARS/BEER[@name="Bud"]<< \$d/BARS/BEER[@name="Miller"] este adevărat dacă elementul "Bud" apare înainte de elementul "Miller" în documentul \$d.

Operatori pe Multimi

- `union`, `intersect`, `except` operează pe secvențe de noduri.
- Semnificațiile sunt asemănătoare cu SQL.
- Rezultatul elimină duplicatele.
- Rezultatul apare în ordinea din document.

XSLT

- XSLT (*extensible stylesheet language – transforms*) este un alt limbaj de procesare a documentelor XML.
- La origine era intenționat ca un limbaj pentru prezentare: să transforme XML într-o pagină HTML care să fie afișată.
- Dar poate de asemenea să transforme XML -> XML, astfel servind ca un limbaj de interogare.

Programe XSLT

- Ca și Schema XML, un program XSLT este el însuși un document XML.
- XSLT are un namespace special de tag-uri, de obicei indicat prin **xsl:**.

Template-uri

- Elementul **xsl:template** descrie un set de elemente (ale documentului procesat) și ce trebuie făcut cu ele.
- Formatul: <xsl:template **match = *cale***>
... </xsl:template>

Atributul **match** specifică o expresie XPath ce descrie cum să fie găsite nodurile cărora li se aplică template-ul.

Exemplu: Documentul BARS -> Tabel

- Ca un exemplu, se convertește documentul **bars.xml** într-un document HTML ce arată ca și relația **Sells(bar, beer, price)**.
- Primul template se va potrivi cu rădăcina documentului și va obține tabelul fără nici un rând.

Template pentru Rădăcină

```
<xsl:template match = “/”>  
  <TABLE><TR>  
    <TH>bar</th><TH>beer</th>  
    <TH>price</th></tr>  
  </table>  
</xsl:template>
```

Trebuie consolidat.

După cum arată, nu se pot introduce rânduri.

Template-ul se potrivește doar cu rădăcina.

Ieșirea template-ului este un tabel cu atributele capului de tabel, fără alte rânduri.

Schițarea Strategiei

1. În tabelul HTML există **xsl:apply-templates** pentru a extrage datele din document.
2. Pentru fiecare BAR, se folosește **xsl:variable** *b* pentru a memora numele barului.
3. **xsl:for-each** pentru subelementul PRICE generează un rând; se folosește *b* și **xsl:value-of** pentru a extrage denumirea și prețul mărcii de bere.

Folosirea Recursivă a Template-urilor

- Un document XSLT în mod obișnuit conține mai multe template-uri.
- Se începe cu găsirea primului care se aplică rădăcinii.
- Orice template poate avea `<xsl:apply-templates/>`, ce cauzează “match” al template-ului să se aplique recursiv începând cu nodul curent.

Apply-Templates

- Atributul **select** precizează o expresie XPath ce descrie subelementele cărora li se aplică template-urile.
- **Exemplu:** `<xsl:apply-templates select = "BARS/BAR" />` precizează să se urmeze toate căile etichetate BARS, BAR începând cu nodul curent și să le aplice toate template-urile.

Exemplu: Apply-Templates

```
<xsl:template match = "/">  
  <TABLE><TR>  
    <TH>bar</th><TH>beer</th>  
    <TH>price</th></tr>  
  <xsl:apply-templates select =  
    "BARS" />  
  </table>  
</xsl:template>
```

Extragerea Valorilor

- <xsl:value-of select = **expresie XPath**/> obține o valoare care să fie afișată.
- **Exemplu:** presupunem că se aplică un template elementului BAR și se dorește afișarea denumirii barului într-un tabel.
`<xsl:value-of select = "@name" />`

Variabile

□ Se declară **x** ca fiind o variabilă în felul următor:

```
<xsl:variable name = "x" />
```

□ **Exemplu:**

```
<xsl:variable name = "bar">  
  <xsl:value-of select = "@name" />  
</xsl:variable>
```

într-un template ce se aplică elementelor BAR
variabila **bar** este setată la denumirea barului.

Folosirea Variabilelor

- Se folosește \$ în fața numelui variabilei.
- Exemplu: <TD>\$bar</td>

Completarea Tabelului

1. Se va aplica template-ul fiecărui element BAR.
2. Template-ul va atribui unei variabile **b** valoarea barului, și va itera pentru fiecare fiu PRICE.
3. Pentru fiecare fiu PRICE, se va afișa un rând, folosind **b**, atributul **theBeer**, și PRICE.

Iterația

□ <xsl:for-each select = expresie Xpath>

...

</xsl:for-each>

la fiecare fiu al nodului curent la care se ajunge cu calea expresie Xpath se execută corpul *for-each*.

Variabila
pentru
 fiecare bar

Template-ul pentru BARS

```
<xsl:template match = "BAR">
```

```
    <xsl:variable name = "b">  
        <xsl:value-of select = "@name" />  
    </xsl:variable>
```

```
    <xsl:for-each select = "PRICE">
```

```
        <TR><TD>$b</td><TD>  
            <xsl:value-of select = "@theBeer" />  
        </td><TD>  
            <xsl:value-of select = "data(.)" />  
        </td></tr>
```

```
    </xsl:for-each>
```

```
</xsl:template>
```

Construiește un rând
bar-beer-price.

Iterează pentru toate
subelementele PRICE
ale barului.

Elementul
curent

Exemplu practic

Fișierul catalog.xml

```
<?xml version="1.0"?>
<Catalog>

<Book>
  <Title>Teach Yourself XML in 24 Hours</Title>
  <Author>
    <Name>Charles Ashbacher</Name>
    <Address>119 Northwood Drive</Address>
    <City>Hiawatha</City> 10: <State>Iowa</State>
    <PostalCode>52233</PostalCode>
    <Country>United States</Country>
    <Phone>(319) 378-4646</Phone>
    <Email>ashbacher@ashbacher.com</Email>
    <URL>http://www.ashbacher.com</URL>
  </Author>
  <Publisher>
    <Name>Sams Publishing</Name>
    <Address>201 West 103rd Street</Address>
    <City>Indianapolis</City>
    <State>Indiana</State>
    <PostalCode>46290</PostalCode>
    <Country>United States</Country>
    <Phone>(111) 123-4567</Phone>
    <Editor>Dummy Name</Editor>
    <EmailContact>Dummy. Name@samspublishing.com</EmailContact>
    <URL>http://www.samspublishing.com</URL>
  </Publisher>
  <ISBN>0-672-31950-0</ISBN>
  <PublisherBookID>1234567</PublisherBookID>
  <PublicationDate>2000-04-01</PublicationDate>
  <Price>24.95</Price>
</Book>
```

Fișierul catalog.xml (continuare)

```
<Book>
<Title>Teach Yourself Cool Stuff in 24 Hours</Title>
<Author>
  <Name>ReallySmart Person</Name>
  <Address>110 Main Street</Address>
  <City>Silicon City</City>
  <State>California</State>
  <PostalCode>10101</PostalCode>
  <Country>United States</Country>
  <Phone>(101) 101-0101</Phone>
  <Email>RSmart@MyCompany.com</Email>
  <URL>http://www.MyCompany.com</URL>
</Author>
<Publisher>
  <Name>Sams Publishing</Name>
  <Address>201 West 103rd Street</Address>
  <City>Indianapolis</City>
  <State>Indiana</State>
  <PostalCode>46290</PostalCode>
  <Country>United States</Country>
  <Phone>(111) 123-4567</Phone>
  <Editor>Dummy Name</Editor>
  <EmailContact>Dummy.Name@samspublishing.com</EmailContact>
  <URL>http://www.samspublishing.com</URL>
</Publisher>
<ISBN>0-672-1010-0</ISBN>
<PublisherBookID>1234568</PublisherBookID>
<PublicationDate>2000-06-01</PublicationDate>
<Price>24.95</Price>
</Book>
```

Fișierul catalog.xml (continuare)

```
<Book>
  <Title>Teach Yourself Uses of Silicon in 24 Hours</Title>
  <Author>
    <Name>Missy Fixit</Name>
    <Address>119 Hard Drive</Address>
    <City>Diskette City</City>
    <State>California</State>
    <PostalCode>10101</PostalCode>
    <Country>United States</Country>
    <Phone>(586) 212-3638</Phone>
    <Email>MFixit@Googol.com</Email>
    <URL>http://www.Googol.com</URL>
  </Author>
  <Publisher>
    <Name>Sams Publishing</Name>
    <Address>201 West 103rd Street</Address>
    <City>Indianapolis</City>
    <State>Indiana</State>
    <PostalCode>46290</PostalCode>
    <Country>United States</Country>
    <Phone>(111) 123-4567</Phone>
    <Editor>Dummy Name</Editor>
    <EmailContact>Dummy.Name@samspublishing.com</EmailContact>
    <URL>http://www.samspublishing.com</URL>
  </Publisher>
  <ISBN>0-578-31950-0</ISBN>
  <PublisherBookID>1234569</PublisherBookID>
  <PublicationDate>2000-01-01</PublicationDate>
  <Price>26.95</Price>
</Book>
</Catalog>
```

Fișierul catalog.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
<xsl:template match="/">
<html>
<body>
<table border="1">
<tr style="font-weight:bold;color:blue">
<td>Author Name</td>
<td>Author Address</td>
<td>Author City</td>
<td>Author State</td>
<td>Author e-mail</td>
<td>Publisher Name</td>
<td>Publisher Address</td>
<td>Publisher Phone</td>
</tr>
```

Fisierul catalog.xsl (continuare)

```
<xsl:for-each select="Catalog/Book">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
<xsl:template match="Author">
  <td><xsl:value-of select="Name"/></td>
  <td><xsl:value-of select="Address"/></td>
  <td><xsl:value-of select="City"/></td>
  <td><xsl:value-of select="State"/></td>
  <td><xsl:value-of select="Email"/></td>
</xsl:template>
<xsl:template match="Publisher">
  <td><xsl:value-of select="Name"/></td>
  <td><xsl:value-of select="Address"/></td>
  <td><xsl:value-of select="Phone"/></td>
</xsl:template>
</xsl:stylesheet>
```

Fișierul catalog.html

