# System Rozproszonej Bazy Danych dla Prywatnej Szkoły Podstawowej

## Spis treści

    - Testowanie pakietu
10. Raporty i Analizy Rozproszone (10-raporty.md)
    - Procedury raportowe SQL Server
    - Zapytania wieloźródłowe
    - Widoki rozproszone SQL Server
    - Procedury analityczne Oracle
    - Eksport i integracja danych
    - Monitorowanie wydajności raportów
11. Testowanie i Weryfikacja (11-testowanie.md)
    - Testy integracji PostgreSQL
    - Testy pakietu Oracle
    - Testy transakcji rozproszonych
    - Testy wydajnościowe
    - Testy integralności danych
    - Testy recovery i failover
12. Instrukcja Instalacji i Uruchomienia (12-instalacja.md)
    - Wymagania systemowe
    - Kolejność instalacji
    - Konfiguracja replikacji
    - Instalacja funkcjonalności systemowych
    - Testowanie instalacji
    - Rozwiązywanie problemów
13. Podsumowanie i Wnioski (13-podsumowanie.md)
    - Realizacja celów projektu
    - Architektura jako best practice
    - Wyzwania i rozwiązania
    - Możliwości rozwoju
    - Wartość edukacyjna projektu
    - Wnioski końcowe

---

## Informacje o projekcie

**Temat:** System Rozproszonej Bazy Danych dla Prywatnej Szkoły Podstawowej

**Technologie:**

- Microsoft SQL Server 2019+ (dane edukacyjne, replikacja)
- Oracle Database 19c+ (dane finansowe, database links)
- PostgreSQL 12+ (uwagi pedagogiczne)

**Kluczowe funkcjonalności:**

- Linked Servers między wszystkimi systemami
- Database Links Oracle (symulacja środowiska rozproszonego)
- Transakcje rozproszone z MS DTC
- Replikacja transakcyjna
- Zapytania OPENQUERY i OPENROWSET
- Eksport do Excel
- Kompletny system CRUD
- Raporty wielosystemowe

**Charakterystyka systemu:**

- 15 tabel w SQL Server (uczniowie, nauczyciele, oceny, frekwencja)
- 2 tabele główne + 2 zdalne w Oracle (kontrakty, płatności)
- 1 tabela w PostgreSQL (uwagi nauczycieli)
- 17+ procedur składowanych SQL Server
- Pakiet PL/SQL z 3 procedurami i 1 funkcją
- 3 widoki rozproszone
- 2 procedury transakcji rozproszonych

**Dokumentacja:**
Kompletna dokumentacja techniczna zawierająca szczegółowe opisy wszystkich komponentów systemu, instrukcje instalacji, konfiguracji i testowania oraz analizy architektoniczne.

*Dokumentacja została utworzona dla projektu z przedmiotu Rozproszone Bazy Danych. System demonstruje praktyczne zastosowanie wszystkich kluczowych konceptów RBD w środowisku heterogenicznym.*

# Sistem Rozproszonej Bazy Danych dla Prywatnej Szkoły Podstawowej

## 1. Wprowadzenie

### 1.1 Opis systemu

System Rozproszonej Bazy Danych (RBD) został zaprojektowany dla potrzeb zarządzania prywatną szkołą podstawową. Rozwiązanie integruje trzy różne systemy bazodanowe w środowisku heterogenicznym, zapewniając kompleksowe wsparcie dla wszystkich aspektów działalności szkolnej.

### 1.2 Cele projektu

Główne cele realizowane przez system to:

- **Zarządzanie danymi edukacyjnymi**: Przechowywanie informacji o uczniach, nauczycielach, klasach, przedmiotach, ocenach i frekwencji
- **Obsługa finansowa**: Kompleksowe zarządzanie kontraktami finansowymi, opłatami szkolnymi i płatnościami
- **System uwag pedagogicznych**: Rejestrowanie uwag, pochwał i obserwacji dotyczących postępów uczniów
- **Integracja danych**: Zapewnienie spójnego dostępu do danych rozproszonych między różnymi systemami

### 1.3 Architektura systemu

System składa się z trzech głównych komponentów bazodanowych:

#### Microsoft SQL Server (Serwer główny)

- **Baza**: SchoolDB
- **Rola**: Centralny system zarządzania danymi szkolnymi
- **Zawartość**: Dane uczniów, nauczycieli, klas, przedmiotów, ocen, lekcji, frekwencji

#### Oracle Database (Serwer finansowy)

- **Baza**: FINANCE_DB
- **Rola**: Obsługa aspektów finansowych szkoły
- **Zawartość**: Kontrakty finansowe, płatności, rozliczenia

#### PostgreSQL (Serwer uwag)

- **Baza**: remarks_main
- **Rola**: System uwag i komentarzy pedagogicznych
- **Zawartość**: Uwagi nauczycieli, pochwały, obserwacje postępów

### 1.4 Kluczowe technologie

System wykorzystuje następujące technologie i mechanizmy:

- **Linked Servers**: Połączenia między SQL Server a innymi systemami
- **Database Links**: Połączenia Oracle z innymi schematami
- **OPENQUERY/OPENROWSET**: Zapytania ad-hoc do zdalnych źródeł
- **Replikacja transakcyjna**: Synchronizacja danych między serwerami SQL Server
- **Transakcje rozproszone**: Zapewnienie spójności danych między systemami
- **Widoki rozproszone**: Jednolity dostęp do danych z różnych źródeł
- **Triggery INSTEAD OF**: Zarządzanie modyfikacjami w widokach rozproszonych

### 1.5 Korzyści rozwiązania

- **Specjalizacja**: Każdy system jest zoptymalizowany do swojej domeny
- **Skalowalność**: Możliwość niezależnego rozwijania poszczególnych komponentów
- **Bezpieczeństwo**: Separacja krytycznych danych finansowych
- **Wydajność**: Równoległe przetwarzanie zapytań na różnych serwerach
- **Elastyczność**: Możliwość wykorzystania najlepszych cech każdego systemu

### 1.6 Miejsce na diagram architektury

*[Tutaj zostanie umieszczony diagram przedstawiający architekturę systemu, pokazujący połączenia między serwerami MS SQL, Oracle i PostgreSQL, oraz przepływ danych między nimi]*

# 2. Architektura i Podział Systemu

## 2.1 Strategia podziału danych

### 2.1.1 Podział funkcjonalny

System został podzielony według kryterium funkcjonalnego, gdzie każdy serwer obsługuje określoną domenę biznesową:

**Microsoft SQL Server - Dane edukacyjne**

**Uzasadnienie wyboru:**

- SQL Server oferuje doskonałe wsparcie dla aplikacji typu OLTP
- Natywne wsparcie dla replikacji transakcyjnej
- Zaawansowane mechanizmy zarządzania tożsamością (IDENTITY)
- Dobre integracje z narzędziami Microsoft

**Przechowywane dane:**

- **Dane podstawowe**: uczniowie, nauczyciele, rodzice, klasy
- **Dane akademickie**: przedmioty, lekcje, oceny, frekwencja
- **Dane organizacyjne**: sale lekcyjne, godziny lekcyjne, dni tygodnia

**Oracle Database - Dane finansowe**

**Uzasadnienie wyboru:**

- Oracle zapewnia wysoką wydajność dla operacji finansowych
- Zaawansowane mechanizmy transakcyjne i blokowania
- Wsparcie dla database links umożliwiające symulację środowiska rozproszonego
- Silne zabezpieczenia danych finansowych

**Przechowywane dane:**

- **Kontrakty**: umowy finansowe z rodzicami
- **Płatności**: obsługa opłat szkolnych, status płatności
- **Podsumowania**: agregowane dane finansowe w zdalnych schematach

**PostgreSQL - Uwagi pedagogiczne**

**Uzasadnienie wyboru:**

- PostgreSQL oferuje elastyczność w przechowywaniu tekstów
- Dobra wydajność dla operacji odczytu uwag
- Prosty model danych odpowiedni dla komentarzy tekstowych
- Symulacja trzeciego środowiska w architekturze heterogenicznej

**Przechowywane dane:**

- **Uwagi**: komentarze nauczycieli o postępach uczniów
- **Pochwały**: pozytywne obserwacje
- **Dane czasowe**: automatyczne znaczniki czasowe

## 2.2 Schemat komunikacji między serwerami

### 2.2.1 SQL Server jako centrum integracji

SQL Server pełni rolę głównego serwera integracyjnego, który:

- Łączy się ze wszystkimi innymi systemami poprzez Linked Servers
- Udostępnia zunifikowane widoki danych rozproszonych
- Koordynuje transakcje rozproszone
- Realizuje eksport danych do formatów zewnętrznych (Excel)

### 2.2.2 Oracle z database links

Oracle symuluje środowisko rozproszone poprzez:

- **REMOTE_DB1**: symulacja zdalnego serwera z dodatkowymi kontraktami
- **REMOTE_DB2**: system podsumowań i analityki finansowej
- **Database links**: połączenia między schematami symulujące rzeczywiste środowisko rozproszone

### 2.2.3 PostgreSQL jako zewnętrzny system

PostgreSQL działa jako niezależny system, który:

- Przechowuje uwagi w izolowanym środowisku
- Jest dostępny poprzez ODBC i Linked Server z SQL Server
- Symuluje zewnętrzny system pedagogiczny

## 2.3 Zalety przyjętego podziału

### 2.3.1 Bezpieczeństwo danych

- **Separacja danych finansowych**: krytyczne dane finansowe są odizolowane w dedykowanym systemie Oracle
- **Kontrola dostępu**: każdy system ma niezależne mechanizmy uwierzytelniania
- **Minimalizacja ryzyka**: awaria jednego systemu nie wpływa na pozostałe

### 2.3.2 Wydajność

- **Specjalizacja serwerów**: każdy serwer jest zoptymalizowany pod konkretne operacje
- **Równoległe przetwarzanie**: możliwość jednoczesnego wykonywania operacji na różnych serwerach
- **Redukcja obciążenia**: rozdzielenie obciążenia między systemy

### 2.3.3 Skalowalność

- **Niezależne skalowanie**: każdy komponent może być skalowany niezależnie
- **Łatwość rozbudowy**: dodawanie nowych funkcjonalności w dedykowanych systemach
- **Modularność**: jasne granice między komponentami systemu

### 2.3.4 Zgodność z wymaganiami projektu

- **Środowisko heterogeniczne**: wykorzystanie trzech różnych systemów bazodanowych
- **Database links Oracle**: implementacja połączeń między schematami
- **Linked Servers**: pełne wsparcie dla różnych typów połączeń
- **Zapytania ad-hoc**: implementacja OPENQUERY i OPENROWSET

# 3. Konfiguracja Środowiska

## 3.1 Przygotowanie serwerów bazodanowych

### 3.1.1 Microsoft SQL Server

System wymaga konfiguracji głównego serwera SQL Server z następującymi ustawieniami:

**Baza danych:** SchoolDB
**Konfiguracja zaawansowana:**

- Włączenie Ad Hoc Distributed Queries
- Konfiguracja MS DTC (Distributed Transaction Coordinator)
- Ustawienie opcji RPC dla linked servers

**Użytkownicy i uprawnienia:**

- Konto systemowe dla replikacji (repl_user)
- Uprawnienia do zarządzania linked servers
- Uprawnienia do wykonywania transakcji rozproszonych

### 3.1.2 Oracle Database

**Konfiguracja użytkowników:**
System Oracle wymaga utworzenia trzech głównych użytkowników:

**FINANCE_DB (Główny użytkownik finansowy)**

- **Hasło:** Finance123
- **Tablespace:** USERS (unlimited quota)
- **Uprawnienia:** CONNECT, RESOURCE, CREATE DATABASE LINK, CREATE SYNONYM, CREATE VIEW, CREATE MATERIALIZED VIEW, CREATE SEQUENCE, CREATE TRIGGER, CREATE PROCEDURE

**REMOTE_DB1 (Symulacja zdalnego serwera 1)**

- **Hasło:** Remote123
- **Tablespace:** USERS
- **Uprawnienia:** CONNECT, RESOURCE, CREATE DATABASE LINK, CREATE SYNONYM, CREATE VIEW, CREATE PROCEDURE
- **Rola:** Przechowywanie dodatkowych kontraktów

**REMOTE_DB2 (Symulacja zdalnego serwera 2)**

- **Hasło:** Remote123
- **Tablespace:** USERS
- **Uprawnienia:** CONNECT, RESOURCE, CREATE DATABASE LINK, CREATE SYNONYM, CREATE VIEW, CREATE PROCEDURE
- **Rola:** System podsumowań płatności

**Database Links:**

- Połączenia między użytkownikami symulujące rozproszone środowisko
- Synonimy dla ułatwienia dostępu do zdalnych tabel
- Wzajemne uprawnienia SELECT, INSERT, UPDATE, DELETE

### 3.1.3 PostgreSQL

**Baza danych:** remarks_system
**Schemat:** remarks_main

**Konfiguracja użytkowników:**

- **remarks_user** z hasłem Remarks123
- Uprawnienia USAGE na schemat remarks_main
- Pełny dostęp do tabeli remark

**Konfiguracja ODBC:**

- Sterownik PostgreSQL30 dla połączenia z SQL Server
- Konfiguracja DSN dla Linked Server

## 3.2 Konfiguracja Linked Servers

### 3.2.1 Linked Server do Oracle

**Nazwa:** ORACLE_FINANCE
**Provider:** OraOLEDB.Oracle
**Data Source:** 127.0.0.1:1521/PD19C
**Mapowanie loginów:** FINANCE_DB/Finance123

### 3.2.2 Linked Server do PostgreSQL

**Nazwa:** POSTGRES_REMARKS
**Provider:** MSDASQL
**Data Source:** PostgreSQL30 (DSN)
**Mapowanie loginów:** remarks_user/Remarks123

