Capitolul 2: Baze de date relaționale

- Relații, atribute, domenii; schema relației
- Reprezentarea relațiilor prin tabele
- Limbajul SQL:
  - Convenții lexicale
  - Expresii, operatori, functii
  - Instructiuni de definire a datelor: CREATE, ALTER, DROP
  - Instructiuni de manipulare a datelor: SELECT, INSERT, UPDATE, DELETE
- Constrângerile de integritate ale relațiilor
  - Constrângeri de domeniu
  - Constrângeri de tuplu: cheia primară – chei secundare
  - Constrângeri de integritate referențială – chei străine
- Indexarea relațiilor
  - Indexul primar
  - Indexuri secundare

Relații – Atribute – Domenii

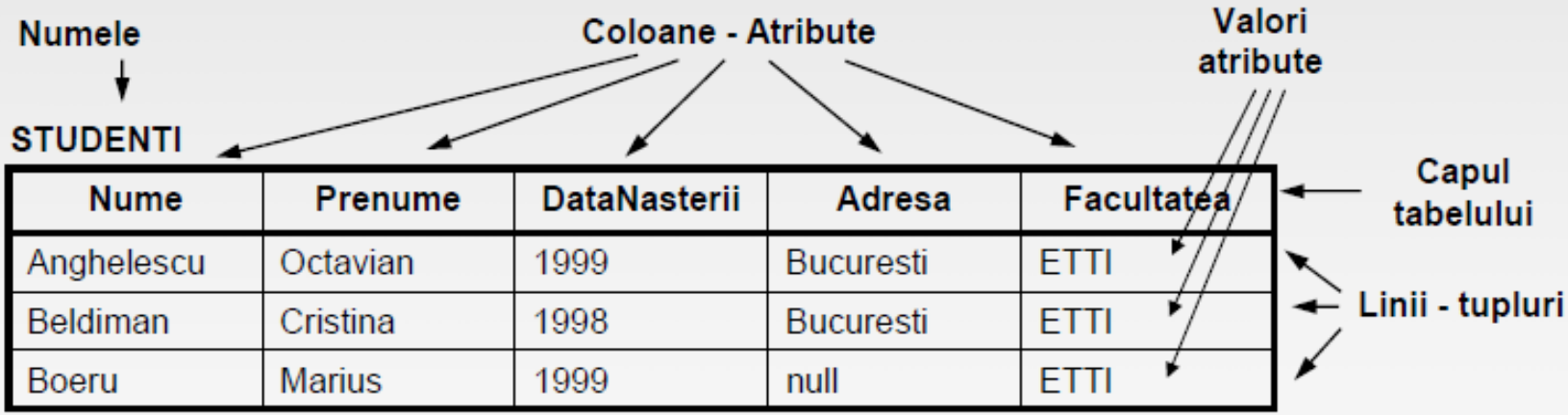
- Modelul relațional: E.F.Codd, 1970 – IBM
- ***O bază de date relațională*** este compusă dintr-o mulțime finită de relații
  - fiecare relație reprezinta o mulțime (tip) de entitati sau o mulțime (tip) de asocieri
  - fiecare relație este unica intr-o baza de date
  - o relație se defineste prin intermediul atributelor sale
- ***Atributele*** unei relații corespund atributelor tipului de entitate sau de asociere pe care îl reprezintă relația respectivă
  - fiecare atribut are un nume (Ai) și un domeniu de definiție D(Ai)
  - pentru o entitate data, un atribut poate lua o singură valoare (scalar)
- Atributele pot fi: simple (un element) sau compuse (o submulțime de atribute)
- **Domeniu**: o mulțime de valori D = {di | i = 1,…, n }, definit printr-o specificare de tip, unde:
  - D este numele domeniului
  - di este un element al domeniului care satisface anumite constrângeri
  - Elementele domeniilor sunt atomice (indivizibile)
  - O valoare speciala, null, poate apartine oricarui domeniu (inseamna lipsa de informatie sau valoare necunoscuta)

Schema relației

- Schema relației: descriere a unei relații (tipul, intensiunea relației)
- Schema relației: R(A1,A2,...Ai,...An), unde:
  - **R** este numele schemei relației
  - lista ordonată a atributelor sale **A1,A2,...Ai,..An**
  - fiecare atribut **Ai** definit pe domeniul său de definiție, **D(Ai)**
  - Gradul relației: numărul de atribute ale schemei acelei relații (n)
  - Exemplu: **STUDENTI (Nume, Prenume, DataNasterii, Adresa, Facultatea)**
- O relație r definita prin schema R(A1,A2,...Ai,...An) este:
  - o mulțime finita de n-tupluri **t**
  - tuplul **t** este o listă ordonată de n valori: **t = <v1,v2,...vi,...vn>**, unde **1 ≤ i ≤ n**
  - **vi** este o valoare a atributului **Ai**, **vi ∈ D(Ai)**
- Relația r(R): r este variabila, instanta a schemei (tipului) R
  - Valoarea variabilei: starea sau extensiunea relației
  - Numarul de tupluri ale unei relații: cardinalitatea relației
  - Fiecare tuplu este unic intr-o relație (nu exista tupluri duplicat)
  - Corespondenta: relație→mulțime de entitati (sau de asocieri); tuplu → entitate
- În mod curent: se foloseste R atat pentru schema cat și pentru relația insasi

Reprezentarea relațiilor prin tabele

- ***Un tabel*** (*table*) = reprezentarea grafică a unei relații; compusă din:
  - Numele tabelului - identic cu numele relației
  - Coloanele corespund atributelor relației
  - Capul tabelului- contine numele atributelor (coloanelor) → schema relației
  - O mulțime de linii, fiecare linie corespunzând unui tuplu→ starea relației
  - Valori ale atributelor fiecarui tuplu
- Exemplu: Tabelul care reprezinta relația (starea relației) STUDENTI

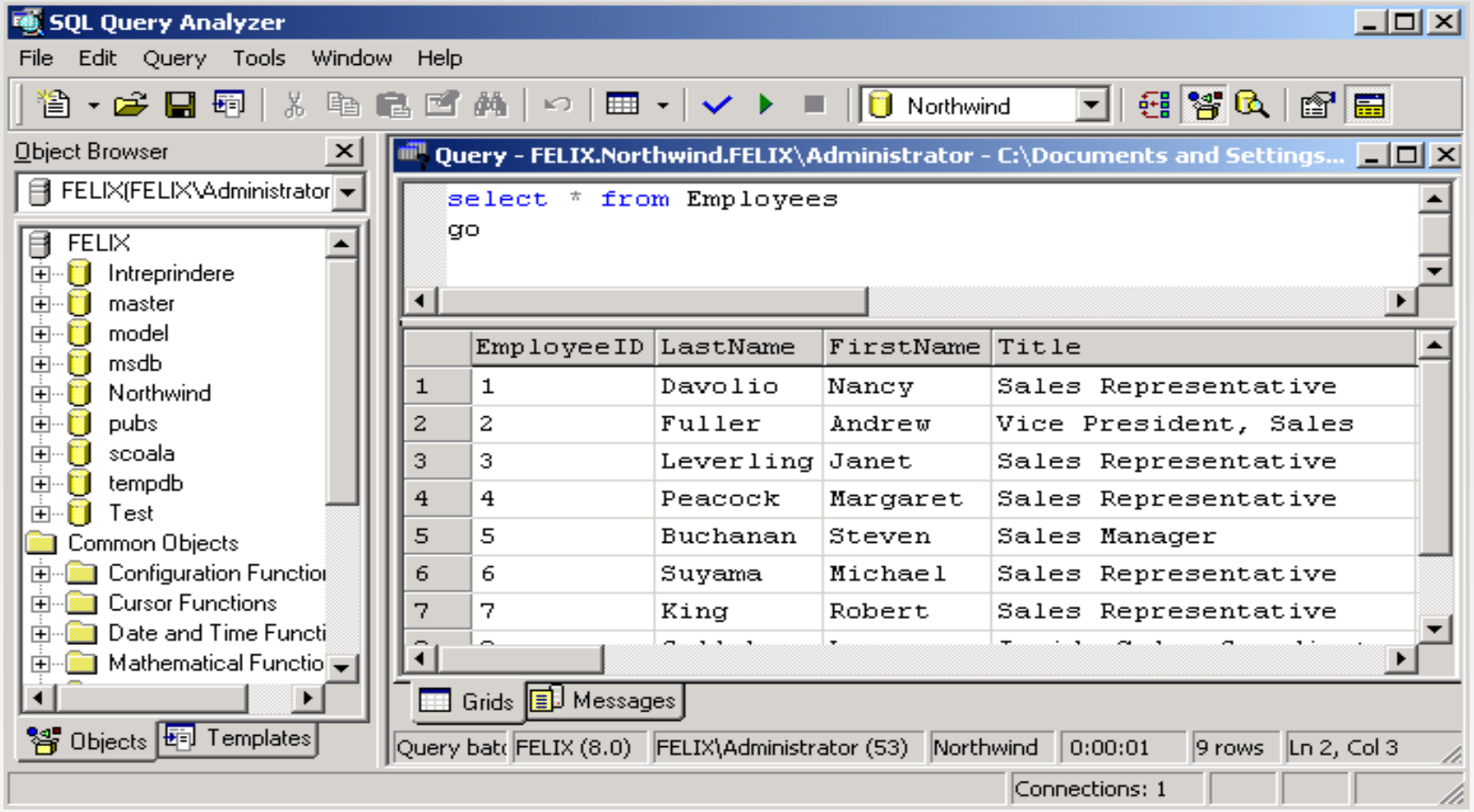


Nume	Prenume	DataNasterii	Adresa	Facultatea
Anghelescu	Octavian	1999	Bucuresti	ETTI
Beldiman	Cristina	1998	Bucuresti	ETTI
Boeru	Marius	1999	null	ETTI

