6. Uniuni de tabele

Pentru a putea selecta date din două sau mai multe tabele trebuie să unim acele tabele, deci, introducem noţiunea de uniune de tabele. Întrucât ne ocupăm de baze de date relaţionale, ştim că tabelele din baza de date sunt legate între ele (sunt relaţionate). Nu este indicat să avem tabele izolate într-o bază de date. Pe parcursul acestui curs am făcut referire în mai multe rânduri la relaţionarea dintre tabelele unei baze de date. De asemenea, au fost prezentate şi tipurile de relaţii posibile între tabele.

Aşa cum am spus, extragerea datelor din două sau mai multe tabele ale unei baze de date se realizează prin crearea unei uniuni între aceste tabele. Uniunile dintre tabele poartă şi numele de **join**. De altfel, sub această denumire, de **join**, sunt cel mai mult cunoscute.

Rezultatul returnat de o interogare **SELECT** în care se extrag date din mai multe tabele, deci un **join** sau o uniune de **tabele**va fi o singură tabelă. Deci, printr-o singură interogare extragem date din mai multe tabele, iar rezultatul returnat va fi o singură tabelă.

Există multe situaţii în care este necesară extragerea informaţiilor din mai multe tabele. Pentru acest lucru trebuie realizată o relaţie între tabele pe baza unor informaţii comune. Această interogare a bazei de date în care se realizează relaţii între tabele pe baza unor informaţii comune (câmpuri comune, coloane comune) se numeşte **joncţiune** (**JOIN**). Cu alte cuvinte, tabelele sunt unite în cadrul interogărilor de regăsire a datelor.

6.2 Alias-uri de tabele

Pentru tabelele care alcătuiesc uniunea se pot stabili **alias**-uri de nume, sub forma **nume\_tabelă as alias\_nume**, introduse în clauza **FROM** a instrucţiunii **SELECT**. **Alias**-urile pot fi utilizate în orice parte a instrucţiunii **SELECT**. Adresarea unei coloane a unei tabele se va face, atunci când tabela are un **alias**, sub forma **nume\_alias.nume\_coloană**.

Este recomandat să asigurăm nume scurte pentru **alias**-urile tabelelor. Un alt avantaj al **alias**-urilor este că putem realiza auto-uniuni, în cadrul cărora aceeaşi tabelă poate avea **alias**-uri diferite. Acest aspect l-am explicat şi când am prezentat **SELF JOIN**. Utilizarea unui **alias** duce şi la o scurtare a sintaxei **SQL** într-o comandă.

Pe lângă **alias**-urile pentru tabele se mai pot folosi **alias**-uri şi pentru coloanele unei tabele, dar şi pentru câmpuri cu valoare calculată sau pentru câmpuri rezultate prin aplicarea unei funcţii de agregare.

Expresiile sau coloanele dintr-o interogare **SELECT**, vor trebui să conţină numele coloanelor din tabele, dacă nu există ambiguități ce pot crea confuzii (coloane care au în ambele tabele acelaşi nume) sau, în caz contrar, adică în caz că există confuzii, vor conține numele tabelelor sau **alias**-urile acelor tabele, din care fac parte coloanele, urmate de caracterul „**.**” (**punct**) şi de numele coloanelor.

# 6.3 Tipuri de join-uri

Există mai multe *tipuri de join-uri*. O clasficare a lor s-ar putea face în două mari categorii:

·         **JOIN-uri fără condiții** sau **asocieri fără restricții** (sau **fără clauza WHERE**dacă ne referim la **condiția de asociere** din **clauza WHERE** a interogării);

·         **JOIN-uri cu condiții** sau **asocieri cu restricții** (sau **cu clauză WHERE**dacă ne referim la **condiția de asociere** din **clauza WHERE** a interogării).

