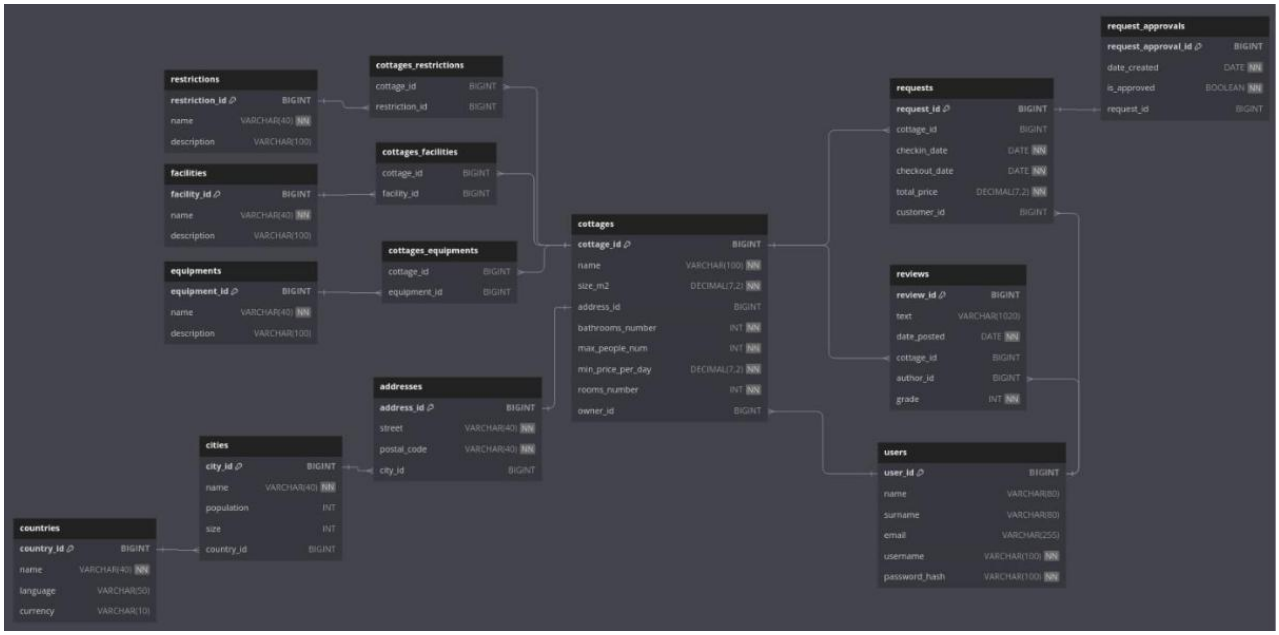
BD1 – Opis rozwiązania

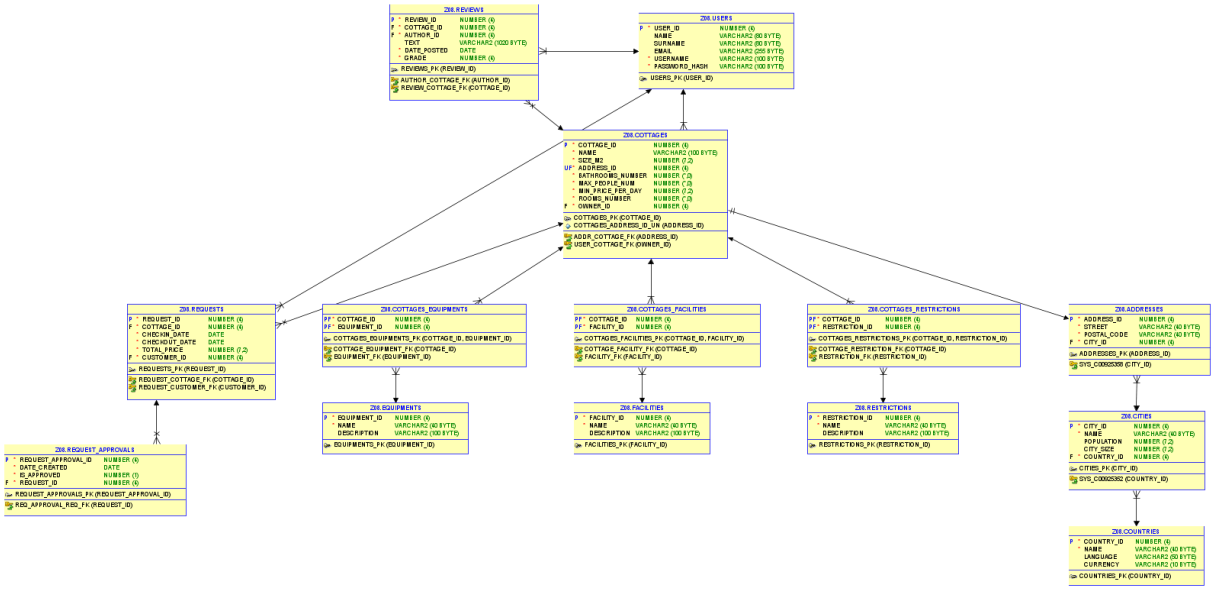
Aplikacja umożliwia:

