Najlepsze Praktyki Formatowania PL/SQL

Najlepsze Praktyki Formatowania Kodu PL/SQL

Spis treści

Wprowadzenie

Dokument ten przedstawia standardy i najlepsze praktyki formatowania kodu PL/SQL, których celem jest zwiększenie czytelności, utrzymywalności i spójności kodu w projektach bazodanowych Oracle.

Konwencje nazewnictwa

1.1 Nazwy tabel i kolumn

Założenie: Używaj wielkich liter dla słów kluczowych SQL i małych liter dla nazw obiektów użytkownika.

Przykład:

```sql
-- ✓ POPRAWNIE
CREATE TABLE pracownicy (
  id_pracownika  NUMBER(10) NOT NULL,
  imie       VARCHAR2(50),
  nazwisko     VARCHAR2(50),
  data_zatrudnienia DATE,
  CONSTRAINT pk_pracownicy PRIMARY KEY (id_pracownika)
);
```

Wyjaśnienie: Stosowanie spójnej konwencji wielkości liter pozwala szybko rozróżnić słowa kluczowe SQL od nazw obiektów utworzonych przez użytkownika. Ułatwia to czytanie i rozumienie kodu.

1.2 Nazwy obiektów bazodanowych

Założenie: Stosuj standardową konwencję nazewniczą dla wszystkich obiektów bazodanowych.

1.2.1 Tabele i podstawowe ograniczenia

Typ obiektu	Konwencja nazewnicza	Przykład
Tabela	Nazwa w liczbie mnogiej (ostatni człon)	`policies`
		`policy_roles`
		`policy_role_types`
Primary Key	`<table>_pk`	`policies_pk`
Foreign Key	`<table>_<ref_table>_<optional_name>_fk`	`policies_customers_fk`
	⚠️ **Zawsze tworzyć indeks na kolumnie FK!**	`customers_customers_fk`
Unique Key	`<table>_<optional_name>_uk`	`policies_uk`
Check Constraint	`<table>_<optional_name>_chk`	`policies_status_chk`
		`policies_flag_chk`

1.2.2 Indeksy

Typ indeksu	Konwencja nazewnicza	Przykład
Indeks dla PK	`<pk_name>_idx`	`policies_pk_idx`
Indeks dla UK	`<uk_name>_idx`	`policies_uk_idx`
Indeks dla FK	`<fk_name>_idx`	`policies_customers_fk_idx`
Indeks pozostałe	`<table>_<optional_name>_idx`	`policies_customer_name_idx`

1.2.3 Triggery

Element	Opis	Przykład
Konwencja	`<table>_<ba_flag><iud> <rs_flag>_<hist>_trg`	
BA_flag	B (before) lub A (after)	`policies_biur_trg`
IUD	I (insert), U (update), D (delete)	`policies_aiudr_hist_trg`
RS_flag	R (row) lub S (statement)	
HIST	Opcjonalny element dla triggerów historycznych	

Przykłady triggerów: - `policies_biur_trg` - before insert/update row - `policies_aiudr_hist_trg` - after insert/update/delete row (historyzujący)

1.2.4 Sekwencje

Typ obiektu	Konwencja nazewnicza	Przykład
Sequence	`<optional_table>_<optional_column>_<optional_name>_seq`	`policies_id_seq`

1.2.5 Widoki i tabele specjalne

Typ obiektu	Konwencja nazewnicza	Przykład
Widoki	`<name>_v`	`active_policies_v`
Widoki zmaterializowane	`<name>_mv`	`policy_summary_mv`
External Table	`<name>_ext`	`customer_data_ext`
Global Temporary Table	`<name>_tmp`	`calculation_results_tmp`

1.2.6 Typy danych

Typ obiektu	Konwencja nazewnicza	Przykład
Typy	`<name>_t`	`employee_record_t`
Typy - table of	`<name>_tt`	`employee_list_tt`

1.2.7 Obiekty systemowe

Typ obiektu	Konwencja nazewnicza	Przykład
Directory	`<name>_dir`	`data_export_dir`

1.2.8 Obiekty PL/SQL

Typ obiektu	Konwencja nazewnicza	Przykład
Pakiet PL/SQL (INSIS events)	`<name>_srv`	`policy_handler_srv`
Procedura PL/SQL	`<name>_prc`	`calculate_premium_prc`
Funkcja PL/SQL	`<name>_fun`	`get_customer_rating_fun`

Przykład zastosowania:

```
 -- ✓ POPRAWNIE
CREATE TABLE policies (
  policy_id          NUMBER(10) NOT NULL,
  customer_id        NUMBER(10) NOT NULL,
  policy_number      VARCHAR2(50) NOT NULL,
  status             VARCHAR2(20) DEFAULT 'ACTIVE',
  created_date       DATE DEFAULT SYSDATE,

  CONSTRAINT policies_pk PRIMARY KEY (policy_id),
  CONSTRAINT policies_customers_fk FOREIGN KEY (customer_id)
    REFERENCES customers (customer_id),
  CONSTRAINT policies_status_chk CHECK (status IN ('ACTIVE', 'INACTIVE', 'SUSPENDED'))
);

-- Indeksy (OBOWIĄZKOWE dla FK!)
CREATE INDEX policies_pk_idx ON policies (policy_id);
CREATE INDEX policies_customers_fk_idx ON policies (customer_id);
CREATE INDEX policies_policy_number_idx ON policies (policy_number);

-- Sequence
CREATE SEQUENCE policies_id_seq
  START WITH 1
  INCREMENT BY 1
  NOCACHE;

-- Trigger historyzujący
CREATE OR REPLACE TRIGGER policies_aiudr_hist_trg
  AFTER INSERT OR UPDATE OR DELETE ON policies
  FOR EACH ROW
BEGIN
  -- Logika historyzacji
  NULL;
END;
/
```

1.3 Nazwy zmiennych PL/SQL

Założenie: Używaj standardowych prefiksów dla wszystkich zmiennych, parametrów i obiektów PL/SQL.

Prefiksy zmiennych PL/SQL

Zmienne podstawowe:

Typ zmiennej	Prefiks	Przykład
Zmienna globalna	`g_`	`g_session_user`
Zmienna lokalna	`l_`	`l_counter`
Stała	`c_`	`c_max_attempts`

Struktury danych:

Typ struktury	Prefiks	Przykład
Typ danych	`t_`	`t_employee_rec`
Kursor	`cur_`	`cur_employees`
Zmienna lokalna rekordowa	`lr_`	`lr_employee_data`
Zmienna globalna rekordowa	`gr_`	`gr_session_info`
Zmienna lokalna typu kolekcja	`lt_`	`lt_employee_list`
Zmienna globalna typu kolekcja	`gt_`	`gt_lookup_cache`

Wyjątki i parametry:

Typ elementu	Prefiks	Przykład
Wyjątek	`e_`	`e_invalid_data`
Parametr IN	`i_`	`i_employee_id`
Parametr OUT	`o_`	`o_result_count`
Parametr IN OUT	`io_`	`io_error_message`

Przykład:

```
  -- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE oblicz_wynagrodzenie (
  i_id_pracownika    IN     NUMBER,
  i_uwzglednij_bonus IN     BOOLEAN DEFAULT TRUE,
  o_wynagrodzenie    OUT    NUMBER,
  io_komunikat       IN OUT VARCHAR2
) IS
  -- Stałe
  c_bonus_multiplier  CONSTANT NUMBER := 1.2;
  c_max_salary       CONSTANT NUMBER := 50000;

  -- Zmienne lokalne
  l_podstawa         NUMBER;
  l_bonus           NUMBER := 0;

  -- Kursor
  cur_dane_pracownika CURSOR IS
    SELECT p.wynagrodzenie_podstawowe,
           p.premia_roczna
      FROM pracownicy p
     WHERE p.id_pracownika = i_id_pracownika;

  -- Zmienna rekordowa
  lr_pracownik       cur_dane_pracownika%ROWTYPE;

  -- Wyjątek własny
  e_nieprawidlowy_id EXCEPTION;

BEGIN
  OPEN cur_dane_pracownika;
  FETCH cur_dane_pracownika INTO lr_pracownik;
```

```
  IF cur_dane_pracownika%NOTFOUND THEN
    CLOSE cur_dane_pracownika;
    RAISE e_nieprawidlowy_id;
  END IF;
  CLOSE cur_dane_pracownika;

  l_podstawa := lr_pracownik.wynagrodzenie_podstawowe;

  IF i_uwzglednij_bonus THEN
    l_bonus := lr_pracownik.premia_roczna * c_bonus_multiplier;
  END IF;

  o_wynagrodzenie := l_podstawa + l_bonus;
  io_komunikat := io_komunikat || ' - Przeliczono z bonusem: ' || l_bonus;

EXCEPTION
  WHEN e_nieprawidlowy_id THEN
    RAISE_APPLICATION_ERROR(-20001, 'Nie znaleziono pracownika o ID: ' || i_id_pracownika);
END oblicz_wynagrodzenie;
/
```

Wyjaśnienie: Standardowe prefiksy pozwalają natychmiast zidentyfikować typ i zakres zmiennej. Szczególnie ważne jest rozróżnienie między zmiennymi lokalnymi (`l_`) a globalnymi (`g_`), oraz między różnymi typami kolekcji i rekordów. Stosowanie wyjątków z prefiksem `e_` ułatwia identyfikację własnych wyjątków biznesowych.

Wcięcia i formatowanie

2.1 Wcięcia

Założenie: Używaj 2 spacji dla każdego poziomu wcięcia. Nie używaj tabulatorów.

Przykład:

```
 -- ✓ POPRAWNIE
BEGIN
  IF l_wartosc > 0 THEN
    FOR i IN 1..10 LOOP
      IF i MOD 2 = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Parzysta: ' || i);
      ELSE
        DBMS_OUTPUT.PUT_LINE('Nieparzysta: ' || i);
      END IF;
    END LOOP;
  END IF;
END;
/
```

Wyjaśnienie: Konsekwentne wcięcia tworzą wizualną hierarchię struktury kodu, ułatwiając identyfikację bloków BEGIN-END, pętli i instrukcji warunkowych. 2 spacje oferują dobry kompromis między czytelnością a oszczędnością miejsca, jednocześnie zwiększając czytelność w długich blokach kodu z wieloma poziomami zagnieżdżenia.

2.2 Formatowanie bloków BEGIN-END

Założenie: Słowa kluczowe BEGIN i END umieszczaj na osobnych liniach, na tym samym poziomie wcięcia.

Przykład:

```plsql
 -- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE proces_zamowienia (
  i_id_zamowienia IN NUMBER
) IS
  l_status VARCHAR2(20);
BEGIN
  SELECT z.status
    INTO l_status
    FROM zamowienia z
   WHERE z.id_zamowienia = i_id_zamowienia;

  IF l_status = 'NOWE' THEN
    BEGIN
      -- Zagnieżdżony blok
      UPDATE zamowienia z
         SET z.status = 'W_REALIZACJI'
       WHERE z.id_zamowienia = i_id_zamowienia;

      COMMIT;
    EXCEPTION
      WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
    END;
  END IF;
END proces_zamowienia;
/
```

Wyjaśnienie: Umieszczanie BEGIN i END na osobnych liniach z odpowiednim wcięciem tworzy czytelną strukturę blokową. Każdy blok jest wyraźnie odgraniczony, co ułatwia debugowanie i zrozumienie logiki programu.

