Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Bazy Danych 1

Sprawozdanie z projektu

Michał Pióro, Radosław Ślepowroński

Warszawa, 2024

Spis treści

1. Wstęp

1.1. Cel projektu

Celem projektu jest zaprojektowanie i implementacja bazy danych wspierającej zarządzanie produkcją w zakładzie przemysłowym. Baza danych ma umożliwiać przechowywanie informacji o elementach infrastruktury produkcyjnej, takich jak linie produkcyjne, maszyny, historia użytkowania oraz usługi serwisowe. Dodatkowo system ma pozwalać na zarządzanie personelem, w tym ich stanowiskami i kompetencjami, a także na śledzenie produkcji i powiązanych produktów.

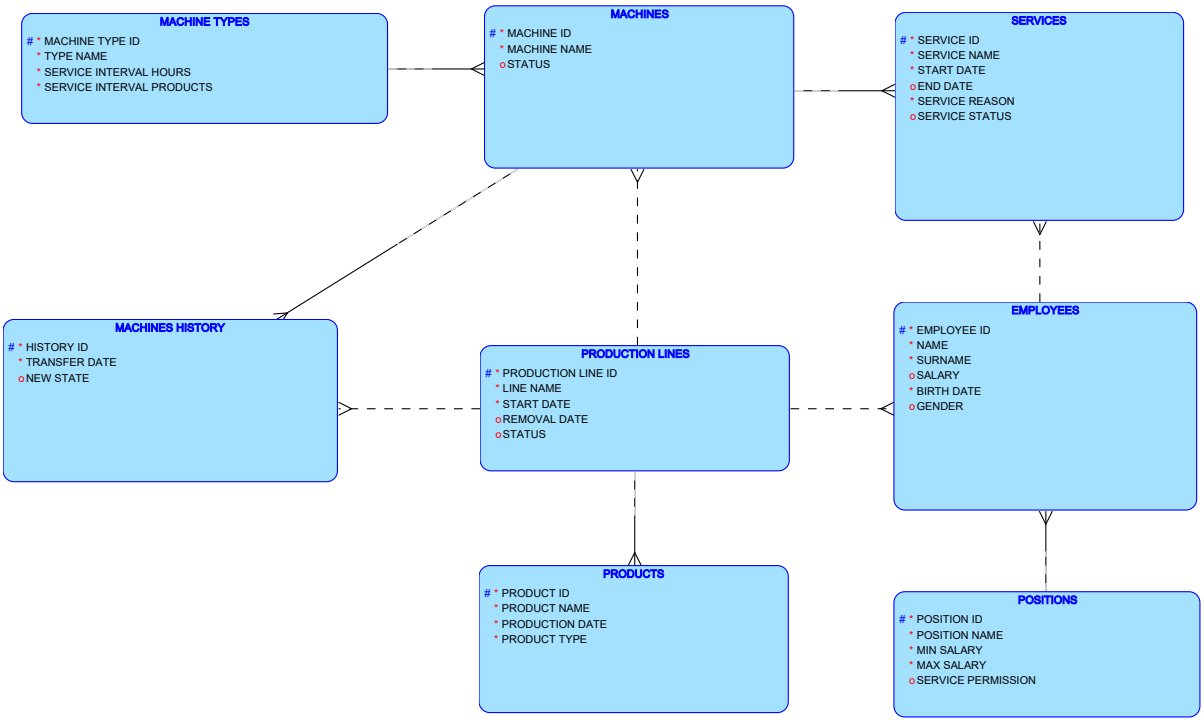
1.2. Opis rozwiązania

Projektowana baza danych składa się z ośmiu tabel, odzwierciedlających różne aspekty funkcjonowania zakładu produkcyjnego. Struktura bazy uwzględnia relacje między tabelami, takie jak powiązania maszyn z liniami produkcyjnymi, pracowników z liniami produkcyjnymi i usługami serwisowymi oraz produkty z liniami, na których zostały wytworzone. Tabele zostały zaprojektowane w sposób umożliwiający przechowywanie i przetwarzanie danych, z uwzględnieniem ograniczeń, takich jak klucze obce, zakresy wartości czy warunki integralności.

Rozwiązanie obejmuje również implementację wyzwalaczy, procedur i funkcji bazodanowych, wspierających automatyzację codziennych operacji, np. monitorowanie serwisów maszyn czy przesunięcia sprzętu między liniami produkcyjnymi.