- Rejestrację i logowanie
- Dodawanie, usuwanie i edytowanie domów
- Wyszukiwanie domów
- Dodawanie i usuwanie rezerwacji
- Zatwierdzanie rezerwacji
- Dodawanie i usuwanie opinii

Model ER



Model relacyjny



Tabele:

1. Users

Opis: Przechowuje informacje o użytkownikach: imię, nazwisko, adres e-mail, nazwę użytkownika i zaszyfrowane hasło.

Relacje:

- Użytkownik może być właścicielem wielu domków.
- Użytkownik może tworzyć wiele opinii.
- Użytkownik może składać wiele rezerwacji.

2. Cottages

Opis: Przechowuje szczegółowe informacje o domach: nazwę, wielkość (m²), adres, liczbę pokoi (łazienek), maks. liczbę osób, cenę za dzień i właściciela.

Relacje:

- Każdy domek ma właściciela (użytkownik).
- Każdy domek ma przypisany adres.
- Każdy domek może mieć wiele rezerwacji, opinii, udogodnień, wyposażenia i ograniczeń.

3. Adresses

Opis: Przechowuje adresy przypisane do domów: ulicę, kod pocztowy i miasto.

Relacje:

- Każdy adres jest powiązany z miastem.
- Każdy domek ma przypisany jeden adres.

4. Cities

Opis: Przechowuje informacje o miastach: nazwę, populację, wielkość i kraj.

Relacje:

- Każde miasto jest powiązane z krajem.
- Każde miasto może mieć wiele adresów.

5. Countries

Opis: Przechowuje informacje o krajach: nazwę, język urzędowy i walutę.

Relacje:

- Każdy kraj może mieć wiele miast.

6. Facilities

Opis: Przechowuje informacje o dostępnych udogodnieniach: nazwę i opis.

7. Cottages_facilities

Opis: Łączy domki z udogodnieniami. Dany dom może mieć wiele udogodnień.

8. Equipments

Opis: Przechowuje informacje o wyposażeniu: nazwę i opis.

9. Cottages_equipments

Opis: Łączy domy z wyposażeniem. Dany dom może mieć wiele elementów wyposażenia.

10. Restrictions

Opis: Przechowuje informacje o ograniczeniach: nazwę i opis.

11. Cottages_restrictions

Opis: Łączy domki z ograniczeniami. Dany dom może mieć wiele ograniczeń.

12. Requests

Opis: Przechowuje informacje o rezerwacjach: którego domu dotyczy, datę zameldowania i wymeldowania, całkowitą cenę i danego klienta.

13. Request_approvals

Opis: Przechowuje informacje o zatwierdzeniach rezerwacji: datę utworzenia zatwierdzenia, status (zatwierdzony lub odrzucony) i rezerwację, której dotyczy.

14. Reviews

Opis: Przechowuje opinie użytkowników: treść, datę publikacji, ocenę, dom, którego dotyczy i użytkownika wystawiającego opinię.

