

**Uniwersytet Gdańskie**



## **SERWERY BAZY DANYCH**

**Projekt: Komis samochodowy**

**Środowisko: Oracle**

**Zadani:** Zadanie 8 (Oracle - typy obiektowe)

**Prowadzący:**  
dr Robert Fidytek

**Wykonała:**  
Paulina Kimak  
Nr albumu 292511

## Spis treści

1. Opis Projektu.....	2
2. Typy obiektowe.....	3
3. Tabele obiektowe oraz dane.....	6
4. REF/DEREF + SCOPE.....	8
5. Zapytania z VALUE/DEREF.....	10
6. Metody w typach.....	13
7. ALTER TYPE.....	14
8. MAP/ORDER.....	15

## 1. Opis Projektu

Zaprojektowano komis samochodowy, w którym przechowywane są oferty aut oraz dane klientów zainteresowanych zakupem. Komis rejestruje zdarzenia związane z kontaktem klienta z konkretnym autem, w szczególności jazdy próbne oraz rezerwacje oglądania.

- **Encja A (Samochód)** opisuje pojazd wystawiony w komisie wraz z ceną i danymi identyfikacyjnymi.
- **Encja B (Klient)** przechowuje dane kontaktowe osoby zainteresowanej.
- **Encja zdarzenia (Zdarzenie)** łączy klienta z autem i przechowuje informacje o typie zdarzenia oraz czasie jego wystąpienia.

### Relacje:

- jeden klient może mieć wiele zdarzeń,
- jedno auto może mieć wiele zdarzeń;
- każde zdarzenie dotyczy dokładnie jednego auta i jednego klienta.

Zdarzenie jest bytem, który wprost realizuje powiązanie pomiędzy dwoma bytami: Samochód (A) i Klient (B).

## 2. Typy obiektowe

Zdefiniowano trzy typy obiektowe:

- **T\_CAR** - encja A (samochód), atrybuty: car\_id, vin, marka, model, rok\_prod, cena\_netto, waluta wraz metodami

```
29  CREATE OR REPLACE TYPE t_car AS OBJECT (
30    car_id      NUMBER(10),
31    vin         VARCHAR2(20),
32    marka       VARCHAR2(30),
33    model       VARCHAR2(30),
34    rok_prod    NUMBER(4),
35    cena_netto NUMBER(10,2),
36    waluta     VARCHAR2(3),
37
38    -- (6a) Metoda obliczeniowa
39    MEMBER FUNCTION cena_brutto(p_vat NUMBER DEFAULT 0.23) RETURN NUMBER,
40
41    -- (6b) Metoda opisowa
42    MEMBER FUNCTION opis RETURN VARCHAR,
43
44    -- (8A) MAP METHOD - sortowanie po cenie netto
45    MAP MEMBER FUNCTION map_key RETURN NUMBER
46  );
47 /
48
49 CREATE OR REPLACE TYPE BODY t_car AS
50
51   MEMBER FUNCTION cena_brutto(p_vat NUMBER DEFAULT 0.23) RETURN NUMBER IS
52   BEGIN
53     RETURN ROUND(self.cena_netto * (1 + NVL(p_vat,0)), 2);
54   END;
55
56   MEMBER FUNCTION opis RETURN VARCHAR IS
57   BEGIN
58     RETURN self.marka || ' ' || self.model ||
59           ' (' || self.rok_prod || ')', VIN=' || self.vin;
60   END;
61
62   MAP MEMBER FUNCTION map_key RETURN NUMBER IS
63   BEGIN
64     RETURN NVL(self.cena_netto, 0);
65   END;
66
67 END;
68 /
69
```

The screenshot shows the Oracle SQL Developer interface with the 'Script output' tab selected. The script area displays the creation of the **t\_car** object type, including its attributes (car\_id, vin, marka, model, rok\_prod, cena\_netto, waluta) and three member functions: `cena_brutto`, `opis`, and `map_key`. The `cena_brutto` function calculates the brutto price by rounding the net price multiplied by a VAT rate (0.23). The `opis` function returns a string combining the brand, model, year of production, and VIN number. The `map_key` function returns the net price. The creation process is completed successfully with no errors.

```
SQL> PROMPT === [01] Typy obiektowe ===
===
SQL> CREATE OR REPLACE TYPE t_car AS OBJECT (
  car_id      NUMBER(10),
  vin         VARCHAR2(20),
  marka       VARCHAR2(30),
  model       VARCHAR2(30),
  rok_prod    NUMBER(4),
  cena_netto NUMBER(10,2),
  waluta     VARCHAR2(3),
)
/
Type T_CAR compiled
No errors.
Elapsed: 00:00:00.031
```