### 3.2.3 Linked Server do replikacji

**Nazwa:** MSSQL_REPLICA
**Provider:** MSOLEDBSQL
**Data Source:** 127.0.0.1,1434
**Mapowanie loginów:** sa/Str0ng!Passw0rd

### 3.2.4 Linked Server do Excel

**Nazwa:** EXCEL_DATA
**Provider:** Microsoft.ACE.OLEDB.12.0
**Data Source:** C:\excel_exports\SchoolData.xlsx
**Connection String:** Excel 12.0;HDR=YES;

## 3.3 Konfiguracja replikacji

### 3.3.1 Distributor

- **Baza dystrybucji:** distribution
- **Katalogi:** C:\MSSQL\Data (dane), C:\ReplData (working directory)
- **Publisher:** lokalny serwer SQL Server

### 3.3.2 Publikacja

- **Nazwa:** SchoolDB_StudentsOnly
- **Typ:** replikacja transakcyjna (continuous)
- **Artykuł:** tabela students
- **Sync method:** concurrent

### 3.3.3 Subskrypcja

- **Subscriber:** 127.0.0.1,1434 (MSSQL_REPLICA)
- **Destination DB:** SchoolDB_Replica
- **Typ:** Push subscription
- **Update mode:** read only

## 3.4 Konfiguracja MS DTC

Microsoft Distributed Transaction Coordinator musi być skonfigurowany dla obsługi transakcji rozproszonych:

- **Network DTC Access:** włączony
- **Allow Inbound/Outbound:** włączone
- **Authentication:** None required (środowisko testowe)
- **Enable XA Transactions:** włączone

## 3.5 Kolejność uruchamiania

**Prawidłowa kolejność konfiguracji środowiska:**

1. **Oracle Database**
    - Utworzenie użytkowników (1_user.sql)
    - Konfiguracja schematów finansowych (2_finance_create.sql)
    - Konfiguracja zdalnych schematów (3_remote_db1.sql, 4_remote_db2.sql)
    - Utworzenie synonimów (5_finance_synonyms.sql)
2. **PostgreSQL**
    - Utworzenie bazy i schematu (create.sql)
    - Konfiguracja użytkowników i uprawnień
3. **SQL Server**
    - Utworzenie bazy SchoolDB (create.sql, data.sql)
    - Konfiguracja zaawansowana (config.sql)
    - Konfiguracja linked servers
    - Konfiguracja replikacji
4. **Wypełnienie danymi**
    - Dane Oracle (6_finance_data.sql, 7_remote_db1_data.sql, 8_remote_db2_data.sql)
    - Dane PostgreSQL (data.sql)
5. **Weryfikacja połączeń**
    - Testowanie linked servers
    - Weryfikacja replikacji
    - Testowanie transakcji rozproszonych

# 4. Struktura Baz Danych

## 4.1 Microsoft SQL Server - SchoolDB

### 4.1.1 Tabele słownikowe

**genders**

- **id** (int IDENTITY, PK): Unikalny identyfikator płci
- **value** (nvarchar(50)): Wartość (Male, Female, Other)

**years**

- **id** (int IDENTITY, PK): Unikalny identyfikator roku szkolnego
- **value** (int): Rok (2023, 2024, 2025)

**days**

- **id** (int IDENTITY, PK): Unikalny identyfikator dnia
- **value** (nvarchar(20)): Nazwa dnia (Monday, Tuesday, ...)

**hours**

- **id** (int IDENTITY, PK): Unikalny identyfikator godziny lekcyjnej
- **start_hour** (int): Godzina rozpoczęcia
- **start_minutes** (int): Minuty rozpoczęcia
- **end_hour** (int): Godzina zakończenia
- **end_minutes** (int): Minuty zakończenia

### 4.1.2 Tabele podmiotów

**teachers**

- **id** (int IDENTITY, PK): Unikalny identyfikator nauczyciela
- **firstName** (nvarchar(100)): Imię nauczyciela
- **lastName** (nvarchar(100)): Nazwisko nauczyciela
- **birthday** (date): Data urodzenia
- **phoneNumber** (nvarchar(20)): Numer telefonu
- **email** (nvarchar(100)): Adres email
- **additionalInfo** (nvarchar(500)): Dodatkowe informacje

**parents**

- **id** (int IDENTITY, PK): Unikalny identyfikator rodzica
- **firstName** (nvarchar(100)): Imię rodzica
- **lastName** (nvarchar(100)): Nazwisko rodzica
- **phoneNumber** (nvarchar(20)): Numer telefonu
- **email** (nvarchar(100)): Adres email

**groups**

- **id** (int IDENTITY, PK): Unikalny identyfikator klasy
- **yearId** (int, FK): Odniesienie do roku szkolnego
- **home_teacher_id** (int, FK): Odniesienie do wychowawcy klasy

**students**

- **id** (int IDENTITY, PK): Unikalny identyfikator ucznia
- **groupId** (int, FK): Odniesienie do klasy
- **firstName** (nvarchar(100)): Imię ucznia
- **lastName** (nvarchar(100)): Nazwisko ucznia
- **birthday** (date): Data urodzenia
- **genderId** (int, FK): Odniesienie do płci

### 4.1.3 Tabele relacyjne

**parents_students**

- **id** (int IDENTITY, PK): Unikalny identyfikator relacji
- **parentId** (int, FK): Odniesienie do rodzica
- **studentId** (int, FK): Odniesienie do ucznia

### 4.1.4 Tabele edukacyjne

**classrooms**

- **id** (int IDENTITY, PK): Unikalny identyfikator sali
- **location** (nvarchar(100)): Lokalizacja sali

**subjects**

- **id** (int IDENTITY, PK): Unikalny identyfikator przedmiotu
- **shortName** (nvarchar(10)): Skrócona nazwa (np. MATH, ENG)
- **longName** (nvarchar(100)): Pełna nazwa przedmiotu

**lessons**

- **id** (int IDENTITY, PK): Unikalny identyfikator lekcji
- **teacherId** (int, FK): Odniesienie do nauczyciela
- **subjectId** (int, FK): Odniesienie do przedmiotu
- **groupId** (int, FK): Odniesienie do klasy
- **hourId** (int, FK): Odniesienie do godziny lekcyjnej
- **classroomId** (int, FK): Odniesienie do sali
- **dayId** (int, FK): Odniesienie do dnia tygodnia

**marks**

- **id** (int IDENTITY, PK): Unikalny identyfikator oceny
- **subjectId** (int, FK): Odniesienie do przedmiotu
- **studentId** (int, FK): Odniesienie do ucznia
- **value** (int): Wartość oceny (1-6)
- **comment** (nvarchar(500)): Komentarz do oceny
- **weight** (int): Waga oceny (domyślnie 1)

### 4.1.5 Tabele frekwencji

**attendances**

- **id** (int IDENTITY, PK): Unikalny identyfikator sprawdzania frekwencji
- **dateTimeChecked** (datetime): Data i czas sprawdzenia
- **lessonId** (int, FK): Odniesienie do lekcji

**attendance_student**

- **id** (int IDENTITY, PK): Unikalny identyfikator obecności ucznia
- **attendanceId** (int, FK): Odniesienie do sprawdzania frekwencji
- **studentId** (int, FK): Odniesienie do ucznia
- **present** (bit): Flaga obecności (1 - obecny, 0 - nieobecny)

## 4.2 Oracle Database - FINANCE_DB

### 4.2.1 Główny schemat finansowy

**contracts**

- **id** (NUMBER, PK): Unikalny identyfikator kontraktu (sekwencja)
- **studentId** (NUMBER): Identyfikator ucznia (odniesienie do SQL Server)
- **parentId** (NUMBER): Identyfikator rodzica (odniesienie do SQL Server)
- **startDate** (DATE): Data rozpoczęcia kontraktu
- **endDate** (DATE): Data zakończenia kontraktu
- **monthlyAmount** (NUMBER(10,2)): Miesięczna kwota opłaty

**payments**

- **id** (NUMBER, PK): Unikalny identyfikator płatności (sekwencja)
- **contractId** (NUMBER, FK): Odniesienie do kontraktu
- **dueDate** (DATE): Termin płatności
- **paidDate** (DATE): Data dokonania płatności
- **amount** (NUMBER(10,2)): Kwota płatności
- **status** (VARCHAR2(20)): Status płatności (PENDING, PAID, OVERDUE)

### 4.2.2 Zdalne schematy (symulacja środowiska rozproszonego)

**REMOTE_DB1.contracts_remote**

- **id** (NUMBER, PK): Identyfikator kontraktu (sekwencja startująca od 1000)
- **studentId** (NUMBER): Identyfikator ucznia
- **parentId** (NUMBER): Identyfikator rodzica
- **startDate** (DATE): Data rozpoczęcia
- **endDate** (DATE): Data zakończenia
- **monthlyAmount** (NUMBER(10,2)): Miesięczna kwota

**REMOTE_DB2.payment_summary**

- **id** (NUMBER, PK): Identyfikator podsumowania
- **contractId** (NUMBER): Identyfikator kontraktu
- **totalAmount** (NUMBER(10,2)): Łączna kwota płatności
- **paymentCount** (NUMBER): Liczba płatności
- **lastPaymentDate** (DATE): Data ostatniej płatności

### 4.2.3 Mechanizmy automatyzacji

**Sekwencje:**

- **contract_seq**: Auto-increment dla kontraktów (start: 1)
- **payment_seq**: Auto-increment dla płatności (start: 1)
- **contract_remote_seq**: Auto-increment dla zdalnych kontraktów (start: 1000)
- **payment_summary_seq**: Auto-increment dla podsumowań (start: 1)

**Triggery:**

- **contract_trigger**: Automatyczne przypisywanie ID dla nowych kontraktów
- **payment_trigger**: Automatyczne przypisywanie ID dla nowych płatności
- **contract_remote_trigger**: Automatyczne ID dla zdalnych kontraktów
- **payment_summary_trigger**: Automatyczne ID dla podsumowań

**Indeksy:**

- **idx_contracts_student**: Indeks na studentId w tabeli contracts
- **idx_contracts_parent**: Indeks na parentId w tabeli contracts
- **idx_payments_contract**: Indeks na contractId w tabeli payments
- **idx_payments_status**: Indeks na status w tabeli payments

## 4.3 PostgreSQL - remarks_main

### 4.3.1 Struktura uwag

**remarks_main.remark**

- **id** (SERIAL, PK): Unikalny identyfikator uwagi
- **studentId** (INTEGER): Identyfikator ucznia (odniesienie do SQL Server)
- **teacherId** (INTEGER): Identyfikator nauczyciela (odniesienie do SQL Server)
- **value** (TEXT): Treść uwagi lub pochwały
- **created_date** (TIMESTAMP): Data utworzenia (domyślnie CURRENT_TIMESTAMP)

**Indeksy:**

- **idx_remark_student**: Indeks na studentId
- **idx_remark_teacher**: Indeks na teacherId
- **idx_remark_date**: Indeks na created_date

## 4.4 Relacje międzysystemowe

### 4.4.1 Klucze obce logiczne

Ze względu na rozproszoną naturę systemu, relacje między systemami są utrzymywane logicznie poprzez:

- **studentId/teacherId/parentId**: Wspólne identyfikatory łączące dane między systemami
- **Widoki rozproszone**: Łączenie danych z różnych źródeł
- **Procedury walidacyjne**: Sprawdzanie integralności referencyjnej przy operacjach rozproszonych

### 4.4.2 Strategia spójności danych

- **Eventual consistency**: Dane są spójne po zakończeniu wszystkich operacji
- **Transakcje rozproszone**: Dla krytycznych operacji wymagających atomowości
- **Kompensacja**: Mechanizmy rollback dla operacji rozproszonych

# 5. Połączenia Rozproszone

## 5.1 Linked Servers (SQL Server)

### 5.1.1 Konfiguracja połączeń

System wykorzystuje pięć głównych typów linked servers, każdy skonfigurowany dla określonego celu:

#### ORACLE_FINANCE

- **Cel**: Dostęp do danych finansowych w Oracle
- **Provider**: OraOLEDB.Oracle
- **Technologia**: Native Oracle provider dla optymalnej wydajności
- **Uwierzytelnianie**: Login mapping (FINANCE_DB/Finance123)
- **Użycie**: Zapytania finansowe, transakcje płatności, raporty

#### POSTGRES_REMARKS

- **Cel**: Dostęp do systemu uwag w PostgreSQL
- **Provider**: MSDASQL (ODBC)
- **Technologia**: ODBC driver przez DSN
- **Uwierzytelnianie**: Login mapping (remarks_user/Remarks123)
- **Użycie**: Operacje CRUD na uwagach, raporty pedagogiczne

#### MSSQL_REPLICA

- **Cel**: Replikacja danych między instancjami SQL Server
- **Provider**: MSOLEDBSQL
- **Technologia**: Native SQL Server provider
- **Uwierzytelnianie**: SQL Server authentication (sa account)
- **Użycie**: Replikacja transakcyjna tabeli students

#### EXCEL_DATA

- **Cel**: Eksport danych do plików Excel
- **Provider**: Microsoft.ACE.OLEDB.12.0
- **Technologia**: ACE OLEDB dla plików Office
- **Konfiguracja**: HDR=YES (pierwszy wiersz jako nagłówki)
- **Użycie**: Eksport raportów do formatu Excel

### 5.1.2 Konfiguracja RPC

Wszystkie linked servers zostały skonfigurowane z włączonymi opcjami RPC:

- **rpc**: true - umożliwia wykonywanie procedur na zdalnym serwerze
- **rpc out**: true - umożliwia wywołania wychodzące
- **data access**: true - umożliwia dostęp do danych

## 5.2 Database Links (Oracle)

### 5.2.1 Architektura linków

Oracle wykorzystuje database links do symulacji prawdziwego środowiska rozproszonego:

**Linki między schematami Oracle**

System zawiera kilka typów database links:

**FINANCE_DB → REMOTE_DB1**

- **Cel**: Dostęp do dodatkowych kontraktów w "zdalnym" systemie
- **Typ**: Private database link
- **Implementacja**: Poprzez system uprawnień Oracle