# 6.4 Asocieri de tabele fără restricții (CROSS JOIN)

În prima categorie, a *asocierilor de tabele fără restricții* sau a *joncțiunilor fără condiții*, avem **joncțiunea încrucișată** (sau **CROSS JOIN**) în care sunt extrase coloane din două sau mai multe tabele, fără însă a avea specificate condiții de egalitate pe coloanele comune acestor tabele. *Acest tip de****join****va returna****produsul cartezian****obținut din încrucișarea înregistrărilor din acele tabele specificate în clauza****FROM****a****instrucțiunii SELECT****.*

Astfel, observăm că acest **tip de join** *nu este unul util*, *el trebuie evitat pentru că obținerea unui produs cartezian nu duce la un rezultat care să satisfacă anumite condiții ce au impus utilizarea asocierilor de tabele (JOIN-urilor)*.

Practic, o *instrucțiune****SELECT****fără condiții de egalitate pe coloana comună* (*condiția de realizare a JOIN-ului*) care extrage date din două tabele ale unei baze de date *va duce la obținerea unui produs cartezian*, adică, *fiecare înregistrare din prima tabelă este combinată (încrucișată, împerecheată) cu toate înregistrările din cea de-a doua tabelă*.

La uniunile de tip produs cartezian se realizează o selecţie în care se trec toate câmpurile pe care dorim să le obţinem, din cadrul ambelor tabele, iar la clauza **FROM** se trec ambele tabele, fiecare dintre ele putând avea definit şi un **alias**, adică un nume mai scurt prin care pot fi accesate.

Nu există o limită în ceea ce priveşte numărul de tabele ce pot fi unite în cadrul unei instrucţiuni **SELECT**. Aşadar, putem avea două sau oricât de multe tabele unite printr-o **joncţiune**. Însă, cu cât avem mai multe tabele legate într-un **join**, iar tabelele sunt populate cu date numeroase, timpul de răspuns la o anumită solicitare poate fi mai mare.

*Această asociere se mai numește și****CARTESIAN JOIN****, adică****asociere tip produs cartezian****.*

Sintaxa unei asocieri de tabele fără restricții (CROSS JOIN este următoarea):

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1]

**CROSS** **JOIN** table2 [**AS** t2];

O altă formă a instrucțiunii **SELECT** care realizează o asociere de tip cartezian este cea în care tabelele sunt enumerate și separate prin virgulă în clauza **FROM** a instrucțiunii **SELECT**, dar **lipsește clauza WHERE** a instrucțiunii (în care ar putea să apară condiția de asociere, de unire). De aici și denumirea de **JOIN fără clauză WHERE** sau fără condiții de asociere:

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1], table2 [**AS** t2];

6.5 Asocieri de tabele cu restricții

A doua categorie de *JOIN-uri*, după cum am realizat clasificarea de mai sus, și anume, *JOIN-uri cu clauză WHERE*, sau *cu condiții de asociere între tabele* sunt cele care *au specificate condițiile de egaliate între coloanele comune ale tabelelor extrase în clauza****FROM****a instrucțiunii****SELECT***.

*Condițiile de egalitate* (sau de realizarea a **join**-ului) pot fi plasate în cadrul *clauzei****WHERE*** (de unde și această clasificare simplistă în jocțiuni *cu clauză WHERE* și *fără clauză WHERE*) dacă se folosește această sintaxă pentru realizarea **join**-ului (**sintaxa cu clauză WHERE este utilizată de toate SGBD-urile**) sau în **clauza ON** dacă se folosește **sintaxa de ANSI SQL** (adică standardul general valabil acceptat și recunoscut de toate sistemele gestiune a bazelor de date (**SGBD**) bazate pe **standardul SQL** – **Structured Query Language**).

# 6.5.1 Clasificare

În această categorie, a **asocierilor de tabele cu restricții (join-uri cu condiții de asociere)**, distingem următoarele **tipuri de join-uri**:

·         **INNER JOIN** (**joncțiunea simplă**);

·         **OUTER JOIN** (**joncțiunea externă**);

·         **SELF JOIN** (**auto-uniunea** sau *joncțiunea unei tabele cu ea însăși*).

Toate aceste tipuri de **join**-uri au fost prezentate pe larg în prima parte a cursului în care au fost implementate și foarte multe exemple pentru înțelegerea fiecărui *tip de* *join* în parte. În acest modul vom trece în revistă, succint, aceste *joncțiuni,* scopul fiind acela de a ne reaminti ce reprezintă fiecare tip în parte, cum și în ce context se folosește.