2. Modelowanie danych

2.1. Model ER (związki między encjami)



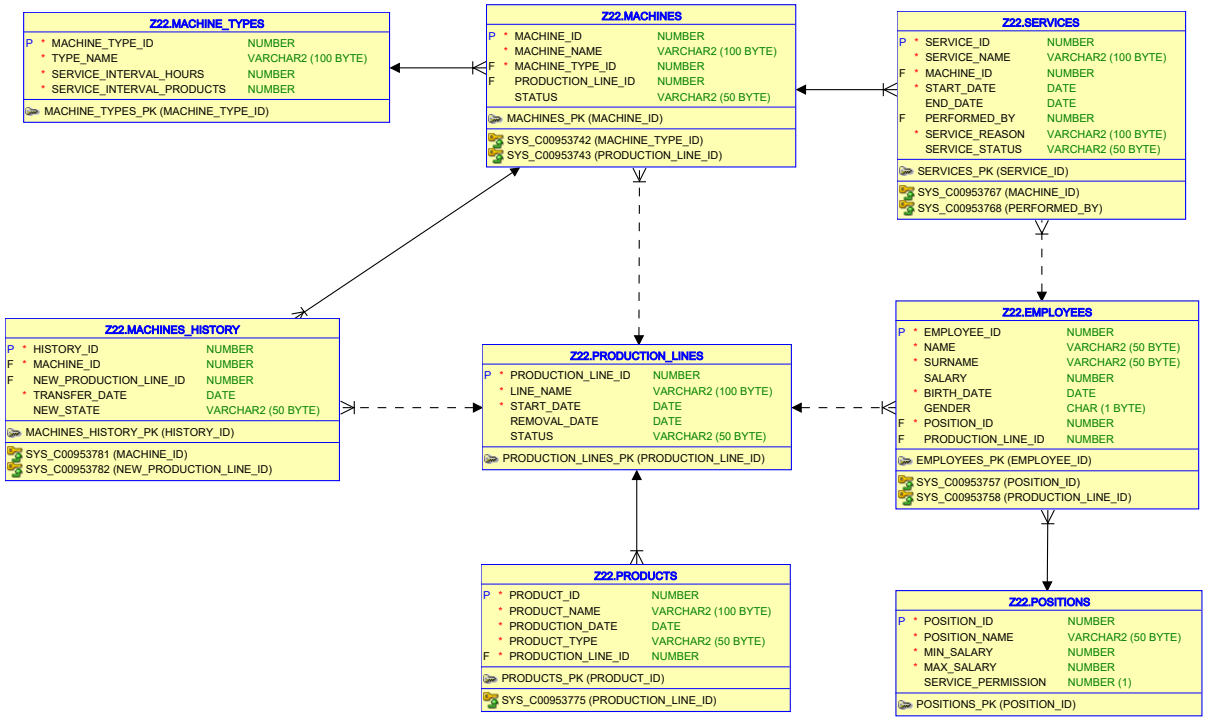
Rys. 2.1. Model ER (związki między encjami)

2.2. Opis związków

— Może istnieć wiele maszyn tego samego typu i każda maszyna musi mieć swój jeden typ.
— Może istnieć wiele wpisów odnośnie jednej maszyny i każdy wpis musi mieć określoną maszynę.
— Maszyna może być przypisana do jednej linii produkcyjnej do której może być przypisane wiele maszyn.
— Może istnieć wiele serwisów dotyczących danej maszyny, a do każdego serwisu musi być przypisana jedna maszyna.
— Jeden wpis może dotyczyć jednej linii produkcyjnej, a do jednej linii produkcyjnej może być przypisanych wiele wpisów w historii maszyn.
— Może zostać stworzonych wiele produktów na jednej linii produkcyjnej, a dany produkt musi zostać stworzony na jednej konkretnej linii.
— Do linii produkcyjnej może zostać przypisanych wielu pracowników, a jeden pracownik może być przypisany do jednej linii.

— Do pracownika może być przypisanych wiele serwisów, a jeden serwis może zostać przypisany do jednego pracownika.
— Do pracownika musi być przypisane jego stanowisko, a do stanowiska może być przypisanych wielu pracowników.

2.3. Model relacyjny



Rys. 2.2. Model relacyjny

2.4. Opis tabel i ich relacji

— **SERVICES**
  — *Opis*: Reprezentuje usługi wykonywane na maszynach. Przechowuje szczegóły takie jak identyfikator usługi, nazwę, maszynę, na której usługa została wykonana, powód i status usługi.
  — *Relacje*:
    — Odnosi się do tabeli `MACHINES` poprzez `MACHINE_ID`.
    — Odnosi się do tabeli `EMPLOYEES` przez `PERFORMED_BY`.
— **MACHINE_TYPES**
  — *Opis*: Definiuje typy maszyn, ich nazwę oraz interwały serwisowe (czasowe lub produktowe).
  — *Relacje*: Relacja do `MACHINES` poprzez `MACHINE_TYPE_ID`.
— **EMPLOYEES**
  — *Opis*: Przechowuje dane o pracownikach, takie jak imię, nazwisko, data urodzenia, płeć, stanowisko i linia produkcyjna, do której są przypisani.
  — *Relacje*:
    — Odnosi się do tabeli `POSITIONS` poprzez `POSITION_ID`.

  - Odnosi się do tabeli PRODUCTION_LINES przez PRODUCTION_LINE_ID.
- **POSITIONS**
  - *Opis*: Definiuje stanowiska pracy, ich nazwy, minimalne i maksymalne wynagrodzenie oraz uprawnienia serwisowe.
  - *Relacje*: Relacja do EMPLOYEES poprzez POSITION_ID.
- **MACHINES**
  - *Opis*: Przechowuje dane o maszynach, takie jak nazwa, typ, linia produkcyjna i status.
  - *Relacje*:
    - Odnosi się do tabeli MACHINE_TYPES przez MACHINE_TYPE_ID.
    - Odnosi się do PRODUCTION_LINES przez PRODUCTION_LINE_ID.
- **PRODUCTION_LINES**
  - *Opis*: Definiuje linie produkcyjne, ich nazwę, datę rozpoczęcia, datę zakończenia oraz status.
  - *Relacje*:
    - Relacja do MACHINES i EMPLOYEES przez PRODUCTION_LINE_ID.
- **MACHINES_HISTORY**
  - *Opis*: Śledzi historię maszyn, w tym zmiany linii produkcyjnych i daty transferu.
  - *Relacje*:
    - Odnosi się do tabeli MACHINES przez MACHINE_ID.
    - Odnosi się do tabeli PRODUCTION_LINES przez NEW_PRODUCTION_LINE_ID.
- **PRODUCTS**
  - *Opis*: Przechowuje dane o produktach, takie jak identyfikator, nazwa, data produkcji, typ produktu i linia produkcyjna.
  - *Relacje*:
    - Relacja do PRODUCTION_LINES przez PRODUCTION_LINE_ID.