- **T\_KLIENT** - encja B (klient), atrybuty: klient\_id, imie, nazwisko, email, data\_ur.

```

72
73  CREATE OR REPLACE TYPE t_klient AS OBJECT (
74      klient_id    NUMBER(10),
75      imie         VARCHAR2(30),
76      nazwisko     VARCHAR2(40),
77      email        VARCHAR2(80),
78      data_ur      DATE,
79
80      MEMBER FUNCTION pelna_nazwa RETURN VARCHAR2
81  );
82 /
83
84  CREATE OR REPLACE TYPE BODY t_klient AS
85
86      MEMBER FUNCTION pelna_nazwa RETURN VARCHAR2 IS
87          BEGIN
88              RETURN self.imie || ' ' || self.nazwisko;
89          END;
90
91      END;
92 /
93

```

Query result    Script output **DBMS output**    Explain Plan    SQL history

---

SQL> CREATE OR REPLACE TYPE t\_klient AS OBJECT (

klient\_id NUMBER(10),  
imie VARCHAR2(30),  
nazwisko VARCHAR2(40),...

Show more...

---

Type T\_KLIENT compiled

No errors.  
Elapsed: 00:00:00.012

---

SQL> CREATE OR REPLACE TYPE BODY t\_klient AS

MEMBER FUNCTION pelna\_nazwa RETURN VARCHAR2 IS  
BEGIN...  
Show more...

- **T\_EVT** - encja zdarzenia, atrybuty: evt\_id, evt\_ts, evt\_typ, czas\_min, notatka oraz 2 referencje REF: car\_ref (REF T\_CAR) i klient\_ref (REF T\_KLIENT).

```

CREATE OR REPLACE TYPE t_evt AS OBJECT (
    evt_id      NUMBER(10),
    evt_ts      TIMESTAMP,
    evt_typ     VARCHAR2(20),
    car_ref     REF t_car,
    klient_ref  REF t_klient,
    czas_min    NUMBER(5),
    notatka     VARCHAR2(200),

    MEMBER FUNCTION koszt RETURN NUMBER,
    MEMBER FUNCTION to_text RETURN VARCHAR2
);
/
CREATE OR REPLACE TYPE BODY t_evt AS

    MEMBER FUNCTION koszt RETURN NUMBER IS
    BEGIN
        IF UPPER(self.evt_typ) = 'JAZDA' THEN
            RETURN NVL(self.czas_min,0) * 5;
        ELSE
            RETURN 0;
        END IF;
    END;

    MEMBER FUNCTION to_text RETURN VARCHAR2 IS
    BEGIN
        RETURN 'EVT['||self.evt_id||'] '||self.evt_typ||' @ ' ||
               TO_CHAR(self.evt_ts,'YYYY-MM-DD HH24:MI') ||
               ', min='||NVL(TO_CHAR(self.czas_min),'NULL') ||
               ', note='||NVL(self.notatka,'-');
    END;
END;
/

```

Query result    Script output    DBMS output    Explain Plan    SQL history

No errors.  
Elapsed: 00:00:00.008

---

SQL> CREATE OR REPLACE TYPE t\_evt AS OBJECT (
 evt\_id NUMBER(10),
 evt\_ts TIMESTAMP,
 evt\_typ VARCHAR2(20),...
Show more...

---

Type T\_EVT compiled

No errors.  
Elapsed: 00:00:00.010

---

SQL> CREATE OR REPLACE TYPE BODY t\_evt AS

```

    MEMBER FUNCTION koszt RETURN NUMBER IS
    BEGIN

```

### 3. Tabele obiektowe oraz dane

Utworzono tabele obiektowe:

```
CREATE TABLE car_tab OF t_car (
    CONSTRAINT car_pk PRIMARY KEY (car_id),
    CONSTRAINT car_vin_uq UNIQUE (vin)
);

CREATE TABLE klient_tab OF t_klient (
    CONSTRAINT klient_pk PRIMARY KEY (klient_id),
    CONSTRAINT klient_email_uq UNIQUE (email)
);

CREATE TABLE evt_tab OF t_evt (
    CONSTRAINT evt_pk PRIMARY KEY (evt_id)
);
```

The screenshot shows the Oracle SQL Developer interface with the 'Script output' tab selected. The output window displays the creation scripts for three tables: CAR\_TAB, Klient\_TAB, and EVT\_TAB. Each table is created with a primary key constraint and either a unique constraint or a unique index.

```
Query result Script output DBMS output Explain Plan SQL history

== [01] Tabele obiektove ==

SQL> CREATE TABLE car_tab OF t_car (
    CONSTRAINT car_pk PRIMARY KEY (car_id),
    CONSTRAINT car_vin_uq UNIQUE (vin)
)

Table CAR_TAB created.

Elapsed: 00:00:00.021

SQL> CREATE TABLE klient_tab OF t_klient (
    CONSTRAINT klient_pk PRIMARY KEY (klient_id),
    CONSTRAINT klient_email_uq UNIQUE (email)
)

Table Klient_TAB created.

Elapsed: 00:00:00.017

SQL> CREATE TABLE evt_tab OF t_evt (
    CONSTRAINT evt_pk PRIMARY KEY (evt_id)
)

Table EVT_TAB created.
```