- Tabelul sugerează ordonarea atributelor (coloanelor) și a tuplurilor (liniilor) –ceea ce nu corespunde modelului matematic (relație = mulțime de tupluri)

Afișarea tabelelor

- SGBD-urile oferă instrumnente de proiectare și afisare a tabelelor
  - De exemplu, afișarea tabelului Employees din baza de date Northwind folosind toolset-ul SQL Query Analyser din Microsoft SQL Server



Ordonarea valorilor atributelor în tupluri

- Din punct de vedere logic, ordinea valorilor atributelor într-un tuplu nu conteaza; această structurare poate fi exprimată prin următoarele definiții:
- Schema relației: R = {A1,A2, ...Ai,...An} (o mulțime de atribute)
- Relația r(R): o mulțime de n-tupluri t, unde:
  - fiecare tuplu t este o mulțime de n perechi ordonate **<Ai,vi>,** unde 1 ≤ i ≤ n,
  - **t = {<A1,v1>,<A2,v2>,...<Ai,vi>, ...<An,vn>}**
  - vi este valoarea atributului Ai, **vi ∈ D(Ai)**
- Observații asupra celor două definiții:
  - A doua definiție a relației este mult mai generală decat prima definitie
  - Prima definiție simplifică notațiile şi corespunde reprezentării prin tabel a relației şi de aceea va fi folosită în continuare destul de frecvent
  - În implementările reale, există o anumită ordine a valorilor atributelor memorate în fișiere, dar aceasta nu este relevantă din punct de vedere logic

Limbajul SQL

- Limbajul IBM Sequel dezvoltat ca parte a proiectului System R la IBM San Jose Research Laboratory (1970)
- Redenumit Structured Query Language (SQL)
- Standarde SQL - ANSI și ISO:

Anul	Denumire	Caracteristici
1986	SQL-86	Publicat de ANSI (SQL1); ratificat de ISO in 1987
1989	SQL-89	Revizii minore
1992	**SQL-92**	**Revizii majore, redenumit SQL2**
1999	**SQL-1999**	**Redenumit SQL3, adauga unele caracteristici obiect-relationale**
2003	SQL-2003	Adauga unele trasaturi referitoare la limbajul XML
2006	SQL-2006	Utilizare SQL in conjunctie cu XML

- Fiecare SGBDR implementează un dialect al limbajului SQL, ceea ce micşorează gradul de portabilitate a aplicațiilor
- În diferitele implementări ale limbajului SQL pot să lipsească unele comenzi prevăzute în standard, dar pot exista extensii specifice SGBD-ului respectiv

Caracteristicile generale ale limbajului SQL

- Limbajul SQL foloseste reprezentarea prin tabele a relațiilor, reprezentare care este mai simplă şi mai intuitivă (foloseste termenii tabel, linie, coloană)
- Limbajul SQL cuprinde:
  - Componenta de descriere a datelor (Limbaj de Descriere a Datelor – LDD)
  - Componenta de manipulare a datelor (Limbaj de Manipulare a Datelor – LMD)
  - Alte componente: controlul tranzactiilor, controlul securitatii, protectia datelor etc.
- Limbajul SQL2 este un limbaj neprocedural:
  - o instrucțiune SQL2 specifică ce informații trebuie să fie setate sau obținute, nu modul (procedura) în care se operează
  - limbajul SQL2 nu conține instrucțiuni de control al fluxului execuției (instrucțiuni ca for, while, if, etc)
- Standardul SQL3 prevede instructiuni de control și crearea de tipuri definite de utilizator, fiind implementat în SGBD-urile obiect-relaționale
- Pentru aplicațiile de baze de date, s-au dezvoltat extensii procedurale ale limbajului SQL, biblioteci şi interfețe de programare care integrează instrucțiunile SQL

Structura lexicala a limbajului SQL

- O instrucțiune SQL (*statement*) este o secvență de elemente - de regula terminată cu semnul punct şi virgulă (;)
- Fiecare instrucțiune SQL conține *o comandă SQL* (*command*), care specifică ce acțiune se efectuează, urmată de alte elemente, care specifică operații, clauze, parametri etc.
  - Exemplu: **SELECT * FROM ANGAJATI;**
- Elementele (*tokens*) instrucțiunilor SQL
  - cuvânte cheie (*key words*): **CREATE**, **INSERT**, **SELECT** , **WHERE**, **FROM** etc.
  - identificatori (*identifiers*):
    - simpli - numai caractere alfa-numerice și underscore(_): ANGAJATI, Nume, Prenume etc.
    - delimitati (*quoted*) - pot contine orice caracter, foloseste ghilimele : 'Nume', 'Prenume' etc.
  - constante (literali): 1000, 100.5, 'Ionescu', NULL
  - caractere speciale: *, ., ;
- Spațiile albe (*whitespaces*) separa elementele: spațiu, linie nouă, tab
- O instructiune se poate scrie pe una sau mai multe linii, iar într-o linie se pot introduce una sau mai multe instructiuni
- Limbajul SQL este *case-insensitive* (nu deosebeste literele mici de cele mari) cu exceptia identificatorilor delimitati (*quoted*) care sunt *case-sensitive*

Expresii și operatori în limbajul SQL

- O expresie SQL constă dintr-unul sau mai mulți operanzi, operatori şi paranteze
  - Parantezele se pot folosi pentru a preciza o anumită ordine a operațiilor, dacă aceasta este diferită de ordinea implicită data de precedenta operatorilor.
- Un operand poate fi:
  - numele unei coloane – în acest caz se foloseste valoarea memorata în acea colona intr-una sau mai multe linii ale tabelului
  - o constantă (literal)
  - valoarea returnată de o functie
- Un operator SQL este exprimat prin:
  - unul sau mai mai multe caractere speciale; exemple: +, -, *, /, %, <= etc.
  - un cuvânt cheie; exemple: AND, OR, NOT, LIKE etc.
- Operatori SQL - dupa numarul de operanzi: binari sau unari
- Operatori SQL - dupa tipul operatiei: aritmetici, de comparație SQL, logici, relaționali
  - Operatori aritmetici de operatii cu numere intregi sau reale: +, -, *, /, %, ^
  - Operatori aritmetici orientati pe biti: ~, &, |, #
  - Operatori aritmetici de comparatie: <, >, =, <> (sau !=), <=, >=
  - Operatori de comparatie SQL: IS NULL, IS NOT NULL, BETWEEN, IN, LIKE
  - Operatori relaționali: UNION, INTERSECT, MINUS

Operatorii logici SQL

- Operatorii de comparație (atat cei aritmetici cat si operatorii de comparatie SQL) returnează o valoare logica:
  - **false** (0), condiția nu este îndeplinită
  - **null –** nu se cunoaste daca condiția este indeplinita sau nu
  - **true** (1) condiția este îndeplinită
- Operatorii logici (NOT, AND, OR):
  - se aplică unor valori logice trivalente (cu 3 valori: true (1), false (0) şi null - lipsa info
  - returnează o valoare logică trivalentă

A	B	A and B	A or B	A	not A
true	true	true	true	true	false
true	false	false	true	false	true
true	null	null	true	null	null
false	false	false	false		
false	null	false	null		
null	null	null	null		