Instrukcje SQL

3.1 Formatowanie SELECT

Założenie: Umieszczaj każdą kolumnę w SELECT na osobnej linii. Klauzule (FROM, WHERE, ORDER BY) rozpoczynaj od nowej linii.

Przykład:

```plsql
 -- ✓ POPRAWNIE
SELECT
  p.id_pracownika,
  p.imie,
  p.nazwisko,
  p.data_zatrudnienia,
  d.nazwa AS nazwa_dzialu,
  d.lokalizacja,
  s.stanowisko,
  s.wynagrodzenie
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
  LEFT JOIN stanowiska s
    ON p.id_stanowiska = s.id_stanowiska
WHERE
  p.aktywny = 'T'
```

```
  AND p.data_zatrudnienia >= ADD_MONTHS(SYSDATE, -12)
  AND d.lokalizacja IN ('Warszawa', 'Kraków', 'Wrocław')
ORDER BY
  d.nazwa,
  p.nazwisko,
  p.imie;
```

Wyjaśnienie: Umieszczanie każdej kolumny na osobnej linii pozwala łatwo zobaczyć wszystkie wybierane pola. W JOIN-ach klauzula `ON` powinna być w nowej linii z wcięciem dla lepszej czytelności warunków łączenia. Formatowanie klauzul WHERE z warunkami na oddzielnych liniach ułatwia modyfikację zapytań i szybkie lokalizowanie błędów logicznych.

3.2 Formatowanie INSERT

Założenie: Wymień wszystkie kolumny jawnie, umieszczając je na osobnych liniach lub grupując logicznie.

Przykład:

```
 -- ✓ POPRAWNIE - dla małej liczby kolumn
INSERT INTO pracownicy (
  id_pracownika,
  imie,
  nazwisko,
  email,
  data_zatrudnienia
) VALUES (
  pracownicy_id_seq.NEXTVAL,
  'Jan',
  'Kowalski',
  'jan.kowalski@firma.pl',
  SYSDATE
);

-- ✓ POPRAWNIE - dla większej liczby kolumn
INSERT INTO pracownicy (
  id_pracownika,
  imie,
  nazwisko,
  email,
  telefon,
  adres,
  kod_pocztowy,
  miasto,
  data_zatrudnienia,
  id_dzialu,
  id_stanowiska
) VALUES (
  pracownicy_id_seq.NEXTVAL,
  'Jan',
  'Kowalski',
  'jan.kowalski@firma.pl',
  '123456789',
  'ul. Główna 1',
  '00-001',
  'Warszawa',
  SYSDATE,
  10,
  5
);
```

Wyjaśnienie: Jawne wymienianie kolumn zapobiega błędom przy zmianach struktury tabeli (dodanie/usunięcie kolumn). Grupowanie wartości w logiczne zestawy (dane osobowe, adresowe, służbowe) zwiększa czytelność dla tabel z wieloma kolumnami.

3.3 Formatowanie UPDATE

Założenie: Każde przypisanie SET na osobnej linii. Warunki WHERE z odpowiednim wcięciem.

Przykład:

```
 -- ✓ POPRAWNIE
UPDATE pracownicy p
   SET p.wynagrodzenie = p.wynagrodzenie * 1.1,
       p.data_ostatniej_podwyzki = SYSDATE,
       p.zaktualizowal = USER,
       p.data_aktualizacji = SYSTIMESTAMP
 WHERE p.id_dzialu = 10
   AND p.ocena_roczna >= 4
   AND p.data_ostatniej_podwyzki < ADD_MONTHS(SYSDATE, -12);
```

Wyjaśnienie: Rozdzielenie każdego przypisania na osobną linię pozwala łatwo śledzić, które pola są aktualizowane. Używanie aliasu tabeli (`p`) zwiększa czytelność i jest spójne z praktykami stosowanymi w SELECT i JOIN. Jest to szczególnie ważne podczas code review i debugowania, gdy trzeba szybko zidentyfikować, co dokładnie jest zmieniane.

Procedury i funkcje

4.1 Deklaracja parametrów

Założenie: Każdy parametr na osobnej linii z wyrównaniem typów danych i trybów (IN/OUT/IN OUT). Używaj prefiksów: i_ dla IN, o_ dla OUT, io_ dla IN OUT.

Przykład:

```
 -- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE przetworz_zamowienie (
  i_id_zamowienia    IN     NUMBER,
  i_id_klienta       IN     NUMBER,
  i_tryb_platnosci   IN     VARCHAR2,
  i_kod_rabatowy     IN     VARCHAR2 DEFAULT NULL,
  o_status_wynikowy  OUT    VARCHAR2,
  o_kwota_koncowa    OUT    NUMBER,
  io_komunikat_bledu IN OUT VARCHAR2
) IS
  l_rabat        NUMBER := 0;
  l_kwota_brutto NUMBER;
  l_kwota_netto  NUMBER;
BEGIN
  -- Implementacja
  NULL;
END przetworz_zamowienie;
/

**Wyjaśnienie:** Wyrównanie parametrów w kolumnach (nazwa, tryb, typ) tworzy czytelną "tabelę" parametrów.
        Prefiksy i_, o_, io_ natychmiast informują o kierunku przepływu danych. Łatwo zauważyć, które
        parametry są wejściowe, wyjściowe czy dwukierunkowe. Wyrównanie typów danych pomaga szybko
        zidentyfikować niezgodności typów.

### 4.2 Struktura procedury/funkcji

**Założenie:** Zachowaj spójną strukturę: nagłówek → deklaracje → ciało → obsługa wyjątków.

**Przykład:**
```

```sql
-- ✓ POPRAWNIE
CREATE OR REPLACE FUNCTION oblicz_podatek (
  i_kwota_brutto   IN NUMBER,
  i_stawka_vat     IN NUMBER DEFAULT 23
) RETURN NUMBER
IS
  -- Sekcja deklaracji stałych
  c_max_kwota    CONSTANT NUMBER := 1000000;
  c_min_stawka   CONSTANT NUMBER := 0;

  -- Sekcja deklaracji zmiennych
  l_kwota_podatku  NUMBER;
  l_kwota_netto    NUMBER;

  -- Sekcja deklaracji wyjątków własnych
  e_nieprawidlowa_kwota  EXCEPTION;
  e_nieprawidlowa_stawka EXCEPTION;

BEGIN
  -- Walidacja danych wejściowych
  IF i_kwota_brutto <= 0
    OR i_kwota_brutto > c_max_kwota
  THEN
    RAISE e_nieprawidlowa_kwota;
  END IF;

  IF i_stawka_vat < c_min_stawka
    OR i_stawka_vat > 100
  THEN
    RAISE e_nieprawidlowa_stawka;
  END IF;

  -- Logika biznesowa
  l_kwota_netto := i_kwota_brutto / (1 + i_stawka_vat / 100);
  l_kwota_podatku := i_kwota_brutto - l_kwota_netto;

  -- Logowanie
  INSERT INTO logi_podatkowe (
    data_obliczenia,
    kwota_brutto,
    stawka_vat,
    kwota_podatku
  ) VALUES (
    SYSTIMESTAMP,
    i_kwota_brutto,
    i_stawka_vat,
    l_kwota_podatku
  );

  RETURN ROUND(l_kwota_podatku, 2);

EXCEPTION
  WHEN e_nieprawidlowa_kwota THEN
    RAISE_APPLICATION_ERROR(-20001,
      'Kwota brutto musi być między 0 a ' || c_max_kwota);

  WHEN e_nieprawidlowa_stawka THEN
    RAISE_APPLICATION_ERROR(-20002,
      'Stawka VAT musi być między 0 a 100');

  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20999,
      'Błąd podczas obliczania podatku: ' || SQLERRM);
END oblicz_podatek;
/
```

Wyjaśnienie: Uporządkowana struktura z wyraźnie oddzielonymi sekcjami (stałe, zmienne, wyjątki, walidacja, logika, obsługa błędów) sprawia, że kod jest przewidywalny i łatwy w nawigacji. Programista wie, gdzie

szukać konkretnych elementów. Grupowanie deklaracji według typu ułatwia zarządzanie zakresem zmiennych.

Obsługa wyjątków

5.1 Struktura EXCEPTION

Założenie: Obsługuj wyjątki specyficzne przed ogólnym WHEN OTHERS. Zawsze loguj błędy.

Przykład:

```
 -- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE aktualizuj_zamowienie (
  i_id_zamowienia IN NUMBER,
  i_nowy_status   IN VARCHAR2
) IS
  l_aktualny_status VARCHAR2(20);

  e_nieprawidlowy_status EXCEPTION;
  e_zamowienie_nieznalezione EXCEPTION;

BEGIN
  -- Sprawdzenie istnienia zamówienia
  BEGIN
    SELECT z.status
      INTO l_aktualny_status
      FROM zamowienia z
     WHERE z.id_zamowienia = i_id_zamowienia;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE e_zamowienie_nieznalezione;
  END;

  -- Walidacja statusu
  IF i_nowy_status NOT IN ('NOWE', 'W_REALIZACJI', 'WYSLANE', 'ZAKONCZONE') THEN
    RAISE e_nieprawidlowy_status;
  END IF;

  -- Aktualizacja
  UPDATE zamowienia z
     SET z.status = i_nowy_status,
         z.data_modyfikacji = SYSTIMESTAMP,
         z.uzytkownik_modyfikujacy = USER
   WHERE z.id_zamowienia = i_id_zamowienia;

  COMMIT;

  -- Logowanie sukcesu
  INSERT INTO logi_zmian (
    tabela,
    id_rekordu,
    akcja,
    uzytkownik,
    data_operacji
  ) VALUES (
    'ZAMOWIENIA',
    i_id_zamowienia,
    'ZMIANA_STATUSU',
    USER,
    SYSTIMESTAMP
  );

EXCEPTION
  WHEN e_zamowienie_nieznalezione THEN
```

```
      ROLLBACK;
      -- Logowanie błędu
      INSERT INTO logi_bledow (
        procedura,
        komunikat,
        uzytkownik,
        data_bledu
      ) VALUES (
        'AKTUALIZUJ_ZAMOWIENIE',
        'Nie znaleziono zamówienia o ID: ' || i_id_zamowienia,
        USER,
        SYSTIMESTAMP
      );
      COMMIT;
      RAISE_APPLICATION_ERROR(-20001,
        'Zamówienie o podanym ID nie istnieje');

    WHEN e_nieprawidlowy_status THEN
      ROLLBACK;
      INSERT INTO logi_bledow (
        procedura,
        komunikat,
        uzytkownik,
        data_bledu
      ) VALUES (
        'AKTUALIZUJ_ZAMOWIENIE',
        'Nieprawidłowy status: ' || i_nowy_status,
        USER,
        SYSTIMESTAMP
      );
      COMMIT;
      RAISE_APPLICATION_ERROR(-20002,
        'Nieprawidłowy status zamówienia');

    WHEN OTHERS THEN
      ROLLBACK;
      INSERT INTO logi_bledow (
        procedura,
        komunikat,
        kod_bledu,
        uzytkownik,
        data_bledu
      ) VALUES (
        'AKTUALIZUJ_ZAMOWIENIE',
        SQLERRM,
        SQLCODE,
        USER,
        SYSTIMESTAMP
      );
      COMMIT;
      RAISE;

  END aktualizuj_zamowienie;
  /
```

Wyjaśnienie: Prawidłowa obsługa wyjątków jest kluczowa dla stabilności aplikacji. Obsługa specyficznych wyjątków przed WHEN OTHERS pozwala na precyzyjną reakcję na różne sytuacje błędne. Logowanie błędów jest niezbędne do debugowania i monitorowania. Nigdy nie używaj pustego WHEN OTHERS, który ukrywa błędy.

