

Baze de Date

Cap. 1. BD. SGBD. Algebra relațională



2023 UPT

Conf. Dan Pescaru

Organizare curs

1. Curs BD

- Conf.dr.ing. Dan Pescaru
- Miercuri, 8:00-10:00

2. Laborator

- B623 / B529
- Oracle Cloud

3. Evaluare

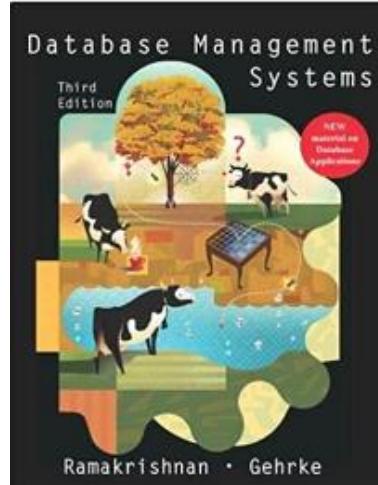
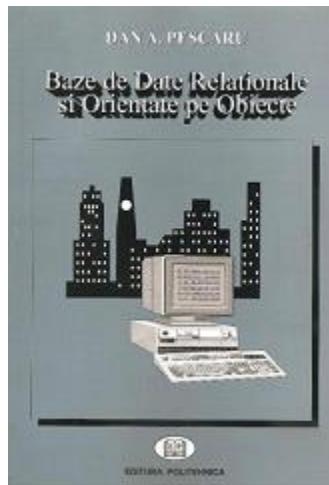
- Nota Laborator (1/2)
- Nota Examen (1/2)

Conținut

1. Introducere. BD. SGBD. Algebra relațională.
2. SQL utilizare DDL. Gestiune date. Interogari active.
3. SQL implementare Proiecție, Selectie și Join.
4. SQL utilizare subinterrogări. Operații pe mulțimi.
5. SQL agregare date. Obținerea informațiilor statistice.
6. Implementare aplicațiilor client folosind Oracle APEX.
7. Implementarea unei BD Web folosind MySQL și PHP.
8. Introducere in modelarea si organizarea datelor intr-o BD relațională.
9. Optimizarea accesului la o BD. Utilizare indecsi.
10. Introducere in optimizarea interogarilor SQL.
Generare plan de execuție.

Bibliografie

1. D. Pescaru, "Baze de Date Relationale si Orientate pe Obiecte", Editura Politehnica, Timisoara, 2001, ISBN 973-8247-53-5.
2. R- Ramakrishnan și J. Gehrke, "Database Management Systems", 3rd edition, 2003, ISBN 007-2465-63-8, McGraw-Hill.
3. Oracle® 21c Database SQL Language Reference, Jan. 2023
<https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/>



Oracle Database
Documentation Library

Exemplele din curs/laborator

1. Tehnologii utilizeaza

- Oracle, Oracle XE, APEX
- MySQL
- Visual dBase Plus

2. Limbaje

- De interogare date: SQL, xBase
- De programare: Java, Javascript, PHP

Managementul datelor

1. Solutia simplă: fișierul
2. Organizarea fișierelor: structuri de directoare
3. Avantaje
 1. Simplitate
 2. Acces liniar (text) sau secvențial (binar)
4. Dezavantaje:
 1. Căutarea datelor în fișiere – algoritmi complecși specifici fiecărei aplicații
 2. Slaba protecție a datelor
 3. Controlul accesului și securitate – la nivelul SO

DB și SGBD

1. Solutia: utilizarea unei baze de date
2. Baza de date (**BD**) = o alternativă de stocare adecvată pentru colecții de date de mari dimensiuni. Include organizarea fizică eficientă pe suport extern, algoritmi avansati de căutare, regăsire, protecție și mecanisme de securitate a datelor
3. Sisteme de gestiune a bazelor de date (**SGBD**) = un sistem software proiectat să asigure stocarea și gestionarea unor baze de date

De ce sunt importante bazele de date?

1. Largă răspândire în lume: majoritatea oamenilor le utilizează în fiecare zi

- 1. Majoritatea site-urilor Web utilizează BD
(magazine virtuale, forumuri, broadcasting etc.)**
- 2. Sistemele de telefonie**
- 3. Sistemele bancare**
- 4. Sistemele de facturare din magazine, etc.**

2. Există o cerere constantă de specialiști în domeniul bazelor de date (administratori, proiectanți, analiști, programatori etc.)

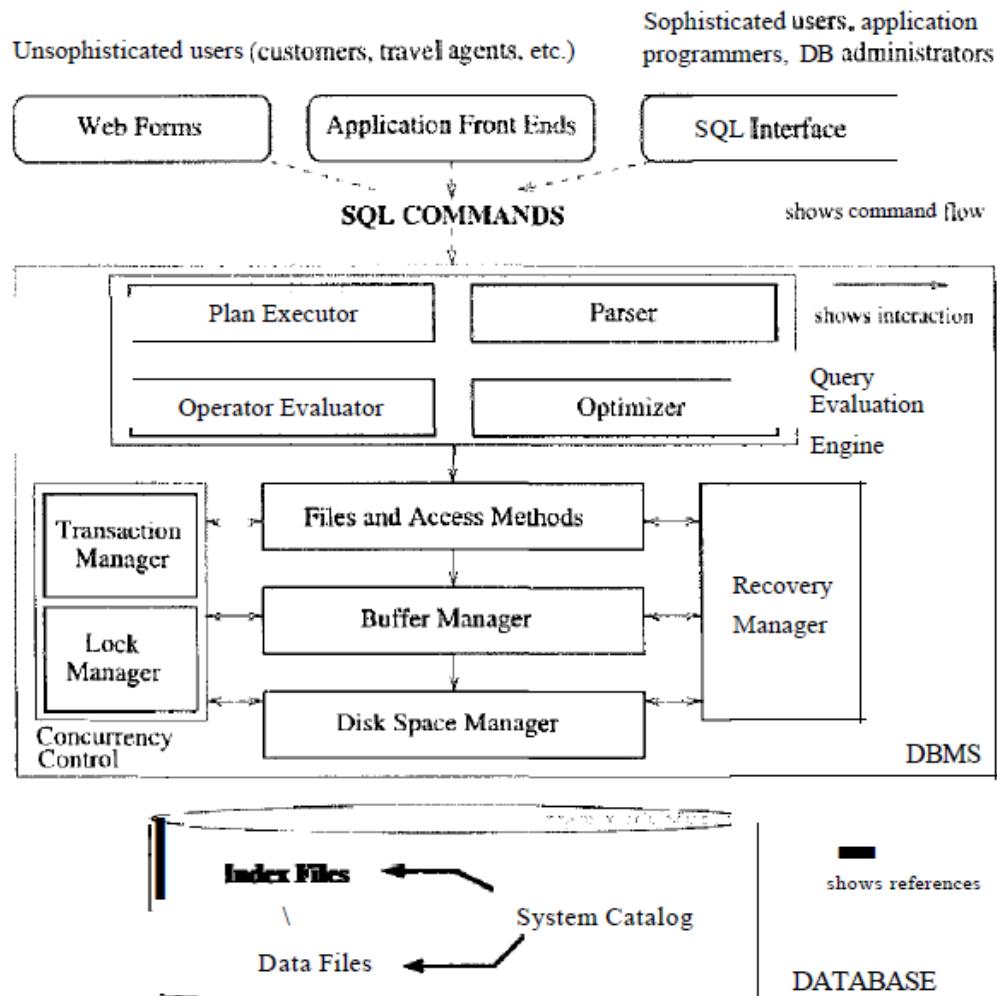
Meserii legate de BD

1. Implementator SGBD (angajați de Oracle, Microsoft, IBM etc.)
 - Creează sistemele de gestiune a bazelor de date
2. Utilizatori finali – un număr cvasi-infinnit
 - Manageri, secretare, contabili, bancheri, etc.
3. Programatori de aplicații (SQL + limbaje imperative)
 - Dezvoltă aplicații prin care utilizatorii interacționează cu BD
4. Administratori BD
 - Proiectează schemele conceptuale și fizice ale BD
 - Gestionează securitatea datelor
 - Asigură recuperarea datelor și disponibilitatea lor
 - Optimizează performanța BD (DB benchmarking/tunning)

Avantajele utilizării SGBD

1. Independența datelor de aplicații
2. Eficiența accesului la date
 - Utilizare sisteme complexe de indexare
 - Optimizarea interogărilor
3. Suport pentru tehnici RAD
4. Asigură integritatea datelor
5. Asigură securitatea datelor
6. Scalabilitate

Aritectura unui SGBD



*Ref: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, 2003

Scurtă istorie a domeniului BD

1. Modelul ierarhic – 1960

- Structură arborescentă (relații 1-la-N)
- Ex. IBM IMS – Information Management System – utilizat în programul Apollo (inventarul pieselor pentru racheta Saturn V)
- American Airlines + IBM – SABRE



NASA: <http://www.hq.nasa.gov/>

1. Modelul rețea – 1962

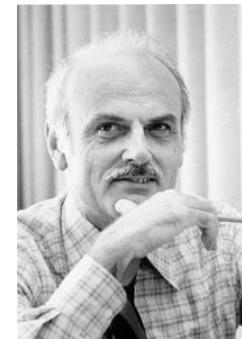
- Strutură de garf deneralizat (N-la-M)
- IDS (Integrated Data Store) - realizat de Charles Bachman pentru General Electric (în Cobol)
 - » A primit premiul ACM Turing (\leftrightarrow Nobel) în 1970 pentru activitatea sa în domeniul bazelor de date



Actual - modelul relațional

1. Modelul relațional – 1970

- Edgar Codd, la IBM's San Jose Research Laboratory
 - A câștigat Turing Award in 1981
- Noutatea: are la bază un model matematic
- Sisteme navigaționale (imperative) pentru desktop
 - xBase: FoxPro, Clipper, Visual dBase **dBASE™**
- Sisteme declarative – SQL (IBM's System R project)
 - Oracle, IBM DB2, Ms SQL Server, MySQL



Modele alternative

1. Modelul obiectual

- Obiecte persistente în limbajele OO
- ODMG - Object Query Language (OQL) 
- Ex: Jasmine (Fujitsu), Versant , O2, POET, ObjectStore, JADE
- Soluție de compromis: model obiectual-relațional



2. Modele NoSQL (post-relaționale)

- Diverse: de la modele de tip cheie-valoare la BD orientate pe documente
- Ex: MongoDB, Redis, Apache Cassandra etc.)



Cassandra



mongoDB®



Proiectare unei BD

1. Analiza cerințelor: primul pas la proiectarea BD

2. Se utilizează un model semantic

- Abstract, de nivel înalt, capabil să descrie semantica datelor (indiferent de implementare)
- Servește la înțelegerea datelor și a rolului lor

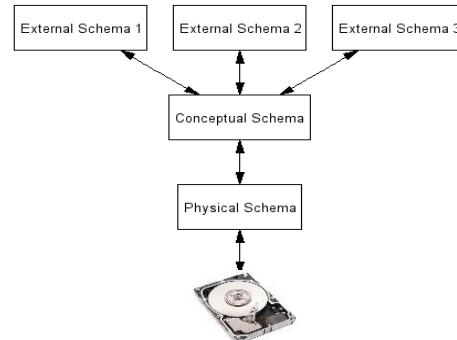
3. Limbaje de modelare semantice

- Entity Relationship (ER) – utilizează imagini pentru a descrie entități și relațiile dintre ele (modelare la nivel logic)
- UML – tot grafic dar mai general decât ER (modelare business-operare, sistem-integrare, logic, fizic-stocare, hardware)

Nivele de abstractizare la implementare

1. Datele sunt descrise într-un SGBD pe trei nivele de abstractizare

- **Fizic**: cum și unde sunt stocate fizic datele
- **Conceptual**: modelul datelor
- **External**: model simplificat destinat unei anumite aplicații



MODELUL RELAȚIONAL

1. BD relatională: o colecție de relații (tabele) și legăturile dintre ele
2. Relație (tabel):
 - Schemă: nume relație, atrbute (coloane) și domeniul atributelor (tipul datelor)
 - Ex: Studenti(marca:String, nume:String, an:Integer, nota:Real)
 - Instantă: tabelă fizică, având un număr fix de coloane și un număr variabil de rânduri (înregistrări)
 - Numărul de coloane = GRADUL relației (arietatea)
 - Numărul de rânduri = CARDINALITATEA relației
 - Obs.: Rândurile (înregistrările) NU sunt ordonate!
Coloanele SUNT ordonate (ex. pt. SQL INSERT)

Instanță (tabelă)

Student

marca	nume	an	media
AC2153	Pop Angela	2	8.50
AC1078	Avram Ioan	1	9.35
AC2056	Ionescu Mihai	2	7.80
AC3098	Georgescu Ana	3	9.00
AC3023	Mihu Andrei	3	6.30

- Relația Student
 - Grad: 4
 - Cardinalitate: 5
- Obs: *Contine doar înregistrări distincte!*

Constrângerile de integritate

1. O colecție de condiții (expresii) logice care trebuie să fie îndeplinite de orice instanță a relației
2. Constrângerile de integritate:
 - Specificate la crearea schemei BD (tabelelor)
 - Verificate automat la orice modificare a datelor
3. O instanță este legală ("într-o stare consistentă") dacă satisfac toate constrângerile de integritate. Verificarea consistentei se face automat de SGBD
4. Obs.: Verificarea constrângerilor previne în mare măsură introducerea de date eronate

Chei

1. Un set de attribute formează o cheie a unei relații dacă:
 - a. Nu există două înregistrări care să aibă toate valorile atributelor identice (asigură unicitatea)
 - b. Orice subset nu respectă proprietatea anterioară (cheia este minimală)
2. Supercheie: un set de attribute care conține cel puțin o cheie

Cheie Primară (PK)

1. Cheie primară: o cheie aleasă de proiectant din setul de chei a unei relații, a.î. să satisfacă condițiile:

- Este scurtă (număr minim de attribute – ideal unul singur)
 - Obs: (neuzual) o singură cheie cu toate câmpurile relației
- Este reprezentativă în contextul problemei (ex. *marca* vs. *cnp*)
- Nu admite valori NULL

2. Alternative: crearea unei chei artificiale (de obicei numerică de tip *AutoIncrement*)

Integritatea referențială

1. Cheie externă (FK) – legă logic două tabele (va corespunde cheii primare din cea de a doua tabelă)
2. Utilizată pentru verificarea integrității referențiale
 - La adăugare în tabelă: valoarea cheii trebuie să existe în tabela referită
 - La ștergerea unei înregistrări din tabela referită: se va evita crearea de înregistrări "orfane"
 - La modificarea valorii unei chei externe sau a unei chei primare în tabela referită

Asigurarea integrității referențiale

1. La adăugare (INSERT)

- Se blochează adăugarea unei chei fără corespondent

2. La ștergere (DELETE) se poate alege

- "Cascade delete related records"
- "Avoiding operation that breaks constraints"
- "Voiding the reference" (nu prea des)

3. La modificare (UPDATE) se poate alege

1. "Cascade update related records"
2. "Avoiding operation that breaks constraints"

4. Probleme: blocaje (deadlocks – referințe încrucișate), Soluție: utilizarea tranzacțiilor (verificare întârziată)

Integritate referențială. Example

Student

marca	nume	an	media
AC2153	Pop Angela	2	8.50
AC1078	Avram Ioan	1	9.35
AC2056	Ionescu Mihai	2	7.80

Contract

cid	marca	lab	ex
PLA2	AC2153	9	8
UC1	AC1078	10	9
SO2	AC2056	8	7

1. **INSERT** un nou contract: verifică **marca** in tabela Student
2. **DELETE** sterge un student: verifică **marca** in Contract
3. **UPDATE** modifică **marca** in Student: verifică **marca** in Contract / modifică **marca** in Contract – verify **marca** in Student

Algebra Relațională

1. Derivată din Algebra "clasică"
2. Operatori de bază:
 - Proiecție (Π) – selectează coloanele specificate
 - Selectie (σ) – selectează înregistrările specificate
 - Produs Cartezian (\times) – combină două relații
 - Diferență mulțimi (\setminus) – înregistrările din R1, care nu \in R2
 - Reuniune (\cup) – înregistrările din R1 și R2 (fară duplicate)
3. Operatori adiționali:
 - JOIN (\bullet), Intersecție (\cap), Divizare (\div), Redenumire
4. Din moment ce orice operator returnează o relație, ei pot fi compuți în expresii (Algebra este "închisă")

Proiecția

1. Proiecția (Π) – Selectează doar attributele specificate în lista de proiecție
2. Schema rezultatului va conține exact câmpurile din lista de proiecție, cu același nume ca și cel din relația de intrare
3. Operatorul de proiecție trebuie să eliminate duplicatele! (*De ce ??*)
4. Obs: sistemele reale în general nu elimină duplicatele duplicates decât dacă se specifică de către programator. (*De ce nu ??*)
5. Ex: $\Pi_{\text{media}}(\text{Student}) = \{10.0, 9.5, 8, 6.5\}$

Selectia

1. Selectia (σ) – selectează rândurile care satisfac condiția de selecție
2. Condiția de selecție: simplă (compară attribute cu valori) sau complexă (utilizează conectori logici precum AND , OR , NOT)
3. Schema rezultatului va fi identică cu cea a relației de intrare
4. Obs: Nu apar duplicate în rezultat! (*De ce??*)
5. Cardinalitate rezultat \leq Cardinalitate intrare
6. Ex: $\sigma_{\text{media} < 9.0}(\text{Student})$

Reuniunea, Intersecția, Diferența

- Toate au doi operanți (relații) compatibile ("union-compatible"):
 - Au același număr de attribute
 - Attributele corespondente au același tip

S1

marca	nume	an	media
AC2034	Popescu Ion	2	9.25
AC2056	Ionescu Vasile	2	8.70

marca	nume	an	media
AC2056	Ionescu Vasile	2	8.70

S1 \cap S2

marca	nume	an	media
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

S2 / S1

S2

marca	nume	an	media
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

marca	nume	an	media
AC2034	Popescu Ion	2	9.25
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

S1 \cup S2

Produsul cartezian

1. Fiecare rând din R1 este împerecheat cu fiecare rând din R2
2. Schema rezultatului va avea câte un atribut pentru fiecare atribut din cele două relații, cu același nume (conflictele rezolvate prin redenumire)

Student

marca	nume	an	media
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

Masina

marca	nrm	tip
AC2056	TM06ABC	Dacia Logan
ET3045	TM01AAA	Renault Megane
ME1078	TM08BBB	Ford Focus
AC1012	TM09PPP	Opel Corsa

(marca)	nume	an	media	(marca)	nrm	tip
AC2056	Ionescu Vasile	2	8.70	AC2056	TM06ABC	Dacia Logan
AC2056	Ionescu Vasile	2	8.70	ET3045	TM01AAA	Renault Megane
AC2056	Ionescu Vasile	2	8.70	ME1078	TM08BBB	Ford Focus
AC2056	Ionescu Vasile	2	8.70	AC1012	TM09PPP	Opel Corsa
ET3045	Vasilescu Ana	3	9.00	AC2056	TM06ABC	Dacia Logan
ET3045	Vasilescu Ana	3	9.00	ET3045	TM01AAA	Renault Megane
...

SxM

Join

1. JOIN = selecție cu o condiție de Join peste un produs cartezian
2. $R1 \circ_{J\text{cond}} R2 = \sigma_{J\text{cond}}(R1 \times R2)$

Student

marca	nume	an	media
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

Masina

marca	nrm	tip
AC2056	TM06ABC	Dacia Logan
ET3045	TM01AAA	Renault Megane
ME1078	TM08BBB	Ford Focus
AC1012	TM09PPP	Opel Corsa

(marca)	nume	an	media	(marca)	nrm	tip
AC2056	Ionescu Vasile	2	8.70	AC2056	TM06ABC	Dacia Logan
ET3045	Vasilescu Ana	3	9.00	ET3045	TM01AAA	Renault Megane
ME1078	Pop Angela	1	8.25	ME1078	TM08BBB	Ford Focus
AC1012	Ticu Gelu	1	9.50	AC1012	TM09PPP	Opel Corsa

Student \circ $\sigma_{\text{Student.marca}=\text{Masina.marca}}$ **Masina**

Join

1. Aceeași schemă a rezultatului ca și la produs cartezian dar mai puține înregistrări. Se poate procesa mai eficient. (*Cum??*)
2. Tipuri de JOIN:
 - Theta-join – condiție generală de Join ($=, <, >$)
 - Equi-join – caz special care testează doar egalitate ($=$)
 - Natural-join – Equi-join pe toate câmpurile comune (implicit în lipsa condiției de \bowtie)
 - Self-join – R1 și R2 sunt aceeași relație

Exemple

Rezervări bărci într-un port – Schema BD apare în ref. [1]

Marinar (M)

<u>mid:int</u>	<u>mnume:string</u>	<u>rating:int</u>	<u>varsta:real</u>
22	Dustin	7	45.0
...			
58	John	10	52.0

Barca (B)

<u>bid:int</u>	<u>bnume:string</u>	<u>culoare:string</u>
103	Clipper	verde
...		
127	Neptun	albastru

Rezervare (R)

<u>mid:int</u>	<u>bid:int</u>	<u>data:date</u>
58	103	02/11/2010
...		
22	127	09/02/2013

[1] Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, 2003

Exemple interogări

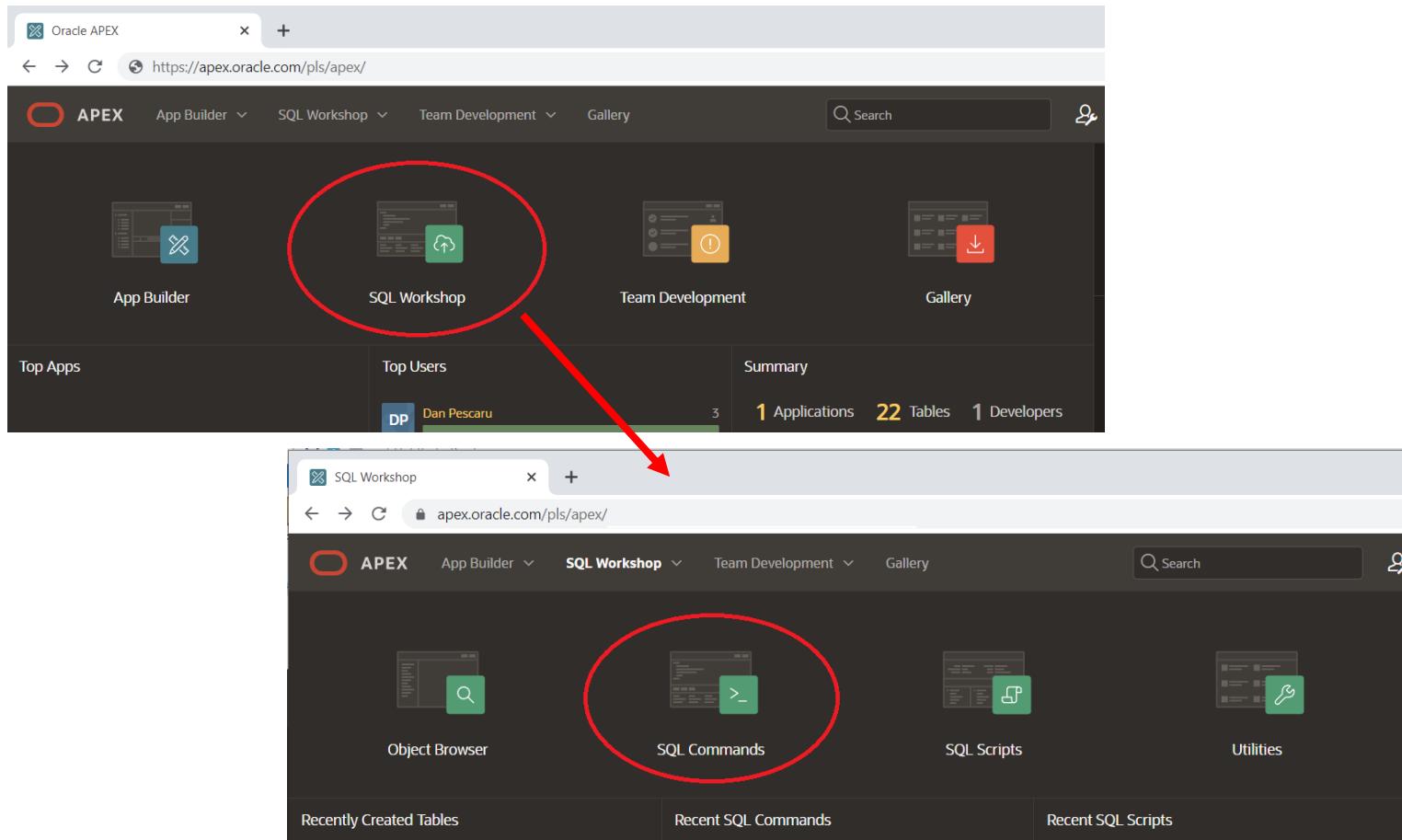
- Problemă: aflați toti marinarii care au rezervat o barcă verde
- Soluții:
 1. $\pi_{\text{mid, numeM}}(\sigma_{\text{culoare}=\text{'verde'}}((M \circ_{M.\text{mid}=R.\text{mid}} R) \circ_{R.\text{bid}=B.\text{bid}} B))$
 2. $\pi_{\text{mid, numeM}}((S \circ_{M.\text{mid}=R.\text{mid}} R) \circ_{R.\text{bid}=B.\text{bid}} (\sigma_{\text{color}=\text{'verde'}}(B)))$
- Două planuri de execuție diferite
- Care credeți că este cel mai eficient?

Limbaje de interogare relaționale

1. Limbaje de interogare relaționale: proiectate pentru a interoga, modifica și gestiona date
2. Modelul relational permite limbaje de interogare simple și puternice (ex. SQL,QBE):
 - Fundație puternica bazată pe algebra relatională
 - Permite optimizarea automată a interogărilor
3. Limbajele de interogare nu sunt limbaje de programare:
 1. Nu sunt Turing-complete (SQL92), dar extensii ale lor pot fi (ex. Oracle PL/SQL)
 2. Ex. Inchidere recursivă nu poate fi exprimată
4. Dar ele asigură accesul la date complexe!

Rulare example curs in Oracle APEX

1. Exemplele pot fi rulate folosind APEX SQL Workshop / SQL Commands



Rulare cod SQL

1. Codul se introduce in fereastra de editare (1) si apoi se apasă butonul [Run] (2).
Obs: tabelele trebuie create in prealabil

The screenshot shows the Oracle SQL Workshop interface. In the top-left, there's a browser-like header with 'SQL Commands' and a URL 'https://apex.oracle.com/pls/apex/'. Below it is a navigation bar with tabs for 'APEX', 'App Builder', 'SQL Workshop' (which is selected), 'Team Development', and 'Gallery'. A search bar and user profile 'Dan Pescaru' are also present. The main area is titled 'SQL Commands' and contains a code editor with the following SQL command:

```
1 SELECT * FROM Student;
```

The line number '1' is highlighted with a red circle labeled '1'. The 'Run' button at the top right of the editor is also highlighted with a red circle labeled '2'. Below the editor, a 'Results' tab is active, displaying a table with four rows of student data:

MARCA	NUME	AN	MEDIA
2	Ana Olteanu	3	9.25
4	Nicoleta Coman	1	7.75
3	Eduard Stoica	2	9.5
1	Alexandru Popescu	1	8.45

At the bottom left of the results table, it says '4 rows returned in 0.01 seconds'.

