

Project final

TRIF CRISTIAN

DATA EXAMEN:
27.06.2025

Cerințe de business

- **Definitie:** Cerințele de business sunt nevoile, așteptările sau obiectivele organizației care justifică dezvoltarea unui produs sau implementarea unei soluții. Ele definesc ce vrea să obțină afacerea prin proiect și stau la baza specificațiilor funcționale și tehnice.
- **Cerințele de business** sunt evaluate în **etapa de analiză a cerințelor**. În această fază, echipa de testare analizează documentația de specificații pentru a înțelege ce anume trebuie testat și pentru a identifica cerințele funcționale și non-funcționale ale aplicației. Scopul este să se asigure că cerințele sunt clare, complete, corecte și testabile.

Exemplu practic (click): Am analizat nevoile unui atelier auto și am creat tabelele Clienți, Mașini, Reparații și Piese, pentru a gestiona datele esențiale. Structura bazei reflectă cerințele reale ale afacerii.

Entry criteria vs Exit criteria

- **Definiție Criterii de intrare:** Condiții care trebuie îndeplinite pentru a începe testarea (ex: tabele create, date disponibile)
- **Definiție Criterii de ieșire:** Condiții care indică finalizarea testării (ex: date inserate corect, interogări executate fără erori)

Exemplu practic:

Entry criteria: Pentru a insera date în Masini, tabela Clienti trebuie să existe și să conțină clienți, deoarece id_client este cheie străină (FOREIGN KEY).

The screenshot shows a MySQL Workbench interface. A query window displays the following SQL code:

```
18 •  SELECT * FROM Clienti;
```

Below the query is a "Result Grid" table with the following data:

	id_client	nume	telefon	email	cnp
▶	1	Cristi	0711111111	cristi@email.com	NULL
▶	2	Gina	0722222222	gina.nou@email.com	NULL
●	3	Vasile Pop	0724000000	vasile@email.com	1900101223344
	NULL	NULL	NULL	NULL	NULL

Exit criteria: Comenzile INSERT s-au executat corect, iar datele apar în SELECT fără erori.

The screenshot shows a MySQL Workbench interface. A query window displays the following SQL code:

```
20 •  INSERT INTO Masini (id_client, marca, model, an_fabricatie)  
21      VALUES (1, 'Dacia', 'Logan', 2015);
```

Below the query is an "Output" window titled "Action Output" with the following log entry:

#	Time	Action
1	21:13:05	INSERT INTO Masini (id_client, marca, model, an_fabricatie) VALUES (1, 'Dacia', 'Logan', 2015)

Precondition vs Test condition

- **Precondition** reprezintă o stare care trebuie să fie adevărată înainte de executarea unui test (ex: datele există, tabelele sunt create)
- **Test condition** reprezintă o funcționalitate sau comportament care trebuie verificat printr-un test.

Exemplu practic:

- Precondiție: Tabelele Clienti, Masini și Reparatii trebuie să existe și să conțină date corelate.
- Test condition: Interogarea JOIN trebuie să returneze numele clientului, marca mașinii și detalii despre reparatie:

```
26 •   SELECT c.nume, m.marca, r.descriere
27     FROM Clienti c
28     JOIN Masini m ON c.id_client = m.id_client
29     JOIN Reparatii r ON m.id_masina = r.id_masina;
```

Result Grid | Filter Rows: _____ | Export: _____ | Wrap C

nume	marca	descriere
Cristi	Dacia	Schimb bujii și filtru ulei
Gina	Volkswagen	Înlocuire discuri frână

Ce se întâmplă când **precondițiile NU** sunt îndeplinite:

```
31 •   INSERT INTO Masini (id_client, marca, model, an_fabricatie)
32     VALUES (99, 'Renault', 'Clio', 2019);
```

Output ::::::::::::::::::::

#	Time	Action	Message
1	21:40:03	INSERT INTO Masini (id_client, marca, model, an_fabricatie) VALUES (99, 'Renault', 'Clio', 2019)	Error Code: 1452.

