**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**RAPORT**

Lucrarea de laborator nr.8

Disciplina: *Baza de date*

Tema**:** Interogări cu joncțiuni și cu operatori din teoria mulțimilor

**A efectuat: Vlasitchi Stefan**

**st. gr. TI-216**

**A verificat:**

**asist. univ. Cebotar Gabriela**

**Chişinău - 2023**

**Lucrare de laborator nr. 8**

**Sarcini practice:**

## 

## Efectuarea sarcinilor

**Sarcina 1**

**INNER JOIN** și **LEFT OUTER JOIN** sunt două tipuri de operații de îmbinare (join) folosite în interogările SQL pentru a combina rândurile din două sau mai multe tabele. Iată o explicație a diferențelor dintre acestea:

**INNER JOIN** returnează numai rândurile care au cel puțin o corespondență în ambele tabele. Dacă nu există nicio potrivire între valorile cheie ale celor două tabele, acele rânduri nu vor fi incluse în rezultat. Sintaxa este simplă: `SELECT coloane FROM tabela1 INNER JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;`

Exemplu:

SELECT employees.employee\_id, employees.employee\_name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

**LEFT OUTER JOIN** returnează toate rândurile din tabela din stânga și rândurile corespunzătoare din tabela din dreapta.Dacă nu există nicio potrivire pentru o anumită linie din tabela din stânga, atunci coloanele din tabela din dreapta vor avea valori nule în rezultat. Sintaxa: `SELECT coloane FROM tabela1 LEFT OUTER JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;` sau forma echivalentă `SELECT coloane FROM tabela1 LEFT JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;`

Exemplu:

SELECT employees.employee\_id, employees.employee\_name, departments.department\_name

FROM employees

LEFT OUTER JOIN departments ON employees.department\_id = departments.department\_id;

În esență, `INNER JOIN` returnează doar rândurile care au potrivire în ambele tabele, în timp ce `LEFT OUTER JOIN` returnează toate rândurile din tabela din stânga, împreună cu rândurile corespunzătoare din tabela din dreapta, sau valori nule acolo unde nu există potrivire.

**Sarcina 2**

**LEFT OUTER JOIN** returnează toate rândurile din tabela din stânga și rândurile corespunzătoare din tabela din dreapta. Dacă nu există nicio potrivire pentru o anumită linie din tabela din stânga, atunci coloanele din tabela din dreapta vor avea valori nule în rezultat. Sintaxa: `SELECT coloane FROM tabela1 LEFT OUTER JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;` sau forma echivalentă `SELECT coloane FROM tabela1 LEFT JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;`

Exemplu:

SELECT employees.employee\_id, employees.employee\_name, departments.department\_name

FROM employees

LEFT OUTER JOIN departments ON employees.department\_id = departments.department\_id;

**RIGHT OUTER JOIN** returnează toate rândurile din tabela din dreapta și rândurile corespunzătoare din tabela din stânga. Dacă nu există nicio potrivire pentru o anumită linie din tabela din dreapta, atunci coloanele din tabela din stânga vor avea valori nule în rezultat. Sintaxa: `SELECT coloane FROM tabela1 RIGHT OUTER JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;` sau forma echivalentă `SELECT coloane FROM tabela1 RIGHT JOIN tabela2 ON tabela1.col\_key = tabela2.col\_key;`

Exemplu:

SELECT employees.employee\_id, employees.employee\_name, departments.department\_name

FROM employees

RIGHT OUTER JOIN departments ON employees.department\_id = departments.department\_id

În esență, `LEFT OUTER JOIN` returnează toate rândurile din tabela din stânga, împreună cu rândurile corespunzătoare din tabela din dreapta sau valori nule acolo unde nu există potrivire. În schimb, `RIGHT OUTER JOIN` face același lucru, dar pentru tabela din dreapta. Din punct de vedere practic, majoritatea sistemelor de baze de date permit utilizarea oricărui stil (`LEFT` sau `RIGHT`), dar `LEFT OUTER JOIN` este mai frecvent utilizat și mai ușor de citit.

**Sarcina 3**

O subinterogare, cunoscută și sub numele de subquerie, este o interogare (query) încorporată într-o altă interogare SQL. Subinterogările sunt folosite pentru a obține informații care vor fi utilizate în cadrul condițiilor sau în clauzele `FROM`, `WHERE`, `HAVING`, sau `SELECT` ale unei interogări mai mari. Ele pot fi folosite într-o varietate de moduri și sunt un instrument util pentru a construi interogări complexe.

Există mai multe tipuri de subinterogări, inclusiv:

1. Subinterogare scalară (Scalar Subquery)

O subinterogare scalară returnează un singur rezultat (o singură valoare) și este folosită într-o expresie pentru a furniza o valoare pentru o condiție sau o coloană.

Exemplu:

SELECT employee\_name

FROM employees

WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'IT');

1. Subinterogare în clauza FROM

O subinterogare poate fi folosită în clauza `FROM` pentru a furniza date pentru interogarea principală.