**FINANCE_DB → REMOTE_DB2**

- **Cel**: Dostęp do podsumowań płatności w systemie analitycznym
- **Typ**: Private database link
- **Implementacja**: Cross-schema access

### 5.2.2 Synonimy

Dla ułatwienia dostępu do zdalnych tabel zostały utworzone synonimy:

**remote_contracts**

- **Cel**: Alias dla REMOTE_DB1.contracts_remote
- **Lokalizacja**: Schema FINANCE_DB
- **Użycie**: Transparentny dostęp do zdalnych kontraktów

**remote_payment_summary**

- **Cel**: Alias dla REMOTE_DB2.payment_summary
- **Lokalizacja**: Schema FINANCE_DB
- **Użycie**: Dostęp do podsumowań płatności

### 5.2.3 Wzajemne uprawnienia

Między schematami Oracle zostały skonfigurowane wzajemne uprawnienia:

- FINANCE_DB ma uprawnienia SELECT, INSERT, UPDATE, DELETE na tabelach REMOTE_DB1 i REMOTE_DB2
- REMOTE_DB1 i REMOTE_DB2 mają uprawnienia dostępu do tabel FINANCE_DB
- Uprawnienia umożliwiają pełną synchronizację danych

## 5.3 Mechanizmy dostępu do danych

### 5.3.1 OPENQUERY

OPENQUERY jest używany do wykonywania zapytań na zdalnych serwerach z przetwarzaniem po stronie zdalnej:

**Dostęp do Oracle**

- **Zapytania finansowe**: Agregacje płatności wykonywane w Oracle
- **Filtrowanie zdalne**: Przekazywanie warunków WHERE do Oracle
- **Optymalizacja**: Zmniejszenie transferu danych przez sieć

**Dostęp do PostgreSQL**

- **Zapytania uwag**: Pobieranie uwag z filtrami na studentId/teacherId
- **Agregacje**: Liczenie uwag per student/teacher na serwerze PostgreSQL
- **Sortowanie zdalne**: Przekazywanie ORDER BY do PostgreSQL

**Przykłady zastosowań:**

- Pobieranie danych finansowych z grupowaniem
- Filtrowanie uwag według daty utworzenia
- Agregowanie statystyk na zdalnych serwerach

### 5.3.2 OPENROWSET

OPENROWSET jest wykorzystywany do zapytań ad-hoc i operacji na plikach:

**Dostęp do Excel**

- **Eksport danych**: Wstawianie danych do arkuszy Excel
- **Format**: Automatyczne rozpoznawanie nagłówków (HDR=YES)
- **Multiple worksheets**: Eksport do różnych arkuszy (Student$, ContractsPayments$, Remarks$)

**Zapytania ad-hoc**

- **Jednorazowe operacje**: Dostęp bez konfiguracji linked server
- **Testowanie połączeń**: Weryfikacja dostępności zdalnych źródeł
- **Import danych**: Jednorazowy import z zewnętrznych źródeł

### 5.3.3 Standardowy dostęp przez Linked Server

Dla rutynowych operacji używany jest standardowy dostęp:

**Four-part naming**

- **Format**: [LinkedServer].[Database].[Schema].[Object]
- **Przykład**: ORACLE_FINANCE..FINANCE_DB.CONTRACTS
- **Zalety**: Prostota składni, przejrzystość kodu

**Zapytania łączące**

- **Cross-database JOINs**: Łączenie danych z różnych systemów
- **Widoki rozproszone**: Jednolity dostęp do danych rozproszonych
- **Procedury rozproszone**: Koordynacja operacji między systemami

## 5.4 Optymalizacja wydajności

### 5.4.1 Strategia przetwarzania

**Przetwarzanie zdalne:**

- Agregacje i filtrowanie wykonywane na zdalnych serwerach
- Minimalizacja transferu danych przez sieć
- Wykorzystanie indeksów na zdalnych systemach

**Przetwarzanie lokalne:**

- Łączenie wyników z różnych systemów
- Formatowanie danych dla prezentacji
- Aplikowanie reguł biznesowych

### 5.4.2 Cachowanie połączeń

Linked servers są skonfigurowane z opcjami wydajnościowymi:

- **Connection pooling**: Ponowne wykorzystanie połączeń
- **Query timeout**: Odpowiednie limity czasowe
- **Lazy schema validation**: Optymalizacja walidacji schematów

## 5.5 Bezpieczeństwo połączeń

### 5.5.1 Uwierzytelnianie

**Oracle:**

- Dedykowany użytkownik FINANCE_DB z ograniczonymi uprawnieniami
- Hasła przechowywane w konfiguracji linked server
- Brak konta administracyjnego dla połączeń aplikacyjnych

**PostgreSQL:**

- Użytkownik remarks_user z dostępem tylko do schematu remarks_main
- Izolacja od innych danych w PostgreSQL
- Połączenie przez ODBC z uwierzytelnianiem użytkownika

### 5.5.2 Ograniczenia dostępu

**Filtrowanie na poziomie aplikacji:**

- Weryfikacja uprawnień przed dostępem do danych rozproszonych
- Logowanie operacji rozproszonych
- Kontrola dostępu na poziomie procedur składowanych

**Network security:**

- Wszystkie połączenia w sieci lokalnej (127.0.0.1)
- Brak połączeń zewnętrznych w środowisku produkcyjnym
- Konfiguracja firewalla dla dostępu między serwerami

# 6. Funkcjonalności Systemowe

## 6.1 Operacje CRUD

### 6.1.1 Zarządzanie uczniami

**sp_CreateStudent**

**Parametry wejściowe:**

- @GroupId (INT): Identyfikator klasy (wymagany)
- @FirstName (NVARCHAR(100)): Imię ucznia (wymagane)
- @LastName (NVARCHAR(100)): Nazwisko ucznia (wymagane)
- @Birthday (DATE): Data urodzenia (opcjonalna)
- @GenderId (INT): Identyfikator płci (opcjonalny)

**Parametry wyjściowe:**

- @StudentId (INT OUTPUT): Identyfikator utworzonego ucznia

**Funkcjonalność:**

- Walidacja wymaganych pól (GroupId, FirstName, LastName)
- Sprawdzenie istnienia klasy w tabeli groups
- Walidacja płci (jeśli podana)
- Walidacja wieku (0-25 lat)
- Obsługa transakcji z mechanizmem rollback
- Zwrócenie ID nowo utworzonego ucznia

**sp_UpdateStudent**

**Parametry:**

- @StudentId (INT): ID ucznia do aktualizacji (wymagany)
- @GroupId, @FirstName, @LastName, @Birthday, @GenderId: Nowe wartości (opcjonalne)

**Funkcjonalność:**

- Aktualizacja tylko podanych pól (ISNULL pattern)
- Walidacja referencji do grup i płci
- Sprawdzenie istnienia ucznia przed aktualizacją

**sp_DeleteStudent**

**Parametry:**

- @StudentId (INT): ID ucznia do usunięcia
- @ForceDelete (BIT): Flaga wymuszenia usunięcia (domyślnie 0)

**Funkcjonalność:**

- Sprawdzenie istnienia powiązanych danych (oceny, frekwencja)
- Opcjonalne wymuszenie usunięcia z kaskadowym usuwaniem powiązań
- Automatyczne usuwanie relacji z rodzicami
- Mechanizm bezpieczeństwa przed przypadkowym usunięciem

**sp_GetStudentById**

**Parametry:**

- @StudentId (INT): ID ucznia

**Zwraca:**

- Podstawowe informacje o uczniu (imię, nazwisko, wiek, płeć)
- Informacje o klasie i roku szkolnym
- Informacje o wychowawcy
- Listę rodziców z danymi kontaktowymi

**sp_GetStudentsByGroup**

**Parametry:**

- @GroupId (INT): ID klasy

**Zwraca:**

- Listę uczniów w klasie z podstawowymi danymi
- Wyliczony wiek na podstawie daty urodzenia
- Statystyki ocen (liczba ocen, średnia)

### 6.1.2 Zarządzanie nauczycielami

**sp_CreateTeacher**

**Parametry wejściowe:**

- @FirstName, @LastName (wymagane)
- @Birthday, @PhoneNumber, @Email, @AdditionalInfo (opcjonalne)

**Funkcjonalność:**

- Walidacja formatu email (pattern @%.%)
- Sprawdzenie unikalności email
- Automatyczne przypisanie ID przez IDENTITY

**sp_GetTeacherDetails**

**Parametry:**

- @TeacherId (INT): ID nauczyciela

**Zwraca trzy zestawy wyników:**

1. **Podstawowe dane nauczyciela** z statystykami (liczba lekcji, klas, przedmiotów)
2. **Klasy pod opieką** (jako wychowawca) z liczbą uczniów
3. **Plan lekcji** z detalami (przedmiot, dzień, godzina, sala, rok)

### 6.1.3 Zarządzanie ocenami

**sp_AddMark**

**Parametry:**

- @SubjectId (INT): ID przedmiotu
- @StudentId (INT): ID ucznia
- @Value (INT): Wartość oceny (1-6)
- @Comment (NVARCHAR(500)): Komentarz (opcjonalny)
- @Weight (INT): Waga oceny (1-10, domyślnie 1)

**Walidacje:**

- Sprawdzenie istnienia przedmiotu i ucznia
- Walidacja zakresu oceny (1-6)
- Walidacja wagi (1-10)

**sp_GetStudentMarks**

**Parametry:**

- @StudentId (INT): ID ucznia
- @SubjectId (INT): ID przedmiotu (opcjonalny - filtr)

**Zwraca dwa zestawy:**

1. **Szczegółowa lista ocen** (przedmiot, wartość, komentarz, waga)
2. **Statystyki per przedmiot** (liczba ocen, średnia zwykła, średnia ważona, min/max)

## 6.2 Operacje rozproszone z PostgreSQL

### 6.2.1 Zarządzanie uwagami

**pg_add_remark**

**Parametry:**

- @studentId (INT): ID ucznia
- @teacherId (INT): ID nauczyciela
- @value (NVARCHAR(MAX)): Treść uwagi

**Mechanizm działania:**

- Dynamiczne tworzenie zapytania INSERT przez OPENQUERY
- Zabezpieczenie przed SQL injection (REPLACE dla apostrofów)
- Wykorzystanie linked server POSTGRES_REMARKS

**pg_delete_remark**

**Parametry:**

- @id (INT): ID uwagi do usunięcia

**Mechanizm działania:**

- Usuwanie przez OPENQUERY z filtrami WHERE
- Bezpośrednie odwołanie do PostgreSQL przez linked server

### 6.2.2 Test integracji PostgreSQL

**Funkcjonalność test.sql:**

1. **Dodawanie uwagi** przez pg_add_remark
2. **Pobieranie ID** nowej uwagi przez OPENQUERY z ORDER BY DESC
3. **Weryfikacja danych** przez SELECT z filtrem WHERE
4. **Usuwanie testowe** przez pg_delete_remark
5. **Weryfikacja usunięcia** przez ponowne SELECT

## 6.3 Statystyki i raporty

### 6.3.1 Statystyki frekwencji

**sp_GetAttendanceStatistics**

**Parametry:**

- @StudentId (INT): Filtr na ucznia (opcjonalny)
- @GroupId (INT): Filtr na klasę (opcjonalny)
- @StartDate, @EndDate (DATE): Zakres czasowy

**Zwraca:**

- Liczbę wszystkich sesji frekwencji
- Liczbę obecności i nieobecności
- Wyliczony procent frekwencji
- Dane per uczeń lub per klasa

### 6.3.2 Statystyki ocen klasowych

**sp_GetClassMarksStatistics**

**Parametry:**

- @GroupId (INT): ID klasy
- @SubjectId (INT): Filtr na przedmiot (opcjonalny)

**Zwraca dwa zestawy:**

1. **Statystyki per uczeń**: średnie, min/max per przedmiot
2. **Statystyki per przedmiot**: średnie klasowe, rozstęp ocen, liczba uczniów z ocenami

## 6.4 Widoki rozproszone

### 6.4.1 vw_StudentCompleteInfo

**Źródła danych:** Wyłącznie SQL Server
**Zawartość:**

- Podstawowe dane ucznia (imię, nazwisko, urodzenie)
- Informacje o klasie i roku szkolnym
- Dane rodziców (imię, nazwisko, kontakt)

**Zastosowanie:**

- Raporty administracyjne
- Widoki dla nauczycieli
- Dane kontaktowe rodziców

### 6.4.2 vw_StudentFinancialInfo

**Źródła danych:** SQL Server + Oracle (ORACLE_FINANCE)
**Zawartość:**

- Podstawowe dane ucznia z SQL Server
- Dane finansowe z Oracle (miesięczna opłata, zapłacone, do zapłaty)
- Kalkulacje zaległości

**Mechanizm:**

- LEFT JOIN z Oracle przez linked server
- Obsługa przypadków bez kontraktu finansowego (ISNULL)
- Agregacja płatności w podzapytaniu Oracle

### 6.4.3 vw_DistributedStudentData

**Źródła danych:** SQL Server + Oracle + PostgreSQL
**Zawartość:**

- Dane ucznia z SQL Server
- Miesięczne opłaty z Oracle
- Liczba uwag z PostgreSQL

**Mechanizm:**

- Triple-JOIN między trzema systemami
- OPENQUERY do PostgreSQL dla agregacji uwag
- Casting typów danych dla kompatybilności

## 6.5 Eksport danych

### 6.5.1 sp_ExportStudentToExcel

**Parametry:**

- @StudentId (INT): ID ucznia do eksportu

**Funkcjonalność:**
Procedura eksportuje dane ucznia do trzech arkuszy Excel:

**Arkusz Student$**

- Podstawowe dane ucznia z tabeli students
- Bezpośrednie OPENROWSET INSERT do Excel