```
<html>
<head>
<title>This program demonstrates the use of an XSL template to read XML
      data</title>
<script language="JavaScript">
function StartUp()
{
    var DataSource=new ActiveXObject("microsoft.xmlDOM");
    DataSource.load("catalog.xml");
    var XslStyle=new ActiveXObject("microsoft.xmlDOM");
    XslStyle.load("catalog.xsl");
    document.all.item("xslcontainer").innerHTML=
        DataSource.transformNode(XslStyle.documentElement);
}
</script>
</head>
<body onLoad="StartUp()">
<span ID="xslcontainer"></span>
</body>
</html>
```

Rezultatul

| Author Name | Author Address | Author City | Author State | Author e-mail | Publisher Name | Publisher Address | Publisher Phone |
|--------------------|---------------------|---------------|--------------|-------------------------|-----------------|-----------------------|-----------------|
| Charles Ashbacher | 119 Northwood Drive | Huawatha | Iowa | ashbacher@ashbacher.com | Sams Publishing | 201 West 103rd Street | (111) 123-4567 |
| ReallySmart Person | 110 Main Street | Silicon City | California | RSmart@MyCompany.com | Sams Publishing | 201 West 103rd Street | (111) 123-4567 |
| Missy Fict | 119 Hard Drive | Diskette City | California | MFixt@Googol.com | Sams Publishing | 201 West 103rd Street | (111) 123-4567 |

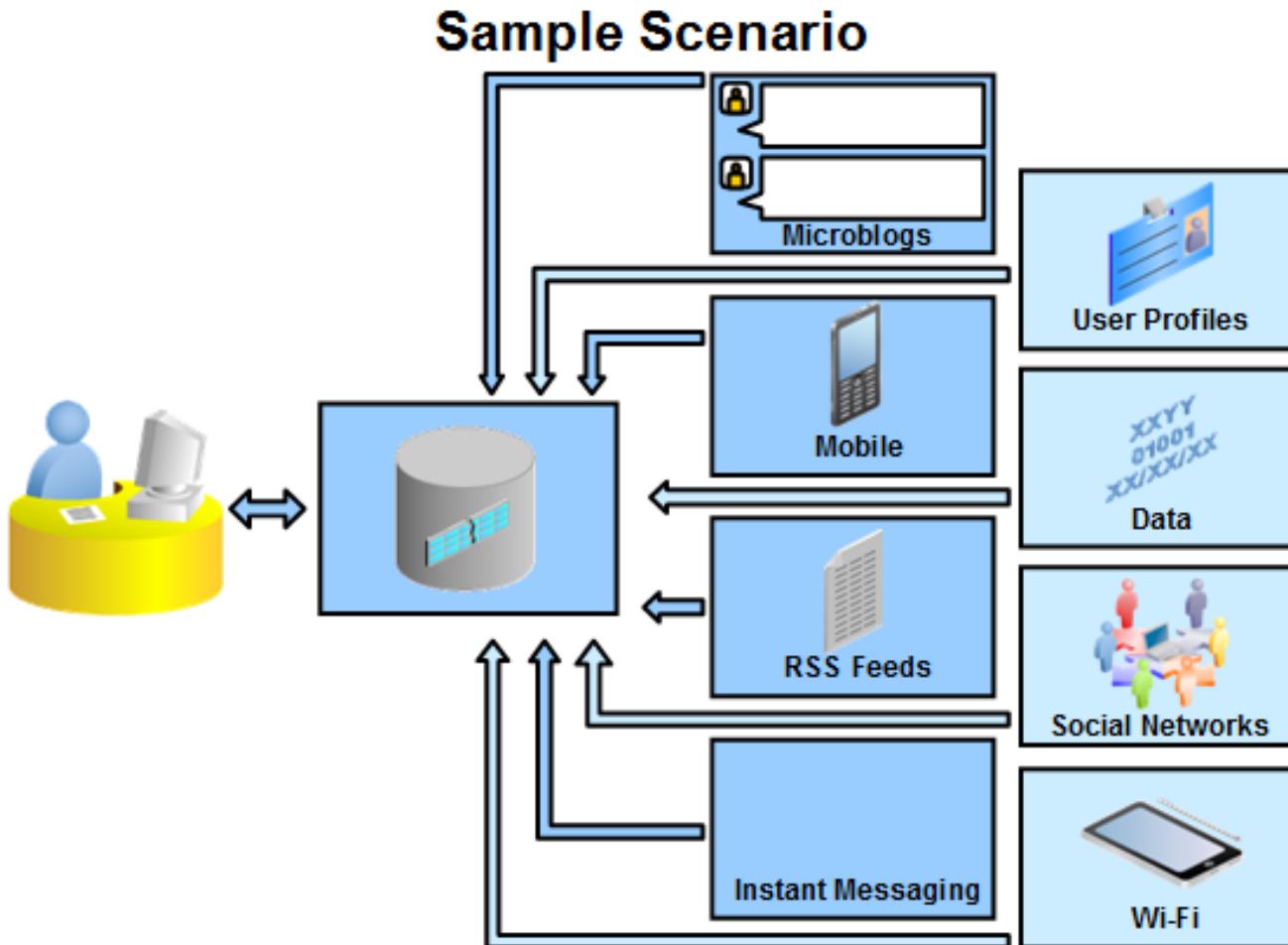
Fișierul catalog.html apelat în browser conduce la afișarea tabelară a conținutului fișierului catalog.xml folosind template-ul catalog.xsl.

În acest exemplu practic se folosește ActiveXObject XMLDOM pentru prelucrarea documentelor XML cu javascript.

BD nestructurate

Concepțele “Big Data” și NoSQL
Acces la Oracle NoSQL cu Java API

Scenariu de aplicație

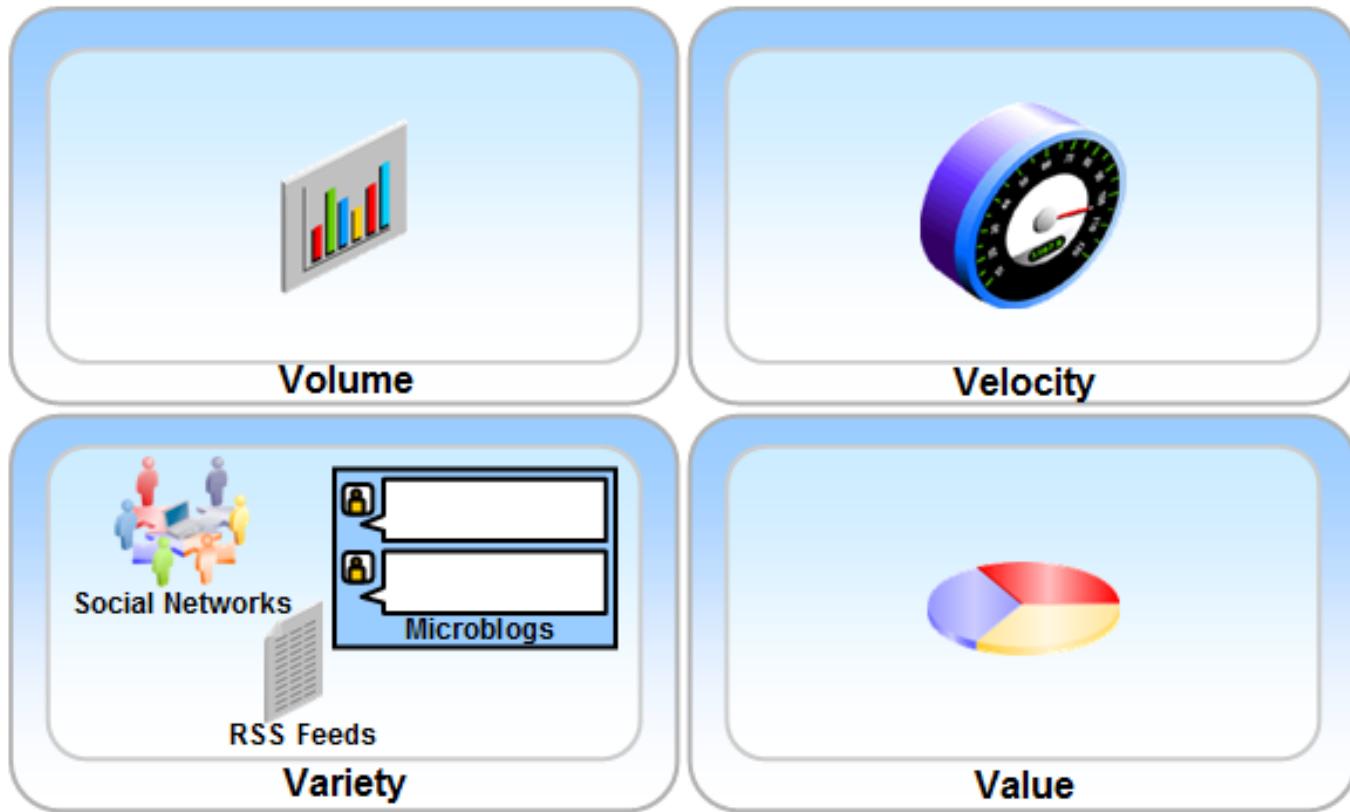


Scenariu de aplicație

- Consider the following scenario: Mark developed a database application for his enterprise.
As the application became widely used, the amount of data generated by the application increased rapidly. Mark and his team decided to scale up their database storage.
- New requirements to capture application users' profile details were passed on to Mark. He worked hard to revise the existing database schema to accommodate the new requirements.
- Now that the application has become more popular, Mark's management has asked him to capture additional details about users, such as their mobile, Facebook, and Twitter footprints on the Internet. Mark agrees that these new requirements will make the application more useful for the business. However, the volume of data to be captured and the velocity at which the data needs to be stored are estimated to be very high. Also, the data to be stored is not of high value unless it is aggregated and evaluated as a whole.
- Trying to fulfill all these requirements with the relational database will be very expensive. Moreover, with so many new requirements, Mark is now overwhelmed with the amount of work that is needed to change the existing database schemas. In addition, management wants the updated application to be rolled out as soon as possible.
- Can you relate to Mark's situation? What should he do in this situation?

Conceptual “Big Data”

What Is Big Data?



ORACLE®

Conceptual “Big Data”

- What data can be defined as “big data”? As the term itself suggests, data sets that are very large in size are generally called *big data*. The slide shows the main characteristics of big data (referred to as “the four Vs”):
 - Volume
 - Velocity
 - Variety
 - Value
- You might be familiar with the first two characteristics. They mean that the data grows tremendously in volume with rapid velocity.
- The last two characteristics are unique to big data.
- *Variety* means that the big data datasets can be from different sources. This makes it difficult for these datasets to be stored in traditional relational databases. Because big data does not have a defined structure, it needs to be handled differently.
- *Value* means that, out of all the big data that is generated from these various sources, only a little data is of value to drive business decisions. That is, a piece of information in big data is not valuable on its own, but it becomes valuable in the aggregate.

Conceptual “Big Data”

- ❑ More people interacting with data

- ❑ Smart phones
 - ❑ Internet



- ❑ Greater volumes of data being generated

- ❑ Sensors
 - ❑ GPRS



Conceptul “Big Data”

- As more and more people have started using the Internet and new technologies such as smart phones, greater volumes of data are being generated all over the world.
- These data are generated in various formats. Because traditional databases could not handle these volumes of data and process them instantly, there was a need for a different approach to storing data

Conceptul “NoSQL”

- Schema-less storage
- Stores nonrelational data
- Examples of NoSQL databases:
 - Oracle NoSQL
 - Cassandra
 - Voldemort
 - MongoDB

Conceptul “NoSQL”

- A NoSQL database is a nonrelational database that does not store information in the traditional relational format.
- There is no defined schema for the data.
 - The data are stored as key-value pairs.
- The term *NoSQL* is an abbreviation of “Not Only SQL.”

Modele de date NoSQL

- A NoSQL database uses one of the following data models:
 - Key-value
 - Columnar
 - Document
 - Graph

Modele de date NoSQL

- There are four data models for NoSQL databases:
 - **Key-value:** This is the simplest data model for unstructured data. It is highly efficient and highly flexible. The drawback of this model is that the data is not self-describing.
 - **Columnar:** This data model is good for sparse data sets, grouped subcolumns, and aggregated columns.
 - **Document:** This data model is good for XML repositories and self-describing objects. However, storage in this model can be inefficient.
 - **Graph:** This is a relatively new model that is good for relationship traversal. It is not efficient for general searches.