Komentarze i dokumentacja

6.1 Komentarze nagłówkowe

Założenie: Każda procedura/funkcja powinna mieć komentarz nagłówkowy opisujący cel, parametry, wartość zwracaną i autora.

Przykład:

```
 -- ✓ POPRAWNIE
/*******************************************************************************
 * Nazwa: oblicz_wynagrodzenie_roczne
 * Cel: Oblicza roczne wynagrodzenie pracownika z uwzględnieniem premii i bonusów
 *
 * Parametry:
 *   i_id_pracownika  - ID pracownika (klucz główny tabeli PRACOWNICY)
 *   i_rok      - Rok, dla którego obliczamy wynagrodzenie
 *   i_uwzglednij_bonus - Czy uwzględnić bonus roczny (domyślnie TAK)
 *
 * Zwraca:
 *   NUMBER - Całkowite wynagrodzenie roczne w PLN
 *
 * Wyjątki:
 *   -20001: Nie znaleziono pracownika o podanym ID
 *   -20002: Nieprawidłowy rok (mniejszy niż rok zatrudnienia)
 *
 * Historia zmian:
 *   2025-11-07  Jan Kowalski  Utworzenie
 *   2025-10-15  Anna Nowak  Dodanie obsługi bonusów kwartalnych
 *
 * Przykład użycia:
 *   DECLARE
 *   l_wynagrodzenie NUMBER;
 *   BEGIN
 *   l_wynagrodzenie := oblicz_wynagrodzenie_roczne(
 *     i_id_pracownika => 1001,
 *     i_rok => 2025,
 *     i_uwzglednij_bonus => TRUE
 *   );
 *   DBMS_OUTPUT.PUT_LINE('Wynagrodzenie: ' || l_wynagrodzenie);
 *   END;
 ******************************************************************************/
CREATE OR REPLACE FUNCTION oblicz_wynagrodzenie_roczne (
  i_id_pracownika  IN NUMBER,
  i_rok      IN NUMBER,
  i_uwzglednij_bonus   IN BOOLEAN DEFAULT TRUE
) RETURN NUMBER
IS
  l_wynagrodzenie_podstawowe NUMBER;
  l_premie         NUMBER := 0;
  l_bonusy         NUMBER := 0;
BEGIN
  -- Implementacja
  RETURN l_wynagrodzenie_podstawowe + l_premie + l_bonusy;
END oblicz_wynagrodzenie_roczne;
/
```

Wyjaśnienie: Szczegółowy komentarz nagłówkowy służy jako dokumentacja API dla innych programistów. Zawiera wszystkie informacje niezbędne do prawidłowego użycia procedury/funkcji bez konieczności czytania implementacji. Historia zmian pozwala śledzić ewolucję kodu.

6.2 Komentarze w kodzie

Założenie: Komentuj “dlaczego”, nie “co”. Używaj komentarzy do wyjaśnienia logiki biznesowej i nietypowych rozwiązań.

Przykład:

```sql
 -- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE przelicz_zamowienia IS
  l_kurs_waluty NUMBER;
BEGIN
  -- Pobieramy kurs z NBP dla daty bieżącej, ale jeśli nie ma kursu
  -- (np. weekend), cofamy się do ostatniego dnia roboczego
  BEGIN
    SELECT k.kurs
      INTO l_kurs_waluty
      FROM kursy_nbp k
     WHERE k.waluta = 'EUR'
       AND k.data_kursu = TRUNC(SYSDATE);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      -- Szukamy ostatniego dostępnego kursu (max 7 dni wstecz)
      SELECT k.kurs
        INTO l_kurs_waluty
        FROM kursy_nbp k
       WHERE k.waluta = 'EUR'
         AND k.data_kursu = (
        SELECT MAX(k2.data_kursu)
          FROM kursy_nbp k2
         WHERE k2.waluta = 'EUR'
           AND k2.data_kursu <= TRUNC(SYSDATE)
           AND data_kursu >= TRUNC(SYSDATE) - 7
      );
  END;

  -- UWAGA: Celowo nie używamy MERGE, ponieważ w naszej wersji Oracle (11g)
  -- występuje znany bug przy MERGE z funkcjami deterministycznymi (Bug #14719814)
  FOR rec IN (SELECT z.id_zamowienia,
                     z.kwota_eur
                FROM zamowienia z
               WHERE z.status = 'AKTYWNE') LOOP
    UPDATE zamowienia z
       SET z.kwota_pln = rec.kwota_eur * l_kurs_waluty
     WHERE z.id_zamowienia = rec.id_zamowienia;
  END LOOP;

  COMMIT;
END przelicz_zamowienia;
/
```

Wyjaśnienie: Dobre komentarze wyjaśniają intencje i kontekst biznesowy, nie oczywiste operacje. Komentarz "pobieramy kurs" jest zbędny, bo kod jest czytelny. Natomiast wyjaśnienie, dlaczego cofamy się do ostatniego dnia roboczego lub dlaczego nie używamy **MERGE**, dostarcza cennej informacji, której nie da się wywnioskować z samego kodu.

6.3 Nagłówki sekcji

Założenie: Używaj komentarzy do oznaczenia głównych sekcji w dłuższych procedurach.

Przykład:

```sql
-- ✓ POPRAWNIE
CREATE OR REPLACE PROCEDURE kompleksowe_przetwarzanie_zamowien IS
BEGIN
  /*============================================================================
   * SEKCJA 1: WALIDACJA I PRZYGOTOWANIE DANYCH
   *============================================================================*/
  -- Sprawdzanie integralności danych wejściowych
  NULL;

  /*============================================================================
   * SEKCJA 2: PRZETWARZANIE GŁÓWNE
   *============================================================================*/
  -- Logika biznesowa przetwarzania zamówień
  NULL;

  /*============================================================================
   * SEKCJA 3: AGREGACJA I RAPORTOWANIE
```

```
    *============================================================================*/
  -- Generowanie statystyk i raportów
  NULL;

  /*============================================================================
   * SEKCJA 4: FINALIZACJA I CZYSZCZENIE
   *============================================================================*/
  -- Operacje końcowe i czyszczenie danych tymczasowych
  NULL;

END kompleksowe_przetwarzanie_zamowien;
/
```

Wyjaśnienie: W długich procedurach (powyżej 200 linii) wizualne oddzielenie głównych sekcji znacząco ułatwia nawigację. Programista może szybko zlokalizować interesującą go część kodu. Nagłówki sekcji pełnią rolę "spisu treści" procedury.

Najlepsze praktyki

7.1 Używanie aliasów tabel

Założenie: Zawsze używaj krótkich, znaczących aliasów dla tabel, szczególnie w JOIN-ach.

Przykład:

```
 -- ✓ POPRAWNIE
SELECT
  p.id_pracownika,
  p.nazwisko,
  d.nazwa AS nazwa_dzialu,
  m.nazwisko AS nazwisko_menadzera,
  s.wynagrodzenie
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
  LEFT JOIN pracownicy m
    ON p.id_menadzera = m.id_pracownika
  INNER JOIN stanowiska s
    ON p.id_stanowiska = s.id_stanowiska
WHERE
  p.data_zatrudnienia >= ADD_MONTHS(SYSDATE, -12)
  AND d.aktywny = 'T';
```

Wyjaśnienie: Aliasy skracają zapytania i zwiększają czytelność, szczególnie przy wielu JOIN-ach. Używanie pełnych nazw tabel jest rozwlekłe i utrudnia szybkie skanowanie kodu. Dobrze dobrane aliasy (p dla pracownicy, d dla dzialy) są intuicyjne i łatwe do zapamiętania.

7.2 Formatowanie długich warunków

Założenie: Rozbijaj długie warunki logiczne na osobne linie z odpowiednim wcięciem, grupując operatory.

Przykład:

```
 -- ✓ POPRAWNIE
SELECT *
FROM zamowienia z
```

```
  WHERE
    -- Warunki czasowe
    (
      z.data_zamowienia >= TO_DATE('2025-01-01', 'YYYY-MM-DD')
      AND z.data_zamowienia < TO_DATE('2026-01-01', 'YYYY-MM-DD')
    )
    -- Warunki statusowe
    AND (
      z.status IN ('NOWE', 'W_REALIZACJI', 'WYSLANE')
      OR (z.status = 'ANULOWANE' AND z.przyczyna_anulowania = 'KLIENT')
    )
    -- Warunki finansowe
    AND (
      z.wartosc_brutto >= 1000
      OR (z.wartosc_brutto >= 500 AND z.klient_vip = 'T')
    )
    -- Warunki dodatkowe
    AND z.kraj IN ('PL', 'DE', 'CZ')
    AND z.waluta = 'PLN';
```

Wyjaśnienie: Grupowanie warunków według kategorii (czasowe, statusowe, finansowe) z komentarzami tworzy logiczną strukturę. Odpowiednie wcięcie pokazuje pierwszeństwo operatorów logicznych. To szczególnie ważne przy skomplikowanych warunkach biznesowych, gdzie błąd w nawiasach może prowadzić do nieprawidłowych wyników.

7.3 Używanie nazwanych parametrów

Założenie: Przy wywołaniach procedur/funkcji używaj nazwanych parametrów zamiast pozycyjnych.

Przykład:

```
 -- ✓ POPRAWNIE
DECLARE
  l_wynik VARCHAR2(100);
BEGIN
  przetworz_platnosc(
    i_id_zamowienia => 12345,
    i_kwota => 1500.00,
    i_waluta => 'PLN',
    i_metoda_platnosci => 'KARTA',
    o_status_wynikowy => l_wynik
  );

  DBMS_OUTPUT.PUT_LINE('Status: ' || l_wynik);
END;
/
```

Wyjaśnienie: Nazwane parametry zwiększają czytelność i bezpieczeństwo kodu. Są odporne na zmianę kolejności parametrów w definicji procedury. Przy wywołaniu natychmiast widać, co oznacza każda wartość, bez konieczności sprawdzania sygnatur procedur.

7.4 Obsługa wartości NULL

Założenie: Zawsze jawnie obsługuj potencjalne wartości NULL w warunkach i porównaniach.