**Arkusz ContractsPayments$**

- Połączone dane kontraktów i płatności z Oracle
- JOIN między CONTRACTS i PAYMENTS przez linked server
- Filtrowanie na studentId

**Arkusz Remarks$**

- Uwagi z PostgreSQL przez OPENQUERY
- Dynamiczne tworzenie zapytania z filtrem na studentId
- Zabezpieczenie przed SQL injection

**Lokalizacja pliku:** C:\excel_exports\StudentData.xlsx

**Wymagania:**

- Konfiguracja linked server EXCEL_DATA
- Uprawnienia zapisu do katalogu eksportu
- Provider Microsoft.ACE.OLEDB.12.0

# 7. Transakcje Rozproszone

## 7.1 Konfiguracja MS DTC

### 7.1.1 Microsoft Distributed Transaction Coordinator

System wykorzystuje MS DTC do koordynacji transakcji rozproszonych między różnymi systemami bazodanowych. Konfiguracja obejmuje:

**Ustawienia bezpieczeństwa:**

- Network DTC Access: włączony
- Allow Inbound: włączony dla akceptowania połączeń przychodzących
- Allow Outbound: włączony dla inicjowania połączeń wychodzących
- No Authentication Required: dla środowiska testowego
- Enable XA Transactions: wsparcie dla transakcji XA

**Konfiguracja SQL Server:**

- SET XACT_ABORT ON: automatyczny rollback przy błędach
- Włączenie RPC dla linked servers
- Odpowiednie uprawnienia dla kont usług

### 7.1.2 Mechanizm Two-Phase Commit

System implementuje protokół 2PC (Two-Phase Commit) dla krytycznych operacji:

**Faza przygotowania (Prepare Phase):**

- Sprawdzenie dostępności wszystkich zasobów
- Walidacja danych w każdym systemie
- Blokowanie odpowiednich rekordów

**Faza zatwierdzenia (Commit Phase):**

- Synchroniczne zatwierdzenie we wszystkich systemach
- Lub rollback we wszystkich systemach w przypadku błędu

## 7.2 Operacje transakcji rozproszonych

### 7.2.1 sp_AddStudentWithFinanceContract

**Cel:** Atomowe dodanie ucznia z jednoczesnym utworzeniem kontraktu finansowego

**Parametry wejściowe:**

- @GroupId (INT): Klasa ucznia
- @FirstName, @LastName: Dane osobowe
- @Birthday, @GenderId: Dodatkowe dane (opcjonalne)
- @ParentId (INT): Identyfikator rodzica
- @ContractStart, @ContractEnd (DATE): Okres obowiązywania kontraktu
- @MonthlyAmount (DECIMAL): Miesięczna opłata

**Parametry wyjściowe:**

- @StudentId (INT OUTPUT): ID utworzonego ucznia
- @OracleContractId (INT OUTPUT): ID utworzonego kontraktu w Oracle

**Przepływ transakcji:**

1. **BEGIN TRANSACTION** (SQL Server)
    - Rozpoczęcie transakcji głównej z SET XACT_ABORT ON
2. **Utworzenie ucznia** (SQL Server)
    - Wywołanie sp_CreateStudent
    - Walidacja danych ucznia
    - Otrzymanie StudentId z OUTPUT parameter
3. **Powiązanie z rodzicem** (SQL Server)
    - INSERT do tabeli parents_students
    - Utworzenie relacji uczeń-rodzic
4. **Utworzenie kontraktu finansowego** (Oracle)
    - Wywołanie sp_CreateContractWithPayments przez linked server
    - Przekazanie parametrów przez RPC
    - Automatyczne utworzenie płatności miesięcznych
5. **COMMIT/ROLLBACK**
    - COMMIT przy powodzeniu wszystkich operacji
    - ROLLBACK przy błędzie w którymkolwiek kroku

**Obsługa błędów:**

- TRY-CATCH block dla przechwytywania błędów
- Automatyczny rollback przy błędach (XACT_ABORT)
- Szczegółowe komunikaty błędów z informacją o źródle

### 7.2.2 sp_ProcessStudentPayment

**Cel:** Atomowe przetwarzanie płatności ucznia z aktualizacją statusów

**Parametry:**

- @StudentId (INT): Identyfikator ucznia
- @PaidAmount (DECIMAL): Kwota wpłaty
- @PaidDate (DATE): Data wpłaty (domyślnie bieżąca)

**Przepływ transakcji:**

1. **BEGIN TRANSACTION** (SQL Server)
    - Rozpoczęcie transakcji rozproszonej
2. **Identyfikacja kontraktu** (Oracle przez linked server)
    - Pobranie najnowszego kontraktu dla ucznia
    - Walidacja istnienia kontraktu
3. **Identyfikacja płatności** (Oracle)
    - Znalezienie najbliższej płatności PENDING
    - Sortowanie według dueDate (najwcześniejsza pierwsza)
4. **Przetwarzanie płatności** (Oracle)
    - Wywołanie sp_ProcessPayment przez RPC
    - Obsługa płatności częściowych i pełnych
    - Aktualizacja statusów (PENDING → PAID)
5. **Aktualizacja podsumowań** (Oracle REMOTE_DB2)
    - Automatyczna aktualizacja payment_summary
    - Przeliczenie łącznych kwot i liczników
6. **COMMIT/ROLLBACK**
    - Sprawdzenie rezultatu z Oracle
    - COMMIT przy sukcesie, ROLLBACK przy błędzie

**Obsługa płatności częściowych:**

- Pierwszy call: zmniejszenie kwoty, status pozostaje PENDING
- Drugi call: dopłata reszty, status zmienia się na PAID
- Tracking progressu płatności

## 7.3 Mechanizmy zapewnienia spójności

### 7.3.1 Walidacja krzyżowa

**Sprawdzanie integralności referencyjnej:**

- Walidacja istnienia ucznia przed utworzeniem kontraktu
- Sprawdzenie istnienia kontraktu przed płatnością
- Weryfikacja powiązań rodzic-uczeń

**Walidacja biznesowa:**

- Sprawdzenie limitów wiekowych uczniów
- Walidacja kwot płatności (dodatnie, w rozsądnych granicach)
- Kontrola dat (contract start < end, payment date <= current date)

### 7.3.2 Obsługa konfliktów

**Deadlock prevention:**

- Konsekwentna kolejność dostępu do zasobów
- Krótkie transakcje dla minimalizacji blokad
- Użycie READ COMMITTED isolation level

**Retry mechanism:**

- Automatyczne ponowienie przy błędach sieciowych
- Exponential backoff dla powtarzanych prób
- Maksymalna liczba prób (configuration-driven)

### 7.3.3 Monitoring transakcji

**Logging operacji:**

- Rejestracja rozpoczęcia i zakończenia transakcji
- Logowanie parametrów wejściowych i wyjściowych
- Czas wykonania operacji rozproszonych

**Alerting:**

- Powiadomienia przy długo trwających transakcjach
- Alerty przy wysokim poziomie rollback
- Monitoring dostępności linked servers

## 7.4 Scenariusze testowe

### 7.4.1 Test sukcesu

**Scenariusz:** Pełne dodanie ucznia z kontraktem

1. Utworzenie ucznia "AnnaTest Testowa"
2. Powiązanie z rodzicem ID=3
3. Utworzenie kontraktu 2025-09-01 do 2026-06-30
4. Miesięczna opłata 1111.50 PLN
5. Weryfikacja danych we wszystkich systemach

### 7.4.2 Test płatności częściowej

**Scenariusz:** Płatność w dwóch ratach

1. Znalezienie płatności PENDING
2. Pierwsza wpłata 40% kwoty
3. Weryfikacja statusu (nadal PENDING, zmniejszona kwota)
4. Druga wpłata pozostałej części
5. Weryfikacja zmiany statusu na PAID

### 7.4.3 Test rollback

**Scenariusz:** Błąd w Oracle po utworzeniu ucznia w SQL Server

1. Utworzenie ucznia w SQL Server (sukces)
2. Błąd przy tworzeniu kontraktu w Oracle
3. Automatyczny rollback całej transakcji
4. Weryfikacja braku ucznia w SQL Server

### 7.4.4 Test concurrent access

**Scenariusz:** Równoczesne operacje na tym samym uczniu

1. Dwie sesje próbujące zaktualizować tego samego ucznia
2. Jedna transakcja powinna zostać wstrzymana
3. Po zakończeniu pierwszej, druga kontynuuje
4. Weryfikacja spójności końcowej danych

# 8. Replikacja Danych

## 8.1 Konfiguracja replikacji transakcyjnej

### 8.1.1 Architektura replikacji

System implementuje replikację transakcyjną w modelu Publisher-Subscriber między dwoma instancjami SQL Server:

**Publisher (Serwer główny):**

- Instance: localhost (port 1433)
- Database: SchoolDB
- Rola: Publikuje zmiany w tabeli students

**Subscriber (Serwer repliki):**

- Instance: 127.0.0.1,1434 (port 1434)
- Database: SchoolDB_Replica
- Rola: Odbiera i aplikuje zmiany

**Distributor:**

- Lokalizacja: Serwer główny (self-hosting)
- Database: distribution
- Working directory: C:\ReplData
- Data folder: C:\MSSQL\Data

### 8.1.2 Publikacja SchoolDB_StudentsOnly

**Charakterystyka publikacji:**

- **Nazwa**: SchoolDB_StudentsOnly
- **Typ**: Replikacja transakcyjna (continuous)
- **Sync method**: concurrent
- **DDL replication**: włączona (replicate_ddl = 1)

**Artykuły publikacji:**

- **Tabela**: students
- **Typ**: logbased (oparty na logu transakcji)
- **Identity range management**: MANUAL
- **Filtrowanie**: brak (cała tabela)

**Uzasadnienie wyboru tabeli students:**

- Kluczowa tabela dla systemu szkolnego
- Częste operacje INSERT (nowi uczniowie)
- Moderate UPDATE operations (zmiany danych osobowych)
- Minimalne operacje DELETE (uczniowie rzadko usuwani)
- Stosunkowo mała tabela (wydajność replikacji)

### 8.1.3 Konfiguracja subskrypcji

**Typ subskrypcji:** Push subscription

- Publisher aktywnie dostarcza zmiany do Subscriber
- Większa kontrola nad procesem replikacji
- Lepszy monitoring z perspektywy Publisher

**Parametry subskrypcji:**

- **Sync type**: automatic (automatyczna synchronizacja początkowa)
- **Update mode**: read only (Subscriber tylko do odczytu)
- **Article**: all (wszystkie artykuły publikacji)

**Authentication:**

- **Subscriber security mode**: SQL Server Authentication (mode 0)
- **Login**: repl_user
- **Password**: Pa55w0rd!

## 8.2 Procesy replikacji

### 8.2.1 Log Reader Agent

**Funkcja:**

- Monitoruje log transakcji bazy SchoolDB
- Identyfikuje zmiany w replikowanych tabelach
- Przenosi transakcje do bazy distribution

**Konfiguracja:**

- **Job name**: DESKTOP-U8K1QHA-SchoolDB-SchoolDB_StudentsOnly-1
- **Frequency**: Continuous monitoring
- **Security**: Integrated authentication

**Monitorowanie:**

- Status job można sprawdzić przez sp_help_jobhistory
- Automatyczne uruchamianie przez sp_start_job
- Logging błędów w system event log

### 8.2.2 Distribution Agent

**Funkcja:**

- Odbiera transakcje z bazy distribution
- Aplikuje zmiany na serwerze Subscriber
- Obsługuje konflikt resolution

**Konfiguracja:**

- **Job name**: DESKTOP-U8K1QHA-SchoolDB-MSSQL_REPLICA-SchoolDB_Replica
- **Frequency**: Na żądanie lub continuous
- **Retry logic**: Automatyczne ponowienie przy błędach

**Parametry wydajności:**

- **Batch size**: Optymalna wielkość batch dla INSERT/UPDATE
- **Commit frequency**: Częstotliwość commit na Subscriber
- **Query timeout**: Limity czasowe dla operacji

### 8.2.3 Snapshot Agent

**Funkcja:**

- Utworzenie początkowej migawki danych
- Inicjalizacja nowych subskrypcji
- Reinicjalizacja przy problemach

**Konfiguracja:**

- **Job name**: Auto-generated podczas dodawania publikacji
- **Schedule**: On demand (ręczne uruchamianie)

- **Snapshot folder**: C:\ReplData

**Proces inicjalizacji:**

1. Utworzenie schematów tabel na Subscriber
2. Bulk copy danych z Publisher do Subscriber
3. Zastosowanie indeksów i constraintów
4. Rozpoczęcie ciągłej replikacji

## 8.3 Monitoring i zarządzanie

### 8.3.1 Monitoring replikacji

**Stored procedures monitorujące:**

**sp_replmonitorhelpsubscription**

- **Parametry**: @publisher, @publisher_db, @publication
- **Zwraca**: Status subskrypcji, opóźnienia, błędy
- **Użycie**: Sprawdzanie zdrowia replikacji

**System views:**

- **syspublications**: Informacje o publikacjach
- **syssubscriptions**: Lista subskrypcji
- **MSrepl_commands**: Oczekujące komendy replikacji

**Job monitoring:**

- **msdb.dbo.sysjobs**: Lista wszystkich job replikacji
- **sp_help_jobhistory**: Historia wykonań job
- **Alert na długo trwające operacje**

### 8.3.2 Zarządzanie konfliktami

**Conflict detection:**

- System automatycznie wykrywa konflikty UPDATE/DELETE
- Logging konfliktów w tabelach systemowych
- Email notifications dla administratorów

**Conflict resolution:**

- **Publisher wins**: Domyślna strategia (Publisher ma priorytet)
- **Custom resolvers**: Możliwość implementacji własnej logiki
- **Manual resolution**: Interwencja administratora przy krytycznych konfliktach