Explicație: Comanda eșuează deoarece id_client = 99 nu există în tabela Clienti și Precondiția nu este îndeplinită, deci nu putem continua inserarea.

Nivelurile de testare sunt următoarele:

- Unit Testing – testarea unităților individuale de cod (de exemplu, o funcție sau o metodă). De obicei este făcută de dezvoltatori.
- Integration Testing – testarea interacțiunii dintre mai multe unități sau module, pentru a verifica dacă funcționează bine împreună.
- Acceptance Testing – testarea finală realizată pentru a vedea dacă aplicația este acceptabilă pentru utilizator sau client (de exemplu, User Acceptance Testing - UAT).
- System Testing – testarea întregului sistem ca un tot unitar, pentru a verifica dacă îndeplinește cerințele specificate.

Unit testing: verifică funcționarea corectă a unei comenzi SQL individuale. Exemplu concret: Adăugarea coloanei cnp în tabela Clienti și testarea printr-un INSERT cu date conforme.

```
42 • ALTER TABLE Clienti ADD cnp CHAR(13);  
  
Output  
Action Output  
# Time Action  
1 21:58:15 ALTER TABLE Clienti ADD cnp CHAR(13)  
  
55 • INSERT INTO Clienti (nume, telefon, email, cnp)  
56 VALUES ('Marcel', '0724000000', 'marcel@email.com', '1900101223344');  
57 • SELECT * FROM Clienti WHERE nume = 'Marcel';  
  
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C  
id_client nume telefon email cnp  
4 Marcel 0724000000 marcel@email.com 1900101223344
```

Integration Testing: Verifică interacțiunea dintre tabele conectate prin chei străine. Exemplu: Inserarea în Masini esuează dacă id_client nu există în Clienti, deci relația este testată corect.

```
45 • INSERT INTO Masini (id_client, marca, model, an_fabricatie)  
46 VALUES (99, 'Renault', 'Clio', 2019);  
  
Output  
Action Output  
# Time Action  
1 21:59:28 INSERT INTO Masini (id_client, marca, model, an_fabricatie) VALUES (99, 'Renault', 'Clio', 2019) Error Code: 1452  
  
55 • INSERT INTO Clienti (nume, telefon, email, cnp)
```

Acceptance Testing: testează întregul sistem de baze de date ca un tot unitar. Exemplu: Interrogarea JOIN între Clienti, Masini și Reparatii trebuie să returneze toate datele corelate.

```
48 • SELECT c.nume, m.marca, r.descriere  
49 FROM Clienti c  
50 JOIN Masini m ON c.id_client = m.id_client  
51 JOIN Reparatii r ON m.id_masina = r.id_masina;  
  
Result Grid | Filter Rows: | Export: | Wrap C  
nume marca descriere  
Cristi Dacia Schimb bujii și filtru ulei  
Gina Volkswagen Înlăturare discuri frână
```

System Testing: simulează folosirea bazei de date de către un angajat al atelierului. Exemplu: Angajatul extrage lista pieselor utilizate într-o reparație pentru a genera un deviz.

```
53 • SELECT * FROM Reparatii WHERE id_masina = 1;  
  
Result Grid | Filter Rows: | Edit: | Export: | Wrap C  
id_reparatie id_masina data_reparatiei descriere cost  
1 1 2025-05-21 Schimb bujii și filtru ulei 81.00  
* NULL NULL NULL NULL NULL
```

Proiect SQL:

Tipuri de date:

Un **tip de date** definește ce fel de informație poate stoca o coloană dintr-un tabel (ex: numere, texte, date calendaristice).