Funcții SQL predefinite

- In SQL exista:
  - Functii SQL prefedinite
  - Functii SQL definite de utilizator (se vor studia in capitolele următoare)
- Funcții SQL prefedinite: funcții agregat şi funcții scalare.
- Funcțiile agregat calculează un rezultat din mai multe linii ale unui tabel
  - Aceste funcții vor fi detaliate ulterior, la descrierea instrucțiunii SELECT
- Funcțiile scalare:
  - Primesc unul sau mai multe argumente şi returnează valoarea calculată sau NULL în caz de eroare
  - Argumentele funcțiilor pot fi constante (literale) sau valori ale atributelor specificate prin numele coloanelor corespunzatoare
- Tipuri de funcții scalare SQL:
  - Funcții de calcul trigonometric (**sin**, **cos**, **tan** etc.), funcții de calcul al logaritmului (**ln**, **log**), al puterii (**power**), funcții de rotunjire (**floor**, **ceil**), etc.
  - Funcții pentru manipularea şirurilor de caractere: **concat**, **replace**, **upper** etc.
  - Funcții pentru data calendaristică şi timp: add_months, next_day, last_day etc.
  - Funcții de conversie: **to_number**, **to_char** etc.
- Funcțiile scalare se folosesc în expresii, care pot să apară în diferite clauze ale instrucțiunilor SQL

Tipuri de date SQL (1)

- Tipuri de date SQL2: numeric, şiruri de caractere, şiruri de biți, data (calendaristică) si timp
- Tipul numeric:
  - numere întregi: **integer** sau **int** (4 octeți), **smallint** (2 octeți)
  - numere reale reprezentate în virgulă flotanta: **float** (4 octeți), **real** şi **double [precision]** (8 octeți)
  - numere zecimale reprezentate cu precizia dorită (tipul numeric sau decimal, memorate ca şir de caractere): **numeric[(p,s)]** (sau **decimal [(p,s)]**), unde p (precizia) este numărul total de cifre, iar s (scara) este numărul de cifre după punctul zecimal
- Siruri de caractere:
  - **character(n),** prescurtat, **char(n)** - şir de caractere de lungime fixă (n)
  - **character varying(n),** prescurtat **varchar(n)** - şir de caractere de lungime variabilă, maximum n
- Siruri de biți - secvențe de cifre binare (care pot lua valoarea 0 sau 1):
  - **bit(n))** - sir de biti de lungime fixă (n)
  - **bit varying(n)** sir de biti lungime variabilă, maxim n

Tipuri de date SQL (2)

- Tipurile SQL pentru data calendar şi timp sunt: date, time, timestamp, interval:
  - Tipul **date**: memorarea datelor calendaristice prin utilizarea a trei câmpuri (year, month, day), în formatul yyyy-mm-dd; se admit numai date valide
  - Tipul **time**: memorarea timpului, folosind trei câmpuri (hour, minute, second) în formatul HH:MM:SS; se admit numai valori valide
  - Tipul **timestamp(p)**: memorarea combinată a datei calendaristice şi a timpului, cu precizia p pentru câmpul second. Valoarea implicită a lui p este 6
  - Tipul **interval** este utilizat pentru memorarea intervalelor de timp
- Variante de tipuri de date SQL specifice în diferite sisteme SGBD; Exemple:
  - SGBD Microsoft SQL Server: **tinyint -** număr întreg pe 1 octet
  - SGBD Oracle: **varchar2 -** şir de caractere de lungime variabilă
- Standardul SQL2 nu suportă tipuri de date şi operații definite de utilizator
- Standardul SQL3 suportă tipuri de date şi operații definite de utilizator, care sunt caracteristice ale modelului de date obiect-relațional
- Actualmente, producătorii de sisteme de baze de date relaționale introduc treptat diferite caracteristici ale modelului obiect-relațional cuprinse în standardele SQL3 și urmatoarele

Domenii SQL

- În SQL2 domeniile atributelor se specifică pe baza tipurilor de date predefinite ale limbajului SQL, și nu corespund întru totul noțiunii de domeniu relațional
  - Noțiunea de domeniu relațional implică o anumita semantica (semnificatie) si anumite restricții; un tip de date nu are semantica, dar se pot impune anumite constrangeri; ex.: se impune ca atributul Varsta (unei persoane, într-un tabel ca ANGAJATI etc.) să nu ia valori negative sau mai mari decât, sa spunem 110 (?)
  - SQL2 prevede comanda CREATE DOMAIN, care definește un domeniu pe baza unui tip predefinit SQL2 şi cu unele constrângeri, dar nu este implem. în toate SGBD
- SQL3 (și urm.) prevede comanda CREATE TYPE care crează tipuri definite de utilizator (user-defined types), folosite ca domenii ale atributelor
- SGBD-urile actuale implementează diferite versiuni din standarde:
  - În Oracle (8i, 9i, 10g,11g, 12c) se pot crea tipuri de date noi, pot fi folosit ca domenii ale atributelo, rfolosind comanda CREATE TYPE
  - În SQL Server se pot crea domenii cu comanda SQL CREATE DOMAIN; se pot crea tipuri de date definite de utilizator ca și clase într-un limbaj .NET, se compilează ca un ansamblu, și se inregistreaza în SQL Server cu comanda CREATE ANSAMBY
  - În PostgreSQL se pot crea tipuri de date noi, folosind comanda CREATE TYPE
  - În MySQL nu se pot crea tipuri definite de utilizator (până în versiunea curenta, dar probabil se vor introduce în versiunile următoare)

Conventii de notatie

- Pentru prezentarea limbajului SQL și a altor limbaje, biblioteci și interfete se folosesc următoarele convenții

[ ] (paranteze drepte)	Element opțional al instrucțiunii
{ } (acolade)	Element obligatoriu al instrucțiunii
| (bară verticală)	Separă elementele din parantezele drepte sau acolade; numai unul dintre acestea se poate introduce în instrucțiunea respectivă
[, . . . n]	Elementul precedent poate fi repetat de n ori; elementele repetate sunt separate prin virgulă
element1, ... elementn	Listă de n elemente de acelaşi tip; elementele repetate sunt separate prin virgulă
lista_elemente	Listă de elemente de acelaşi tip separate prin virgulă

- Caracterele folosite pentru a specifica o anumită convenție sintactică (paranteze, bara verticală, virgula, etc.) nu apar în instrucțiunile propriu-zise
- Listele de elemente (compuse din elemente separate prin virgulă) vor fi repr. folosind una cele trei din construcțiile de mai sus

Instrucțiuni SQL

- Componenta de definire a datelor din SQL (LDD - Limbajul de Definire a Datelor):
  - Crearea (**CREATE**), modificarea (**ALTER**) şi distrugerea (**DROP**) obiectelor bazei de date
  - Obiectele bazei de date sunt: tabele de bază (**TABLE**), tabele vedere (**VIEW**), indexuri (**INDEX**), proceduri (**PROCEDURE**), trigere (**TRIGGER**), utilizatori (**USER**)
- Exemple de comenzi SQL de definire a datelor:

  CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE USER

  CREATE FUNCTION, CREATE TRIGGER, CREATE PROCEDURE

  ALTER TABLE, ALTER VIEW, ALTER FUNCTION, ALTER PROCEDURE

  DROP TABLE, DROP VIEW, DROP INDEX, DROP USER

  DROP FUNCTION, DROP PROCEDURE, DROP TRIGGER
- Componenta de manipulare a datelor din limbajul SQL (Limbajul de Manipulare a Datelor - LMD) conține comenzile: SELECT, INSERT, UPDATE și DELETE
- Instructiunile SQL se transmit SGBD-ului:
  - de către diferite programe client (client grafic, linie de comanda, program executabil)
  - SGBD-ul compilează și execută instructiunea SQL
  - returnează un răspuns (rezultatul operației sau un cod de eroare)

Crearea tabelelor

- Instrucțiunea CREATE TABLE are următoarea sintaxă:

```
CREATE TABLE nume_tabel (
  col1 domeniu1 [constrângeri_coloana],
  col2 domeniu2 [constrângeri_coloana],
  . . . . . .
  coln domeniun [constrângeri_coloana],
  [constrângeri_tabel] );
```

- Constrângerile impuse fiecărei coloane (atribut), ca şi constrângerile de tabel, sunt opționale şi vor fi discutate în sectiunea următoare. Exemplu:

```
CREATE TABLE ANGAJATI (
   Nume  varchar(20),
   Prenume    varchar(20),
   DataNasterii      date,
   Adresa       varchar(50),
   Functia      varchar(20),
   Salariu      numeric);
```

- Instrucțiunea CREATE TABLE defineşte atât tipul relației cât şi o variabilă relație de acel tip care inițial este vidă (nu conține nici un tuplu)
- Ulterior, se pot introduce linii în tabel (tupluri)

Crearea vederilor

- Tabelele create cu instrucțiunea CREATE TABLE:
  - se numesc şi ***tabele de bază*** (base tables)
  - ele sunt memorate în fişierele bazei de date şi pot fi accesate pentru introducerea, modificarea şi regăsirea (interogarea) datelor
- *Un tabel vedere* (*view*) este un tabel virtual care:
  - nu este memorat fizic în fişiere
  - reprezintă o selecție (după un anumit criteriu) a datelor memorate în unul sau mai multe tabele de bază
- Un tabel vedere se creeaza cu instrucțiunea SQL:

