GESTIUNEA BAZELOR DE DATE RELAȚIONALE CU LIMBAJUL SQL - STRUCTURED QUERY LANGUAGE

Cursul 4

Cuprins

- Interogarea bazelor de date. Instrucțiunea SELECT
- Strategia de evaluare conceptuală a instrucțiunii SELECT
- Operații de JOIN: join condițional, join extern stânga, join extern dreapta, join complet, join natural, cross join
- Reuniune, intersecție, diferență
- View
- Funcții de agregare
- Clauzele GROUP BY și HAVING
- Funcții de rang
- Clauzele OVER, PARTITION BY, ORDER BY

Instrucțiunea SELECT

- Instrucțiunea **SELECT** se folosește pentru a extrage date din baza de date
- Rezultatul este stocat într-un tabel rezultat numit result-set
- Sintaxa:

  SELECT column_name(s) FROM table_name

  SAU

  SELECT * FROM table_name
- Exemple:

  SELECT * FROM Clienți

  SELECT IDClient, Nume, Prenume FROM Clienți

Instrucțiunea SELECT

- Într-un tabel, unele coloane pot conține valori duplicate
- Atunci când dorim să returnăm doar valorile distincte, folosim cuvântul cheie **DISTINCT**
- Sintaxa:

```
SELECT DISTINCT column_name(s)
FROM table_name
```

- Exemplu:

```
SELECT DISTINCT Localitate FROM Clienți
```

Instrucțiunea SELECT

- Clauza **WHERE** se folosește cu scopul de a filtra înregistrări
- Sunt extrase doar înregistrările care îndeplinesc un anumit criteriu
- Sintaxa:

```
SELECT column_name(s) FROM table_name
WHERE column_name operator value
```

- Exemplu:

```
SELECT IDClient, Nume FROM Clienți
WHERE IDClient =3
```

Instrucțiunea SELECT

- SQL folosește apostrof pentru a delimita valorile de tip text/string
- Valorile numerice nu se delimitează cu apostrof
- Exemple:

```
SELECT IDClient, Nume, Prenume FROM Clienți WHERE Prenume='Anda'
AND Nume='Pop'

SELECT Nume, Prenume, Localitate, Data_nașterii FROM Studenți WHERE
An_înmatriculare=2015
```

Instrucțiunea SELECT

- Operatori care pot fi folosiți în clauza **WHERE**:

Operator	Descriere
=	Egalitate
<>, !=	Inegalitate
>	Mai mare
<	Mai mic
<=	Mai mic sau egal
>=	Mai mare sau egal
!<	Nu mai mic decât
!>	Nu mai mare decât

Instrucțiunea SELECT

- Operatori care pot fi folosiți în clauza **WHERE**:

Operator	Descriere
IN	Într-o mulțime enumerată explicit
NOT IN	În afara unei mulțimi enumerate explicit
BETWEEN	Într-un interval închis
NOT BETWEEN	În afara unui interval închis
LIKE	Ca un șablon
NOT LIKE	Diferit de un șablon

Instrucțiunea SELECT

- Operatorul LIKE este folosit în clauza WHERE pentru a specifica un șablon de căutare într-o coloană
- Sintaxa:

  SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern

- Exemple:

  SELECT * FROM Persoane WHERE oraș LIKE '%s'

  SELECT * FROM Persoane WHERE oraș LIKE 'S%'

  SELECT * FROM Persoane WHERE oraș NOT LIKE 'M%'

Instrucțiunea SELECT

- Putem folosi următoarele caractere pentru șablon:

Caracter	Descriere
_	Înlocuiește un singur caracter
%	Înlocuiește zero sau mai multe caractere
[charlist]	Orice caracter din listă
[^charlist]	Orice caracter care nu este în listă

Instrucțiunea SELECT

Să se returneze toate persoanele pentru care data_nașterii are valoarea diferită de NULL:

```
SELECT * FROM Persoane WHERE data_nașterii IS NOT NULL
```

Să se returneze toate filmele pentru care an_apariție se află în afara intervalului [1980,1988]:

```
SELECT * FROM Filme WHERE an_apariție NOT BETWEEN 1980 AND 1988
```

Să se returneze toate filmele care au nota 9 sau 5:

```
SELECT * FROM Filme WHERE nota=9 OR nota=5
```

Să se returneze numele și orașul tuturor persoanelor pentru care telefon are valoarea NULL:

```
SELECT nume, oraș FROM Persoane WHERE telefon IS NULL
```

Instrucțiunea SELECT

Să se returneze toate persoanele ale căror nume încep cu litera a, b sau c:

```
SELECT * FROM Persoane WHERE nume LIKE '[abc]%'
```

Să se returneze toate persoanele care locuiesc în Cluj-Napoca, Sibiu sau Oradea

```
SELECT * FROM Persoane WHERE oraș IN ('Cluj-Napoca', 'Sibiu', 'Oradea')
```

Să se returneze toate persoanele pentru care data_nașterii este diferită de '2000-12-12'

```
SELECT * FROM Persoane WHERE data_nașterii <> '2000-12-12'
```