Baze de Date

Cap. 2. SQL - utilizare DDL, gestiune date, interogari active



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, Cap. 3

2023 UPT

Conf.Dr. Dan Pescaru

Introducere

1. SQL – Structured Query Language
2. IBM SystemR project - SEQUEL (1970)
(Structured English
Query Language)
3. Oracle V2 (1979)
4. Standardul curent in
industrie (ANSI/ISO)
5. Limbaj declarativ
6. SGBD-uri comerciale: dialecte SQL + extensii
procedurale (Oracle PL/SQL, Ms T-SQL etc.)



Standardul SQL

1. ANSI/ISO:

- SQL'1986 (SQL-87) – Propus de ANSI
- SQL'1992 (SQL2) – revizie majoră (ISO 9075), cel mai folosit în zilele noastre în BD relationale de uz general
- SQL'1999 (SQL3) – adaugă interogări recursive și tipuri de date orientate pe obiecte
- SQL'2003 (SQL/XML) – adaugă facilități XML
- SQL'2011 – adaugă suport pentru date temporale
- SQL'2016 – adaugă suport pentru tipare și JSON
- SQL'2019 – în dezvoltare – suport pentru tablouri multidimensionale

Limbajul SQL

1. Data Definition Language (DDL)

- Instrucțiuni pentru definirea unei baze de date și a obiectelor acesteia (tabele, indecsi, constrângeri etc.)

2. Data Manipulation Language (DML)

- Interogări pentru modificarea și extragerea datelor dintr-o BD

3. Data Control Language (DCL)

- Instrucțiuni pentru control al BD precum administrare, privilegii și tranzacții

De ce SQL?

1. DDL

- Creare/ștergere bază de date, tabele și vederi
- Creare/ștergere de secvențe și indecsi
- Creare/ștergere proceduri stocate și triggere

2. DML

- Inserare/ștergere/modificare înregistrări
- Interogare date

3. DCL

- Adăugare/ștergere utilizatori, roluri și privilegii
- Include TCL (Transactions Control Language)

SQL DDL – Tipuri de date

1. Depind de SGBD
2. Esențiale pentru optimizarea cerințelor de spațiu de stocare
3. În cele mai multe cazuri include și un interval specific de valori posibile
4. Specifică operatorii asociati tipurilor
5. Obs: tipuri de date din limbajul de programare client necesită mapare

Tipuri de date ANSI SQL

1. Siruri de caractere

- **CHARACTER(n)** sau **CHAR(n)**: sir de n caractere cu lungime fixă
- **VARCHAR(n)**: sir cu lungime variabilă cu dimensiune maximă de n caractere

2. Numere

- **INTEGER**, **SMALLINT** și **BIGINT**
- **FLOAT**, **REAL** și **DOUBLE PRECISION**
- **NUMERIC(nr. cifre, zecimale)** și **DECIMAL(nr. cifre, zecimale)**

3. Dată calendaristică și timp

- **DATE**: pt. date calendaristice (ex. 2023-01-02)
- **TIME**: pentru momente de timp (ex. 13:50:12)
- **TIMESTAMP**: compus din **DATE** și **TIME** (ex. 2023-01-02 13:50:12)

Ex. Tipuri de date Oracle SQL

1. Siruri de caractere

- **CHAR(n), VARCHAR2(n) , NCHAR(n), NVARCHAR2(n)** – max. 2000 de caractere (conversie la codarea locală - ex. ASCII pt. Windows)
- **RAW, LONG RAW** – siruri binare (32KB, 2GB – fără conversie)

2. Numere

- **BINARY_INTEGER** (4B), **BINARY_FLOAT** (4B), **BINARY_DOUBLE** (8B) - suportă valorile infinit și NaN
- **NUMERIC(nr. cifre[, zecimale])**, nr. cifre<=38

3. Dată calendaristică și timp

- **DATE**: între 1 Ian 4712 BC și 31 Dec 9999 AD
- **TIMESTAMP**: include факțiuni de secundă - un număr între 0 și 9 cifre (implicit este 6)

4. Obiecte de mari dimensiuni – large objects (LOB)

- **BLOB, CLOB, NCLOB**: pentru binar/car/Unicode până la 128TB

Ex. Tipuri de date MySQL

1. Siruri de caractere

- **CHAR(n)** max 255 caractere, **VARCHAR2(n)** max 255 caractere
- **TEXT** – 64KB caractere, **LONGTEXT** – 4GB caractere
- **ENUM(x,y,z, ...)**, **SET(x,y,z, ...)** liste și multimi de indentificatori

2. Numere

- **TINYINT** (1B), **SMALLINT** (2B), **INT(n)** (4B), **BIGINT(n)** (8B),
FLOAT(nr.cifre, zecimale) (4B), **DOUBLE(nr.cifre,zecimale)** (8B)
- **DECIMAL(nr.cifre, zecimale)** numere în virgulă fixă

3. Dată calendaristică și timp

- **DATE**: dată în intervalul '1000-01-01' și '9999-12-31'
- **DATETIME**: de la '1000-01-01 00:00:00' la '9999-12-31 23:59:59'
- **TIME**: de la '-838:59:59' la '838:59:59', **YEAR**: de la 1901 la 2155
- **TIMESTAMP**: de la '1970-01-01 00:00:01' la '2038-01-09 03:14:07' cu inițializare automată

4. Obiecte de mari dimensiuni – large objects (LOB)

- **BLOB**, **LONGBLOB**: pentru 64KB / 4GB

SQL DDL– Crearea unei baze de date

1. Instrucțiunea **CREATE DATABASE**
2. Depinde de SGBD-ul considerat
3. Sisteme mici – operație simplă. Ex. MySQL:

```
CREATE {DATABASE | SCHEMA}
[IF NOT EXISTS] numeBD
[DEFAULT] CHARACTER SET=charset_name ]
[DEFAULT] COLLATE=collation_name ]
```

- **IF NOT EXISTS** – nu dă eroare dacă deja există
 - **CHARACTER SET** – set caractere - diacritice (ex. latin1, utf8, cp1250...)
 - **COLLATE** – specifică cum sunt comparate caracterele (UNICODE sau fără diacritice)
-
- Sisteme mari = operație complexă
 - Presupune configurarea spațiului de stocare (ex. Oracle ASM - automatic storage management etc.)

Oracle – Crearea unei baze de date

1. Proces complex, o BD poate găzdui mai multe scheme de date

http://docs.oracle.com/cd/B28359_01/server.111/b28310/creat e003.htm#ADMIN11073

2. Pași necesari

- Pas 1: Specificarea unui identificator de instanță (SID)
- Pas 2: Setarea variabilelor de mediu
- Pas 3: Alegerea unei metode de autentificare administrator
- Pas 4: Crearea unui fișier cu parametrii pentru inițializare
- Pas 5: (pt. Windows) Crearea unei instanțe
- Pas 6: Conectarea unei instanțe
- Pas 7: Crearea unui fișier cu parametrii pentru configurare server
- Pas 8: Pornire instanță
- Pas 9: Rulare comandă **CREATE DATABASE**
- Pas 10: Crearea de Tablespaces

Ex. Oracle – CREATE DATABASE

1. Folosind Oracle Managed Files:

CREATE DATABASE numeBD

USER SYS IDENTIFIED BY sys_password

USER SYSTEM IDENTIFIED BY system_password

EXTENT MANAGEMENT LOCAL

DEFAULT TEMPORARY TABLESPACE temp

UNDO TABLESPACE undotbs1

DEFAULT TABLESPACE users;

2. O BD Oracle poate conține mai multe SCHEME – echivalent cu BD MySQL

SQL DDL – Crearea unei tabele

1. Utilizată pentru a crea structura de tabele a unei BD
2. Sintaxă

```
CREATE TABLE nume_tabelă (
    coloană1 tip(dimensiune) constrângerii,
    coloană2 tip(dimensiune) constrângerii,
    .... );
```

3. Constrângerii

- NOT NULL – nu permite valori NULL în acea coloană
- UNIQUE – coloana poate conține doar valori unice
- PRIMARY KEY - combină NOT NULL și UNIQUE
- FOREIGN KEY – pentru integritatea referențială
- CHECK – verifică o condiție logică generică
- DEFAULT – o valoare implicită pentru acea coloană

Ex. Oracle – CREATE TABLE

```
CREATE TABLE nume_tabelă [(  
    coloană1 tip(dimensiune) [NULL | NOT NULL],  
    coloană2 tip(dimensiune) [NULL | NOT NULL],  
    ....  
    CONSTRAINT nume_c PRIMARY KEY (coloană1, ... coloană_n)  
    CONSTRAINT nume_c  
        FOREIGN KEY (coloană1, ... coloană_n)  
        REFERENCES tabelă părinte (coloană1, ... coloană_n)  
        [ON DELETE CASCADE | ON DELETE SET NULL]  
    CONSTRAINT nume_c UNIQUE (coloană1, ... coloană_n)  
    CONSTRAINT nume_c CHECK (expresie logică) ); ]  
    | [AS (SELECT ...)]
```

- Constrângeri suplimentare
 - O cheie primară poate conține cel mult 32 de coloane
 - Unele dintre câmpurile care fac parte dintr-o constrângere unică pot conține valori NULL atât timp cât combinația este unică

SQL DDL – ALTER TABLE (I)

1. Utilizată pentru modificare tabele
2. Permite redenumire, adăugare/ștergere/modificare câmpuri

ALTER TABLE nume_tabela_vechi
 RENAME TO nume_tabela_nou;

ALTER TABLE tabela
 ADD nume_colana tip [constrângeri coloană];

ALTER TABLE tabela
 DROP COLUMN nume_colana; sau **RENAME COLUMN** nume_nou;

ALTER TABLE tabela
 MODIFY nume_colana tip [constrângeri coloană];

3. Constrângeri coloană: **NOT NULL**, **UNIQUE**, **DEFAULT** etc.

SQL DDL – ALTER TABLE (II)

1. Importantă pentru gestionarea constrângerilor

ALTER TABLE tabela

ADD CONSTRAINT nume_constrangere { CHECK,
PRIMARY KEY sau FOREIGN KEY }

ALTER TABLE tabela

DROP CONSTRAINT nume_constrangere;

ALTER TABLE tabela

{ENABLE | DISABLE} CONSTRAINT nume_constrangere;

2. Ex.

ALTER TABLE marinari

ADD CONSTRAINT c_varsta_limita CHECK varsta<70;

SQL DDL – Ștergerea obiectelor

1. Ștergerea unei BD

MySQL: `DROP DATABASE [IF EXISTS] numeBD`

Oracle: `DROP DATABASE numeBD [INCLUDING BACKUPS]
[NOPROMPT] (via RECOVERY MANAGER)`

2. Ștergere tabele

MySQL: `DROP [TEMPORARY] TABLE [IF EXISTS] nume_tabelă`

Oracle: `DROP TABLE nume_tabelă [CASCADE CONSTRAINTS]
[PURGE]`

- PURGE nu permite ROLLBACK

3. Ștergere indecsi

MySQL: `DROP INDEX nume ON tabelă`

Oracle: `DROP INDEX nume`

SQL DML – Interogări active

1. Permite modificarea datelor din tabele unei BD:

- Adăugarea de înregistrări noi folosind **INSERT**
- Modificare/corecție unor date din anumite coloane folosind **UPDATE**
- Ștergerea înregistrărilor care nu mai sunt necesare cu **DELETE**

BD folosită în exemple (BD Port)

- **Tabelă Marinar**

mid	nume	rang	varsta
22	Ion	7	45
31	Horatiu	1	33
58	Oana	8	54
71	Constantin	9	55

- **Tabelă Barcă**

bid	nume	culoare
101	Cleo	Albastra
102	Triton	Rosie
103	Poseidon	Verde

- **Tabelă Rezervare**

rid	Mid	Bid	dată
22	31	101	10/03/2022
231	58	103	18/10/2022
71	31	101	22/10/2022

SQL DML - INSERT

1. Adăugare înregistrări:

INSERT INTO tabelă(listă câmpuri)
 VALUES (listă valori);

INSERT INTO tabelă
 VALUES (listă valori 1:1);

INSERT INTO tabelă
 SET câmp1=exp1, ... ;

Obs: - Poziție aleatoare după inserare – nici o garanție că va fi la sfârșitul tabelei
 - valorile lipsă = **NULL** (eroare dacă a fost specificată o constrângere *not null*)

2. Ex: **INSERT INTO** Marinari **VALUES** (58, "Dinu", 8, 54);

SQL DML – INSERT + SELECT

1. **INSERT** se poate combina cu **SELECT** pentru a copia datele dintr-o tabelă în alta:

INSERT INTO tabelă2

SELECT * FROM tabelă1 **WHERE** condiție;

INSERT INTO tabelă2 (coloană1, coloană1, ...)

SELECT coloană1, coloană1, ...

FROM tabelă1 **WHERE** condiție;

Obs: - câmpurile din cele două tabele trebuie să corespundă ca și tip

2. Ex: **INSERT INTO** Pensionari(nume, varsta)

SELECT nume, varsta

FROM Marinari **WHERE** varsta >=65;

SQL DML - UPDATE

1. Modificare/ corecție date:

UPDATE tabelă

SET coloană1=exp1, ...

WHERE condiție logică;

2. Ex:

UPDATE Marinari

SET rang = rang+1

WHERE rang<9 **AND** varsta>40;

3. Obs: pentru revenire: **SET** rang = rang – 1?

Nu toate expresiile sunt REVERSIBILE !!!

SQL DML - DELETE

1. Ștergere înregistrări:

DELETE FROM tabelă

WHERE condiție logică;

2. Ex:

DELETE FROM Marinari

WHERE varsta>65;

!!! DELETE FROM Marinari;

3. Obs: nu există UNDO. Singura posibilitate este utilizarea unei tranzacții și **ROLLBACK!**

Ex. MySQL - DELETE

```
DELETE [LOW_PRIORITY] [IGNORE]
      FROM tabelă
      [WHERE condiție] [ORDER BY ...]
      [LIMIT nr.rânduri];
```

- Parametrii
 - ORDER BY permite efectuarea într-o ordine prestativă
 - LIMIT permite limitarea la un anumit număr de înregistrări
 - Pot fi folosite împreună
 - Ex. DELETE FROM log WHERE user = 'Joe'
 ORDER BY entry_time LIMIT 1;
 // șterge doar cea mai veche înregistrare din tabela **log**

Baze de Date

Cap. 3. SQL. Proiecție. Selectie. Join



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, Cap. 5

2023 UPT

Conf.Dr. Dan Pescaru

Operatorii algebrei relaționale

1. Operatorii algebrei relaționale

- Proiecție (Π) – selectează doar atributele specificate în lista de proiecție
- Selectie (σ) – selectează rândurile care satisfac condiția de selecție
- Produs Cartezian (\times) – combină două relații R_1 și R_2 incluzând toate perechile ($t_1 \in R_1$ și $t_2 \in R_2$)
- Join (\bullet) – \times cu selecție perechi $\sigma_{\text{cond_join}}(R_1 \times R_2)$
- Reuniune (\cup) – toate înregistrările din R_1 și din R_2 dar fără duplicate
- Intersecție (\cap) – înregistrările comune din R_1 și R_2
- Diferență (/) – înregistrările din R_1 care nu sunt și în R_2

BD folosită în exemple (Port)

Tabelă Marinar

mid	nume	rang	varsta
22	Ion	7	45
31	Horatiu	1	33
58	Oana	8	54
71	Constantin	9	55

Tabelă Barca

bid	nume	culoare
101	Cleo	Albastra
102	Triton	Rosie
103	Poseidon	Verde

Tabelă Rezervare

rid	Mid	Bid	dată
22	31	101	10/03/2022
231	58	103	18/10/2022
71	31	101	22/10/2022

SQL – Proiecție

1. Selectează doar attributele (coloanele) specificate în lista de proiecție:

```
SELECT [DISTINCT] listă_proiecție  
      FROM Tabela;
```

- listă_proiecție: listă de attribute separată prin virgulă sau markerul '*' pentru proiecție 1:1
- **DISTINCT**: implicit nu șterge duplicate!
 - Când este necesar?

2. Ex.: **SELECT** mid, nume **FROM** Marinar;
SELECT DISTINCT rang **FROM** Marinar;

Proiecție – redenumire câmpuri

1. Aliasurile sunt utile pentru a evita ambiguitatea:

- pentru câmpuri calculate
 - poate conține apeluri de funcții de bibliotecă (funcțiile sunt dependente de sistem) + funcții de agregare
 - când listă_proiecție include câmpuri cu același nume din tabele diferite (ex. la Join)

```
SELECT [DISTINCT] câmp1 AS alias1, câmp2 AS  
alias2, ..., exp1 AS aliase1, ...  
FROM Tabela;
```

- E.g.: `SELECT bid AS ID_barca,`
`year(data) AS Anul_rezervarii`
`FROM Rezervare; // -- syntaxă MySQL`

Oracle SQL – Proiecție

1. Sintaxă dialect Oracle SQL:

SELECT [ALL | DISTINCT | UNIQUE]

câmp₁ **AS** alias₁, ..., exp₁ **AS** nume_exp₁, ...
FROM Tabela;

- Se poate folosi **ROWNUM** pentru numerotare rânduri în rezultat

2. Ex.:

SELECT ROWNUM AS nrcrt, bid, nume **AS denumire**
FROM Barca;

MySQL SQL – Proiecție

1. Sintaxă MySQL:

SELECT [ALL | DISTINCT | DISTINCTROW]

câmp₁ **AS** alias₁, ..., exp₁ **AS** nume_exp₁, ...
FROM Tabela;

- Pentru numerotare înregistrări:

SET @r=0;

SELECT @r:="@r+1 AS nrcrt, bid, nume
FROM Barca;

SQL – Selectie

1. Selectează un subset de înregistrări bazată pe o condiție:

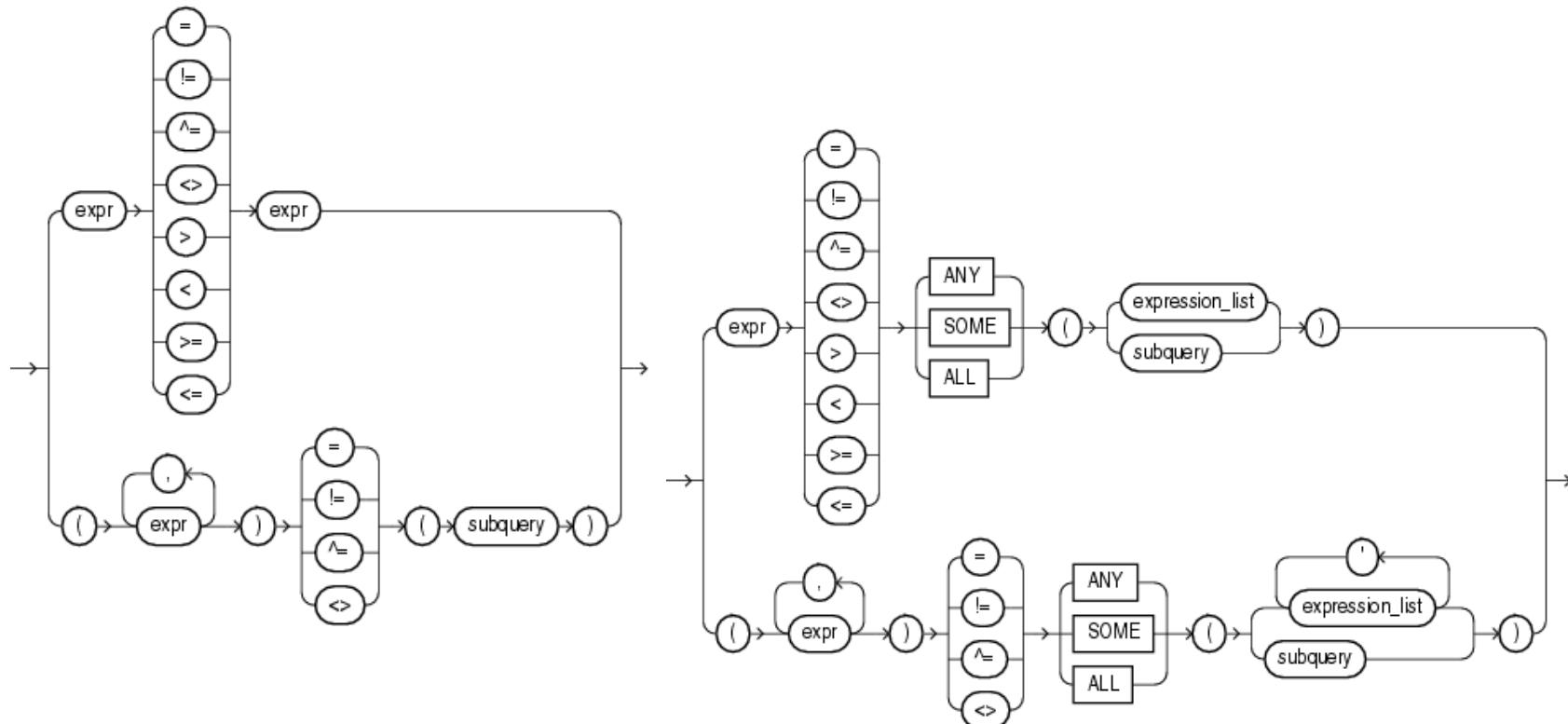
SELECT listă_proiecție **FROM** tabla
WHERE condiție;

- Sau * pentru selecție pură (fără proiecție)
- Pot exista duplicate?
- Condiție selecție: expresii logice cuprinzând operatori **AND**, **OR**, **NOT** și funcții SQL

Ex.: **SELECT** mid, nume **FROM** Marinar
WHERE varsta>30;

Oracle SQL – Selectie

1. Operatori de comparare



Oracle SQL – condiții de selecție

1. Ex.

`SELECT * FROM Marinar`

- `WHERE rang=7;`
- `WHERE rang<3 AND varsta>25 OR vasta>=50;`
- `WHERE rang IN (1,2,7);`
- `WHERE rang = ANY (2,5,6);`
- `WHERE rang NOT IN (1,3);`
- `WHERE varsta > ANY (18, 13, 42, 27);`
- `WHERE varsta > ALL (18, 13, 42, 27);`
- `WHERE nume LIKE 'M_ _ _ %'`
- `WHERE varsta IS NULL`

Oracle SQL – funcții (I)

1. Funcții de bibliotecă

- **abs(*n*)** – valoarea absolută (fără semn)
- **round(*n*)** – rotunjește număr
- **trunc(*n*)** – trunchiază număr
- **concat(*sir1,sir2*)** – concatenează șiruri (sau ‘||’)
- **lower/upper(*sir*)** – modifică *sir* în litere mici/mari
- **substr(*sir, pos, [l]ng*)** – extrage un subsir dintr-un *sir* de la poziția *pos* eventual de lungime *lng*
- **instr(*subsir, sir[, startpos, aparN]*)** – caută *subsir*
- **trim(*sir*)** – șterge spațiile din față/spate
- **length(*sir*)** – returnează lungimea șirului

Oracle SQL – funcții (II)

1. Funcții de bibliotecă

- `sysdate, current_timestamp` – data curentă

Ex. `SELECT sysdate FROM Dual;`

- `extract(spec, d)` – extrage o parte din dată (spec ∈{year, month, day, hour, minute, second})
- `to_char(data, format)` – convertește la sir (format include Y, M, D, HH24, Mi, SS)
- `nvl(expr1,expr2)` returnează *exp2* dacă *exp1* este NULL

Ex. `SELECT nvl(nume, 'necunoscut') FROM Marinar;`

- `case when exp1 then exp2 else exp3 end`

MySQL SQL – condiții selecție

1. Ex.

`SELECT * FROM Marinar`

- { - `WHERE rang=7;`
- `WHERE rang=7 AND varsta>=25;`
- `WHERE rang IN (1,2,7);`
- `WHERE rang BETWEEN 3 AND 7;`
- `WHERE varsta IS NULL;`
- `WHERE varsta<=>NULL; //egalitate NULL-safe`

`SELECT 1 <=> 1, NULL<=>NULL, 1 <=> NULL;`

`SELECT 1 = 1, NULL = NULL, 1 = NULL;`

MySQL SQL – funcții (I)

1. Funcții de bibliotecă:

- **abs(*n*)** – valoarea absolută
- **concat(*str1,str2,...*)** – concatenare șiruri
- **curtime()** – data curentă pe server
- **day()** – ziua din lună (0-31)
- **dayofyear()** – ziua din an (1-366)
- **format(*n, decPlaces*)** – formatare grupe
- **if(*exp1,exp2,exp3*)** – ca și C: *exp1?exp2:exp3*
- **ifnull(*exp1,exp2*)** – dacă *exp1=NULL* atunci *exp2*
- **left(*str, len*)** – subșirul din stânga
- **length(*str*)** – lungimea șirului

MySQL SQL – funcții (II)

1. Funcții de bibliotecă:

- **locate(*subsir, sir, [s_pos]*)** – poziția primei apariției subșirului în sir
- **lower(*sir*)** - transformă în litere mici
- **mid(*sir, pos, len*)** – un substring începând cu poziția specificată
- **month(*d*)** – luna din data specificată
- **now()** – data curentă, include timpul
- **password(*sir*)** – criptare ireversibilă a sir
- **rand()** – număr aleator $\in [0.0\dots1.0]$
- **trim(*sir*)** - șterge spațiile din față/spate din sir

SQL Exemple selecție

- Extragăți rangul tuturor marinilor care au numele în prima jumătate a alfabetului. Clasificați marinarii în juniori și seniori dacă vârsta este sub/peste 31 de ani
 - Soluție Oracle SQL:

```
SELECT ('Marinarul ' ||  
        case when varsta<31 then 'junior' else  
              'senior ' end || nume || ' are rangul ' ||  
        to_char(rang) || '.') AS marinar,  
        nvl(varsta, 0) AS varsta  
FROM Marinar  
WHERE nume < 'L';
```

SQL – Ordonarea rezultatului

1. ORDER BY este utilizat pentru ordonarea rezultatului după unul sau mai multe attribute/expresii
2. Ordinea ascendentă (ASC) este implicită
 - SELECT *coloană1, coloană2, ...*
 - FROM *Tabelă*
 - ORDER BY *coloană_i, coloană_j* [ASC|DESC];
3. Ex.: SELECT * FROM Marinar
 ORDER BY rang, varsta DESC;

SQL – Produs cartezian

1. Nu foarte întâlnit în aplicații

2. Sintaxa:

- Tabele multiple în **FROM** (utile: aliasuri pentru tabele)

SELECT t₁. *, t₂. *

FROM *Tabela₁* t₁, *Tabela₂* t₂;

3. Ex.

SELECT m. *, r.*

FROM Marinar m, Rezervare r;