Przykład:

```
 -- ✓ POPRAWNIE
CREATE OR REPLACE FUNCTION porownaj_wartosci (
  i_wartosc1 IN NUMBER,
```

```
  i_wartosc2 IN NUMBER
) RETURN VARCHAR2
IS
BEGIN
  IF i_wartosc1 IS NULL AND i_wartosc2 IS NULL THEN
    RETURN 'OBIE_NULL';
  ELSIF i_wartosc1 IS NULL THEN
    RETURN 'PIERWSZA_NULL';
  ELSIF i_wartosc2 IS NULL THEN
    RETURN 'DRUGA_NULL';
  ELSIF i_wartosc1 = i_wartosc2 THEN
    RETURN 'ROWNE';
  ELSIF i_wartosc1 > i_wartosc2 THEN
    RETURN 'PIERWSZA_WIEKSZA';
  ELSE
    RETURN 'DRUGA_WIEKSZA';
  END IF;
END porownaj_wartosci;
/

-- Przykład użycia NVL i COALESCE
SELECT p.id_pracownika,
       p.nazwisko,
       NVL(p.telefon_komorkowy, p.telefon_stacjonarny) AS telefon_kontaktowy,
       COALESCE(p.email_prywatny, p.email_firmowy, 'brak@email.pl') AS email,
       NVL(p.premia, 0) + NVL(p.bonus, 0) AS dodatki
  FROM pracownicy p;
```

Wyjaśnienie: NULL w SQL ma specjalną semantykę - nie jest równy niczemu, nawet innemu NULL. Brak obsługi NULL prowadzi do subtelnych błędów logicznych. Funkcje NVL i COALESCE pomagają zapewnić wartości domyślne, ale należy ich używać świadomie, rozumiejąc implikacje biznesowe zastępowania NULL wartością domyślną.

7.5 Używanie kursora FOR LOOP

Założenie: Preferuj niejawne kursory FOR LOOP zamiast jawnego otwierania i zamykania kursorów.

Przykład:

```
 -- ✓ POPRAWNIE - kursor niejawny
CREATE OR REPLACE PROCEDURE aktualizuj_premie IS
BEGIN
  FOR rec IN (
    SELECT p.id_pracownika,
           p.wynagrodzenie,
           p.ocena_roczna
      FROM pracownicy p
     WHERE p.data_zatrudnienia < ADD_MONTHS(SYSDATE, -12)
       AND p.aktywny = 'T'
  ) LOOP
    UPDATE pracownicy p
       SET p.premia = rec.wynagrodzenie * (rec.ocena_roczna / 100)
     WHERE p.id_pracownika = rec.id_pracownika;
  END LOOP;

  COMMIT;
END aktualizuj_premie;
/

-- ✓ AKCEPTOWALNE - kursor jawny z parametrem
CREATE OR REPLACE PROCEDURE generuj_raport (
  i_id_dzialu IN NUMBER
) IS
  CURSOR c_pracownicy (cp_id_dzialu NUMBER) IS
    SELECT p.id_pracownika,
```

```
            p.imie,
            p.nazwisko,
            p.wynagrodzenie
        FROM pracownicy p
       WHERE p.id_dzialu = cp_id_dzialu
       ORDER BY p.nazwisko;

  l_suma_wynagrodzen NUMBER := 0;
BEGIN
  FOR rec IN c_pracownicy(i_id_dzialu) LOOP
    DBMS_OUTPUT.PUT_LINE(rec.nazwisko || ': ' || rec.wynagrodzenie);
    l_suma_wynagrodzen := l_suma_wynagrodzen + rec.wynagrodzenie;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Suma: ' || l_suma_wynagrodzen);
END generuj_raport;
/
```

Wyjaśnienie: Pętla FOR automatycznie otwiera kursor, pobiera wiersze i zamyka kursor, nawet w przypadku wyjątku. Eliminuje to ryzyko pozostawienia otwartego kursora (wyciek zasobów). Kod jest krótszy i bardziej czytelny. Jawne kursory są przydatne tylko wtedy, gdy potrzebujesz większej kontroli (np. kursory z parametrami, sprawdzanie %NOTFOUND w specyficznych przypadkach).

7.6 Bulk operations (BULK COLLECT i FORALL)

Założenie: Dla operacji na dużych zbiorach danych używaj bulk operations zamiast row-by-row processing.

Przykład:

```
 -- ✓ POPRAWNIE - bulk operations dla wydajności
CREATE OR REPLACE PROCEDURE archiwizuj_stare_zamowienia IS
  TYPE t_id_table IS TABLE OF zamowienia.id_zamowienia%TYPE;
  TYPE t_data_table IS TABLE OF zamowienia.data_zamowienia%TYPE;

  l_ids t_id_table;
  l_daty t_data_table;

  c_limit CONSTANT PLS_INTEGER := 1000; -- Przetwarzamy partiami
BEGIN
  -- Pobieranie danych bulk
  SELECT z.id_zamowienia,
         z.data_zamowienia
    BULK COLLECT INTO l_ids, l_daty
    FROM zamowienia z
   WHERE z.data_zamowienia < ADD_MONTHS(SYSDATE, -24)
     AND z.status = 'ZAKONCZONE';

  -- Wstawianie do archiwum bulk
  FORALL i IN 1..l_ids.COUNT
    INSERT INTO zamowienia_archiwum (
      id_zamowienia,
      data_zamowienia,
      data_archiwizacji
    ) VALUES (
      l_ids(i),
      l_daty(i),
      SYSDATE
    );

  -- Usuwanie bulk
  FORALL i IN 1..l_ids.COUNT
    DELETE FROM zamowienia z
    WHERE z.id_zamowienia = l_ids(i);

  COMMIT;
```

```
    DBMS_OUTPUT.PUT_LINE('Zarchiwizowano ' || l_ids.COUNT || ' zamówień');

  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      RAISE;
  END archiwizuj_stare_zamowienia;
  /



   -- ✓ POPRAWNIE - przetwarzanie partiami dla bardzo dużych zbiorów
  CREATE OR REPLACE PROCEDURE aktualizuj_masowo IS
    TYPE t_id_table IS TABLE OF pracownicy.id_pracownika%TYPE;
    l_ids t_id_table;

    CURSOR c_pracownicy IS
      SELECT p.id_pracownika
        FROM pracownicy p
       WHERE p.data_ostatniej_podwyzki < ADD_MONTHS(SYSDATE, -12);

    c_limit CONSTANT PLS_INTEGER := 5000;
  BEGIN
    OPEN c_pracownicy;
    LOOP
      FETCH c_pracownicy BULK COLLECT INTO l_ids LIMIT c_limit;
      EXIT WHEN l_ids.COUNT = 0;

      FORALL i IN 1..l_ids.COUNT
        UPDATE pracownicy p
           SET p.wynagrodzenie = p.wynagrodzenie * 1.05,
               p.data_ostatniej_podwyzki = SYSDATE
         WHERE p.id_pracownika = l_ids(i);

      COMMIT; -- Commit po każdej partii

      DBMS_OUTPUT.PUT_LINE('Przetworzono partię: ' || l_ids.COUNT || ' rekordów');
    END LOOP;
    CLOSE c_pracownicy;

    DBMS_OUTPUT.PUT_LINE('Zakończono przetwarzanie masowe');
  END aktualizuj_masowo;
  /
```

Wyjaśnienie: Bulk operations drastycznie zwiększają wydajność dla operacji na wielu wierszach, redukując context switching między silnikiem PL/SQL a SQL. BULK COLLECT pobiera wszystkie wiersze jednocześnie, a FORALL wykonuje wszystkie DML w jednej operacji. Dla bardzo dużych zbiorów danych używamy LIMIT, aby uniknąć przepełnienia pamięci. Różnica w wydajności może być 10-50x dla tysięcy rekordów.

7.7 Transakcje i COMMIT

Założenie: Świadomie zarządzaj transakcjami. Unikaj auto-commit po każdej operacji.

Przykład:

```
   -- ✓ POPRAWNIE - transakcje w jednostkach biznesowych
  CREATE OR REPLACE PROCEDURE przetwórz_zamowienie (
    i_id_zamowienia IN NUMBER
  ) IS
    l_id_faktury NUMBER;
    l_id_wysylki NUMBER;
  BEGIN
    -- Jedna transakcja dla całej operacji biznesowej
    SAVEPOINT przed_przetworzeniem;
```

```
    -- Krok 1: Aktualizacja statusu zamówienia
    UPDATE zamowienia z
       SET z.status = 'W_REALIZACJI'
     WHERE z.id_zamowienia = i_id_zamowienia;

    -- Krok 2: Utworzenie faktury
    INSERT INTO faktury (
      id_zamowienia,
      data_wystawienia,
      kwota
    ) VALUES (
      i_id_zamowienia,
      SYSDATE,
      (SELECT z.suma_zamowienia
         FROM zamowienia z
        WHERE z.id_zamowienia = i_id_zamowienia)
    ) RETURNING id_faktury INTO l_id_faktury;

    -- Krok 3: Utworzenie zlecenia wysyłki
    INSERT INTO wysylki (
      id_zamowienia,
      id_faktury,
      data_utworzenia,
      status
    ) VALUES (
      i_id_zamowienia,
      l_id_faktury,
      SYSDATE,
      'OCZEKUJE'
    ) RETURNING id_wysylki INTO l_id_wysylki;

    -- Krok 4: Aktualizacja stanów magazynowych
    UPDATE stany_magazynowe sm
       SET ilosc = ilosc - pz.ilosc
      FROM pozycje_zamowienia pz
     WHERE sm.id_produktu = pz.id_produktu
       AND pz.id_zamowienia = i_id_zamowienia;

    -- Wszystko OK - zatwierdzamy całą transakcję biznesową
    COMMIT;

    -- Logowanie sukcesu (w osobnej transakcji autonomicznej - patrz 7.8)
    log_operacja('PRZETWORZENIE_ZAMOWIENIA', i_id_zamowienia, 'SUCCESS');

  EXCEPTION
    WHEN OTHERS THEN
      -- Coś poszło nie tak - wycofujemy całą operację
      ROLLBACK TO przed_przetworzeniem;

      -- Logowanie błędu
      log_operacja('PRZETWORZENIE_ZAMOWIENIA', i_id_zamowienia,
            'ERROR: ' || SQLERRM);

      RAISE;
  END przetworz_zamowienie;
  /
```

Wyjaśnienie: Transakcja powinna odpowiadać jednej kompletnej operacji biznesowej. Wszystkie powiązane zmiany w bazie powinny być zatwierdzone razem (atomowość). COMMIT po każdej operacji prowadzi do niespójności danych w przypadku błędu. Używanie SAVEPOINT pozwala na częściowe wycofanie bez utraty całej transakcji.

7.8 Autonomous transactions

Założenie: Używaj autonomous transactions dla operacji logowania, które powinny być zachowane niezależnie od głównej transakcji.