# 6.6 INNER JOIN

*Cel mai folosit tip de****join****este reprezentat de****joncțiunea simplă****sau****INNER JOIN****care reprezintă****selectarea înregistrărilor comune tabelelor****specificate în clauza****FROM****a instrucțiunii****SELECT****.*

*Este vorba, practic, de****intersecția valorilor****din tabelele care sunt unite (deci, valorile comune).*Dacă sunt linii (înregistrări) în prima tabelă care nu au corespondent în cealaltă, ele nu sunt extrase (selectate).

În continuare va fi prezentată o **reprezentare grafică a acestui tip de join** (**INNER JOIN**), pentru o înțelegere mai bună a ceea ce reprezintă.

Există mai multe forme în care poate fi utilizată instrucțiunea **SELECT** în care se realizează **INNER JOIN** între două sau mai multe tabele ale unei baze de date relaționale.

# 6.6.1 Forme posibile de utilizare INNER JOIN

Avem, astfel, **sintaxa standard ANSI SQL**, sintaxă care **funcționează** în **orice** sistem de gestiune a bazelor de date (**SGBD**):

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1]

[**INNER**] **JOIN** table2 [**AS** t2]

**ON** table1.column\_name=table2.column\_name;

*În cadrul sintaxei unei instrucțiuni SQL, părțile cuprinse între paranteze drepte sunt opționale.* Așadar, **tabelele din care se extrag informații pot primi sau nu alias-uri iar cuvântul cheie INNER este optional**. *Este suficient să avem cuvântul cheie****JOIN****intre tabelele relaționate și se subînțelege că avemde-a face cu un****INNER JOIN***.

*Alias-urile de tabele sunt folosite, în general, ca prescurtări ale numelor tabelelor, fiind utile atunci când este necesară prefixarea anumitor coloane cu numele tabelei din care face parte.*

De asemenea, realizarea unei joncțiuni (**INNER JOIN**) se poate face și prin scrierea condiției de egalitate între coloanele comune (*condiția care realizează JOIN-ul, relaționarea*) *în cadrul clauzei****WHERE***.

În acest caz, avem sintaxa următoare:

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1], table2 [**AS** t2]

**WHERE** table1.column\_name=table2.column\_name;

Mai există o formă prin care se poate scrie instrucțiunea **SELECT** în care se realizează **join**-ul între tabele, utilizând **cuvântul cheie USING** *urmat de numele coloanei de legătură între tabele* (este vorba de coloană ce conține valori comune în cele 2 tabele).

**Condiția necesară pentru ca să poată fi utilizată această formă a instrucțiunii SELECT este ca numele coloanei de legătură între tabele să fie același în ambele tabele.**

De altfel, *această formă a instrucțiunii****SELECT****simplifică scriere*a în cazul în care acea coloană comună are același nume în tabelele relaționate.

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1]

**JOIN** table2 [**AS** t2]

**USING** (column\_name);

Am prezentat *sintaxa implementării unui****INNER JOIN****pentru două tabele*. *Se poate realiza asocierea mai multor tabele în cadrul unei****singure instrucțiuni SELECT****, sintaxa modificând-se doar prin adăugarea celorlalte tabele și condiții de egalitate între coloanele tabelelor*, dar modul de gândire a instrucțiunii și de realizare a asocierilor de tabele fiind același.

# 6.7 OUTER JOIN

*Al doilea tip de join este reprezentat de****joncțiunea externă****sau****OUTER JOIN*** care, la rândul ei, se împarte în *două tipuri de****OUTER JOIN***:

·         **LEFT OUTER JOIN**;

·         **RIGHT OUTER JOIN**.

*Cuvântul cheie****OUTER poate fi omis****în ambele tipuri de joncțiuni externe, fiind suficient să specifcăm, ca tip de join,****LEFT JOIN****sau****RIGHT JOIN****, subînțelegându-se faptul că este vorba de****OUTER JOIN****.*

# 6.7.1 LEFT OUTER JOIN

***LEFT JOIN*** *reprezintă uniunea de tabele în care sunt returnate atât****înregistrările din prima tabelă****(tabela din stânga, prima în ordinea de scriere a tabelelor în instrucțiunea****SELECT****)****care au corespondent în cea de-a doua tabelă****, cât și cele****care nu au corespondent în cea de-a doua tabelă****.* În cazul în care ***înregistrările din prima tabelă nu au corespondent în cea de-a doua***, pentru **coloanele extrase din a doua tabelă se va returna NULL**.