Strategia de evaluare conceptuală a instrucțiunii SELECT

- Evaluarea surselor de date, filtrare, join-uri
- Generare grupare, filtrare cu having
- Sortare
- Determinarea valorilor pentru coloane

Operații de JOIN

- Folosim join-uri atunci când trebuie să extragem date din mai multe tabele pe baza unei relații între anumite coloane din aceste tabele într-un singur result-set
- Un join definește modul în care două tabele sunt legate într-o interogare prin:

  - specificarea coloanei din fiecare tabel care urmează a fi folosită pentru a realiza join-ul (de obicei un join specifică un foreign key dintr-un tabel și key-ul asociat din celălalt tabel)

  - specificarea unui operator logic (de exemplu: = sau <>) care va fi folosit pentru a compara valorile din coloane

Operații de JOIN

Tabelul Studenți

sid	nume	email	grupă
1234	Ana	ana@ymail.com	331
1235	Andrei	andrei@gmail.com	332
1236	Mihai	mh@yahoo.com	333

sid	cid	notă
1234	Alg1	9
1235	Alg1	10
1237	Db2	9

Tabelul Note

cid	denumire	credite
Alg1	Algebră 1	7
Db1	Baze de date 1	6
Db2	Baze de date 2	6

Tabelul Cursuri

Operații de JOIN - Tipuri

- Inner Join (Join condițional) – extrage înregistrări când este cel puțin o potrivire în cele două tabele
- Left Outer Join (Join extern stânga) – extrage toate înregistrările din tabelul din partea stângă, inclusiv atunci când nu există potriviri în tabelul din partea dreaptă
- Right Outer Join (Join extern dreapta) – extrage toate înregistrările din tabelul din partea dreaptă, inclusiv atunci când nu există potriviri în tabelul din partea stângă
- Full Outer Join (Join complet) – combinație între Left Outer Join și Right Outer Join
- Natural join (Join natural)
- Cross join

Inner Join

- Extrage înregistrări doar când există cel puțin o potrivire în ambele tabele
- Sintaxa:

  SELECT column_name(s) FROM

  table_name1 INNER JOIN

  table_name2 ON table_name1.column_name =

  table_name2.column_name

Inner Join

- Exemplu – dorim să extragem numele studentului și denumirea cursului doar pentru acei studenți care au note:

```
SELECT S.nume, C.denumire FROM Studenți S
INNER JOIN Note N ON S.sid=N.sid
INNER JOIN Cursuri C ON N.cid=C.cid
```

- Rezultat:

nume	denumire
Ana	Algebră 1
Andrei	Algebră 1

Left Outer Join

- Se mai numește și Left Join
- Extrage toate înregistrările din tabelul din partea stângă, chiar și în cazul în care nu există potriviri în tabelul din partea dreaptă
- Sintaxa:

```
SELECT column_name(s) FROM table_name1

LEFT OUTER JOIN table_name2

ON table_name1.column_name = table_name2.column_name
```

Left Outer Join

- Exemplu - dorim să extragem numele studentului și denumirea cursului pentru toți studenții, indiferent dacă au note la un curs sau nu:

```
SELECT S.nume, C.denumire FROM Studenți S LEFT OUTER JOIN Note N
ON S.sid = N.sid LEFT OUTER JOIN Cursuri C ON N.cid = C.cid
```

- Rezultat:

nume	denumire
Ana	Algebră 1
Andrei	Algebră 1
Mihai	NULL

Right Outer Join

- Se mai numește și Right Join
- Extrage toate înregistrările din tabelul din partea dreaptă, indiferent dacă există sau nu potriviri în tabelul din partea stângă
- Sintaxa:

```
SELECT column_name(s) FROM table_name1

RIGHT OUTER JOIN table_name2

ON table_name1.column_name  =  table_name2.column_name
```

Right Outer Join

- Exemplu - dorim să returnăm toate notele, inclusiv cele introduse din greșeală unor studenți inexistenți:

```
SELECT S.nume, N.notă, C.denumire FROM Studenți S RIGHT OUTER JOIN
Note N ON S.sid = N.sid INNER JOIN Cursuri C ON N.cid = C.cid
```

- Rezultat:

nume	notă	denumire
Ana	9	Algebră 1
Andrei	10	Algebră 1
NULL	9	Baze de date 2

Full Outer Join

- Se mai numește și Full Join
- Este o combinație între Left Outer Join și Right Outer Join
- Extrage toate înregistrările din tabelul din partea stângă și toate înregistrările din tabelul din partea dreaptă, indiferent dacă există sau nu potriviri
- Sintaxa:

```
SELECT column_name(s) FROM table_name1
FULL OUTER JOIN table_name2
ON table_name1.column_name = table_name2.column_name
```

Full Outer Join

- Exemplu – dorim să extragem toți studenții, toate notele și toate cursurile, indiferent dacă există sau nu potriviri:

```
SELECT S.nume, C.denumire FROM Studenți S FULL OUTER JOIN Note N
ON S.sid = N.sid   FULL OUTER JOIN Cursuri C ON N.cid = C.cid
```

- Rezultat:

nume	denumire
Ana	Algebră 1
Andrei	Algebră 1
Mihai	NULL
NULL	Baze de date 2
NULL	Baze de date 1