  CREATE VIEW nume_vedere AS (SELECT …..);

  Ex: CREATE VIEW INGINERI AS (SELECT * FROM ANGAJATI WHERE Funcția='inginer');
- Formatul comenzii SELECT va fi descris în capitolul următor
- Datele (valorile atributelor) sunt memorate o singură dată, în tabelele de bază, dar pot fi accesate atât prin tabelele de bază cât şi prin tabelele vederi
- Un tabel vedere este întotdeauna actualizat ("la zi"), adică orice modificare efectuată în tabelele de bază se regăseşte imediat în orice tabel vedere creat pe baza acestora

Modificarea și stergerea tabelelor și a vederilor

- Comanda de modificare a tabelelor (ALTER TABLE) permite:
  - adăugarea sau ştergerea unor atribute
  - modificarea domeniilor unor atribute
  - adăugarea, modificarea sau ştergerea unor constrângeri ale tabelului (chei străine)
- Pentru adăugare unei coloane într-un tabel se folosește clauza ADD, urmata de numele coloanei și numele domeniului (tipul SQL) atributului corespunzător. Exemplu:

  ALTER TABLE ANGAJATI **ADD** DataAngajarii date;
- Pentru ştergerea unei coloane dintr-un tabel se foloseşte clauza DROP, urmata de numele coloanei care se va sterge. Exemplu:

  ALTER TABLE ANGAJATI **DROP** DataAngajarii;
- Instrucțiunile de ştergere a tabelelor de bază şi a vederilor sunt:

  DROP TABLE nume_tabel;

  DROP VIEW nume_vedere;

Instrucțiunea SELECT

- SELECT - instrucțiune de interogare, prin care se regăsesc informațiile din unul sau mai multe tabele ale bazei de date dupa un criteriu (conditie) dat
- Sintaxa generală:

  SELECT [DISTINCT] lista_coloane

  [FROM lista_tabele]

  [WHERE conditie]

  [clauze_secundare];

- SELECT returneaza un tabel cu coloanele din “lista_coloane”
  - ale acelor linii (tupluri) ale produsului cartezian al tabelelor din **“lista_tabele”** pentru care expresia logică **“conditie”** este adevărată (are valoarea **TRUE**).
- Instructiunea SELECT are urmatoarele secțiuni (clauze):
  - Clauza **SELECT** defineşte lista de coloane a tabelului rezultat
  - Clauza **FROM** indică lista de tabele din care se selectează rezultatul
  - Clauza **WHERE** defineşte condiția pe care trebuie să o îndeplinească fiecare linie a tabelului rezultat
  - Clauze secundare (**ORDER BY**, **GROUP BY**, **HAVING**): permit ordonări sau grupări ale tuplurilor (liniilor) rezultate

Clauza SELECT

- Clauza SELECT specifică lista coloanelor tabelului rezultat; o coloană a tabelului rezultat poate fi:
  - o coloană a unuia dintre tabelele date în “lista_tabele”
  - o expresie care poate fi calculată
- Exemple:

  SELECT Name, CountryCode from city;

  SELECT 3*4, cos(45), floor(12.45);
- Eliminarea liniilor duplicat – cu parametrul DISTINCT. Exemplu:

  SELECT [DISTINCT] CountryCode FROM city;
- Selectarea tuturor coloanelor produsului cartezian al tabelelor date - cu caracterul * ca și “lista_coloane”. Exemplu:

  SELECT * FROM city;
- În clauza SELECT se pot redenumi tabelele și coloanele tabelelor sau se pot specifica nume pentru expresii, folosind următoarea sintaxă:

  SELECT nume1 [AS] noul_nume1 [,...n] FROM lista_tabele [alte_clauze];

  SELECT ID, Name Oras, CountryCode ‘Cod Tara’ FROM city;

Clauzele FROM și WHERE

- ***Clauza FROM*** specifică “lista_tabele” din care se selectează rezultatul
  - Dacă sunt mai multe tabele, se face produsul cartezian al acestora
- Numele coloanelor din “lista_coloane” (clauza SELECT) trebuie să fie distincte
- Dacă nu sunt distincte, se califică unele coloane cu numele tabelului caruia îi aparțin - folosind operatorul “punct” (.). De exemplu:

  SELECT ANGAJATI.Nume, SECTII.Nume FROM ANGAJATI, SECTII;
- ***Clauza WHERE*** specifica “conditia” pe care trebuie sa o îndeplinesca rezultatul:
  - conditia este o expresie logică compusa din valori logice, operatori logici (NOT, AND, OR) şi paranteze
  - o valoare logică se obtine ca rezultat al comparației între doi operanzi folosind un operator de comparație
  - un operand poate fi un atribut (nume de coloană), o constantă, valoarea unei expresii aritmetice sau o valoare returnată de o funcție
  - operatorii de comparație pot fi operatori aritmetici sau operatori SQL de comparație
- Exemple:

  SELECT * FROM city WHERE Population > 100000;

  SELECT * FROM city WHERE (Population > 200000) AND (CountryCode=‘ROM‘);

Clauze secundare – funcții agregat

- Clauzele secundare sunt: ORDER BY, GROUP BY, HAVING
- ***Clauza ORDER BY*** specifică numele atributului după care se face ordonarea liniilor tabelului rezultat

  SELECT * FROM city order by CountryCode;
- Ordonarea în ordine crescătoare: parametrul ASC (implicit); în ordine descrescatoare: DESC. Exemplu:

  SELECT * FROM city order by CountryCode DESC;
- Clauzele GROUP BY și HAVING se folosesc împreună cu funcțiile agregat
- Funcțiile agregat definite în limbajul SQL2 sunt următoarele:

Functia	Valoarea returnata
COUNT	Numarul de linii al tabelului rezultat
SUM	Suma valorilor din coloana data ca argument
MAX	Valoarea maxima din coloana data ca argument
MIN	Valoarea minima din coloana data ca argument
AVG	Valoarea medie din coloana data ca argument

Funcţii agregat

- Utilizarea funcțiilor agregat fără clauze de grupare (GROUP BY, HAVING):

```
SELECT COUNT(*) FROM city;          -- returneaza numarul de linii din tabel
SELECT COUNT(col) FROM city;        -- return nr de valori dif de null din acea col.
SELECT MAX(Population) FROM city;
SELECT MIN(Population) FROM city;
SELECT AVG(Population)  FROM city;
```

- Utilizarea funcțiilor agregat împreună cu clauze de grupare (GROUP BY, HAVING)
- ***Clauza GROUP BY*** se foloseşte pentru gruparea rezultatelor funcțiilor agregat (aplicate unei anumite coloane) dupa valoarea uneia sau mai multor (alte) coloane, date ca argument al clauzei GROUP BY. Exemplu:

```
SELECT CountryCode, AVG(Population) FROM city GROUP BY CountryCode;
```

Funcția **AVG (POPULATION)** calculeaza media populatiei pentru fiecare grup de linii din tabelul City care au aceeași valoare a atributului CountryCode. Instr. SELECT returnează un tabel format din atâtea linii câte valori distincte are atributul CountryCode (argumentul clauzei GROUP BY)

- ***Clauza HAVING*** înlocuiește clauza WHERE atunci când în condiția care trebuie să fie îndeplinită se folosesc funcții agregat. Exemplu:

```
SELECT CountryCode, AVG(Population) FROM city GROUP BY CountryCode HAVING
    AVG(Population) >200000;
```

Instrucțiunea INSERT

- Instrucțiunea INSERT se foloseşte pentru introducerea datelor în tabele şi are următoarea sintaxă:

  INSERT INTO nume_tabel (col1,col2,...coln) VALUES(val1,val2,...valn);

- Între valori şi numele de coloane trebuie să existe o corespondență pozițională

  De exemplu, inserarea unei linii în tabelul SECTII(Numar, Nume, Buget):

  INSERT INTO SECTII (Numar, Nume, Buget) VALUES (1,'Productie', 40000);

- Lista de coloane poate să lipsească dacă se introduc valori în toate coloanele tabelului și în această situatie:
  - ordinea valorilor introduse trebuie să respecte ordinea coloanelor tabelului
  - ordinea coloanelor provine din ordinea de definire a atributelor prin instrucțiunea CREATE TABLE, precum şi din operațiile ulterioare de alterare a tabelului
  - ordinea coloanelor se poate afla prin instrucțiunea DESCRIBE nume_tabel.