SQL – JOIN

1. SQL JOIN este folosit pentru a combina rânduri din două sau mai multe tabele pe baza legăturilor logice. De obicei: $t_1.PK=t_2.FK$
2. Două sintaxe posibile:
 - JOIN implicit: tabele multiple în FROM; condiția de join înglobată în clauza WHERE
 $\text{SELECT } t_1.col_1, \dots, t_2.col_1, \dots$
 $\text{FROM } tabela_1 t_1, tabela_2 t_2 \dots$
 $\text{WHERE } t_1.col_i = t_2.col_j [\text{AND cond_selectie}];$
 - JOIN explicit: tabelele și condițiile ambele în FROM

Tipuri de JOIN

1. **INNER JOIN** - returnează toate rândurile cu corespondent în ambele tabele. Sintaxa implicită implică INNER JOIN
2. **LEFT (OUTER) JOIN** - returnează toate rândurile din tabelul din stânga și cele cu corespondent din tabelul din dreapta (cu NULL în câmpurile fără corespondent)
3. **RIGHT (OUTER) JOIN** – invers față de LEFT
4. **FULL (OUTER) JOIN** - returnează toate rândurile din ambele tabele indiferent dacă există sau nu potrivire ($\text{LEFT} \cup \text{RIGHT}$)

INNER JOIN

1. INNER JOIN – sintaxa explicită

```
SELECT list_proiectie  
      FROM tabela1 [INNER] JOIN tabela2  
            ON tabela1.PK=tabela2.FK ...  
            [WHERE conditie_selectie...];
```

2. Ex.

```
SELECT m.nume, m.rang, r.data, b.nume AS barca  
      FROM ((Marinar m INNER JOIN Rezervare r ON  
            m.mid=r.mid) INNER JOIN Barca b ON r.bid=b.bid)  
WHERE m.varsta>25;
```

LEFT JOIN

1. LEFT [OUTER] JOIN – sintaxă

```
SELECT list_proiectie  
      FROM tabela1 LEFT JOIN tabela2  
            ON tabela1.PK=tabela2.FK ...  
      [WHERE conditie_selectie ...];
```

2. Ex.

```
SELECT m.nume, m.rang, r.data, b.nume AS barca  
      FROM ((Marinar m LEFT JOIN Rezervare r ON  
            m.mid=r.mid) LEFT JOIN Barca b ON r.bid=b.bid)  
      WHERE m.varsta>25;
```

RIGHT JOIN

1. RIGHT [OUTER] JOIN – sintaxă

```
SELECT list_proiectie  
      FROM tabela1 RIGHT JOIN tabela2  
            ON tabela1.PK=tabela2.FK ...  
      [WHERE conditie_selectie ...];
```

2. Ex.

```
SELECT m.nume, m.rang, r.data, b.nume AS barca  
      FROM ((Marinar m LEFT JOIN Rezervare r ON  
            m.mid=r.mid) RIGHT JOIN Barca b ON r.bid=b.bid)  
      // WHERE s.age>25; //contradicție – de ce ?
```

FULL JOIN

1. FULL [OUTER] JOIN – sintaxă

```
SELECT list_proiectie  
      FROM tabela1 FULL JOIN tabela2  
            ON tabela1.PK=tabela2.FK ...  
            [WHERE conditie_selectie ...];
```

2. Obs: nu există în toate DBMS (ex. MySQL)

3. Ex.

```
SELECT m.nume, m.rang, r.data, b.nume AS barca  
      FROM ((Marinar m FULL JOIN Rezervare r ON  
            m.mid=r.mid) FULL JOIN Barca b ON r.bid=b.bid)  
            // WHERE s.age>25; // la fel ?
```

SELF JOIN

1. O singură tabelă implicată (legături recursive). Poate fi de tip INNER, LEFT, RIGHT or FULL JOIN, dar ...
2. Ex.

Persoana

id	nume	id_tata	id_mama
B012	John	B026	NULL
B026	Horace	NULL	G018
G114	Anna	B026	NULL
G018	Sara	NULL	NULL

```
SELECT p.nume, t.nume AS tata, m.nume AS mama  
FROM ((Persoana p LEFT JOIN Persoana t  
ON p.id_tata=t.id) LEFT JOIN Persoana m  
ON p.id_mama=m.id);
```

// Obs: dar INNER+INNER JOIN sau LEFT+RIGHT JOIN ?

Oracle SQL JOIN

1. Implementează **FULL OUTER JOIN**
2. Definește un operator pentru outer join (+) care poate fi folosit în condiția de **WHERE** pentru a specifica **LEFT/RIGHT JOIN** în loc de **INNER JOIN**. Nu poate înlocui **FULL JOIN**
3. Ex. Pentru **LEFT JOIN**

```
SELECT m.nume, m.rang, r.data  
      FROM Marinar m, Rezervare r  
     WHERE m.mid = r.mid(+);
```

Obs: pentru **RIGHT JOIN**: $m.mid(+) = r.mid$

MySQL JOIN

1. Nu implementează **FULL OUTER JOIN**. Poate fi obținut cu **UNION** între **LEFT** și **RIGHT JOIN**
2. **USING** poate înlocui **ON** dacă PK și FK au același nume
 - Ex. **SELECT m.* , r.***
FROM Marinar m INNER JOIN Rezervare r
USING (mid);
3. **STRAIGHT_JOIN** este similar cu **JOIN**, dar tabela din stânga ca fi citită înaintea celei din dreapta. Acest lucru poate corecta ordinea pentru optimizarea execuției

Baze de Date

Cap. 4. SQL. Subinterrogări. Operații pe multimi



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C. 15

2023 UPT

Conf.Dr. Dan Pescaru

SQL Subinterrogari

1. Subquery este o interogare în interiorul altei interogări. Alternativ: nested query
2. Subinterrogările permit scrierea de interogări care selectează înregistrări bazat pe criterii dezvoltate la execuție (în timpul rulării interogării)

Ex.

```
SELECT listă_proiecție FROM tabelă  
WHERE cond_cu_subinterrogare
```

Categorii (I)

1. Există trei tipuri de subinterrogări funcție de rezultatul calculat:
 - A. Subinterrogări care întorc un set (mulțime) de valori. Fiecare element din mulțime trebuie să fie o singură valoare scalară. Se pot folosi împreună cu operatorul **IN** sau în comparații utilizând modificatorii **ANY** sau **ALL**

Categorii (II)

- B. Subinterrogări care întorc o singură valoare scalară. Utilizabile în comparații simple, fără modifieri
- C. Subinterrogări care întorc un set (multime) de înregistrări. Utilizabilă cu ajutorul operatorului **EXISTS** pentru testare dacă multime este vidă

Subinterrogări corelate și necorelate

- 1. Subinterrogări necorelate – sunt independente de interogarea principală. Poate fi rezolvată separat, înainte de evaluarea interogării principale**
- 2. Subinterrogări corelate – depind de valori transmise din interogarea principală și ca atare nu pot fi evaluate independent**

Subinterrogări – reguli generale

1. O subinterrogare **SELECT** este similară cu o interrogare singulară
2. Subinterrogările nu pot manipula intern rezultatul. Ca atare o subinterrogare nu va include clauza **ORDER BY** pentru ordonare rezultat
3. Rezultatul exprimate prin lista de proiecție a subinterrogării trebuie să corespundă ca și fel/tip scopului utilizării (conform celor prezentate la expunerea categoriilor după rezultat)

BD folosită în exemple (Port)

Tabelă Marinar

mid	nume	rang	varsta
22	Ion	7	45
31	Horatiu	1	33
58	Oana	8	54
71	Constantin	9	55

Tabelă Barca

bid	nume	culoare
101	Cleo	Albastra
102	Triton	Rosie
103	Poseidon	Verde

Tabelă Rezervare

rid	Mid	Bid	dată
22	31	101	10/03/2022
231	58	103	18/10/2022
71	31	101	22/10/2022

Subinterrogări – operatorul IN

1. Subinterrogările folosite împreună cu operatorul **IN** au forma generală:

WHERE expresie [**NOT**] **IN** (subinterrogare)

2. Ex.: toți marinarii care nu au rezervat barca 103

SELECT m.mid, m.nume **FROM** Marinar m

WHERE m.mid **NOT IN** (**SELECT** r.mid

FROM Rezervare r

WHERE r.bid = 103)

3. Evaluarea neoptimizată este echivalentă cu două cicluri suprapuse: pentru fiecare înregistrare din Marinar se va testa condiția cu subinterrogare

Despre operatorul IN (I)

1. Pentru o mai bună înțelegere urmărim evaluarea optimizată pas cu pas
2. Nefiind corelată, subinterrogarea se poate evalua independent de interrogarea principală

```
SELECT r.mid  
      FROM Rezervare r  
     WHERE r.bid=103
```

3. Rezultatul este multimea { 58 }

Despre operatorul IN (II)

1. În continuare se va substitui rezultatul subinterrogării ca și operand pentru operatorul IN și se va evalua apoi interogare principală

```
SELECT m.mid, m.nume FROM Marinar m  
WHERE m.mid NOT IN { 58 }
```

2. Rezultatul va fi { Ion, Horatiu, Constantin }

Implementare INTERSECT folosind IN

1. Să se listeze toți marinarii care au rezervat și bărci albastre și bărci verzi.

```
SELECT DISTINCT m.mid, m.nume  
FROM Marinar m, Barca b, Rezervare r  
WHERE m.mid=r.mid AND r.bid=b.bid AND  
b.culoare= 'albastra' AND  
m.mid IN  
(SELECT m1.mid  
FROM Marinar m1, Barca b1, Rezervare r1  
WHERE m1.mid=r1.mid AND r1.bid=b1.bid  
AND b1.culoare= 'verde')
```

Subinterrogări – operatori de comparare

1. Subinterrogarea trebuie să returneze o singură valoare scalară. De exemplu o agregare fără grupare (discutată în următorul capitol)
2. Aceasta poartă denumirea de subinterrogare scalară deoarece returnează un singur rând și o singură coloană conținând doar o valoare scalară
3. Dacă va returna mai mult de o valoare se va genera o eroare.

Subinterrogări – operatori de comparare

1. Ex. Găsiți toți marinarii care sunt mai în vîrstă decât marinarul care a rezervat barca 103 în data de {10-18-2022}

```
SELECT * FROM Marinar WHERE  
varsta > (SELECT m.varsta  
          FROM Marinar m INNER JOIN  
              Rezervare r ON m.mid=r.mid  
          WHERE r.bid=103 AND  
                r.data='10-18-2022')
```

Subinterrogări – modificatorii ALL și ANY

1. Modificatorii **ALL** și **ANY** pot schimba semantica operatorilor de comparare pentru a permite tratarea valorilor multiple din subinterrogare
2. O formă generală de **WHERE** pentru aceştia este

WHERE <expresie> <operator comparație>
[**ALL** | **ANY**] (subinterrogare)

Utilizare ALL

1. Modificatorul **ALL** schimbă operatorul mai mare (sau mai mic) pentru a însemna mai mare decât (sau mai mic decât) toate valorile returnate

Ex. Găsiți toti marinarii cu rangul cel mai mare

SELECT * FROM Marinar

WHERE

rang >= **ALL (SELECT rang
FROM Marinar)**

Utilizare ANY

1. Mai puțin restrictiv decât **ALL**, **ANY** modifică \geq (sau \leq) pentru a însemna mai mare decât (sau mai mic decât) cel puțin una dintre valori

Ex. Găsiți toti marinarii cu rang mai mare decât un marinar numit Horatiu

```
SELECT m.* FROM Marinar m  
WHERE m.rang > ANY  
(SELECT m1.rang FROM Marinar m1  
WHERE m1.nume='Horatiu')
```

IN, ANY, ALL

1. Echivalențe

- **IN** este echivalent cu **=ANY**
- **NOT IN** nu este echivalent cu
<>ANY !!!
- **NOT IN** este echivalent cu
<>ALL

Operatorul EXISTS

1. Clauza **WHERE** a interogării principale testează existența înregistrărilor returnate de subinterrogare
2. Subinterrogarea nu produce efectiv nici un rezultat; operatorul returnează **TRUE** sau **FALSE**

Ex. Găsiți toți marinarii care au rezervat cel puțin o barcă (=> interogări *corelate* !)

```
SELECT m.* FROM Marinar m WHERE  
EXISTS ( SELECT r.* FROM Rezervare r  
WHERE r.mid=m.mid)
```

DIVISION implementată cu EXISTS

1. Toți marinarii care au rezervat toate bărcile
(subinterrogări corelate multiple)

```
SELECT m.mid, m.nume FROM Marinar m
WHERE NOT EXISTS ( SELECT b.bid
                    FROM Barca b
                    WHERE NOT EXISTS (
                        SELECT r.bid
                            FROM Rezervare r
                            WHERE r.bid = b.bid
                                AND r.mid = m.mid ))
```

Subinterrogări în FROM

1. Subinterrogările pot fi plasate și în clauza FROM. Este utilă la precalcularea agregărilor
2. Ex. Să se formeze perechi dintre bărcile nerezervate încă și marinari

```
SELECT m.* , b.*  
      FROM Marinari m,  
            (SELECT b1.bid, b1.nume  
              FROM Barca b1  
             WHERE NOT EXISTS (  
                   SELECT r.*  
                     FROM Rezervare r  
                    WHERE r.bid=b1.bid)) b;
```

UNION

1. Combină rezultatele a două interogări și elimină duplicatele
2. Relațiile trebuie să fie Union compatibile
3. UNION ALL permite păstrarea dupliilor

```
SELECT m1.*  
      FROM Port1.Marinar m1  
UNION  
SELECT m2.*  
      FROM Port2.Marinar m2;
```

UNION pentru FULL JOIN

1. UNION poate fi folosit pentru implementarea operatorului de full join in sistemele care nu îl oferă (ex. MySQL)

```
SELECT * FROM T1  
    LEFT JOIN T2 ON T1.fk = T2.pk  
UNION  
SELECT * FROM T1  
    RIGHT JOIN T2 ON T1.fk = T2.pk;
```

SQL DIFFERENCE (I)

1. Implementat de operatorul **MINUS** sau **EXCEPT**, dependent de sistem
2. Relațiile trebuie să fie union compatibile
3. Nu este implementat de MySQL (poate fi obținut cu subinterrogări și operatorul **NOT IN**)

SQL DIFFERENCE (II)

1. Ex. Marinarii care au rezervat o barcă roșie dar nu și una verde:

```
SELECT DISTINCT m.mid, m.nume  
FROM Marinar m, Rezervare r, Barca b  
WHERE m.mid=r.mid AND r.bid=b.bid AND  
      b.culoare='rosie'
```

MINUS

```
SELECT DISTINCT m.mid, m.nume  
FROM Marinar m1, Rezervare r1, Barca b1  
WHERE m1.mid=r1.mid AND r1.bid=b1.bid AND  
      b1.culoare= 'verde'
```

Baze de Date

Cap. 5. SQL – Agregare Date



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C15

2023 UPT

Conf.Dr. Dan Pescaru

Funcții de agregare

1. Funcțiile de agregare au ca intrare o mulțime de valori și returnează o singură valoare scalară
2. Sunt folosite pentru a calcula date statistice despre informațiile din tabelă
3. Majoritatea funcțiilor de agregare (toate cu excepția MIN, MAX și COUNT) primesc o singură expresie numerică ca parametru de intrare

Proprietățile funcțiilor de agregare

1. Principalele proprietăți ale funcțiilor de agregare sunt:

- Operează pe o singură coloană sau expresie dintr-un grup de coloane
- Returnează o singură valoare pentru un grup de înregistrări
- Utilizabile doar în **lista de proiecție**, în **subinterrogări** și în clauza **HAVING**. **NU** se pot utiliza în clauza **WHERE !!!**
- Rezultatul este anonim – se va redenumi corespunzător scopului printr-un alias

Limitări la utilizarea funcțiilor de agregare

1. **Lista de proiecție** va conține doar funcții de agregare și valori **distințe** (care includ chei):

- Expresii care generează o singură valoare pentru grupul de înregistrări din intrare (aceste expresii trebuie incluse în clauza **GROUP BY**)

Ex. ~~SELECT nume, COUNT(*) FROM Marinar;~~

- Obs. Nu se verifică de unele versiuni de MySQL, dar nu are semnificație logică!

Controlul setului de intrare

1. Pentru a controla setul de intrare se pot folosi:

- **DISTINCT**: ia în considerare numai valori distincte din setul de intrare
- **ALL** (implicit): ia în considerare toate valorile, inclusiv duplicate

2. Ex.: **SELECT COUNT(DISTINCT col) AS Nr FROM Tabelă**

Funcții de agregare

- A. **COUNT**: întoarce numărul de înregistrări
- B. **SUM**: returnează suma valorilor expresiilor/coloanelor din setul de intrare
- C. **AVG**: returnează media valorilor expresiilor/coloanelor din setul de intrare
- D. **MIN, MAX**: returnează minimul/maximul expresiilor/coloanelor din setul de intrare
- E. **VARIANCE**: măsoară împrăștierea unui set de numere în jurul mediei lor
- F. **STDDEV**: returnează deviația standard a unui set de valori numerice

COUNT()

1. Returnează numărul de valori/valori distincte din setul de intrare
2. Parametrii de intrare
 - [DISTINCT/ALL] coloană/expresie: nu ia în calcul valorile Null și ia/nu ia în calcul duplicate
 - * : ia în calcul toate înregistrările din intrare indiferent dacă sunt Null sau duplicate

```
SELECT COUNT(DISTINCT ClientID) AS NrDeClienti  
      FROM Factura;
```

SUM()

1. Returnează suma valorilor coloanei/expresiei de intrare
 2. Ignoră valorile Null (tratare ca 0). Dacă toate valorile sunt Null returnează Null
- Ex. **SELECT SUM(salar) AS buget
FROM Angajat
WHERE id_departament=101;**

AVG()

1. Returnează media aritmetică a valorilor din coloana/expresia de intrare
2. Ignoră valorile Null (nu sunt tratate ca 0). Dacă toate sunt Null, returnează Null
3. Atenție la utilizarea **DISTINCT** și **ALL** în **AVG**
 - Ex.

```
SELECT SUM(vârsta)/COUNT(*) AS vmSUM,  
          AVG(vârsta)  AS vmAVG  
      FROM Marinar;
```

MIN(), MAX()

1. MIN() - returnează cea mai mică valoare
2. MAX() - returnează cea mai mare valoare
3. Ignoră valorile Null (nu sunt tratate ca 0). Dacă toate sunt Null, întoarce Null
4. La siruri de caractere: comparația este case sensitive / insensitive la Oracle/MySQL
5. Obs: **DISTINCT** nu are nici un efect
 - Ex.

```
SELECT MIN(vârstă) AS vmin, MAX(vârstă) AS vmax  
      FROM Marinar;
```

VARIANCE()

1. Returnează varianța unui set de numere (măsoară cât de departe setul de numere este împrăștiat în jurul valorii medii). Se calculează ca media pătratelor diferențelor valorilor față de valoarea medie

$$\sigma = \text{SUM}(d^2)/\text{COUNT}(d) - (\text{SUM}(d)/\text{COUNT}(d))^2$$

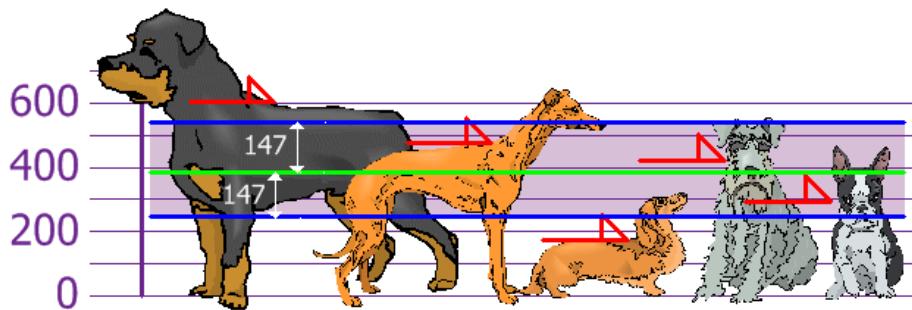
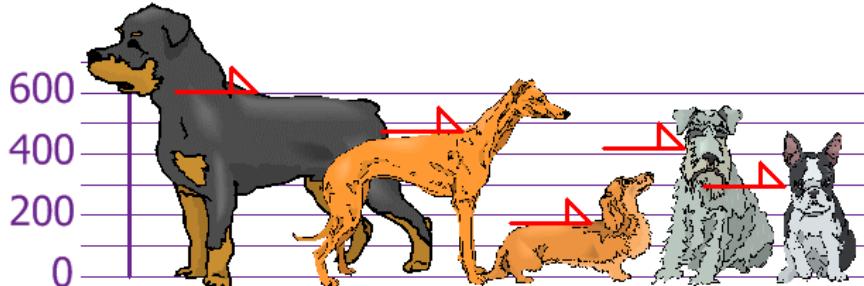
1. Returnează **0** dacă setul conține doar o **înregistrare**
2. Ignoră valorile **Null** (nu sunt tratate ca 0). Dacă toate sunt **Null**, returnează **Null**
3. Obs: **DISTINCT** are efect.

SELECT VARIANCE(vârsta) AS variV FROM Marinar;

STDDEV()

1. Returnează deviația standard a unui set de valori (definește un interval de "Normalitate" în jurul mediei valorilor)
2. Se calculează ca și rădăcină pătrată din Varianță
3. Returnează 0 dacă intrarea conține un singur rând
4. Ignoră valorile Null, DISTINCT are efect
 - Ex.
`SELECT STDDEV(vârsta) AS devStdVarsta
FROM Marinar;`

Deviatia standard. Interpretare



1. Media = 394 mm
2. Varianta = 21704
3. Deviatia standard = 147 mm
4. Înălțime standard ("normală") = [247.. 541] mm
5. Foarte-mari: 600 mm, Foarte-mici: 170 mm

(Ref: <http://www.mathsisfun.com/>)

Gruparea datelor (I)

1. De ce: Calculați count(*) pentru fiecare rang (dar pentru fiecare vârstă?)
2. **GROUP BY**: grupează datele de intrare și produce o singură valoare a funcției de agregare pentru fiecare grup
3. Ex.

```
SELECT idDep AS departament,  
       MAX(salar) AS salMax  
  FROM Angajat  
 GROUP BY idDep;
```

Gruparea datelor (II)

1. Lista de proiecție pot să conțină doar funcții de agregare și eventual expresii cu valori unice pentru întregul grup
2. GROUP BY poate conține mai mult de un singur criteriu (însă nu ierarhic)
3. Lista de proiecție trebuie să conțină toate expresiile GROUP BY care nu sunt agregări
 - Ex. **SELECT** vârsta, rang, **COUNT(*) AS** nrMar
FROM Marinar
GROUP BY rang, vârsta; // sau vârsta, rang?

Gruparea datelor: ordonare grupuri

- ORDER BY poate fi utilizată pentru ordonarea grupurilor (ierarhizare)

```
SELECT rang, vârsta, COUNT(*) AS nrMar  
      FROM Marinar GROUP BY rang, vârsta  
      ORDER BY rang, vârsta;
```

vs.

```
SELECT rang, vârsta, COUNT(*) AS nrMar  
      FROM Marinar GROUP BY rang, vârsta  
      ORDER BY vârsta, rang;
```

Gruparea datelor: adăugare de câmpuri

- Toate expresiile unice care se adaugă în lista de proiecție se vor adăuga și în GROUP BY

~~SELECT d.idDep, **d.nume**, AVG(a.salar) AS salMed
FROM (Angajat a INNER JOIN Departament d
ON a.idDep=d.idDep)
GROUP BY d.idDep;~~

SELECT d.idDep, **d.nume, AVG(a.salar) AS salMed
FROM (Angajat a INNER JOIN Departament d
ON a.idDep=d.idDep)
GROUP BY d.idDep, **d.nume**;**

HAVING (I)

1. Se utilizează cu **GROUP BY** pentru a restricționa grupurile din rezultat
2. Câteodată înlocuibil cu **WHERE**

```
SELECT rang, COUNT(*) AS nrMar  
      FROM Marinar  
      GROUP BY rang HAVING rang>3;
```

```
SELECT rang, COUNT(*) AS nrMar  
      FROM Marinar WHERE rang>3  
      GROUP BY rang;
```

HAVING (II)

1. De obicei mai eficient decât **WHERE**
2. Nu se poate înlocui în orice situație cu **WHERE** (ex. când conține o agregare)

```
SELECT rang, COUNT(*) AS nrMar  
      FROM Marinar  
     GROUP BY rang  
    HAVING COUNT(*)>1;  
  
( WHERE COUNT(*)>1 )
```

Combinarea funcțiilor de agregare

1. Funcțiile de agregare pot fi combinate în expresii aritmetice
2. Ex.

```
SELECT idDep AS departament  
      (MAX(salar) - MIN(salar)) AS  
          intervalSal  
FROM Angajat  
GROUP BY idDep;
```

Funcții de agregare îmbricate

1. Funcțiile de agregare pot fi aplicate imbricat (suportat doar de unele sisteme ex. Oracle)
2. Ex.

```
SELECT idDep, AVG(MAX(salar)) AS medieSalDep  
      FROM Angajat  
      GROUP BY idDep; // rezultă o singură valoare
```

- Se efectuează prima dată aggregarea internă (**MAX(salar)**) pentru fiecare grup generat de **GROUP BY** (fiecare idDep), apoi se aplică cea de a doua agregare peste rezultate

Alternativă la imbricarea agregărilor

1. Pentru MySQL se pot folosi subinterrogări
2. Ex.

```
SELECT AVG(depsal.maxsal)
FROM
( SELECT MAX(salar) AS maxsal
  FROM Angajat
 GROUP BY idDep
) depsal;
```

Agregare și subinterrogări (I)

1. Funcțiile de agregare pot fi folosite în subinterrogări
2. Ex. Toți marinarii care au vârstă mai mare decât media vîrstelor marinilor

```
SELECT *
  FROM Marinar
 WHERE vârsta >
      (SELECT AVG(vârsta) FROM Marinar);
```

Agregare și subinterrogări (II)

1. Ex. Cei mai bătrâni marinari (vârsta lor nu o cunoaștem încă prealabil)

SELECT *

FROM Marinar

WHERE vârsta =

(SELECT MAX(vârsta) FROM Marinar);

Agregare directă utilizând OVER

1. Clauza **OVER** permite definirea unei ferestre. Agregarea se va aplica peste grupurile definite de parametrul **PARTITION BY** din clauza **OVER**. Pentru toata tabela se poate folosi **PARTITION BY 1**
2. Generează un sumar pentru fiecare rând (în loc de fiecare grup definit prin **GROUP BY**)

```
SELECT mid, nume, varsta, rang,  
       AVG(varsta) OVER (PARTITION BY rang)  
             AS varstaMediePerRang  
FROM Marinar  
ORDER BY nume;
```

Ierarhizarea utilizând OVER

1. Funcția **ROW_NUMBER()** împreună cu clauza **OVER** permite ierarhizarea și ordonarea rezultatelor după poziția înregistrării curente în cadrul grupului considerat (ferestrei)

```
SELECT rang, nume, varsta,  
       ROW_NUMBER() OVER (PARTITION BY  
                           rang order by varsta) as nrrang,  
       AVG(varsta) OVER (PARTITION BY  
                           rang) as vmedrang  
FROM Marinar
```

Exemple (I)

1. Problemă:

- Să se listeze toți marinarii împreună cu numărul total de rezervări făcute de fiecare în parte.