Exemplu:

SELECT department\_name, total\_salaries

FROM (SELECT department\_id, SUM(salary) AS total\_salaries FROM employees GROUP BY department\_id) AS department\_totals;

1. Subinterogare în clauza IN

O subinterogare în clauza `IN` este folosită pentru a verifica dacă o valoare este inclusă în rezultatul unei alte interogări.

Exemplu:

SELECT employee\_name

FROM employees

WHERE department\_id IN (SELECT department\_id FROM departments WHERE location = 'New York');

Subinterogările permit scrierea de interogări mai complexe și mai flexibile, deoarece rezultatul unei subinterogări poate fi utilizat dinamic în interogarea principală. Cu toate acestea, este important să se acorde atenție performanței atunci când se lucrează cu subinterogări, deoarece acestea pot afecta timpul de răspuns al interogării în funcție de volumul de date și de structura interogării.

**Sarcina 4**

Construirea subinterogărilor în SQL implică respectarea unor reguli specifice pentru a obține rezultate precise și pentru a evita erorile. Iată câteva reguli importante pentru construirea subinterogărilor:

1. Sintaxa Corectă:

Asigurați-vă că subinterogarea respectă sintaxa SQL corectă. Ea trebuie să fie o interogare validă pe cont propriu.

2. Numărul de Rezultate:

Subinterogările trebuie să returneze un număr specific de rezultate, în funcție de contextul în care sunt utilizate. De exemplu, o subinterogare scalară ar trebui să returneze exact o valoare.

3. Locurile de Utilizare:

Subinterogările pot fi utilizate în mai multe locuri într-o interogare, cum ar fi în clauza `WHERE`, `FROM`, `SELECT`, `HAVING`, sau chiar în clauza `VALUES` (în cazul subinterogărilor cu predicatul `IN`).

4. Compatibilitatea Tipurilor de Date:

Asigurați-vă că tipurile de date returnate de subinterogare sunt compatibile cu coloanele sau valorile cu care sunt comparate în interogarea principală.

5. Aliase pentru Subinterogări în Clauza FROM:

Dacă utilizați o subinterogare în clauza `FROM`, este recomandat să folosiți un alias pentru a face codul mai lizibil.

6. Limitarea Rezultatelor:

Subinterogările trebuie să fie scrise astfel încât să returneze un număr limitat de rezultate. De exemplu, o subinterogare folosită într-o clauză `IN` ar trebui să returneze o listă de valori.

7. Împărțirea Subinterogărilor Mai Mari:

Dacă o subinterogare este complexă, împărțiți-o în subinterogări mai mici și mai ușor de înțeles. Acest lucru face codul mai modular și mai ușor de întreținut.

8. Optimizarea Performanței:

Luați în considerare optimizarea performanței subinterogărilor, deoarece acestea pot afecta timpul de răspuns al interogării. Utilizați indexuri și evaluați impactul subinterogărilor asupra performanței.

Respectarea acestor reguli vă va ajuta să construiți subinterogări eficiente și să evitați problemele comune legate de sintaxă și logica interogărilor în SQL.

**Sarcina 5**

O subinterogare scalară este o formă specifică de subinterogare în SQL care returnează un singur rând și o singură coloană, adică un singur rezultat scalar. Aceasta înseamnă că subinterogarea furnizează o singură valoare și poate fi utilizată într-o expresie pentru a obține sau verifica o valoare specifică într-o interogare mai mare.

Caracteristici cheie ale subinterogării scalare:

1. Returnează o singură valoare:

Subinterogarea scalară furnizează un singur rezultat, indiferent de numărul de înregistrări pe care le examinează.

1. Poate fi utilizată în expresii:

De obicei, este folosită într-o expresie pentru a furniza o valoare pentru o condiție, o coloană sau o operație.

Exemplu de subinterogare scalară într-o clauză `WHERE`:

SELECT employee\_name

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

În acest exemplu, subinterogarea scalară `(SELECT AVG(salary) FROM employees)` furnizează o valoare scalară, respectiv media salariilor, și este utilizată pentru a compara salariul fiecărui angajat în interogarea principală.

Subinterogările scalare sunt utile pentru a efectua comparații, filtrări și alte operații care implică manipularea a unei singure valori returnate de subinterogare.

**Sarcina 6**

Interogările corelate și necorelate sunt două tipuri diferite de subinterogări în SQL. Diferența principală între ele se referă la modul în care subinterogarea este legată de interogarea principală și cum sunt evaluate. Iată explicații pentru fiecare:

1. Interogări Necorelate:

Într-o interogare necorelată, subinterogarea este independentă de rândurile returnate de interogarea principală. Subinterogarea este evaluată o singură dată și rezultatul său este folosit în interogarea principală. De obicei, interogările necorelate sunt mai eficiente, deoarece sunt evaluate o singură dată și nu depind de rândurile exterioare ale interogării principale.