## Inserty

Wstawiono co najmniej 10 rekordów do każdej z tabel:

### Car insert

```
[ SQL Worksheet ]* ▾ ▶ ⏷ ⏸ Aa ▾ ⏹
```

```
1 SET SERVEROUTPUT ON
2
3
4 PROMPT === [02] INSERT: CAR_TAB (A) ===
5 INSERT INTO car_tab VALUES (t_car(1,'VIN0000000000000001','Toyota','Corolla',2017, 42000,'PLN'));
6 INSERT INTO car_tab VALUES (t_car(2,'VIN0000000000000002','Toyota','Yaris',2019, 52000,'PLN'));
7 INSERT INTO car_tab VALUES (t_car(3,'VIN0000000000000003','Skoda','Octavia',2016, 39000,'PLN'));
8 INSERT INTO car_tab VALUES (t_car(4,'VIN0000000000000004','Volkswagen','Golf',2018, 61000,'PLN'));
9 INSERT INTO car_tab VALUES (t_car(5,'VIN0000000000000005','Ford','Focus',2015, 31000,'PLN'));
10 INSERT INTO car_tab VALUES (t_car(6,'VIN0000000000000006','BMW','320i',2014, 57000,'PLN'));
11 INSERT INTO car_tab VALUES (t_car(7,'VIN0000000000000007','Audi','A4',2013, 54000,'PLN'));
12 INSERT INTO car_tab VALUES (t_car(8,'VIN0000000000000008','Hyundai','i30',2020, 69000,'PLN'));
13 INSERT INTO car_tab VALUES (t_car(9,'VIN0000000000000009','Kia','Ceed',2021, 75000,'PLN'));
14 INSERT INTO car_tab VALUES (t_car(10,'VIN0000000000000010','Mazda','3',2018, 63000,'PLN'));
15 COMMIT;
16
```

---

```
SQL> PROMPT === [02] INSERT: CAR_TAB (A) ===

===[02] INSERT: CAR_TAB (A) ===

SQL> INSERT INTO car_tab VALUES (t_car(1,'VIN0000000000000001','Toyota','Corolla',2017, 42000,'PLN'))
```

1 row inserted.

```
Elapsed: 00:00:00.158
```

---

```
SQL> INSERT INTO car_tab VALUES (t_car(2,'VIN0000000000000002','Toyota','Yaris',2019, 52000,'PLN'))
```

### Client insert

```
PROMPT === [02] INSERT: KLIENT_TAB (B) ===
INSERT INTO klient_tab VALUES (t_klient(1,'Anna','Nowak','anna.nowak@example.com', DATE '1994-03-12'));
INSERT INTO klient_tab VALUES (t_klient(2,'Piotr','Kowalski','piotr.kowalski@example.com', DATE '1988-11-02'));
INSERT INTO klient_tab VALUES (t_klient(3,'Katarzyna','Wisniewska','k.wisniewska@example.com', DATE '1990-07-21'));
INSERT INTO klient_tab VALUES (t_klient(4,'Marek','Wojcik','marek.wojcik@example.com', DATE '1985-01-09'));
INSERT INTO klient_tab VALUES (t_klient(5,'Ewa','Kaczmarek','ewa.k@example.com', DATE '1997-05-30'));
INSERT INTO klient_tab VALUES (t_klient(6,'Tomasz','Mazur','t.mazur@example.com', DATE '1992-12-14'));
INSERT INTO klient_tab VALUES (t_klient(7,'Olga','Zielinska','olga.z@example.com', DATE '1999-09-03'));
INSERT INTO klient_tab VALUES (t_klient(8,'Pawel','Szymanski','pawel.s@example.com', DATE '1983-06-18'));
INSERT INTO klient_tab VALUES (t_klient(9,'Agnieszka','Dabrowska','aga.d@example.com', DATE '1991-02-26'));
INSERT INTO klient_tab VALUES (t_klient(10,'Michal','Lewandowski','m.lewandowski@example.com', DATE '1989-10-05'));
COMMIT;
```