- In this course, you learn about Oracle NoSQL Database, which belongs to the key-value data model category.

Comparatie relational/NoSQL

RDBMS

- High-value, high-density, complex data
- Complex data relationships
- Joins
- Schema-centric
- Designed to scale up, not out
- Well-defined standards
- Database-centric

NoSQL

- Low-value, low-density, simple data
- Very simple relationships
- Avoids joins
- Schema-free, unstructured or semi-structured data
- Distributed storage and processing
- Standards not yet evolved
- Application-centric and developer-centric

Use Case: Scenariu 1

Healthcare System

- A city needs a centralized healthcare system with the following requirements:
 - It should store the health records of all the people in the city across all the different hospitals.
 - Doctors should be able to use this system to understand the health history of patients.
 - The system should be able to storage different kinds of data.
 - Data that needs to be stored might change over time.

What technology would you recommend to build this application?

Use Case: Scenario 1

Healthcare System

- This scenario is an example of big data. A NoSQL database should be used to acquire the data.
- What are some of the considerations that helped to select the back-end storage technology for the application?
 - **Data volume:** How big is the data that needs to be stored? In this scenario, supposed the city's population is twenty million. The health information is estimated to reach 5.8 petabytes.
 - **Real-time information:** The doctors should be able to retrieve information about a patient instantly. In many hospitals, doctors get less than five minutes to deal with each patient. So the response time should be immediate.
 - **Data variety:** The application should be able to store any kind of information about the patient (x-ray reports, scans, laboratory results, doctor inputs, medical bills, and so on).
 - **Data change:** Different hospitals will have different policies about the data that needs to be stored. These policies might change from time to time.

Use Case: Scenariu 2

Human Resources System

- A multinational company needs a centralized human resources system with the following requirements:
 - It should store the information of all employees in the organization (both current and former employees).
 - For each employee, it should store information such as date of hire, family details, job history, health history, benefits received from the company, and date of resignation or retirement.
 - It should be able to store scans of important documents, employee fingerprints, voice samples, and so on.
 - Company benefits and policies applying to an employee might change from time to time.

Use Case: Scenariu 2

Human Resources System

- This scenario is an example of high-value and confidential information. A RDBMS database should be used to acquire the data.
- What are some of the considerations that helped to select the back-end storage technology for the application?
 - **Data value:** The information that this application is required to handle is of very high value. Information about an employee is always considered to be confidential and should be securely stored.
 - **Data structure:** Although this application needs to handle different kinds of data, you can still predict a structure for the information to be stored.

Use Case: Scenariu 3

Retail Marketing System

- In a new marketing strategy, you want to offer discount coupons to your customers when they are near your business.

You will need to store:

- Customer profiles
- Customer purchase history
- GPRS signals from mobile devices
- Promotion details

Use Case: Scenariu 3

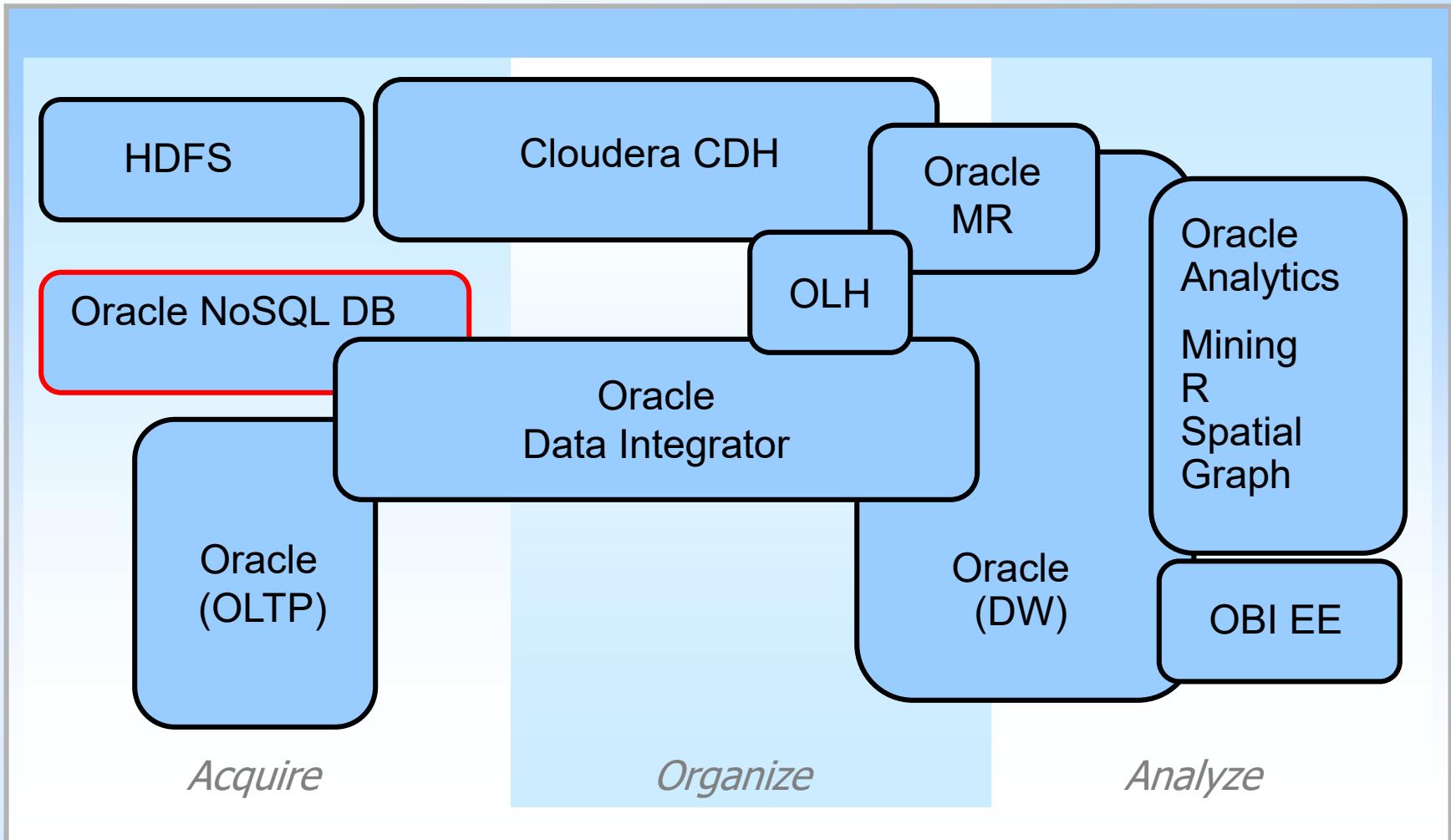
Retail Marketing System

- In the scenario described in the previous slide, you want to send discount coupons to your customers as they approach your business. You should process all information and send the coupons as soon as the customer is near the business. If you send the coupon when the customer has finished shopping and has already reached the parking lot, it is too late. Only when the customer is approaching the business does the information become valuable.
- You will need to store many details (for example, the GPRS feed). You also need to store the promotions and offers that your business is featuring—which might change on a daily basis.
- With all these requirements in mind, you should choose a NoSQL database as the best storage option for this application.

Criterii pentru a alege NoSQL

- You should make the following analyses when deciding on an applications database technology:
 - Analyze the data to be stored.
 - High volume, low value?
If answer is “yes,” then NoSQL is a better choice.
 - Analyze the application schema.
 - Dynamic?
If answer is “yes,” then NoSQL is a better choice.

Oracle Big Data Solution



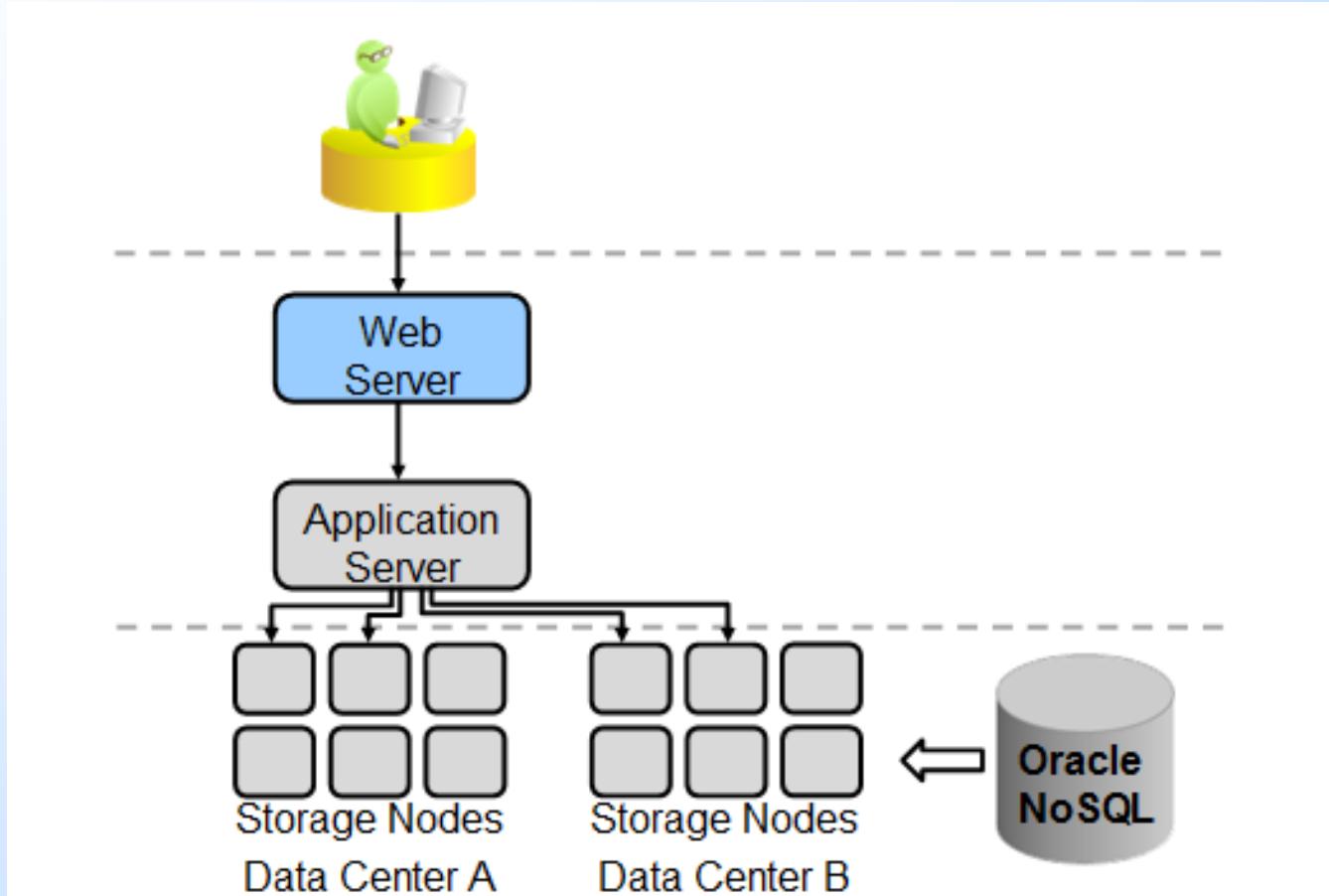
Oracle NoSQL Database

- Oracle NoSQL Database is:
 - A key-value database
 - Written in Java
 - Accessible using Java APIs
 - Built on Oracle Berkeley DB Java Edition
 - The Oracle solution to acquiring big data

Tipuri de date Oracle NoSQL



Arhitectura unei aplicații cu Oracle NoSQL

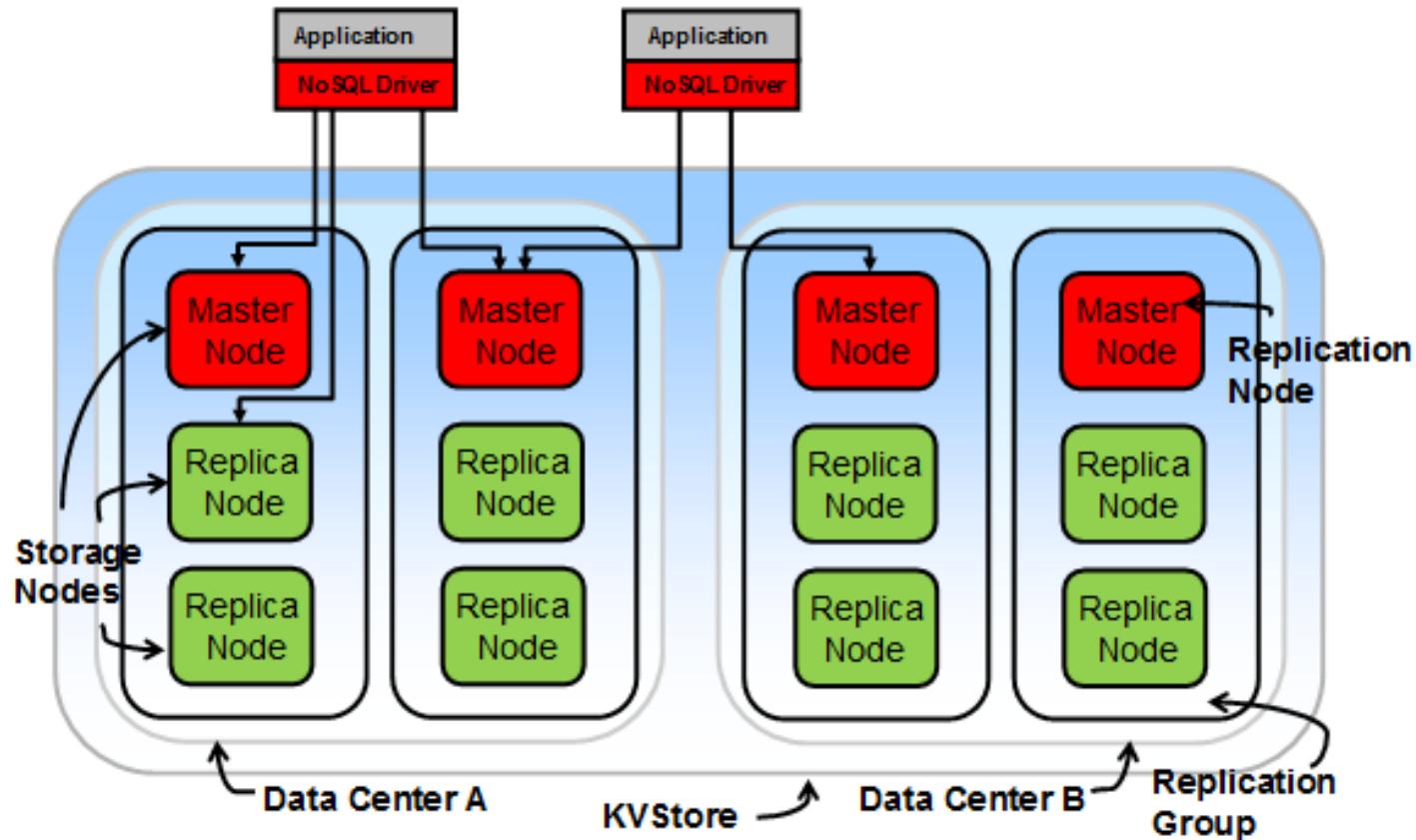


Arhitectura unei aplicații cu Oracle NoSQL

- Consider a typical web application scenario in which the application services requests across the traditional three-tier architecture: web server, application server, and database server.
- In this scenario, Oracle NoSQL Database is installed behind the application server. Oracle NoSQL Database either takes the place of the back-end database server or runs alongside the back-end database.
- Oracle NoSQL Database is installed in a set of storage nodes that may be located in different data centers.

Componente Oracle NoSQL

Oracle NoSQL Database Components



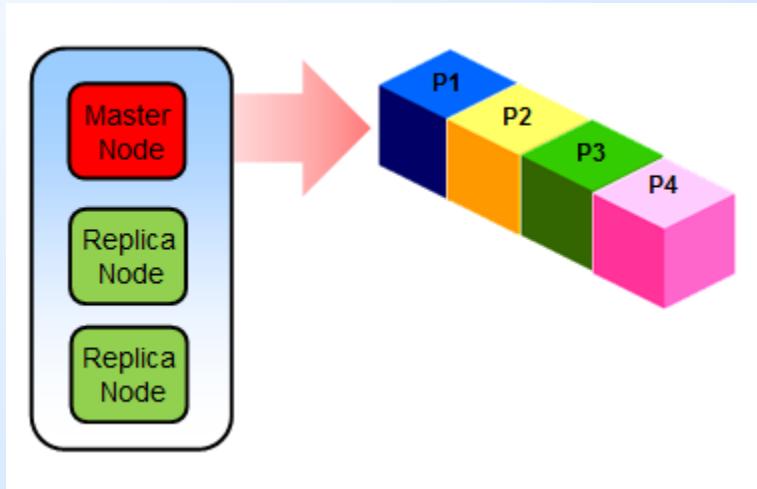
Componente Oracle NoSQL