Join natural

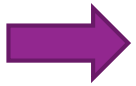
- Creează în mod implicit condiția de potrivire pe baza coloanelor comune (care au acelasi nume în ambele tabele) din cele două tabele implicate în join
- Nu este disponibil în Microsoft SQL Server, dar este disponibil în MySQL și Oracle
- Exemplu:

Tabelul Țări

Cod_țară	Nume_țară
1	România
2	Slovenia

Tabelul Orașe

Cod_oraș	Nume_oraș	Cod_țară
1	Brașov	1
2	Oradea	1



Rezultat Join Natural

Cod_țară	Nume_țară	Cod_oraș	Nume_oraș
1	România	1	Brașov
1	România	2	Oradea

Cross Join

- Un cross join care nu are o clauză WHERE face produsul cartezian al tabelelor implicate
- Sintaxa:

```
SELECT column_name(s) FROM table_name1

CROSS JOIN table_name2
```

- Exemplu:

```
SELECT  O.denumire AS obiect , C.nume AS culoare FROM Obiecte O
CROSS JOIN Culori C
```

Cross Join

- Rezultat:

obiect	culoare
tricou	verde
pahar	verde
caiet	verde
tricou	mov
pahar	mov
caiet	mov

Tabelul Obiecte

ido	denumire
1	tricou
2	pahar
3	caiet

Tabelul Culori

idc	nume
1	verde
2	mov

Reuniune, intersecție și diferență

- UNION (reuniune) se folosește pentru a îmbina rezultatele a două sau mai multe interogări într-un singur result-set
- Sintaxa:

```
SELECT <col1>,<col2>,<col3> FROM table1

UNION [ALL]

SELECT <col4>,<col5>,<col6> FROM table2
```

- Fiecare interogare trebuie să conțină același număr de coloane, iar tipurile coloanelor trebuie să fie compatibile

Reuniune, intersecție și diferență

- UNION ALL va include înregistrări duplicate
- Exemplu (cu duplicate):

```
SELECT nume FROM Clienți
UNION ALL
SELECT nume FROM Angajați
```

- Exemplu (fără duplicate):

```
SELECT nume FROM Clienți
UNION
SELECT nume FROM Angajați
```

Reuniune, intersecție și diferență

- INTERSECT (intersecție) este folosit pentru a returna într-un singur result-set acele înregistrări care apar atât în result-set-ul interogării din partea dreaptă cât și în cel al interogării din partea stângă
- Sintaxa:

```
SELECT <col1>,<col2>,<col3>
FROM table1
INTERSECT
SELECT <col4>,<col5>,<col6>
FROM table2
```

Reuniune, intersecție și diferență

- Exemplu:

```
SELECT nume, prenume FROM Clienți
INTERSECT
SELECT nume, prenume FROM Angajați
INTERSECT
SELECT nume, prenume FROM Furnizori
```

Reuniune, intersecție și diferență

- EXCEPT (diferență) este folosit pentru a returna acele înregistrări care apar în result-set-ul interogării din partea stângă dar nu apar în result-set-ul interogării din partea dreaptă
- Sintaxa:

```
SELECT <col1>,<col2>,<col3>
FROM table1
EXCEPT
SELECT <col4>,<col5>,<col6>
FROM table2
```

Reuniune, intersecție și diferență

- Exemplu:

```
SELECT nume, prenume FROM Clienți
EXCEPT
SELECT nume, prenume FROM Angajați
```

- Exemplu:

```
SELECT id_client FROM Clienți
EXCEPT
SELECT id_client FROM Comenzi
```

View

- Un view este un tabel virtual bazat pe result-set-ul unei interogări
- Conține înregistrări și coloane ca un tabel real
- Un view nu stochează date, stochează definiția unei interogări
- Cu ajutorul unui view putem prezenta date din mai multe tabele ca și cum ar veni din același tabel
- De fiecare dată când un view este interogat, motorul bazei de date va recrea datele folosind instrucțiunea SELECT, deci un view va prezenta întotdeauna date actualizate
- Numele coloanelor dintr-un view trebuie să fie unice (în cazul în care avem două coloane cu același nume provenind din tabele diferite, putem folosi un alias pentru una dintre ele)

View

- Sintaxa pentru crearea unui view:

  CREATE VIEW view_name AS

  SELECT column_name(s) FROM table_name

- Sintaxa pentru modificarea unui view:

  ALTER VIEW view_name AS

  SELECT column_name(s) FROM table_name

- Sintaxa pentru ștergerea unui view:

  DROP VIEW view_name

View

- Crearea unui view care conține date din două tabele, Customer și Person:

```
CREATE VIEW vw_Customer AS
SELECT c.CustomerID, c.AccountNumber,
p.FirstName, p.MiddleName, p.LastName
FROM Customer AS c
 INNER JOIN Person AS p
ON c.PersonID = p.BusinessEntityID
```

View

- Modificarea unui view:

```
ALTER VIEW vw_Customer AS
SELECT c.CustomerID, c.AccountNumber, p.Title,
p.FirstName, p.MiddleName, p.LastName
FROM Customer AS c
INNER JOIN Person AS p
ON c.PersonID = p.BusinessEntityID
```