### 8.3.3 Optymalizacja wydajności

**Network optimization:**

- **Compression**: Kompresja danych między Publisher-Subscriber
- **Batch processing**: Grupowanie małych transakcji
- **Off-peak scheduling**: Planowanie dużych operacji poza godzinami szczytu

**Storage optimization:**

- **Distribution database sizing**: Odpowiedni rozmiar bazy distribution
- **Log file management**: Automatyczne czyszczenie starych log
- **Index maintenance**: Regularne przebudowy indeksów

## 8.4 Scenariusze awaryjne

### 8.4.1 Failover do Subscriber

**W przypadku awarii Publisher:**

1. **Przełączenie aplikacji** na SchoolDB_Replica
2. **Zmiana trybu** z read-only na read-write
3. **Aktualizacja connection strings** w aplikacjach

4. **Monitoring integralności** danych na replice

**Przywracanie po awarii:**

1. **Naprawa Publisher** i przywrócenie usługi
2. **Synchronizacja zmian** z okresu awarii
3. **Przełączenie z powrotem** na Publisher
4. **Weryfikacja spójności** danych

### 8.4.2 Reinicjalizacja replikacji

**Sytuacje wymagające reinicjalizacji:**

- Błędy spójności danych
- Długie przerwy w replikacji
- Zmiany schematu tabel
- Problemy z korupcją danych

**Proces reinicjalizacji:**

1. **Zatrzymanie Distribution Agent**
2. **Uruchomienie Snapshot Agent**
3. **Ponowna synchronizacja** całej tabeli
4. **Restart replikacji** od aktualnego momentu

### 8.4.3 Backup i recovery strategy

**Backup Publisher:**

- **Full backup** bazy SchoolDB codziennie
- **Log backup** co 15 minut
- **Distribution backup** codziennie

**Backup Subscriber:**

- **Full backup** SchoolDB_Replica codziennie
- **Verification** spójności z Publisher
- **Point-in-time recovery** capability

## 8.5 Testowanie replikacji

### 8.5.1 Test operacji INSERT

**Scenariusz:**

1. Dodanie nowego ucznia na Publisher
2. Weryfikacja pojawienia się w distribution database
3. Sprawdzenie aplikacji na Subscriber
4. Porównanie danych Publisher vs Subscriber

### 8.5.2 Test operacji UPDATE

**Scenariusz:**

1. Modyfikacja danych ucznia na Publisher
2. Monitoring propagacji przez Log Reader Agent
3. Weryfikacja aktualizacji na Subscriber
4. Sprawdzenie spójności wszystkich pól

### 8.5.3 Test wydajności

**Metryki:**

- **Latency**: Czas propagacji zmian Publisher → Subscriber
- **Throughput**: Liczba transakcji replikowanych na minutę
- **Resource usage**: CPU, memory, I/O na obu serwerach
- **Network bandwidth**: Wykorzystanie pasma między serwerami

# 9. Procedury i Funkcje Oracle

## 9.1 Pakiet pkg_DistributedFinance

### 9.1.1 Specyfikacja pakietu

Pakiet pkg_DistributedFinance stanowi główny interfejs do zarządzania operacjami finansowymi w środowisku rozproszonym Oracle.

**Typ rekordowy t_finance_record**

```
contract_id NUMBER
student_id NUMBER
parent_id NUMBER
monthly_amount NUMBER
total_paid NUMBER
outstanding_balance NUMBER
```

**Typ tabelowy t_finance_table**

Kolekcja rekordów t_finance_record do zwracania zbiorów danych.

### 9.1.2 Funkcje pakietu

**fn_GetStudentFinanceData**

**Parametry:**

- p_student_id (NUMBER): Identyfikator ucznia

**Zwraca:** t_finance_table PIPELINED

**Funkcjonalność:**

- Pobiera wszystkie kontrakty dla danego ucznia
- Łączy dane z tabeli contracts i payments
- Kalkuluje total_paid jako sumę płatności PAID
- Wylicza outstanding_balance na podstawie czasu trwania kontraktu i kwoty miesięcznej
- MONTHS_BETWEEN dla wyliczenia liczby miesięcy
- Zwraca wyniki w formie PIPELINED TABLE FUNCTION

**Algorytm kalkulacji należności:**

```
outstanding_balance = (monthly_amount × MONTHS_BETWEEN(endDate, startDate)) - total_paid
```

**sp_DistributedPaymentProcessing**

**Parametry wejściowe:**

- p_student_id (NUMBER): ID ucznia
- p_payment_amount (NUMBER): Kwota płatności
- p_payment_date (DATE): Data płatności (domyślnie SYSDATE)

**Parametry wyjściowe:**

- p_result (OUT VARCHAR2): Status operacji

**Funkcjonalność:**

1. **Znalezienie kontraktu**: Pobiera najnowszy kontrakt dla ucznia (ORDER BY id DESC)
2. **Identyfikacja płatności**: Znajduje płatność PENDING o odpowiedniej kwocie (ORDER BY dueDate)
3. **Przetwarzanie płatności**: Wywołuje sp_ProcessPayment
4. **Aktualizacja podsumowań**: MERGE w REMOTE_DB2.payment_summary
5. **Commit/Rollback**: Automatyczne zarządzanie transakcją

**MERGE statement dla payment_summary:**

- WHEN MATCHED: Aktualizacja totalAmount, paymentCount, lastPaymentDate
- WHEN NOT MATCHED: INSERT nowego rekordu podsumowania

**sp_GenerateFinanceReport**

**Parametry:**

- p_report_type (VARCHAR2): Typ raportu ('SUMMARY', 'DETAILED', 'OVERDUE')

- p_cursor (OUT SYS_REFCURSOR): Kursor z wynikami

**Rodzaje raportów:**

**SUMMARY:**

- Łączna liczba kontraktów i studentów
- Całkowity miesięczny przychód
- Suma pobranych płatności
- Kwota oczekujących płatności

**DETAILED:**

- Szczegółowy widok z vw_DistributedFinanceData
- Dane z różnych schematów (source_schema)
- Sortowanie według schematu i studentId

**OVERDUE:**

- Lista zaległych płatności (status = PENDING AND dueDate < SYSDATE)
- Kalkulacja dni opóźnienia (TRUNC(SYSDATE - dueDate))
- Sortowanie według daty płatności

## 9.2 Procedury autonomiczne

### 9.2.1 sp_CreateContractWithPayments

**Parametry:**

- p_student_id, p_parent_id (NUMBER): Identyfikatory ucznia i rodzica
- p_start_date, p_end_date (DATE): Okres obowiązywania kontraktu
- p_monthly_amount (NUMBER): Miesięczna opłata
- p_contract_id (OUT NUMBER): ID utworzonego kontraktu

**Algorytm tworzenia płatności:**

1. **INSERT kontraktu** z RETURNING clause dla otrzymania ID
2. **Iteracja miesięczna** od start_date do end_date
3. **TRUNC(..., 'MM')** dla normalizacji dat na pierwszy dzień miesiąca
4. **ADD_MONTHS** dla przechodzenia do kolejnego miesiąca
5. **INSERT płatności** z statusem PENDING dla każdego miesiąca

**Przykład logiki iteracji:**

```
v_current_date := p_start_date;
WHILE v_current_date <= p_end_date LOOP
    v_payment_date := TRUNC(v_current_date, 'MM');
    INSERT INTO payments (contractId, dueDate, amount, status)
    VALUES (p_contract_id, v_payment_date, p_monthly_amount, 'PENDING');
    v_current_date := ADD_MONTHS(v_current_date, 1);
END LOOP;
```

### 9.2.2 sp_ProcessPayment

**Parametry:**

- p_payment_id (NUMBER): ID płatności do przetworzenia
- p_paid_amount (NUMBER): Kwota wpłacona
- p_paid_date (DATE): Data wpłaty (domyślnie SYSDATE)
- p_result (OUT VARCHAR2): Rezultat operacji

**Logika przetwarzania:**

1. **Walidacja**: Sprawdzenie istnienia płatności ze statusem PENDING
2. **Porównanie kwot**: expected_amount vs paid_amount
3. **Płatność częściowa**: Zmniejszenie amount, status pozostaje PENDING
4. **Płatność pełna**: Status → PAID, ustawienie paidDate
5. **Error handling**: EXCEPTION block z ROLLBACK

**Obsługa płatności częściowych:**

```
IF p_paid_amount < v_expected_amt THEN
    UPDATE payments SET amount = v_expected_amt - p_paid_amount
    WHERE id = p_payment_id;
    p_result := 'INFO: Partial payment accepted, remaining: ' || TO_CHAR(v_expected_amt - p_paid_amount);
ELSE
    UPDATE payments SET status = 'PAID', paidDate = p_paid_date
    WHERE id = p_payment_id;
    p_result := 'SUCCESS: Payment fully processed';
END IF;
```

### 9.2.3 sp_CrossSchemaFinanceReport

**Parametry:**

- p_student_id (NUMBER): Filtr na ucznia (opcjonalny)
- p_cursor (OUT SYS_REFCURSOR): Wyniki

**Logika:**

- **IF p_student_id IS NOT NULL**: Szczegółowe dane dla konkretnego ucznia
- **ELSE**: Agregowane dane per source_schema
- **Wykorzystanie vw_DistributedFinanceData** dla unifikacji danych z różnych schematów

### 9.2.4 sp_SyncBetweenSchemas

**Parametry:**

- p_operation (VARCHAR2): Typ operacji ('SYNC_TO_REMOTE', 'SYNC_FROM_REMOTE')

**Operacje synchronizacji:**

**SYNC_TO_REMOTE:**

- INSERT z contracts do REMOTE_DB1.contracts_remote
- WHERE NOT EXISTS dla unikania duplikatów
- Bazuje na kombinacji studentId + parentId

**SYNC_FROM_REMOTE:**

- INSERT z REMOTE_DB1.contracts_remote do contracts
- Odwrotny kierunek synchronizacji
- Identyczna logika unikania duplikatów

**Monitoring:**

- SQL%ROWCOUNT dla liczenia zsynchronizowanych rekordów
- DBMS_OUTPUT dla raportowania wyników
- COMMIT po zakończeniu operacji

## 9.3 Funkcje kalkulacyjne

### 9.3.1 fn_CalculateOutstandingBalance

**Parametry:**

- p_student_id (NUMBER): ID ucznia

**Zwraca:** NUMBER (kwota do zapłaty)

**Algorytm:**

1. **Kalkulacja total_due**: SUM(monthlyAmount × MONTHS_BETWEEN(endDate, startDate))
2. **Kalkulacja total_paid**: SUM płatności ze statusem PAID
3. **Outstanding calculation**: NVL(total_due, 0) - NVL(total_paid, 0)
4. **NVL handling**: Obsługa wartości NULL dla nowych kontraktów

**Użycie MONTHS_BETWEEN:**

- Dokładna kalkulacja miesięcy między datami
- Uwzględnienie części miesięcy (frakcje)
- Automatyczna obsługa lat przestępnych

## 9.4 Widoki rozproszone Oracle

### 9.4.1 vw_DistributedFinanceData

**Źródła danych:**

- **MAIN schema**: contracts + payments (z JOIN)
- **REMOTE1 schema**: REMOTE_DB1.contracts_remote
- **UNION ALL** dla połączenia danych

**Struktura wyników:**

- source_schema ('MAIN' | 'REMOTE1')
- contract_id (z offsetem +1000 dla REMOTE1)
- studentId, parentId, monthlyAmount
- payment_status, payment_amount, paidDate

**Logika REMOTE1:**

- contract_id z offsetem: cr.id + 1000
- Sztywny status: 'PENDING'
- payment_amount = monthlyAmount
- paidDate = NULL

### 9.4.2 Triggery INSTEAD OF

**tr_DistributedFinanceData_Insert**

**Logika:**

- IF :NEW.studentId > 100: INSERT do REMOTE_DB1.contracts_remote
- ELSE: INSERT do głównej tabeli contracts
- Automatyczne ustawienie dat (SYSDATE, ADD_MONTHS(SYSDATE, 12))

**tr_DistributedFinanceData_Update**

**Logika per source_schema:**

- **MAIN**: UPDATE contracts i payments w głównym schemacie
- **REMOTE**: UPDATE REMOTE_DB1.contracts_remote
- Offset handling: (:OLD.contract_id - 1000) dla REMOTE
- Automatyczne ustawienie paidDate przy zmianie statusu na PAID

## 9.5 Testowanie pakietu

### 9.5.1 Test kompleksowy

**Scenariusz testu (test.sql):**

1. **Tworzenie kontraktu testowego** (studentId=999, parentId=888)
2. **Przetwarzanie płatności** przez pkg_DistributedFinance.sp_DistributedPaymentProcessing
3. **Wywołanie funkcji finansowej** pkg_DistributedFinance.fn_GetStudentFinanceData
4. **Generowanie raportu** pkg_DistributedFinance.sp_GenerateFinanceReport('SUMMARY')
5. **Weryfikacja remote summary** w REMOTE_DB2.payment_summary
6. **Cleanup**: Usunięcie danych testowych

**DBMS_OUTPUT dla monitoringu:**

- Każdy krok z komunikatami tekstowymi
- Wyświetlanie wyników funkcji i procedur
- Błędy w EXCEPTION block z SQLERRM

# 10. Raporty i Analizy Rozproszone

## 10.1 Procedury raportowe SQL Server

### 10.1.1 sp_DistributedStudentReport

**Cel:** Kompleksowy raport o uczniach z danymi ze wszystkich trzech systemów

**Parametry:**

- @StartDate (DATE): Data początkowa dla analiz frekwencji (opcjonalna)
- @EndDate (DATE): Data końcowa dla analiz frekwencji (opcjonalna)

**Źródła danych:**

1. **SQL Server**: Podstawowe dane ucznia, frekwencja
2. **Oracle**: Dane finansowe (miesięczne opłaty, płatności)
3. **PostgreSQL**: Liczba uwag