- The main component of Oracle NoSQL Database is the Key-Value Store (KVStore). The KVStore is a collection of storage nodes that may be distributed across different data centers. A data center could be physically located at a different location than other data centers. The previous slide shows a KVStore with a number of storage nodes distributed across two data centers: A and B.
- A storage node is a physical or virtual machine with its own local storage. It is recommended that all the storage nodes within a KVStore be identical. A storage node hosts one or more replication nodes. For better performance, it is recommended that each storage node host only one replication node. In the slide example, each storage node is considered to contain one replication node.

Componente Oracle NoSQL

- A replication node is the place where the key-value pairs of data are stored. The replication nodes are organized into replication groups (also referred to as *shards*). Each replication group contains a master node and one or more replica nodes. The master node within a replication group handles all the database write operations and keeps the replica nodes updated. The replica nodes handle all the database read operations.
- To enable your application to communicate with the KVStore, you must link an Oracle NoSQL Database driver to your application. This driver is a Java library that you access from your application by using Java APIs.

Componente Oracle NoSQL

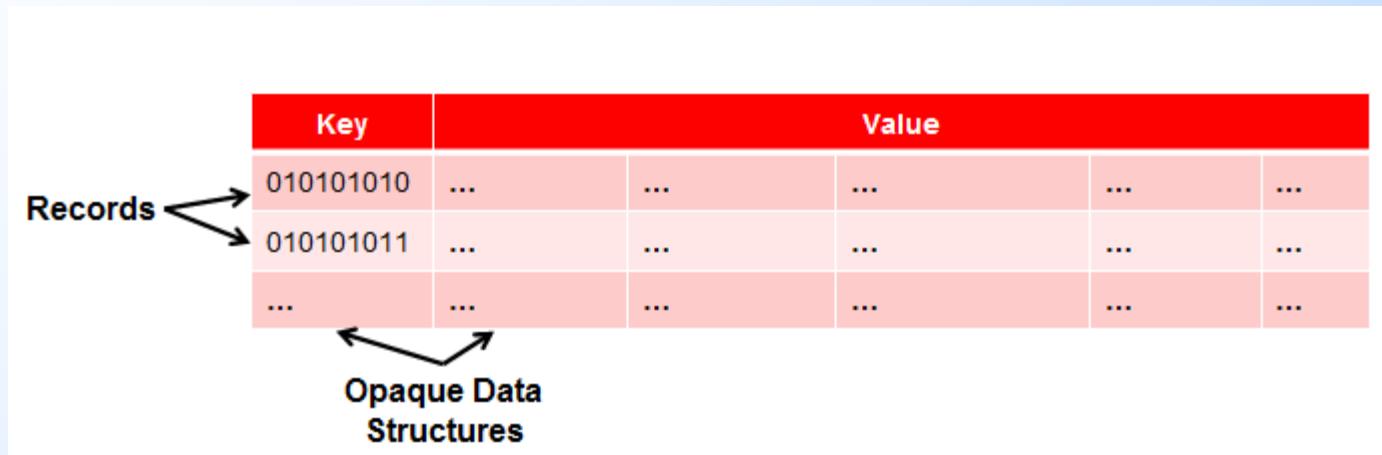


- A replication group is divided into partitions. A partition holds a key or a subset of keys. After a key is placed in a partition, it cannot be moved to a different partition. There are tools (shipped with the product) that enable you to plan how many partitions you need depending on your workload requirements.
- The task of evenly distributing the keys into available partitions is automatically handled by Oracle NoSQL Database. Also, Oracle NoSQL Database ensures that the data is read from and written to the correct partition

Acces la Oracle NoSQL

- You access the KVStore for two different needs.
 - For access to key-value data:
 - Use Java APIs.
 - For administrative actions:
 - Use the command-line interface.
 - Use the graphical web console.

Schema Structure for Oracle NoSQL Database



Recall that one of the key features of Oracle NoSQL Database is that it is schema-less. This means that, in Oracle NoSQL Database, the data is not stored in fixed table-like structures.

Schema Structure for Oracle NoSQL Database

- To understand the schema structure for Oracle NoSQL Database, you can relate it to a two-column relational table with key and value columns. However, the structure of the key and value columns can take any form. Each record in Oracle NoSQL Database consists of a key-and-value pair.
- The schema for Oracle NoSQL Database is not self-describing. The application using Oracle NoSQL Database is considered to know the structure of the key and value fields.

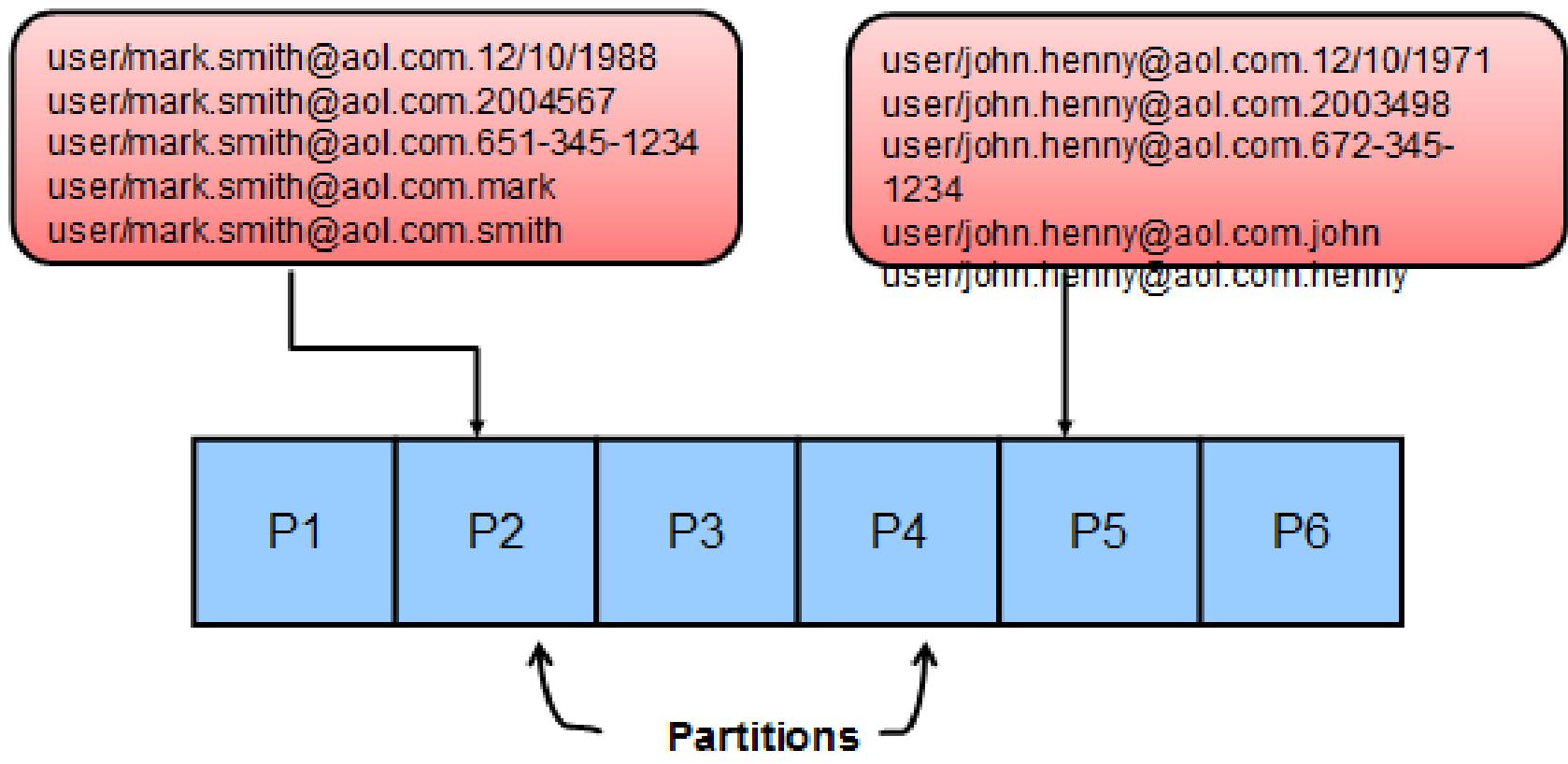
Conceptual “Key”

- The Oracle NoSQL Database key uniquely identifies each record.
 - It is application defined.
 - It is of the String data type.
 - It has a value attached.
 - It consists of major and minor components.

Conceptual “Key”

- You use a key in Oracle NoSQL Database to uniquely identify a record from your data. A key is a list of values of the `String` data type. The structure of a key is defined by you, and the application is assumed to know how the key is defined. Each key component is paired with a value component.
- A key can consist of major- and minor-key components. It can have more than one major or minor component. However, all keys must have at least one major component. If a key has minor components, the combination of both the minor- and major-key components uniquely identifies a record.
- How you design a key affects the performance of your application.

Exemplu Key



Exemplu Key

- An application's keys are evenly spread across the KVStore's partitions based on the keys' major components.
- That is, records that share the same combination of major-key components are guaranteed to be in the same partition.
- When a set of records that you want to work with are in the same partition, you can efficiently query them.
- This means that you can perform multiple operations on these records under a single atomic operation.

Exemplu Key

- The previous slide shows two sets of records that have the following record structure:
 - user/email.birthdate
 - user/email.id
 - user/email.phone
 - user/email.firstname
 - user/email.lastname

Exemplu Key

- Here, `user` is a string constant. Along with the email ID of a person, `user` and `email` form the major-key components of a record.
- These records have additional minor keys such as birth date, ID, phone number, first name, and last name.
- In this example and other examples in this lesson, a slash character (/) is used to separate the major-key components, and . is used to specify minor-key components.
- This convention is used only for the purposes of illustration.

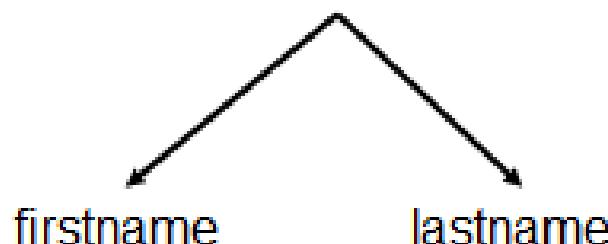
Definirea Key

When defining a key for your application, consider the following:

- Do you want to define both major and minor-key components?
- Do you want to define one or more major-key components?
- Do you want to define one or more minor-key components?

Major Key Components

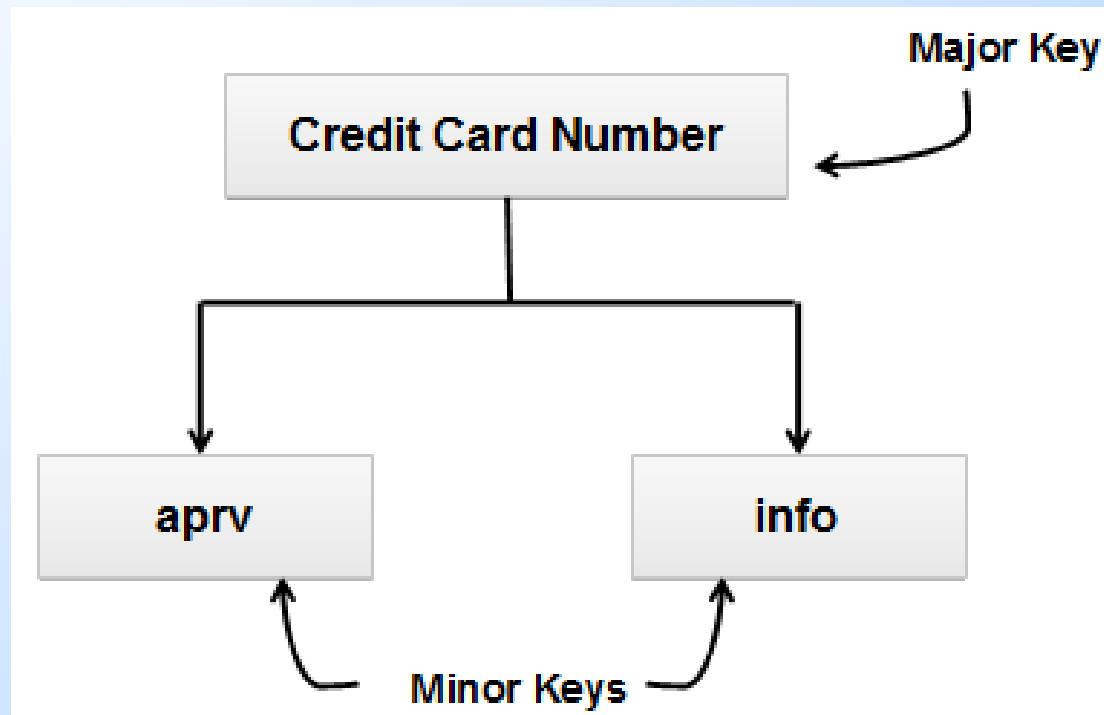
Example:



Definirea Key

- As mentioned previously in this lesson, a key is application defined. Choosing a key for your application data requires you to consider the issues listed in the previous slide.
- In some situations, defining a key with only one major-key component is sufficient. For example, for an application tracking details of all employees within an organization, the employee ID can be defined as the key component. In some situations, you might want to define a person's name as the key component. In this case, the first name and last name of a person can be defined as the two major-key components.