View

- Interogarea unui view

```
SELECT CustomerID, AccountNumber, Title,
 FirstName, MiddleName, LastName
FROM vw_Customer
```

- Ștergerea unui view

```
DROP VIEW vw_Customer
```

View

- Nu se poate folosi clauza ORDER BY în definiția unui view (decât dacă se specifică în definiția view-ului clauza TOP, OFFSET sau FOR XML)
- Dacă dorim să ordonăm înregistrările din result-set, putem folosi clauza ORDER BY atunci când interogăm view-ul
- Pentru a afișa definiția unui view, putem folosi funcția OBJECT_DEFINITION sau procedura stocată sp_helptext:

```
SELECT OBJECT_DEFINITION(OBJECT_ID('schema_name.view_name')

EXEC sp_helptext 'schema_name.view_name'
```

View

- Se pot insera sau actualiza date într-un view doar dacă inserarea sau actualizarea afectează un singur base table (în cazul în care view-ul conține date din mai multe tabele)
- Nu se pot șterge date dintr-un view decât dacă view-ul conține date dintr-un singur base table
- Operațiunile de inserare într-un view sunt posibile doar dacă view-ul expune toate coloanele care nu permit valori NULL

Funcții de agregare

- Funcțiile de agregare realizează un calcul pe o mulțime de valori și returnează o singură valoare
- Exemple: COUNT, AVG, SUM, MIN, MAX
- În afară de COUNT, toate funcțiile de agregare ignoră valorile NULL
- Funcțiile de agregare se folosesc de obicei împreună cu clauzele GROUP BY și HAVING
- Exemplu:

  SELECT COUNT(*) FROM Studenți – calculează numărul total de înregistrări din tabelul Studenți

Funcții de agregare

- Exemplu:

  SELECT MAX(nr_matricol) FROM Studenți – returnează cel mai mare număr matricol din tabelul Studenți

- Exemplu:

  SELECT MIN(nr_matricol) FROM Studenți – returnează cel mai mic număr matricol din tabelul Studenți

- Exemplu:

  SELECT SUM(preț) FROM Produse – calculează prețul total al produselor din tabelul Produse

Clauza GROUP BY

- Se folosește pentru a grupa înregistrările dintr-un result-set după una sau mai multe coloane
- Cel mai adesea funcțiile de agregare folosesc clauza GROUP BY
- Fiecare grup este reprezentat în rezultatul final al interogării de către o singură înregistrare
- Dacă o interogare conține GROUP BY, toate fazele interogării care se execută după GROUP BY (inclusiv SELECT, HAVING, ORDER BY) vor opera pe grupuri

Clauza GROUP BY

- Sintaxa:

```
SELECT column_name1, aggregate_function(column_name2)
FROM table_name
WHERE column_name3 operator value
GROUP BY column_name1
```

Clauza GROUP BY

- Exemplu:
- Vrem să aflăm care este valoarea totală a comenzilor pentru fiecare client

```
SELECT CustomerID,
SUM(Freight) AS TotalFreight
FROM Orders
GROUP BY CustomerID
```

- SUM(Freight) face suma valorilor din coloana Freight pentru fiecare CustomerID

Clauza GROUP BY

- Dorim să aflăm numărul total de comenzi efectuate pentru fiecare client

```
SELECT CustomerID,
COUNT(OrderID) AS NumberOfOrders
FROM Orders
GROUP BY CustomerID
```

- COUNT(OrderID) va genera o singură valoare pentru fiecare CustomerID

Clauza GROUP BY

- Putem să grupăm și după mai multe coloane

  SELECT CustomerID, OrderID

  FROM Orders

  GROUP BY CustomerID, OrderID

- Coloanele care nu apar în clauza GROUP BY nu pot apărea în SELECT decât într-o funcție de agregare cum ar fi COUNT, SUM, AVG, MIN, MAX

Clauza HAVING

- Se folosește pentru a filtra grupurile rezultate după procesarea clauzei GROUP BY
- Doar grupurile pentru care expresia specificată în clauza HAVING returnează TRUE vor fi returnate
- Clauza HAVING este procesată după ce sunt grupate înregistrările și poate fi folosită cu funcții de agregare

Clauza HAVING

- Sintaxa:

```
SELECT column_name1,
aggregate_function(column_name2)
FROM table_name
WHERE column_name3 operator value
GROUP BY column_name1
HAVING aggregate_function(column_name2) operator value
```

Clauza HAVING

- Dorim să găsim clienții care au comenzi totale mai mici decât 300

```
SELECT CustomerID,
SUM(Freight) AS TotalFreight
FROM Orders
GROUP BY CustomerID
HAVING SUM(Freight) < 300
```

Clauza HAVING

- Vrem să aflăm dacă totalul comenzilor unor anumiți clienți este mai mic decât 300

```
SELECT CustomerID,
SUM(Freight) AS TotalFreight
FROM Orders
WHERE CustomerID='ANTON' OR CustomerID='BOLID'
GROUP BY CustomerID
HAVING SUM(Freight) < 300
```