---

```
PROMPT === [02] INSERT: EWT_TAB (B) ===
61 INSERT INTO evt_tab
```

Query result    Script output    DBMS output    Explain Plan    SQL history

Query result

```
SQL> PROMPT === [02] INSERT: KLIENT_TAB (B) ===

===[02] INSERT: KLIENT_TAB (B) ===

SQL> INSERT INTO klient_tab VALUES (t_klient(1,'Anna','Nowak','anna.nowak@example.com', DATE '1994-03-12'))
```

1 row inserted.

```
Elapsed: 00:00:00.133
```

---

```
SQL> INSERT INTO klient_tab VALUES (t_klient(2,'Piotr','Kowalski','piotr.kowalski@example.com', DATE '1988-11-02'))
```

1 row inserted.

```
Elapsed: 00:00:00.002
```

## 4. REF/DEREF + SCOPE

W typie T\_EVT zastosowano dwa pola REF: **car\_ref** oraz **klient\_ref**.

Utworzono tabele EVT\_TAB OF T\_EVT, a następnie ustawiono ograniczenia zakresu referencji (SCOPE IS):

- ALTER TABLE evt\_tab ADD SCOPE FOR (car\_ref) IS car\_tab;
- ALTER TABLE evt\_tab ADD SCOPE FOR (klient\_ref) IS klient\_tab;

```
[ SQL Worksheet ]* ▾ ▷ ⚡ ⌂ ⌁ Aa ▾ ⏪  
1 ---  
2 -- Tabela pomocnicza do testu REF poza SCOPE  
3 ---  
4 CREATE TABLE car_tab_tmp OF t_car;  
5 ---  
7 -- (4) SCOPE dla referencji  
8 ---  
9 ALTER TABLE evt_tab ADD SCOPE FOR (car_ref) IS car_tab;  
10 ALTER TABLE evt_tab ADD SCOPE FOR (klient_ref) IS klient_tab;
```

Wstawiono 15 rekordów do EVT\_TAB

```
[ SQL Worksheet ]* ▾ ▷ ⚡ ⌂ ⌁ Aa ▾ ⏪  
32 INSERT INTO evt_tab  
33 SELECT t_evt(101, SYSTIMESTAMP-INTERVAL '10' DAY, 'JAZDA', REF(c), REF(k), 30, 'pierwsza jazda')  
34 FROM car_tab c, klient_tab k WHERE c.car_id=1 AND k.klient_id=1;  
35  
36 INSERT INTO evt_tab  
37 SELECT t_evt(102, SYSTIMESTAMP-INTERVAL '9' DAY, 'REZERW', REF(c), REF(k), NULL, 'rezerwacja na ogladanie')  
38 FROM car_tab c, klient_tab k WHERE c.car_id=2 AND k.klient_id=2;  
39  
40 INSERT INTO evt_tab  
41 SELECT t_evt(103, SYSTIMESTAMP-INTERVAL '9' DAY, 'JAZDA', REF(c), REF(k), 20, 'miasto')  
42 FROM car_tab c, klient_tab k WHERE c.car_id=2 AND k.klient_id=3;  
43  
44 INSERT INTO evt_tab  
45 SELECT t_evt(104, SYSTIMESTAMP-INTERVAL '8' DAY, 'JAZDA', REF(c), REF(k), 45, 'trasa S8')  
46 FROM car_tab c, klient_tab k WHERE c.car_id=3 AND k.klient_id=4;  
47  
48 INSERT INTO evt_tab  
49 SELECT t_evt(105, SYSTIMESTAMP-INTERVAL '8' DAY, 'REZERW', REF(c), REF(k), NULL, 'zaliczka potwierdzona')  
50 FROM car_tab c, klient_tab k WHERE c.car_id=4 AND k.klient_id=5;  
51  
52 INSERT INTO evt_tab  
53 SELECT t_evt(106, SYSTIMESTAMP-INTERVAL '7' DAY, 'JAZDA', REF(c), REF(k), 15, 'parking')  
54 FROM car_tab c, klient_tab k WHERE c.car_id=4 AND k.klient_id=1;  
55  
56 INSERT INTO evt_tab  
57 SELECT t_evt(107, SYSTIMESTAMP-INTERVAL '7' DAY, 'JAZDA', REF(c), REF(k), 35, 'obwodnica')  
58 FROM car_tab c, klient_tab k WHERE c.car_id=5 AND k.klient_id=6;  
59  
60 INSERT INTO evt_tab  
61 SELECT t_evt(108, SYSTIMESTAMP-INTERVAL '6' DAY, 'REZERW', REF(c), REF(k), NULL, 'czeka na finansowanie')  
62 FROM car_tab c, klient_tab k WHERE c.car_id=6 AND k.klient_id=7;  
63  
64 INSERT INTO evt_tab  
65 SELECT t_evt(109, SYSTIMESTAMP-INTERVAL '6' DAY, 'JAZDA', REF(c), REF(k), 25, 'krotka jazda')  
66 FROM car_tab c, klient_tab k WHERE c.car_id=6 AND k.klient_id=8;  
67  
68 INSERT INTO evt_tab  
69 SELECT t_evt(110, SYSTIMESTAMP-INTERVAL '5' DAY, 'JAZDA', REF(c), REF(k), 40, 'autostrada')  
70 FROM car_tab c, klient_tab k WHERE c.car_id=7 AND k.klient_id=9;  
71  
72 INSERT INTO evt_tab  
73 SELECT t_evt(111, SYSTIMESTAMP-INTERVAL '5' DAY, 'REZERW', REF(c), REF(k), NULL, 'rezerwacja weekend')  
74 FROM car_tab c, klient_tab k WHERE c.car_id=8 AND k.klient_id=10;  
75  
76 INSERT INTO evt_tab  
77 SELECT t_evt(112, SYSTIMESTAMP-INTERVAL '4' DAY, 'JAZDA', REF(c), REF(k), 50, 'trasa podmiejska')  
78 FROM car_tab c, klient_tab k WHERE c.car_id=8 AND k.klient_id=2;
```

```
Query result Script output DBMS output Explain Plan SQL history  
SQL> PROMPT === [02] INSERT: EVT_TAB (zdarzenia) ===  
==== [02] INSERT: EVT_TAB (zdarzenia) ===  
SQL> INSERT INTO evt_tab  
  SELECT t_evt(101, SYSTIMESTAMP-INTERVAL '10' DAY, 'JAZDA', REF(c), REF(k), 30, 'pierwsza jazda')  
  FROM car_tab c, klient_tab k WHERE c.car_id=1 AND k.klient_id=1  
  
1 row inserted.  
Elapsed: 00:00:00.088  
  
SQL> INSERT INTO evt_tab  
  SELECT t_evt(102, SYSTIMESTAMP-INTERVAL '9' DAY, 'REZERW', REF(c), REF(k), NULL, 'rezerwacja na ogladanie')  
  FROM car_tab c, klient_tab k WHERE c.car_id=2 AND k.klient_id=2
```