Exemple (I)

Răspuns 1:

```
SELECT m.*,
       (SELECT count(*) FROM Rezervare r
        WHERE r.mid=m.mid) AS nrRez
    FROM Marinar m ORDER BY nrRez;
```

Răspuns 2:

```
SELECT m.mid, m.nume, COUNT(*) AS nrRez
      FROM Marinar m INNER JOIN Rezervare r
        ON m.mid=r.mid
 GROUP BY m.mid,m.nume ORDER BY nrRez;
```

Obs: greșit pentru 0 rezervări (De ce **LEFT JOIN** este de asemenea greșit? Solutie (rezolvare NULL)?

Exemple (II)

1. Problemă :

- Găsiți vârsta celui mai Tânăr marinăru peste de 20 de ani pentru fiecare rang care are cel puțin 3 marinari de orice vârstă.

Exemple (II)

1. Răspuns

```
SELECT m1.rang, MIN(m1.varsta) AS tanar  
      FROM Marinar m1  
     WHERE m1.varsta > 20  
   GROUP BY m1.rang  
 HAVING 1< (SELECT COUNT(*)  
           FROM Marinar m2  
          WHERE m1.rang=m2.rang);
```

Obs: De ce nu este corect **HAVING COUNT(*)>1?**

Baze de Date

Cap. 6. Implementarea aplicațiilor în Oracle APEX



2023 UPT

Ref: <https://apex.oracle.com/en/learn/tutorials/>
Conf.Dr. Dan Pescaru

Oracle APEX

1. Oracle Application Express (APEX) este o platformă de dezvoltare "low-code"
2. Creează aplicații Web (tip client) pentru baze de date Oracle
3. Oferă un IDE (mediu integrat de dezvoltare) ce sprijină tehnici RAD
 - accesibil în browser
 - nu necesită instalare
4. Cuprinde: GUI, Wizards, Controle etc.

Istoric

1. Funcționează cu Oracle Database și Oracle Database Express Edition (XE)
2. Versiuni:
 - HTML DB 1.5 – 2004
 - HTML DB 2.0 – 2005
 - APEX 2.1 – 2006, inclusă în Oracle XE
 - APEX 3.1 – 2008, raportări interactive
 - APEX 5.0 – 2015, page designer, UnivTheme
 - APEX 19.1 – 2019, formulare REST
 - APEX 22.2 – 2022, geolocation, datagen

Avantaje

1. Ușor de creat pagini folosind machete și controale incluse
2. Rulează direct pe serverul de APEX pe Web prin URL (nu necesită instalare)
3. Scalabil (laptop, desktop, cloud)
4. Procesare și validări pe server
5. Suport pentru dezvoltarea în echipă
6. Aplicația poate fi instalată la client

Dezavantaje

1. Aplicațiilor lucrează doar cu baze de date Oracle
2. Nu toți furnizării ISP oferă APEX (multi oferă doar MySQL+PHP). La nevoie se poate însă instala Oracle XE + APEX pe o mașină virtuală în orice cloud
3. Nu are control integrat al versiunilor
4. Probleme de incompatibilități între versiuni de APEX și Oracle DB

Utilizzatori APEX

Screenshot of the Oracle APEX Community Success Stories page:

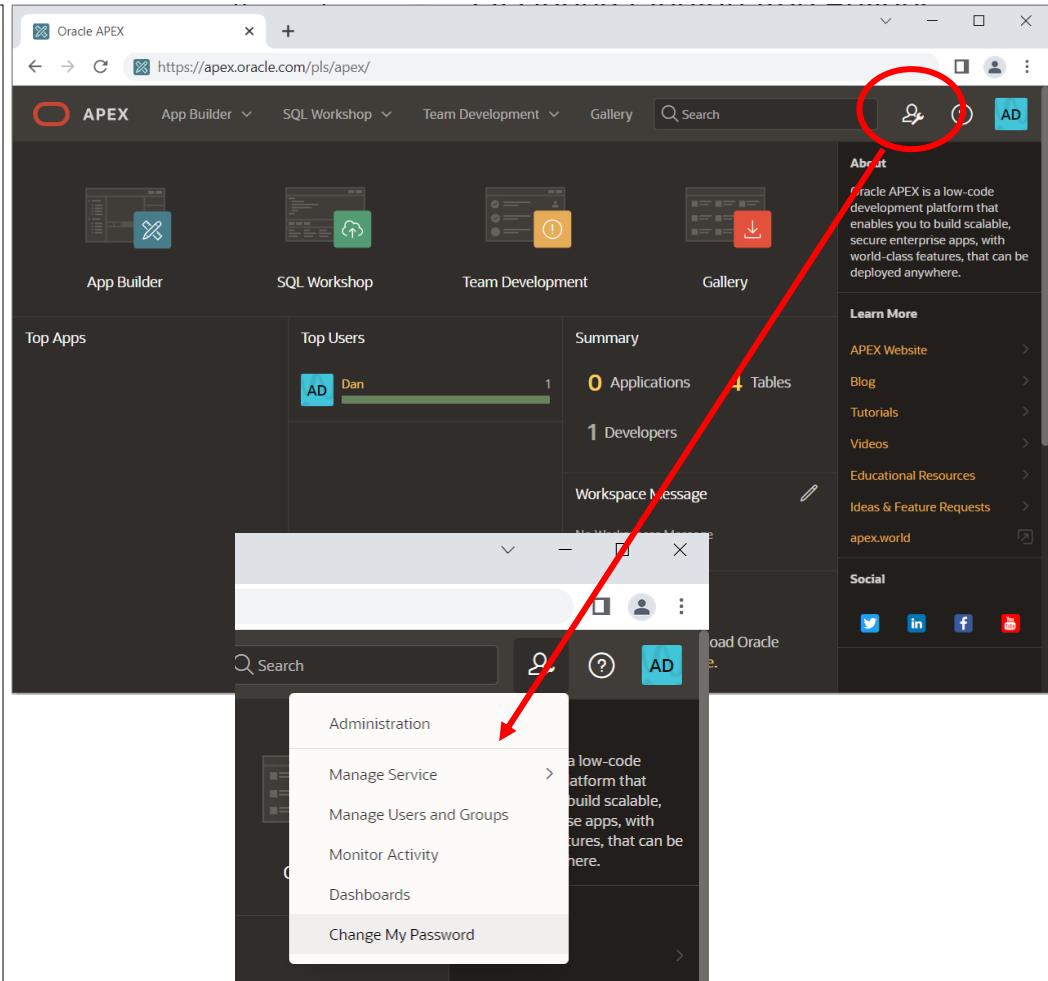
The page displays a grid of success stories from various companies using Oracle APEX. Each story includes the company logo, a brief description, and the industry and date of the success story.

Company	Description	Industry	Date
SIEMENS	Siemens Mobility Tames Project Complexity with Oracle Cloud	Transportation	[April 2022]
Novatech	Novatech modernizes manufacturing processes with Oracle APEX	Engineering and Construction	[April 2022]
SATLOG	SATLOG modernizes fleet management at scale on Oracle Cloud	High Technology	[November 2021]
Norfolk County Council	Norfolk County Council simplifies payroll and lifts staff engagement	Public Sector	[November 2021]
KING HAMAD UNIVERSITY HOSPITAL	King Hamad University Hospital modernizes healthcare	Healthcare	[October 2021]
CARPET COURT	Carpet Court turns data into insights with Oracle Cloud	Retail	[October 2021]
starCRM	Star CRM lifts productivity by 40% and reduces costs by 30%	High Technology	[October 2021]
Samsontite	Samsonite Europe scales luggage-damage service with Oracle	Consumer Goods	[October 2021]
eclix	Eclix Group gains agility and modernizes with Oracle Cloud	Financial Services	[October 2021]
WPS	WPS enhances app development and security with Oracle Cloud	Professional Services	[October 2021]
iAdvantage Software, Inc.	iAdvantage Software gains agility by moving to Oracle Cloud	High Technology	[October 2021]
AgrarMarkt Austria	AgrarMarkt Austria accelerates grant processing with Oracle Database	Public Sector	[October 2021]
SOHAM	Soham ERP builds new app using Oracle Cloud	[October 2021]	
CWT Globelink	CWT Globelink delivers key applications faster with Oracle Cloud	[October 2021]	
Bleckmann	Bleckmann boosts warehouse efficiency with Oracle Cloud and APEX	[October 2021]	

https://apex.oracle.com/pls/apex/r/apex_pm/apex-community/home

Interfață APEX

- **AppBuilder:** crearea și editarea unei aplicații Web
- **SQL Workshop:** rulare SQL, acces la BD
- **Team Development:** gestiune echipă, urmărire probleme, schimb mesaje
- **Gallery:** acces la exemple de baze de date, aplicații și extensii tip plug-in



APEX: AppBuilder

1. Permite crearea și gestionarea unei aplicații Web conectate la BD, precum și editarea paginilor acesteia
2. În plus permite pentru fiecare aplicație
 - Rularea
 - Editarea
 - Importul
 - Exportul
 - Copierea
 - Ștergerea

Crearea unei aplicații noi

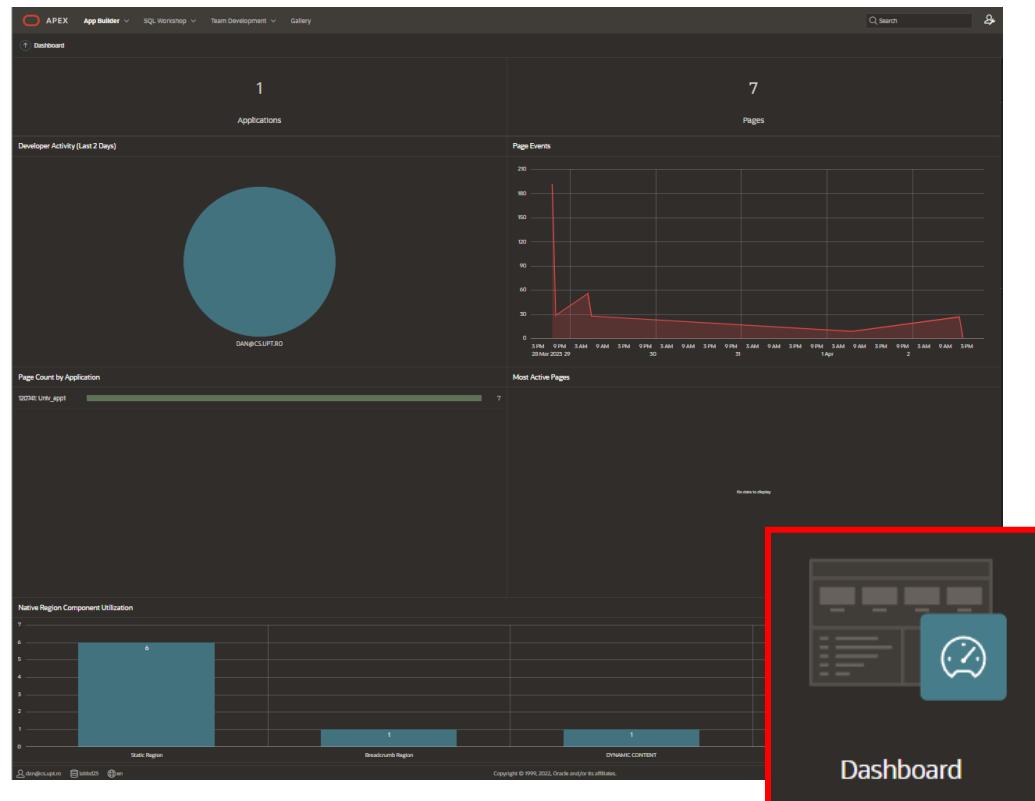
The screenshot shows the Oracle APEX App Builder interface. On the left, a sidebar titled 'Workspace Utilities' has a red circle around the 'Create >' button, with a red arrow pointing to the main application creation screen. The main screen is titled 'Create an Application'. It shows the following details:

- Name:** Univ_app1
- Appearance:** Vita, Side Menu
- Pages:** Home, Blank
- Features:** Install Progressive Web App (unchecked), About Page (checked), Activity Reporting (unchecked), Configuration Options (unchecked), Theme Style Selection (checked), Access Control (unchecked), and Feedback (unchecked).
- Settings:** Application ID: 120741, Schema: WKSP_LABBD23, Authentication: Oracle APEX Accounts, Language: English (en), Advanced Settings, and User Interface Defaults.

At the bottom right are 'Cancel' and 'Create Application' buttons. The URL in the browser is https://apex.oracle.com/pls/apex/.

APEX Dashboard

1. Permite vizualizarea de informații despre o aplicație: numărul de pagini, contribuția autorilor, mai utilizate componente și regiuni, frecvența evenimentelor de acces la pagină etc.



Editare pagina About

The screenshot shows the Oracle APEX App Builder interface for editing a page named 'About'. The top navigation bar includes links for APEX, App Builder, SQL Workshop, Team Development, and Gallery, along with a search bar and user information.

The left sidebar displays the page structure:

- Application 120741 \ Page Designer
- Layout
- Page Search
- Help
- Page 10020: About
- Pre-Rendering
- Components
 - Body
 - </> About Page
- Post-Rendering

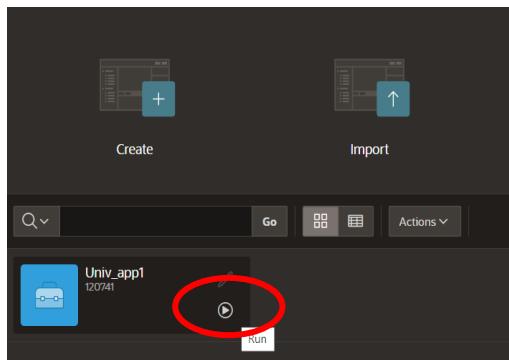
A red circle highlights the 'About Page' component in the Components list. Another red circle highlights the 'About Page' region in the main content area. A third red circle highlights the 'Source' tab in the right-hand Region Attributes panel, which contains the following HTML code:

```
Aplicație client pentru baza de date Universitate.
```

The Region Attributes panel also shows the following settings:

- Title:** About Page
- Type:** Static Content
- Source:** HTML Code (with the highlighted text)
- Layout:**
 - Sequence: 20
 - Parent Region: No Parent
 - Position: Body
 - Start New Row: On
 - Row CSS Classes:
 - Column: Automatic
 - Column Span: Automatic
 - Column CSS Classes:
 - Column Attributes:
- Appearance:**
 - Template: Content Block
 - Use Template Defaults:

Rulare aplicație



Application 120741

Application 120741 - Univ.app1

Edit Application Definition

Run Application

Supporting Objects

Shared Components

Utilities

Export / Import

Create Page >

0 - Global Page

1 - Home

9999 - Login Page

10000 - Administration

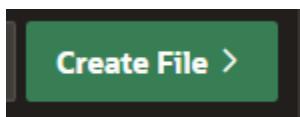
10010 - Application Appearance

10020 - About

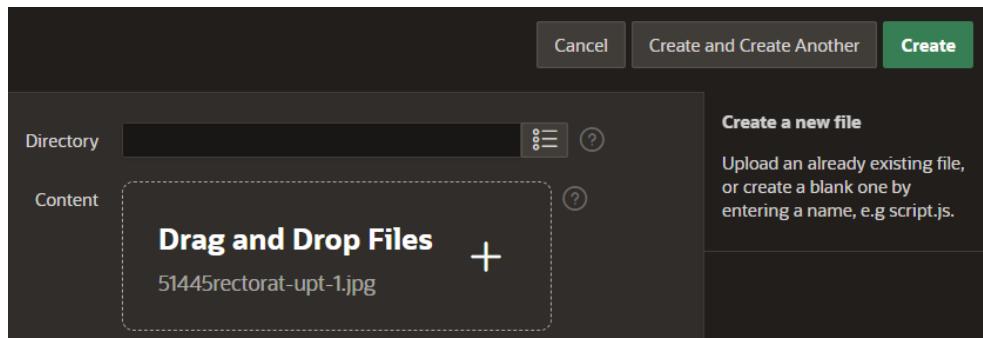
10021 - Help

Încărcare resurse externe (ex. img)

1. Se selectează aplicația din Application Builder și apoi se selectează butonul Shared Components / Static Application Files apoi se apasă Create File.

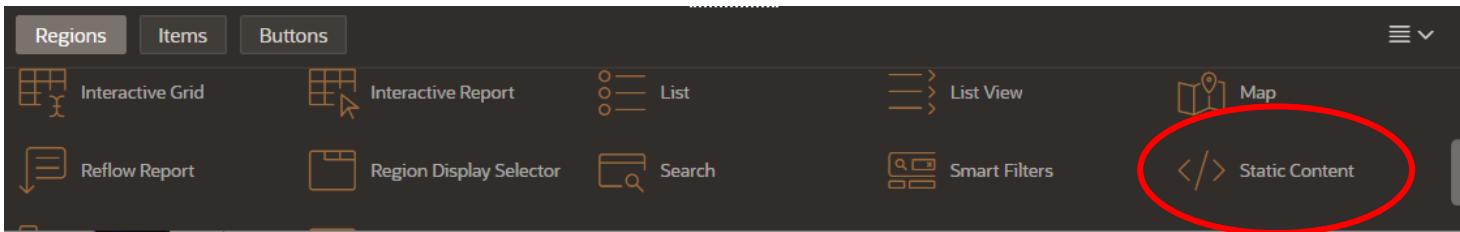


2. Se adaugă prin drag-and-drop fișierul cu imaginea dorită și se apasă butonul Create

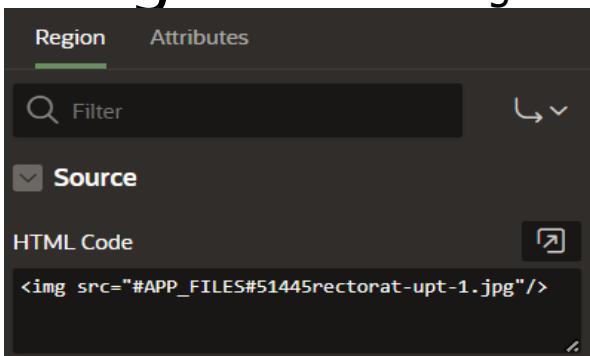


Afișarea unei imagini statice

1. În pagina dorită se adaugă o regiune de tip Static Content

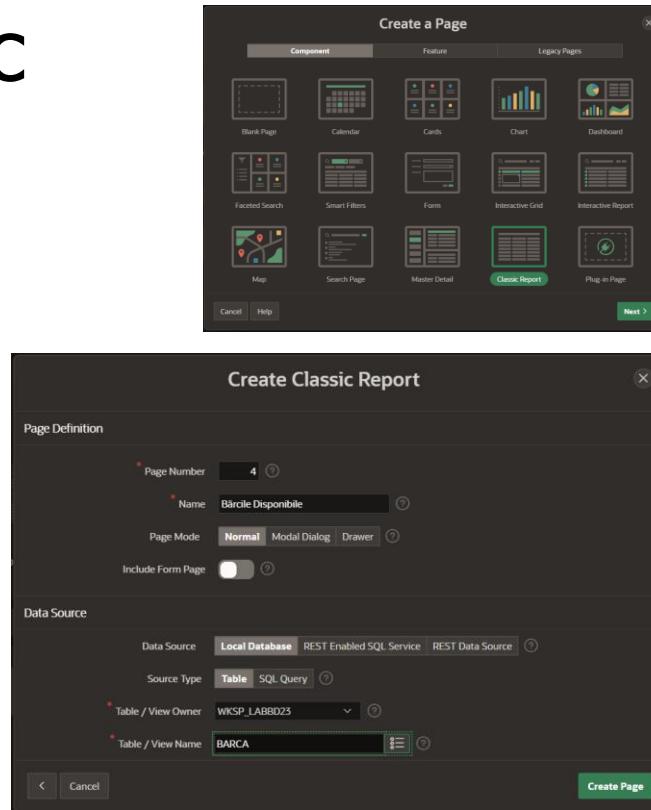


2. Se modifică proprietatea Source/HTML Code a regiunii în

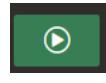


Pagină de vizualizare a unei tabele

1. Se apasă butonul Create page
2. Se selectează Classic Report și apoi Next
3. Se completează Name și tabela în Table/View name
4. Se apasă butonul Create Page



Vizualizare pagină tabelă

1. Se apasă butonul de Save și apoi Run 

Bărcile Disponibile

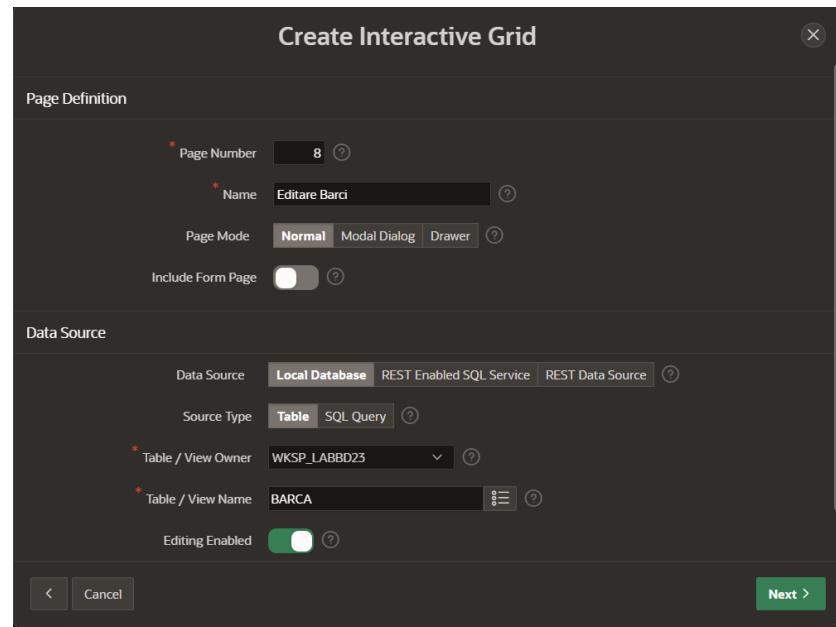
Bid ↑↓	Nume	Culoare
101	Cleo	albastra
102	Triton	rosie
103	Poseidon	verde
104	Delfinul	albastra
105	Uragan	alba
106	Mica sirena	roz

Release 1.0



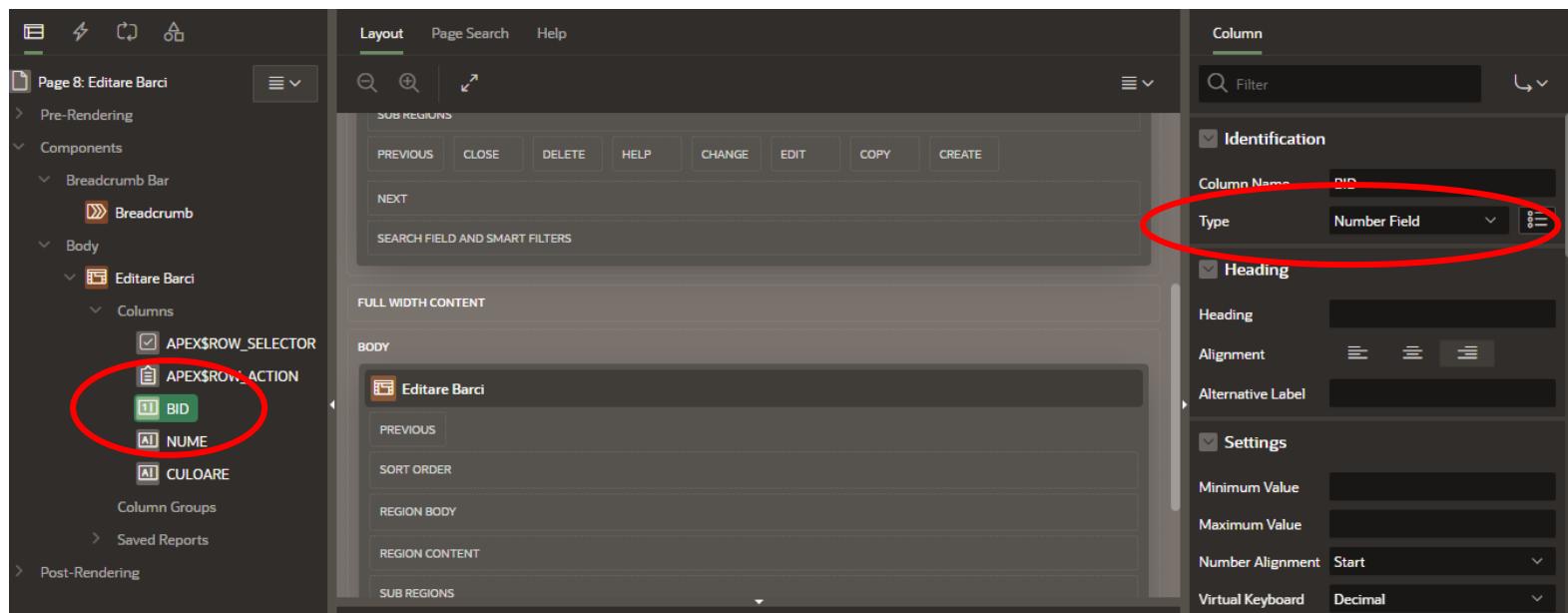
Formular editare date

1. Se apasă butonul Create page
2. Se selectează Interactive Grid și apoi Next
3. Se completează Name și tabela în Table/View name
4. Se selectează Editing Enabled
5. După Next se specifică cheia primară



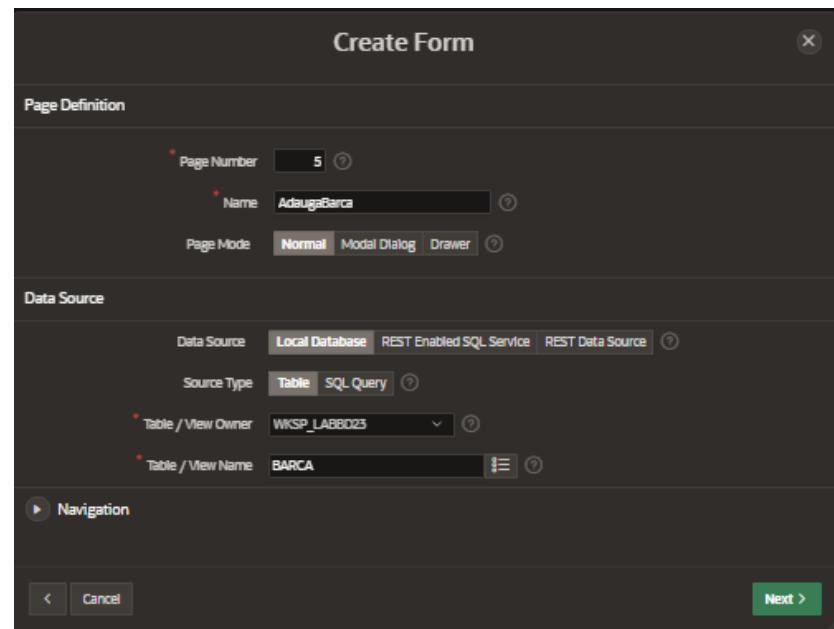
Cheie primară vizibilă pe formular

1. Se va selecta din meniul stânga Body apoi Columns și cheia primară
2. Se modifică proprietatea Identification /Type cu tipul cheii în loc de "Hidden" (invizibil)