Przykład:

```
 -- ✓ POPRAWNIE - logowanie w transakcji autonomicznej
CREATE OR REPLACE PROCEDURE log_operacja (
  i_nazwa_operacji IN VARCHAR2,
  i_id_obiektu   IN NUMBER,
  i_status     IN VARCHAR2,
  i_komunikat  IN VARCHAR2 DEFAULT NULL
) IS
  PRAGMA AUTONOMOUS_TRANSACTION; -- Kluczowa dyrektywa
BEGIN
  INSERT INTO logi_aplikacji (
    id_logu,
    data_operacji,
    nazwa_operacji,
    id_obiektu,
    status,
    komunikat,
    uzytkownik,
    sesja
  ) VALUES (
    logi_id_seq.NEXTVAL,
    SYSTIMESTAMP,
    i_nazwa_operacji,
    i_id_obiektu,
    i_status,
    i_komunikat,
    USER,
    SYS_CONTEXT('USERENV', 'SESSIONID')
  );

  -- Transakcja autonomiczna MUSI mieć własny COMMIT lub ROLLBACK
  COMMIT;

EXCEPTION
  WHEN OTHERS THEN
    -- Nawet jeśli logowanie się nie powiedzie, nie przerywamy głównej operacji
    ROLLBACK;
    -- Możemy spróbować zapisać do pliku lub DBMS_OUTPUT
    DBMS_OUTPUT.PUT_LINE('Błąd logowania: ' || SQLERRM);
END log_operacja;
/

-- Przykład użycia
CREATE OR REPLACE PROCEDURE usun_pracownika (
  i_id_pracownika IN NUMBER
) IS
BEGIN
  -- Logujemy początek operacji (zostanie zachowane nawet przy ROLLBACK)
  log_operacja('USUWANIE_PRACOWNIKA', i_id_pracownika, 'START');

  -- Główna operacja
  DELETE FROM pracownicy p WHERE p.id_pracownika = i_id_pracownika;

  IF SQL%ROWCOUNT = 0 THEN
    log_operacja('USUWANIE_PRACOWNIKA', i_id_pracownika, 'BRAK_PRACOWNIKA');
    RAISE_APPLICATION_ERROR(-20001, 'Pracownik nie istnieje');
  END IF;

  -- Logujemy sukces
  log_operacja('USUWANIE_PRACOWNIKA', i_id_pracownika, 'SUCCESS');

  COMMIT;

EXCEPTION
  WHEN OTHERS THEN
```

```
        -- Logujemy błąd (log zostanie zachowany pomimo ROLLBACK)
        log_operacja('USUWANIE_PRACOWNIKA', i_id_pracownika, 'ERROR', SQLERRM);
        ROLLBACK;
        RAISE;
    END usun_pracownika;
    /
```

Wyjaśnienie: Autonomous transactions wykonują się w oddzielnej, niezależnej transakcji. Są idealne do logowania i audytu, ponieważ logi pozostają w bazie nawet gdy główna transakcja zostanie wycofana. To pozwala śledzić nieudane próby operacji, co jest kluczowe dla debugowania i bezpieczeństwa. UWAGA: Każda autonomous transaction MUSI kończyć się COMMIT lub ROLLBACK.

Konfiguracja automatycznego formatowania w Oracle SQL Developer

8.1 Wprowadzenie do formattera w SQL Developer

Oracle SQL Developer posiada wbudowany, potężny system formatowania kodu PL/SQL, który można dostosować do przedstawionych w tym dokumencie standardów. Właściwa konfiguracja formattera pozwala na:

- **Automatyczne formatowanie** całego kodu jednym skrótem klawiszowym (Ctrl+F7 / Cmd+F7)
- **Spójność kodu** w całym zespole dzięki eksportowi/importowi ustawień
- **Oszczędność czasu** - brak ręcznego wyrównywania i formatowania
- **Egzekwowanie standardów** podczas code review

8.2 Dostęp do ustawień formattera

Krok 1: Otwórz okno preferencji

1. Z menu wybierz: **Tools → Preferences** (lub **Oracle SQL Developer → Preferences** na macOS)
2. Alternatywnie użyj skrótu: **Ctrl+Shift+P** (Windows/Linux) lub **Cmd+,** (macOS)

Krok 2: Nawiguj do ustawień formattera

W lewym panelu preferencji rozwiń drzewo:

```
Database → SQL Formatter
```

Zobaczysz następujące sekcje: - **Oracle Formatting** - główne ustawienia formatowania - **Custom Format** - możliwość utworzenia własnych profili - **Third Party Formatters** - integracja z zewnętrznymi narzędziami

8.3 Konfiguracja podstawowych ustawień formatowania

8.3.1 Wcięcia (Indentation)

Lokalizacja: `Preferences → Database → SQL Formatter → Oracle Formatting`

Sekcja: Indentation

Ustawienie	Wartość	Opis
Spaces Per Indent	`2`	Liczba spacji na jeden poziom wcięcia
Use Tab Character	☐ `unchecked`	NIE używaj tabulatorów - tylko spacje
Indent SELECT items	☑ `checked`	Wcięcie dla kolumn w SELECT
Indent FROM items	☑ `checked`	Wcięcie dla tabel w FROM
Indent WHERE conditions	☑ `checked`	Wcięcie dla warunków WHERE

Przykład wyniku:

```sql
SELECT
  p.id_pracownika,
  p.nazwisko,
  d.nazwa
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
WHERE
  p.aktywny = 'T'
  AND d.lokalizacja = 'Warszawa';
```

8.3.2 Formatowanie klauzul SQL

Sekcja: Line Breaks

Ustawienie	Wartość	Opis
SELECT list items	`One item per line`	Każda kolumna w osobnej linii
FROM clause	`One item per line`	Każdy JOIN w osobnej linii
WHERE conditions	`One item per line`	Każdy warunek w osobnej linii
After SELECT	`New line`	SELECT w nowej linii
After FROM	`New line`	FROM w nowej linii
After WHERE	`New line`	WHERE w nowej linii
After AND/OR	`Keep with condition`	AND/OR przy warunku

Przykład wyniku:

```sql
SELECT
  z.id_zamowienia,
  z.data_zamowienia,
  k.nazwa AS klient,
  z.wartosc_brutto
FROM
  zamowienia z
  INNER JOIN klienci k
```

```
    ON z.id_klienta = k.id_klienta
WHERE
  z.status = 'AKTYWNE'
  AND z.data_zamowienia >= TRUNC(SYSDATE)
  AND k.vip = 'T';
```

8.3.3 Wielkość liter (Case)

Sekcja: Case

Element	Ustawienie	Przykład
SQL Keywords	`UPPER CASE`	`SELECT` , `FROM` , `WHERE`
PL/SQL Keywords	`UPPER CASE`	`BEGIN` , `END` , `IF` , `THEN`
Identifiers	`lower case`	`pracownicy` , `id_pracownika`
Data Types	`UPPER CASE`	`NUMBER` , `VARCHAR2` , `DATE`

Jak ustawić: 1. W sekcji **Case** znajdź dropdowny dla każdej kategorii 2. Wybierz odpowiednią opcję z listy: `UPPER CASE` , `lower case` , `Title Case` , lub `Unchanged`

Przykład wyniku:

```
 CREATE OR REPLACE PROCEDURE oblicz_wynagrodzenie (
  i_id_pracownika IN     NUMBER,
  o_wynagrodzenie OUT    NUMBER
) IS
  l_podstawa NUMBER;
  l_bonus    NUMBER;
BEGIN
  SELECT p.wynagrodzenie_podstawowe,
         p.premia
    INTO l_podstawa, l_bonus
    FROM pracownicy p
   WHERE p.id_pracownika = i_id_pracownika;

  o_wynagrodzenie := l_podstawa + l_bonus;
END oblicz_wynagrodzenie;
/
```

8.3.4 Formatowanie INSERT

Sekcja: INSERT Statement

Ustawienie	Wartość	Opis
Column list	`One item per line`	Każda kolumna w osobnej linii
Values list	`One item per line`	Każda wartość w osobnej linii
Align column and value lists	☑ `checked`	Wyrównanie kolumn i wartości

Przykład wyniku:

```
INSERT INTO pracownicy (
  id_pracownika,
  imie,
  nazwisko,
  email,
  data_zatrudnienia
) VALUES (
  pracownicy_id_seq.NEXTVAL,
  'Jan',
  'Kowalski',
  'jan.kowalski@firma.pl',
  SYSDATE
);
```

8.3.5 Formatowanie bloków PL/SQL

Sekcja: PL/SQL Blocks

Ustawienie	Wartość	Opis
BEGIN on new line	☑ **checked**	BEGIN w nowej linii
END on new line	☑ **checked**	END w nowej linii
Indent declarations	☑ **checked**	Wcięcie dla deklaracji zmiennych
Align variable declarations	☑ **checked**	Wyrównanie deklaracji
Align assignments	☑ **checked**	Wyrównanie przypisań

Przykład wyniku:

```
CREATE OR REPLACE PROCEDURE test_formatowania IS
  l_zmienna1  NUMBER := 100;
  l_zmienna2  VARCHAR2(50) := 'tekst';
  l_zmienna3  DATE := SYSDATE;
BEGIN
  IF l_zmienna1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Wartość: ' || l_zmienna1);
  END IF;
END test_formatowania;
/
```

8.4 Tworzenie własnego profilu formatowania

Aby zachować niestandardowe ustawienia i móc je udostępnić zespołowi:

Krok 1: Stwórz nowy profil

1. W oknie `Preferences → Database → SQL Formatter`
2. Kliknij przycisk **Create Profile** (ikona “+” lub “New”)
3. Nadaj nazwę profilowi, np. “Firma - PL/SQL Standard v2.3”
4. Kliknij **OK**

Krok 2: Dostosuj ustawienia

Skonfiguruj wszystkie opcje zgodnie z tabelami powyżej (punkty 8.3.1 - 8.3.5).

Krok 3: Testuj formatowanie

1. Otwórz przykładowy plik SQL z nieformatowanym kodem
2. Zaznacz cały kod (Ctrl+A / Cmd+A)
3. Naciśnij **Ctrl+F7** (Windows/Linux) lub **Cmd+F7** (macOS)
4. Sprawdź rezultat i dostosuj ustawienia w razie potrzeby

Krok 4: Eksportuj profil

1. W `Preferences → Database → SQL Formatter`
2. Wybierz swój profil z listy
3. Kliknij **Export** (ikona eksportu)
4. Zapisz plik XML, np. `sql_developer_formatter_standard.xml`
5. Udostępnij plik zespołowi (przez Git, Confluence, itp.)