- De exemplu, introducerea unei linii cu toate valorile în tabelul

  ANGAJATI (IdAngajat, Nume, Prenume, DataNasterii, Adresa, Functia, Salariu)

  INSERT INTO ANGAJATI

  VALUES(100,'Mihailescu', 'Mihai','1950-04-05','Craiova','inginer', 3000);

  Dacă nu se specifică valori pentru toate atributele, atributele nespecificate primesc valoare implicită (DEFAULT), dacă a fost specificată sau NULL (dacă este acceptată); altel se dă eroare:

  INSERT INTO ANGAJATI (IdAngajat, Nume, Prenume, Adresa, Functia)
  VALUES(100,'Duru', 'Ion','Ploiesti','inginer');

Instrucțiunile UPDATE și DELETE

- ***Instrucțiunea UPDATE*** permite actualizarea valorilor coloanelor (atributelor) din una sau mai multe linii ale unui tabel și are sintaxa:

  UPDATE nume_tabel SET col1 = expr1 [, . . . n] [WHERE conditie];

- Clauza WHERE: actualizarea valorilor se efectueaza numai asupra acelor linii care îndeplinesc condiția dată. Exemplu:

  UPDATE ANGAJATI SET Adresa = 'Bucuresti' WHERE Nume = 'Popescu';

- Dacă este omisă clauza WHERE, vor fi modificate valorile coloanelor din toate liniile tabelului.

- ***Instrucțiunea DELETE*** permite ştergerea uneia sau mai multor linii dintr-un tabel şi are sintaxa:

  DELETE FROM nume_tabel [WHERE conditie];

- Din tabel se şterg acele linii care îndeplinesc condiția dată în clauza WHERE Dacă este omisă clauza WHERE, vor fi sterse toate liniile din tabel

- Exemplu:

  DELETE FROM ANGAJATI WHERE Nume ='Ionescu';

Constrângeri de integritate (1)

- Constrângerile de integritate (*integrity constraints*) sunt reguli care se impun pentru ca datele memorate să corespundă cât mai bine celor din realitate:
  - se definesc la proiectarea bazei de date
  - trebuie să fie respectate de orice stare a acesteia
- Clasificare după locul unde se definesc: constrângeri de coloana și constrângeri de tabel
- Clasificare după numărul de relații implicate: constrângeri intra-relație şi constrângeri inter-relații.
- Constrângerile intra-relație - reguli care se impun în cadrul unei singure relații; sunt de trei categorii:
  - Constrângeri de domeniu - condiții ce se impun valorilor domeniilor atributelor
  - Constrângeri de tuplu - condiții ce se impun tuplurilor unei relații prin chei (primare sau secundare)
  - Constrângeri impuse prin dependențe de date (dependențe funcționale, multivalorice sau de joncțiune); acestea sunt constrângeri intre valorile atributelor dintr-o relație
- Constrângerile inter-relații - reguli care se impun între două sau mai multe relații; asigura integritarea referențială (asocierea corectă a relațiilor) prin intermediul cheilor străine

Constrângeri de integritate (2)

- Clasificare din punct de vedere al modului de definire și de verificare a respectării constrângerilor: inerente, implicite şi explicite.
- ***Constrângerile inerente*** sunt cele ale modelului de date însuşi, care nu trebuie să fie definite deoarece sunt incluse în sistemul de gestiune
  - De exemplu: în modelul relațional constrângerea ca valoarea fiecărui atribut să fie atomică (indivizibilă) este o constrângere inerentă
- ***Constrângerile implicite*** sunt reguli specifice fiecărui sistem de gestiune; acestea se definesc de către proiectantul bazei de date, iar sistemul de gestiune le verifică şi le impune automat
  - Fiecare SGBD poate are propriile constrângeri implicite, dar, în general connstrângerile de domeniu, constrângerile de tuplu şi constrângerile de integritate referențială sunt constrângeri implicite în orice SGBD
- ***Constrângerile explicite*** sunt constrângeri suplimentare, specifice bazei de date respective; proiectantul definește constrângerile explicite precum și procedurile de verificare ale accestora (funcții, proceduri stocate, triggere)
  - Exemple: dependențele de date care nu sunt determinate de cheile relațiilor

Constrângeri de domeniu (1)

- ***Constrângerile de domeniu***: constrângerea NOT NULL, constrângerea de valoare implicită (DEFAULT), constrângerea de verificare (CHECK)
- ***Constrângerea NOT NULL*** însemna că atributul respectiv nu poate lua valoarea NULL în nici un tuplu al relației.
- Valoarea NULL a unui atribut într-un tuplu semnifică faptul că valoarea acelui atribut nu este cunoscută pentru acel tuplu. Exemple:
  - nu se cunoaste deloc data de nastere a unei personalitati istorice;
  - nu se cunoaşte valoarea unui atribut în momentul inserarii tuplului, dar aceasta va fi cunoscuta și completată ulterior
- La crearea unui tabel opțiunea NULL este implicită (dacă nu se specifică nimic), sau se poate introduce explicit NULL sau NOT NULL; optiunea NOT NULL se introduce numai explicit
- Optiunile NULL și NOT NULL se introduc ca și constrângeri de coloana în instructiunea SQL CREATE TABLE. Exemplu:

```
CREATE TABLE ANGAJATI (
   Nume  varchar(20) NOT NULL,
   Prenume    varchar(20) NOT NULL,
   DataNasterii      date NULL,
   Functie      varchar(20),
   Salariu      numeric);
```

Constrângeri de domeniu (2)

- ***Constrangerea de valoare implicită a unui atribut (DEFAULT)***: dacă la inserarea unui tuplu nu se specifică valoarea unui atribut, atunci:
  - atributul primeşte valoarea implicită (DEFAULT), dacă a fost definită; ex. Tara
  - atributul primeşte valoarea NULL, dacă nu a fost definită valoare implicită, dar sunt admise valori NULL; ex. DataNasterii
  - se generează o eroare, dacă nu a fost definită o valoare implicită şi nici nu sunt admise valori NULL; ex: Nume

```
CREATE TABLE STUDENTI (
    Nume        varchar (20) NOT NULL,
    Prenume     varchar (20) NOT NULL,
    DataNasterii      date,
    Tara    varchar (20) DEFAULT 'Romania' ) ;
```

- ***Constrângerea de verificare (CHECK)*** **–** pentru verificarea valorilor atributelor printr-o conditie care trebuie sa ia valoarea TRUE.
- Se introduce ca o constrangere de tabel în instructiunea CREATE TABLE:

```
[CONSTRAINT nume_constrangere] CHECK (conditie); Exemplu:
CREATE TABLE ANGAJATI (
    Nume        varchar(20) NOT NULL,
    Prenume     varchar(20) NOT NULL,
    Salariu     numeric,
    CONSTRAINT Verificare_Salariu CHECK (Salariu >= 1500 ));
```

- MySql 5.0 nu face verificarea CHECK, chiar daca admite acest cuvânt cheie

Constrângeri de tuplu

- O relație = mulțime de tupluri → tuplurile unei relații trebuie să fie distincte (nu pot exista două sau mai multe tupluri identice)
- Pentru ca tuplurile unei relații să fie distincte se folosește câte o cheie primară (*primary key*) în fiecare relație
- *O cheie primară* PK a unei relații este un atribut (simplu sau compus) al acelei relații care are proprietatea de unicitate, adică fiecare ***valoare*** a cheii primare este ***unică*** în acea relație. Aceasta înseamnă că:
  - Nu există două tupluri distincte (diferite) care să aibă aceeași valoare a cheii primare (sau combinație de valori) pentru orice stare a relației, adică:

    ti[PK] ≠ tj[PK] dacă i ≠ j, unde ti și tj sunt 2 tupuri diferite ale relației
- Proprietatea de unicitate a cheii primare este o constrângere de integritate a tuplurilor: fiecare tuplu poate fi identificat în mod precis
- Cheia primară trebuie să respecte următoarele cerințe:
  - Să fie ***ireductibilă***: să nu existe o submulțime proprie nevidă a cheii PK care să aibă proprietatea de unicitate
  - Să fie ***definită (cunoscută)*** pentru orice tuplu din relație; de aceea nu se admit valori de NULL pentru nici unul dintre atributele cheii primare
- *Cheia primară este o constrângere implicită*: se definește de proiectant la crearea tabelului, iar SGBD-ul verifică respectarea și menț. integrității tuplurilor:
  - La INSERT, tuplul trebuie să aibă cheia primară definită și unică (să nu existe alte tupluri în relație cu aceeași valoare a cheii primare)
  - Principial, la UPDATE, se interzice modificarea valorii cheii primare (dar unele SGBD-uri pot sa admită modificarea, cu condiția ca valoarea cheii mod. să fie unica)

Chei primare naturale și artificiale (1)