Așadar, în cazul joncțiunilor externe*contează****ordinea****în care sunt scrise tabelele în cadrul instrucțiunii****SELECT***. De altfel, din punct de vedere al *optimizării instrucțiunii****SELECT***, și în cazul **INNER JOIN** contează *ordinea tabelelor* în instrucțiunea de realizare a *join-ului*.

Astfel, *indiferent de ordinea de scriere a tabelelor*, în cazul **INNER JOIN**, se va *returna același rezultat*, dar optimizarea constă în faptul că, *dacă prima (deci, cea din partea stângă) este trecută tabela cu mai puține înregistrări, atunci la realizarea join-ului este parcursă integral această tabelă, urmând ca din a doua tabelă (cea din dreapta) să fie preluate doar valorile corespondente.*

În cazul în care este trecută ca *primă tabelă cea cu mai multe înregistrări, rezultatul va fi același, doar că va crește timpul de execuție al instrucțiunii deoarece sunt parcurse toate înregistrările din această tabelă* mai mare, urmând să fie luate valorile corespondente *din tabela cu mai puține înregistrări.*

*Evident, timpul de execuție mai mare se va reflecta în cazul în care tabelele sunt foarte mari, conțin foarte multe înregistrări*. Dacă tabelele au dimensiune mică (conțin un număr mic de înregistrări) diferențele în ceea ce privește timpul de execuție sunt nesemnificative.

În termeni de teoria mulţimilor, considerând cele două tabele ca fiind două mulţimi **A** şi **B**, **LEFT JOIN** între cele două tabele (**A** şi **B**) reprezintă **(A \ B) U (A∩ B)**. Deci, **LEFT JOIN**reprezintă rezultatul diferenţei dintre cele două mulţimi reunit cu rezultatul intersecţiei celor două mulţimi.

Prezentăm în continuare **sintaxa ANSI SQL** a instrucțiunii **LEFT OUTER JOIN**, sintaxă ce funcționează în orice sistem de gestiune a bazelor de date:

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1]

**LEFT** [**OUTER**] **JOIN**

table2 [**AS** t2]

**ON** table1.column\_name=table2.column\_name;

La fel ca în cazul **INNER JOIN**, și la folosirea unei *joncțiuni externe*, *cuvântul cheie****OUTER****poate fi omis din sintaxa instrucțiunii,* prin specificarea termenilor **LEFT JOIN** sau **RIGHT JOIN** se subînțelege că este vorba despre o joncțiune externă de tip **LEFT OUTER JOIN** sau **RIGHT OUTER JOIN**.

*Observăm faptul că, în****MySQL****, nu putem scrie acest tip de join utilizând forma instrucțiunii****SELECT****cu clauză****WHERE***, deoarece într-o astfel de formă nu se poate specifica faptul că este vorba despre o joncțiune externă.

Avem mai jos și ***reprezentarea grafică*** a joncțiunii de tip **LEFT OUTER JOIN** în scopul de a înțelege mai bine, din această reprezentare, ce înregistrări întoarce acest join:

# 6.7.2 RIGHT OUTER JOIN

În mod similar, **RIGHT JOIN** reprezintă toate înregistrările din tabela din partea dreaptă (a doua tabelă) care au corespondent sau nu în tabela din partea stângă (prima tabelă).

Cu alte cuvinte, **RIGHT JOIN** returnează acelaşi rezultat, cu deosebirea că se vor extrage înregistrările din a doua tabelă care au sau nu înregistrări în prima. Este, practic, reciproca joncţiunii **LEFT JOIN**.

**RIGHT JOIN** este mult mai puţin folosită deoarece se poate transforma foarte simplu într-un **LEFT JOIN** prin schimbarea ordinii celor două tabele. De aceea, forma cea mai utilizată a joncţiunilor externe este **LEFT JOIN**.