2.5. Skrypty DDL do stworzenia schematu bazy danych

```
-- Tabela: Machine_Type
CREATE TABLE Machine_Types (
    machine_type_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    type_name VARCHAR2(100) NOT NULL,
    service_interval_hours NUMBER NOT NULL,
    service_interval_products NUMBER NOT NULL
);

-- Tabela: Production_Line
CREATE TABLE Production_Lines (
    production_line_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    line_name VARCHAR2(100) NOT NULL,
    start_date DATE NOT NULL,
    removal_date DATE DEFAULT null,
    status VARCHAR2(50) DEFAULT 'UNKNOWN'
);

-- Tabela: Machines
CREATE TABLE Machines (
    machine_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    machine_name VARCHAR2(100) NOT NULL,
    machine_type_id NUMBER NOT NULL,
    production_line_id NUMBER NOT NULL,
    status VARCHAR2(50) DEFAULT 'UNKNOWN',
    FOREIGN KEY (machine_type_id) REFERENCES Machine_Types(machine_type_id),
    FOREIGN KEY (production_line_id) REFERENCES
```

```
    Production_Lines(production_line_id)
);

-- Tabela: Positions
CREATE TABLE Positions (
    position_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    position_name VARCHAR2(50) NOT NULL,
    min_salary NUMBER NOT NULL,
    max_salary NUMBER NOT NULL,
    service_permission NUMBER(1) CHECK (service_permission IN (0, 1))
);

-- Tabela: Employees
CREATE TABLE Employees (
    employee_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name VARCHAR2(50) NOT NULL,
    surname VARCHAR2(50) NOT NULL,
    salary NUMBER,
    birth_date DATE NOT NULL,
    gender CHAR(1) CHECK (gender IN ('M', 'F')),
    position_id NUMBER NOT NULL,
    production_line_id NUMBER,
    FOREIGN KEY (position_id) REFERENCES Positions(position_id),
    FOREIGN KEY (production_line_id) REFERENCES
    Production_Lines(production_line_id)
);


-- Tabela: Services
CREATE TABLE Services (
    service_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    service_name VARCHAR2(100) NOT NULL,
    machine_id NUMBER NOT NULL,
    service_date DATE NOT NULL,
    performed_by NUMBER,
    service_reason VARCHAR2(100),
    FOREIGN KEY (machine_id) REFERENCES Machines(machine_id),
    FOREIGN KEY (performed_by) REFERENCES Employees(employee_id)
);

-- Tabela: Products
CREATE TABLE Products (
    product_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    product_name VARCHAR2(100) NOT NULL,
    production_date DATE NOT NULL,
    product_type VARCHAR2(50) NOT NULL,
    production_line_id NUMBER NOT NULL,
    FOREIGN KEY (production_line_id) REFERENCES
    Production_Lines(production_line_id)
);

-- Tabela: Machine_History
CREATE TABLE Machines_History (
    history_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    machine_id NUMBER NOT NULL,
    new_production_line_id NUMBER NOT NULL,
    transfer_date DATE NOT NULL,
    FOREIGN KEY (machine_id) REFERENCES Machines(machine_id),
```

```
    FOREIGN KEY (new_production_line_id) REFERENCES
    Production_Lines(production_line_id)
);
```

Listing 2.1. Skrypty DDL

2.6. Skrypty do załadowania przykładowych danych

```
-- Wstawianie danych do Machine_Types
INSERT INTO Machine_Types
(type_name, service_interval_hours, service_interval_products)
VALUES ('Fermenter', 1000, 5000);
...
COMMIT;

-- Wstawianie danych do Production_Lines
INSERT INTO Production_Lines (line_name, start_date, status)
VALUES ('Fermentation Line 1', DATE '2024-01-01', 'ACTIVE');
...
COMMIT;

-- Wstawianie danych do Machines
INSERT INTO Machines
(machine_name, machine_type_id, production_line_id, status)
VALUES ('Fermenter A', 1, 1, 'ACTIVE');
...
COMMIT;

-- Wstawianie danych do Positions
INSERT INTO Positions
(position_name, min_salary, max_salary, service_permission)
VALUES ('Brewmaster', 5000, 10000, 1);
...
COMMIT;

-- Wstawianie danych do Employees
INSERT INTO Employees
(name, surname, birth_date, gender, position_id, production_line_id)
VALUES ('John', 'Smith', DATE '1990-03-15', 'M', 1, 1);
...
COMMIT;

-- Wstawianie danych do Services
INSERT INTO Services
(service_name, machine_id, service_date, performed_by)
VALUES ('Filter Replacement', 1, DATE '2024-03-01', 2);
...
COMMIT;

-- Wstawianie danych do Products
BEGIN
    FOR i IN 1..10000 LOOP
        INSERT INTO Products
        (product_name, production_date, product_type, production_line_id)
        VALUES ('Beer ' || i, DATE '2024-01-01' +
        DBMS_RANDOM.VALUE(0, 365), 'Pale Ale', MOD(i, 5) + 1);
```

```
    END LOOP;
END;
/
COMMIT;

-- Wstawianie danych do Machines_History
INSERT INTO Machines_History
(machine_id, new_production_line_id, transfer_date)
VALUES (1, 2, DATE '2024-01-01');
...
```

Listing 2.2. Skrypty do załadowania przykładowych danych

3. Procedury, funkcje i wyzwalacze

3.1. Procedury

W poniższym rozdziale przedstawiono wszystkie procedury zaimplementowane w systemie bazy danych, wraz z opisem ich funkcjonalności oraz listą przyjmowanych argumentów. Procedury te realizują kluczowe operacje związane z zarządzaniem maszynami, liniami produkcyjnymi, serwisami oraz produktami.