Sekwencje

Wszystkie sekwencje mają podobną strukturę:

```
CREATE SEQUENCE COUNTRIES_SEQ START WITH 1 INCREMENT BY 1;
```

```
CREATE SEQUENCE CITIES_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE ADDRESSES_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE USERS_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE COTTAGES_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE FACILITIES_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE EQUIPMENTS_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE RESTRICTIONS_SEQ START WITH 1 INCREMENT BY
1; CREATE SEQUENCE REVIEWS_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE REQUESTS_SEQ START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE REQUEST_APPROVALS_SEQ START WITH 1 INCREMENT BY 1;
```

- Zaczynają się od 1.
- Inkrementują o 1.
- Służą do generowania unikalnych identyfikatorów dla odpowiednich tabel.

Funkcje

1. **is_grade_good(v_grade number)** – Zwraca 1, jeśli ocena jest większa od 3, w przeciwnym razie 0.

```
CREATE OR REPLACE FUNCTION IS_GRADE_GOOD(V_GRADE NUMBER)
RETURN NUMBER
AS
    V_RESULT NUMBER;
BEGIN
    IF V_GRADE > 3 THEN
        V_RESULT := 1;
    ELSE
        V_RESULT := 0;
    END IF;
    RETURN V_RESULT;
END;
```

Przykład: is_grade_good(2) – 0, is_grade_good(4) - 1

2. **request_status(p_request_id number)** – Sprawdza status rezerwacji, zwraca:
    - Accepted – Rezerwacja została rozpatrzona pozytywnie (istnieje request_approval o is_approved = 1)
    - Rejected – Rezerwacja została odrzucona (istnieje request_approval o is_approved = 0)
    - Pending – Rezerwacja jeszcze nie została rozpatrzona (nie istnieje request_approval dla daneg rezerwacji)

```
CREATE OR REPLACE FUNCTION REQUEST_STATUS(P_REQUEST_ID NUMBER)
RETURN VARCHAR2
AS
    V_IS_APPROVED NUMBER;
BEGIN
    SELECT RA.IS_APPROVED
    INTO V_IS_APPROVED
    FROM REQUESTS R
    LEFT JOIN REQUEST_APPROVALS RA ON RA.REQUEST_ID = R.REQUEST_ID
    WHERE R.REQUEST_ID = P_REQUEST_ID;

    IF V_IS_APPROVED = 1 THEN
        RETURN 'Accepted';
    ELSIF V_IS_APPROVED = 0 THEN
        RETURN 'Rejected';
    ELSE
        RETURN 'Pending';
    END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN RETURN 'Null';
END;
```

Przykład:

- request_status(1) – ‘Pending’ - jeśli istnieje rezerwacja o id=1, nie istnieje zatwierdzenie danej rezerwacji
- request_status(1) – ‘Accepted’ - jeśli istnieje rezerwacja o id=1, istnieje zatwierdzenie danej rezerwacji o is_approved = 1 (pozytywnie)
- request_status(1) – ‘Rejected’ - jeśli istnieje rezerwacja o id=1, istnieje zatwierdzenie danej rezerwacji o is_approved = 0 (niegatywnie)

3. **calculate_total_tax(p_request_id number)** - oblicza łączny podatek dla rezerwacji domu na podstawie liczby dni wynajmu oraz ceny za dzień. Uwzględnia różne stawki podatkowe w zależności od długości pobytu. ()

```
CREATE OR REPLACE FUNCTION CALCULATE_TOTAL_TAX(P_REQUEST_ID NUMBER)
RETURN NUMBER
AS
  V_DAYS  NUMBER;
  V_TOTAL_PRICE  DECIMAL(7, 2) := 0;
  V_PRICE_PER_DAY DECIMAL(7, 2);
  V_COTTAGE_ID NUMBER;
  V_TOTAL_TAX DECIMAL(7, 2) := 0;
  V_FIRST_WEEK_TAX DECIMAL(7, 2) := 0;
```

```
  V_SECOND_WEEK_TAX DECIMAL(7, 2) := 0;
BEGIN
    SELECT (CHECKOUT_DATE - CHECKIN_DATE), COTTAGE_ID
    INTO   V_DAYS, V_COTTAGE_ID
    FROM   REQUESTS
    WHERE  REQUEST_id = P_REQUEST_ID;

    SELECT MIN_PRICE_PER_DAY
    INTO V_PRICE_PER_DAY
    FROM COTTAGES
    WHERE COTTAGE_ID = V_COTTAGE_ID;

    V_TOTAL_PRICE := V_DAYS * V_PRICE_PER_DAY;
    V_FIRST_WEEK_TAX := 0.36 * 7 + 0.33 * 7* V_PRICE_PER_DAY;
    V_SECOND_WEEK_TAX := 0.26 * 7 + 0.23 * 7* V_PRICE_PER_DAY;
    IF V_DAYS = 1 THEN
        V_TOTAL_TAX := 10;
    ELSIF V_DAYS < 7 THEN
        V_TOTAL_TAX := 0.36 * V_DAYS + 0.33 *
    V_TOTAL_PRICE; ELSIF V_DAYS < 14 THEN
        V_TOTAL_TAX := V_FIRST_WEEK_TAX + 0.26 * (V_DAYS - 7) + 0.23
* (V_DAYS - 7) * V_PRICE_PER_DAY;
    ELSE
        V_TOTAL_TAX := V_FIRST_WEEK_TAX + V_SECOND_WEEK_TAX + 0.15
* (V_DAYS - 14) + 0.12 * (V_DAYS - 14) * V_PRICE_PER_DAY;
    END IF;

    RETURN V_TOTAL_TAX;
END;
```