**Struktura wyniku:**

- StudentName: Pełne imię i nazwisko
- TotalFinancialDue: Roczna kwota do zapłaty
- TotalFinancialPaid: Suma wpłaconych płatności
- TotalRemarks: Liczba uwag z PostgreSQL
- AttendanceRate: Procent frekwencji z okresu

**Implementacja JOIN rozproszonych:**

```
LEFT JOIN (Oracle financial data) finance ON s.id = finance.studentId
LEFT JOIN (PostgreSQL OPENQUERY remarks) remarks ON s.id = remarks.studentId
LEFT JOIN (Local attendance data) attendance ON s.id = attendance.studentId
```

**Kalkulacje lokalne:**

- AttendanceRate: AVG(CAST(present AS FLOAT)) * 100
- Filtrowanie według zakresu dat: WHERE a.dateTimeChecked BETWEEN @StartDate AND @EndDate

### 10.1.2 sp_AggregatedReport

**Cel:** Raport agregacyjny z wszystkich trzech systemów

**Sekcje raportu:**

**Local Student Statistics**

- Statystyki per grupa: liczba uczniów, średni wiek
- GROUP BY g.id z COUNT(s.id) i AVG(DATEDIFF(YEAR, s.birthday, GETDATE()))

**Remote Financial Statistics**

- Zapytanie OPENQUERY do Oracle z agregacjami:
    - COUNT(c.id): Łączna liczba kontraktów
    - AVG(c.monthlyAmount): Średnia miesięczna opłata
    - SUM płatności PAID: Łączna kwota wpłacona
    - COUNT płatności PENDING: Liczba oczekujących płatności

**Remote Remarks Statistics**

- Zapytanie OPENQUERY do PostgreSQL:
    - COUNT(*): Łączna liczba uwag
    - COUNT(DISTINCT studentId): Uczniowie z uwagami
    - COUNT(DISTINCT teacherId): Nauczyciele wystawiający uwagi

### 10.1.3 sp_GetCompleteStudentInfo

**Cel:** Pełny profil pojedynczego ucznia ze wszystkich systemów

**Parametry:**

- @StudentId (INT): Identyfikator ucznia

**Wyniki (trzy zestawy):**

**Podstawowe dane ucznia**

- Źródło: SQL Server (students, genders, groups, years)
- JOIN dla pełnych informacji o klasie i roku szkolnym

**Dane finansowe**

- Źródło: Oracle przez linked server
- Miesięczna opłata, daty kontraktu, statystyki płatności
- Obsługa błędów: TRY-CATCH z komunikatem 'Financial data unavailable'

**Uwagi pedagogiczne**

- Źródło: PostgreSQL przez OPENQUERY
- Lista uwag z identyfikatorami nauczycieli i datami
- Obsługa błędów: TRY-CATCH z szczegółami błędu (ERROR_NUMBER, ERROR_MESSAGE)

**Mechanizm failover:**

- Każda sekcja w osobnym TRY-CATCH
- Kontynuacja działania mimo błędów połączenia
- Informacyjne komunikaty o niedostępności danych

## 10.2 Zapytania wieloźródłowe

### 10.2.1 Multi-source CTE query

**Struktura zapytania:**

```
WITH StudentBasic AS (dane z SQL Server),
     OracleFinance AS (dane z Oracle przez linked server),
     PostgresRemarks AS (dane z PostgreSQL przez OPENQUERY)
SELECT ... FROM StudentBasic sb
LEFT JOIN OracleFinance orf ON sb.id = orf.studentId
LEFT JOIN PostgresRemarks pr ON sb.id = pr.studentId
```

**Optymalizacje:**

- CTE dla organizacji kodu i czytelnością
- LEFT JOIN dla zachowania wszystkich uczniów
- ISNULL dla obsługi brakujących danych finansowych

### 10.2.2 Oracle financial aggregation

**Zapytanie oracle_paid.sql:**

- OPENQUERY z kompleksowym GROUP BY w Oracle
- Agregacje wykonywane zdalnie (COUNT, SUM, CASE WHEN)
- Minimalizacja transferu danych przez sieć

### 10.2.3 PostgreSQL remarks aggregation

**Zapytanie postgres_remarks.sql:**

- OPENQUERY z GROUP BY studentId, teacherId
- Kalkulacje MAX(created_date) dla ostatniej uwagi
- ORDER BY dla sortowania wyników na serwerze PostgreSQL

## 10.3 Widoki rozproszone SQL Server

### 10.3.1 vw_DistributedStudentData

**Cel:** Zunifikowany widok danych ucznia z trzech systemów

**Źródła i mechanizmy:**

- **SQL Server**: students (podstawowe dane)
- **Oracle**: ORACLE_FINANCE..FINANCE_DB.CONTRACTS (linked server)
- **PostgreSQL**: OPENQUERY z agregacją COUNT(*) uwag

**Wyzwania integracyjne:**

- **Casting typów**: CAST dla zapewnienia kompatybilności między systemami
- **Obsługa NULL**: LEFT JOIN z CAST(oracle_data.monthlyAmount AS DECIMAL(10,2))
- **Aliasy**: Czytelne nazwy kolumn (StudentId, StudentName, MonthlyAmount)

### 10.3.2 vw_StudentFinancialInfo

**Cel:** Widok finansowy z kalkulacjami należności

**Kalkulacje rozproszone:**

- **total_paid**: Suma płatności PAID grupowana w podzapytaniu Oracle
- **pending_amount**: Kalkulacja (monthly_amount * 12 - total_paid)
- **ISNULL handling**: Obsługa uczniów bez kontraktów finansowych

**Struktura podzapytania Oracle:**

```
LEFT JOIN (
    SELECT contractId,
           SUM(CASE WHEN status = 'PAID' THEN amount ELSE 0 END) as totalPaid
    FROM ORACLE_FINANCE..FINANCE_DB.PAYMENTS
    GROUP BY contractId
) p ON c.id = p.contractId
```

## 10.4 Procedury analityczne Oracle

### 10.4.1 Package reporting procedures

**sp_GenerateFinanceReport z pakietu pkg_DistributedFinance:**

**Report type: 'SUMMARY'**

- Agregacje na poziomie całego systemu finansowego
- COUNT(DISTINCT) dla unikalnych kontraktów i studentów
- SUM z CASE WHEN dla podziału na zapłacone/oczekujące

**Report type: 'DETAILED'**

- Wykorzystanie vw_DistributedFinanceData
- Dane z różnych schematów z oznaczeniem source_schema
- ORDER BY source_schema, studentId dla logicznego sortowania

**Report type: 'OVERDUE'**

- Filtrowanie płatności: WHERE p.status = 'PENDING' AND p.dueDate < SYSDATE
- Kalkulacja opóźnienia: TRUNC(SYSDATE - p.dueDate) as days_overdue
- JOIN z contracts dla danych ucznia i rodzica

### 10.4.2 Cross-schema reporting

**sp_CrossSchemaFinanceReport:**

- Warunkowy kursor: IF p_student_id IS NOT NULL vs ELSE
- **Individual report**: Szczegółowe dane dla konkretnego ucznia
- **Aggregate report**: Podsumowania per source_schema
- Wykorzystanie vw_DistributedFinanceData jako źródła ujednoliconego

## 10.5 Eksport i integracja danych

### 10.5.1 Excel export functionality

**sp_ExportStudentToExcel - architektura:**

**Arkusz Student$**

- Bezpośredni eksport z SQL Server przez OPENROWSET
- Provider: Microsoft.ACE.OLEDB.12.0 z opcją HDR=YES

**Arkusz ContractsPayments$**

- JOIN między Oracle contracts i payments
- Linked server ORACLE_FINANCE z four-part naming
- Wszystkie pola finansowe w jednym arkuszu

**Arkusz Remarks$**

- OPENQUERY do PostgreSQL z filtrowaniem na studentId
- Dynamiczne budowanie zapytania z parametrem
- REPLACE dla zabezpieczenia przed SQL injection

**Wymagania środowiskowe:**

- ACE OLEDB provider dla plików Office
- Katalog eksportu z uprawnieniami zapisu
- Linked server EXCEL_DATA z odpowiednią konfiguracją

### 10.5.2 Data transformation

**Mapowanie typów danych:**

- SQL Server NVARCHAR → Excel Text
- Oracle NUMBER → Excel Number
- PostgreSQL TIMESTAMP → Excel DateTime
- Automatyczne wykrywanie nagłówków (HDR=YES)

**Obsługa błędów eksportu:**

- TRY-CATCH dla każdego arkusza osobno
- Continuation przy błędach pojedynczych arkuszy
- Logging błędów z informacją o źródle danych

## 10.6 Monitorowanie wydajności raportów

### 10.6.1 Query optimization strategies

**Przetwarzanie zdalne:**

- Maksymalizacja obliczeń na zdalnych serwerach
- GROUP BY i agregacje w OPENQUERY
- Filtrowanie WHERE przekazywane do systemów źródłowych

**Przetwarzanie lokalne:**

- JOIN wyników z różnych systemów
- Formatowanie i prezentacja danych
- Kalkulacje wymagające danych z wielu źródeł

### 10.6.2 Performance monitoring

**Metryki wydajności:**

- Czas wykonania per system źródłowy
- Volume transferowanych danych
- Resource utilization na każdym serwerze
- Network latency między systemami

**Optymalizacja zapytań:**

- Indeksy na kluczach łączących (studentId, teacherId)
- Partycjonowanie dużych tabel według dat
- Materialized views dla często używanych agregacji

# 11. Testowanie i Weryfikacja

## 11.1 Testy integracji PostgreSQL

### 11.1.1 Test CRUD operacji uwag

**Lokalizacja:** mssql\basic_crud\teachers\postgres\test.sql

**Scenariusz testowy:**

1. **Dodanie uwagi testowej**
    - Wywołanie pg_add_remark z parametrami (studentId=1, teacherId=2)
    - Treść uwagi: "To jest testowy remark"
    - Mechanizm: OPENQUERY INSERT przez linked server

2. **Pobranie ID nowej uwagi**
    - Wykorzystanie tymczasowej tabeli (#tmp_id)
    - OPENQUERY z ORDER BY id DESC dla najnowszej uwagi
    - Dynamic SQL dla budowania zapytania z ID
3. **Weryfikacja poprawności danych**
    - SELECT przez OPENQUERY z filtrem WHERE id = @newId
    - Sprawdzenie czy uwaga została zapisana z poprawnymi danymi
4. **Usunięcie testowej uwagi**
    - Wywołanie pg_delete_remark z pobranym ID
    - Czyszczenie danych testowych
5. **Weryfikacja usunięcia**
    - Ponowne SELECT z tym samym ID
    - Potwierdzenie braku rekordu

**Mechanizmy techniczne:**

- **Dynamic SQL**: Budowanie zapytań z parametrami w runtime
- **Temporary tables**: #tmp_id dla przechowania ID między zapytaniami
- **OPENQUERY**: Wszystkie operacje przez linked server
- **Error handling**: Implicit przez linked server connection

### 11.1.2 Test odporności na błędy

**Scenariusze testowe:**

- **Nieprawidłowe ID**: Próba usunięcia nieistniejącej uwagi
- **Błędy połączenia**: Symulacja niedostępności PostgreSQL
- **SQL injection**: Testowanie zabezpieczeń w pg_add_remark
- **Encoding issues**: Test polskich znaków w uwagach

## 11.2 Testy pakietu Oracle

### 11.2.1 Test kompleksowy pkg_DistributedFinance

**Lokalizacja:** oracle\finance\package_distributed_finance\test.sql

**Dane testowe:**

- studentId: 999 (unikalny dla testu)
- parentId: 888
- Okres kontraktu: 2025-05-07 do 2025-09-04
- Miesięczna opłata: 1000

**Sekwencja testowa:**

**1. Utworzenie kontraktu testowego**

```
sp_CreateContractWithPayments(
    p_student_id => 999,
    p_parent_id => 888,
    p_start_date => TO_DATE('2025-05-07', 'YYYY-MM-DD'),
    p_end_date => TO_DATE('2025-09-04', 'YYYY-MM-DD'),
    p_monthly_amount => 1000,
    p_contract_id => v_contract_id
);
```

**2. Przetwarzanie płatności**

```
pkg_DistributedFinance.sp_DistributedPaymentProcessing(
    p_student_id => 999,
    p_payment_amount => 1000,
    p_result => v_result
);
```

**3. Test funkcji finansowej**

```
FOR rec IN (
    SELECT * FROM TABLE(pkg_DistributedFinance.fn_GetStudentFinanceData(999))
) LOOP
    -- Weryfikacja contract_id, total_paid, outstanding_balance
END LOOP;
```

**4. Generowanie raportu**

```
pkg_DistributedFinance.sp_GenerateFinanceReport('SUMMARY', v_cursor);
FETCH v_cursor INTO variables;
-- Weryfikacja: total_contracts, total_students, total_revenue
```

**5. Weryfikacja remote summary**

```
SELECT * FROM REMOTE_DB2.payment_summary WHERE contractId = v_contract_id;
-- Sprawdzenie totalAmount, paymentCount, lastPaymentDate
```

**6. Cleanup**

```
DELETE FROM payments WHERE contractId = v_contract_id;
DELETE FROM contracts WHERE id = v_contract_id;
DELETE FROM REMOTE_DB2.payment_summary WHERE contractId = v_contract_id;
COMMIT;
```

**Monitoring przez DBMS_OUTPUT:**

- Każdy krok z komunikatami statusu
- Wyświetlanie wyników funkcji i procedur
- Error handling w EXCEPTION block

### 11.2.2 Test scenariuszy płatności

**Test płatności częściowej:**

1. Utworzenie kontraktu z opłatą 1000
2. Pierwsza wpłata 400 (40%)
3. Weryfikacja: status PENDING, amount = 600
4. Druga wpłata 600 (pozostałe 60%)
5. Weryfikacja: status PAID, paidDate ustawione

**Test nadpłaty:**

1. Płatność większa niż wymagana kwota
2. Weryfikacja: status PAID, nadwyżka obsłużona
3. Sprawdzenie komunikatu zwrotnego