Instrucțiunea **RIGHT OUTER JOIN** este foarte asemănătoare **cu LEFT OUTER JOIN**. *În cazul acestei instrucțiuni vor fi extrase din****tabela din dreapta****(deci, a doua în ordinea de scriere a tabelelor în clauza****FROM****a instrucțiunii****SELECT****) atât****înregistrările care au corespondent în prima tabelă****(tabela din stânga), cât și cele care****nu au corespondent******în prima tabelă****.*

**Sintaxa** **ANSI SQL** a instrucțiunii **RIGHT OUTER JOIN** este următoarea:

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1]

**RIGHT** [**OUTER**] **JOIN**

table2 [**AS** t2]

**ON** table1.column\_name=table2.column\_name;

La fel ca în situația **LEFT OUTER JOIN**, nici instrucțiunea **SELECT** ce va conține o *joncțiune externă* de tip **RIGHT OUTER JOIN** *nu poate fi implementată*, în **MySQL** *prin utilizarea clauzei****WHERE****pentru stabilirea condițiilor de asociere între tabele*.

***Reprezentarea grafică*** a *joncțiunii externe* de tip **RIGHT OUTER JOIN** este redată în continuare pentru a înțelege mai bine ce înregistrări întoarce acest tip de *join*:

*Așadar, se observă faptul că o asociere de tabele de tip****RIGHT OUTER JOIN****se poate transforma foarte ușor într-o asociere de tabele de tip****LEFT OUTER JOIN****prin****inversarea******ordinii tabelelor****în clauza****FROM****a instrucțiunii****SELECT****.*

*Este valabilă și reciproca, adică, o joncțiune de tip****LEFT OUTER JOIN****se poate transforma într-o joncțiune de tip****RIGHT OUTER JOIN****prin****inversarea ordinii tabelelor****în clauza****FROM****a instrucțiunii****SELECT****.*

# 6.8 SELF JOIN

Sunt situații în care, pentru a putea extrage anumite informații dintr-o tabelă, *este nevoie să se realizeze o uniune a acelei tabele cu ea însăși*. Avem, astfel, un alt tip de join, și anume **SELF JOIN**, adică ***realizarea unei asocieri a unei tabele cu ea însăși***. *Acest tip de uniune de tabele mai poartă numele de****auto-uniune****.*

**SELF JOIN**-ul este, de fapt, tot un **INNER JOIN**sau un**OUTER JOIN**, dar în care, *în clauza****FROM****avem o singură tabelă, care pentru a putea fi diferențiată va primi****alias****-uri diferite*. Este vorba, deci, despre o*redenumire temporară a tabelei în cadrul instrucțiunii****SELECT***.

**Sintaxa ANSI SQL** a unei joncțiuni de tip **SELF JOIN** *este similară cu sintaxa unei joncțiuni* de tip **INNER JOIN**sau**OUTER JOIN**, deosebirea fiind aceia că *o singură tabelă, redenumită temporar prin utilizarea de alias-uri, participă la realizarea join-ului*:

**SELECT** column\_name(s)

**FROM** table1 **AS** t1

[**INNER**] **JOIN** | **LEFT** [**OUTER**] **JOIN** | **RIGHT** [**OUTER**] **JOIN**

table1 **AS** t2 **ON** t1.column\_name=t2.column\_name;

Se observă, așa cum am precizat anterior, că este tot un **INNER JOIN**sau un**OUTER JOIN (LEFT**sau**RIGHT)** în care, însă, fiind vorba de *participarea unei singure tabele la realizarea asocierii de tabele* (joncțiunii**) prezența alias-urilor pentru tabele în cadrul instrucțiunii SELECT devine obligatorie**.