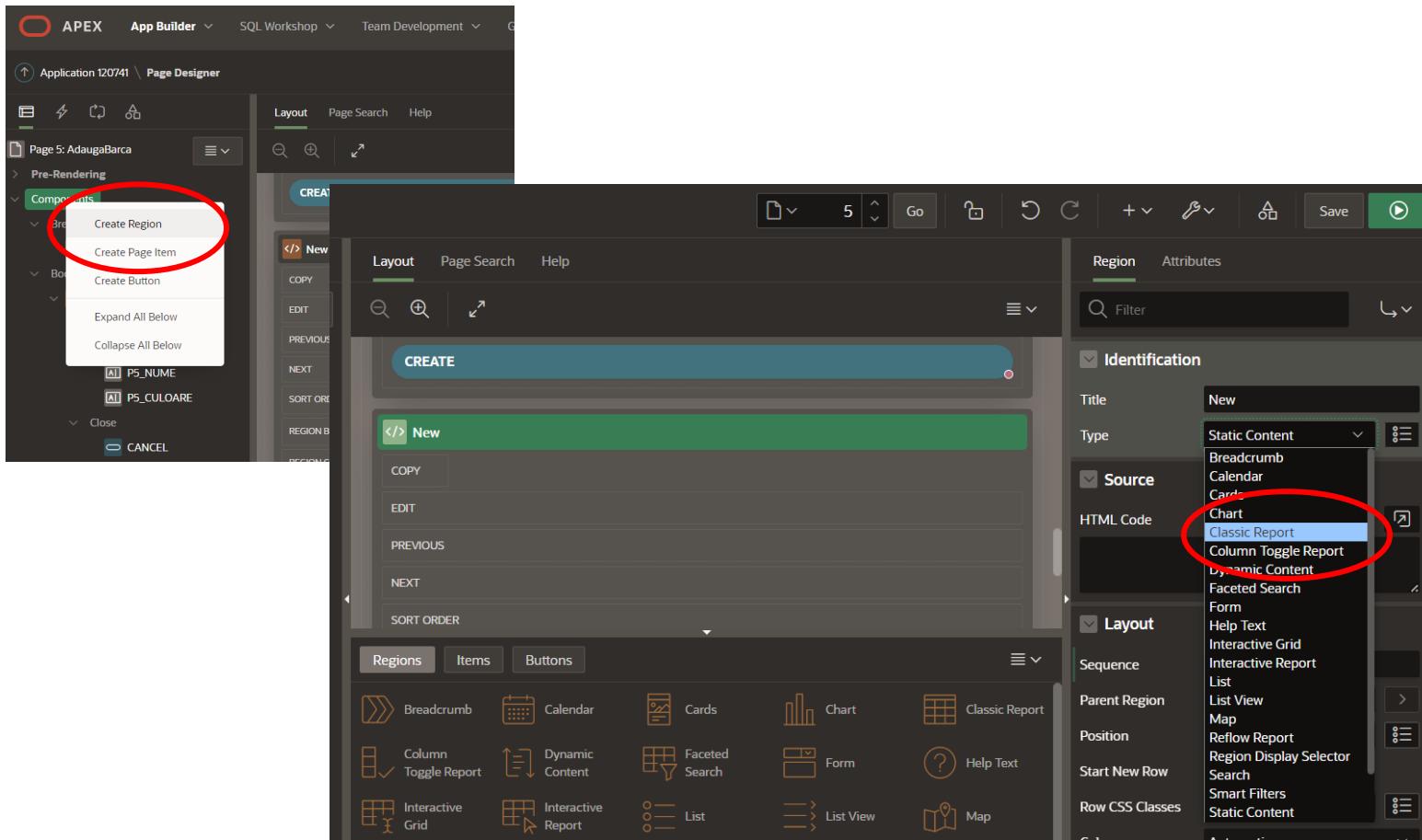
Formular adăugare Înregistrare nouă

1. Se apasă butonul Create page
2. Se selectează Form și apoi Next>
3. Se completează Name și tabela în Table/View name
4. După Next se specifică cheia primară
5. Se face apoi vizibilă cheia primară



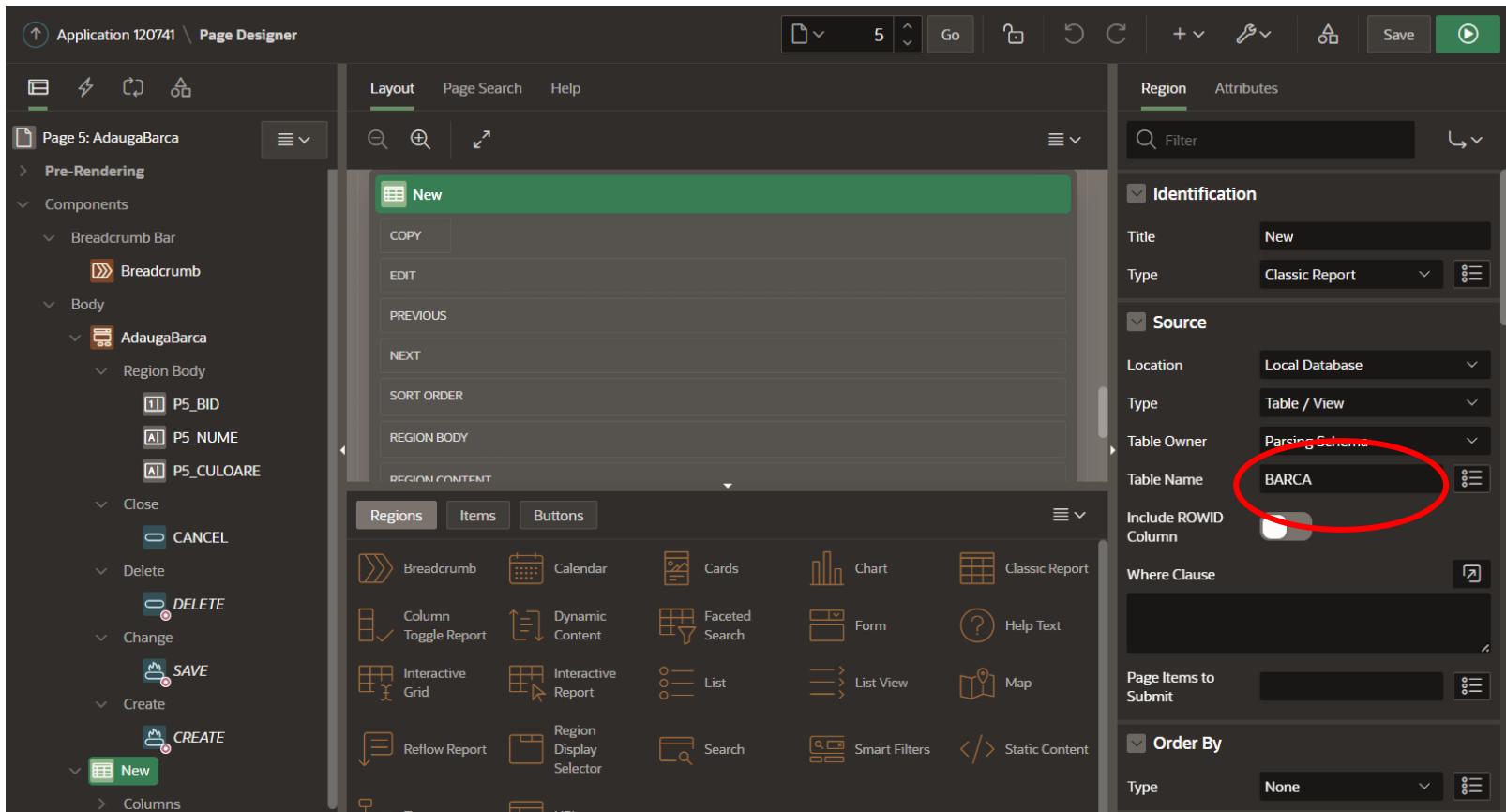
Adăugare listă înregistrări

1. Se creează o regine nouă tip Classic Report



Stabilire conținut de vizualizat

1. Se selectează tabela dorită



Formular de adăugare înregistrare

The screenshot shows a web application interface. On the left is a sidebar with the following menu items:

- Home
- Bărcile Disponibile
- Editare Barci
- AdaugaBarca** (This item is highlighted with a blue border)
- Administration

The main content area has two sections:

AdaugaBarca

This section contains a form with four input fields:

- New
- Nume
- Culoare
-

Bărci Existente

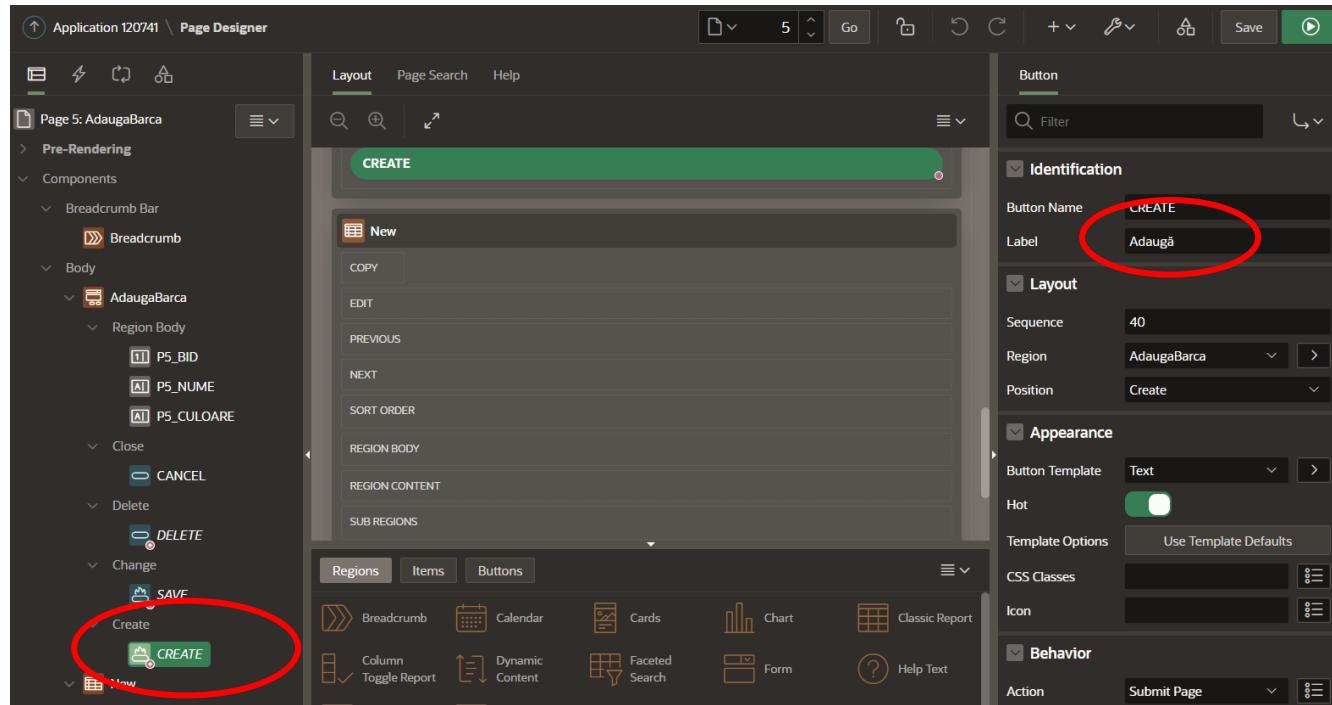
This section displays a table of existing boat registrations:

Bid ↑	Nume	Culoare
101	Cleo	albastra
102	Triton	rosie
103	Poseidon	verde
104	Delfinul	albastra
105	Uragan	alba
106	Mica sirena	roz

Below the table, there is a page number indicator: 1 - 6.

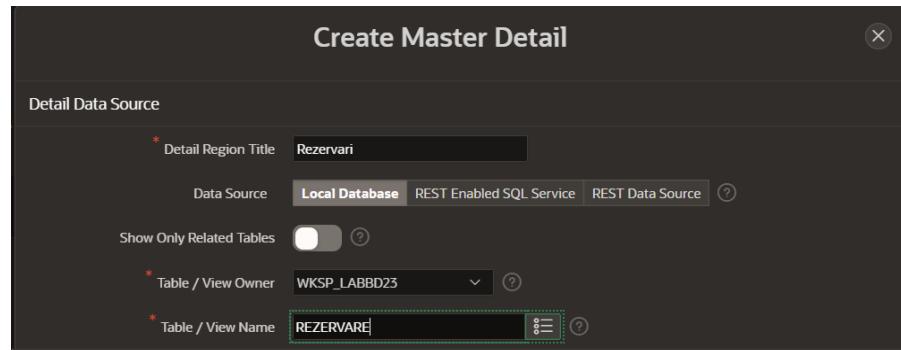
Proprietăți elemente de interfață

1. Proprietățile elementelor de interfață se pot edita din meniul din partea dreaptă (ex. Label pentru numele afișat pe un buton)



Pagini de vizualizare ierarhii

1. Se apasă butonul Create page
2. Se selectează Master Detail și apoi Next>
3. Se completează Name și tabela în Table/View name
4. După Next se specifică cheia primară și table secundară
5. Se completează apoi Master Detail Relationship (FK – PK)



Pagini vizualizare ierarhii

The screenshot displays a web-based application interface with a sidebar menu and two main content sections.

Left Sidebar:

- Home
- Bărcile Disponibile
- Editare Barci
- AdaugaBarca
- Rezervari** (highlighted)
- Administration

Top Content Area:

Rezervari

Marinari

	Nume	Rang
22	Ion	7
<input checked="" type="checkbox"/> 31	Horatiu	1
58	Oana	8
71	Constantin	9
16	Dinu	1
11	Hermina	3
4	Liviu	9
7	Dorel	2
8	Vincentiu	8
49	Gina	1
101	Vasile	4

1 rows selected Total 11

Bottom Content Area:

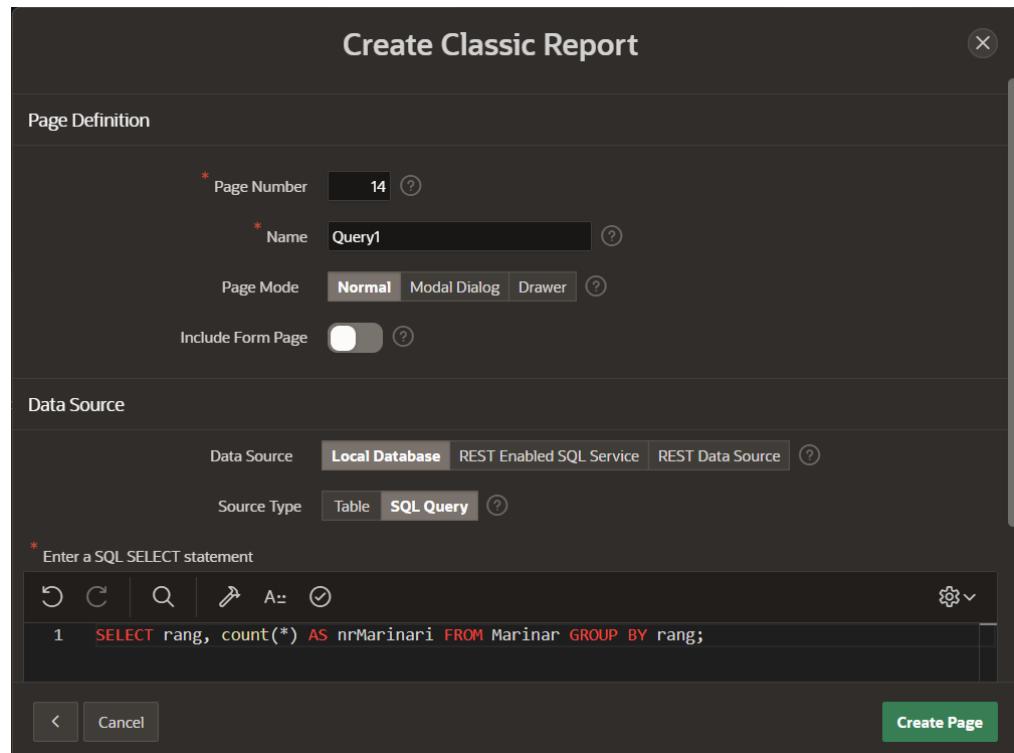
Rezervari

	Bid	Data		
<input checked="" type="checkbox"/>	22	31	101	3/10/2022
	71	31	101	10/22/2022
	11	31	101	2/17/2022
	14	31	101	1/8/2022

1 rows selected Total 4

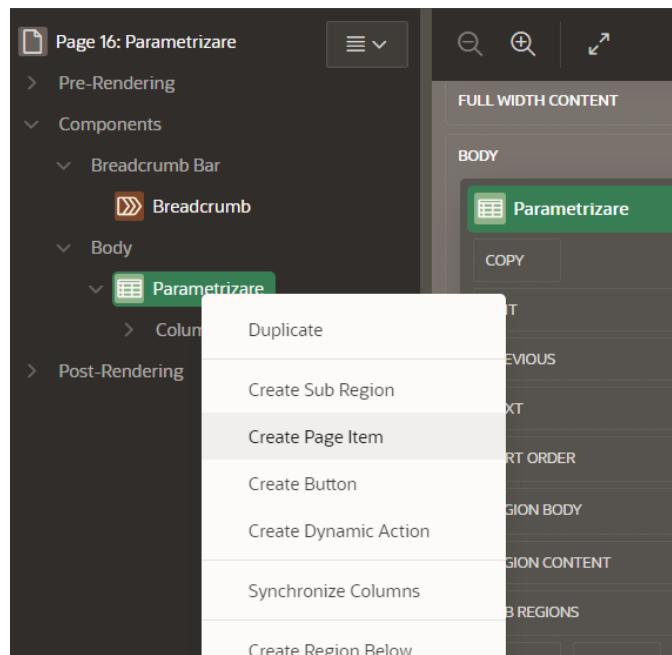
Pagini bazate pe interogări

1. Se creează o pagină tip Classic Report
2. Se selectează la Source Type: SQL Query
3. Se scrie interogarea dorită în câmpul “Enter a SQL SELECT statement”



Parametrizare vizualizare interogări

1. Se creează o pagină pentru vizualizare tabelă Marinar bazată pe interogarea
SELECT nume, rang, varsta
FROM Marinar
WHERE varsta<20;
2. Se adaugă o componentă de tip Page Item la regiunea din Body



Setare parametru

- Se modifică numele câmpului în P_VMAX

The screenshot shows the Oracle APEX Page Item settings for a page item named "P_VMAX". The "Identification" section shows the name as "P_VMAX" and the type as "Number Field". The "Settings" section shows the minimum value as 18 and the maximum value as 100. A red circle highlights the "Identification" and "Settings" sections.

- Se adaugă parametru ":P_VMAX" la interogare

The screenshot shows the Oracle APEX Region settings for a region titled "Parametrizare". The "Source" section shows the location as "Local Database" and the type as "SQL Query". The SQL query is: "SELECT nume, rang, varsta FROM Marinar WHERE varsta < :P_VMAX;". A red circle highlights the "Source" section and the SQL query.

Afișare parametrizată

1. La rulare se poate schimba valoarea de prag vârsta și apoi apăsa tastea Enter

The screenshot shows a web application interface. On the left is a sidebar menu with the following items:

- Home
- Bărcele Disponibile
- Editare Barci
- AdaugaBarca
- Rezervari
- Query1
- Parametrizare
- Administration

The main content area has a title "Parametrizare". Below it, there is a parameter input field labeled "P Vmax" with the value "50" and a table with three columns: "Nume", "Rang", and "Varsta". The table contains the following data:

Nume	Rang	Varsta
Dinu	1	19
Dorel	2	25
Gina	1	19
Hermina	3	29
Horatiu	1	33
Vincentiu	8	46

Câmpuri cu conținut dinamic

1. Se adaugă în fereastra precedentă, la Body, un câmp Page Item denumit P_SQL de tip Display Only
2. Proprietatea Source/Type se setează pe SQL Query(return single value)
3. Se adaugă interogarea:
SELECT 'SELECT nume, rang, varsta FROM
 Marinar WHERE varsta > ' || :P_VMAX
 FROM Dual
4. Se pune Used pe "Always, replacing any existing value in session state"

Câmpuri cu conținut dinamic

- Home
- Bărcile Disponibile
- Editare Barci
- AdaugaBarca
- Rezervari
- Query1
- Parametrizare
- Administration

Parametrizare

Varsta Maxima	30
---------------	----

Nume ↑=	Rang	Varsta
Dinu	1	19
Dorel	2	25
Gina	1	19
Hermina	3	29

Interogarea SQL

```
SELECT nume, rang, varsta FROM Marinar WHERE varsta > 30
```

Câmpuri BLOB Imagine

1. Pentru stocarea într-o tabelă a unei imagini se poate crea un câmp de tip BLOB
2. Adăugarea unei imagini într-o înregistrare se poate face creând un formular de adăugare înregistrări aşa cum a fost prezentat anterior
3. Pentru a afişa imaginea într-un raport se va folosi pentru câmpul respectiv tipul Display Image
4. Dimensiunea poate fi controlată din Cell Width

Câmpuri BLOB Imagine

The screenshot shows a web page editor interface with the following components:

- Left Sidebar:** Displays the page structure with "Page 19: pers view". It includes sections for Pre-Rendering, Components (Breadcrumb Bar, Body), and Columns (PID, NUME, TID, MID, POZA).
- Top Bar:** Includes Layout, Page Search, Help, and search/filter icons.
- Central Area:** Shows the "pers view" component configuration. The "BANNER" section has slots for AFTER LOGO, BEFORE NAVIGATION BAR, and AFTER NAVIGATION BAR. The "TOP NAVIGATION" and "BREADCRUMB BAR" sections are also visible.
- Right Panel:** The "Column" panel is open, showing the "Identification" section with Column Name set to "POZA" and Type set to "Display Image". The "Heading" section shows "Poza" with alignment options. The "Layout" section shows a sequence of 5.
- Preview Area:** Shows the "pers view" component in action. It contains a table with columns: Nume, Tid, Mid, and Poza. The "Nume" column has the value "Nice Face". The "Poza" column displays a cartoon illustration of a young man's face.

Baze de Date

Cap. 7. Dezvoltare aplicații BD folosind MySQL și PHP



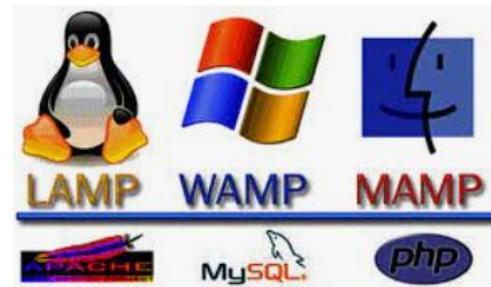
Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C15

2023 UPT

Conf.Dr. Dan Pescaru

SGBD MySQL

1. MySQL - sistem de gestiune a bazelor de date relationale tip open-source
2. A fost creat de compania suedeza MySQL AB (1995) și distribuit sub licență GNU GPL
3. Este des utilizat împreună cu limbajul PHP pentru baze de date Web mici și mijlocii
4. Este inclus în pachetele LAMP și WAMP



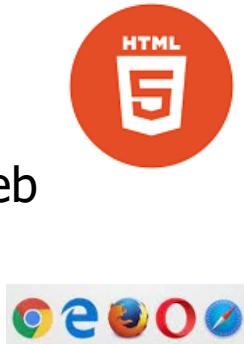
Despre MySQL

1. Include unelte de gestionare și configurare precum MySQL Administrator, MySQL Query Browser și phpMyAdmin
2. Rulează sub Linux, Windows, MacOS...
3. Cumpărat și dezvoltat de Oracle (v.5-8)
4. Derivat în MariaDB (Michael Widenius)
5. Versiuni MySQL Community Server (gratuit) și Enterprise Server (comercial)

Limbajul HTML

1. HTML = Hyper Text Mark-up Language
2. Este un limbaj de marcare (formatare)

- Folosit pentru aranjarea informațiilor în paginile web
- Se bazează pe tag-uri HTML (incluse în <...>)
- Este interpretat de navigatoare (browsere) Web



3. Este un standard întreținut de W3C
4. Prin intermediul său (+ limbaje adiționale precum CSS sau SVG) se pot aranja și formația într-o pagină Web diverse elemente precum text, tabele, imagini
5. Organizează pagina Web în două secțiuni
 - HEAD – conține meta-informații despre pagină
 - BODY – conține informația propriu-zisă



Exemplu pagină HTML

```
<head>
  <title>Prima pagina HTML</title>
  <style>
    table, th, td { border:2px solid blue; border-collapse:collapse; }
  </style>
</head>
<body>
  <h1>Pagina HTML exemplu</h1>
  <p>
    Aceasta pagina este un exemplu de utilizare taguri HTML.
  </p>
  <h2>Text formatat</h2>
  <b>Acesta</b> este un <i>text</i> formatat folosind <u>taguri <b>HTML</b></u>. <br/>
  Se pot folosi diverse <span style="color:blue">culori</span> si <span style="font-family:'Courier New'; font-size:15px">fonturi</span>
  <h2>Imagini</h2>
  Aceasta este o imagine adaugata folosind tagul &lt;img&gt;. <br/>
  
  <h2>Liste</h2>
  Listele pot fi ne-numerotate sau numerotate (ordonate).<br/>
  Lista ne-numerotata
  <ul>
    <li>Prima intrare</li>
    <li>A doua intrare</li>
  </ul>
  <br/>
  Lista numerotata
  <ol>
    <li>Prima intrare</li>
    <li>A doua intrare</li>
  </ol>
  <br/>
  <h2>Tablouri</h2>
  Tablourile HTML contin randuri si coloane definite cu &lt;tr&gt; si respectiv &lt;th&gt; pentru cap de colana si &lt;td&gt; pentru continut
  Tablou <b>persoane</b>
  <table>
    <tr style="background-color: #D6EEEE;"> <th>Nr</th><th>nume</th><th>varsta</th> </tr> <!-- cap de tabel -->
    <tr> <td>1</td><td>Popescu</td><td>23</td> </tr>
    <tr> <td>2</td><td>Ionescu</td><td>27</td> </tr>
    <tr> <td>3</td><td>Vasilescu</td><td>22</td> </tr>
  </table>
</body>
```

Exemplu pagină HTML

← → C G pagina.html

Pagina HTML exemplu

Aceasta pagina este un exemplu de utilizare taguri HTML.