Clauza ORDER BY

- Este folosită pentru a sorta înregistrările după o anumită coloană sau coloane
- Sortarea se face crescător în mod implicit
- Clauza ORDER BY se procesează înainte de clauza TOP
- Deoarece clauza ORDER BY este procesată după SELECT, putem folosi în interiorul ei alias-uri pentru coloane
- Sintaxa:

  SELECT column_name(s)

  FROM table_name

  ORDER BY column_name(s) ASC | DESC
- Pentru a ordona descrescător, se scrie DESC după numele coloanei

Clauza ORDER BY

- Exemplu:
- Dorim să ordonăm clienții în ordine crescătoare în funcție de valoarea totală a comenzilor

```
SELECT CustomerID,
SUM(Freight) AS TotalOrderValue
FROM Orders
GROUP BY CustomerID
ORDER BY TotalOrderValue
```

Funcții de rang

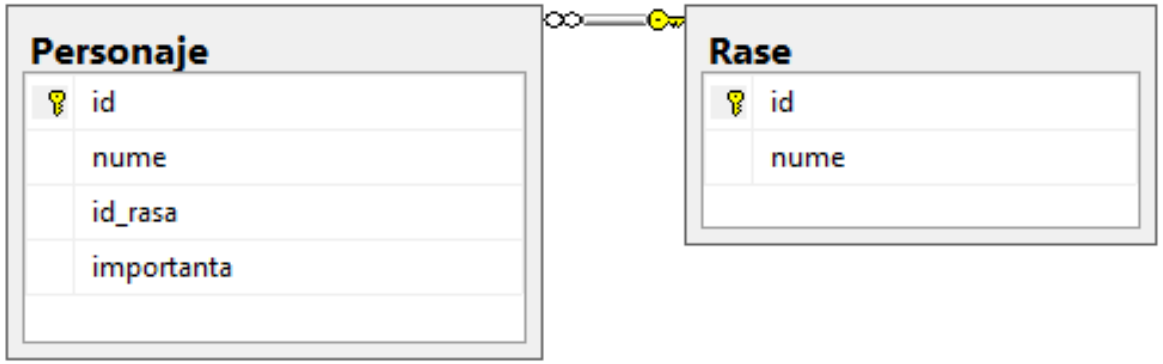
- Funcțiile de rang se folosesc pentru a returna valoarea rangului pentru fiecare înregistrare din result-set-ul unei interogări
- Avem următoarele funcții de rang:
- ROW_NUMBER()
- RANK()
- DENSE_RANK()
- NTILE()

Funcții de rang

- Funcțiile de rang se folosesc în instrucțiuni SELECT și necesită clauzele OVER și ORDER BY (opțional și clauza PARTITION BY)
- O funcție de rang atribuie fiecărei înregistrări din result-set o anumită valoare
- Înregistrările din result-set trebuie cel puțin ordonate și eventual împărțite în sectoare în funcție de valorile anumitor coloane pentru a putea primi valori de rang generate de funcțiile de rang

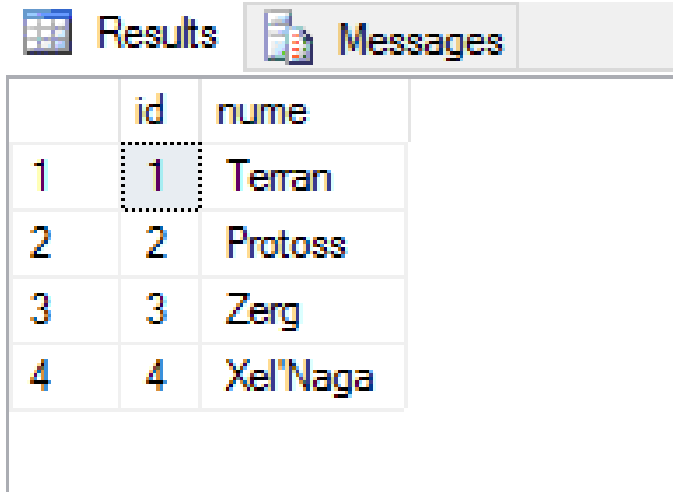
Funcții de rang - Exemplu

- Avem în baza de date două tabele, tabelul Rase și tabelul Personaje, care se află într-o relație one to many:



Funcții de rang - Exemplu

- În tabelul Rase avem următoarele înregistrări (care au fost obținute executând interogarea SELECT * FROM Rase) :



	id	nume
1	1	Terran
2	2	Protoss
3	3	Zerg
4	4	Xel'Naga

Funcții de rang - Exemplu

- În tabelul Personaje, avem următoarele înregistrări:

Results | Messages

	id	nume	id_rasa	importanta
1	1	James Raynor	1	mare
2	2	Sarah Kerrigan	3	mare
3	3	Zeratul	2	mare
4	4	Artanis	2	mare
5	5	Tassadar	2	mare
6	6	Overmind	3	mare
7	7	Arcturus Mengsk	1	mare
8	8	Amon	4	mare
9	9	Alarak	2	mare
10	10	Valerian Mengsk	1	mare
11	11	Matt Horner	1	medie
12	12	Zagara	3	medie
13	13	Abathur	3	mica
14	14	Selendis	2	medie