## Przypadek testowy

Dodatkowo wykonano przypadek testowy, który próbuje wstawić zdarzenie z REF wskazującym na obiekt auta w innej tabeli (CAR\_TAB\_TMP).

The screenshot shows the FreeSQL interface with the 'Worksheet' tab selected. The left sidebar has 'Navigator' and 'Files' tabs, with 'My Schema' and 'Tables' dropdowns. A search bar says 'Search objects'. The main area is titled '[SQL Worksheet]\*' and contains the following PL/SQL code:

```
PROMPT === [02] Przypadek testowy: bledny REF (niezgodny SCOPE) ===
-- CAR_TAB_TMP został utworzony w 01_create.sql
INSERT INTO car_tab_tmp VALUES (t_car(999,'VIN9999999999999999','Tesla','Model 3',2022, 160000,'PLN'));
COMMIT;

DECLARE
    v_err VARCHAR2(4000);
BEGIN
    -- Proba wstawienia zdarzenia z REF do CAR_TAB_TMP (poza zakresem SCOPE dla car_ref)
    INSERT INTO evt_tab
    SELECT t_evt(
        999,
        SYSTIMESTAMP,
        'JAZDA',
        REF(c2),
        (SELECT REF(k) FROM klient_tab k WHERE k.klient_id=1),
        10,
        'NIE POWINNO SIE WSTAWIC'
    )
    FROM car_tab_tmp c2
    WHERE c2.car_id = 999;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('BLAD: wstawiono zdarzenie z ref poza SCOPE (to nie powinno sie wydarzyc.');
EXCEPTION
    WHEN OTHERS THEN
        v_err := SQLERRM;
        DBMS_OUTPUT.PUT_LINE('OCZEKIWANE: blad przy wstawianiu zdarzenia z REF poza SCOPE.');
        DBMS_OUTPUT.PUT_LINE('Szczegoly: '||v_err);
        ROLLBACK;
END;
/
```

Z powodu ustawionego SCOPE, taki INSERT zostaje odrzucony przez Oracle (błąd wstawiania REF poza SCOPE), co jest widoczne w DBMS\_OUTPUT.

The screenshot shows the FreeSQL interface with the 'DBMS output' tab selected. It displays the following error message:

OCZEKIWANE: blad przy wstawianiu zdarzenia z REF poza SCOPE.  
Szczegoly: ORA-22889: REF value does not point to scoped table

Query result    Script output **DBMS output**    Explain Plan    SQL history

Commit complete.

