SGBD - Proiect part 2

* Preluăm date de la client de pe front (din html-uri). Request-urile ajung la serverul nodejs, care apelează funcții și proceduri din baza de date Oracle.
* Pentru a ne conecta la baza de date folosim package-ul *node-oracledb*.
* În *server.js* menționăm ce se întâmplă când se primește un request de tip POST sau GET.
* Funcțiile returnează un obiect de tipul JSON.
* Am folosit triggere, de exemplu *updatePricesOnNewPromotions*.

1. Să fie apelată o funcție:

*var sql = "select signUp(:1, :2, :3, :4, :5, :6, :7, :8, :9) from dual";*

2. Să fie apelată o procedură:

*var sql = 'BEGIN placeOrder(:id\_user, :id\_product, :quantity); END;';*

3. Să fie captată o excepție din PLSQL:

În nodejs, tratăm excepțiile din PLSQL folosind *try{} catch{}*.

În PLSQL, tratăm excepții custom (de exemplu folosind package-ul *AuthenticationExceptions*).

4. Să fie interogată baza de date printr-un SQL care sa permită injectare:

Doar în funcția *getOrders*, am folosit SQL injection:

*var sql = 'select id from orders where id\_user = ' + id\_user;*

Pentru [http://localhost:3000/orders?id\_user=101 or 1=1](http://localhost:3000/orders?id_user=101%20or%201=1) se afișează toate id-urile pentru orders.

În rest, nu am permis SQL injection. Librăria *oracledb* se ocupă de prelucrarea valorilor astfel încât să nu permită injectare.

5. Să vedem un index și impactul lui asupra vitezei de reacție a aplicației:

Indecșii ne ajută când dorim să filtrăm produsele (de exemplu *index\_color*).

Afișarea produselor de o anumită culoare (verde), folosind index, se execută (în medie) în 50 secunde. Dacă nu folosim indexul, se execută în 80 secunde.

6. Paginări ale rezultatelor acolo unde este cazul:

Pentru pagina de produse, am folosit funcția *filterProducts* care afișează doar o porțiune de câte 50 de produse.

7. Complexitatea funcțiilor:

Am folosit funcția *filterProducts*, folosită la paginare, care returneză numai o porțiune de produse.

8. Folosiri de pachete adiționale sau alte elemente care nu au fost predate și cu motivația folosirii lor:

Funcția *format\_message* din package-ul *utl\_lms* este un fel de *printf* din C, adică formatează un string.

Imaginile sunt reținute codificate în baza 64, pentru că obiectele JSON nu suportă informații binare, ci doar string-uri. Aceste codificări pot fi foarte mari, de aceea am folosit tipul de date *clob*, care suportă până la 4GB.

9. Nr de înregistrări:

*users* - 100

*categories* - 50

*products* - 1000000

*promotions* - 50

*orders* - 100