- Se pot defini fie *chei primare naturale,* fie *chei primare artificile*, cu condiția ca acestea să îndeplinească ***condițiile de unicitate și ireductibilitate***
- *O cheie primară naturală* este un atribut (simplu sau compus) al relației:
  - reprezintă o proprietate a tipului de entitate (sau asociere) reprezntat de acea relație
  - are în mod natural valori unice: nu există două tupluri cu aceeași valoare a cheii primare, deoarece nu există două entități cu aceeași valoare a proprietății respective; de ex. CNP-ul persoanelor (din relatii ca ANGAJATI, STUDENTI etc.) din Romania
- *O cheie primară artificială* este un atribut (de obicei simplu) care nu reprezintă o proprietate a tipului de entitate sau asociere reprezentat de relație, ci se adaugă în schema relației special pentru identificarea unică a tuplurilor
  - *Unicitatea* cheii primare artificiale trebuie să fie asigurată de proiectant și SGBD
  - *Ireductibilitatea* cheii primare artificiale este asigurată dacă este atribut simplu
- Ex.:ANGAJATI (IdAngajat, CNP, Nume, Prenume, DataNasterii, Adresa, Functia, Salariu)
  - IdAngajat este o cheie primară artificială
  - Ar putea fi definite și chei primare naturale prin atribute simple sau compuse care au proprietatea de unicitate în anumite condiții:
    - atributul simplu {CNP} – valabil numai pentru persoanele din Romania
    - atributul compus {Nume, Prenume, DataNasterii, Adresa} – are prea multe atribute
- Din motive de eficiență a operațiilor de identificare a tuplurilor, se preferă chei primare cu un număr cât mai mic de atribute (atribut simplu)

Chei primare naturale și artificiale (2)

- SGBD-urile oferă diferite mijoace de asigurare a unicității valorii cheii primare artificiale. De exemplu:
  - În Microsoft SQL Server se pot obține valori unice ale cheii primare folosind parametrul IDENTITY, care asigură incrementarea valorii atributului cheii la introducerea fiecărei linii noi
  - În sistemele Oracle se pot genera chei artificiale folosind obiecte SEQUENCE; un obiect SEQUENCE geneaza un numar unic la fiecare apel al metodei NEXTVAL
  - În MySQL, se foloseste parametrul AUTO_INCREMENT pentru generarea numerelor unice pentru cheile primare
- SGBD-urile interzic introducerea într-o relație a unui tuplu (linie în tabelul corespunzător) care are valoarea cheii primare identică cu valoarea cheii primare a unui tuplu existent în acea relație

Definirea cheii primare în SQL

- În SQL cheia primară se defineste prin instructiunea CREATE TABLE, ca o constrângere de tabel sau ca o constrângere de coloană
- Definirea cheii primare ca o constrângere de tabel:

  [CONSTRAINT nume_constr] PRIMARY KEY (lista_atribute)

  Exemplu:

```
CREATE TABLE SECTII (
IdSectie     int,
Nume         varchar(50)  NOT NULL,
Buget        numeric,
   CONSTRAINT PK PRIMARY KEY (IdSectie)
 );
```

- Dacă cheia primară este simplă (formată dintr-un singur atribut), ea se poate specifica și ca o constrângere de coloană; exemplu:

```
CREATE TABLE ANGAJATI (
    IdAngajat          int PRIMARY KEY AUTO_INCREMENT,
    Nume                     varchar(20)     NOT NULL,
    Prenume            varchar(20)     NOT NULL,
    DataNasterii       date,
    Adresa             varchar(50),
    CNP   char(10),
    Functia            varchar(20),
    Salariu            numeric
) ;
```

Superchei, chei candidate

- ***O supercheie*** (*superkey*) este o submulțime SK de atribute ale unei relații care prezintă proprietatea de ***unicitate*** (orice combinație de valori ale atributelor supercheii este unică pentru orice stare a relației)
  - Dacă se cunoaşte valoarea (combinația de valori ale atributelor) supercheii, atunci acel tuplu poate fi identificat în mod unic
  - Orice relație are cel puțin o supercheie, care este mulțimea tuturor atributelor sale
- ***O cheie candidată*** (*candidate key*) este o supercheie ireductibilă, deci are proprietățile:
  - ***Unicitate***: nu există două tupluri diferite ale relației care să conțină aceeaşi combinație de valori ale atributelor cheii CK;
  - ***Ireductibilitate***: nu există nici o submulțime proprie, nevidă a cheii CK care să aibă proprietatea de unicitate
- O cheie candidată poate fi simplă (un atribut), sau compusă (mai multe atribute)
  - Exemplu: ANGAJATI (Nume, Prenume, CNP, DataNasterii, Adresa, Functia, Salariu)

    SK1 = {Nume, Prenume, CNP, DataNasterii, Adresa, Functia, Salariu} --unicitate

    SK2 = {Nume, Prenume, CNP, DataNasterii, Adresa} --unicitate

    CK1 = {CNP}, CK2= {Nume, Prenume, DataNasterii, Adresa} --unicitate si ireductibilitate

Chei primare și secundare

- ***Cheia primară:*** se definește la crearea tabelului în instrucțiunea CREATE TABLE
  - fie se alege dintre cheile candidate
  - fie se introduce o cheie primară artificială
  - Pentru eficiența, se prefera chei primare formate dintr-un singur atribut; de aceea, în general, se folosesc chei primare artificiale deoarece puține tipuri de entități au un atribut simplu cu proprietatea de unicitate (de ex. CNP, în anumite situații)
- ***O cheie secundară*** (alternativă, unică) (*secondary*, *alternate*, *unique key*) este o cheie candidată care nu a fost aleasă ca și cheie primară de către proiectant
  - O cheie secundară (unică) se poate defini în instrucțiunea CREATE TABLE folosind specificatorul UNIQUE [KEY] în loc de PRIMARY KEY
  - Dacă a fost definită o cheie secundară, SGBD-ul verifică unicitatea valorilor acesteia la insert si update; dacă nu a fost definită, SGBD-ul nu verifică nimic
- Alegerea cheii primare dintre mai multe chei candidate este arbitrară, dar, din motive de eficiență, se alege cheia cu cel mai mic număr de atribute
- Cheile secundare se deosebesc de cele primare prin:
  - Pot fi modificate prin instrucțiuni UPDATE, dacă se respectă proprietatea de unicitate
  - Cheile secundare compuse admit valori NULL pentru unele din atributele lor

Constrângeri inter-relații

- ***Asocierile*** (*relationships*) 1: N între două mulțimi de entități (din modelul Entitate-Asociere) se realizează în modelul relațional prin chei străine
- Exemplu: Pentru a realiza asocierea 1: N dintre relațiile SECTII și ANGAJATI, se adaugă în relația ANGAJATI cheia străină IdSectie, care reprezintă identificatorul (numărul) secției în care lucrează angajatul respectiv:

ANGAJATI (IdAngajat, Nume, Prenume, DataNasterii, Adresa, Salariu, IdSectie)

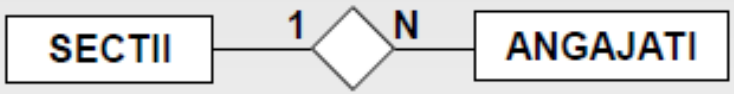


Diagrama E-A

SECTII

IdSectie	Nume	Buget
1	Productie	400000
2	Proiectare	300000
3	Cercetare	200000
4	Documentare	100000

ANGAJATI

IdAngajat	Nume	Prenume	DataNasterii	Adresa	Salariul	IdSectie
1	Ionescu	Ion	1960.01.05	Bucuresti	4000	1
2	Popescu	Petre	1965.02.97	Bucuresti	3200	1
3	Vasilescu	Ana	1961.03.06	Bucuresti	2000	2
4	Ionescu	Ion	1970.03.98	Bucuresti	2000	3

Cheia străină

- Fie două relații R1 și R2, între care exista o asociere cu raportul 1: N.
  O cheie străină (*foreign key*) este o submulțime FK de atribute ale relației R2 care referă cheia CK din relația R1 şi satisface următoarele condiții:
  - atributele cheii străine FK sunt definite pe domenii compatibile cu cele ale atributelor cheii candidate CK a relației R1
  - valorile atributelor FK într-un tuplu din relația R2, fie sunt identice cu valorile atributelor CK ale unui tuplu oarecare din starea curentă a relației R1, fie sunt NULL
- Două domenii sunt compatibile dacă ele sunt compatibile din punct de vedere al tipului de date și semantic (are sens să fie comparate)
  - În limbajul SQL verificarea domeniilor se rezumă la verificarea compatibilității tipurilor de date, iar compatibiltatea semantică trebuie să fie asigurată de proiectant
- Cheia străină reprezintă o constrângere referențială între cele 2 relații
  - Relația referită (R1) – relație părinte, relația care referă (R2) – relație fiu
  - Referința se face prin valoare: val. cheii străine este egală cu val cheii candid. referite
- Cheia străină se specifică în comanda CREATE TABLE sau ALTER TABLE:

  [CONSTRAINT nume_constr] FOREIGN KEY (cheie_străina)
  REFERENCES relația_referita (cheie_candidata)

- Ex:
```
CREATE TABLE ANGAJATI (
    IdAngajat       int PRIMARY KEY,
    Nume            varchar(20) NOT NULL,
    Prenume         varchar(20) NOT NULL,
    IdSecție        int,
    CONSTRAINT FK FOREIGN KEY (IdSectie) REFERENCES SECTII(IdSectie));
```

Mentinerea integrității referențiale a relațiilor (1)

- ***Integritatea referențială*** (*referential integrity*) este proprietatea bazei de date prin care orice cheie străină:
  - fie are valoarea cheii candidate a tuplului referit din relația referită
  - fie are valoarea NULL
- Pentru menținerea integrității referențiale trebuie să fie inpuse restrictii operațiilor de modificare a stării relațiilor (INSERT, DELETE, UPDATE)
- Restricțiile care se impun operațiilor de modificare a relațiilor depind de rolul relației (relație care referă, relație referită, sau poate avea ambele roluri)
- Operația INSERT:
  - Într-o relație care nu referă altă relație, inserarea se poate face fără restricții
  - Într-o relație care referă (care conține o cheie străină): SGBD-ul permite introducerea unui tuplu nou numai dacă: (a) valoarea cheii străine a tuplului nou este NULL sau (b) există o valoare a cheii referite egală cu valoarea cheii străine a tuplului nou
- Operatia DELETE:
  - Într-o relație care nu este referită ștergerea se poate face fără restricții
  - Într-o relație referită se admite: ***ştergere restricționată, ştergere în cascadă, anularea (SET NULL) a cheilor străine care refereau tuplul șters***
- ***Ştergerea restricționată*** interzice ştergerea unui tuplu din relația referită dacă acesta este referit de un tuplu din relația care o referă

Mentinerea integrității referențiale a relațiilor (2)

- ***Ştergerea în cascadă*** permite ştergerea unui tuplu din relația referită; dacă tuplul şters era referit de unul sau mai multe tupluri, atunci se şterg şi acestea din relația care o referă; dacă tuplurile şterse din relația care referă sunt, la rândul lor referite de alte tupluri din alte relații, atunci trebuie să fie şterse şi acestea, ş.a.m.d.; se execută deci o ştegere în cascadă
- Operația UPDATE este o ştergere urmată de o introducere, deci restricțiile de actualizare reprezintă combinația restricțiilor de introducere şi de ştergere
- În limbajul SQL se specifică opțiunile ON DELETE și ON UPDATE constrîngerii de cheie străină; valorile posibile ale acestor opțiuni sunt:
  - **RESTRICT** – ştergerea restricționată
  - **CASCADE** – ştergerea în cascadă
  - **SET NULL** – setare valoare NULL a cheilor străine care refereau tuplul șters
  - **NO ACTION** – se admit valori care nu respectă integritatea relațională
- Exemplu:

```
CREATE TABLE ANGAJATI (
    IdAngajat       int PRIMARY KEY,
    Nume            varchar (20) NOT NULL,
    . . . . . .
    Sectie          int,
    CONSTRAINT FK FOREIGN KEY (Sectie) REFERENCES SECTII (IdSectii) ,
            ON DELETE CASCADE ON UPDATE RESTRICT);
```

Indexarea relațiilor (1)

- Timpul de execuție a operatiilor asupra datelor din relații depinde de modul de reprezentare a mulțimii de elemente (tupluri) ale relațiilor
- Operația de căutare a unui element într-o mulțime se execută mai rapid dacă elementele mulțimii sunt reprezentate printr-o colecție ordonată, cum sunt liste, arbori, tabele de dispersie (*hash table*). De exemplu:
  - Timpul de căutare a unui element într-o mulțime neordonată de N elemente este proportional cu N: $T_C = k_1 * N = O(N)$
  - Timpul de căutare al unui element memorat într-o structură arbore binar de căutare ordonat după valoarea etichetei (cheii) de ordonare este $T_C = k_2 * \log (N+1) = O(\log N)$
- Un arbore binar ordonat complet cu d niveluri:

  pe nivelul 0 are $2^0 = 1$ nod

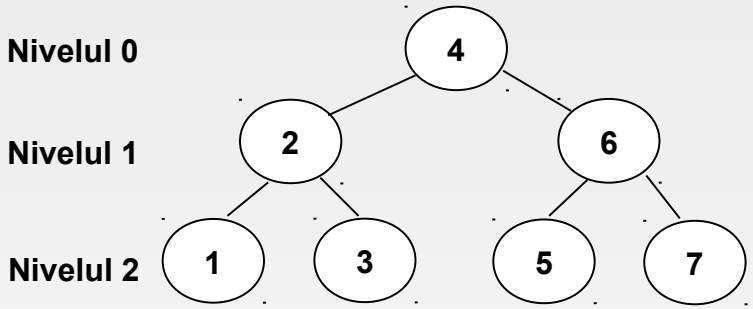
  pe nivelul 1 are $2^1 = 2$ noduri

  pe nivelul j are 2^j noduri

  Nr. total noduri $N = 2^0 + 2^1 + \ldots 2^j + 2^{d-1} = 2^d - 1$

  $d = \log (N + 1)$

  Pentru căutare se parcurg max d pași, deci timpul de căutare $T_C = k_2 * \log (N+1) = O(\log N)$

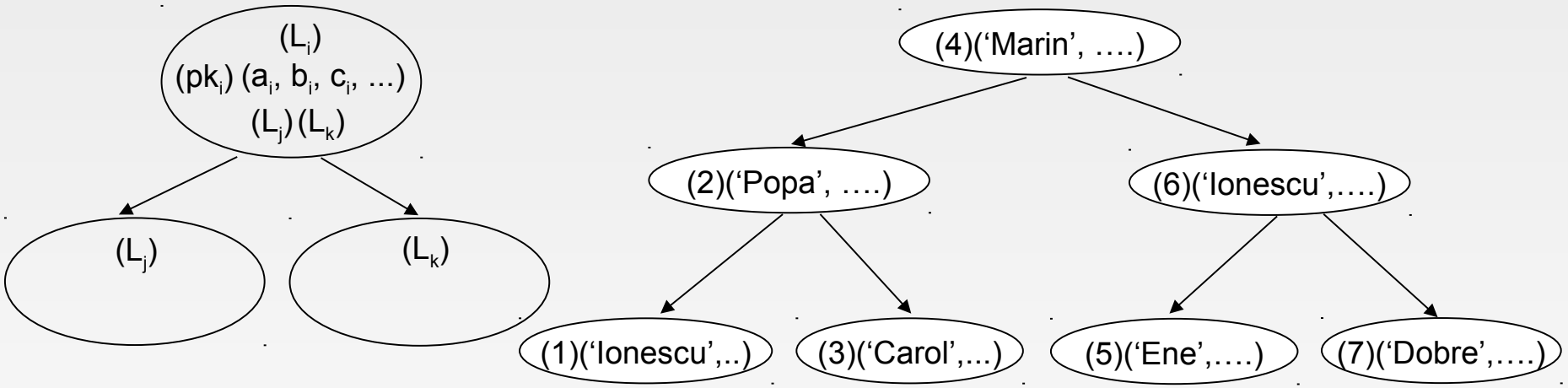


Indexarea relațiilor (2)

- Rezultă că, deși nu este obligatoriu ca tuplurile unei relații să fie ordonate, pentru accelerarea căutării (operația SELECT) a unui tuplu după o cheie unică (candidată - primară sau secundară), se folosesc colecții ordonate de tupluri
- Și celelate operații (INSERT, UPDATE, DELETE) se execută mai rapid în colecții ordonate, deoarece toate caută mai întâi tuplul cu cheia dată
  - Pentru inserarea unui tuplu, se verifică mai întâi să nu existe deja un tuplu cu aceeași valoare a cheii și inserarea se face numai dacă nu există un astfel de tuplu
  - Pentru modificarea unui tuplu, se caută mai întâi tuplul cu cheia dată, apoi se fac modificările
  - Pentru ștergere, se caută mai întâi tuplul cu cheia dată și apoi se șterge
- ***Un index*** al unei relații este o structură auxiliară, memorată în baza de date, care permite accesul rapid la tuplurile relației prin ordonarea acestora
- Structuri folosite în indexare: arbori binari de căutare, arbori *BTREE*, arbori *RTREE*, tabele de dispersie (*HASH*) etc.
- Există două categorii de indexuri ale unei relații:
  - un index primar, care ordonează și localizează tuplurile în fişierele bazei de date
  - zero, unul sau mai multe indexuri secundare, care nu modifică localizarea tuplurilor, dar sunt folosite pentru găsirea rapidă a tuplurilor după valorile unor atribute

Indexul primar (1)

- ***Indexul primar*** (*primary index*) se defineşte pe cheia primară a relației
- Fiecare element (nod) al indexului primar conține un tuplu al relației și elementele sunt ordonate după valoarea cheii primare PK
- De exemplu, pentru o structură arbore binar ordonat a indexului primar al unei relații cu ckeia primară PK și atributele (A, B, C, ...), un element (nod) N_i este memorat la adresa L_i pe hard-disk și conține:
  - Valoarea cheii primare a tuplului (pk_i), care este și eticheta de ordonare a arborelui
  - Valorile celorlalte atribute ale tuplului (a_i, b_i, c_i, ...)
  - Adresele fiilor (L_j, L_k) (locațiile de memorare pe hard-disk a nodurilor fii)



Indexul primar și indexuri secundare

- Deci fiecare element (nod) al indexului primar conține un tuplu al relației
- Operațiile de interogare care se fac după indexul primar (cheia primară) se execută eficient, fiind o căutare într-o mulțime ordonată după acea valoare
  - Exemplu: “Care sunt numele, prenumele și salariul angajatului **cu cheia primara 3**?”