Krok 5: Import profilu (dla innych członków zespołu)

1. Pobierz plik XML z profilem
2. W SQL Developer: `Preferences → Database → SQL Formatter`
3. Kliknij **Import** (ikona importu)
4. Wybierz pobrany plik XML
5. Profil zostanie dodany do listy dostępnych profili
6. Ustaw go jako domyślny: zaznacz i kliknij **Set as Default**

8.5 Konfiguracja zaawansowana

8.5.1 Białe znaki i puste linie

Sekcja: White Space

Ustawienie	Wartość	Opis
Blank lines before END	`0`	Brak pustych linii przed END
Blank lines after BEGIN	`0`	Brak pustych linii po BEGIN
Blank lines between sections	`1`	Jedna pusta linia między sekcjami
Remove trailing spaces	☑ `checked`	Usuń spacje na końcu linii

8.5.2 Długie linie

Sekcja: Line Wrapping

Ustawienie	Wartość	Opis
Maximum line width	`100`	Maksymalna długość linii (znaki)

Ustawienie	Wartość	Opis
Wrap long lines	☑ **checked**	Automatyczne łamanie długich linii
Wrap SELECT list	☑ **checked**	Łam długą listę SELECT
Wrap WHERE conditions	☑ **checked**	Łam długie warunki WHERE

Przykład wyniku dla długiego SELECT:

```
SELECT
  p.id_pracownika,
  p.imie,
  p.nazwisko,
  p.email,
  p.telefon,
  d.nazwa AS nazwa_dzialu,
  d.lokalizacja,
  s.stanowisko,
  s.wynagrodzenie,
  s.zakres_odpowiedzialnosci
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
  LEFT JOIN stanowiska s
    ON p.id_stanowiska = s.id_stanowiska;
```

8.5.3 Komentarze

Sekcja: Comments

Ustawienie	Wartość	Opis
Preserve formatting in comments	☑ **checked**	Zachowaj formatowanie w komentarzach
Align single-line comments	☐ **unchecked**	Nie wyrównuj komentarzy jednoliniowych
Keep comments on same line as code	☑ **checked**	Zachowaj komentarze przy kodzie

Przykład:

```
BEGIN
  -- To jest komentarz przed blokiem IF
  IF l_wartosc > 0 THEN
    l_wynik := l_wartosc * 2; -- Komentarz przy kodzie
  END IF;

  /*
   * To jest komentarz
   * wieloliniowy
   */
  RETURN l_wynik;
END;
```

8.6 Skróty klawiszowe i automatyzacja

8.6.1 Podstawowe skróty

Akcja	Windows/Linux	macOS	Opis
Format	`Ctrl+F7`	`Cmd+F7`	Formatuj zaznaczony kod lub cały plik
Format All	`Ctrl+Shift+F7`	`Cmd+Shift+F7`	Formatuj wszystkie otwarte pliki
Preferences	`Ctrl+Shift+P`	`Cmd+,`	Otwórz okno preferencji

8.6.2 Formatowanie przed zapisem

Aby automatycznie formatować kod przy każdym zapisie:

1. `Preferences → Database → Worksheet`
2. Znajdź opcję **Format on Save**
3. Zaznacz `☑ Format SQL and PL/SQL on save`

UWAGA: Ta opcja może spowalniać pracę przy bardzo dużych plikach!

8.6.3 Formatowanie wielu plików

Aby sformatować wiele plików jednocześnie:

1. W panelu **Files** lub **Connections** zaznacz wiele plików SQL
2. Kliknij prawym przyciskiem myszy
3. Wybierz **Format** z menu kontekstowego
4. Wszystkie pliki zostaną sformatowane zgodnie z aktywnym profilem

8.7 Najczęstsze problemy i rozwiązania

Problem 1: Formatter nie działa

Objawy: Skrót Ctrl+F7 nie formatuje kodu

Rozwiązania: 1. Sprawdź, czy kod jest zaznaczony (jeśli nie - zaznacz wszystko: Ctrl+A) 2. Upewnij się, że edytor rozpoznaje plik jako SQL/PLSQL (sprawdź w dolnym pasku) 3. Sprawdź, czy formatter jest włączony: `Preferences → Database → SQL Formatter → Enable SQL Formatter`

Problem 2: Nieprawidłowe wcięcia

Objawy: Wcięcia są zbyt duże lub wykorzystują tabulatory

Rozwiązanie: 1. `Preferences → Database → SQL Formatter → Oracle Formatting` 2. Sekcja **Indentation**: - Ustaw **Spaces Per Indent** na `2` - Odznacz **Use Tab Character** 3. Zapisz i przeformatuj kod ponownie (Ctrl+F7)

Problem 3: Formatter zmienia wielkość liter w niechciany sposób

Objawy: Nazwy tabel/kolumn są zamieniane na wielkie litery

Rozwiązanie: 1. `Preferences → Database → SQL Formatter → Oracle Formatting` 2. Sekcja **Case** → **Identifiers** 3. Ustaw na `lower case` lub `Unchanged` 4. Przeformatuj kod ponownie

Problem 4: Długie linie nie są łamane

Objawy: Zapytania SQL wychodzą poza ekran

Rozwiązanie: 1. `Preferences → Database → SQL Formatter → Oracle Formatting` 2. Sekcja **Line Wrapping**: - Zaznacz **Wrap long lines** - Ustaw **Maximum line width** na `100` - Zaznacz **Wrap SELECT list** i **Wrap WHERE conditions**

Problem 5: Komentarze są przesuwane

Objawy: Komentarze lądują w nieoczekiwanych miejscach

Rozwiązanie: 1. `Preferences → Database → SQL Formatter → Oracle Formatting` 2. Sekcja **Comments**: - Zaznacz **Preserve formatting in comments** - Zaznacz **Keep comments on same line as code**

8.8 Przykładowy plik konfiguracyjny XML

Poniżej znajduje się przykładowy fragment pliku XML z profilem formatowania zgodnym z tym dokumentem. Możesz go zapisać i zaimportować do SQL Developer:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<sqlDeveloperSettings>
  <profile name="PL/SQL Best Practices v2.3">
    <indentation>
      <spacesPerIndent>2</spacesPerIndent>
      <useTabCharacter>false</useTabCharacter>
      <indentSelectItems>true</indentSelectItems>
      <indentFromItems>true</indentFromItems>
      <indentWhereConditions>true</indentWhereConditions>
    </indentation>

    <lineBreaks>
      <selectListItems>onePerLine</selectListItems>
      <fromClause>onePerLine</fromClause>
      <whereConditions>onePerLine</whereConditions>
      <afterSelect>newLine</afterSelect>
      <afterFrom>newLine</afterFrom>
      <afterWhere>newLine</afterWhere>
    </lineBreaks>

    <case>
      <sqlKeywords>upperCase</sqlKeywords>
      <plsqlKeywords>upperCase</plsqlKeywords>
      <identifiers>lowerCase</identifiers>
      <dataTypes>upperCase</dataTypes>
    </case>

    <insertStatement>
      <columnList>onePerLine</columnList>
      <valuesList>onePerLine</valuesList>
      <alignLists>true</alignLists>
    </insertStatement>

    <plsqlBlocks>
      <beginOnNewLine>true</beginOnNewLine>
      <endOnNewLine>true</endOnNewLine>
      <indentDeclarations>true</indentDeclarations>
```

```
      <alignDeclarations>true</alignDeclarations>
      <alignAssignments>true</alignAssignments>
    </plsqlBlocks>

    <whiteSpace>
      <blankLinesBeforeEnd>0</blankLinesBeforeEnd>
      <blankLinesAfterBegin>0</blankLinesAfterBegin>
      <blankLinesBetweenSections>1</blankLinesBetweenSections>
      <removeTrailingSpaces>true</removeTrailingSpaces>
    </whiteSpace>

    <lineWrapping>
      <maximumLineWidth>100</maximumLineWidth>
      <wrapLongLines>true</wrapLongLines>
      <wrapSelectList>true</wrapSelectList>
      <wrapWhereConditions>true</wrapWhereConditions>
    </lineWrapping>

    <comments>
      <preserveFormattingInComments>true</preserveFormattingInComments>
      <alignSingleLineComments>false</alignSingleLineComments>
      <keepCommentsOnSameLineAsCode>true</keepCommentsOnSameLineAsCode>
    </comments>
  </profile>
</sqlDeveloperSettings>
```

Jak użyć tego pliku:

1. Skopiuj powyższą zawartość do pliku tekstowego
2. Zapisz jako `plsql_formatter_config.xml`
3. W SQL Developer: `Preferences → Database → SQL Formatter`
4. Kliknij **Import**
5. Wybierz zapisany plik XML
6. Profil "PL/SQL Best Practices v2.3" zostanie dodany do listy
7. Ustaw go jako domyślny

8.9 Integracja z kontrolą wersji (Git)

Aby zapewnić spójne formatowanie w całym zespole:

Krok 1: Dodaj plik konfiguracyjny do repozytorium

```
# Utwórz folder dla konfiguracji
mkdir -p .sqldeveloper

# Skopiuj wyeksportowany profil
cp ~/path/to/plsql_formatter_config.xml .sqldeveloper/

# Dodaj do Git
git add .sqldeveloper/plsql_formatter_config.xml
git commit -m "Add SQL Developer formatter configuration"
git push
```

Krok 2: Dodaj instrukcje do README projektu

```
## Konfiguracja SQL Developer

1. Pobierz najnowszą wersję z repozytorium
2. Otwórz SQL Developer
3. Przejdź do Tools → Preferences → Database → SQL Formatter
4. Kliknij Import i wybierz plik `.sqldeveloper/plsql_formatter_config.xml`
```

```
5. Ustaw profil "PL/SQL Best Practices v2.3" jako domyślny
6. Przed każdym commit'em formatuj kod: Ctrl+F7
```

Krok 3: Git pre-commit hook (opcjonalnie)

Możesz utworzyć Git hook, który sprawdza formatowanie przed commit'em:

```
#!/bin/bash
# .git/hooks/pre-commit

# Sprawdź, czy są pliki SQL do sformatowania
sql_files=$(git diff --cached --name-only --diff-filter=ACM | grep -E '\.(sql|prc|fnc|pks|pkb)$')

if [ -n "$sql_files" ]; then
  echo "🔍 Sprawdzanie formatowania plików SQL..."

  # Tutaj możesz dodać skrypt sprawdzający formatowanie
  # Np. wywołanie zewnętrznego narzędzia formatującego

  echo "✅ Formatowanie OK"
fi

exit 0
```

8.10 Przykłady przed i po formatowaniu

Przykład 1: Zapytanie SELECT

Przed formatowaniem:

```
select p.id_pracownika,p.imie,p.nazwisko,d.nazwa as nazwa_dzialu from pracownicy p inner join dzialy d on
        p.id_dzialu=d.id_dzialu where p.aktywny='T' and d.lokalizacja in ('Warszawa','Kraków') order by
        p.nazwisko;
```

Po formatowaniu (Ctrl+F7):

```
SELECT
  p.id_pracownika,
  p.imie,
  p.nazwisko,
  d.nazwa AS nazwa_dzialu
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
WHERE
  p.aktywny = 'T'
  AND d.lokalizacja IN ('Warszawa', 'Kraków')
ORDER BY
  p.nazwisko;
```

Przykład 2: Procedura PL/SQL

Przed formatowaniem:

```
create or replace procedure oblicz_wynagrodzenie(i_id_pracownika in number,o_wynagrodzenie out number)is
        l_podstawa number;l_bonus number;begin select p.wynagrodzenie_podstawowe,p.premia into
        l_podstawa,l_bonus from pracownicy p where
        p.id_pracownika=i_id_pracownika;o_wynagrodzenie:=l_podstawa+l_bonus;exception when no_data_found then
        raise_application_error(-20001,'Pracownik nie istnieje');end oblicz_wynagrodzenie;
```

Po formatowaniu (Ctrl+F7):