- It is recommended that you do not create all the records under a single major-key component. As the number of records grows in the store, performance problems can result.
- You can define minor-key components to further organize your data. Later in this lesson, you learn when to use minor-key components.

Exemplu: Credit Card Approval Application:Major Key



□ Structure:

- /<creditcardnumber>/-/aprv
- /<creditcardnumber>/-/info

Exemplu: Credit Card Approval Application:Major Key

- In the Credit Card Approval application sample, there is only one major-key component: the credit card number.
- There is also a minor-key component, which takes two values: aprv and info.
- The structure of the records formed using these key components is shown in the slide.

Minor Key

Minor keys help to organize data that you want to store.

Example:

firstname.lastname.birthdate

firstname.lastname.phonenumber

firstname.lastname.email

firstname.lastname.image

Minor Key Components

Minor Key

- You can define minor-key components to further organize your data.
- For example, when you want to store a person's information, you can organize the data under minor-key components such as birth date, phone number, image, email ID, and so on.
- When designing a key, remember that there is an overhead involved with each key that the store maintains.
- Defining too many minor keys might not be the best solution for the store's performance.

Proiectarea componentelor Key

- Consider the following before finalizing your key structure:
 - What is the data (value) that you want to store?
 - What data always must be accessed together?
 - What data can be independently accessed?
 - How big is the data (value) component?
 - How frequently is the data accessed?
 - How many partitions are there in the store?

Proiectarea componentelor Key

- How you design your key components will greatly affect your application's performance. Consider the issues listed in the previous slide while designing the key components.
- From these considerations, you will see that you must have a clear picture of the type of data you are dealing with before finalizing the key structure. You need to know what data you need to store, how large or small that data is, how frequently it will be accessed, and so on.
- It is also recommended that you have as many different major-key components as you have partitions in the store.

Conceptul “Value”

- In a key-value pair, the value component:
 - Is the data that you want to store and manage
 - Is a byte array
 - Is application-defined

Conceptual “Value”

- The value part of a key-value record is the data that you want to store and manage using Oracle NoSQL Database. It is stored in Oracle NoSQL Database as a byte array.
- The value component can be as large or as small as you want it to be. However, larger records take a longer time to read and write than do shorter records.

Exemplu: Credit Card Approval Application:Value

Some values stored in the Credit Card Approval application are:

- 0000000000000001012013#1000
 - 0000000000000002022011#11000
-

aprv

- 0000000000000004042014#18000.00#1000.00#Y#Ms. Rachel Bard#8 South St, Boston, MA#617-635-2222
- 0000000000000005052014#8000.00#1200.00#N#Mr. Harry Davis#10 Bond St, Seattle, WA#202-264-0000

info

Proiectarea componentelor Value

- Consider the following before finalizing your value structure:
 - How large will you allow the individual records to grow?
 - How frequently will the records be accessed?
 - Strike a balance between too many small records or a small number of very large records.
 - The maximum size of value component is four gigabytes (4GB).

Aplicație cu NoSQL

□ Consistency guarantees

□ Durability guarantees

□ Versioning

Consistency

- Consistency guarantee:
 - The policy that describes whether a record in the replica nodes can be different from the same record in the master node
- Two ranges:
 - High consistency guarantee
 - Low consistency guarantee

Consistency

- Recall that a KVStore consists of replication groups. Each replication group has a master node and one or more replica nodes. At any specific point in time, there is a possibility that a record in the replica nodes is different from the same record in the master node. This is known as a *consistency guarantee*.
- When there is a high probability that a record in the replica node is identical to the same record in the master node, the application is said to have *high consistency guarantees*.

Consistency

- Similarly, if there is a low probability that a record in the replica node is identical to the same record in the master node, the application is said to have *low consistency guarantees*.
- You can control how high you want an application's consistency guarantee to be. However, a high consistency guarantee results in slow write performance. Similarly, a low consistency guarantee results in fast write performance.

Implementarea Consistency

- You implement consistency in the following ways:
 - Set a default consistency for the entire store.
 - Set consistency for a particular operation.
 - Override the default consistency for a particular operation

Implementarea Consistency

- Oracle NoSQL Database enables you to choose how to implement consistency guarantees for your application. You can set a default consistency for the entire store. This policy will be applied to all the operations you perform on the store. In this case, you can also override the default policy by setting a different policy for an operation.
- There are different types of consistency policies that you can implement and use:
 - Predefined consistency guarantees
 - Time-based consistency guarantees
 - Version-based consistency guarantees

Exemplu Consistency