Principalele tipuri de date în MySQL:

• Numerice:

- INT, SMALLINT, BIGINT – numere întregi
- DECIMAL, FLOAT, DOUBLE – numere zecimale

• Text:

- CHAR(n) – text fix
- VARCHAR(n) – text variabil
- TEXT – text lung

• Date și timp:

- DATE – doar data (ex: 2025-05-21)
- DATETIME – data + ora
- TIME – doar ora

Diferenta dintre CHAR si VARCHAR:

CHAR(n): stochează exact n caractere, chiar dacă textul este mai scurt (completează cu spații).

Exemplu: CHAR(5) cu valoarea 'ab' → stochează 'ab '. Se folosește când toate valorile au aceeași lungime (ex: coduri poștale, CNP).

VARCHAR(n): stochează până la n caractere, doar cât e nevoie.

Exemplu: VARCHAR(5) cu valoarea 'ab' → stochează 'ab'. Se folosește când lungimea textului variază (ex: nume, adrese).

Cheia primara (primary key) vs cheia secundara (foreign key):

Cheie primară (PRIMARY KEY):

Identifică unic fiecare rând dintr-un tabel.

Nu poate fi NULL

Cheie secundară (FOREIGN KEY):

Creează o legătură între două tabele.

Este o coloană care face referire la cheia primară din alt tabel. Asigură integritatea datelor.

LEFT JOIN vs RIGHT JOIN:

LEFT JOIN:

Afișează toate rândurile din tabelul din stânga și valorile corespunzătoare din cel din dreapta (sau NULL dacă nu există potrivire).

RIGHT JOIN: Afișează toate rândurile din tabelul din dreapta și valorile corespunzătoare din cel din stânga (sau NULL dacă nu există potrivire).

Relația many-to-many :

O relație **many-to-many** (**n la n**) apare când un rând dintr-un tabel poate avea **mai multe legături** cu rânduri din alt tabel și invers.

WHERE vs HAVING:

WHERE filtrează rândurile înainte de aplicarea funcțiilor aggregate (SUM, COUNT, etc.).

HAVING filtrează după ce s-au aplicat funcțiile aggregate.

Proiectul de față simulează o aplicație de gestiune pentru un atelier auto, folosind o bază de date MySQL. Am creat și interogat tabele care reflectă relațiile dintre clienți, mașini, reparații și piese auto. Proiectul respectă toate cerințele (DDL, DML, DQL, relații între tabele) și este însoțit de capturi de ecran și explicații pentru fiecare pas.

În această etapă am creat tabelele principale ale aplicației AtelierAuto:

Clienti – stochează datele clientilor, cu id_client ca PRIMARY KEY.

Masini – conține detalii despre mașini, legate de clienți prin id_client (FOREIGN KEY, relație 1:N).

Reparatii – înregistrează lucrările efectuate pentru fiecare mașină, având o legătură 1:N cu tabela Masini.

Piese – stochează informații despre piesele auto disponibile, folosind VARCHAR și DECIMAL.

```
68 • CREATE TABLE Clienti (
69     id_client INT PRIMARY KEY AUTO_INCREMENT,
70     nume VARCHAR(100),
71     telefon CHAR(10),
72     email VARCHAR(100)
73 );
74 • CREATE TABLE Masini (
75     id_masina INT PRIMARY KEY AUTO_INCREMENT,
76     id_client INT,
77     marca VARCHAR(50),
78     model VARCHAR(50),
79     an_fabricatie INT,
80     FOREIGN KEY (id_client) REFERENCES Clienti(id_client)
81 );
82 • CREATE TABLE Reparatii (
83     id_reparatie INT PRIMARY KEY AUTO_INCREMENT,
84     id_masina INT,
85     data_reparatiei DATE,
86     descriere TEXT,
87     cost DECIMAL(10,2),
88     FOREIGN KEY (id_masina) REFERENCES Masini(id_masina)
89 );
90 • CREATE TABLE Piese (
91     id_piesa INT PRIMARY KEY AUTO_INCREMENT,
92     nume_piesa VARCHAR(100),
93     pret DECIMAL(10,2)
94 );
95
96
```