1. `register_machine`

Procedura służy do rejestracji nowej maszyny w systemie. Dodaje wpis do tabeli maszyn oraz rejestruje zdarzenie w historii maszyn. Argumenty:

— `machine_name` (`VARCHAR2`) – nazwa maszyny,
— `machine_type_id` (`NUMBER`) – identyfikator typu maszyny,
— `new_production_line_id` (`NUMBER`) – identyfikator linii produkcyjnej, do której przypisywana jest maszyna.

2. `scrap_machine`

Procedura oznacza maszynę jako wycofaną z eksploatacji, aktualizując jej status oraz rejestrując zdarzenie w historii maszyn. Argumenty:

— `act_machine_id` (`NUMBER`) – identyfikator maszyny.

3. `transfer_machine`

Procedura odpowiada za przeniesienie maszyny do innej linii produkcyjnej. Aktualizuje odpowiednie dane w tabeli maszyn oraz zapisuje zdarzenie w historii. Argumenty:

— `transfered_machine_id` (`NUMBER`) – identyfikator przenoszonej maszyny,
— `new_production_line_id` (`NUMBER`) – identyfikator nowej linii produkcyjnej,
— `transfer_date` (`DATE`) – data przeniesienia maszyny.

4. `scrap_production_line`

Procedura oznacza linię produkcyjną jako wycofaną z użycia, zmieniając jej status i zapisując datę usunięcia. Argumenty:

— `act_production_line_id` (`NUMBER`) – identyfikator linii produkcyjnej.

5. `start_service`

Procedura inicjuje serwisowanie maszyny. Aktualizuje status maszyny, dodaje wpis do tabeli serwisów i przypisuje jej odpowiednie parametry. Argumenty:

— `act_machine_id` (`NUMBER`) – identyfikator maszyny,
— `new_start_date` (`DATE`) – data rozpoczęcia serwisu,
— `new_service_name` (`VARCHAR2`) – nazwa serwisu,

— `new_service_reason` (`VARCHAR2`) – powód serwisu,
— `new_performed_by` (`NUMBER`) – identyfikator osoby wykonującej serwis.

6. `complete_service`

Procedura kończy serwisowanie maszyny, aktualizując odpowiednie statusy w tabelach oraz dodając datę zakończenia serwisu. Argumenty:

— `act_service_id` (`NUMBER`) – identyfikator serwisu,
— `new_service_status` (`VARCHAR2`) – nowy status serwisu (`COMPLETED` lub `FAILED`),
— `new_end_date` (`DATE`) – data zakończenia serwisu.

7. `create_product`

Procedura służy do utworzenia nowego produktu na wybranej linii produkcyjnej. Sprawdza stan maszyn w linii produkcyjnej oraz aktualizuje statusy maszyn wymagających serwisu. Argumenty:

— `product_name` (`VARCHAR2`) – nazwa produktu,
— `production_line_id` (`NUMBER`) – identyfikator linii produkcyjnej,
— `product_type` (`VARCHAR2`) – typ produktu.

3.2. Funkcje

W systemie zaimplementowano szereg funkcji PL/SQL, które realizują kluczowe operacje i zapewniają poprawne działanie aplikacji. Poniżej przedstawiono listę funkcji wraz z ich argumentami oraz opisem przeznaczenia.

— `check_machine_service`
  — Argumenty:
    — `p_machine_id` – identyfikator maszyny, której status serwisowy jest sprawdzany.
  — Opis: Funkcja sprawdza, czy dana maszyna wymaga serwisu na podstawie interwału czasowego oraz liczby wyprodukowanych produktów od ostatniego serwisu. Zwraca 1, gdy maszyna wymaga serwisu, oraz 0 w przeciwnym przypadku.
— `check_production_line_service`
  — Argumenty:
    — `p_production_line_id` – identyfikator linii produkcyjnej, dla której sprawdzany jest status serwisowy maszyn.
  — Opis: Funkcja zwraca kursor, który zawiera identyfikatory maszyn w ramach danej linii produkcyjnej wymagających serwisu. Wykorzystuje funkcję `check_machine_service` do oceny statusu każdej maszyny.

3.3. Wyzwalacze

W systemie zastosowano szereg wyzwalaczy PL/SQL, które zapewniają integralność danych, automatyzują operacje aktualizacji oraz dodają mechanizmy walidacji. Poniżej przedstawiono listę wyzwalaczy wraz z ich argumentami oraz opisem przeznaczenia.

— `trg_machines_fk_check`
  — Argumenty: brak (wyzwalacz zainicjowany przed operacją INSERT lub UPDATE na tabeli Machines).
  — Opis: Wyzwalacz weryfikuje istnienie powiązanych rekordów w tabelach `Machine_Types` oraz `Production_Lines` przed dodaniem lub modyfikacją danych w tabeli `Machines`. Zgłasza błąd, gdy wymagane rekordy nie istnieje.