```
package developerday;

public class CreditCard
{

    public void getAccountData(DbStore ds)
    {
        final ValueVersion vv2 =
        myStore.getStore().get(creditCardKey,
            Consistency.ABSOLUTE, 0, TimeUnit.MILLISECONDS
        );
    }

}
```

Exemplu Consistency

- The previous slide illustrates an example from the Credit Card Approval application.
- In the CreditCard class, the consistency of a particular operation is specified.
- In this example, the Consistency.ABSOLUTE parameter is used to specify that the record should always be consistent with the master node.

Durability

- Durability guarantee:

- The policy that describes the persistence of data in a store in case of failure in the store

- Two ranges:

- High durability guarantee
 - Low durability guarantee

Durability

- Write operations in a KVStore are performed on master nodes. These write operations might be the creation of new records, updates of existing records, or the deletion of records.
- The master node is responsible for ensuring that the write operation has made it to stable storage.
- It is also responsible for transmitting the write operation to the replica nodes.
- You can set the rules or policies for when a write operation on the master node is said to be “complete.” This is called a *durability guarantee*.

Durability

- A *high durability guarantee* means that in a store failure such as a power outage or disk crash, the write operation is still retained.
- Likewise, a *low durability guarantee* means that in a failure, the write operation will be lost.
- With higher durability guarantees, the write performance of the store becomes slower.

Implementarea Durability

- You implement durability by using one of the following:
 - Synchronization-based durability policies
 - Acknowledgement-based durability policies

Implementarea Durability

- A write operation consists of the following operations:
 1. Data in the in-memory cache is modified.
 2. Data is written to the file system's data buffers.
 3. Buffer data is synchronized to stable storage.

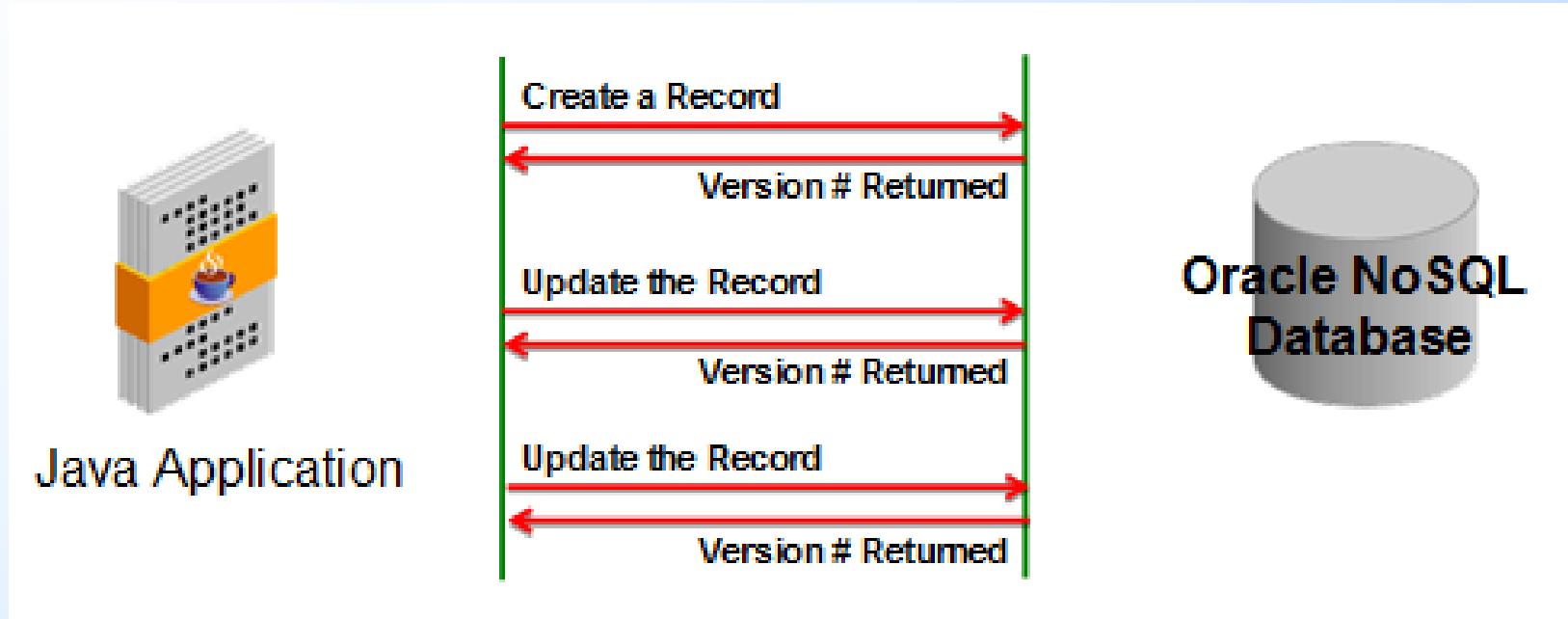
Implementarea Durability

- When using synchronization-based durability policies, you can control which of these operations the master node will wait to complete before it considers the write process to be completed.
- The master node's performance will become slower if it must wait to complete more operations.

Implementarea Durability

- When you use the acknowledgment-based durability policy, you are specifying that the master node must wait for acknowledgements from the replica nodes before considering the write operation to be completed.
- You can set the master node to wait for acknowledgements from all replica nodes, no replica nodes, or a majority of replica nodes.
- If the master node requires more acknowledgements, its write performance will become slower.

Versioning



- With this Oracle NoSQL Database feature, a version token is maintained each time a record is updated.

Versioning

- Oracle NoSQL Database automatically maintains a version number when a record is initially inserted and each time it is updated. This version information is important for two reasons:
 - When performing an update or a delete, you might want to perform the operation only if the record's value has not changed.
 - When performing a read operation, you might want to ensure that you read the value that was previously written.

Exemplu Versioning