Text formatat

Acesta este un *text* formatat folosind **taguri HTML**.
Se pot folosi diverse **culori** si fonturi

Imagini

Aceasta este o imagine adaugata folosind tagul .



Liste

Listele pot fi ne-numerotate sau numerotate (ordonate).
Lista ne-numerotata

- Prima intrare
- A doua intrare

Lista numerotata

1. Prima intrare
2. A doua intrare

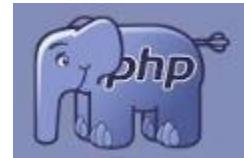
Tablouri

Tablourile HTML contin randuri si coloane definite cu <tr> si respectiv <th> pentru cap de colana si <td> pentru continut de coloana.
Tablou persoane

Nr	nume	varsta
1	Popescu	23
2	Ionescu	27
3	Vasilescu	22

Limbajul PHP

1. Limbaj de tip scripting, open-source, rulează pe partea de server
 - Folosit pentru a genera pagini web dinamice
 - Scripturi PHP încorporate în paginile HTML
2. Lansat în 1995, de Rasmus Lerdorf ca un set de instrumente de script Perl/CGI (Personal Home Page sau PHP)
3. PHP2 lansat în 1997 (ca "PHP Hypertext Processor")
4. PHP5 lansat în 2004 (> 100 biblioteci, > 1.000 de funcții, programare OO extinsă etc.)
5. PHP8 lansat în 2020 (compilare just-in-time, expresii regulate, operator Nullsafe etc.)



Avantaje PHP

1. Limbajul interpretat, scripturile sunt analizate în timpul execuției și nu compilate în prealabil
2. Execut pe partea de server (ascuns pentru browserul client – opțiunea Vizualizare sursă în browser nu afișează scriptul)
3. Diverse funcții încorporate permit aplicații variate - grafică, e-mail, pdf etc.
4. Compatibil cu multe SGBD-uri populare (MySQL, Oracle, MsSQL Server, IBM DB2...)

Codul PHP

1. Similar ca sintaxa/instructiuni de bază cu C/C++/Java
2. Inclus în fișiere HTML folosind tag-ul `<?php ... ?>`. Se permit mai multe secțiuni PHP în aceeași pagină
3. Permite programare structurată / POO
4. Instructiunile se despart prin `';'`
5. Permite comentarii gen C/C++/Shell (`/* ... */`, `// ...`, `# ...`)

Ref: <https://www.w3schools.com/php/>

Variabilele PHP

1. Numele unei variabile începe cu **\$**
2. Sunt case sensitive (`$a != $A`)
3. PHP este un limbaj slab tipizat
4. Variabilele au vizibilitate locală sau globală
 - Cele locale restricționate în cadrul unei funcții
 - Cele globale utilizabile și în afara funcțiilor
5. Există câteva variabile predefinite
 - Pt. date din formulare (`$_GET`, `$_POST`, `$_FILE`)
 - Date aplicație Web (`$_SESSION`, `$_COOKIE`)
 - Date despre serverul Web (`$_SERVER`)

Tipurile de date PHP

1. Tipuri de bază

- Logic (boolean) `true, false`
- Întreg (integer) `[-2,147,483,648 .. 2,147,483,647]`
- Real (float)
- Sir de caractere (string)
- Tablou, tablou asociativ (array)

2. Tipuri complexe

- Obiecte PHP (object)
- Tipuri specifice (pdfdoc, pdfinfo)

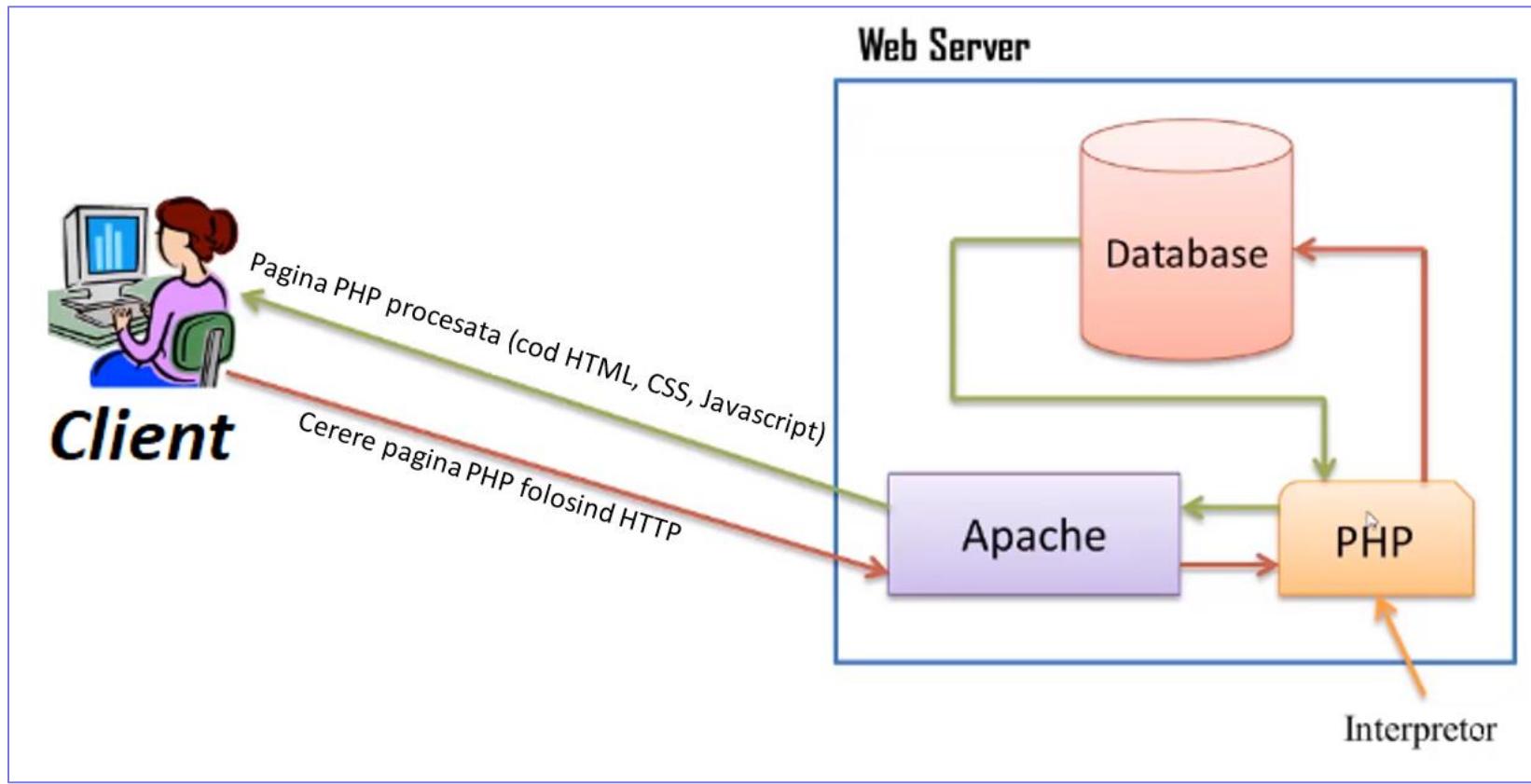
3. Lipsă valoare – `null`

Generare HTML

1. Prin instrucțiunea PHP "echo" sau prin unele funcții PHP (print, printf, print_r, die, ...).
2. Se va insera informații în pagina încărcată de browser-ul web al clientului

```
<?php
    $a = 3;                      // Variabilă numerică
    $b = "abc";                   // Variabilă sir
    echo $a;                      // Inserează: 3
    echo $a, $b;                  // Inserează: 3abc
    echo "1+2=".$b;               // Inserează: 1+2=3
    echo "1+2=$b";                // Inserează: 1+2=3
    echo '1+2='.$b.'<br />'; // Inserează: 1+2=$b + rând nou
?>
```

Interpretare limbaj PHP



Prelucrata după: <https://www.devopschool.com/>

Operatori PHP

1. Arithmetici

- (+, -, *, /, %, ++, --)

2. Concatenare şiruri, evaluare şiruri

- (.), ("... ")

3. Comparatii

- (==, ===, !=, <>, <, <=, >, >=)

4. Pentru tipuri de date

- (instanceof, is_bool, is_int, is_string, is_object ...)

5. Atribuire

- (=, +=, -=, *=, /=, %=, .=)

6. Extragere referință

- (&\$var)

Structuri condiționale de control

1. IF/ELSE/ELSEIF

```
<?php  
    if($a<3) {  
        echo $a;  
        $a=5;  
    }  
    elseif($a<6) {  
        echo "corect!";  
    }  
    else  
        echo "greșit";  
?>
```

2. SWITCH

```
<?php  
switch ($a) {  
    case 0:  
        echo "Este zero";  
        break;  
    case 1:  
        echo "Este unu";  
        break;  
    default:  
        echo "Prea mare";  
    }  
?>
```

Cicluri WHILE

1. WHILE

```
<?php  
    $a=0;  
    while ($a<5)  
    {  
        echo "a = $a; ";  
        $a ++;  
    }  
?>
```

2. DO WHILE

```
<?php  
    $a = 0;  
    do {  
        echo "a = $a; ";  
        $a++;  
    } while ($a < 5);  
    echo "<br />done.";  
?>
```

Cicluri FOR

1. FOR

```
<?php
for ($a=1; $a<$b; $a++)
{
    if ($a >5) {
        break;
    }
    echo $a;
}
?>
```

2. FOREACH

```
<?php
$bon = array(
    "pâine" => 4.5,
    "roșii"  => 9.0,
    "carne"   => 26.5
);
foreach ($bon as
    $prod => $pret) {
    echo "Prețul pt. $prod
este $pret lei.";
}
```

Accesare date formulare HTML

1. Pentru formularul (fișier "citire.html")

```
<form name="formular" method="post" action="frmproc.php">
    <p>Date personale <br />
    Nume<input type="text" name="nm" maxlength="36" />
    <br />
    Prenume<input type="text" name= "pn" maxlength="36" />
    <br /> Transport favorit
    <select name="transp">
        <option value="Masina">Masina</option>
        <option value = "Tren">Tren</option>
        <option value = "Avion">Avion</option>
    </select> <br />
    <input type="submit" name="send" value="Salvare"/ >
    <input type="reset" name="del" value="Resetare" />
</form>
```

Cod accesare date formulare HTML

1. Fișierul PHP țintă "frmproc.php"

<h1>Date transmise:</h1>

<?php

\$nume = \$_POST["nm"];

\$prenume = \$_POST["pn"];

\$transport = \$_POST["transp"];

echo "Nume: ".\$nume."
";

echo "Prenume: ".\$prenume."
";

echo "Optiune transport: ".\$transport."
";

?>

Functii PHP

1. Funcțiile trebuie definite înainte de a putea fi apelate
2. Antetul unei funcții este de forma
function numeFct(\$arg1, \$arg2, ..., \$argN)
3. Nu este specificat niciun tip returnat
4. Spre deosebire de variabile, la numele funcțiilor nu se face distincție între caractere mici/mari (fct1(...), FCT1(...) sau Fct1(...))
5. Pot fi grupate în biblioteci utilizate cu funcția PHP **include("numeBiliotecă.php")**

Funcții PHP de acces la BD mySQL

1. new mysqli(\$host, \$user, \$pass, \$dbName)
2. mysqli_query() // trimite o interogare
3. mysqli_num_rows () // valabilă pt. SELECT
4. mysqli_affected_rows ()// pt. INSERT, UPDATE, DELETE
5. mysqli_fetch_array() // returnează un tablou
6. mysqli_fetch_assoc() // returnează un tablou asociativ
7. mysqli_fetch_object() // returnează un obiect
8. mysqli_error() // returnează ultima eroare
9. mysqli_close() // închide conexiunea la server

Exemplu de acces la o BD mySQL

```
<?php
    $host = 'localhost';
    $username = 'utilizator';
    $paswd = '12345';
    $dbName = 'persoane';
    $conn = mysqli_connect($host, $username, $paswd,
                           $dbName);

    if ($conn->connect_error){
        die('Nu ma pot conecta la server: '.
            mysqli_error($conn));
    }
?>
```

Citirea datelor dintr-o BD mySQL

```
<?php
$query = "SELECT nume, prenume, transp FROM Persoana";
$rez = mysqli_query($conn, $query);
if (!$rez) die(' Interogare eșuată: '. mysqli_error($conn));
$nr_inreg = mysqli_num_rows($rez);
echo "<h3>".$nr_inreg." persoane in BD.</h3>";
echo "<table><tr><th>Nume</th>";
echo "<th>Prenume</th><th>Transport</th></tr>";
while($inreg = mysqli_fetch_assoc($rez)) {
    echo "<tr><td>".$inreg["nume"]."</td>";
    echo "<td>".$inreg["prenume"]."</td>";
    echo "<td>".$inreg["transp"]."</td></tr>";
}
echo "</table>";
?>
```

Baze de Date

Cap. 8. Modelare date relationale



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C15

2023 UPT

Conf.Dr. Dan Pescaru

Modelul Relațional

1. O colecție de relații (tabele) și legăturile dintre acestea
2. Relație (tabelă)

- Schema: nume relației, numele și tipul atributelor (coloanelor)
- Instanța: tabela fizică (pe disc) având un număr fix de coloane și un număr variabil de rânduri
- Numărul de coloane (attribute): gradul relației
- Numărul de rânduri (înregistrări): cardinalitatea
- Rândurile sunt DISTINCTE și NU sunt ordonate!

Constrângerile de integritate

1. O colecție de expresii logice care trebuie să fie satisfăcute de orice instanță a bazei de date
2. Sunt specificate la definirea schemei bazei de date
3. Sunt verificate la fiecare modificare efectuată în baza de date
4. O instanță de bază de date care satisface toate constrângerile de integritate se numește "legală" sau "consistentă"

Chei

1. Un set de attribute formează o **cheie** pentru o relație dacă:

- Nu există pereche de înregistrări care să aibă valori egale pentru toate attributele din set (asigură unicitatea)
- Orice subset al acestuia încalcă cerința de mai sus (trebuie să fie minimă)

2. O **supercheie**: un set de attribute care include cel puțin o cheie

Cheie primară (PK)

1. Cheie primară: o cheie aleasă de proiectantul bazei de date din setul de chei, care îndeplinește următoarele condiții:
 - Este scurtă (cuprinzând un număr minim de atribute - ideal doar unul).
Obs: (caz neobișnuit) doar o cheie candidat formată din toate câmpurile tablei
 - Este reprezentativă în contextul problemei (de exemplu, SID vs. CNP) și nu permite valori NULL
2. Alternativă: introducerea unei chei artificiale (de obicei numerică, folosind AutoIncrement)

Integritate referențială

1. Cheie externă (FK) – leagă două tabele (coresponde cheii primare a relației principale)
2. Folosită pentru a verifica integritatea referențială
 - Când se adaugă o înregistrare în tabelul secundar: cheia corespunzătoare trebuie să existe
 - La ștergerea unei înregistrări din tabelul principal: pentru evitarea înregistrărilor “orfane” din tabelul secundar
 - La modificarea valorii unei chei primare sau a unei chei externe în tabelele aferente

Asigurarea integrității referențiale

1. La adăugare (INSERT)

- "Avoiding operation that breaks constraints"

2. La ștergere (DELETE) se poate alege

- "Cascade delete related records"
- "Avoiding operation that breaks constraints"
- "Voiding the reference" (nu prea des)

3. La modificare (UPDATE) se poate alege

1. "Cascade update related records"
2. "Avoiding operation that breaks constraints"

4. Probleme: blocaje (referințe încrucișate) – soluție: utilizarea tranzacțiilor (verificare întârziată)

Integritate referențială. Example

Student

marca	nume	an	media
AC2153	Pop Angela	2	8.50
AC1078	Avram Ioan	1	9.35
AC2056	Ionescu Mihai	2	7.80

Contract

cid	marca	lab	ex
PLA2	AC2153	9	8
UC1	AC1078	10	9
SO2	AC2056	8	7

1. INSERT un nou contract: verifică marca in tabela Student
2. DELETE sterge un student: verifică marca in Contract
3. UPDATE modifică marca in Student: verifică marca in Contract / modifică marca in Contract – verify marca in Student

Modelul relațional: redundanță

1. Redundanță este la baza multor probleme asociate cu BD relationale
 - Risipirea spațiului de depozitare
 - Anomalii la inserare / ștergere / actualizare
2. Exemplu:

Student

sid	nume	an	media	facultatea	adresa
AC2153	Pop Angela	2	8.50	Automatică și Calculatoare	Bd. V. Pârvan no,2, Electro
AC1078	Avram Ioan	1	9.35	Automatică și Calculatoare	Bd. V. Pârvan no,2, Electro
AC2056	Ionescu Mihai	2	7.80	Automatică și Calculatoare	Bd. V. Pârvan no,2, Electro

Abordarea redundanței

1. Constrângerile de integritate, în special dependențele funcționale, pot fi utilizate pentru a identifica scheme cu redundanță mare și pentru a sugera pași de rafinare
2. Tehnica principală de rafinare: decompoziția
 - Înlocuirea unei relații **ABCD** cu, de exemplu, **AB** și **BCD**, sau **ACD** și **ABD**
3. Descompunerea ar trebui folosită cu prudentă:
 - Există motive pentru a descompune o relație?
 - Ce probleme (dacă există) sunt provocate de decompoziție?

Dependențe funcționale DF

1. O dependență funcțională $X \rightarrow Y$ există în relația R dacă, pentru fiecare instanță admisă r a lui R:
 $t_1 \in R, t_2 \in R, \pi_X(t_1) = \pi_X(t_2) \Rightarrow \pi_Y(t_1) = \pi_Y(t_2)$
2. Având două înregistrări în R, dacă valorile X sunt de egale, atunci și valorile Y trebuie să fie egale - unde X și Y sunt seturi de attribute
3. O DF este o declarație despre toate instanțele posibile și trebuie să fie identificat pe baza semanticii sistemului de implementat
4. Având în vedere o instanță r_1 a lui R, putem verifica dacă aceasta încalcă unele DF f , dar nu și dacă un set de DF f este adevărat pentru R

Exemplu de DF

1. Se consideră o relație Angajati_Sezonieri (**Cnp**, **Nume**, **nivel_Pregatire**, **pRoiect**, **Salar_orar**, **Ore_lucrare**)
2. Notație: vom nota această schemă prin enumerarea atributelor sale – **CNPRSO** – corespunzător setului de attribute a relației
3. Uneori, ne vom referi la toate attributele unei relații folosind numele relației
4. Exemple de DF pentru Angajati_Sezonieri :
 - cnp este cheia primară: **C** → **NPRSO**
 - nivelPregătire determină salarOrar: **P** → **S**

Exemplu de probleme generate de DF

1. Probleme din cauza $P \rightarrow S$

- Anomalii de actualizare: dacă schimbam S doar în prima înregistrare a relației?
- Anomalie de inserare: ce se întâmplă dacă dorim să inserăm un angajat cu un salariu orar diferit decât cel pentru pregătirea sa?
- Anomalie de ștergere: dacă ștergem toți angajații cu nivelPregătire 3, pierdem informațiile despre salariu pentru nivelPregătire 3!

C	N	P	R	S	O
1850213988166	Marius	4	P3	12	40
1870925988352	Ion	2	P1	8	36
2891105988058	Rodica	3	P1	10	30
1911204988125	Petru	1	P2	5	32
2900217988855	Mona	2	P3	8	40

Determinarea DF prin inferență

- Dacă știm DF de bază, putem infera DF adiționale:
 - $cnp \rightarrow codf \wedge codf \rightarrow decan$ implică $cnp \rightarrow decan$
- O DF f este implicată de un set de DF F dacă f este adevărată oricând F este adevărat
 - F^+ (*închiderea* lui F): setul tuturor DF f implicate de F
- Axiomele lui Armstrong (X, Y, Z: seturi de attribute):
 - Reflexivitatea: dacă $X \subseteq Y$, atunci $Y \rightarrow X$
 - Augmentarea: dacă $X \rightarrow Y$, atunci $XZ \rightarrow YZ$ pentru orice Z
 - Tranzitivitatea: dacă $X \rightarrow Y$ și $Y \rightarrow Z$, atunci $X \rightarrow Z$
- Acestea reguli de inferență sunt *necesare și suficiente* pentru a determina toate dependențele funcționale!

Exemplu de extragere DF

- O relație R are attribute (S, C, T, A, N) care denotă student, curs, timp, amfiteatru și nota. Din cerințe se pot deduce următoarele DF:
 - $SC \rightarrow N$
 - $ST \rightarrow A$
 - $C \rightarrow T$
 - $TA \rightarrow C$

Normalizare

- În ceea ce privește proiectarea schemei bazei de date, o întrebare bună de pus este când este necesară rafinare ei. Un răspuns simplu: în caz de redundanță
- Rezolvare: normalizare prin decompoziție
- Motiv: dacă o relație este într-o anumită formă normală, anumite tipuri de probleme sunt evitate/minimalizate
- Utilizarea DF în detectarea redundanței: (R:ABC)
 - Nu există FD-uri: nu există redundanță
 - Dacă avem $A \rightarrow B$: având mai multe înregistrări cu valori A rezultă mai multe cu aceeași valoare B!

Normalizare – scurt istoric

- Edgar F. Codd a propus procesul de normalizare și ceea ce a ajuns să fie cunoscut sub numele de prima formă normală în lucrarea sa <A Relational Model of Data for Large Shared Data Banks>: „Există o procedură de eliminare foarte simplă pe care o vom numi normalizare. Prin decompoziție, domeniile non-simple sunt înlocuite cu domenii ale căror elemente sunt valori atomice (ne-decompozabile).”
- El a stabilit inițial trei forme normale: 1NF, 2NF și 3NF. Există acum și altele care sunt acceptate, dar 3NF este considerat în general a fi suficient (BD sunt considerate normalize dacă sunt în 3NF)

Decompoziția unei scheme relaționale

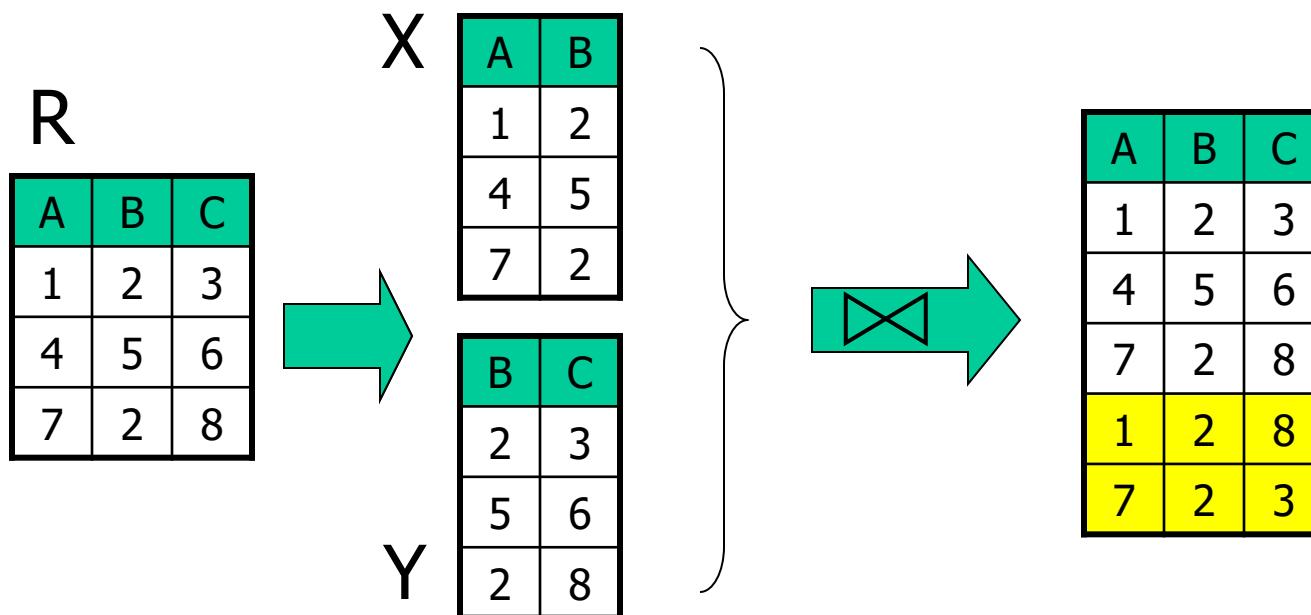
1. Să presupunem că relația R conține atributele $A_1 \dots A_n$. O descompunere a lui R constă în înlocuirea lui R cu două sau mai multe relații astfel încât:
 - Fiecare nouă schemă conține un subset de atribută ale lui R (și nici un atribut care nu apare în R)
 - Fiecare atribut al lui R apare ca un atribut al cel puțin uneia dintre noile relații
2. Intuitiv, descompunerea lui R înseamnă că vom stoca instanțe ale schemei relației produse prin decompoziție în loc de instanțe ale lui R
3. De exemplu, putem descompune **ABCDEFG** în **ABCDE** și **BFG**

Decompoziții tip "lossless join"

1. Decompoziția lui R în X și Y este fără pierderi la JOIN în raport cu o mulțime de DF F dacă, pentru fiecare instanță r care satisface F
 - $\Pi_X(r) \bowtie \Pi_Y(r) = r$
2. Definiția se extinde la decompoziția în 3 sau mai multe relații într-un mod simplu
3. Este esențial ca toate decompozițiile utilizate pentru reducerea redundanței să fie fără pierderi la JOIN!

Contra exemplu la "lossless join"

1. Decompoziția următoare a lui R în X și Y nu are proprietatea dorită!



Conservarea dependențelor

1. Decompozițiile trebuie să păstreze toate dependențele funcționale ale relației originale pentru a păstra toate constrângerile!