- **1 dzień:** Stała wartość podatku wynosi **10**.
- **Mniej niż 7 dni:** Podatek wynosi **0.36** za każdy dzień + **0.33** ceny za każdy dzień.
- **Od 7 do 13 dni:** Podatek za pierwszy tydzień + **0.26** za każdy kolejny dzień + **0.23** ceny za każdy kolejny dzień.
- **14 dni i więcej:** Podatek za pierwszy i drugi tydzień + **0.15** za każdy dodatkowy dzień + **0.12** ceny za każdy dodatkowy dzień.

Przykład: istnieje rezerwacja o request_id = 1, liczba dni = 9, cena za dzień domu = 100
Całkowita cena: 9 * 100 = 900,
Podatek za 7 dni: 233.52,
Podatek za 2 dni: 0.26 * 2 + 0.23 * 2 * 100 = 0.52 + 46 = 46.52
Łącznie: 280.04

4. **avg_price_in_country_with_tax(p_country_id long)** - oblicza średnią cenę wynajmu domu w kraju o danym country_id, z uwzględnieniem podatków.

```
CREATE OR REPLACE FUNCTION AVG_PRICE_IN_COUNTRY_WITH_TAX
(P_COUNTRY_ID LONG)
RETURN NUMBER
AS
  V_AVG_PRICE  DECIMAL(7, 2) := 0;
  V_MIN_PRICE  DECIMAL(7, 2) := 0;
  V_TOTAL_PRICE  DECIMAL(7, 2) := 0;
BEGIN
    SELECT AVG(TOTAL_PRICE) INTO V_AVG_PRICE
    FROM REQUESTS JOIN COTTAGES USING (COTTAGE_ID)
    JOIN ADDRESSES USING (ADDRESS_ID)
    JOIN CITIES USING (CITY_ID)
    JOIN COUNTRIES USING (COUNTRY_ID)
    WHERE COUNTRY_ID = P_COUNTRY_ID;

    SELECT MIN(MIN_PRICE_PER_DAY) INTO V_MIN_PRICE
    FROM COTTAGES JOIN ADDRESSES USING (ADDRESS_ID)
    JOIN CITIES USING (CITY_ID)
    JOIN COUNTRIES USING (COUNTRY_ID)
    WHERE COUNTRY_ID = P_COUNTRY_ID;

    IF V_MIN_PRICE > 30 THEN
        V_TOTAL_PRICE := V_AVG_PRICE + V_MIN_PRICE;
    ELSIF  V_MIN_PRICE > 10 THEN
        V_TOTAL_PRICE := V_AVG_PRICE + V_MIN_PRICE * 2;
    ELSE
        V_TOTAL_PRICE := V_AVG_PRICE;
    END IF;
    RETURN V_TOTAL_PRICE;
END;
```

Pobiera średnią cenę wynajmu (V_AVG_PRICE) dla rezerwacji w podanym kraju na podstawie tabel: REQUESTS, COTTAGES, ADDRESSES, CITIES i COUNTRIES.

Pobiera minimalną cenę za dzień (V_MIN_PRICE) dla domków w tym kraju.