Output

#	Time	Action	Message
1	21:33:12	CREATE TABLE Clienti (id_client INT PRIMARY KEY AUTO_INCREMENT, nume VARCHAR(100), telefon CHAR(10), e...)	0 row(s) affected
2	21:33:12	CREATE TABLE Masini (id_masina INT PRIMARY KEY AUTO_INCREMENT, id_client INT, marca VARCHAR(50), mod...)	0 row(s) affected
3	21:33:12	CREATE TABLE Reparatii (id_reparatie INT PRIMARY KEY AUTO_INCREMENT, id_masina INT, data_reparatiei DATE, ...)	0 row(s) affected
4	21:33:12	CREATE TABLE Piese (id_piesa INT PRIMARY KEY AUTO_INCREMENT, nume_piesa VARCHAR(100), pret DECIMAL(1...))	Error Code: 1064.
5	21:33:22	CREATE TABLE Piese (id_piesa INT PRIMARY KEY AUTO_INCREMENT, nume_piesa VARCHAR(100), pret DECIMAL(1...))	0 row(s) affected

Am creat tabela **Mecanici** pentru a stoca informații despre personalul tehnic al atelierului. După ce am inserat câțiva mecanici de test, am aplicat comanda TRUNCATE TABLE Mecanici pentru a șterge toate înregistrările. Am ales această comandă în locul DELETE deoarece este mai rapidă și păstrează structura tabelului intactă. Tabela Mecanici rămâne în continuare activă, urmând să fie populată cu alți angajați reali în etapele următoare.

Am folosit comanda ALTER TABLE ... DROP COLUMN pentru a elmina coloana adresa din tabela Clienti, deoarece informația nu mai era necesară. Această comandă modifică structura tabelei fără a afecta celelalte date existente.

Am adăugat coloana “cnp” în tabela Clienti folosind comanda ALTER, specificând tipul CHAR(13), deoarece CNP-ul are întotdeauna 13 caractere. Apoi am inserat un client folosind un INSERT cu coloane explicate, care include și câmpul “cnp”.

```

154 • CREATE TABLE Mecanici (
155     id_mecanic INT PRIMARY KEY AUTO_INCREMENT,
156     nume VARCHAR(100),
157     specializare VARCHAR(100)
158 );
159
160 • INSERT INTO Mecanici (nume, specializare)
161     VALUES ('Ionel', 'electrician'), ('Marian', 'mecanic motor');
162
163 • TRUNCATE TABLE Mecanici;
164
165
166
167
168
169

```

Output

#	Time	Action	Message
1	22:05:04	CREATE TABLE Mecanici (id_mecanic INT PRIMARY KEY AUTO_INCREMENT, nume VARCHAR(100), specializare VA...)	0 row(s) affected
2	22:05:15	INSERT INTO Mecanici (nume, specializare) VALUES ('Ionel', 'electrician'), ('Marian', 'mecanic motor')	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
3	22:05:29	TRUNCATE TABLE Mecanici	0 row(s) affected

```

167 • ALTER TABLE Clienti DROP COLUMN adresa;
168
169

```

Output

#	Time	Action
1	22:12:24	ALTER TABLE Clienti DROP COLUMN adresa

```

13 • ALTER TABLE Clienti ADD cnp CHAR(13);
14 • INSERT INTO Clienti (nume, telefon, email, cnp)
15     VALUES ('Vasile Pop', '0724000000', 'vasile@email.com', '1900101223344');
16 • SELECT * FROM Clienti WHERE nume = 'Vasile Pop';
--
```

Result Grid

id_client	nume	telefon	email	cnp
3	Vasile Pop	0724000000	vasile@email.com	1900101223344

Edit: **Export/Import:** **Wrap Cell:**

În această etapă am definit două relații de tip many-to-many (N:N), esențiale pentru simularea realistă a unui atelier auto:

- Reparatii_Piese – leagă reparațiile de piesele utilizate, deoarece o reparatie poate necesita mai multe piese, iar o piesă poate fi folosită în mai multe reparații.

```
95 • CREATE TABLE Reparatii_Piese (
96     id_reparatie INT,
97     id_piesa INT,
98     cantitate INT,
99     PRIMARY KEY (id_reparatie, id_piesa),
100    FOREIGN KEY (id_reparatie) REFERENCES Reparatii(id_reparatie),
101    FOREIGN KEY (id_piesa) REFERENCES Piese(id_piesa)
102 );
```

- Mecanici_Reparatii – leagă mecanicii de reparațiile la care participă, încrucât o reparatie poate fi efectuată de mai mulți mecanici, iar un mecanic poate lucra la mai multe reparații.

```
CREATE TABLE Mecanici_Reparatii (
    id_mecanic INT,
    id_reparatie INT,
    PRIMARY KEY (id_mecanic, id_reparatie),
    FOREIGN KEY (id_mecanic) REFERENCES Mecanici(id_mecanic),
    FOREIGN KEY (id_reparatie) REFERENCES Reparatii(id_reparatie)
);
```

În această etapă am populat baza de date cu date reale, folosind atât comenzi INSERT cu coloane **explicite**, cât și cu coloane **implicite**.

- Am folosit INSERT cu coloane explicite în cazul în care am dorit să specific exact ce câmpuri completăm, pentru claritate și flexibilitate.
- Am folosit INSERT cu coloane implicite atunci când am completat toate coloanele în ordinea definită în tabel, fără a le specifica în comandă. (Vezi în imaginea alăturată)

```
104 • INSERT INTO Clienti (nume, telefon, email)
105     VALUES
106         ('Cristi', '0711111111', 'cristi@email.com'),
107         ('Gina', '0722222222', 'gina@email.com');
108
109 -- INSERT implicit (toate coloanele, fără a specifica numele lor)
110 • INSERT INTO Clienti
111     VALUES (NULL, 'Denisa', '0733333333', 'denisa@email.com');
112
113 • INSERT INTO Masini (id_client, marca, model, an_fabricatie)
114     VALUES (1, 'Dacia', 'Logan', 2015),
115         (2, 'Volkswagen', 'Golf', 2018);
116
117 • INSERT INTO Piese (nume_piesa, pret)
118     VALUES ('Bujie', 25.50),
119         ('Filtru ulei', 30.00),
120         ('Discuri frână', 150.00);
121
122 • INSERT INTO Reparatii (id_masina, data_reparatiei, descriere, cost)
123     VALUES (1, '2025-05-21', 'Schimb bujii și filtru ulei', 81.00),
124         (2, '2025-05-22', 'Înllocuire discuri frână', 300.00);
125
126 • INSERT INTO Reparatii_Piese (id_reparatie, id_piesa, cantitate)
127     VALUES (1, 1, 4), -- 4 bujii
128         (1, 2, 1), -- 1 filtru ulei
129         (2, 3, 2); -- 2 discuri frână
130
131
132
133
```

Action	Output
CREATE TABLE Reparatii_Piese (id_reparatie INT, id_piesa INT, cantitate INT, PRIMARY KEY (id_reparatie, id_piesa)...)	Message 0 row(s) affected
INSERT INTO Clienti (nume,telefon,email) VALUES ('Cristi','0711111111','cristi@email.com'), ('Gina','0722222222','gina@email.com');	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
INSERT INTO Masini (id_client,marca,model,an_fabricatie) VALUES (1,'Dacia','Logan',2015), (2,'Volkswagen','Golf',2018);	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
INSERT INTO Piese (nume_piesa,pret) VALUES ('Bujie',25.50), ('Filtru ulei',30.00), ('Discuri frână',150.00);	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
INSERT INTO Reparatii (id_masina,data_reparatiei,descriere,cost) VALUES (1,'2025-05-21','Schimb bujii și filtru ulei',81.00), (2,'2025-05-22','Înllocuire discuri frână',300.00);	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
INSERT INTO Reparatii_Piese (id_reparatie,id_piesa,cantitate) VALUES (1,1,4), -- 4 bujii (1,2,1), -- 1 filtru ulei (2,3,2); -- 2 discuri frână	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