La implementarea unei auto-uniuni, mai poate fi întâlnită următoarea formă a instrucțiunii **SELECT**, formă în care *condiția de egalitate (adică, de realizare a asocierii) va fi plasată în clauza****WHERE*** (formă posibilă, în **MySQL**, doar pentru implementarea **INNER JOIN**):

**SELECT** column\_name(s)

**FROM** table1 [**AS** t1], table1 [**AS** t2]

**WHERE** t1.column\_name=t2.column\_name;

Pentru o mai bună înțelegere, prezentăm și o **reprezentare grafică** a acestui tip de join. După cum se observă, ***reprezentarea grafică****a****SELF JOIN****este aproape identică cu cea de la****INNER JOIN***(în mod similar, reprezentarea pentru o joncțiune externă unei tabele cu ea însăși este cea prezentată la **LEFT**, respectiv, **RIGHT JOIN**, deosebirea fiind dată de daptul că ambele mulțimi (tabele) participante la joncțiune au același nume (pentru că e vorba de aceeași tabelă) dar alias-uri diferite), deosebirea constă în faptul că ambele mulțimi (tabele) participante la realizarea asocierii sunt de fapt una și aceași, dar ele primesc un alias diferit pentru a permite identificarea lor în cadrul asocierii (join-ului):

# 6.9 Reuniuni. UNION și UNION ALL

Majoritatea interogărilor conţin o singură instrucţiune **SELECT**, care returnează date din una sau mai multe tabele. Dar este permisă şi efectuarea mai multor interogări şi returnarea rezultatelor sub forma unui singur set de rezultate ale interogării. Aceste interogări combinate se numesc reuniuni sau interogări compuse.

Pentru a combina rezultatele obținute în urma execuției a două sau mai multe instrucțiuni **SELECT**se utilizează operatorul **UNION**. O astfel de instrucțiune *va concatena (alipi) rezultatele returnate de fiecare instrucțiune****SELECT****în ordinea în care apar în cadrul instrucțiunii compuse ce utilizează****UNION***. Se realizează, de fapt, o *reuniune a rezultatelor instrucțiunilor****SELECT****participante la instrucțiunea compusă*.

Dacă interogările vor genera înregistrări duplicate, atunci, în urma utilizării operatorului **UNION**, *rândurile care se repetă (sunt duplicate) vor fi eliminate*. Deci, funcționează întocmai ca operația de reuniune pe mulțimi. *Este posibilă păstrarea înregistrărilor duplicate prin folosirea****UNION ALL****în loc de****UNION****între interogările componente ale comenzi compuse*.

În cazul utilizării operatorilor **UNION** sau **UNION ALL**, *este necesar* ca instrucțiunile **SELECT** conținute de comanda compusă *să returneze același număr de coloane*. Altfel, dacă instrucțiunile **SELECT** vor genera un număr diferit de coloane, se va întoarce o eroare.

Instrucțiunile **SELECT** participante la realizarea reuniunii pot fi oricât de complexe. *Ele pot conține toate clauzele cunoscute ale comenzii****SELECT***.

*Excepție face clauza****ORDER BY****care poate fi plasată doar la****finalul****instrucțiunii compuse, deci, la finalul ultimei instrucțiuni****SELECT****, deoarece clauza****ORDER BY****realizează o ordonare a rezultatelor afișate, iar ordonarea trebuie să fi pentru rezultatele obținute după aplicarea operatorului****UNION****.*

Instrucțiunile **SELECT** participante la o astfel de comandă compusă pot conține și *asocieri de tabele* (*join*-uri).

Deci, o reuniune trebuie să fie alcătuită din două sau mai multe instrucţiuni **SELECT**, separate prin cuvântul cheie **UNION**sau **UNION ALL**, iar fiecare instrucţiune **SELECT** trebuie să conţină acelaşi număr de coloane, expresii sau funcţii agregat, dar nu contează ordinea în care sunt specificate coloanele. Coloanele din instrucţiunile **SELECT** trebuie să aibă tipuri de date compatibile.

# 6.9.1 Operatorul UNION

Operatorul **UNION** elimină în mod automat toate rândurile duplicate din setul de rezultate al interogării. Deci, dacă sunt înregistrări returnate de mai multe instrucţiuni **SELECT** dintr-o reuniune, este afişată o singură dată această înregistrare.