Na podstawie wartości minimalnej ceny stosuje różne reguły:

- **Jeśli minimalna cena > 30:** Dodaje średnią cenę do minimalnej ceny.
- **Jeśli minimalna cena > 10:** Dodaje średnią cenę do podwójnej minimalnej ceny.
- **W przeciwnym razie:** Zwraca tylko średnią cenę.

Zwraca obliczoną wartość jako średnią cenę wynajmu z uwzględnieniem podatków.

Przykład: istnieje kraj o country_id = 1, średnia cena wynajmu domów = 150, minimalna cena za dzień = 20
20 > 10, więc: 150 + 20 * 2 = 190

5. **Is_cottage_available(p_cottage number, p_checkin_date date, p_checkout_date number) –** zwraca 1, jeśli dany dom jest dostępny w danym przedziale czasu, w przeciwnym wypadku 0

```
CREATE OR REPLACE FUNCTION
    IS_COTTAGE_AVAILABLE( P_COTTAGE_ID NUMBER,
    P_CHECKIN_DATE DATE,
    P_CHECKOUT_DATE DATE
) RETURN NUMBER IS
    V_CONFLICTING_REQUESTS NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO V_CONFLICTING_REQUESTS
    FROM REQUESTS R
    JOIN REQUEST_APPROVALS RA ON R.REQUEST_ID =
    RA.REQUEST_ID WHERE R.COTTAGE_ID = P_COTTAGE_ID
      AND RA.IS_APPROVED = 1
      AND (
           (R.CHECKIN_DATE <= P_CHECKIN_DATE AND R.CHECKOUT_DATE >=
P_CHECKIN_DATE) OR
           (R.CHECKIN_DATE <= P_CHECKOUT_DATE AND R.CHECKOUT_DATE >=
P_CHECKOUT_DATE) OR
           (R.CHECKIN_DATE >= P_CHECKIN_DATE AND R.CHECKOUT_DATE <=
P_CHECKOUT_DATE)
      );

    IF V_CONFLICTING_REQUESTS = 0 THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;
```

Przykład: Jeśli istnieje dla domu o danym cottage_id request z request_approval=1 to jeśli zakresy (p_checkin_date, p_checkout_date) i zakres pobytu danego requestu jakkolwiek na siebie nachodzą, to funkcja zwraca 0.

Procedury

1. **change_review(p_review_id long, p_cottage_id long)** – Zmienia, którego domu dotyczy opinia o danym review_id.

```
CREATE OR REPLACE PROCEDURE CHANGE_REVIEW (P_REVIEW_ID LONG,
P_COTTAGE_ID LONG)
```

```
AS
    V_REVIEW NUMBER (4);
BEGIN
    UPDATE REVIEWS
    SET COTTAGE_ID = P_COTTAGE_ID
    WHERE REVIEW_ID = P_REVIEW_ID;
END;
```

2. **exchange_cottages(first_cottage_id long, second_cottage_id long)** - zamienia właścicieli dwóch domków oraz ustawia minimalną cenę za dzień dla obu domków na stałą wartość 39.99.

```
CREATE OR REPLACE PROCEDURE EXCHANGE_COTTAGES (FIRST_COTTAGE_ID LONG,
SECOND_COTTAGE_ID LONG)
AS
    V_BASE_PRICE DECIMAL(7, 2) := 39.99;
    V_FIRST_OWNER LONG;
    V_SECOND_OWNER LONG;
BEGIN
    SELECT OWNER_ID INTO V_FIRST_OWNER FROM COTTAGES
    WHERE COTTAGE_ID = FIRST_COTTAGE_ID;

    SELECT OWNER_ID INTO V_SECOND_OWNER FROM COTTAGES
    WHERE COTTAGE_ID = SECOND_COTTAGE_ID;

    UPDATE COTTAGES
    SET MIN_PRICE_PER_DAY = V_BASE_PRICE, OWNER_ID = V_SECOND_OWNER
    WHERE COTTAGE_ID = FIRST_COTTAGE_ID;

    UPDATE COTTAGES
    SET MIN_PRICE_PER_DAY = V_BASE_PRICE, OWNER_ID = V_FIRST_OWNER
    WHERE COTTAGE_ID = SECOND_COTTAGE_ID;
END;
```