Funcții de rang - Exemplu

- Dorim să obținem un result-set care să conțină următoarele coloane: numele personajului, importanța personajului și numele rasei
- Vom crea un view cu numele vw_PersonajeRase care va conține doar coloanele nume și importanță din tabelul Personaje și coloana nume din tabelul Rase (coloana nume din tabelul Rase va primi alias-ul "rasa" deoarece nu pot exista două coloane cu același nume în definiția unui view):

```
CREATE VIEW vw_PersonajeRase AS

SELECT Personaje.nume, Personaje.importanta, Rase.nume AS rasa FROM
Personaje INNER JOIN Rase ON Rase.id = Personaje.id_rasa
```

Funcții de rang - Exemplu

- În urma interogării view-ului, vom obține următorul result-set:

Results | Messages

	nume	importanta	rasa
1	James Raynor	mare	Terran
2	Sarah Kerrigan	mare	Zerg
3	Zeratul	mare	Protoss
4	Artanis	mare	Protoss
5	Tassadar	mare	Protoss
6	Overmind	mare	Zerg
7	Arcturus Mengsk	mare	Terran
8	Amon	mare	Xel'Naga
9	Alarak	mare	Protoss
10	Valerian Mengsk	mare	Terran
11	Matt Horner	medie	Terran
12	Zagara	medie	Zerg
13	Abathur	mica	Zerg
14	Selendis	medie	Protoss

Funcții de rang - Exemplu

- Funcția de rang ROW_NUMBER() creează un șir crescător de valori unice întregi pe care le atribuie tuturor înregistrărilor din result-set în funcție de ordinea definită în interiorul clauzei OVER cu ajutorul clauzei ORDER BY
- Exemplu:

```
SELECT nume, importanta, rasa, ROW_NUMBER() OVER (ORDER BY nume)
AS nr_inregistrare FROM vw_PersonajeRase
```

- Interogarea de mai sus folosește funcția de rang ROW_NUMBER() pentru a atribui fiecărei înregistrări din result-set o valoare întreagă unică în funcție de valoarea coloanei nume folosită în clauza ORDER BY din clauza OVER

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	nr_inregistrare
1	Abathur	mica	Zerg	1
2	Alarak	mare	Protoss	2
3	Amon	mare	Xel'Naga	3
4	Arcturus Mengsk	mare	Terran	4
5	Artanis	mare	Protoss	5
6	James Raynor	mare	Terran	6
7	Matt Horner	medie	Terran	7
8	Overmind	mare	Zerg	8
9	Sarah Kerrigan	mare	Zerg	9
10	Selendis	medie	Protoss	10
11	Tassadar	mare	Protoss	11
12	Valerian Mengsk	mare	Terran	12
13	Zagara	medie	Zerg	13
14	Zeratul	mare	Protoss	14

Funcții de rang - Exemplu

- Cu ajutorul clauzei PARTITION BY putem împărți result-set-ul unei interogări în mai multe sectoare, în funcție de valorile unor anumite coloane
- De exemplu, dorim să împărțim result-set-ul interogării în mai multe sectoare în funcție de rasă, iar apoi să ordonăm fiecare sector rezultat după numele personajului și să oferim câte o valoare unică de rang pe sector fiecărei înregistrări:

```
SELECT nume, importanta, rasa, ROW_NUMBER() OVER( PARTITION BY
rasa ORDER BY nume) AS numar_pe_sector FROM vw_PersonajeRase
```

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	numar_pe_sector
1	Alarak	mare	Protoss	1
2	Artanis	mare	Protoss	2
3	Selendis	medie	Protoss	3
4	Tassadar	mare	Protoss	4
5	Zeratul	mare	Protoss	5
6	Arcturus Mengsk	mare	Terran	1
7	James Raynor	mare	Terran	2
8	Matt Horner	medie	Terran	3
9	Valerian Mengsk	mare	Terran	4
10	Amon	mare	Xel'Naga	1
11	Abathur	mica	Zerg	1
12	Overmind	mare	Zerg	2
13	Sarah Kerrigan	mare	Zerg	3
14	Zagara	medie	Zerg	4

Funcții de rang

- Funcția de rang RANK() creează un șir crescător de valori și oferă câte o valoare de rang fiecărei înregistrări în funcție de valoarea coloanei sau de valorile coloanelor folosite pentru ordonare în clauza ORDER BY
- În cazul în care există mai multe înregistrări cu aceeași valoare pentru coloana folosită în clauza ORDER BY, ele vor primi aceeași valoare de rang
- Un contor ține evidența numărului de apariții pentru fiecare valoare din coloana după care se face ordonarea, iar la schimbarea valorii se va sesiza un salt în cazul valorii de rang (dacă sunt două înregistrări cu prima valoare a coloanei după care se face ordonarea, amândouă vor primi valoarea de rang 1, iar înregistrările care conțin valoarea următoare din coloana folosită pentru ordonare vor primi valoarea 3)

Funcții de rang - Exemplu

- De exemplu, dorim să ordonăm înregistrările obținute prin interogarea view-ului vw_PersonajeRase după importanță și să dăm o valoare de rang folosind funcția RANK():

```
SELECT nume, importanta, rasa, RANK() OVER(ORDER BY importanta) AS rang FROM vw_PersonajeRase
```