  In SQL: SELECT Nume, Prenume, Salariu FROM Angajati **WHERE IdAngajat = 3;**

  Se caută nodul arborelui care are valoarea etichetei de ordonare (care este și cheia primară a relației) egală cu valoarea dată (3)

  După găsirea nodului se extrag valorile atributelor tuplului memorat în acel nod

  Sunt necesari maximum d (log N) pași de căutare (N este nr total de tupluri ale relației)
- Operatiile de interogare care se fac după valoarea altor atribute (decât indexul primar) se execută mult mai ineficient, fiind o căutare într-o mulțime neordonată după acea valoare
  - Exemplu: “Care sunt numele, prenumele și salariul angajatului **cu numele ‘Carol’** ?”

  In SQL: SELECT Nume, Prenume, Salariu FROM Angajati **WHERE Nume = 'Carol'**;

  Pentru căutare se vor parcurge pe rând toate tuplurile relației pentru a găsi tuplul (sau tuplurile) cu valoarea atributului Nume egală cu ‘Carol’

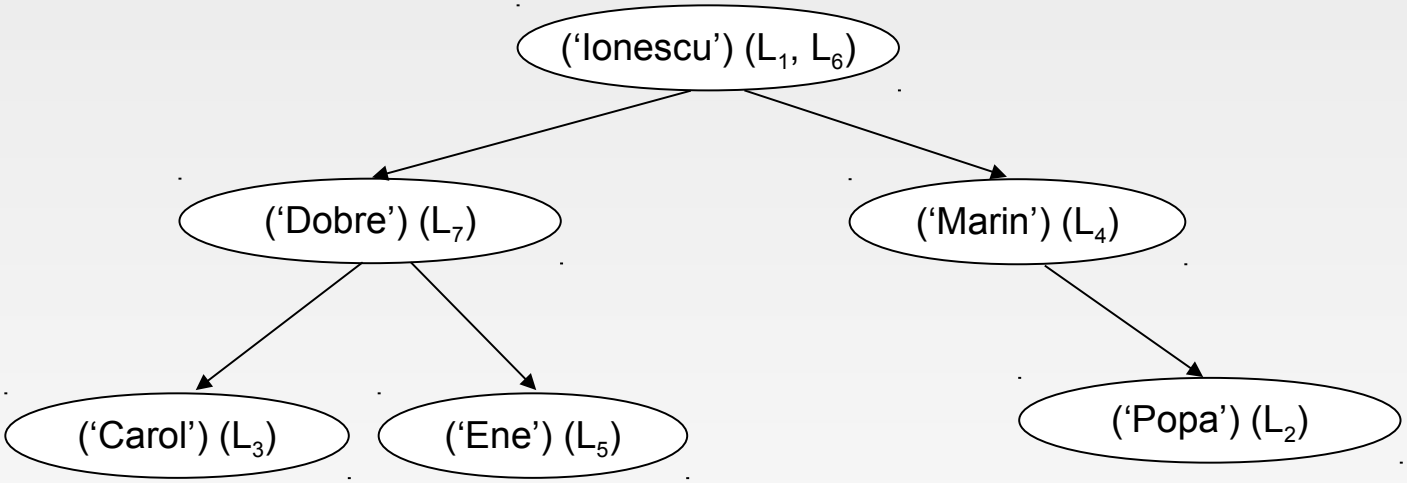
  Sunt necesari maximum N pași (N este nr total de tupluri ale relației)
- Pentru rezolvarea mai eficientă a unor astfel de interogări se definesc indexuri secundare pe acele atribute care intervin în clauza WHERE din interogări

Indexuri secundare (1)

- *Un index secundar pe un atribut al unei relații (secondary index) este o structură ordonată după valoarea acelui atribut; u*n element al unui index secundar conține:
  - valoarea atributului indexat (care este etichetă de ordonare)
  - adresa (sau adresele) tuplurilor care conțin acea valoare a atributului respectiv
- Sunt două categorii de indexuri secundare: unice (UNIQUE) și "normale"
- Un index secundar UNIQUE este definit pe un atribut A (simplu sau compus) al relației care ia valori unice (cum este o cheie unică - secundară sau alternativă)
  - Un element (nod) al indexului este compus din valoarea a_i a atributului indexat A și adresa (L_i) a unui singur tuplu care are acea valoare a atributului A
  - Dacă relația are N tupluri, indexul va avea M = N elemente (noduri)
- Index secundar "normal" (care nu este unic - nu are o denumire specifică) este definit pe un atribut A care nu ia valori unice (nu este cheie unică)
  - Un element (nod) al indexului este compus din valoarea a_i a atributului indexat A și lista (L_{i1} , L_{i2} , …) a adreselor (pe hard-disk) a tuplurilor t_{i1}, t_{i2}, … care au valoarea a_i a atributului A
  - Dacă relația are N tupluri, indexul va avea M ≤ N elemente
- Pentru o structură arbore binar a indexului, fiecare nod mai conține și adresele nodurilor fii (stânga, dreapta) (nereprezentate în figura următoare)

Indexuri secundare (2)

- Exemplu: indexul secundar (cu structură arbore binar) definit pe atributul Nume al relației ANGAJATI, al cărei index primar este cel dat în figura precedentă
  - La interogarea “Care sunt numele, prenumele și salariul angajaților cu numele ‘Carol’ ?” se parcurge indexul secundar definit pe atributul Nume și se află adresa unui singur tuplu (L_3)
  - La interogarea “Care sunt numele, prenumele și salariul angajatilor cu numele ‘Ionescu’ ?” se parcurge indexul secundar definit pe atributul Nume și se află adresele tuplurilor (L_1 și L_6) care au valoarea atributului Nume egală cu ‘Ionescu’
  - Dacă indexul are o structură arbore binar ordonat, se vor executa max (log N) pași
- Un index secundar nu modifică adresa de memorare a unui tuplu (care se află în indexul primar), dar conține informații pentru găsirea rapidă a unui tuplu după valoarea acestui index



Indexuri secundare (3)

- În SQL, un index secundar se poate crea prin comanda CREATE TABLE (ca o costrângere de tabel), prin ALTER TABLE sau cu CREATE INDEX; ex.:

  CREATE [optiuni] INDEX nume_index ON nume_tabel (lista_atribute_index);

  Una din opțiunile care se pot introduce în CREATE INDEX este UNIQUE
- În general, sistemele SGBD adaugă:
  - Un index secundar UNIQUE pentru fiecare cheie candidată (definită prin constrângerea UNIQUE KEY)
  - Un index secundar normal pentru fiecare cheie străină; un astfel de index secundar ajută la găsirea rapidă a tuturor tuplurilor asociate cu o valoare a cheii străine (“Care sunt angajații care lucrează în secția cu numărul (identificatorul IdSectie) 1?”
- În sistemele SGBD avansate (obiect-relaționale), pot exista și indexuri secundare speciale, cum sunt:
  - Indexuri spațiale (indexarea obiectelor reprezentate în spațiul bi sau tridimensional)
  - Indexuri de context (indexarea textelor)
  - Indexuri XML (indexarea documentelor XML)
- Indexurile secundare au avantaje și dezavantaje:
  - Avantaje: accelerează operațiile de interogare care se fac după valoarea indexului
  - Dezavantaje: ocupă spațiu de memorie și consumă timp la actualizarea relațiilor
- Se recomandă utilizarea unui număr cât mai mic de indexuri secundare, definite pe atributele care intervin cel mai frecvent în clauza WHERE din interogări