În această etapă am aplicat comanda UPDATE pentru a modifica adresa de email a clientei Gina, și comanda DELETE pentru a șterge clienta Denisa din tabela Clienti. Inițial, comanda UPDATE a generat eroarea 1175, deoarece MySQL avea activat modul de siguranță SQL_SAFE_UPDATES, care nu permite actualizări fără utilizarea unei chei (id_client) în clauza WHERE. Am rezolvat eroarea dezactivând temporar modul de siguranță cu comanda SET SQL_SAFE_UPDATES = 0;, după care UPDATE a funcționat corect. Comanda DELETE a fost folosită pentru a elimina o înregistrare specifică, folosind WHERE pentru a evita ștergerea tuturor datelor din tabel.

```

169 • UPDATE Clienti
170   SET email = 'gina.nou@email.com'
171   WHERE name = 'Gina';
172 • SET SQL_SAFE_UPDATES = 0;
173
174 • UPDATE Clienti
175   SET email = 'gina.nou@email.com'
176   WHERE name = 'Gina';
177
178 • DELETE FROM Clienti
179   WHERE name = 'Denisa';
180
181

```

Action Output

#	Time	Action	Message
1	22:17:33	UPDATE Clienti SET email = 'gina.nou@email.com' WHERE name = 'Gina'	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To do this set SET SQL_SAFE_UPDATES = 0.
2	22:19:06	SET SQL_SAFE_UPDATES = 0	0 row(s) affected
3	22:19:26	UPDATE Clienti SET email = 'gina.nou@email.com' WHERE name = 'Gina'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
4	23:19:54	DELETE FROM Clienti WHERE name = 'Denisa'	0 row(s) affected

Mai jos este o comparație clară între cele trei comenzi DML de bază din SQL – **INSERT**, **UPDATE** și **DELETE** – care permit adăugarea, modificarea sau ștergerea datelor din tabele.

Comandă	Scop	Efect	Necesită WHERE
+ INSERT	Adaugă rând nou	Creează o nouă înregistrare	X Nu
-pencil UPDATE	Modifică date existente	Actualizează valorile unor coloane	✓ Da
X DELETE	Șterge rânduri existente	Elimină complet una sau mai multe	✓ Da

În această etapă am demonstrat utilizarea relațiilor dintre tabele și aplicarea comenziilor **JOIN** pentru extragerea datelor relevante. Am creat o interogare **SELECT** care folosește două **INNER JOIN** pentru a aduce împreună date din tabelele Clienti, Masini și Reparatii. Comanda rulează următorul scenariu: Asociază fiecare client cu mașina sa (Clienti.id_client = Masini.id_client), apoi asociază fiecare mașină cu reparațiile efectuate (Masini.id_masina = Reparatii.id_masina). Rezultatul afișat arată: numele clientului, marca și modelul mașinii, descrierea reparației și costul acesteia. Prin această interogare am demonstrat legăturile logice dintre entități și modul în care JOIN-urile permit accesul combinat la informații din tabele diferite, folosind cheile străine (FOREIGN KEY) definite anterior.

```

181 •   SELECT c.nume AS client,
182       m.marca,
183       m.model,
184       r.descriere,
185       r.cost
186   FROM Clienti c
187   JOIN Masini m ON c.id_client = m.id_client
188   JOIN Reparatii r ON m.id_masina = r.id_masina;
189
190