## 11.3 Testy transakcji rozproszonych

### 11.3.1 Test sp_AddStudentWithFinanceContract

**Scenariusz sukcesu:**

1. **Przygotowanie parametrów testowych**
    - Dane ucznia: AnnaTest Testowa, urodzenie 2012-05-20
    - Klasa: groupId = 1, płeć: genderId = 2
    - Rodzic: parentId = 3
    - Kontrakt: 2025-09-01 do 2026-06-30, 1111.50 PLN/miesiąc
2. **Wykonanie transakcji rozproszonej**
    - Wywołanie sp_AddStudentWithFinanceContract
    - Automatyczne tworzenie ucznia w SQL Server
    - Automatyczne tworzenie kontraktu w Oracle
3. **Weryfikacja wyników**

```
SELECT * FROM students WHERE id = @StudentId;
SELECT * FROM parents_students WHERE studentId = @StudentId;
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.CONTRACTS WHERE id = @OracleContractId;
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.PAYMENTS WHERE contractId = @OracleContractId;
```

4. **Sprawdzenie integralności**
    - Zgodność studentId między systemami
    - Prawidłowa liczba płatności miesięcznych w Oracle
    - Poprawne daty i kwoty

**Scenariusz błędu:**

1. **Symulacja błędu Oracle**
    - Nieprawidłowe parametry dla kontraktu
    - Lub błąd połączenia z Oracle
2. **Weryfikacja rollback**
    - Sprawdzenie braku ucznia w SQL Server
    - Brak rekordów w tabeli parents_students
    - Potwierdzenie atomowości transakcji

### 11.3.2 Test sp_ProcessStudentPayment

**Test wieloetapowy:**

1. **Identyfikacja płatności testowej**

```
SELECT TOP 1 @PaymentId = id, @ContractId = contractId, @FullAmount = amount
FROM ORACLE_FINANCE..FINANCE_DB.PAYMENTS
WHERE status = 'PENDING'
ORDER BY dueDate;
```

2. **Płatność częściowa (40%)**

```
EXEC sp_ProcessStudentPayment
    @StudentId = @TestStudentId,
    @PaidAmount = @PartialAmount,
    @PaidDate = @Today;
```

3. **Weryfikacja stanu pośredniego**

```
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.PAYMENTS WHERE id = @PaymentId;
-- Expected: status = 'PENDING', amount = reduced
```

4. **Dopłata reszty (60%)**

```
EXEC sp_ProcessStudentPayment
    @StudentId = @TestStudentId,
    @PaidAmount = @RemainAmount,
    @PaidDate = @Today;
```

5. **Weryfikacja końcowa**

```
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.PAYMENTS WHERE id = @PaymentId;
-- Expected: status = 'PAID', paidDate = @Today
```

## 11.4 Testy wydajnościowe

### 11.4.1 Test obciążenia linked servers

**Scenariusze testowe:**

- **Concurrent connections**: Jednoczesne zapytania do Oracle i PostgreSQL
- **Large result sets**: OPENQUERY z dużymi zbiorami danych
- **Connection pooling**: Test ponownego wykorzystania połączeń
- **Timeout handling**: Test zachowania przy długich zapytaniach

**Metryki:**

- Response time per linked server
- Throughput (queries per second)
- Resource utilization (CPU, memory)
- Network bandwidth usage

### 11.4.2 Test replikacji

**Test initial sync:**

1. **Snapshot generation**: Czas utworzenia migawki
2. **Data transfer**: Transfer initial data set
3. **Apply snapshot**: Czas aplikacji na Subscriber
4. **Replication startup**: Uruchomienie continuous replication

**Test ongoing replication:**

1. **Insert operations**: Batch INSERT na Publisher
2. **Latency measurement**: Czas pojawienia się na Subscriber
3. **Update operations**: Mass UPDATE operations
4. **Consistency check**: Porównanie Publisher vs Subscriber

## 11.5 Testy integralności danych

### 11.5.1 Cross-system referential integrity

**Test związków:**

- studentId w Oracle contracts → students w SQL Server
- teacherId w PostgreSQL remarks → teachers w SQL Server
- parentId w Oracle contracts → parents w SQL Server

**Procedury weryfikacyjne:**

```
-- Orphaned contracts (brak ucznia w SQL Server)
SELECT c.studentId FROM ORACLE_FINANCE..FINANCE_DB.CONTRACTS c
WHERE c.studentId NOT IN (SELECT id FROM students);

-- Orphaned remarks (brak nauczyciela w SQL Server)
SELECT DISTINCT teacherId FROM OPENQUERY(POSTGRES_REMARKS,
    'SELECT teacherId FROM remarks_main.remark')
WHERE teacherId NOT IN (SELECT id FROM teachers);
```

### 11.5.2 Data consistency tests

**Financial consistency:**

- Suma płatności ≤ łączna wartość kontraktu
- Daty płatności w zakresie obowiązywania kontraktu
- Status płatności zgodny z paidDate (NULL dla PENDING)

**Temporal consistency:**

- created_date w PostgreSQL ≤ CURRENT_TIMESTAMP
- Contract start_date < end_date
- Student birthday > 1900-01-01 AND < CURRENT_DATE

## 11.6 Testy recovery i failover

### 11.6.1 Linked server failure simulation

**Scenariusze:**

1. **Oracle downtime**: Niedostępność serwera Oracle
2. **PostgreSQL disconnect**: Błąd połączenia ODBC
3. **Network partition**: Symulacja błędów sieciowych

**Handling verification:**

- Graceful degradation w widokach rozproszonych
- Error messages w procedurach
- Rollback w transakcjach rozproszonych

### 11.6.2 Replication failure recovery

**Test scenarios:**

1. **Subscriber downtime**: Awaria serwera repliki
2. **Network interruption**: Przerwa w komunikacji
3. **Distribution database corruption**: Uszkodzenie bazy dystrybucji

**Recovery procedures:**

- Automatic retry mechanisms
- Manual reinitializations
- Data consistency verification post-recovery

# 12. Instrukcja Instalacji i Uruchomienia

## 12.1 Wymagania systemowe

### 12.1.1 Microsoft SQL Server

- **Wersja**: SQL Server 2019 lub nowszy
- **Edycja**: Standard/Enterprise (wsparcie dla linked servers)
- **Komponenty wymagane**:
    - Database Engine Services
    - SQL Server Replication
    - Management Tools (SSMS)
    - MS DTC (Distributed Transaction Coordinator)

**Konfiguracja instancji:**

- Główna instancja: domyślny port 1433
- Replika: port 1434 (dla testów replikacji)
- Mixed Mode Authentication
- sa account włączony z silnym hasłem

### 12.1.2 Oracle Database

- **Wersja**: Oracle 19c lub nowszy
- **Edycja**: Standard/Enterprise
- **Komponenty**:
    - Oracle Database Server
    - Oracle Net Services
    - Oracle Universal Installer

**Konfiguracja:**

- Service Name: PD19C
- Port: 1521
- Character Set: AL32UTF8
- National Character Set: AL16UTF16

### 12.1.3 PostgreSQL

- **Wersja**: PostgreSQL 12 lub nowszy
- **Port**: 5432 (domyślny)
- **Encoding**: UTF8
- **Locale**: pl_PL lub en_US

**ODBC Driver:**

- PostgreSQL ODBC Driver (psqlODBC)
- Wersja 32-bit lub 64-bit zgodna z SQL Server
- DSN Name: PostgreSQL30

### 12.1.4 Dodatkowe komponenty

- **Microsoft ACE OLEDB Provider** (dla eksportu Excel)
- **Oracle OLEDB Provider** (dla linked servers)

- **ODBC Driver Manager**

## 12.2 Kolejność instalacji

### 12.2.1 Etap 1: Przygotowanie Oracle

**1. Utworzenie użytkowników**

```
sqlplus sys/password@PD19C as sysdba
@database_setups/oracle/1_user.sql
```

**Weryfikacja:**

```
SELECT username, default_tablespace, account_status
FROM dba_users
WHERE username IN ('FINANCE_DB', 'REMOTE_DB1', 'REMOTE_DB2');
```

**2. Konfiguracja schematów finansowych**

```
sqlplus FINANCE_DB/Finance123@PD19C
@database_setups/oracle/2_finance_create.sql
```

**Weryfikacja:**

```
SELECT table_name FROM user_tables ORDER BY table_name;
SELECT sequence_name FROM user_sequences;
```

**3. Konfiguracja zdalnych schematów**

```
sqlplus REMOTE_DB1/Remote123@PD19C
@database_setups/oracle/3_remote_db1.sql

sqlplus REMOTE_DB2/Remote123@PD19C
@database_setups/oracle/4_remote_db2.sql
```

**4. Utworzenie synonimów i uprawnień**

```
sqlplus FINANCE_DB/Finance123@PD19C
@database_setups/oracle/5_finance_synonyms.sql
```

**Weryfikacja:**

```
SELECT synonym_name, table_name FROM user_synonyms;
```

### 12.2.2 Etap 2: Przygotowanie PostgreSQL

**1. Utworzenie bazy i schematu**

```
psql -U postgres
@database_setups/postgres/create.sql
```

**Weryfikacja:**

```
\c remarks_system
\dt remarks_main.*
\du remarks_user
```

**2. Konfiguracja ODBC DSN**

**Windows (przez Control Panel):**

1. Administrative Tools → ODBC Data Sources (32-bit/64-bit)
2. System DSN → Add → PostgreSQL Unicode
3. Parametry:
    - Data Source: PostgreSQL30
    - Server: localhost
    - Port: 5432
    - Database: remarks_system
    - User Name: remarks_user

- Password: Remarks123

**Test połączenia:**

```
sqlcmd -Q "SELECT * FROM OPENROWSET('MSDASQL', 'DSN=PostgreSQL30;UID=remarks_user;PWD=Remarks123', 'SELECT version()')"
```

### 12.2.3 Etap 3: Konfiguracja SQL Server

**1. Utworzenie głównej bazy danych**

```
sqlcmd -S localhost -E -i database_setups/main_mssql/create.sql
sqlcmd -S localhost -E -d SchoolDB -i database_setups/main_mssql/data.sql
```

**Weryfikacja:**

```
USE SchoolDB;
SELECT COUNT(*) as TableCount FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE';
SELECT COUNT(*) as StudentCount FROM students;
```

**2. Konfiguracja zaawansowana**

```
sqlcmd -S localhost -E -i linked_servers/config.sql
```

**Weryfikacja:**

```
SELECT name, value_in_use FROM sys.configurations
WHERE name IN ('show advanced options', 'Ad Hoc Distributed Queries');
```

**3. Konfiguracja linked servers**

**Oracle:**

```
sqlcmd -S localhost -E -i linked_servers/oracle.sql
```

**PostgreSQL:**

```
sqlcmd -S localhost -E -i linked_servers/postgres.sql
```

**Excel:**

```
-- Utworzenie katalogu eksportu
mkdir C:\excel_exports
sqlcmd -S localhost -E -i linked_servers/excel.sql
```

**Replikacja (opcjonalna):**

```
sqlcmd -S localhost -E -i linked_servers/2msql.sql
```

**Weryfikacja linked servers:**

```
SELECT srvname, srvproduct, provider, datasource FROM sys.servers WHERE srvid > 0;
```

### 12.2.4 Etap 4: Wypełnienie danymi

**1. Dane Oracle**

```
sqlplus FINANCE_DB/Finance123@PD19C
@database_setups/oracle/6_finance_data.sql

sqlplus REMOTE_DB1/Remote123@PD19C
@database_setups/oracle/7_remote_db1_data.sql

sqlplus REMOTE_DB2/Remote123@PD19C
@database_setups/oracle/8_remote_db2_data.sql
```

**2. Dane PostgreSQL**

```
psql -U remarks_user -d remarks_system
@database_setups/postgres/data.sql
```

**Weryfikacja danych:**

```
-- SQL Server
SELECT COUNT(*) FROM students;

-- Oracle
SELECT COUNT(*) FROM ORACLE_FINANCE..FINANCE_DB.CONTRACTS;

-- PostgreSQL
SELECT COUNT(*) FROM OPENQUERY(POSTGRES_REMARKS, 'SELECT COUNT(*) FROM remarks_main.remark');
```

## 12.3 Konfiguracja replikacji (opcjonalna)

### 12.3.1 Przygotowanie drugiej instancji SQL Server

**Instalacja replikujące instancji:**

1. Instalacja SQL Server na porcie 1434
2. Konfiguracja Database Engine
3. Utworzenie bazy SchoolDB_Replica

### 12.3.2 Konfiguracja replikacji transakcyjnej

```
sqlcmd -S localhost -E -i mssql/replikacja.sql
```

**Kroki konfiguracji:**

1. Konfiguracja distributora
2. Utworzenie publikacji SchoolDB_StudentsOnly
3. Konfiguracja subskrypcji Push
4. Uruchomienie agentów replikacji

**Weryfikacja replikacji:**

```
-- Status publikacji
SELECT name, status FROM syspublications;

-- Status subskrypcji
EXEC sp_replmonitorhelpsubscription;
```

## 12.4 Instalacja funkcjonalności systemowych

### 12.4.1 Procedury CRUD SQL Server

```
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/students/create_student.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/students/update_student.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/students/delete_student.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/students/get_student_by_id.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/students/get_students_by_group.sql
```

**Test procedur:**

```
DECLARE @StudentId INT;
EXEC sp_CreateStudent @GroupId=1, @FirstName='Test', @LastName='Student', @StudentId=@StudentId OUTPUT;
EXEC sp_GetStudentById @StudentId=@StudentId;
```

### 12.4.2 Procedury dla nauczycieli

```
-- Instalacja procedur nauczycielskich
for %%f in (mssql/basic_crud/teachers/*.sql) do sqlcmd -S localhost -E -d SchoolDB -i "%%f"

-- Procedury PostgreSQL
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/teachers/postgres/add_remark.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/basic_crud/teachers/postgres/delete_remark.sql
```