— `trg_services_fk_check`
  — Argumenty: brak (wyzwalacz zainicjowany przed operacją INSERT lub UPDATE na tabeli Services).
  — Opis: Wyzwalacz sprawdza istnienie maszyny o podanym identyfikatorze `machine_id` oraz pracownika wykonującego serwis o identyfikatorze `performed_by`. Zgłasza błąd w przypadku braku powiązanych rekordów.
— `trg_products_fk_check`
  — Argumenty: brak (wyzwalacz zainicjowany przed operacją INSERT lub UPDATE na tabeli Products).
  — Opis: Wyzwalacz weryfikuje istnienie linii produkcyjnej o podanym identyfikatorze `production_line_id` przed dodaniem lub modyfikacją danych w tabeli `Products`.
— `trg_employees_fk_check`
  — Argumenty: brak (wyzwalacz zainicjowany przed operacją INSERT lub UPDATE na tabeli Employees).
  — Opis: Wyzwalacz weryfikuje istnienie stanowiska pracy (`position_id`) oraz linii produkcyjnej (`production_line_id`) powiązanych z nowym lub modyfikowanym rekordem w tabeli `Employees`.
— `trg_machines_history_fk_check`
  — Argumenty: brak (wyzwalacz zainicjowany przed operacją INSERT lub UPDATE na tabeli Machines_History).
  — Opis: Wyzwalacz weryfikuje istnienie maszyny (`machine_id`) oraz nowej linii produkcyjnej (`new_production_line_id`) powiązanych z historią przeniesienia maszyny.
— `update_position_salary`
  — Argumenty: brak (wyzwalacz zainicjowany po operacjach INSERT lub UPDATE na tabeli Employees).
  — Opis: Wyzwalacz automatycznie aktualizuje pola `min_salary` oraz `max_salary` w tabeli `Positions`, bazując na pensji nowego lub zmodyfikowanego pracownika.
— `check_employee_majority`
  — Argumenty: brak (wyzwalacz zainicjowany po operacjach INSERT lub UPDATE na tabeli Employees).
  — Opis: Wyzwalacz sprawdza, czy nowo dodany lub zmodyfikowany pracownik osiągnął pełnoletność. W razie naruszenia warunku zgłasza odpowiedni błąd.

4. Testowanie bazy danych

4.1. Skrypty testujące (zapytania SQL)

```sql
-- Sprawdzenie liczby maszyn na każdej linii produkcyjnej
SELECT production_line_id, COUNT(*) AS machine_count
FROM Machines
GROUP BY production_line_id;

-- Weryfikacja, czy pracownicy mają przypisane ważne stanowiska
SELECT e.name, e.surname, e.position_id, p.position_name
FROM Employees e
LEFT JOIN Positions p ON e.position_id = p.position_id
WHERE p.position_id IS NULL;

-- Sprawdzenie, czy maszyny były serwisowane zgodnie z harmonogramem
SELECT m.machine_name, MAX(s.start_date) AS last_service_date
FROM Machines m
LEFT JOIN Services s ON m.machine_id = s.machine_id
GROUP BY m.machine_name
HAVING MAX(s.start_date) < SYSDATE - INTERVAL '6' MONTH;
```

Listing 4.1. Skrypty testujące

4.2. Nietrywialne zapytania (łączenia, grupowanie, filtrowanie)

```sql
-- Łączenie tabel i filtrowanie maszyn aktywnych z ich typami
SELECT m.machine_name, mt.type_name, m.status
FROM Machines m
JOIN Machine_Types mt ON m.machine_type_id = mt.machine_type_id
WHERE m.status = 'ACTIVE';

-- Zestawienie pracowników i linii produkcyjnych
-- z przypisaniem ich stanowisk
SELECT e.name, e.surname, p.position_name, pl.line_name
FROM Employees e
LEFT JOIN Positions p ON e.position_id = p.position_id
LEFT JOIN Production_Lines pl
ON e.production_line_id = pl.production_line_id;

-- Sumaryczna liczba produktów wytworzonych na każdej linii produkcyjnej
SELECT pl.line_name, COUNT(p.product_id) AS total_products
FROM Production_Lines pl
JOIN Products p ON pl.production_line_id = p.production_line_id
GROUP BY pl.line_name;

-- Lista maszyn, które były przeniesione między liniami produkcyjnymi
SELECT mh.machine_id, mh.transfer_date, pl.line_name AS new_line_name
FROM Machines_History mh
```

```
JOIN Production_Lines pl
ON mh.new_production_line_id = pl.production_line_id
WHERE mh.new_state = 'ACTIVE';

-- Wykaz serwisów przeprowadzonych przez konkretnych pracowników
SELECT s.service_name, e.name
AS employee_name, e.surname, s.start_date, s.service_reason
FROM Services s
LEFT JOIN Employees e ON s.performed_by = e.employee_id
WHERE s.service_status = 'COMPLETED';
```

Listing 4.2. Nietrywialne zapytania

4.3. Testowanie funkcji, procedur i wyzwalaczy

4.3.1. Testy procedur

Rejestracja maszyny

— **Test 1:** Rejestracja maszyny w istniejącej linii produkcyjnej. Testuje możliwość dodania maszyny (Machine Alpha) do linii produkcyjnej o ID 1.
— **Test 2:** Rejestracja maszyny w nieistniejącej linii produkcyjnej. Próba dodania maszyny (Machine Beta) do linii produkcyjnej o ID 999, co powinno zakończyć się błędem, ponieważ linia produkcyjna 999 nie istnieje.

Usuwanie maszyny

— **Test 1:** Usunięcie istniejącej maszyny. Testuje możliwość usunięcia maszyny o ID 1.
— **Test 2:** Usunięcie nieistniejącej maszyny. Próba usunięcia maszyny o ID 999, która nie istnieje, co powinno skutkować błędem.