- Condiția este: $(F_1 \cup F_2)^+ = F^+$

2. Exemplu

$$R = (A, B, C), \quad F = \{A \rightarrow B, \quad A \rightarrow C, \quad B \rightarrow C\}$$

$$R_1 = (A, C), \quad R_2 = (B, C)$$

$$F_1^+ = \{A \rightarrow A, \quad C \rightarrow C, \quad A \rightarrow C, \quad AC \rightarrow CC\}$$

$$F_2^+ = \{B \rightarrow B, \quad C \rightarrow C, \quad B \rightarrow C, \quad BC \rightarrow BC\}$$

- Obs: $A \rightarrow B$ nu este păstrată

Forma Normală 1 (1NF)

1. Formă normală 1 (1NF):

1. Domeniul fiecărui atribut trebuie să conțină **doar valori atomice**; câmpurile compuse sau „relațiile în relație” sunt interzise
2. Fiecare atribut conține doar **o singură valoare** în acel domeniu

Student

<u>sid</u>	nume	hobby	codf	adresa
AC6978	Popescu Mihai	șah, dans	AC	Timisoara, Parvan no 1 , 0256112212
AC8967	Ionescu Georgeta	citit, muzică	AC	Timisoara, Parvan no 1 , 0256112212

Forma Normală 1 - exemplu

1. Soluția: două diviziuni de atrbute și o decompoziție

Student

<u>sid</u>	nume	hobby	codf	adresa
AC6978	Popescu Mihai	șah, dans	AC	Timisoara, Parvan no 1, 0256112212
AC8967	Ionescu Georgeta	citit, muzică	AC	Timisoara, Parvan no 1, 0256112212

Student

<u>sid</u>	prenume	nume	codf	oraș	stradă	telefon
AC6978	Mihai	Popescu	AC	Timisoara	Parvan no. 1	0256112212
AC8967	Georgeta	Ionescu	AC	Timisoara	Parvan no. 1	0256112212

Hobby

<u>sid</u>	hobby
AC6978	șah
AC6978	dans
AC8967	citit



Forma Normală 2 (2NF)

1. Formă normală 2 (2NF):

1. Relația este deja în 1NF
2. Orice atribut non-prim din R (nu face parte din cheia primară) trebuie să fie complet dependent funcțional de cheia primară din R
SAU: Nu există attribute care să depindă doar de o parte a cheii primare

Factura

id	data	poz	cant	produs	pretUnitar	pretTotal
008978	01-03-2014	1	2	pâine	5	10
008978	01-03-2014	2	5	mere	8	40
099488	05-03-2014	1	1	brânză	26	26

Forma Normală 2 - Exemplu

1. Solutia: decompozitia

Factura

<u>id</u>	<u>data</u>
008978	01-03-2014
008978	01-03-2014
099488	05-03-2014



PozitiiFactura

<u>id</u>	<u>poz</u>	<u>cant</u>	<u>produs</u>	<u>pretUnitar</u>	<u>pretTotal</u>
008978	1	2	pâine	5	10
008978	2	5	mere	8	40
099488	1	1	brânză	26	26

Forma Normală 3 (3NF)

1. Formă normală 3 (3NF):
 1. Relația este deja în 2NF
 2. Fiecare atribut non-prim depinde în mod netranzitiv de fiecare cheie candidată din tabel. Cu alte cuvinte, nu este permisă nici o dependentă tranzitivă
2. BCNF: Fiecare dependentă funcțională netrivială din tabel este o dependentă de o supercheie

PozitiiFactura

<u>id</u>	<u>poz</u>	produs	cant	pretUnitar	pretTotal
008978	1	pâine	2	5	10
008978	2	mere	5	8	40
099488	1	brânză	1	26	26

Forma Normală 3 - Exemplu

1. Solutia: decompozitia

PozitiiFactura

<u>id</u>	<u>poz</u>	<u>produs</u>	<u>cant</u>	<u>pretUnitar</u>	<u>pretTotal</u>
008978	1	pâine	2	5	10
008978	2	mere	5	8	40
099488	1	brânză	1	26	26

CatalogProduse

<u>produs</u>	<u>pretUnitar</u>
pâine	5
mere	8
brânză	26



Forma Normală 4 (4NF)

1. Formă normală 4 (4NF):

1. Relația este deja în 3NF
 2. Nu există dependențe funcționale multivaloare
2. Dependențe multivaloare – la o schema ABC, dependența multivaloare există dacă fiecărui A îi corespund mai mulți B și mai mulți C, dar B și C sunt independente unul de celălalt

Componente

<u>Departament</u>	<u>Project</u>	<u>Componenta</u>
D1	Pr1	C1
	Pr2	C2
		C3
D2	Pr2	C2
	Pr3	C4
	Pr5	

Componente (3NF)

<u>Departament</u>	<u>Project</u>	<u>Componenta</u>
D1	Pr1	C1
D1	Pr1	C2
D1	Pr1	C3
D1	Pr2	C1
D2	Pr2	C2
...
D2	Pr5	C4

Forma Normală 4 - Exemplu

1. Problemă: să presupunem că departamentele mențin un stoc de componente și dezvoltă proiecte care pot folosi unele dintre componentele disponibile sau toate. Rezolvare: decompoziția

Componente (3NF)

<u>Departament</u>	<u>Project</u>	<u>Componenta</u>
D1	Pr1	C1
D1	Pr1	C2
D1	Pr1	C3
D1	Pr2	C1
D2	Pr2	C2
...
D2	Pr5	C4



DP (4NF)

<u>Departament</u>	<u>Project</u>
D1	Pr1
D1	Pr2
D2	Pr2
...	...

DC (4NF)

<u>Departament</u>	<u>Componenta</u>
D1	C1
D1	C2
D1	C3
D2	C2
...	...

Baze de Date

Cap. 9. Optimizare acces la date. Indexare.



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C15

2023 UPT

Conf.Dr. Dan Pescaru

Stocarea datelor

1. RAM

- + Procesare eficientă
- Capacitate redusă, cost mare, volatilă

2. HDD

- + Capacitate medie/mare. Preț mediu. Permanentă
- Timp acces mare ($>10^3$ RAM)

3. DVD, optic, bandă magnetică

- + Capacitate medie/mare. Preț mic
- Acces lent la date, bandă - acces secvențial

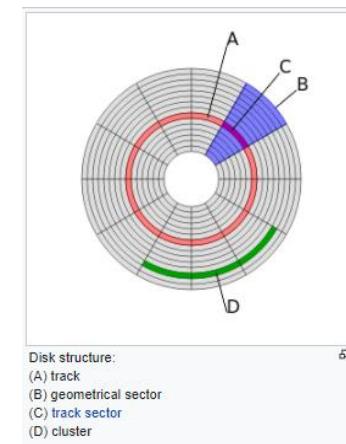
Stocarea pe HDD

1. Sistem de fișiere (S.O.)

- Organizare pe sectoare / blocuri
- Acces lent datorat parților mecanice (cap RW)
- Citirea paginilor consecutive este mai eficientă decât citirea lor în ordine aleatoare
- Poate fi gestionat de SO sau SGBD

2. Citirea/scrierea datelor

- Citire/scriere sectoare + utilizare buffere
- Management buffere: SO sau SGBD?



Fișier de date

1. O secvență de înregistrări

- Înregistrările sunt mapate pe sectoarele discului
- Înregistrări de lungime fixă sau variabilă (căutare eficientă vs. optimizare stocare)
- Înregistrări de lungime variabilă presupun organizarea într-o listă

2. Structura fișierului de date

- File header (ex. la dBase)
- Catalog al bazei de date (ex. Oracle, Microsoft SQL Server, IBM DB2, MySQL etc.)

Optimizarea accesului la date

1. Fișiere neordonate (heap files)

- Înregistrările sunt în ordine aleatoare
- Sunt potrivite dacă este necesar doar parcursarea secvențială a tuturor înregistrărilor
- Căutările bazate pe condiții logice sunt ineficiente

2. Soluții pentru accelerarea căutării

- Sortarea înregistrărilor
- Indexare (arbori B+, tehnici hashing, clusterizare, virtualizare spațiu etc.)

Sortare fișiere

1. Eficiente pentru

- Extragerea înregistrărilor în ordinea respectivă
- Extragerea unui subinterval ordonat
- Căutare binară (complexitate $O(\log_2(N))$)

2. Probleme

- Adăugare/ștergere/modificare ineficiente
- Un singur criteriu de sortare este posibil la un moment dat
- Resortarea după alt criteriu este foarte ineficientă

Sortare fișiere - xBase

SORT TO fisier

ON exp1 [/A][/C][/D], exp2 [/A], ...
[FOR conditie]
[ASCEN/DESC]

- Creează un fișier nou (spațiu necesar dublu)
- Parametrii
 - /A – sortare ascendentă
 - /C – nu ține cont de litere mari/mici
 - /D – sortare descendente
 - ASCE/DESC – ordine sortare globală (pentru toate expresiile de sortare)

Indexare

1. Alternativă mai bună la sortare

2. Eficientă pentru

- Extragerea înregistrărilor în ordinea indexului
- Extragerea unui subinterval ordonat
- Căutare pe arbore B+ (complexitate $O(\log_k(N))$) sau structură hash (acces în medie 1.2 citiri)
- Mai mulți indecsi simultan (criterii de sortare)

3. Probleme

- Adăugare/ștergere/modificare ineficiente
- Spatiu suplimentar pe disc pentru indecsi

Indexare – idea de bază

1. Index simplu

index

cheie	referință
K_1	(1)
K_2	(2)
K_3	(3)
...	
K_N	(N)

tablelă

cheie	câmp ₁	...	câmp _M
K_3	Val ₃₁		Val _{3M}
K_N	Val _{N1}		Val _{NM}
K_2	Val ₁₁		Val _{1M}
...			
K_1	Val ₂₁		Val _{2M}

Index - organizare

1. Un index pe un fisier accelerează selectiile în câmpurile cheie de căutare pentru index

- Orice subset de câmpuri ale unei relații poate fi cheia de căutare pentru un index al relației
- Cheia de căutare nu este aceeași cu o cheie a unei relații (set minim de câmpuri care identifică în mod unic o înregistrare într-o relație)

2. Un index asigură regăsirea tuturor intrărilor de date k^* pentru cheia k

Tipuri de indexi

1. Primar vs. secundar vs. unic

- Dacă cheia de căutare conține cheia primară este index primar, dacă conține o cheie este unic

2. Clusterizat vs. neclusterizat

- Dacă ordinea înregistrărilor relației este la fel cu cea a intrărilor indexul este tip cluster
- Alternativa 1 implică clusterizare
- O tabelă poate fi clusterizată după o singură cheie
- Costul extragerii informației depinde dacă indexul este tip cluster sau nu (+1 citire spre tabelă)

Index – alternative pentru noduri

1. Sunt trei alternative

- Chiar înregistrarea cu cheia k
- $\langle k, rid \rangle$ al înregistrării cu cheia k
- $\langle k, \text{listă de } rid \rangle$ al înregistrărilor cu cheia k

2. Oricare alternativă este posibilă pentru oricare din tehniciile de indexare tip arbore B+, structură hashing etc.

- Pe lângă date și legături cu tabela indexată intrările vor conține și informații pentru direcționarea căutării

Index tip cluster

1. Prima alternativă reprezintă un index tip cluster

- Fișierul index reprezintă doar o formă de organizare a datelor din tabelă (tabelă = index)

2. Cel mult un index de tip cluster poate fi creat pentru o tabelă (altfel datele sunt duplicate integral)

- Trebuie ales cu grijă pentru că poate reduce timpul de căutare cu 20% - 66%

Alternative pentru informația din nod

1. Alternativele 2 și 3:

- Cheile de obicei mult mai mici decât înregistrările din tabelă. Deci în avantaj față de Alternativa 1 dacă înregistrările sunt mari și cheia este redusă
- Portiunea structurii indexului utilizată pentru a direcționa căutarea, care depinde de dimensiunea nodului, este mult mai mică decât în cazul Alternativei 1
- Alternativa 3 mai compactă decât Alternativa 2, dar duce la noduri de dimensiuni variabile chiar dacă cheile de căutare sunt de lungime fixă

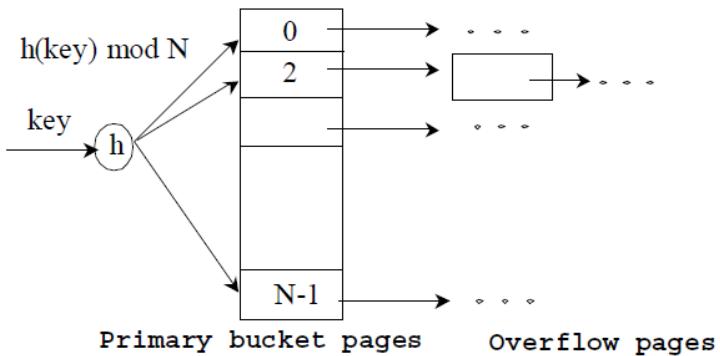
Indecși bazați pe hashing

1. Cheile sunt uniform distribuite în grupuri denumite buckets cu ajutorul unei funcții de hashing

- Bucket este o unitate de stocare ce poate conține 0..N înregistrări
- Funcție de hashing sau funcție de dispersie face o mapare a unor date la un interval numeric întreg
- Cheile care au aceeași valoare de hash sunt stocate în același bucket. Dacă nu mai încap se creează pagini adiționale legate într-o listă (overflow pages) pentru respectivul bucket

Hashing static

1. Numărul N de bucket-uri este fix, se creează pagini adiționale când este cazul
2. $h(k) \bmod N$ - va indica către înregistrarea care are cheia k (N este numărul de bucket-uri)



Hashing static - probleme

1. Bucket-ele conțin nodurile din index
2. Funcția hash trebuie să distribuie valorile în intervalul 0..N-1: $h(k) = (a*k + b)$ este obținută prin optimizarea constantelor a, b
3. Prin adăugarea de date se pot dezvolta lanțuri lungi de pagini adiționale ceea va degrada semnificativ performanța căutării
4. Rezolvarea problemei: tehnici de hashing dinamic precum hash extensibil sau liniar

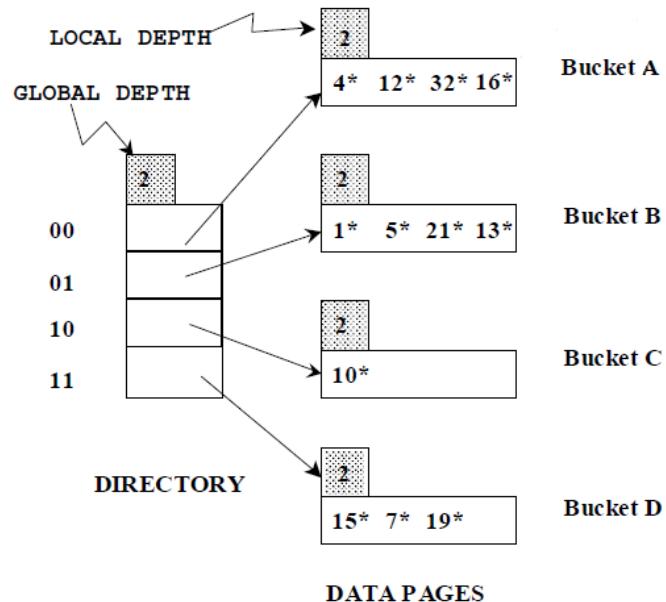
Hashing extensibil

1. Situație: bucket-ul (pagina principală) devine plină. De ce să nu reorganizăm indexul dublând numărul de bucket-uri?

- Citirea/scrierea tuturor paginilor este costisitoare
- Modificare: se utilizează un director de pointeri la bucket-uri. Se dublează virtual capacitatea indexului prin dublarea directorului, rupând doar bucket-urile pline, care necesită pagini adiționale
- Directorul ocupă puțin loc comparativ cu bucketurile, dublarea este eficientă. Evită paginile adiționale. Pentru acces se ajustează funcția hash

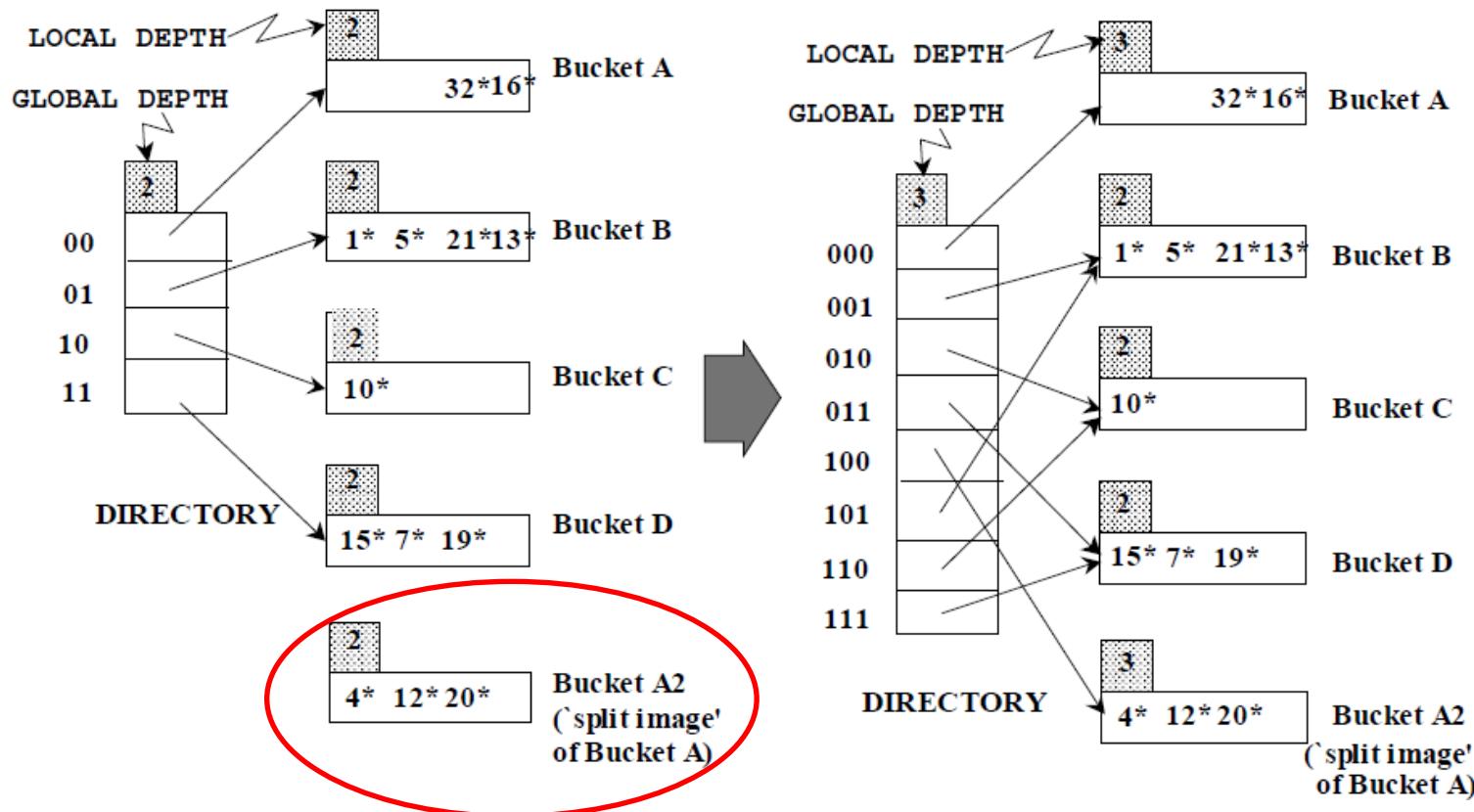
Hashing extensibil - implementare

- Directorul este un tablou de 4 intrări
- Pentru a găsi bucket-ul r , se ia numărul de biți din $h(r)$ indicați de adâncimea globală (global depth)
- Dacă $h(r) = 5$ (binar 101), atunci este bucket-ul indicat de intrarea 01
 - Inserare: dacă bucket-ul este plin se divide (se alocă o nouă pagină și se redistribuie cheile)
 - Dacă este necesar se va dubla directorul (doar dacă adâncimea locală este egală cu cea globală)



Hashing extensibil - exemplu

- Inserarea $h(r)=20$ (cauzează dublarea)



Hashing extensibil - inserarea

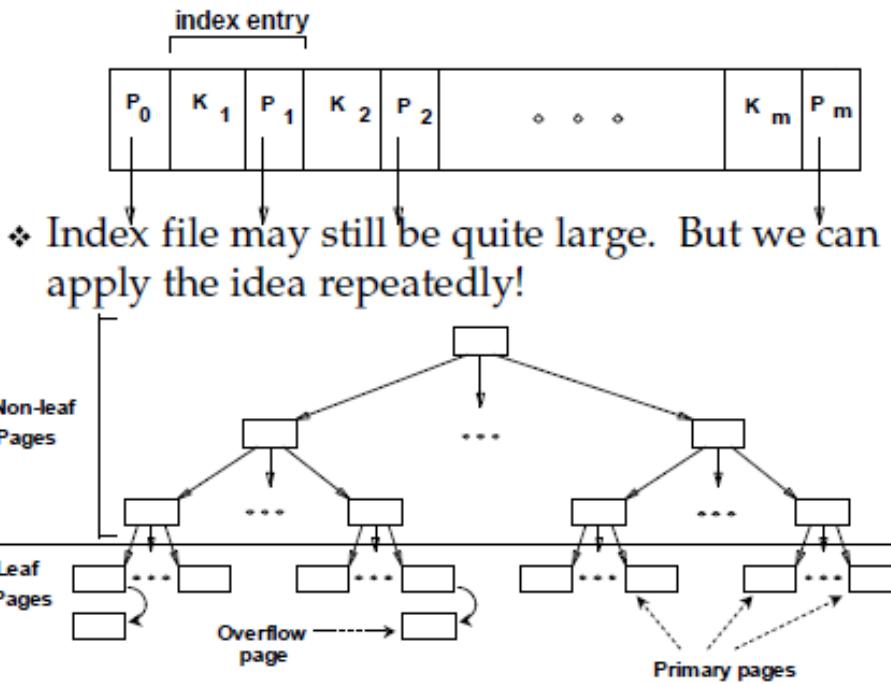
1. **Global depth** (al directorului): numărul maxim de biți necesar pentru a identifica bucket-ul în care ajunge cheia
 2. **Local depth** (al bucket-ului): numărul de biți utilizat pentru a determina dacă cheia aparține de bucket
- Când diviziunea bucket-ului determină dublarea?
 - Înainte de inserare, local depth = global depth. Inserarea cauzează local depth să devină > global depth; directorul se dublează prin copierea pointerilor existenți și ajustarea celor spre bucket-ul divizat.

Hashing extensibil - concluzii

- Dacă directorul încape în memorie, egalitatea testată print-un singur acces la disc; dacă nu prin două. Media este ~ 1.2
 - Fișier de 100MB, 100 octeți/înregistrare, 4K pagini conținând 1M înregistrări și 25K intrări în director; în majoritatea cazurilor va încăpea în memorie
 - Directorul crește în salturi, și există șansa să crească neuniform (același bucket se tot divide)
 - Cheile cu valori repetitive cauzează probleme!
- **Ștergere:** Dacă ștergerea unei intrări lasă un bucket gol acesta poate fi reunit cu perechea sa. Dacă fiecare intrarea arată spre același bucket ca și perechea, se poate înjumătăți directorul

İndeksli ISAM

- Indexed Sequential Access Method – ISAM (ex. Berkeley DB, Paradox, MySQL, Ms Access, dBase)

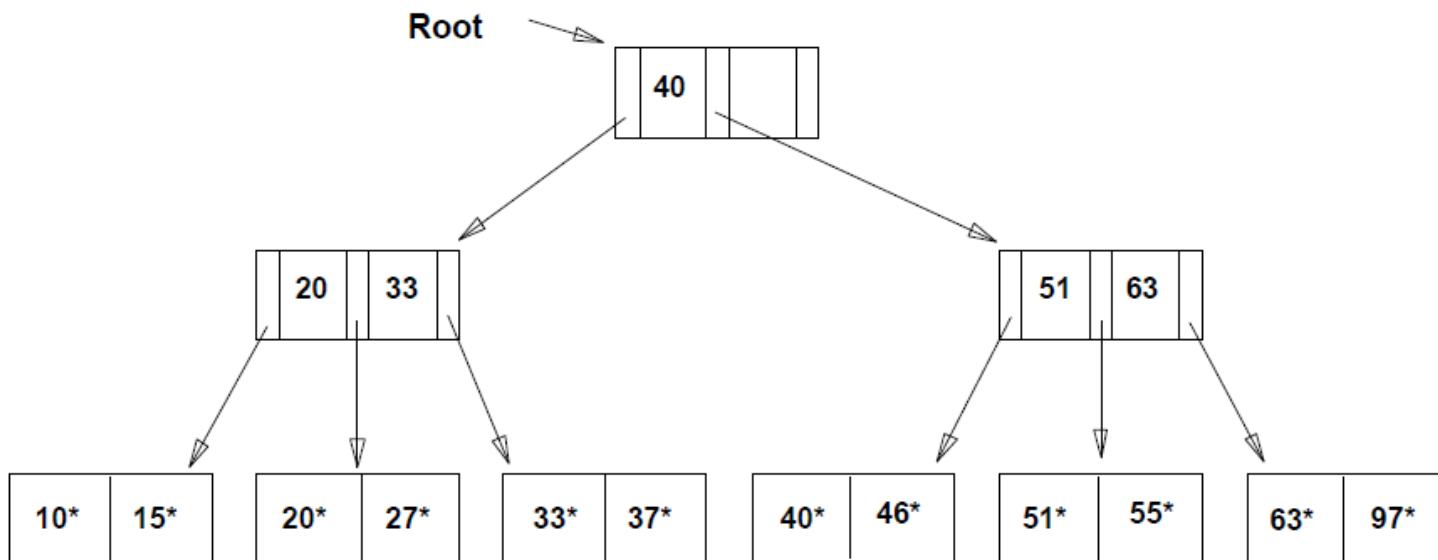


ISAM

- 1. Creare fișier:** frunzele (pagini cu înregistrări) sunt alocate secvențial și sortate după cheia de indexare; apoi nodurile index și spațiu pentru pagini adiționale
 - 2. Noduri index:** <*valoare cheie, id pagină*>; direcționează căutarea către frunzele cu înregistrări
 - 3. Căutarea:** pornește de la rădăcină; merge pe comparație. Cost: $\log_F(N)$; $F=\text{nr intrări}$, $N=\text{nr frunze}$
 - 4. Inserare:** caută frunza și adaugă înregistrarea
 - 5. Ștergere:** caută frunza și șterge înregistrarea; dacă se golește o pagină adițională se va dealoca
- * *Structura de arbore este statică*: inserarea/ștergerea afectează doar nodurile frunză