3. **show_avg_price_per_user** – Przedstawia zestawienie użytkowników ze średnią ceną za dzień domów do nich należących (używa kursora).

```
CREATE OR REPLACE PROCEDURE SHOW_AVG_PRICE_PER_USER
IS
    CURSOR USER_CURSOR IS
        SELECT AVG(MIN_PRICE_PER_DAY) AS AVG_PRICE, OWNER_ID
        FROM COTTAGES
        GROUP BY OWNER_ID;

    USER_RECORD USER_CURSOR%ROWTYPE;

BEGIN
    -- OPEN USER_CURSOR;
```

```
    FOR USER_RECORD IN USER_CURSOR LOOP
        INSERT INTO AVG_PRICE_PER_USER (OWNER_ID, AVG_PRICE) VALUES
(USER_RECORD.OWNER_ID, USER_RECORD.AVG_PRICE);
        DBMS_OUTPUT.PUT_LINE('OWNER ID: ' || USER_RECORD.OWNER_ID ||
            ' AVG PRICE: ' || USER_RECORD.AVG_PRICE);
    END LOOP;
    -- CLOSE USER_CURSOR;
END SHOW_AVG_PRICE_PER_USER;
```

4. **show_avg_grade** – Przedstawia zestawienia domów ze średnią oceną im wystawioną (używa kursora).

```
CREATE OR REPLACE PROCEDURE SHOW_AVG_GRADE
IS
    CURSOR AVERAGE_GRADE_CR IS
        SELECT c.NAME, AVG(r.GRADE) AS AVG_GRADE
        FROM COTTAGES c
        JOIN REVIEWS r ON c.COTTAGE_ID = r.COTTAGE_ID
        GROUP BY c.NAME
        ORDER BY AVG_GRADE DESC;

    V_NAME VARCHAR2(100);
    V_AVG_GRADE NUMBER;
BEGIN
    OPEN AVERAGE_GRADE_CR;
    LOOP
        FETCH AVERAGE_GRADE_CR INTO V_NAME, V_AVG_GRADE;
        EXIT WHEN AVERAGE_GRADE_CR%NOTFOUND;
        INSERT INTO AVG_GRADE (COTTAGE_NAME, AVG_GRADE) VALUES (V_NAME,
V_AVG_GRADE);
        DBMS_OUTPUT.PUT_LINE('Cottage: ' || V_NAME || ' - Average Grade: '
|| V_AVG_GRADE);
    END LOOP;

    CLOSE AVERAGE_GRADE_CR;
END SHOW_AVG_GRADE;
```

5. **remove_old_requests(rejected_threshold number, inactive_threshold number)** – Usuwa requesty, które zostały odrzucone, gdzie odrzucenie jest starsze niż rejected_threshold dni oraz te requesty, które nie otrzymały odpowiedzi przez ostatnie inactive_threshold dni.

```
CREATE OR REPLACE PROCEDURE REMOVE_OLD_REQUESTS(REJECTED_THRESHOLD
NUMBER, INACTIVE_THRESHOLD NUMBER)
AS
BEGIN
    DELETE FROM REQUESTS
```

```
    WHERE REQUEST_ID IN (
        SELECT R.REQUEST_ID
        FROM REQUESTS R
        JOIN REQUEST_APPROVALS RA ON R.REQUEST_ID =
        RA.REQUEST_ID WHERE RA.IS_APPROVED = 0
          AND RA.DATE_CREATED < SYSDATE - REJECTED_THRESHOLD
    );

    DELETE FROM REQUESTS
    WHERE REQUEST_ID NOT IN (
        SELECT REQUEST_ID
        FROM REQUEST_APPROVALS
    )
      AND CHECKIN_DATE < SYSDATE - INACTIVE_THRESHOLD;

END;
```