- Există trei valori pentru coloana importanță pe care o folosim pentru ordonare, așa că vom avea tot trei valori și pentru rang, dar valorile nu vor fi neapărat consecutive (dacă avem cel puțin două înregistrări cu aceeași valoare pentru coloana importanță, valorile de rang nu vor fi consecutive)

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	rang
1	James Raynor	mare	Terran	1
2	Sarah Kerrigan	mare	Zerg	1
3	Zeratul	mare	Protoss	1
4	Artanis	mare	Protoss	1
5	Tassadar	mare	Protoss	1
6	Overmind	mare	Zerg	1
7	Arcturus Mengsk	mare	Terran	1
8	Amon	mare	Xel'Naga	1
9	Alarak	mare	Protoss	1
10	Valerian Mengsk	mare	Terran	1
11	Matt Horner	medie	Terran	11
12	Zagara	medie	Zerg	11
13	Selendis	medie	Protoss	11
14	Abathur	mica	Zerg	14

Funcții de rang - Exemplu

- Dorim să împărțim result-set-ul interogării în mai multe sectoare în funcție de rasă, iar apoi să ordonăm fiecare sector rezultat după importanța personajului și să oferim câte o valoare de rang pe sector fiecărei înregistrări în funcție de importanță:

```
SELECT nume, importanta, rasa, RANK() OVER(PARTITION BY rasa ORDER
BY importanta) AS rang_rasa FROM vw_PersonajeRase
```

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	rang_rasa
1	Zeratul	mare	Protoss	1
2	Artanis	mare	Protoss	1
3	Tassadar	mare	Protoss	1
4	Alarak	mare	Protoss	1
5	Selendis	medie	Protoss	5
6	Valerian Mengsk	mare	Terran	1
7	James Raynor	mare	Terran	1
8	Arcturus Mengsk	mare	Terran	1
9	Matt Horner	medie	Terran	4
10	Amon	mare	Xel'Naga	1
11	Sarah Kerrigan	mare	Zerg	1
12	Overmind	mare	Zerg	1
13	Zagara	medie	Zerg	3
14	Abathur	mica	Zerg	4

Funcții de rang

- Funcția de rang DENSE_RANK() creează un șir crescător de valori și oferă câte o valoare de rang fiecărei înregistrări în funcție de valoarea coloanei sau de valorile coloanelor folosite pentru ordonare în clauza ORDER BY
- În cazul în care există mai multe înregistrări cu aceeași valoare pentru coloana folosită în clauza ORDER BY, ele vor primi aceeași valoare de rang
- Spre deosebire de funcția de rang RANK(), funcția DENSE_RANK() va oferi valori consecutive de rang chiar și în cazul în care există mai multe înregistrări cu aceeași valoare pentru coloana după care se face ordonarea (dacă sunt două înregistrări cu prima valoare a coloanei după care se face ordonarea, amândouă vor primi valoarea de rang 1, iar înregistrările care conțin valoarea următoare din coloana folosită pentru ordonare vor primi valoarea 2)

Funcții de rang - Exemplu

- De exemplu, dorim să ordonăm înregistrările obținute prin interogarea view-ului vw_PersonajeRase după importanță și să dăm o valoare de rang folosind funcția DENSE_RANK():

```
SELECT nume, importanta, rasa, DENSE_RANK() OVER(ORDER BY
importanta) AS rang_dens FROM vw_PersonajeRase
```

- Există trei valori pentru coloana importanță pe care o folosim pentru ordonare, așa că vom avea tot trei valori și pentru rang, dar valorile vor fi de această dată consecutive (chiar dacă avem mai multe înregistrări cu aceeași valoare pentru coloana importanță, valorile de rang vor fi consecutive)

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	rang_dens
1	James Raynor	mare	Terran	1
2	Sarah Kerrigan	mare	Zerg	1
3	Zeratul	mare	Protoss	1
4	Artanis	mare	Protoss	1
5	Tassadar	mare	Protoss	1
6	Overmind	mare	Zerg	1
7	Arcturus Mengsk	mare	Terran	1
8	Amon	mare	Xel'Naga	1
9	Alarak	mare	Protoss	1
10	Valerian Mengsk	mare	Terran	1
11	Matt Horner	medie	Terran	2
12	Zagara	medie	Zerg	2
13	Selendis	medie	Protoss	2
14	Abathur	mica	Zerg	3

Funcții de rang - Exemplu

- Dorim să împărțim result-set-ul interogării în mai multe sectoare în funcție de rasă, iar apoi să ordonăm fiecare sector rezultat după importanța personajului și să oferim câte o valoare de rang dens pe sector fiecărei înregistrări în funcție de importanță:

```
SELECT nume, importanta, rasa, DENSE_RANK()
OVER(PARTITION BY rasa ORDER BY importanta) AS rang_dens_rasa
FROM   vw_PersonajeRase
```