Sintaxa unei instrucţiuni în care se realizează o reuniune a datelor din două tabele prin utilizarea operatorului **UNION** este următoarea:

**SELECT** column\_name(s) **FROM** table1 [**AS** t1] […]

**UNION**

**SELECT** column\_name(s) **FROM** table2 [**AS** t1] [...];

6.9.2 Operatorul UNION ALL

Operatorul **UNION** **ALL** nu elimină rândurile duplicate din setul de rezultate al interogării. Deci, dacă sunt înregistrări returnate de mai multe instrucţiuni **SELECT** dintr-o reuniune în care s-a folosit operatorul **UNION ALL**, vor fi afișate toate înregistrările .

**Sintaxa** unei comenzi ce conține operatorul **UNION ALL** este similară:

**SELECT** column\_name(s) **FROM** table1 [**AS** t1] […]

**UNION ALL**

**SELECT** column\_name(s) **FROM** table2 [**AS** t1] [...];

# 6.9.3 Utilizarea clauzei ORDER BY într-o instrucțiune compusă de reuniune

Pentru ordonarea rezultatelor obţinute în urma efectuării unei reuniuni, clauza **ORDER BY** se trece la final, după ultima instrucţiune **SELECT**.

Reluăm sintaxa instrucţiunii de reuniune de interogări, adăugând şi clauza **ORDER BY**:

**SELECT** column\_name(s) **FROM** table1 [**AS** t1] […]

**UNION | UNION ALL**

**SELECT** column\_name(s) **FROM** table2 [**AS** t1] [...]

**ORDER BY** column\_name(s) [**ASC** | **DESC**] ;

# 6.10 Concluzii

În această lecţie am abordat noţiunile de **joncţiune** sau **uniune de tabele** şi **reuniune**. Am prezentat tipurile de joncţiuni (**join**-uri) care se pot întâlni în **SQL**, şi, de asemenea am discutat despre reuniunea interogărilor folosind operatorul **UNION,**respectiv**, UNION ALL**.

Aşadar, începând cu această lecţie am început să expunem lucruri mai complexe din domeniul bazelor de date. Lecţia următoare continuă să abordeze teme cu o complexitate mai mare, şi anume, vom prezenta conceptul de **subinterogare**și **tipurile de subinterogări**.

## Tema Sedinta 6

În fișierul de la link-ul **Baza de Date classicmodels de la Ședința 5**sunt instrucțiunile pentru crearea bazei de date **classicmodels** și a tabelelor din această bază de date.

**Cerințe**

1. Rulați instrucțiunile din fișier pentru crearea bazei de date și a tabelelor.
2. Scrieți o instrucțiune **SQL** prin care să **afișați numărul de angajați din fiecare birou** (**câmpuri afișate:** cod birou (**officeCode**), oraș (**city**) și **număr de angajați**).
3. Scrieți o instrucțiune **SQL** prin care să **afișați comenzile cu statusul Shipped efectuate de clienții din Franța (France) în anii 2004** **și 2005** (**câmpuri afișate:** nume client într-o singură coloană (**contactLastName+contactFirstName**), țară (**country**), data comenzii (**orderDate**) și statusul comenzii (**status**)).
4. Scrieți o instrucțiune **SQL** prin care să **afișați cantitatea totală comandată pentru comenzile anulate (cu status Cancelled)** (**câmpuri afișate:** total cantitate comandată și statusul comenzii**(status)**).
5. Scrieți o instrucțiune **SQL** prin care să **afișați produsele comandate în luna mai 2005; eliminați duplicatele** (**câmpuri afișate:** cod produs (**productCode**), denumire produs (**productName**), **luna din data comenzii**(**luna din orderDate**)**și** **anul din data comenzii**(**anul din orderDate**).
6. Scrieți o instrucțiune **SQL** prin care să **afișați totalul comenzilor produselor pentru fiecare furnizor** (**câmpuri afișate:** nume furnizor (**productVendor**) și **total comenzi; total comenzi: totalizarea preț\*cantitate comandată: priceEach\*quantityOrdered**).

**Rezultatele vor fi ordonate descrescător după totalul comenzilor, iar totalul va fi rotunjit la 2 zecimale.**

1. Scrieți o instrucțiune **SQL** prin care să **afișați toți clienții fără comenzi efectuate** (**câmpuri afișate:** nume client (**contactLastName+contactFirstName**), oraș (**city**), țară (**country**) și **număr comenzi**.