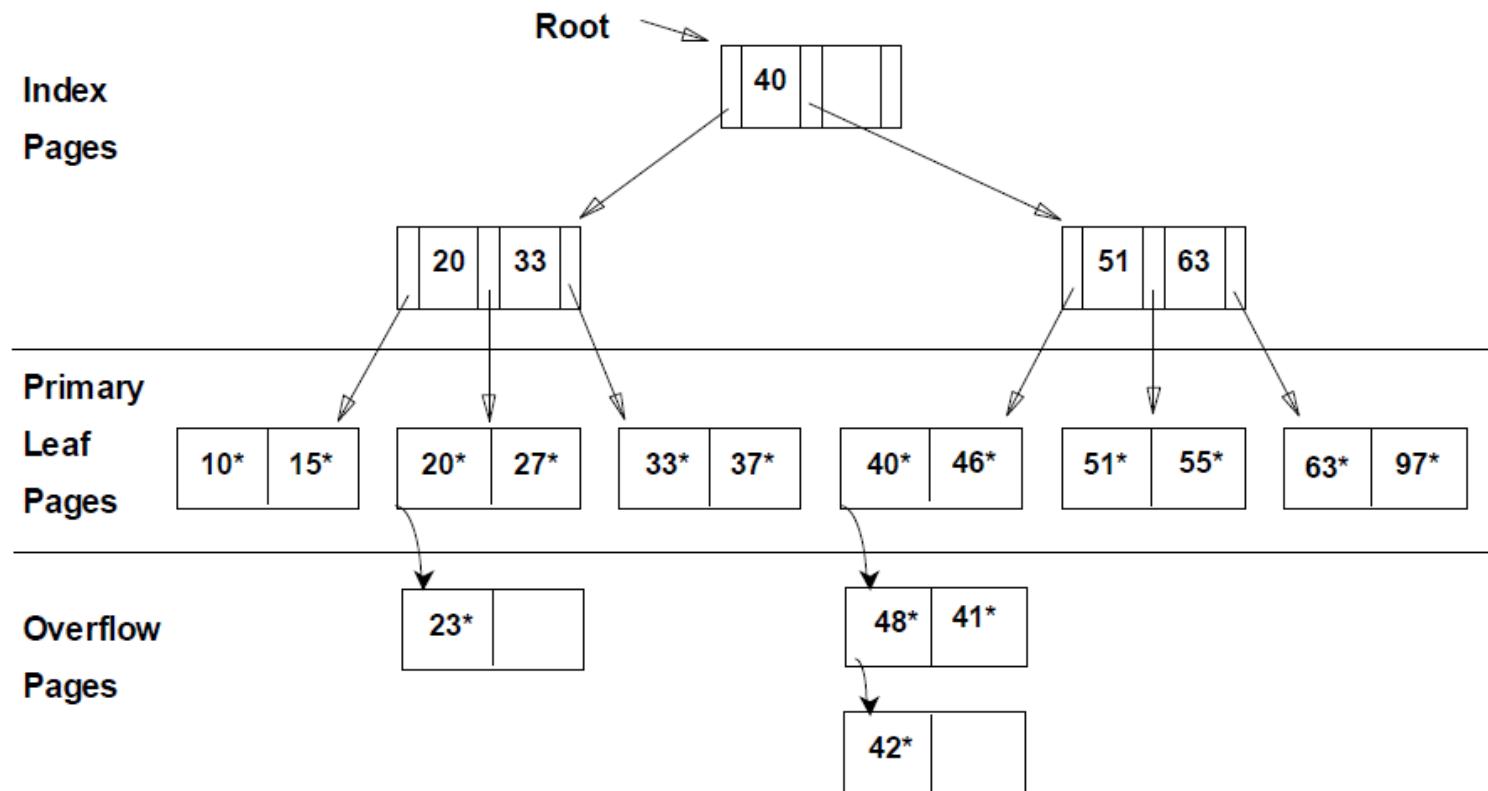
ISAM - exemplu

- Ex: fiecare nod are 2 intrări



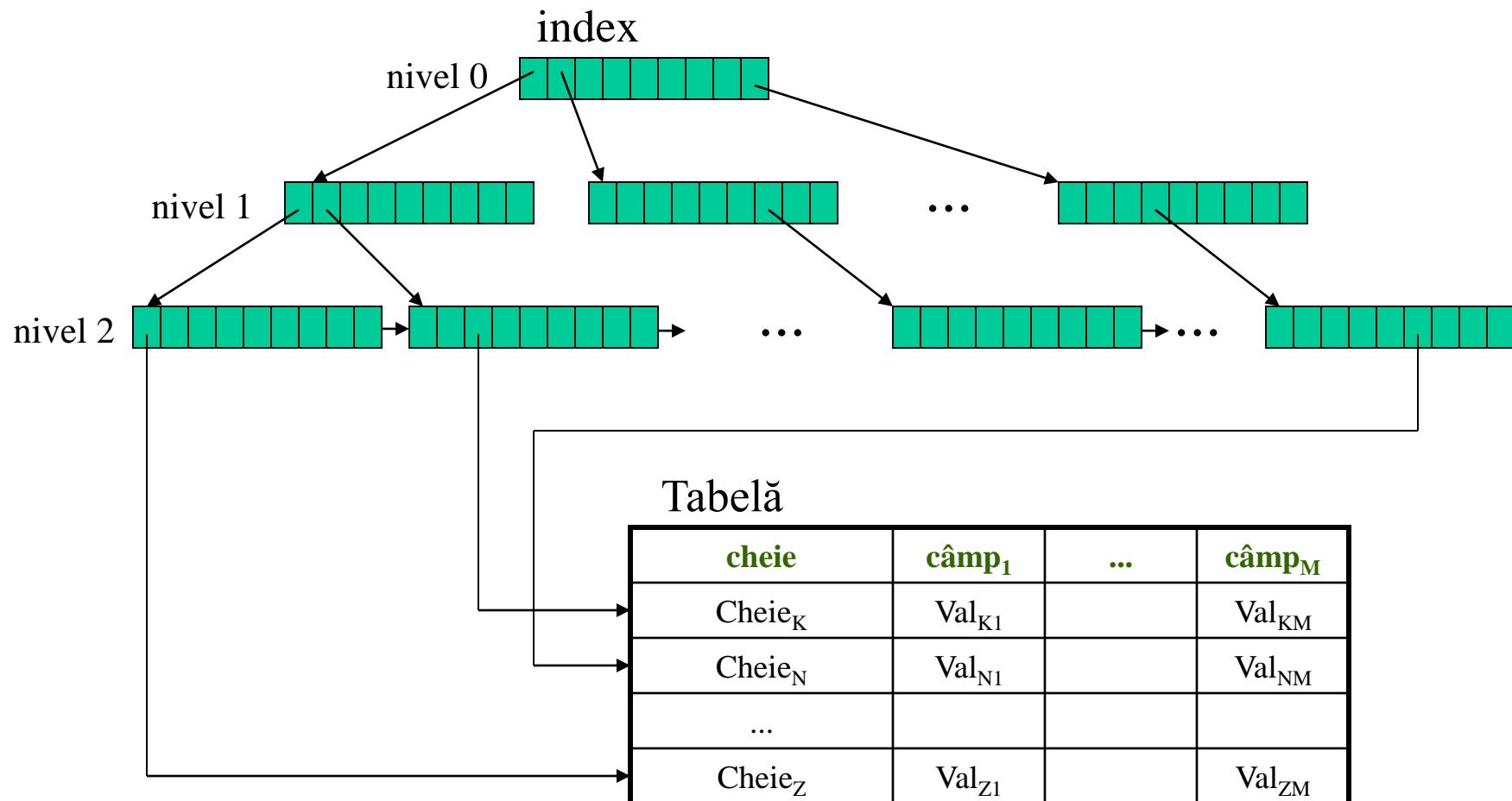
ISAM - exemplu

- După inserarea cheilor 23*, 48*, 41*, 42*



Indecsi tip arbore B+

- Cei mai des utilizati în practică



Indecsi B+

- Cost inserare/ștergere $\log_F(N)$; păstrează înăltimea balansată între subarbori
 - (F = încărcare noduri - fanout, N = nr frunze)
- Minim 50% ocupare nod (cu excepția rădăcinii). Fiecare nod conține $d \leq m \leq 2x d$ intrări, unde parametrul d este numit ordinul arborelui
- Asigură eficiență la căutări de egalități/inegalități
- Încărcare noduri ridicată (F : nr de copii/nod) ce asigură adâncime în jur de 3 sau 4
- Față de hashing permite căutarea de intervale (bazate pe comparații de inegalitate $<$, $>$)

Arbore B+ - inserare date

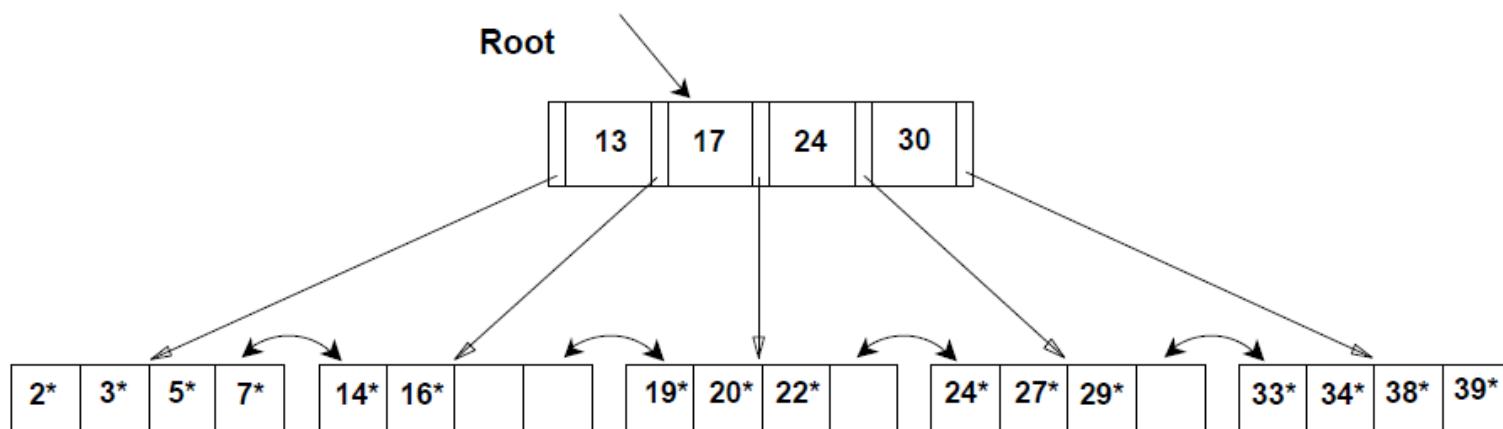
1. Caută frunza destinație L
2. Scrie cheia în L
 - Dacă L are spațiu suficient, gata!
 - Dacă nu, trebuie divizat L (în L și un nou nod L')
 - Redistribuie intrările
 - Inserează o intrare în părintele lui L care arată spre $L2$
3. Acest lucru se poate întâmpla recursiv (în sus)
 - Pentru a diviza un nod index se redistribuie cheile și se împinge în sus cheia mediană
4. Divizările măresc arborele – divizarea rădăcinii crește adâncimea arborelui
 - Creștere arbore: pe lățime sau un nivel pe adâncime

Arbore B+ - ștergere date

1. Pornește din rădăcină, caută frunza L cu acea cheie
2. Șterge intrarea din L
 - Dacă L este mai mult de jumătate plină, gata!
 - dacă L are doar **d-1** intrări,
 - Încearcă redistribuirea, împrumută de la frați (noduri cu același părinte ca și L)
 - Dacă nu se poate, reunește L cu un frate al său
3. Dacă se reunesc noduri, trebuie ștearsă referința din părintele L
4. Când se ajunge la rădăcină, scade adâncimea

Arbore B+ - exemplu

- Căutarea pornește din rădăcină, și se merge spre frunze (ca la ISAM)
 - Ex. de căutări: 5*, 15*, chei \geq 24*



Arbore B+ în practică

1. Ordin tipic: 100. Ocupare medie: 67%.
Fanout mediu = 133

2. Capacități tipice:

- Înălțime 4: $133^4 = 312,900,700$ înregistrări
- Înălțime 3: $133^3 = 2,352,637$ înregistrări

3. Se pot păstra nivelele superioare în memorie:

- Nivel 1 = 1 pagină = 8 KB
- Nivel 2 = 133 pagini = 1 MB
- Nivel 3 = 17,689 pagini = 133 MB

Posibilități indexare

- Dacă considerăm o tabelă Angajați putem avea:
 1. Fișier simplu (ordine aleatoare, inserare la sfârșit)
 2. Fișier sortat, ex. după < vârstă, salar >
 3. Index B+ tip cluster, cheie < vârstă, salar >
 4. Fișier simplu și un index B + cu cheia < vârstă, salar >
 5. Fișier simplu și un index hash cu cheia < vârstă, salar >

Proiectarea indecșilor

1. Care indecsi sunt potriviti?

- Care tabele trebuie indexate?
- Care sunt cheile necesare?
- Cati indecsi sunt utili?

2. Pentru fiecare index ce tip trebuie folosit?

- Simplu sau cluster?
- Hash/B+?

Sugestii de proiectare

1. O abordare bună: se pornește de la cele mai folosite interogări. Se consideră cel mai bun plan cu indecsii existenți apoi se caută un dacă e posibil un plan mai bun cu indecsii adiționali. Dacă da, se vor crea și aceștia
2. Înainte de crearea unui index se va studia impactul asupra inserării/ștergerii datelor!
 - Compromis: indecsii accelerează căutările dar încetinesc modificările.
 - Se va considera cerințele de spațiu suplimentar

Criterii de selecție (I)

- Atributele din WHERE sunt primele candidate pentru cheile de indexare
 - Condițiile de egalitate sugerează indecsări de tip hash
 - Extragerea de intervale sugerează B+
 - Clusterizarea este bună în special pentru extragerea de intervale; poate însă ajuta la testarea de egalitate dacă sunt multe duplicate ale cheii de căutare

Criterii de selecție (II)

- Cheile multi-atribut se vor considera dacă WHERE conține condiții multiple (AND, OR)
 - Ordinea atributelor în cheie este importantă la extragerea de intervale
 - Astfel de indecsări pot favoriza strategii de execuție tip index-only pentru interogările cele mai frecvente
- Se vor alege indecsări care acoperă cele mai multe interogări. Deoarece un singur index poate fi clusterizat per tabelă, alegerea lui va ține cont de cea mai interogată care implică acea tabelă

Gestionare indecsi în xBase

1. Creare

- INDEX ON cheie TO index_file[.ndx]
[UNIQUE][DESCENDING]

2. Deschidere

- USE file INDEX ndx_file_list
- SET INDEX TO ndx_file

3. Schimbare index activ

- SET ORDER TO ndx_file

4. Căutare folosind index

- SEEK expr, FIND valoare

Creare indecsi în Oracle

- Indecșii sunt utilizati în mod automat de optimizor și interpretorul de SQL
- Pentru cheia primară se creează implicit
- Creare explicită a unor indecsi:

```
CREATE INDEX [UNIQUE] nume_index  
    ON tabelă(câmp1 ASC, câmp2 DESC ...)  
    [COMPUTE STATISTICS]; // utilizeaza optimizor
```

```
DROP INDEX nume_index;
```

- Pentru a crea indecsi clusterizați se va folosi **CREATE TABLE** cu opțiunea **ORGANIZATION INDEX INCLUDING cheie** (la final, după paranteză închisă)

Baze de Date

Cap. 10. Optimizarea și execuția interogărilor SQL.



Textbook: Ramakrishnan, Gehrke, "Database Management Systems", McGraw Hill, C15

2023 UPT

Conf.Dr. Dan Pescaru

Operatori relaționali

1. O interogare SQL se traduce într-o expresie utilizând operatorii relaționali
2. Expresia se reprezintă ca și un arbore (plan) de execuție care are operatori în noduri și tabele în frunze
3. Atât intrările (frunzele) cât și rezultatul evaluării planului sunt relații (tabele)
4. Schema rezultatului este fixă și se deduce din lista de proiecție

Operatori relaționali

1. Operatori de bază:

- Proiecție (Π) – selectează coloanele specificate
- Selectie (σ) – selectează înregistrările specificate
- Produs Cartezian (\times) – combină două relații
- Diferență mulțimi (\setminus) – înregistrările din R1, care nu \in R2
- Reuniune (\cup) – înregistrările din R1 și R2 (fără duplicate)

2. Operatori adiționali:

- JOIN (\bullet), Intersecție (\cap) , Divizare (\div), Redenumire

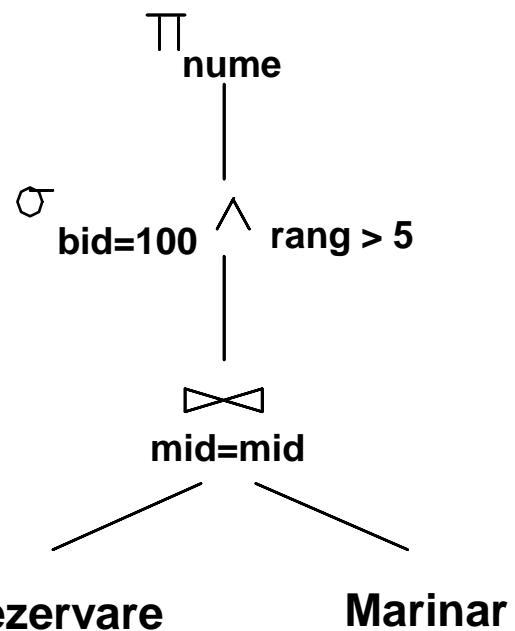
3. Din moment ce orice operator returnează o relație, ei pot fi compuși în expresii (Algebra relațională este "închisă")

Evaluarea interogării

1. Plan de execuție = arborele operatorilor din algebră relațională + algoritmii selectați pentru evaluarea lor
2. Două probleme principale în optimizarea interogărilor:
 1. Pentru o interogare, ce planuri sunt considerate?
 2. Care este cel mai bun și cum este estimat costul?
3. Ideal: găsirea cel mai bun plan
4. Practic: evitarea celor mai slabe!

Exemplu de plan de execuție

- Ex. **SELECT nume FROM Marinar m
INNER JOIN Rezervare r ON m.mid=r.mid
WHERE r.bid=100 AND m.rang>5**



Tehnici de optimizare de bază

1. Algoritmii de evaluare a operatorilor relaționali folosesc câteva idei simple
2. Iterația: uneori, mai rapid este a scana toate înregistrările din tabel chiar dacă există un index
3. Uneori, mai rapid este a scana intrările de date într-un index în loc de cele din tabel
 - Indexare: se poate folosi condițiile WHERE pentru a extrage un set mic de înregistrări (σ, \circ)
 - Partitionare: prin sortare sau hashing, putem partitura înregistrările de intrare și să înlocuim o operație costisitoare cu operații similare pe intrări mai mici

Catalog și statistici

1. Catalogele conțin meta-date și statistici:

- # tupluri (NTuples) și # pagini (NPages) pentru fiecare tabelă
- # chei distincte (NKeys) și NPages pentru fiecare index
- Înălțime index, valorile cele mai mici/mari ale cheilor pentru fiecare index arbore B+

2. Cataloge actualizate periodic (pentru eficiență)

3. Actualizarea ori de câte ori se modifică datele este prea costisitoare; oricum sunt multe aproximări, aşa că o ușoară inconsecvență este acceptabilă

Estimarea costului

1. Pentru fiecare plan de execuție luat în considerare, trebuie estimat costul fiecărei operații
 - Depinde de cardinalitățile de intrare
 - Depinde de dimensiunea tuplului
2. De asemenea, trebuie să estimeze dimensiunea rezultatului pentru fiecare operator din arbore!
 - Se utilizează informațiile despre relațiile de intrare
 - Pentru selecții și join-uri, se presupune independența predicilor

Căi de acces

1. O cale de acces este o metodă de extragere a tuplurilor din BD
 - Scanare fișier, sau un index dacă se potrivește cu o selecție (din interogare)
2. Un index arbore B+ se poate folosi cu o conjuncție de termeni care implică atributăe încr-un prefix al cheii
 - Ex. un index cu cheia $\langle a, b, c \rangle$ se poate utiliza cu selecția "a=5 AND b=3 și a=5 AND b>6", dar nu "b=3"
2. Un index hash se potrivește cu o conjuncție de termeni atribut = valoare pentru fiecare atribut din cheia de căutare a indexului
 - Ex. un index Hash cu $\langle a, b, c \rangle$ se potrivește cu a=5 și b=3 și c=5; dar nu se potrivește cu b=3 sau a>5 și b=3 și c=5

Optimizarea SELECTIEI

1. Se caută calea de acces cea mai selectivă, se extrag înregistrările folosind-o și apoi se aplică termenii rămași care nu se găsesc în index:

- Calea de acces cea mai selectivă: un index sau o scanare a fișierelor despre care estimăm că va necesita cele mai puține operații I/O de accesare pagini pe disc
- Termenii care se potrivesc cu indexul reduc numărul de tupluri extrase; ceilalți termeni sunt folosiți apoi pentru a elimina unele din înregistrările extrase, dar aceasta nu afectează numărul de înregistrări/pagini extrase de pe disc

Optimizarea SELECȚIEI - exemplu

1. Se consideră condiția

$zi < (1/9/23)$ AND $bid = 5$ AND $mid = 31$

2. Un index arbore B+ cu cheia $<zi>$ poate fi utilizat

- $bid = 5$ AND $mid = 31$ trebuie apoi verificate pentru fiecare înregistrare extrasă

3. Similar se poate folosi un index hash cu cheia $<bid, mid>$

- Condiția $zi < (1/9/23)$ trebuie apoi verificată pentru fiecare înregistrare extrasă

Optimizarea PROIECTIEI

1. Partea cea mai costisitoare este înlăturarea duplițatelor:

- Din cauza aceasta interprotoarele de SQL nu înlătură duplițatele decât dacă se specifică **DISTINCT** în interogare
- Se poate implementa prin sortare. Poate fi optimizată și să extragă doar câmpurile proiectate în timpul sortării
- Se poate implementa și prin hash. Se creează partiții folosind o funcție de hash. Se încarcă fiecare partiție în memorie creându-se încă un nivel de partiții pentru a se testa duplițatele

Optimizarea PROIECȚIEI prin sortare

SELECT DISTINCT mid, bid FROM Rezervare

1. O rezolvare prin sortare:

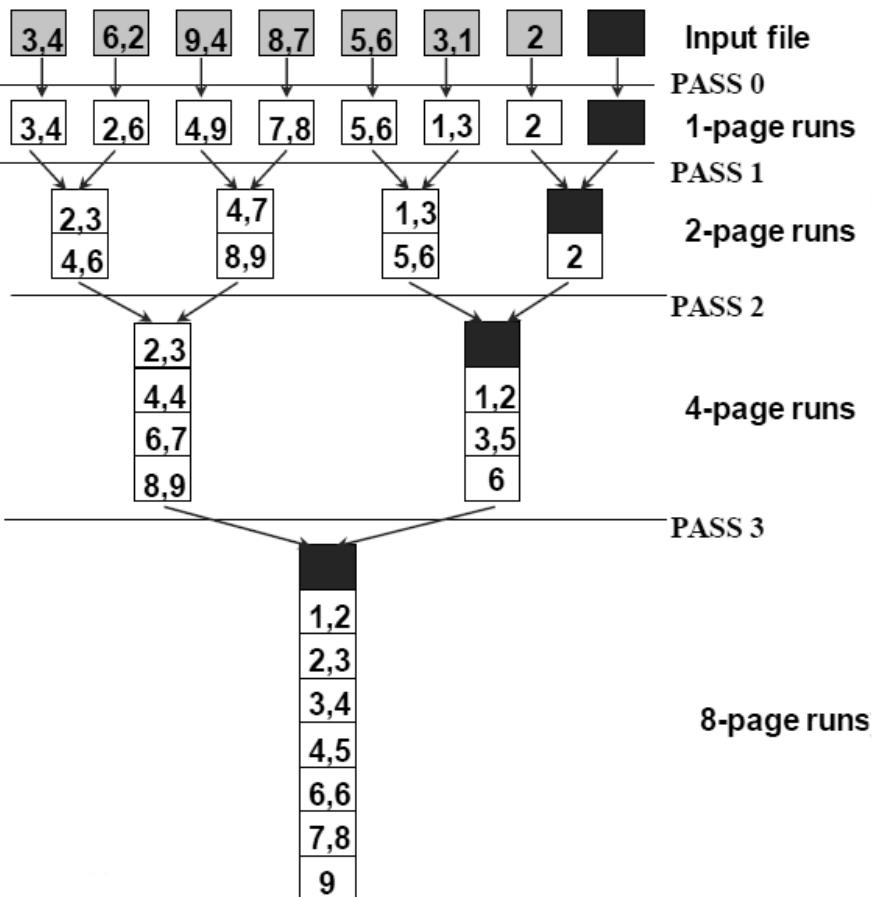
- Sortare-îmbinare (merge sort) externă pe două căi cu eliminarea câmpurilor care nu sunt în lisat de proiecție
- Tuplurile din rezultat sunt mai mici decât cele de intrare (rația depinde de nr și dimensiunea câmpurilor care sunt eliminate)
- Se va modifica pasul de îmbinare pentru a elmina duplicatele rezultând un număr de tupluri mai mic decât la intrare cu nr. de duplicate

Algoritmul de sortare-îmbinare extrenă

1. Idea de bază:
sortarea și îmbinarea
fișierelor partiție
(utilizând divide and
conquer), prin trei
bufere

2. La fiecare pas:
citere + scrie
fiecare pagină

3. Cost $\sim 2N \times \log_2 N$



Optimizarea PROIECȚIEI prin hashing

SELECT DISTINCT mid, bid FROM Rezervare

1. Faza de partitioare utilizând B buffere: se citește R utilizând un buffer. Pentru fiecare tuplu, fără attribute neproiectate, se aplică funcția hash $h1$ pentru a alege unul dintre cele B-1 buffere de ieșire
 - Rezultă B-1 partiții (cu tupluri doar cu attribute proiectate). Două tupluri din partiții diferite nu sunt sigur duplicate
2. Eliminarea dupliilor: pentru fiecare partiție se creează o tabelă hash în memorie utilizând o funcție hash $h2$ ($<> h1$) pe toate câmpurile, iar dupliile se elimină prin testarea egalității funcției hash

Optimizarea JOIN prin Index Nested Loop

```
SELECT r.*, m.* FROM Rezervare r, Marinar m  
WHERE r.mid=m.mid
```

1. Fără index: poate să dureze ore de execuție pentru >1M rezervări și câteva mii de marinari!
2. Dacă există un index după coloana de join pentru o tabelă, se face folosește pentru căutare (ciclul intern)
 - Cost: $\text{NrÎnreg}_{R1} \times \text{cost căutare corespondentă în } R2$
3. Pentru fiecare tuplu din R1, costul căutării în indexul R2 este ~ 1.2 pentru hash (< 5 min), și 2-4 pentru index B+ (8-15 min). Costul depinde și de clusterizare
4. Index clusterizat: + 0 operații de I/O
5. Neclusterizat: + 1 operație de I/O per tuplu găsit

Optimizarea operatorilor pe mulțimi

1. Intersecția și produsul cartezian sunt asemănătoare cu join
2. Reuniunea prin sortare:
 - Se sortează ambele relații (după combinația tuturor atributelor)
 - Îmbinarea relațiilor sortate (algoritm fermoar)
3. Reuniunea prin hash:
 - Se partăionează R1 și R2 cu o funcție hash $h1$
 - Pentru fiecare partitie a lui R2 se va crea o tabelă de hash în memorie (utilizând o funcție $h2$), se scanăză partitiile corelate din R1 și se salvează tuplurile eliminându-se și duplicatele

Optimizarea agregării

1. Fără grupare:

- În general necesită scanarea tabelelor
- Dacă există un index a cărui cheie conține toate atributele referite în **SELECT** și **WHERE**, se poate scana doar indexul fără a se accesa tabela

Optimizarea agregării cu grupare

1. Cu grupare:

- Se sortează în funcție de attributele grupului, apoi se scanează relația și se calculează funcția de agregare pentru fiecare grup. (Se poate îmbunătăți acest lucru combinând sortarea și calculul agregării)
- Abordare similară bazată pe hashing pe attributele de grupare
- Dacă există un index B+cu cheia care să includă toate attributele din **SELECT**, **WHERE** și **GROUP BY**, se poate scana doar indexul; dacă attributele de grupare sunt în prefixul cheii indexului, se pot extrage tuplurile în ordinea grupării și aplica **HAVING** în același timp

Optimizare - exemplu