Wyzwalacze

1. **tg_review_text** – uzupełnia treść recenzji przed jej wstawieniem do tabeli REVIEWS, jeśli użytkownik nie podał własnego tekstu. Bazuje na ocenie wystawionej dla domku.

```
CREATE OR REPLACE TRIGGER TG_REVIEW_TEXT
BEFORE INSERT on REVIEWS FOR EACH ROW
when (new.TEXT IS NULL)
DECLARE
    V_COTTAGE_NAME VARCHAR2(50);
BEGIN
    SELECT NAME INTO V_COTTAGE_NAME FROM COTTAGES
        WHERE COTTAGE_ID = :NEW.COTTAGE_ID;

    :NEW.TEXT :=
        CASE :NEW.GRADE
            WHEN  1
            THEN 'Very Bad Cottage.'
            WHEN 2
            THEN 'Bad Cottage.'
            WHEN 3
            THEN 'Decent Cottage.'
            WHEN 4
            THEN 'Good Cottage, I recommend.'
            WHEN 5
            THEN 'Great Cottage, I highly recommend.'
        END;

    dbms_output.put_line('new REVIEW added with grade ' || :NEW.GRADE ||
' to cottage ' || V_COTTAGE_NAME);
```

```
END;
```

2. **tg_calculate_total_price** – uzupełnia total_price w rezerwacji, jeśli nie została podana. Oblicza ją na podstawie ilości dni (checkout_date – checkin_date) oraz ceny za dzień danego domu.

```
CREATE OR REPLACE TRIGGER
TG_CALCULATE_TOTAL_PRICE BEFORE INSERT OR UPDATE
ON REQUESTS FOR EACH ROW
WHEN (new.TOTAL_PRICE is null)
DECLARE
    V_DAILY_PRICE DECIMAL(7,2);
    V_DAYS NUMBER;
BEGIN
    SELECT MIN_PRICE_PER_DAY INTO V_DAILY_PRICE
    FROM COTTAGES
    WHERE COTTAGE_ID = :NEW.COTTAGE_ID;

    V_DAYS := :NEW.CHECKOUT_DATE - :NEW.CHECKIN_DATE;
    :NEW.TOTAL_PRICE := V_DAILY_PRICE *
V_DAYS; END;
```

3. **tg_cottage_fields** – jeśli pola max_people_num, rooms_number, bathrooms_number nie zostały podane przez użytkownika, to ustawia je bazując na wartości size_m2

```
CREATE OR REPLACE TRIGGER TG_COTTAGE_FIELDS
BEFORE INSERT ON COTTAGES FOR EACH ROW
WHEN (NEW.SIZE_M2 IS NOT NULL )
BEGIN

    IF :NEW.MAX_PEOPLE_NUM IS NULL THEN
        :NEW.MAX_PEOPLE_NUM := FLOOR(:NEW.SIZE_M2/25);
    END IF;

    IF :NEW.ROOMS_NUMBER IS NULL THEN
        :NEW.ROOMS_NUMBER := FLOOR(:NEW.SIZE_M2/30);
    END IF;

    IF :NEW.BATHROOMS_NUMBER IS NULL THEN
        :NEW.BATHROOMS_NUMBER := FLOOR(:NEW.SIZE_M2/50);
    END IF;
END;
```

Kursory

1. **user_cursor** – oblicza średnią cenę za dzień dla każdego **owner_id** z tabeli **cottages**.

```
CURSOR USER_CURSOR IS
        SELECT AVG(MIN_PRICE_PER_DAY) AS AVG_PRICE, OWNER_ID
        FROM COTTAGES
        GROUP BY OWNER_ID;

USER_RECORD USER_CURSOR%ROWTYPE;
```

2. **average_grade_cr** – oblicza średnią cenę dla każdego domu.

```
CURSOR AVERAGE_GRADE_CR IS
        SELECT c.NAME, AVG(r.GRADE) AS AVG_GRADE
        FROM COTTAGES c
        JOIN REVIEWS r ON c.COTTAGE_ID = r.COTTAGE_ID
        GROUP BY c.NAME
        ORDER BY AVG_GRADE DESC;

V_NAME VARCHAR2(20);
V_AVG_GRADE NUMBER;
```

3. **city_req_cr** – oblicza totalną ilość requestów dla domów w danych miastach

```
CURSOR CITY_REQ_CR IS
        SELECT ci.name AS city_name,
               COUNT(r.request_id) AS total_requests
        FROM requests r
        JOIN cottages c ON r.cottage_id = c.cottage_id
        JOIN addresses a ON c.address_id = a.address_id
        JOIN cities ci ON a.city_id = ci.city_id
        GROUP BY ci.name;
```