Mihai Alexandru-Mario
Grupa 363

Temă de casă Sisteme Avansate Baze de Date

1.

a) Un exemplu de atribut multi-valoare în cazul unei baze de date legată de mașini ar fi adresele de e-mail ale proprietarului, deoarece acesta poate avea mai multe.

b) O mașină poate fi reparată în mai multe ateliere de reparații, la date (data_reparație) diferite. Dar un atelier de reparații poate repara mai multe mașini, deci relația „Repară” este many-to-many, iar atributul repetitiv este data efectuării lucrărilor (data_reparație).

c) La crearea design-ului logic al bazei de date, atributul multi-valoare de la a) va fi impărțit în două tabele, unul „Proprietar” cu coloanele „ID_proprietar”, „nume” etc. și un al doilea tabel, „Adrese_mail” cu coloanele „ID_proprietar”, „e_mail”. În cazul exemplului de la b), tabelul „Mașină” va avea coloanele „ID_mașină”, „ID_proprietar”, „Marcă”, „Model”, „Data_fabricației”, „Data_înmatriculării” etc. , tabelul „Atelier_reparații” cu coloanele „ID_atelier”, „nume” etc. Atributul repetitiv „data_reparație” va deveni și el un tabel separat, numit „Detalii_mentenanță” cu coloanele „ID_mașină”, „ID_service”, „data_reparație”. Prin „Detalii_mentenanță” am eliminat și relația many-to-many, dar și atributul repetitiv.

2.

a) Un exemplu de relație de tip 3 ar fi între tabelele „Producător”, „Mașină_produsă” și „Piesă”. Un producător poate produce și mașini dar își poate produce propriile piese pe care le folosește în mașinile sale sau folosi piese de la producători externi, numită „produce_include”. Mașina trebuie să aibă o listă de piese utilizate (pentru acest exemplu voi lăsa acest atribut ca multivaloare chiar dacă acesta trebuie eliminat în proiectare).

b)O relație aparentă de tip 3 ar fi între „Client”, „Mașină” și „Contract”. Clientul semnează unul sau mai multe contracte, iar contractul poate include mai multe mașini, relația fiind spartă în două relații many-to-many dar nu este o relație de tip 3.

3.

a)Tabelul „Mașină” cu atributele ID_mașină (PK), nr_înmatriculare (PK) – cheie primară compusă, model, an_fabricație. Acest tabel este în FN1 deoarece nu avem atribute multivaloare, dar nu este în FN2 deoarece anul de fabricație este determinat de nr_înmatriculare. Atributul model nu determină direct an_fabricație deoarece un model poate fi produs pe mai mulți ani, iar fiecare mașină are alt număr de înmatriculare. Pentru a aduce acest tabel în FN2, vom crea un tabel adițional „NrInmatriculare_AnFabricație” cu atributele nr_înmatriculare (PK) și an_fabricație. În tabelul „Mașină” vom elimina coloana an_fabricație, iar această informație ar fi accesată prin intermediul lui nr_înmatriculare.

b)Vom presupune tabelul „Proprietar”, cu atributele ID_proprietar (PK), nume, nr_tel_principal, e_mail, cnp, tip_asigurare, cost_asigurare,categorie_autovehicul. Din tip_asigurare putem determina costul ei, dar și tipul de autovehicul pe care îl conduce. Dacă are o mașină compactă, vom avea alt tip de asigurare și alt cost față de un SUV. Ca să aducem acest tabel în FN3 ar trebui să eliminăm din tabelul „Proprietar” coloanele cost_asigurare și categorie_autovehicul. Vom crea un tabel nou „Asigurare” cu tip_asigurare (PK), cost_asigurare și categorie_autovehicul.

Mihai Alexandru-Mario
Grupa 363

4. Avem un tabel „Asigurare_auto”, cu coloanele ID_asigurare, data_incepere și tip_asigurare. Unei polițe de asigurare îi pot corespunde mai multe tipuri (de exemplu RCA și CASCO). O dependență multivaloare care să nu fie și funcțională ar fi între ID_asigurare și tip_asigurare, deoarece nu putem determina individual tipul asigurării pentru o poliță necunoscând ambele valori.

ID_asigurare	data_incepere	tip_asigurare
1	01/01/2024	RCA
1	01/01/2024	CASCO
2	12/05/2023	CASCO
3	21/07/2023	RCA

5. Un index de tip B*, folosește în implementare un arbore de tip B*, care este optimizat pentru a stoca chei și valori, care se rebalansează automat. Acesta se folosește în general pe chei primare pentru a optimiza căutările. Un index Bitmap folosește un tabel de 0 și 1 în implementare și acesta accelerează căutările pe coloane care pot avea doar câteva valori, de exemplu tipul caroseriei unei mașini poate fi ori sedan, hatchback, suv, break sau offroad.

```
CREATE TABLE masina (
    vin VARCHAR2(17) PRIMARY KEY,
    brand VARCHAR2(30),
    model VARCHAR2(50),
    tip_caroserie VARCHAR2(15),
    an_fabricatie NUMBER,
    km NUMBER
);


CREATE INDEX idx_masina_vin
ON masina(vin);


SELECT *
FROM masina
WHERE vin = '5YJSA1CN5DFP00001';


CREATE BITMAP INDEX idx_masina_tip_caroserie
ON masina(tip_caroserie);
```

Mihai Alexandru-Mario
Grupa 363

```
SELECT *

FROM masina

WHERE tip_caroserie = 'Sedan';
```