Przenoszenie maszyny

— **Test 1:** Przeniesienie maszyny do istniejącej linii produkcyjnej. Testuje możliwość przeniesienia maszyny o ID 1 do linii produkcyjnej 4.
— **Test 2:** Przeniesienie maszyny do nieistniejącej linii produkcyjnej. Próba przeniesienia maszyny o ID 1 do linii produkcyjnej o ID 999, co powinno zakończyć się błędem.
— **Test 3:** Przenoszenie nieistniejącej maszyny. Próba przeniesienia maszyny o ID 999, która nie istnieje, do linii produkcyjnej 1, co również powinno skończyć się błędem.

Usuwanie linii produkcyjnej

— **Test 1:** Usunięcie istniejącej linii produkcyjnej. Testuje możliwość usunięcia linii produkcyjnej o ID 1.
— **Test 2:** Usunięcie nieistniejącej linii produkcyjnej. Próba usunięcia linii produkcyjnej o ID 999, która nie istnieje, co powinno zakończyć się błędem lub brakiem efektu.

Rozpoczynanie serwisu

— **Test 1:** Rozpoczęcie serwisu dla istniejącej maszyny. Testuje możliwość rozpoczęcia serwisu dla maszyny o ID 1.
— **Test 2:** Rozpoczęcie serwisu dla maszyny, która nie istnieje. Próba rozpoczęcia serwisu dla maszyny o ID 999, która nie istnieje, co powinno zakończyć się błędem.

Zakończenie serwisu

— **Test 1:** Zakończenie serwisu z sukcesem. Testuje możliwość zakończenia serwisu dla maszyny o ID 2.
— **Test 2:** Zakończenie serwisu z niepowodzeniem. Testuje zakończenie serwisu z niepowodzeniem dla maszyny o ID 3.
— **Test 3:** Zakończenie serwisu dla maszyny, która nie istnieje. Próba zakończenia serwisu dla maszyny o ID 999, która nie istnieje, co powinno zakończyć się błędem.

Tworzenie produktu

— **Test 1:** Tworzenie produktu, gdy wszystkie maszyny na linii produkcyjnej są aktywne. Testuje możliwość utworzenia produktu na linii produkcyjnej o ID 1.
— **Test 2:** Tworzenie produktu, gdy maszyny na linii produkcyjnej wymagają serwisu. Próba utworzenia produktu na linii produkcyjnej o ID 1, gdzie przynajmniej jedna maszyna wymaga serwisu, co powinno zakończyć się błędem.

4.3.2. Testy funkcji

Sprawdzenie czy maszyna wymaga serwisu

Testy dla funkcji `check_machine_service` sprawdzają, czy funkcja poprawnie identyfikuje stan maszyny oraz czy spełnione są warunki do serwisu. Oto przykłady:

— **Test 1:** Maszyna w serwisie (status: `'MAINTENANCE'` lub `'PENDING`)
Oczekiwany wynik: `1` (maszyna już jest w serwisie).
— **Test 2:** Maszyna nie wymaga serwisu, ponieważ czas i produkcja są w porządku.
Oczekiwany wynik: `0`.
— **Test 3:** Maszyna wymaga serwisu, ponieważ czas serwisowy został przekroczony.
Oczekiwany wynik: `1`.

Sprawdzenie, które z maszyn wymagają serwisu

Testy dla funkcji `check_production_line_service` sprawdzają, czy funkcja poprawnie identyfikuje maszyny w linii produkcyjnej wymagające serwisu. Oto przykłady:

— **Test 1:** W linii produkcyjnej są maszyny wymagające serwisu.
Oczekiwany wynik: Kursor zawierający maszyny, które wymagają serwisu (np. maszyna 2).
— **Test 2:** W linii produkcyjnej nie ma maszyn wymagających serwisu.
Oczekiwany wynik: Kursor pusty, ponieważ wszystkie maszyny spełniają wymagania serwisowe.
— **Test 3:** W linii produkcyjnej brak maszyn.
Oczekiwany wynik: Kursor pusty, ponieważ brak maszyn w tej linii.

Testy dla danych w tabelach i triggerów

Trigger: `trg_machines_fk_check`

— **Test 1:** Dodanie poprawnych danych do tabeli `Machines`. Testuje możliwość dodania maszyny z istniejącymi identyfikatorami `machine_type_id` i `production_line_id`.
— **Test 2:** Próba dodania maszyny z nieistniejącym `machine_type_id`. Sprawdza, czy system odrzuca próbę dodania maszyny z nieistniejącym identyfikatorem typu maszyny.
— **Test 3:** Próba dodania maszyny z nieistniejącym `production_line_id`. Sprawdza, czy system odrzuca próbę dodania maszyny z nieistniejącym identyfikatorem linii produkcyjnej.

Trigger: `trg_services_fk_check`

— **Test 1:** Dodanie poprawnych danych do tabeli `Services`. Testuje możliwość dodania rekordu serwisu dla istniejącej maszyny oraz pracownika.
— **Test 2:** Próba dodania rekordu serwisu dla nieistnieącej maszyny. Sprawdza, czy system odrzuca próbę dodania serwisu dla nieistniejącej maszyny.
— **Test 3:** Próba dodania rekordu serwisu z nieistniejącym pracownikiem. Sprawdza, czy system odrzuca próbę dodania serwisu z nieistniejącym pracownikiem.

Trigger: `trg_products_fk_check`

— **Test 1:** Dodanie poprawnych danych do tabeli `Products`. Testuje możliwość dodania produktu z istniejącym `production_line_id`.
— **Test 2:** Próba dodania produktu z nieistnieącym `production_line_id`. Sprawdza, czy system odrzuca próbę dodania produktu z nieistniejącym identyfikatorem linii produkcyjnej.