```
public class CreditCard
{
    public boolean approveCharge(double amount, DbStore dbs)
    {
        boolean updateDone = false;
        while (!updateDone) {
            final ReturnValueVersion oldValueVersion =
                new ReturnValueVersion
                (ReturnValueVersion.Choice.ALL);

            final Version newVersion = dbs.getStore().putIfVersion
                (creditCardKey, newValue, getVersion(),
                 oldValueVersion, null, 0, null );
        }
    }
}
```

Exemplu Versioning

- The previous slide illustrates an example from the Credit Card Approval application. It shows how to use the version attributes in the CreditCard class.
- Here, the credit card balance is updated if the version has not changed. If the version has changed, the existing Value and Version details are returned and used to retry the update.

Accesarea KVStore

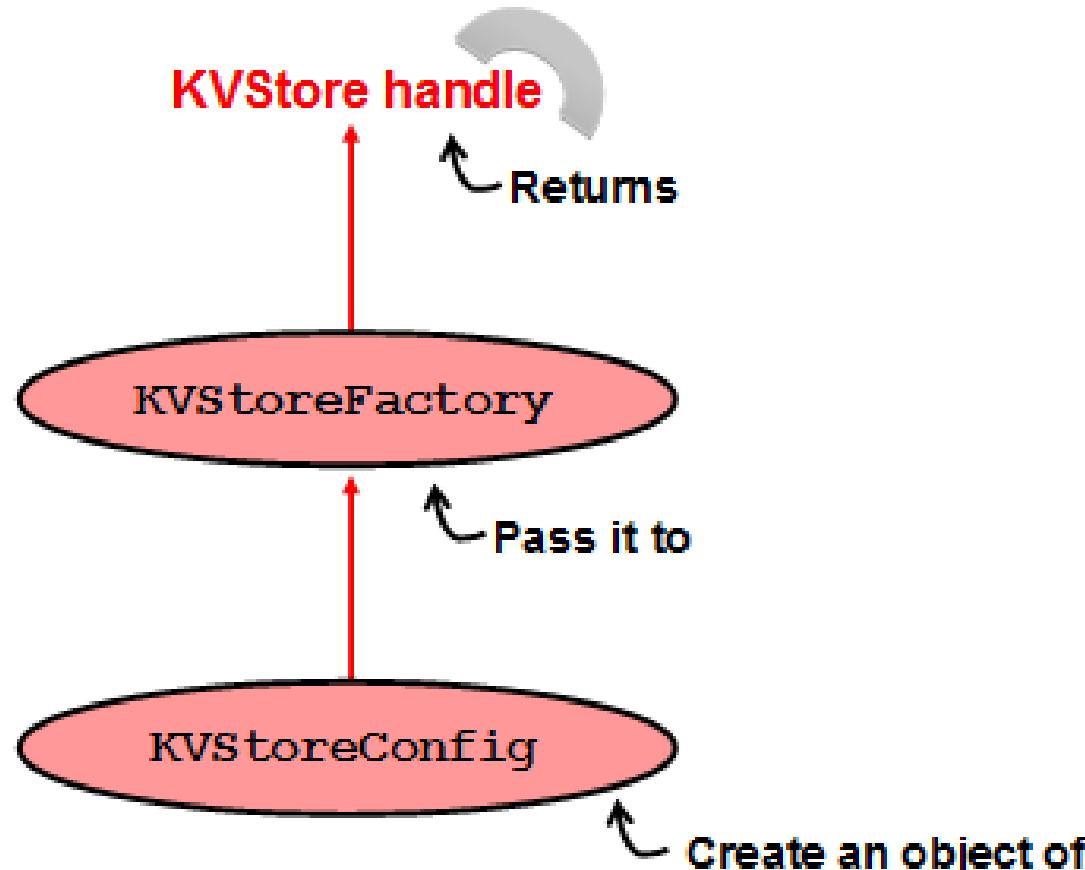
❑ KVStore Handle

- ❑ Is a resource that controls access to Oracle NoSQL Database
- ❑ Is used to open and close an already running store
- ❑ Is required to perform any operation on the store

Accesarea KVStore

- Any kind of access to Oracle NoSQL Database is performed by first obtaining a KVStore handle. The KVStore handle is a resource that is required to access the KVStore and is also used to open and close an already-running store.
- You need to access the KVStore whenever you perform the following operations in Oracle NoSQL Database:
 - Create
 - Read
 - Update
 - Delete

Create a KVStore handle



Crearea KVStore handle

- To create a KVStore handle, you use the KVStoreFactory and KVStoreConfig classes. You first create an object of the class KVStoreConfig. You then pass this KVStoreConfig object to the KVStoreFactory class, which returns the KVStore handle.
- When a KVStore handle is obtained, the store is automatically opened.

Utilizarea KVStoreConfig

```
public KVStoreConfig( string storeName,  
                      string helperHostPort)
```



constructor

Some available methods:

- getStoreName ()
- setStoreName ()
- setRequestLimit ()
- setRequestTimeout ()
- setHelperHosts ()

KVStoreConfig

Utilizarea KVStoreConfig

- The KVStoreConfig class describes a KVStore handle. The previous slide shows the KVStoreConfig constructor, which is used to create an instance of the KVStoreConfig class.
- The storeName parameter specifies the name of the KVStore. It must be entirely uppercase or lowercase letters and digits.
- The helperHostPort parameter is a set of string values that specify the host name and port of an active node in the KVStore. Each string value must be in the format hostname:port. You must specify at least one helper host name and port.
- The KVStoreConfig class has a list of methods that you can use to get and set Store Name, Durability, Consistency, Request Timeout, and other parameters. For a complete list of these methods and their definitions, view the Javadoc from the installation documents.

Exemplu: Utilizarea KVStoreConfig

```
KVStoreConfig kconfig = new KVStoreConfig("teachStore",  
                                         "localhost:5000");
```

- The slide shows an example of using the KVStoreConfig class. Here, an object of type KVStoreConfig is created and initialized.
 - **kconfig** is the object name.
 - **teachStore** is the name of the KVStore, which must already be running.
 - **localhost:5000** is the host name and port number of the active node in the KVStore.

Exemplu: Utilizarea KVStoreFactory

```
public KVStoreFactory()
```

constructor

```
public static KVStore getStore (KVStoreConfig config)
```

method

KVStoreFactory

Exemplu: Utilizarea KVStoreFactory

- The KVStorefactory class is a static class. It has one method, which is called getStore.
- The getStore method takes a KVStoreConfig object as the input parameter and returns the KVStore handle to the store, which is specified in the KVStoreConfig object.

Exemplu: Utilizarea KVStoreFactory

```
KVStore kvstore = KVStoreFactory.getStore(kconfig);
```

- The slide shows an example of using the KVStoreFactory class. Here, an object of type KVStore is created and initialized.
 - **kvstore** is the object name. The value for this object is obtained by calling the getStore method of the KVStoreFactory class and passing an object of type KVStoreConfig.
 - **kconfig** is the object that is passed to the getStore method.

Java Packages for Creating a KVStore Handle

- The following packages should be imported to a Java class before creating a KVStore handle:
 - oracle.kv.KVStore
 - oracle.kv.KVStoreConfig
 - oracle.kv.KVStoreFactory

Exemplu: Crearea KVStore Handle

```
package teach;

import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;

public class createStoreHandle
{
    KVStore myStore;
    public static void main(String args[])
    {
        KVStoreConfig kconfig = new
            KVStoreConfig("teachStore","localhost:5000");
        KVStore myStore = KVStoreFactory.getStore(kconfig);
    }
}
```

Credit Card Approval Application

```
package developerday;

import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
public class DbStore
{
    private final KVStore myStore;

    DbStore(String sname, String host, String port) {
        myStore = KVStoreFactory.getStore
            (new KVStoreConfig(sname, host + ":" + port));
    }