```
CREATE OR REPLACE PROCEDURE oblicz_wynagrodzenie (
  i_id_pracownika IN     NUMBER,
  o_wynagrodzenie OUT    NUMBER
) IS
  l_podstawa  NUMBER;
  l_bonus     NUMBER;
BEGIN
  SELECT p.wynagrodzenie_podstawowe,
         p.premia
    INTO l_podstawa, l_bonus
    FROM pracownicy p
   WHERE p.id_pracownika = i_id_pracownika;

  o_wynagrodzenie := l_podstawa + l_bonus;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'Pracownik nie istnieje');
END oblicz_wynagrodzenie;
/
```

8.11 Dodatkowe wskazówki

1. **Formatuj regularnie** - nie czekaj do końca pracy nad plikiem. Formatuj po każdej większej zmianie.
2. **Code review** - podczas review sprawdź, czy kod został sformatowany. Nieformatowany kod może być odrzucony.
3. **Przed commit'em** - zawsze sformatuj (Ctrl+F7) przed dodaniem do Git. To zapobiega konfliktom merge związanym z formatowaniem.
4. **Dokumentuj odstępstwa** - jeśli musisz świadomie odstąpić od standardu formatowania (np. w tabelach ASCII art w komentarzach), dodaj komentarz wyjaśniający.
5. **Aktualizuj profil** - gdy standardy się zmienią, wyeksportuj nową wersję profilu i zaktualizuj w repozytorium.
6. **Szkolenia** - upewnij się, że nowi członkowie zespołu wiedzą, jak skonfigurować formatter.

Konfiguracja automatycznego formatowania w PL/SQL Developer

9.1 Wprowadzenie do Beautifier w PL/SQL Developer

PL/SQL Developer od firmy Allround Automations posiada narzędzie **Beautifier**, które pozwala na zaawansowane formatowanie kodu PL/SQL. Jest to komercyjne narzędzie, ale oferuje bardzo precyzyjną kontrolę nad formatowaniem.

Główne zalety: - **Szczegółowa konfiguracja** - ponad 100 opcji formatowania - **Profile formatowania** - możliwość tworzenia wielu profili dla różnych projektów - **Integracja z IDE** - formatowanie bezpośrednio w edytorze - **Eksport/Import** - łatwe udostępnianie ustawień w zespole - **Wsparcie dla starszych wersji Oracle** - działa nawet z Oracle 8i

9.2 Dostęp do ustawień Beautifier

Krok 1: Otwórz okno Preferences

1. Z menu wybierz: **Tools → Preferences** lub naciśnij **Configure → Preferences**
2. Alternatywnie użyj skrótu: **Ctrl+Shift+C** (Configure button)

Krok 2: Nawiguj do ustawień Beautifier

W lewym panelu preferencji wybierz:

```
User Interface → Editor → Beautifier
```

Zobaczysz zakładki: - **General** - ogólne ustawienia - **Indentation** - wcięcia - **Capitalization** - wielkość liter - **Line Breaks** - łamanie linii - **White Space** - białe znaki - **Comments** - komentarze

9.3 Konfiguracja podstawowych ustawień formatowania

9.3.1 Ustawienia ogólne (General)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → General`

Ustawienie	Wartość	Opis
Use beautifier on file open	☐ `unchecked`	Nie formatuj automatycznie przy otwarciu
Use beautifier after auto-replace	☑ `checked`	Formatuj po auto-replace
Use beautifier on save	☐ `unchecked`	Nie formatuj automatycznie przy zapisie (opcjonalne)
Beautify selection only	☑ `checked`	Formatuj tylko zaznaczony fragment

9.3.2 Wcięcia (Indentation)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Indentation`

Ustawienie	Wartość	Opis
Indent	`2`	Liczba spacji na wcięcie
Use tabs	☐ `unchecked`	NIE używaj tabulatorów
Indent SELECT items	☑ `checked`	Wcięcie dla kolumn SELECT

Ustawienie	Wartość	Opis
Indent FROM items	☑ checked	Wcięcie dla tabel FROM
Indent WHERE conditions	☑ checked	Wcięcie dla warunków WHERE
Indent JOIN conditions	☑ checked	Wcięcie dla JOIN
Indent WHEN conditions	☑ checked	Wcięcie dla CASE WHEN
Indent declarations	☑ checked	Wcięcie dla deklaracji zmiennych

Przykład wyniku:

```
 SELECT
  p.id_pracownika,
  p.imie,
  p.nazwisko,
  d.nazwa AS nazwa_dzialu
FROM
  pracownicy p
  INNER JOIN dzialy d
    ON p.id_dzialu = d.id_dzialu
WHERE
  p.aktywny = 'T'
  AND d.lokalizacja = 'Warszawa';
```

9.3.3 Wielkość liter (Capitalization)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Capitalization`

Element	Ustawienie	Przykład
SQL Keywords	`UPPERCASE`	`SELECT` , `FROM` , `WHERE`
PL/SQL Keywords	`UPPERCASE`	`BEGIN` , `END` , `IF` , `THEN`
Data Types	`UPPERCASE`	`NUMBER` , `VARCHAR2` , `DATE`
Identifiers	`lowercase` lub `Do not change`	`pracownicy` , `id_pracownika`
Functions	`Do not change`	Zachowaj oryginalną wielkość

Jak ustawić: W każdej sekcji wybierz odpowiednią opcję z dropdownu: - `UPPERCASE` - `lowercase` - `Capitalize` - `Do not change`

Przykład wyniku:

```
 CREATE OR REPLACE PROCEDURE oblicz_wynagrodzenie (
  i_id_pracownika IN     NUMBER,
  o_wynagrodzenie OUT    NUMBER
) IS
  l_podstawa  NUMBER;
  l_bonus     NUMBER;
BEGIN
  SELECT p.wynagrodzenie_podstawowe,
         p.premia
```

```
      INTO l_podstawa, l_bonus
      FROM pracownicy p
     WHERE p.id_pracownika = i_id_pracownika;

    o_wynagrodzenie := l_podstawa + l_bonus;
END oblicz_wynagrodzenie;
/
```

9.3.4 Łamanie linii (Line Breaks)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Line Breaks`

Ustawienie	Wartość	Opis
SELECT list	`Line break after each item`	Każda kolumna w nowej linii
FROM clause	`Line break after each item`	Każdy JOIN w nowej linii
WHERE clause	`Line break after each condition`	Każdy warunek w nowej linii
AND/OR	`Before`	AND/OR przed warunkiem
Comma in lists	`After`	Przecinek po elemencie
Maximum line length	`100`	Maksymalna długość linii

Przykład wyniku:

```
SELECT
  z.id_zamowienia,
  z.data_zamowienia,
  k.nazwa AS klient,
  z.wartosc_brutto
FROM
  zamowienia z
  INNER JOIN klienci k
    ON z.id_klienta = k.id_klienta
WHERE
  z.status = 'AKTYWNE'
  AND z.data_zamowienia >= TRUNC(SYSDATE)
  AND k.vip = 'T';
```

9.3.5 Białe znaki (White Space)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → White Space`

Ustawienie	Wartość	Opis
Before operators	`1`	Jedna spacja przed operatorem
After operators	`1`	Jedna spacja po operatorze
Before comma	`0`	Brak spacji przed przecinkiem
After comma	`1`	Jedna spacja po przecinku

Ustawienie	Wartość	Opis
Around parentheses	`0`	Brak spacji wewnątrz nawiasów
Remove trailing spaces	☑ `checked`	Usuń spacje na końcu linii
Remove empty lines	☐ `unchecked`	Zachowaj puste linie

9.3.6 Komentarze (Comments)

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Comments`

Ustawienie	Wartość	Opis
Align single line comments	☐ `unchecked`	Nie wyrównuj komentarzy jednoliniowych
Keep comments on same line	☑ `checked`	Zachowaj komentarze przy kodzie
Indent multi-line comments	☑ `checked`	Wcięcie dla komentarzy wieloliniowych

9.4 Tworzenie własnego profilu formatowania

PL/SQL Developer pozwala na tworzenie wielu profili formatowania.

Krok 1: Stwórz nowy profil

1. W oknie `Preferences → User Interface → Editor → Beautifier`
2. U góry okna znajdź dropdown **Profile**
3. Kliknij przycisk **New** (lub ikonę "+")
4. Nadaj nazwę profilowi, np. "Firma PL/SQL Standard v3.0"
5. Kliknij **OK**

Krok 2: Dostosuj ustawienia

Skonfiguruj wszystkie zakładki (General, Indentation, Capitalization, Line Breaks, White Space, Comments) zgodnie z punktami 9.3.1 - 9.3.6.

Krok 3: Testuj formatowanie

1. Otwórz przykładowy plik SQL/PL/SQL
2. Zaznacz fragment kodu lub cały kod (Ctrl+A)
3. Naciśnij **F8** (lub wybierz **Edit → Beautify**)
4. Sprawdź rezultat i dostosuj ustawienia w razie potrzeby

Krok 4: Ustaw jako domyślny

1. W dropdownie **Profile** wybierz swój profil
2. Zaznacz **Set as default**
3. Kliknij **Apply** i **OK**

Krok 5: Eksportuj profil

1. W oknie Beautifier Preferences kliknij przycisk **Export**
2. Zapisz plik, np. `plsql_beautifier_config.ini`
3. Udostępnij plik zespołowi (przez Git, Confluence, itp.)

Krok 6: Import profilu (dla innych członków zespołu)

1. Pobierz plik `.ini` z profilem
2. W PL/SQL Developer: `Preferences → User Interface → Editor → Beautifier`
3. Kliknij przycisk **Import**
4. Wybierz pobrany plik `.ini`
5. Profil zostanie dodany do listy
6. Wybierz profil z dropdown i kliknij **Set as default**

9.5 Konfiguracja zaawansowana

9.5.1 Formatowanie bloków PL/SQL

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Line Breaks`

Sekcja: PL/SQL Blocks

Ustawienie	Wartość	Opis
BEGIN on new line	☑ **checked**	BEGIN w nowej linii
END on new line	☑ **checked**	END w nowej linii
THEN on new line	☐ **unchecked**	THEN w tej samej linii co IF
LOOP on new line	☑ **checked**	LOOP w nowej linii
DECLARE on new line	☑ **checked**	DECLARE w nowej linii

Przykład:

```
 CREATE OR REPLACE PROCEDURE test_proc IS
  l_counter  NUMBER := 0;
BEGIN
  IF l_counter = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Zero');
  END IF;

  FOR i IN 1..10 LOOP
    l_counter := l_counter + i;
  END LOOP;
END test_proc;
/
```

9.5.2 Formatowanie INSERT/UPDATE

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Line Breaks`

Ustawienie	Wartość	Opis
INSERT column list	`Line break after each item`	Każda kolumna w nowej linii
INSERT values list	`Line break after each item`	Każda wartość w nowej linii
UPDATE SET clause	`Line break after each item`	Każde przypisanie w nowej linii
Align column/value lists	☑ `checked`	Wyrównaj kolumny i wartości

Przykład:

```
 INSERT INTO pracownicy (
  id_pracownika,
  imie,
  nazwisko,
  email,
  data_zatrudnienia
) VALUES (
  pracownicy_id_seq.NEXTVAL,
  'Jan',
  'Kowalski',
  'jan.kowalski@firma.pl',
  SYSDATE
);