```

client	marca	model	descriere	cost
Cristi	Dacia	Logan	Schimb bujii și filtru ulei	81.00
Gina	Volkswagen	Golf	Înlături discuri frână	300.00

Am creat tabela **Permise** cu **id_client** ca cheie primară și străină, stabilind o **relație 1:1** cu tabela **Clienti**. Apoi am inserat un permis pentru clientul cu **id_client = 1**.

```

59 •   CREATE TABLE Permise (
60       id_client INT PRIMARY KEY,
61       nr_permit VARCHAR(20),
62       categorie VARCHAR(5),
63       FOREIGN KEY (id_client) REFERENCES Clienti(id_client)
64   );
65 •   INSERT INTO Permise (id_client, nr_permit, categorie)
66   VALUES (1, 'B1234567', 'B');

```

Output

#	Time	Action	Message
1	22:35:15	CREATE TABLE Permise (id_client INT PRIMARY KEY, nr_permit VARCHAR(20), cat...)	0 row(s) affected
2	22:36:19	INSERT INTO Permise (id_client, nr_permit, categorie) VALUES (1, 'B1234567', 'B')	1 row(s) affected

Am verificat corectitudinea **relației 1:1** printr-un JOIN între Clienti și Permise. Rezultatul confirmă că fiecărui client îi corespunde un singur permis.

```

68 •   SELECT c.nume, p.nr_permit, p.categorie
69   FROM Clienti c
70   JOIN Permise p ON c.id_client = p.id_client;

```

nume	nr_permit	categorie
Cristi	B1234567	B

În acest slide am demonstrat interogări DQL esențiale: **SELECT ***, **SELECT** coloane, **WHERE**, **LIKE**, **AND**, **OR**, **ORDER BY**, **LIMIT**, **GROUP BY** și **HAVING**. Aceste comenzi permit extragerea și filtrarea eficientă a datelor, fiind folosite frecvent în aplicații de gestiune reală.

```
200      -- 1. Select all
201  •  SELECT * FROM Clienti;
202
203      -- 2. Select coloane
204  •  SELECT nume, email FROM Clienti;
205
206      -- 3. WHERE simplu
207  •  SELECT * FROM Clienti WHERE nume = 'Gina';
208
209      -- 4. LIKE
210  •  SELECT * FROM Clienti WHERE email LIKE '%gmail.com';
211
212      -- 5. WHERE cu AND
213  •  SELECT * FROM Clienti WHERE email LIKE '%gmail.com' AND telefon LIKE '07%';
214
215      -- 6. WHERE cu OR
216  •  SELECT * FROM Clienti WHERE email LIKE '%yahoo.com' OR telefon LIKE '07%';
217
218      -- 7. ORDER BY + LIMIT
219  •  SELECT * FROM Piese ORDER BY pret DESC LIMIT 2;
220
221      -- 8. GROUP BY + COUNT
222  •  SELECT id_masina, COUNT(*) AS nr_reparatii FROM Reparatii GROUP BY id_masina;
223
224      -- 9. HAVING
225  •  SELECT id_masina, COUNT(*) AS nr_reparatii
226  FROM Reparatii
227  GROUP BY id_masina
228  HAVING nr_reparatii > 1;
```

SELECT * FROM Clienti; - Afisează toți clientii și toate coloanele – util pentru vizualizare completă.

SELECT nume, email FROM Clienti; - Afisează doar numele și emailul clientilor – filtrare pe coloane relevante.

SELECT * FROM Clienti WHERE nume = 'Gina'; - Selectează clientul cu numele Gina – filtrare exactă.

SELECT * FROM Clienti WHERE email LIKE '%gmail.com'; - Găsește toți clientii cu adrese Gmail – filtrare parțială folosind LIKE.

SELECT * FROM Clienti WHERE email LIKE '%gmail.com' AND telefon LIKE '07%'; - Filtrează clientii cu email Gmail și număr de telefon care începe cu 07 – combinare condiții.