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	rang_dens_rasa
1	Zeratul	mare	Protoss	1
2	Artanis	mare	Protoss	1
3	Tassadar	mare	Protoss	1
4	Alarak	mare	Protoss	1
5	Selendis	medie	Protoss	2
6	Valerian Mengsk	mare	Terran	1
7	James Raynor	mare	Terran	1
8	Arcturus Mengsk	mare	Terran	1
9	Matt Horner	medie	Terran	2
10	Amon	mare	Xel'Naga	1
11	Sarah Kerrigan	mare	Zerg	1
12	Overmind	mare	Zerg	1
13	Zagara	medie	Zerg	2
14	Abathur	mica	Zerg	3

Funcții de rang

- Funcția de rang NTILE() se folosește pentru a împărți înregistrările din result-set-ul unei interogări într-un anumit număr de grupuri egale sau aproximativ egale
- Numărul pe care îl furnizăm ca parametru funcției NTILE() stabilește numărul de grupuri în care vrem să divizăm result-set-ul
- De exemplu, NTILE(20) va produce 20 de grupuri egale sau aproximativ egale ca număr de înregistrări conținute
- Dacă numărul de grupuri furnizat ca parametru este mai mare decât numărul de înregistrări din result-set-ul interogării, se vor crea grupuri de câte o înregistrare (de exemplu, dacă dorim să împărțim un result-set care conține 14 înregistrări în 20 de grupuri, vor rezulta 14 grupuri cu câte o singură înregistrare)

Funcții de rang - Exemplu

- De exemplu, dorim să ordonăm înregistrările obținute prin interogarea view-ului vw_PersonajeRase după importanță și să împărțim result-set-ul interogării în 3 grupuri egale sau aproximativ egale:

```
SELECT nume, importanta, rasa, NTILE(3) OVER(ORDER BY importanta) AS
nr_grup FROM vw_PersonajeRase
```

- Deoarece result-set-ul interogării conține 14 înregistrări, iar acest număr nu se împarte exact la 3, vor rezulta două grupuri cu câte 5 înregistrări și un grup cu 4 înregistrări

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	nr_grup
1	James Raynor	mare	Terran	1
2	Sarah Kerrigan	mare	Zerg	1
3	Zeratul	mare	Protoss	1
4	Artanis	mare	Protoss	1
5	Tassadar	mare	Protoss	1
6	Overmind	mare	Zerg	2
7	Arcturus Mengsk	mare	Terran	2
8	Amon	mare	Xel'Naga	2
9	Alarak	mare	Protoss	2
10	Valerian Mengsk	mare	Terran	2
11	Matt Horner	medie	Terran	3
12	Zagara	medie	Zerg	3
13	Selendis	medie	Protoss	3
14	Abathur	mica	Zerg	3

Funcții de rang - Exemplu

- Rezultatul în cazul în care încercăm să împărțim result-set-ul în 20 de grupuri:

	nume	importanta	rasa	nr_grup
1	James Raynor	mare	Terran	1
2	Sarah Kerrigan	mare	Zerg	2
3	Zeratul	mare	Protoss	3
4	Artanis	mare	Protoss	4
5	Tassadar	mare	Protoss	5
6	Overmind	mare	Zerg	6
7	Arcturus Mengsk	mare	Terran	7
8	Amon	mare	Xel'Naga	8
9	Alarak	mare	Protoss	9
10	Valerian Mengsk	mare	Terran	10
11	Matt Horner	medie	Terran	11
12	Zagara	medie	Zerg	12
13	Selendis	medie	Protoss	13
14	Abathur	mica	Zerg	14

Funcții de rang - Exemplu

- Dorim să împărțim result-set-ul interogării în mai multe sectoare în funcție de rasă, iar apoi să ordonăm fiecare sector rezultat după importanța personajului și să împărțim fiecare sector în 3 grupuri aproximativ egale în funcție de importanță:

```
SELECT nume, importanta, rasa, NTILE(3)
OVER(PARTITION BY rasa ORDER BY importanta) AS nr_grupuri_rasa
FROM vw_PersonajeRase
```

Funcții de rang - Exemplu

- Rezultatul:

	nume	importanta	rasa	nr_grupuri_rasa
1	Zeratul	mare	Protoss	1
2	Artanis	mare	Protoss	1
3	Tassadar	mare	Protoss	2
4	Alarak	mare	Protoss	2
5	Selendis	medie	Protoss	3
6	Valerian Mengsk	mare	Terran	1
7	James Raynor	mare	Terran	1
8	Arcturus Mengsk	mare	Terran	2
9	Matt Horner	medie	Terran	3
10	Amon	mare	Xel'Naga	1
11	Sarah Kerrigan	mare	Zerg	1
12	Overmind	mare	Zerg	1
13	Zagara	medie	Zerg	2
14	Abathur	mica	Zerg	3

Bibliografie

- https://docs.microsoft.com/en-us/sql/t-sql/language-reference
- https://docs.oracle.com/javadb/10.6.2.1/ref/rrefsqljnaturaljoin.html
- https://www.w3resource.com/mysql/advance-query-in-mysql/mysql-natural-join.php
- Leon Țâmbulea, curs de Baze de Date, UBB Cluj-Napoca
- Andor Camelia, seminar de Baze de Date, Matematică-Informatică, UBB Cluj-Napoca, 2016
- Basit A. Masood-Al-Farooq, SQL Server 2014 Development Essentials, Packt Publishing, 2014