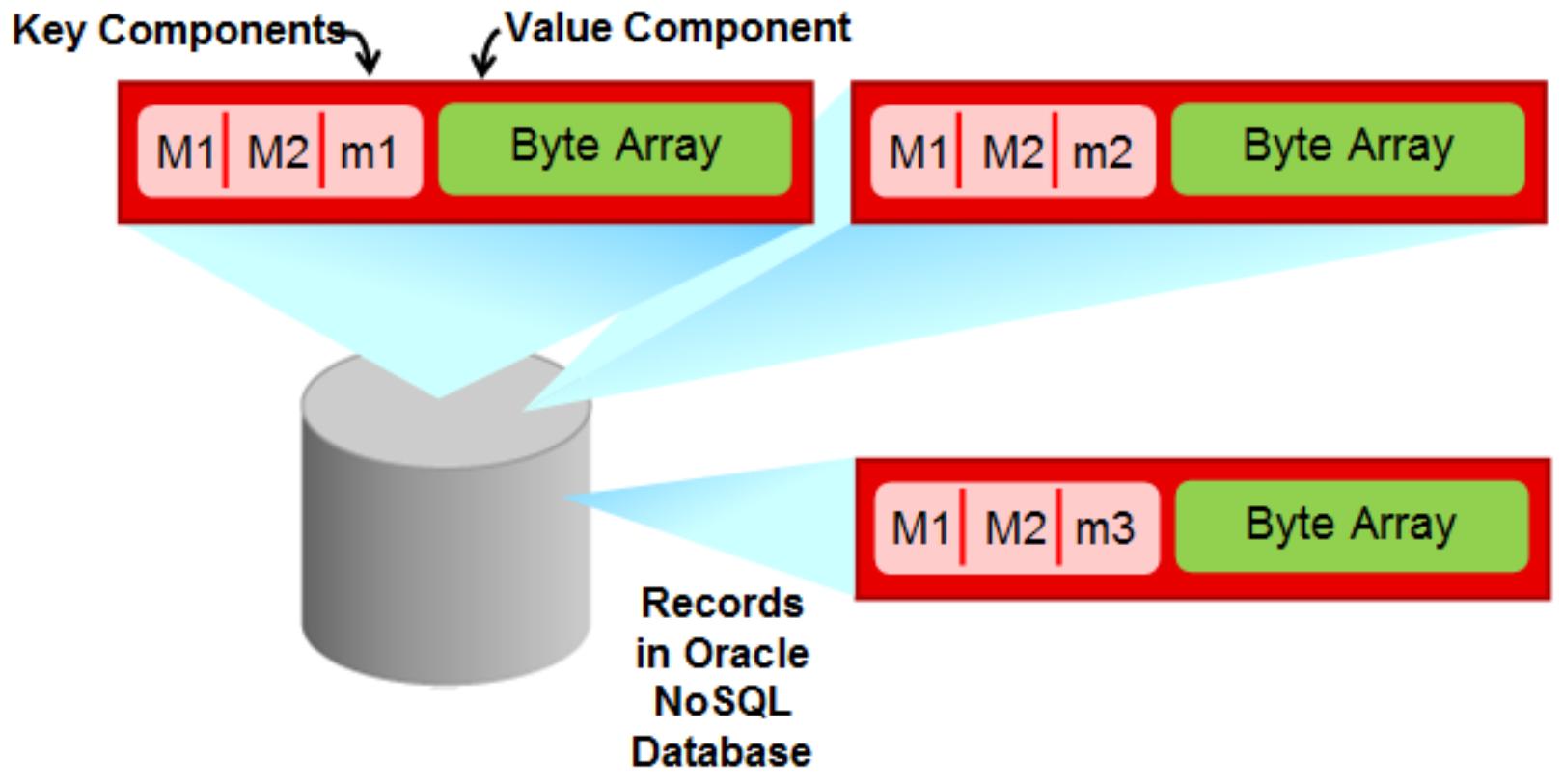
    void close () { myStore.close(); }

    public KVStore getStore () { return myStore; }
}
```

Credit Card Approval Application

- The DbStore class of the Credit Card Approval application contains the code to create a KVStore handle.
- myStore is the variable of KVStore type. The value for this variable is set using a constructor. The constructor takes the name of the store whose handle is to be obtained along with the host name and port number as input parameters. In the constructor, the KVStoreFactory and KVStoreConfig classes are used to obtained the KVStore handle.
- The DbStore class also has two methods:
 - **getStore**: Returns the myStore handle to the calling method
 - **Close**: Closes the store and automatically releases all resources

Componentele unei înregistrări



Componentele unei înregistrări

- Recall that a record consists of a key component and a value component. Also, a key component can consist of major and minor components. If a record consists of major and minor components, each unique combination of major and minor components results in a single record.
- For example, the previous slide illustrates a case where there are two major components and one minor component with three different values. In this lesson, you learn how to construct a record with the structure as shown in the slide. The major components are a person's first and last names. The three values of the minor component are info, image, and voice.

Creare componente Major-Key

- To create major-key components in a Java program:
 1. Create an ArrayList of the String data type, named majorPath.
 2. Add the major-key component values to this ArrayList.

```
List<String> majorPath = new ArrayList<String>();  
...  
majorPath.add("Smith");  
majorPath.add("Bob");
```

Creare componente Major-Key

- The simplest way to create major components is to define an array of the data type `String`.
It is mandatory that you define the data type as `String`, because the key component in Oracle NoSQL Database is restricted to the `String` data type.
- **Note:** You use an `ArrayList` here because the major key has two components. In a case where the major key has only one component, you can consider using a simple `String` data type.

Creare componente Major-Key

- As a naming convention best practice, you can use `majorPath` as the name of the array to store the major-key components. After the array is defined, you can add the major-key component values to the array.
- The slide example adds Smith and Bob to the `majorPath` array.

Creare componente Minor-Key

- To create minor-key components in a Java program:
 1. Create a variable of the `String` data type, named `minorComponent`.
 2. Set the minor-key component value to this variable.

```
String minorComponent;  
...  
minorComponent = "info";
```

Creare componente Minor-Key

- Creating the minor components is similar to creating the major components. You should define a variable of the `String` data type.
- **Note:** If the minor key has more than one component, you can consider storing the key values in an `ArrayList` of `String` data type.
- Again, as a best practice, you can name the variable `minorComponent`.
- The slide example defines a `String` variable called `majorComponent` and assigns the value `info` to this variable.

Creare Key pentru înregistrare

```
Key myKey = Key.createKey(majorPath,  
                           minorComponent);
```

- After you have defined the major- and minor-key components, use the `createKey` method to create the key. The slide shows an example of using the `createKey` method.
 - `myKey` is a variable of type `Key`.
 - `majorPath` contains the major-key component values.
 - `minorComponent` contains the minor-key component values.

Creare Value pentru înregistrare

```
List<String> data = new ArrayList<String>();  
  
data.add("35");  
data.add("male");  
data.add("developer");  
  
Value myValue = Value.createValue(data.getBytes());
```

- The value component of a record is created by using a `createValue` method. Before using this method, create the data content of the record. Either you create a variable of the `String` data type and store the data in it, or you can create an `ArrayList` and add all the data values to the array.
- The slide example creates an `ArrayList` of the `String` data type called `data`. The data values are added to this `ArrayList`.
- The `createValue` method accepts values of the `byte` data type and creates a `value` that can be stored in a `KVStore`.

Required Java Packages for Constructing a Record

- The following packages should be imported to a Java class before constructing a record:
 - `java.util.ArrayList`
 - `oracle.kv.Key`
 - `oracle.kv.Value`

Exemplu: Construire înregistrare

```
public class createRecord
{
    public static void main(String args[])
    {
        List<String> majorPath = new ArrayList<String>();
        String minorComponent;
        String data;
        majorPath.add("Smith");
        majorPath.add("Bob");
        minorComponent = "info";
        Key myKey = Key.createKey(majorPath,
                                   minorComponent);
        data="Male, 35, Developer";
        Value myValue = Value.createValue(data.getBytes());
    }
}
```

Reviewing Sample Code: Credit Card Approval Application

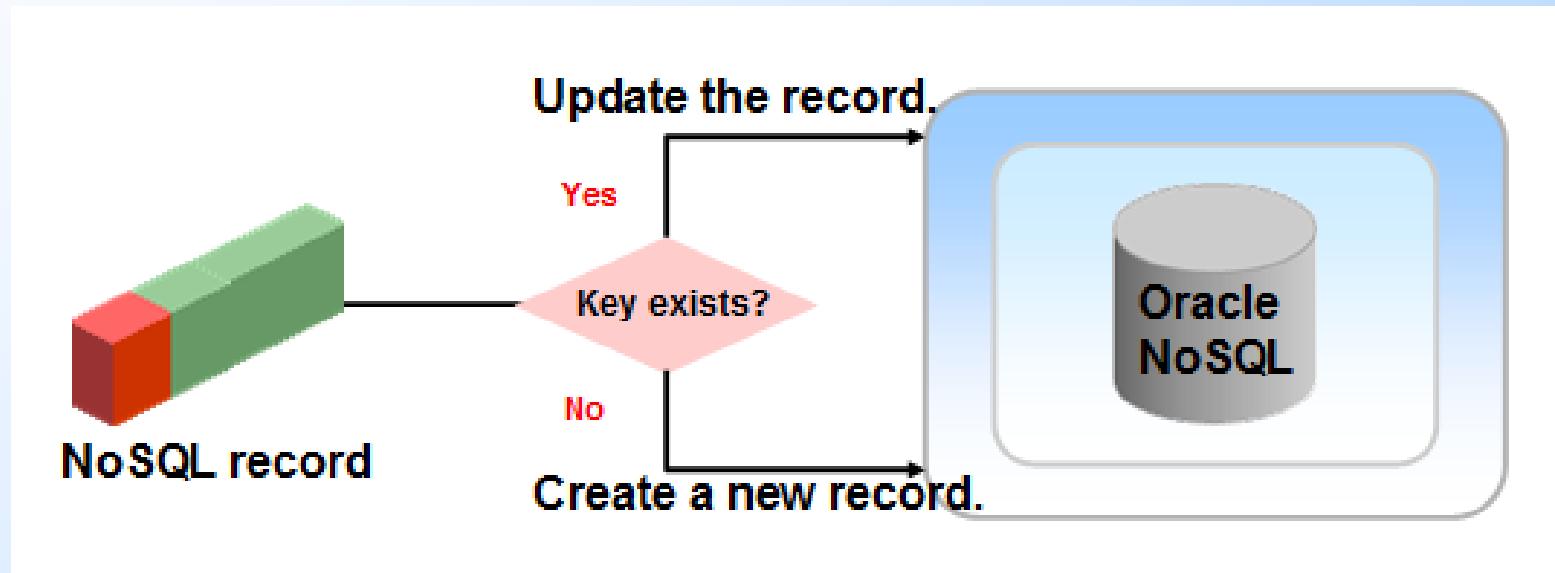
```
public class KeyDefinition
{
    static final String APRV_PROPERTY_NAME = "aprsv";
    static final String INFO_PROPERTY_NAME = "info";

    static Key makeUserAprvKey(String ccNumber)
    {
        return Key.createKey(ccNumber,
APRV_PROPERTY_NAME);
    }
    static Key makeUserInfoKey(String ccNumber)
    {
        return Key.createKey(ccNumber,
INFO_PROPERTY_NAME);
    }
}
```

Reviewing Sample Code: Credit Card Approval Application

- The `KeyDefinition` class of the Credit Card Approval application contains the code to create the key component. This class contains two methods:
 - `makeUserAprvKey`: Takes the credit card number and returns a key with minor key value `aprvt`
 - `makeUserInfoKey`: Takes the credit card number and returns a key with minor key value `info`

Procesul Write



Processul Write

- Records are written to a KVStore based on their key. If a record with a specified key does not exist in the store, the record is created in the KVStore.
- If there is an existing record in the store with the specified key, the record is updated with the data that is being passed.
- However, there are methods available that enable you to write a record if it either exists or does not exist.

Processul Write

□ To write a record to the store:

1. Construct a key.
2. Construct a value.
3. Use an API to write the record to the store.

□ You have already learned how to construct a key and a value. Next, you learn how to write a constructed record to the store.

API Java pentru procesul Write

- Use the following APIs to write records to a KVStore:
 - put ()
 - putIfAbsent ()
 - putIfPresent ()
 - putIfVersion ()

API Java pentru procesul Write

- You can use any one of the four methods listed in the slide to write records to a KVStore.
 - **put ()**: Writes a record to the KVStore by either creating a new record or overwriting an existing record as appropriate
 - **putIfAbsent ()**: Writes a record to the KVStore only if there is no existing record with the specified key
 - **putIfPresent ()**: Writes a record to the KVStore only if there is already an existing record with the specified key
 - **putIfVersion ()**: Writes a record to the KVStore only if the existing record's version matches the Version argument specified with this method

Handling Write and Delete Exceptions

- Handle the following exceptions when writing to or deleting from a KVStore:
 - DurabilityException
 - RequestTimeoutException
 - FaultException

Procesul Write cu put()

```
Version put (Key key, Value value)  
throws DurabilityException,  
RequestTimeoutException,  
FaultException
```

- The slide shows the simplest definition of the put () method. It takes two arguments: the key part and the value part of the record to be inserted. It returns the version number of the inserted record.
- In this method, the default durability and timeout values are used.

Exemplu: Procesul Write

```
package teach;

import java.util.ArrayList;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv

public class WriteRecord
{
//Create Handle
//Create Record

myStore.put(myKey,myValue);

}
```

Reviewing Sample Code: Credit Card Approval Application

```
package developerday;

public class CreditCardExample
{
    private void loadAccount ()
    {
        final List<Operation> ops = new ArrayList<Operation> ();
        ops.add(factory.createPutIfAbsent
                    (cc.getStoreKey (),
                     cc.getStoreValue (),
                     null /*prevReturn*/,
                     true /*abortIfUnsuccessful*/));
    }
}
```