Elapsed: 00:00:00.002

---

```
SQL> DECLARE
      v_err VARCHAR2(4000);
    BEGIN
      -- Proba wstawienia zdarzenia z REF do CAR_TAB_TMP (poza zakresem SCOPE dla car_ref)...
Show more...
```

OCZEKIWANE: blad przy wstawianiu zdarzenia z REF poza SCOPE.  
Szczegoly: ORA-22889: REF value does not point to scoped table

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010

## 5. Zapytania z VALUE/DEREF

W pliku 03\_tests.sql znajdują się 3 zapytania, które wykorzystują VALUE i/lub DEREF:

### 1. Lista zdarzeń wraz z danymi auta i klienta pobranymi przez DEREF.

The screenshot shows the FreeSQL application interface. On the left is the Navigator pane, which includes a 'My Schema' dropdown set to 'My Schema', a 'Tables' dropdown, and a 'Search objects' input field. Below these are expandable sections for 'CAR\_TAB', 'CAR\_TAB\_TMP', 'EVT\_TAB', and 'Klient\_TAB'. On the right is the 'SQL Worksheet' tab, which contains the following SQL code:

```

1  PROMPT === [03] (5) Zapytanie 1: lista zdarzen + dane obiektu A i B przez DEREF ===
2  -- OCZEKIWANE: 15 wierszy (evt_id 101..115), z marka/model, Klient, oraz wyliczony koszt.
3  SELECT
4    e.evt_id,
5    e.evt_typ,
6    TO_CHAR(e.evt_ts,'YYYY-MM-DD') AS dzien,
7    DEREF(e.car_ref).marka AS marka,
8    DEREF(e.car_ref).model AS model,
9    DEREF(e.klient_ref).imie AS imie,
10   DEREF(e.klient_ref).nazwisko AS nazwisko,
11   e.koszt() AS koszt_pln,
12   e.to_text() AS opis_evt
13  FROM evt_tab e
14  ORDER BY e.evt_id;

```

Query result    Script output    DBMS output    Explain Plan    SQL history

Download Execution time: 0.037 seconds

	EVT_ID	EVT_TYP	DZIEN	MARKA	MODEL	IMIE	NAZWISKO	KOSZT_
1	101	JAZDA	2026-01-06	Toyota	Corolla	Anna	Nowak	
2	102	REZERW	2026-01-07	Toyota	Yaris	Piotr	Kowalski	
3	103	JAZDA	2026-01-07	Toyota	Yaris	Katarzyna	Wisniewska	
4	104	JAZDA	2026-01-08	Skoda	Octavia	Marek	Wojcik	
5	105	REZERW	2026-01-08	Volkswagen	Golf	Ewa	Kaczmarek	
6	106	JAZDA	2026-01-09	Volkswagen	Golf	Anna	Nowak	
7	107	JAZDA	2026-01-09	Ford	Focus	Tomasz	Mazur	
8	108	REZERW	2026-01-10	BMW	320i	Olga	Zielinska	
9	109	JAZDA	2026-01-10	BMW	320i	Pawel	Szymanski	
10	110	JAZDA	2026-01-11	Audi	A4	Agnieszka	Dabrowska	
11	111	REZERW	2026-01-11	Hyundai	i30	Michal	Lewandowski	
12	112	JAZDA	2026-01-12	Hundai	i30	Piotr	Kowalski	

## 2. Raport zagregowany: liczba zdarzeń i suma kosztów na auto (GROUP BY po REF).

FreeSQL    Worksheet    Library

Navigator    Files

My Schema    Tables    Search objects

```
[ SQL Worksheet ]* ▶ ▷ ⌂ ⌂ ⌂ Aa ⌂
```

```

1 PROMPT === [03] (5) Zapytanie 2: agregacja - liczba zdarzen na samochod ===
2 -- OCZEKIWANE: do 10 wierszy; pokazuje liczbę zdarzeń i sumę kosztów na VIN
3
4 SELECT
5   | DEREF(e.car_ref).vin AS vin,
6   | DEREF(e.car_ref).marka || ' ' || DEREF(e.car_ref).model AS auto,
7   | COUNT(*) AS liczba_zdarzen,
8   | SUM(e.koszt()) AS suma_kosztow_pln
9   FROM evt_tab e
10  GROUP BY
11    | DEREF(e.car_ref).vin,
12    | DEREF(e.car_ref).marka,
13    | DEREF(e.car_ref).model,
14    ORDER BY liczba_zdarzen DESC, suma_kosztow_pln DESC;
15