SELECT * FROM Clienti WHERE email LIKE '%yahoo.com' OR telefon LIKE '07%'; - Selectează clientii cu email Yahoo sau număr care începe cu 07 – alternativă logică.

SELECT * FROM Piese ORDER BY pret DESC LIMIT 2; - Afisează cele mai scumpe 2 piese – sortare descrescătoare și limitare.

SELECT id_masina, COUNT(*) FROM Reparatii GROUP BY id_masina; - Numără câte reparații are fiecare mașină – analiză de frecvență.

SELECT id_masina, COUNT(*) ... HAVING nr_reparatii > 1; - Afisează doar mașinile cu mai mult de o reparație – filtrare pe grupuri.

În acest slide am demonstrat utilizarea scenariilor complexe de interogare DQL folosind mai multe tipuri de JOIN și o subinterrogare (subquery), aplicate în contextul real al unei aplicații de gestiune pentru un atelier auto.

➤ INNER JOIN

- Afisează clienții, mașinile și descrierea reparațiilor.
- Util pentru istoricul complet al reparațiilor efectuate.

➤ LEFT JOIN

- Returnează toți clienții, inclusiv cei fără mașini înregistrate.
- Ajută la identificarea clienților noi sau inactivi.

➤ RIGHT JOIN

- Returnează toate mașinile, chiar dacă nu au clienți asociați (scenariu de verificare a integrității datelor).

➤ CROSS JOIN

- Combină fiecare client cu fiecare piesă (produs cartezian).
- Folosit teoretic, pentru simulări de ofertare sau testare.

➤ SUBQUERY

- Extrag clienții care dețin mașini marca „Dacia”.
- Util pentru campanii targetate sau rapoarte segmentate.

Result 16 Result 17 Result 18 Result 19 Clienti 20 X			
Output			
Action Output			Message
#	Time	Action	
1	23:49:30	SELECT c.nume, m.marca, r.descriere FROM Clienti c JOIN Masini m ON c.id_client = m.id_client JOIN Reparatii r ON m.id_masina = r.id_masina;	2 row(s) returned
2	23:49:30	SELECT c.nume, m.marca FROM Clienti c LEFT JOIN Masini m ON c.id_client = m.id_client LIMIT 0, 1000	2 row(s) returned
3	23:49:30	SELECT c.nume, m.marca FROM Clienti c RIGHT JOIN Masini m ON c.id_client = m.id_client LIMIT 0, 1000	2 row(s) returned
4	23:49:30	SELECT c.nume, p.num_piesa FROM Clienti c CROSS JOIN Piese p LIMIT 0, 1000	6 row(s) returned
5	23:49:30	SELECT nume FROM Clienti WHERE id_client IN (SELECT id_client FROM Masini WHERE marca = 'Dacia') LIMIT 0, 1000	1 row(s) returned

```
230    -- INNER JOIN
231 •   SELECT c.nume, m.marca, r.descriere
232     FROM Clienti c
233     JOIN Masini m ON c.id_client = m.id_client
234     JOIN Reparatii r ON m.id_masina = r.id_masina;
235
236    -- LEFT JOIN
237 •   SELECT c.nume, m.marca
238     FROM Clienti c
239     LEFT JOIN Masini m ON c.id_client = m.id_client;
240
241    -- RIGHT JOIN
242 •   SELECT c.nume, m.marca
243     FROM Clienti c
244     RIGHT JOIN Masini m ON c.id_client = m.id_client;
245
246    -- CROSS JOIN
247 •   SELECT c.nume, p.num_piesa
248     FROM Clienti c
249     CROSS JOIN Piese p;
250
251    -- SUBQUERY (bonus)
252 •   SELECT nume
253     FROM Clienti
254     WHERE id_client IN (
255         SELECT id_client FROM Masini WHERE marca = 'Dacia'
```



Mulțumesc !