Exemplu de interogare necorlată:

SELECT employee\_name

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

În acest exemplu, subinterogarea `(SELECT AVG(salary) FROM employees)` este necorlată, deoarece nu depinde de niciun rând specific din tabela principală `employees`.

1. Interogări Corelate:

Într-o interogare corelată, subinterogarea este dependentă de rândurile returnate de interogarea principală. Subinterogarea este evaluată pentru fiecare rând returnat de interogarea principală. De obicei, interogările corelate sunt mai puțin eficiente decât cele necorelate, deoarece subinterogarea este evaluată repetat pentru fiecare rând al interogării principale.

Exemplu de interogare corelată:

SELECT department\_name,

(SELECT COUNT(\*) FROM employees WHERE employees.department\_id = departments.department\_id) AS num\_employees

FROM departments;

În acest exemplu, subinterogarea `(SELECT COUNT(\*) FROM employees WHERE employees.department\_id = departments.department\_id)` este corelată, deoarece depinde de rândurile returnate de interogarea principală care sunt departamentele.

În general, dacă este posibil, este recomandat să utilizați interogările necorelate pentru a obține performanță mai bună. Cu toate acestea, în anumite situații, interogările corelate pot fi singura opțiune viabilă pentru a obține rezultatele dorite, cum ar fi atunci când aveți nevoie de informații specifice legate de fiecare rând al interogării principale.

**Sarcina 7**

Operatorul **UNION** în SQL este folosit pentru a combina rezultatele a două sau mai multe interogări și a le returna ca un singur set de rezultate distincte. Acesta se utilizează atunci când doriți să obțineți toate rândurile unice care apar în rezultatele interogărilor specificate

**Sarcina 8**

Utilizarea operatorului `UNION` în SQL este supusă unor restricții și cerințe specifice pentru a asigura funcționarea corectă și coerentă a interogărilor. Iată câteva dintre restricțiile asociate cu utilizarea operatorului `UNION`:

1. Număr și Tipuri de Coloane:

Interogările combinate cu `UNION` trebuie să aibă același număr de coloane în rezultate și aceste coloane trebuie să fie de același tip sau să fie convertibile implicit.

1. Ordinea Coloanelor:

Coloanele în rezultatele interogărilor combinate trebuie să fie ordonate în aceeași ordine.

1. Duplicarea Coloanelor:

Coloanele în rezultatele combinate nu trebuie să fie duplicate. Dacă aveți nevoie să păstrați duplicatele, utilizați `UNION ALL`.

1. Compatibilitatea Tipurilor de Date:

Tipurile de date ale coloanelor care sunt implicate în `UNION` trebuie să fie compatibile sau să poată fi convertite implicit. De exemplu, trebuie să puteți converti un tip de date în altul fără pierderea informației.

1. Ordinea Rezultatelor:

Rezultatele în setul combinat sunt returnate în ordine aleatoare, cu excepția cazului în care utilizați o clauză `ORDER BY` la sfârșitul întregului set de rezultate.

1. Filtrarea Rezultatelor:

Clauza `WHERE` se aplică întregului set de rezultate, nu interogărilor individuale în `UNION`. Dacă aveți nevoie să aplicați condiții specifice pentru fiecare interogare, utilizați paranteze pentru a defini corect aceste condiții.

Exemplu de utilizare corectă a operatorului `UNION`:

SELECT column1, column2 FROM table1

UNION

SELECT column1, column2 FROM table2;

Este important să respectați aceste restricții pentru a evita erorile și pentru a asigura că interogările `UNION` sunt corespunzătoare din punct de vedere logic și structural.

**Sarcina 8**

Pentru a îmbina rezultatele a două sau mai multe instrucțiuni SELECT, fără a elimina rândurile duplicate, se utilizeaza operatorul `UNION ALL`. Acesta este similar cu `UNION`, dar permite menținerea duplicatelor în setul de rezultate combinat.

Exemplu de utilizare a `UNION ALL`:

SELECT column1, column2 FROM table1

UNION ALL

SELECT column1, column2 FROM table2;

În acest exemplu, `UNION ALL` va combina rezultatele interogărilor de la `table1` și `table2`, inclusiv duplicatele.

Este important să observați că `UNION ALL` este mai eficient decât `UNION`, deoarece nu necesită eliminarea duplicatelor, ceea ce poate duce la o performanță mai bună în anumite scenarii. Totuși, dacă doriți să eliminați duplicatele, puteți utiliza operatorul `UNION`.

**Concluzie**

Pentru a combina rezultatele a două sau mai multe instrucțiuni SELECT fără a elimina rândurile duplicate, se utilizează operatorul `UNION ALL`. Acesta păstrează toate înregistrările, inclusiv duplicatelor, oferind eficiență și flexibilitate. Utilizarea sa, precum în exemplul `SELECT column1, column2 FROM table1 UNION ALL SELECT column1, column2 FROM table2;`, contribuie la construirea interogărilor SQL eficiente, evitând procesele inutile de eliminare a duplicatelor și asigurând un rezultat rapid și complet.