```

Query result    Script output    DBMS output    Explain Plan    SQL history

Download Execution time: 0.012 seconds

	VIN	AUTO	LICZBA_ZDARZEN	SUMA_KOSZTOW_P
1	VIN0000000000000000	Mazda 3	2	250
2	VIN0000000000000000	Mazda 3	2	150
3	VIN0000000000000000	Mazda 3	2	125
4	VIN0000000000000000	Mazda 3	2	100
5	VIN0000000000000000	Mazda 3	2	75
6	VIN0000000000000000	Mazda 3	1	225
7	VIN0000000000000000	Mazda 3	1	200
8	VIN0000000000000000	Mazda 3	1	175
9	VIN0000000000000000	Mazda 3	1	150
10	VIN0000000000000000	Mazda 3	1	100

### 3. Filtrowanie po atrybutie auta pobranym przez DEREF (rok\_prod < 2017).

The screenshot shows the FreeSQL interface with a SQL worksheet containing the following code:

```
PROMPT === [03] - (5) Zapytanie 3: filtrowanie po atrybutie pobranym przez DEREF ===  
-- OCZEKIWANE: tylko zdarzenia dla aut z rokiem produkcji < 2017.  
SELECT  
    e.evt_id,  
    DEREF(e.car_ref).rok_prod AS rok,  
    DEREF(e.car_ref).marka AS marka,  
    DEREF(e.klient_ref).nazwisko AS klient  
FROM evt_tab e  
WHERE DEREF(e.car_ref).rok_prod < 2017  
ORDER BY e.evt_id;
```

The query result table has columns: EVT\_ID, ROK, MARKA, KLIENT. The data is:

EVT_ID	ROK	MARKA	KLIENT
1	104	2016	Skoda
2	107	2015	Ford
3	108	2014	BMW
4	109	2014	BMW
5	110	2013	Audi

Dodatkowo: zapytanie pokazujące użycie VALUE(c) zwracające cały obiekt auta.

The screenshot shows the FreeSQL interface with a SQL worksheet containing the following code:

```
PROMPT === [03] - (5) Uzycie VALUE(alias) - pobranie całego obiektu ===  
-- OCZEKIWANE: zwraca obiekty T_CAR; w LiveSQL widoczne kolumny obiektu.  
SELECT VALUE(c) AS car_obj  
FROM car_tab c  
WHERE c.cena_netto >= 65000  
ORDER BY c.cena_netto DESC;
```

The query result table has a single column: CAR\_OBJ. The data is:

CAR_OBJ
[{"car_id":9,"vin":"VIN"}]
[{"car_id":8,"vin":"VIN"}]

## 6. Metody w typach

Zaimplementowano metody w typach:

- **T\_CAR.cena\_brutto(p\_vat)** - metoda obliczeniowa (zwrot NUMBER), wylicza cenę brutto na podstawie cena\_netto.
- **T\_CAR.opis()** - metoda opisowa (zwrot VARCHAR2), buduje opis auta (marka, model, rok, VIN).

Wywołania metod pokazano w SQL w 03\_tests.sql.

```
PROMPT === [03] (6) Wywolanie metod typu T_CAR ===
-- OCZEKIWANE: opis oraz cena_brutto policzona metoda.
SELECT
  c.car_id,
  c.opis() AS opis,
  c.cena_netto,
  c.cena_brutto(0.23) AS cena_brutto_23
FROM car_tab c
ORDER BY c.car_id;
1
2
3
4
5
6
7
8
9
10
```

CAR_ID	OPIS	CENA_NETTO	CENA_BRUTTO_23
1	Toyota Corolla (2017)	42000	51660
2	Toyota Yaris (2019), V	52000	63960
3	Skoda Octavia (2016)	39000	47970
4	Volkswagen Golf (2018)	61000	75030
5	Ford Focus (2015), V	31000	38130
6	BMW 320i (2014), VI	57000	70110
7	Audi A4 (2013), VIN=	54000	66420
8	Hyundai i30 (2020), V	69000	84870
9	Kia Ceed (2021), VIN=	75000	92250
10	Mazda 3 (2018), VIN=	63000	77490

- **T\_EVT.koszt()** - metoda obliczeniowa, dla typu 'JAZDA' liczy koszt (czas\_min \* 5), dla innych typów 0.
- **T\_EVT.to\_text()** - metoda opisowa, zwraca tekstowy opis zdarzenia.