Trigger: `trg_employees_fk_check`

— **Test 1:** Dodanie poprawnych danych do tabeli `Employees`. Testuje możliwość dodania pracownika z istniejącymi identyfikatorami `position_id` i `production_line_id`.
— **Test 2:** Próba dodania pracownika z nieistniejącym `position_id`. Sprawdza, czy system odrzuca próbę dodania pracownika z nieistniejącym identyfikatorem stanowiska.
— **Test 3:** Próba dodania pracownika z nieistniejącym `production_line_id`. Sprawdza, czy system odrzuca próbę dodania pracownika z nieistniejącym identyfikatorem linii produkcyjnej.

Trigger: `trg_machines_history_fk_check`

— **Test 1:** Dodanie poprawnych danych do tabeli `Machines_History`. Testuje możliwość dodania historii maszyny z istniejącymi identyfikatorami `machine_id` i `new_production_line_id`.
— **Test 2:** Próba dodania historii maszyny z nieistniejącym `machine_id`. Sprawdza, czy system odrzuca próbę dodania historii maszyny z nieistniejącym identyfikatorem maszyny.
— **Test 3:** Próba dodania historii maszyny z nieistniejącym `new_production_line_id`. Sprawdza, czy system odrzuca próbę dodania historii maszyny z nieistniejącym identyfikatorem nowej linii produkcyjnej.

Trigger: `update_position_salary`

— **Test 1:** Aktualizacja wynagrodzenia pracownika, który zmienia pensję, która wpływa na `min_salary` i `max_salary` w tabeli `Positions`.
— **Test 2:** Zaktualizowanie pensji pracownika na wartość niższą niż obecne `min_salary`, co powinno zaktualizować wartość `min_salary`.
— **Test 3:** Zaktualizowanie pensji pracownika na wartość wyższą niż obecne `max_salary`, co powinno zaktualizować wartość `max_salary`.

Trigger: `check_employee_majority`

— **Test 1:** Dodanie pracownika pełnoletniego (wiek powyżej 18 lat), co nie powinno wywołać błędu.
— **Test 2:** Próba dodania pracownika niepełnoletniego (wiek poniżej 18 lat), co powinno zakończyć się błędem.

4.4. Testy wydajności z i bez indeksu

```
DROP INDEX idx_products_product_type;
```

```sql
-- Test wydajności bez indeksów (przed dodaniem dodatkowych indeksów)
-- Zapytanie testowe 1: Wyszukiwanie produktów o określonej nazwie
SET TIMING ON;
SELECT *
FROM Products
WHERE product_type = 'Pale Ale 2';
SET TIMING OFF;

-- Zapytanie testowe 2: Grupowanie produktów według nazwy
SET TIMING ON;
SELECT product_type, COUNT(*)
FROM Products
GROUP BY product_type;
SET TIMING OFF;

-- Dodanie indeksu na kolumnie product_name
CREATE INDEX idx_products_product_type ON Products(product_type);

-- Test wydajności z dodatkowym indeksem

-- Powtórzenie zapytań testowych z indeksem na product_name
-- Zapytanie testowe 1: Wyszukiwanie produktów o określonej nazwie
SET TIMING ON;
SELECT *
FROM Products
WHERE product_type = 'Pale Ale 2';
SET TIMING OFF;

-- Zapytanie testowe 2: Grupowanie produktów według nazwy
SET TIMING ON;
SELECT product_type, COUNT(*)
FROM Products
GROUP BY product_type;
SET TIMING OFF;
```

Listing 4.3. Skrypty testujące wydajność

W przeprowadzonych testach wydajności porównano czas wykonania zapytań do bazy danych przed i po dodaniu indeksu na kolumnie `product_type`. Testy miały na celu ocenę wpływu indeksu na szybkość operacji wyszukiwania i grupowania.

Testy bez indeksu

— **Zapytanie 1**: Wyszukiwanie produktów o określonym typie (`'Pale Ale 2'`). Test mierzył czas wykonania zapytania bez zastosowania indeksu na kolumnie `product_type`.
— **Zapytanie 2**: Grupowanie produktów według typu (`product_type`). Czas wykonania tego zapytania również był mierzony bez indeksu.

```
20 000 rows selected.

Elapsed: 00:00:03.380

PRODUCT_TYPE                                        COUNT(*)
--------------------------------------------------- ----------
Pale Ale 3                                               20000
Pale Ale 4                                               20000
Pale Ale 0                                               20000
Pale Ale 2                                               20000
Pale Ale 1                                               20000

Elapsed: 00:00:00.029
```

Rys. 4.1. Wyniki testów bez indeksów

Testy z indeksem

Po utworzeniu indeksu na kolumnie `product_type`, powtórzono te same zapytania, aby zmierzyć czas ich wykonania z uwzględnieniem indeksu:

— **Zapytanie 1**: Powtórne wyszukiwanie produktów o określonym typie (`'Pale Ale 2'`) z indeksem na kolumnie `product_type`.
— **Zapytanie 2**: Powtórne grupowanie produktów według typu (`product_type`) z zastosowaniem indeksu.

```
20 000 rows selected.

Elapsed: 00:00:03.479

PRODUCT_TYPE                                        COUNT(*)
--------------------------------------------------- ----------
Pale Ale 3                                               20000
Pale Ale 4                                               20000
Pale Ale 0                                               20000
Pale Ale 2                                               20000
Pale Ale 1                                               20000

Elapsed: 00:00:00.034
```

Rys. 4.2. Wyniki testów z indeksami

Wnioski

Wydajność bazy danych nie zmieniła się po dodaniu indeksu na typ produktu, co może wynikać z błędu w implementacji lub ze zbyt małej różnorodności danych w tabeli.