UPDATE pracownicy p
   SET p.wynagrodzenie = p.wynagrodzenie * 1.1,
       p.data_ostatniej_podwyzki = SYSDATE,
       p.zaktualizowal = USER
 WHERE p.id_dzialu = 10
   AND p.ocena_roczna >= 4;
```

9.5.3 Formatowanie CASE

Lokalizacja: `Preferences → User Interface → Editor → Beautifier → Indentation`

Ustawienie	Wartość	Opis
Indent WHEN	☑ `checked`	Wcięcie dla WHEN
Indent THEN	☑ `checked`	Wcięcie dla THEN (jeśli w nowej linii)
Align WHEN conditions	☑ `checked`	Wyrównaj warunki WHEN

Przykład:

```
 SELECT
  p.id_pracownika,
  p.nazwisko,
  CASE
    WHEN p.wynagrodzenie < 3000 THEN 'Niskie'
    WHEN p.wynagrodzenie BETWEEN 3000 AND 6000 THEN 'Średnie'
    WHEN p.wynagrodzenie > 6000 THEN 'Wysokie'
    ELSE 'Nieznane'
  END AS poziom_wynagrodzenia
FROM pracownicy p;
```

9.6 Skróty klawiszowe i automatyzacja

9.6.1 Podstawowe skróty

Akcja	Skrót	Opis
Beautify	`F8`	Formatuj zaznaczony kod lub cały plik
Beautify Selection	`Shift+F8`	Formatuj tylko zaznaczenie
Preferences	`Ctrl+Shift+C`	Otwórz okno konfiguracji

Zmiana skrótu: 1. `Tools → Preferences → User Interface → Key Configuration` 2. Znajdź akcję "Beautifier" 3. Ustaw własny skrót klawiszowy 4. Kliknij **OK**

9.6.2 Formatowanie przy zapisie

Aby automatycznie formatować kod przy każdym zapisie:

1. `Preferences → User Interface → Editor → Beautifier → General`
2. Zaznacz `☑ Use beautifier on save`

UWAGA: Ta opcja może spowalniać pracę przy bardzo dużych plikach!

9.6.3 Formatowanie wielu plików

PL/SQL Developer nie ma wbudowanej funkcji formatowania wielu plików jednocześnie, ale można to zrobić poprzez:

Opcja 1: Makro

```
 -- Utwórz makro w Tools → Macros
-- Makro: FormatAllFiles
BEGIN
  -- Otwórz plik
  -- F8 (Beautify)
  -- Zapisz plik
  -- Zamknij plik
  -- Następny plik
END;
```

Opcja 2: Command Window 1. `File → New → Command Window` 2. Wpisz polecenia formatujące 3. Wykonaj wsadowo

9.7 Najczęstsze problemy i rozwiązania

Problem 1: Beautifier nie działa

Objawy: Skrót F8 nie formatuje kodu

Rozwiązania: 1. Sprawdź, czy kod jest poprawny składniowo (Beautifier wymaga poprawnego SQL/PL/SQL) 2. Upewnij się, że edytor rozpoznaje typ pliku (sprawdź w `Window → Window List`) 3. Sprawdź, czy Beautifier jest włączony w Preferences 4. Zrestartuj PL/SQL Developer

Problem 2: Nieprawidłowe wcięcia

Objawy: Wcięcia są zbyt duże lub wykorzystują tabulatory

Rozwiązanie: 1. `Preferences → User Interface → Editor → Beautifier → Indentation` 2. Ustaw **Indent** na `2` 3. Odznacz **Use tabs** 4. Kliknij **Apply** i przeformatuj kod (F8)

Problem 3: Beautifier zmienia wielkość liter w niechciany sposób

Objawy: Nazwy tabel/kolumn są zamieniane na wielkie litery

Rozwiązanie: 1. `Preferences → User Interface → Editor → Beautifier → Capitalization` 2. Sekcja **Identifiers** → ustaw na `lowercase` lub `Do not change` 3. Kliknij **Apply** i przeformatuj kod (F8)

Problem 4: Komentarze są przesuwane

Objawy: Komentarze lądują w nieoczekiwanych miejscach

Rozwiązanie: 1. `Preferences → User Interface → Editor → Beautifier → Comments` 2. Zaznacz **Keep comments on same line** 3. Odznacz **Align single line comments** 4. Kliknij **Apply**

Problem 5: Beautifier formatuje tylko część kodu

Objawy: Formatuje się tylko zaznaczony fragment

Rozwiązanie: 1. `Preferences → User Interface → Editor → Beautifier → General` 2. Odznacz **Beautify selection only** (jeśli chcesz formatować cały plik) 3. Lub użyj **Ctrl+A** przed naciśnięciem **F8** (zaznacz wszystko)

9.8 Przykładowy plik konfiguracyjny INI

Poniżej znajduje się przykładowy fragment pliku INI z profilem formatowania zgodnym z tym dokumentem. Możesz go zapisać i zaimportować do PL/SQL Developer:

```ini
 [Beautifier Profile: PL/SQL Best Practices v3.0]
Version=15.0

[General]
UseOnFileOpen=0
UseAfterAutoReplace=1
UseOnSave=0
BeautifySelectionOnly=1

[Indentation]
Indent=2
UseTabs=0
IndentSelectItems=1
IndentFromItems=1
IndentWhereConditions=1
IndentJoinConditions=1
IndentWhenConditions=1
IndentDeclarations=1

[Capitalization]
SQLKeywords=UPPERCASE
PLSQLKeywords=UPPERCASE
DataTypes=UPPERCASE
Identifiers=lowercase
Functions=DoNotChange
```

```
[LineBreaks]
SelectList=LineBreakAfterEachItem
FromClause=LineBreakAfterEachItem
WhereClause=LineBreakAfterEachCondition
AndOr=Before
CommaInLists=After
MaxLineLength=100
BeginOnNewLine=1
EndOnNewLine=1
ThenOnNewLine=0
LoopOnNewLine=1
DeclareOnNewLine=1

[WhiteSpace]
BeforeOperators=1
AfterOperators=1
BeforeComma=0
AfterComma=1
AroundParentheses=0
RemoveTrailingSpaces=1
RemoveEmptyLines=0

[Comments]
AlignSingleLineComments=0
KeepCommentsOnSameLine=1
IndentMultiLineComments=1
```

Jak użyć tego pliku:

1. Skopiuj powyższą zawartość do pliku tekstowego
2. Zapisz jako `plsql_beautifier_config.ini`
3. W PL/SQL Developer: `Preferences → User Interface → Editor → Beautifier`
4. Kliknij przycisk **Import**
5. Wybierz zapisany plik `.ini`
6. Profil "PL/SQL Best Practices v3.0" zostanie dodany do listy
7. Wybierz go z dropdown i kliknij **Set as default**

9.9 Integracja z kontrolą wersji (Git)

Aby zapewnić spójne formatowanie w całym zespole:

Krok 1: Dodaj plik konfiguracyjny do repozytorium

```
# Utwórz folder dla konfiguracji
mkdir -p .plsqldev

# Skopiuj wyeksportowany profil
cp "C:\Users\Username\AppData\Roaming\Allround Automations\PL SQL Developer 15\Beautifier.ini"
        .plsqldev/beautifier_config.ini

# Dodaj do Git
git add .plsqldev/beautifier_config.ini
git commit -m "Add PL/SQL Developer beautifier configuration"
git push
```

Krok 2: Dodaj instrukcje do README projektu

```
## Konfiguracja PL/SQL Developer

1. Pobierz najnowszą wersję z repozytorium
```

2. Otwórz PL/SQL Developer
3. Przejdź do Tools → Preferences → User Interface → Editor → Beautifier
4. Kliknij Import i wybierz plik `.plsqldev/beautifier_config.ini`
5. Wybierz profil "PL/SQL Best Practices v3.0" i ustaw jako domyślny
6. Przed każdym commit'em formatuj kod: F8

Krok 3: Pre-commit hook (opcjonalnie)

```
#!/bin/bash
# .git/hooks/pre-commit

# Sprawdź, czy są pliki SQL do sformatowania
sql_files=$(git diff --cached --name-only --diff-filter=ACM | grep -E '\.(sql|prc|fnc|pks|pkb)$')

if [ -n "$sql_files" ]; then
  echo "🔍 Przypomnienie: Upewnij się, że sformatowałeś pliki SQL (F8 w PL/SQL Developer)"
  echo "Pliki do sprawdzenia:"
  echo "$sql_files"

  # Opcjonalnie: można dodać automatyczne sprawdzanie
  read -p "Czy sformatowałeś wszystkie pliki? (y/n) " -n 1 -r
  echo
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo "❌ Commit anulowany. Sformatuj pliki i spróbuj ponownie."
    exit 1
  fi
fi

echo "✅ Sprawdzenie formatowania OK"
exit 0
```

9.10 Dodatkowe wskazówki

1. **Testuj na kopii** - przed zastosowaniem nowego profilu na projekcie produkcyjnym, przetestuj go na kopii plików.
2. **Wersjonuj profile** - trzymaj historię profili formatowania w Git, z nazwami typu `beautifier_v1.0.ini`, `beautifier_v2.0.ini`.
3. **Dokumentuj odstępstwa** - jeśli świadomie odstępujesz od standardu (np. dla wygenerowanego kodu), dodaj komentarz:

```
-- BEAUTIFIER OFF
-- Ten kod został wygenerowany automatycznie
SELECT a,b,c FROM table;
-- BEAUTIFIER ON
```

4. **Code Review** - skonfiguruj narzędzie do code review tak, aby sprawdzało formatowanie (np. GitLab CI/CD).
5. **Backup ustawień** - regularnie eksportuj swoje profile do pliku backup.
6. **Synchronizuj z zespołem** - organizuj warsztaty, gdzie cały zespół razem konfiguruje narzędzie.
7. **Użyj Template** - PL/SQL Developer pozwala na tworzenie szablonów kodu (Templates) już sformatowanych według standardów:

```
 -- Template: proc
CREATE OR REPLACE PROCEDURE |name| IS
BEGIN
  |cursor|
END |name|;
/
```

Podsumowanie

Przestrzeganie przedstawionych praktyk formatowania kodu PL/SQL przynosi następujące korzyści:

1. **Czytelność** - Kod jest łatwiejszy do zrozumienia dla całego zespołu
2. **Utrzymywalność** - Modyfikacje i rozszerzenia są szybsze i bezpieczniejsze
3. **Debugowanie** - Błędy są łatwiejsze do zlokalizowania i naprawienia
4. **Współpraca** - Spójny styl ułatwia code review i pracę zespołową
5. **Dokumentacja** - Dobrze sformatowany kod z komentarzami jest samodokumentujący się
6. **Wydajność** - Stosowanie bulk operations i właściwego zarządzania transakcjami poprawia performance

Zalecane narzędzia

Dla automatyzacji formatowania kodu PL/SQL zaleca się użycie:

- **Oracle SQL Developer** - wbudowany formatter (Ctrl+F7)
- **PL/SQL Developer** - narzędzie Beautifier
- **SQLcl** - narzędzie command-line z formatowaniem
- **PLDoc** - generator dokumentacji z kodu PL/SQL

Dodatkowe zasoby

- Oracle Database PL/SQL Language Reference
- Oracle Database Development Guide
- “Oracle PL/SQL Best Practices” - Steven Feuerstein
- Oracle Live SQL - przykłady i tutoriale

Autor: Piotr Tomkiewicz
Data utworzenia: 6 października 2025
Wersja dokumentu: 4.0