Wywołania metod pokazano w SQL w 03\_tests.sql.

```
PROMPT === [03] (5) Zapytanie 1: lista zdarzen + dane obiektu A i B przez Deref ===
-- OCZEKIWANE: 15 wierszy (evt_id 101..115), z marka/model, Klient, oraz wyliczony koszt.
SELECT
  e.evt_id,
  e.evt_typ,
  TO_CHAR(e.evt_te,'YYYY-MM-DD') AS dzien,
  Deref(e.car_ref).marka AS marka,
  Deref(e.car_ref).model AS model,
  Deref(e.klient_ref).imie AS imie,
  Deref(e.klient_ref).nazwisko AS nazwisko,
  e.koszt() AS koszt_pln,
  e.to_text() AS opis_evt
FROM evt_tab e
ORDER BY e.evt_id;
```

DZIEN	MARKA	MODEL	IMIE	NAZWISKO	KOSZT_PLN	OPIS_EVT
2026-01-06	Toyota	Corolla	Anna	Nowak	150	EVT[101] JAZDA @ .
2026-01-07	Toyota	Yaris	Piotr	Kowalski	0	EVT[102] REZERW @ .
2026-01-07	Toyota	Yaris	Katarzyna	Wisniewska	100	EVT[103] JAZDA @ .
2026-01-08	Skoda	Octavia	Marek	Wojcik	225	EVT[104] JAZDA @ .
2026-01-08	Volkswagen	Golf	Ewa	Kaczmarek	0	EVT[105] REZERW @ .
2026-01-09	Volkswagen	Golf	Anna	Nowak	75	EVT[106] JAZDA @ .
2026-01-09	Ford	Focus	Tomasz	Mazur	175	EVT[107] JAZDA @ .

## 7. ALTER TYPE

W pliku 03\_tests.sql wykonano ALTER TYPE na T\_CAR: dodano atrybut SEGMENT (VARCHAR2(20)) z CASCADE.

Następnie zaktualizowano istniejące rekordy w CAR\_TAB, ustawiając segment dla wybranych aut, i potwierdzono działanie poprzez SELECT, ze nowy atrybut jest dostępny na istniejących danych.

The screenshot shows the Oracle SQL Developer interface. The Navigator pane on the left lists tables: CAR\_TAB, CAR\_TAB\_TMP, EVT\_TAB, and KLIENT\_TAB. The SQL Worksheet pane in the center contains the following SQL code:

```
PROMPT === [03] (7) ALTER TYPE: dodanie atrybutu SEGMENT do T_CAR ===
-- Dodanie nowego elementu do typu na istniejących danych.
ALTER TYPE t_car ADD ATTRIBUTE (segment VARCHAR2(20)) CASCADE;
-- Ustawiamy segment dla aut
UPDATE car_tab c SET c.segment = 'Kompakt' WHERE c.car_id IN (1,3,4,5,8,9,10);
UPDATE car_tab c SET c.segment = 'Premium' WHERE c.car_id IN (6,7);
UPDATE car_tab c SET c.segment = 'Miejski' WHERE c.car_id IN (2);
COMMIT;
-- OCZEKIWANE: segment jest widoczny na istniejących rekordach.
SELECT c.car_id, c.marka, c.model, c.segment
FROM car_tab c
ORDER BY c.car_id;
```

The Query result pane at the bottom displays the updated data from the CAR\_TAB table:

CAR_ID	MARKA	MODEL	SEGMENT
1	Toyota	Corolla	Kompakt
2	Toyota	Yaris	Miejski
3	Skoda	Octavia	Kompakt
4	Volkswagen	Golf	Kompakt
5	Ford	Focus	Kompakt
6	BMW	320i	Premium
7	Audi	A4	Premium
8	Hyundai	i30	Kompakt
9	Kia	Ceed	Kompakt
10	Mazda	3	Kompakt

## 8. MAP/ORDER

### Wybrano opcje A (MAP/ORDER).

W typie T\_CAR dodano MAP MEMBER FUNCTION map\_key RETURN NUMBER, która zwraca cena\_netto. Dzięki temu Oracle może porównywać obiekty typu T\_CAR podczas sortowania.

W 03\_tests.sql pokazano sortowanie: ORDER BY VALUE(c), w którym posortowano auta rosnąco po cena\_netto.

The screenshot shows a FreeSQL interface with a SQL Worksheet tab open. The code in the worksheet is:

```
PROMPT === [03] (8A) MAP METHOD: sortowanie po obiekcie (ORDER BY VALUE) ===
-- OCZEKIWANE: auta posortowane rosnaco po cena_netto (map_key).
SELECT c.car_id, c.marka, c.model, c.cena_netto
FROM car_tab c
ORDER BY VALUE(c);
```

The Query result pane displays the following table:

	CAR_ID	MARKA	MODEL	CENA_NETTO
1		5 Ford	Focus	31000
2		3 Skoda	Octavia	39000
3		1 Toyota	Corolla	42000
4		2 Toyota	Yaris	52000
5		7 Audi	A4	54000
6		6 BMW	320i	57000
7		4 Volkswagen	Golf	61000
8		10 Mazda	3	63000
9		8 Hyundai	i30	69000
10		9 Kia	Ceed	75000