5. Aplikacja w Pythonie

5.1. Krótki opis aplikacji

Aplikacja została wykonana w języku Python i jest aplikacją terminalową do centralnego zarządzania danymi z fabryki. Wchodzenie z nią w interakcje, takie jak wybieranie opcji z menu lub wprowadzanie danych, odbywa się poprzez używanie klawiatury.

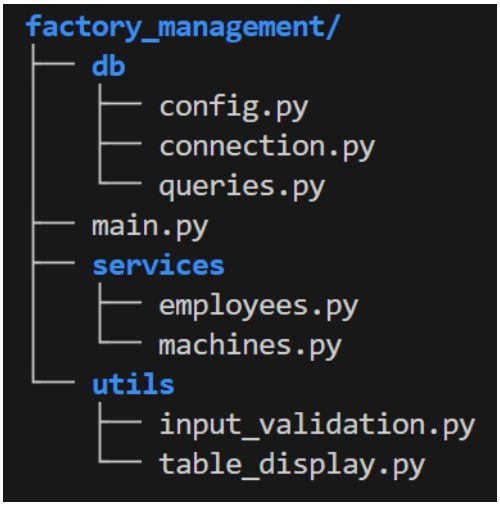
5.2. Wykorzystane technologie

- **Python**
- **Oracle Instant Client for Linux x86-64 (64-bit) wersja 23.6** – lekki klient do połączenia z bazą danych Oracle oraz wykonywania zapytań SQL.
- **cx_Oracle** – biblioteka Python do interakcji z bazą danych Oracle (zapytania, procedury i zarządzanie danymi)
- **rich** – biblioteka Python umożliwiająca tworzenie estetycznych interfejsów tekstowych - wykorzystana do formatowania tabel w konsoli.
- **libaio1** – biblioteka wspierająca asynchroniczne operacje wejścia/wyjścia, wymagana przez Oracle Instant Client.

Szczegółowa instrukcja do instalacji wymaganych bibliotek i klienta znajduje się w pliku `konfiguracja.md`.

5.3. Struktura aplikacji

Aplikacja jest uruchamiana plikiem `main.py`. Struktura projektu została podzielona na odpowiednie foldery w celu zwiększenia rozdzielności między logiką, interfejsem i komunikacją z bazą danych.



Rys. 5.1. Struktura aplikacji

5.4. Wykorzystanie bazy danych do poszczególnych akcji

Zarządzanie pracownikami:

1. **List all employees** - zapytanie z użyciem dwóch lewostronnych złączeń pomiędzy tabelami `Employees`, `Positions`, `Production_Lines`, aby końcowa tabela wyświetlała dane pracownika wraz z nazwami stanowiska i linii produkcyjnej zamiast ich ID.
2. **Add employee** - prosty insert
3. **Fire employee** - select, aby wyszukać pracownika i delete, aby po potwierdzeniu go usunąć
4. **Edit employee** - select, aby wyszukać pracownika i update, aby go edytować

Zarządzanie serwisami:

5. **List all services** - prosty select
6. **Start a service** - wykorzystanie procedury `start_service`
7. **Complete a service** - wykorzystanie procedury `complete_service`
8. **Machines requiring service** - wykorzystanie funkcji `check_production_line_service`, która wewnątrz wykorzystuje `check_machine_service`

Wyświetlanie maszyn:

9. **List machines - advanced search** - rozbudowany select z lewostronnymi złączeniami, filtrowaniem, grupowaniem i sortowaniem według preferencji użytkownika

6. Analiza rozwiązania

1. Mocne strony

— **Modularność:** Podział na funkcje i procedury, takie jak `check_production_line_service` czy `create_product`, wskazuje na dobrze przemyślaną strukturę. Każda jednostka kodu jest odpowiedzialna za konkretną funkcjonalności.
— **Weryfikacja integralności danych:** Triggery, takie jak `trg_machines_fk_check`, dodają dodatkowy poziom kontroli, zwiększając integralność danych.
— **Obsługa błędów:** Mechanizmy zgłaszania wyjątków za pomocą `RAISE_APPLICATION_ERROR` są skuteczne i zapewniają spójne komunikaty diagnostyczne.
— **Dbałość o logikę biznesową:** Procedury i funkcje poprawnie odzwierciedlają reguły biznesowe, np. zatrzymanie procesu produkcji w przypadku maszyn wymagających serwisu.

2. Słabe strony i ryzyka

— **Wydajność:**
  — Użycie kursorów, szczególnie w procedurze `create_product`, jest potencjalnie problematyczne. Przetwarzanie w pętli może być powolne przy dużej ilości danych.
  — W triggerach wielokrotne zapytania `SELECT COUNT(*)` sprawdzające istnienie wartości mogą prowadzić do przeciążeń bazy przy wiékszym obciążeniu.
— **Nadmierna zależność od triggerów:**
  — Triggery, takie jak `trg_services_fk_check`, zwiększają złożoność systemu i utrudniają debugowanie.
  — Mogą powodować nieoczekiwane efekty uboczne w przypadku masowych operacji.
— **Skalowalność:**
  — Procedura `create_product` nie skaluje się dobrze przy dużej ilości danych, ponieważ każda maszyna jest sprawdzana indywidualnie.
— **Potencjalne problemy z równoległością:**
  — Aktualizacje statusu maszyn w pętli mogą powodować konflikty, np. blokady na poziomie wiersza.