7. Am folosi o vedere pentru a accesa o bază de date pentru a ne asigura de exemplu că angajații unui service auto văd doar informațiile esențiale ale mașinii la care lucrează, nu și date despre proprietar, asigurând protecția identității persoanei care deține mașina.

```
CREATE TABLE masina_serv (

    vin VARCHAR(17) PRIMARY KEY,

    km NUMBER,

    marca VARCHAR(50),

    model VARCHAR(50),

    proprietar VARCHAR(100),

    data_inmatriculare DATE,

    tip_asigurare VARCHAR(50)

);


INSERT INTO masina_serv (vin, km, marca, model, proprietar,
data_inmatriculare, tip_asigurare)

VALUES ('1HGCM82633A123456', 50000, 'Renault', 'Arkana', 'Alex Mihai',
TO_DATE('2023-02-01','YYYY-MM-DD'), 'RCA+Casco');


CREATE OR REPLACE VIEW vizualizare_op_service AS

SELECT vin, km, marca, model

FROM masina_serv;
```

8. Deoarece sistemul Oracle folosește implicit „Read commited”, uneori putem vedea date diferite în sesiuni de lucru diferite. Acest lucru se întâmplă deoarece până nu se dă comanda „COMMIT” la datele actualizate (rulăm o cerere UPDATE), nu vor fi scrise în baza de date, ci vor fi disponibile doar local, în sesiunea curentă de lucru, iar ceilalți utilizatori vor vedea copia inițială a bazei de date pe sistemul lor. Din acest motiv, putem rula comanda „Rollback” pentru a ne întoarce la forma inițială. După executarea comenzii „Commit”, comanda „Rollback” nu mai are niciun efect.

```
CREATE TABLE masina_demo_select (

    id_masina NUMBER PRIMARY KEY,
```

```
    marca VARCHAR2(50),
    model VARCHAR2(50),
    an_fabricatie NUMBER,
    pret NUMBER
);

INSERT INTO masina_demo_select VALUES (1, 'Dacia', 'Logan', 2006, 7200);

/*sesiune1*/
SELECT * FROM masina_demo_select WHERE id_masina = 1;

/*sesiune2*/
SELECT * FROM masina_demo_select WHERE id_masina = 1;

/*sesiune3*/
UPDATE masina_demo_select SET pret = 7550 WHERE id_masina = 1;
SELECT * FROM masina_demo_select WHERE id_masina = 1;
```

9. Un exemplu de interblocare ar fi un broker de asigurări ce dorește să modifice datele despre o mașină și tarifele de asigurare. Doi operatori vor executa aceste operații dar în ordine inversă. Primul operator va modifica datele despre mașină și după tabelul cu tarife. Acesta va trebui să aștepte ca al doilea operator să termine de modificat tarifele la asigurare și la rândul lui, el va aștepta ca primul operator să termine de modificat datele din tabelul mașină.

10. Voi scrie un trigger care nu permite introducerea unui kilometraj mai mic decât la ultima intrare în service a unei mașini. Nu s-ar fi putut impune această constrângere fără a folosi un trigger deoarece trebuie să vedem care este numărul precedent de kilometri și să îl comparăm cu cel pe care dorim să-l introducem.

```
CREATE TABLE masina (
    id_masina NUMBER PRIMARY KEY,
    brand VARCHAR2(50),
    an_fabricatie NUMBER,
    km NUMBER
);
```

Mihai Alexandru-Mario
Grupa 363

```
CREATE TABLE intrari_service (
    id_service NUMBER PRIMARY KEY,
    id_masina NUMBER,
    data_service DATE,
    km_service NUMBER,
    FOREIGN KEY (id_masina) REFERENCES masina(id_masina)
);


CREATE OR REPLACE TRIGGER verif_service_km
BEFORE INSERT OR UPDATE ON intrari_service
FOR EACH ROW
DECLARE
    km_actuali NUMBER;
BEGIN
    SELECT km INTO km_actuali
    FROM masina
    WHERE id_masina = :NEW.id_masina;


    IF :NEW.km_service < km_actuali THEN
        raise_application_error(-20001, 'Kilometrajul intrarii service nu
poate fi mai mic decat ultimul kilometraj al masinii!');
    END IF;
END;
```

12. Când se face RAISE cu o excepție predefinită, execuția script-ului SQL se oprește și se generează o eroare în IDE. Când folosim RAISE cu o instrucțiune definită de noi, execuția script-ului continuă, iar excepția este tratată de către blocul EXCEPTION din script-ul PL/SQL în modul definit de utilizator.

```
DECLARE
    /*exceptie definita de mine*/
    conditie_definita EXCEPTION;
```

Mihai Alexandru-Mario
Grupa 363

```
    var BOOLEAN := false;
BEGIN
    IF var = FALSE THEN
        /* ridicam exceptia definita mai sus*/
        RAISE conditie_definita;
    ELSE
        DBMS_OUTPUT.PUT_LINE('"var" este TRUE');
    END IF;
EXCEPTION
    WHEN conditie_definita THEN
        DBMS_OUTPUT.PUT_LINE('S-a declansat exceptia definita de mine. "var"
este egal cu FALSE.');
END;


DECLARE
    de_impartit NUMBER := 5;
    impartitor NUMBER := 0;
BEGIN
    IF impartitor = 0 THEN
        /*ridicam exceptia predefinita zero_divide*/
        RAISE ZERO_DIVIDE;
    ELSE
        DBMS_OUTPUT.PUT_LINE(de_impartit / impartitor);
    END IF;
END;
```