### 12.4.3 Procedury Oracle

```
sqlplus FINANCE_DB/Finance123@PD19C
@oracle/finance/procedures/create_contract_with_payment.sql
@oracle/finance/procedures/process_payment.sql
@oracle/finance/procedures/get_contract_info.sql
@oracle/finance/procedures/distributed_finance_report.sql
@oracle/finance/procedures/sync_data.sql
```

### 12.4.4 Pakiet rozproszony Oracle

```
sqlplus FINANCE_DB/Finance123@PD19C
@oracle/finance/package_distributed_finance/head.sql
@oracle/finance/package_distributed_finance/body.sql
```

**Weryfikacja pakietu:**

```
SELECT object_name, object_type, status FROM user_objects
WHERE object_name = 'PKG_DISTRIBUTEDFINANCE';
```

### 12.4.5 Funkcje i widoki

```
-- SQL Server widoki
sqlcmd -S localhost -E -d SchoolDB -i mssql/views/students_complete_info.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/views/student_financial_info.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/views/distributed_student_data.sql

-- Oracle widoki i funkcje
sqlplus FINANCE_DB/Finance123@PD19C
@oracle/finance/functions/calculate_outstanding_balance.sql
@oracle/finance/views/sim_distributed_finance_data.sql
```

### 12.4.6 Transakcje rozproszone

```
sqlcmd -S localhost -E -d SchoolDB -i mssql/transactions.sql/add_student_with_contract.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/transactions.sql/process_student_payment.sql
```

### 12.4.7 Procedury raportowe

```
sqlcmd -S localhost -E -d SchoolDB -i mssql/stored_procedures/distributed_student_report.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/stored_procedures/distributed_aggregated_report.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/stored_procedures/get_complete_info.sql
sqlcmd -S localhost -E -d SchoolDB -i mssql/excel_export/export.sql
```

## 12.5 Testowanie instalacji

### 12.5.1 Test połączeń międzysystemowych

```
-- Test Oracle
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.CONTRACTS WHERE ROWNUM <= 5;

-- Test PostgreSQL
SELECT * FROM OPENQUERY(POSTGRES_REMARKS, 'SELECT * FROM remarks_main.remark LIMIT 5');

-- Test widoku rozproszonego
SELECT TOP 5 * FROM vw_DistributedStudentData;
```

### 12.5.2 Test transakcji rozproszonych

```
-- Test dodania ucznia z kontraktem
DECLARE @StudentId INT, @OracleContractId INT;
EXEC sp_AddStudentWithFinanceContract
    @GroupId=1, @FirstName='TestInstall', @LastName='User',
    @ParentId=1, @ContractStart='2025-09-01', @ContractEnd='2026-06-30',
    @MonthlyAmount=500.00,
    @StudentId=@StudentId OUTPUT, @OracleContractId=@OracleContractId OUTPUT;

-- Weryfikacja wyników
SELECT * FROM students WHERE id = @StudentId;
SELECT * FROM ORACLE_FINANCE..FINANCE_DB.CONTRACTS WHERE id = @OracleContractId;
```

### 12.5.3 Test procedur PostgreSQL

```
EXEC mssql/basic_crud/teachers/postgres/test.sql
```

### 12.5.4 Test pakietu Oracle

```
sqlplus FINANCE_DB/Finance123@PD19C
@oracle/finance/package_distributed_finance/test.sql
```

### 12.5.5 Test replikacji

```
-- Dodanie ucznia na Publisher
INSERT INTO students (groupId, firstName, lastName, genderId)
VALUES (1, 'ReplicationTest', 'Student', 1);

-- Sprawdzenie na Subscriber (po chwili)
SELECT * FROM [MSSQL_REPLICA].[SchoolDB_Replica].[dbo].[students]
WHERE firstName = 'ReplicationTest';
```

## 12.6 Rozwiązywanie problemów

### 12.6.1 Problemy z linked servers

**Błąd: Login failed for user**

```
-- Sprawdzenie konfiguracji
SELECT * FROM sys.linked_logins WHERE server_id =
    (SELECT server_id FROM sys.servers WHERE name = 'ORACLE_FINANCE');

-- Rekonfiguracja login mapping
EXEC sp_addlinkedsrvlogin 'ORACLE_FINANCE', 'false', NULL, 'FINANCE_DB', 'Finance123';
```

**Błąd: Provider cannot be found**

```
-- Sprawdzenie dostępnych providerów
SELECT * FROM sys.oledb_providers;

-- Instalacja Oracle OLEDB provider (jako administrator)
regsvr32 "C:\Oracle\instantclient\OraOLEDB19.dll"
```

### 12.6.2 Problemy z replikacją

**Błąd: Could not find stored procedure**

```
-- Sprawdzenie statusu agentów
SELECT job_id, name, enabled FROM msdb.dbo.sysjobs
WHERE name LIKE '%repl%';

-- Restart agentów
EXEC msdb.dbo.sp_start_job @job_name = 'Distribution Agent JobName';
```

### 12.6.3 Problemy z Oracle

**Błąd: ORA-00942: table or view does not exist**

```
-- Sprawdzenie uprawnień
SELECT grantee, table_name, privilege FROM user_tab_privs;

-- Sprawdzenie synonimów
SELECT synonym_name, table_name FROM user_synonyms;
```

### 12.6.4 Problemy z PostgreSQL

**Błąd: ODBC connection failed**

- Sprawdzenie DSN w Windows ODBC Administrator
- Test połączenia przez isql: `isql PostgreSQL30 remarks_user Remarks123`
- Sprawdzenie pg_hba.conf dla autentyfikacji

# 13. Podsumowanie i Wnioski

## 13.1 Realizacja celów projektu

### 13.1.1 Wypełnienie wymagań funkcjonalnych

System Rozproszonej Bazy Danych dla prywatnej szkoły podstawowej w pełni realizuje wszystkie wymagania określone w specyfikacji projektu:

**Środowisko heterogeniczne**

- ✅ **SQL Server - Oracle - PostgreSQL**: Implementacja trzech różnych systemów bazodanowych
- ✅ **Różnorodność ról**: Każdy system pełni specjalizowaną funkcję (edukacyjna, finansowa, uwagi)
- ✅ **Integracja systemów**: Seamless integration przez linked servers i database links

**Linked Servers (SQL Server)**

- ✅ **SQLServer ↔ SQLServer**: Replikacja transakcyjna między instancjami
- ✅ **SQLServer ↔ Oracle**: Połączenie przez OraOLEDB.Oracle provider
- ✅ **SQLServer ↔ PostgreSQL**: Integracja przez ODBC/MSDASQL
- ✅ **SQLServer ↔ Excel**: Eksport danych przez Microsoft.ACE.OLEDB.12.0

**Zapytania ad-hoc**

- ✅ **OPENROWSET**: Implementacja dostępu do Excel i zapytań jednorazowych
- ✅ **OPENQUERY**: Przetwarzanie zdalne w Oracle i PostgreSQL
- ✅ **Wielodostęp**: Jednoczesne zapytania do Oracle, PostgreSQL i lokalnych danych

**Database Links Oracle**

- ✅ **Linki prywatne**: Połączenia między schematami FINANCE_DB, REMOTE_DB1, REMOTE_DB2
- ✅ **Symulacja środowiska rozproszonego**: Három oddzielne schematy Oracle
- ✅ **Synonimy**: Transparentny dostęp do zdalnych tabel

**Transakcje rozproszone**

- ✅ **MS DTC**: Konfiguracja Distributed Transaction Coordinator
- ✅ **Two-Phase Commit**: Implementacja atomowych operacji między systemami
- ✅ **Error handling**: Kompletny rollback przy błędach

**Replikacja**

- ✅ **Replikacja transakcyjna**: Publisher-Subscriber model dla tabeli students
- ✅ **Monitoring**: Procedury sprawdzania statusu replikacji
- ✅ **Automatyzacja**: Agent-based continuous replication

### 13.1.2 Dodane wartości systemu

**Funkcjonalności wykraczające poza podstawowe wymagania:**

**Zaawansowane procedury Oracle**

- **Pakiet pkg_DistributedFinance**: Kompleksowy interfejs do operacji finansowych
- **PIPELINED functions**: Wydajne zwracanie zbiorów danych
- **INSTEAD OF triggers**: Obsługa widoków rozproszonych

**Widoki rozproszone**

- **vw_DistributedStudentData**: Unifikacja danych z trzech systemów
- **vw_StudentFinancialInfo**: Zaawansowane kalkulacje finansowe
- **Performance optimization**: Intelligent remote vs local processing

**System raportowy**

- **Raporty wielosystemowe**: Dane z SQL Server + Oracle + PostgreSQL
- **Eksport Excel**: Multi-sheet exports z różnych źródeł

- **Procedury analityczne**: Kompleksowe statystyki i agregacje

## 13.2 Architektura jako best practice

### 13.2.1 Podział funkcjonalny

**Strategia podziału danych okazała się bardzo efektywna:**

**SQL Server jako hub centralny**

- **Uzasadnienie**: Doskonałe wsparcie dla linked servers i transaction coordination
- **Rezultat**: Successful integration point dla wszystkich systemów
- **Korzyści**: Unified access layer, centralized business logic

**Oracle dla danych finansowych**

- **Uzasadnienie**: Enterprise-grade financial processing, strong consistency
- **Rezultat**: Robust financial operations z complex business rules
- **Korzyści**: Database links simulation, advanced PL/SQL capabilities

**PostgreSQL dla uwag tekstowych**

- **Uzasadnienie**: Excellent text handling, cost-effective solution
- **Rezultat**: Efficient pedagogical notes system
- **Korzyści**: Third-party system simulation, ODBC integration

### 13.2.2 Patterns komunikacji

**Różnorodność mechanizmów dostępu:**

**Synchronous integration**

- **Linked servers**: Real-time data access
- **OPENQUERY**: Remote processing optimization
- **Distributed transactions**: Data consistency guarantees

**Asynchronous integration**

- **Replikacja transakcyjna**: Near real-time data synchronization
- **Eventual consistency**: Balance between performance and consistency

## 13.3 Wyzwania i rozwiązania

### 13.3.1 Wyzwania techniczne

**Heterogeniczność systemów**

**Problem**: Różne typy danych, składnie SQL, mechanizmy uwierzytelniania
**Rozwiązanie**:

- Standardization przez casting i aliasy
- Provider-specific optimizations
- Error handling per system type

**Network latency i performance**

**Problem**: Multiple round-trips między systemami
**Rozwiązanie**:

- Remote processing przez OPENQUERY
- Intelligent JOIN strategies (local vs remote)
- Connection pooling i caching

**Transaction coordination**

**Problem**: ACID compliance across distributed systems
**Rozwiązanie**:

- MS DTC integration
- Careful transaction boundary design
- Compensation logic dla complex scenarios

### 13.3.2 Rozwiązania projektowe

**Error handling strategy**

- **Graceful degradation**: System continues przy partial failures
- **Detailed logging**: Comprehensive error information
- **Recovery procedures**: Clear steps dla system restoration

**Security model**

- **Least privilege**: Minimal required permissions per component
- **Separated credentials**: Independent authentication per system
- **Audit trails**: Logging distributed operations

## 13.4 Możliwości rozwoju

### 13.4.1 Rozszerzenia funkcjonalne

**Dodatkowe moduły edukacyjne**

- **Biblioteka**: Catalog books, lending tracking w dedicated PostgreSQL
- **Laboratoria**: Equipment management w specialized Oracle schema
- **Wydarzenia**: Calendar system z MongoDB integration

**Advanced analytics**

- **Data warehouse**: OLAP cube dla historical analysis
- **Machine learning**: Student performance prediction models
- **Reporting services**: SSRS integration dla automated reports

### 13.4.2 Optymalizacje techniczne

**Performance improvements**

- **Materialized views**: Pre-computed aggregations
- **Partitioning**: Large table optimization
- **Indexing strategy**: Cross-system query optimization

**High availability**

- **Clustering**: Multi-node deployments per database type
- **Backup strategies**: Cross-system consistent backups
- **Disaster recovery**: Geographic distribution

## 13.5 Wartość edukacyjna projektu

### 13.5.1 Demonstracja konceptów RBD

**Projekt ilustruje kluczowe aspekty Rozproszonych Baz Danych:**

**Distribution strategies**

- **Horizontal partitioning**: Different data types across systems
- **Replication**: Master-slave configuration
- **Federation**: Logical integration without physical consolidation

**Transaction management**

- **Distributed ACID**: Cross-system consistency
- **Compensation patterns**: Rollback strategies
- **Isolation levels**: Performance vs consistency trade-offs

**Query processing**

- **Cost-based optimization**: Remote vs local execution
- **Join strategies**: Network-aware query planning
- **Result materialization**: Efficient data transfer

### 13.5.2 Real-world applicability

**System demonstruje realistic enterprise scenarios:**

**Legacy system integration**

- **Brownfield development**: Working with existing systems
- **Gradual migration**: Step-by-step modernization approach
- **Vendor diversity**: Multi-vendor environment management

**Compliance and governance**

- **Data sovereignty**: Different systems for different data types
- **Regulatory requirements**: Specialized systems dla sensitive data
- **Audit trails**: Distributed logging and monitoring

## 13.6 Wnioski końcowe

### 13.6.1 Sukces realizacji

System Rozproszonej Bazy Danych dla prywatnej szkoły podstawowej stanowi **kompleksną implementację** wszystkich wymagań projektu RBD. Demonstrates **professional-grade** distributed database patterns while maintaining **educational clarity**.

**Kluczowe osiągnięcia:**

- ✅ **100% coverage** wymagań funkcjonalnych
- ✅ **Production-ready** architecture patterns
- ✅ **Comprehensive testing** strategy
- ✅ **Detailed documentation** dla maintenance i development

### 13.6.2 Wartość dodana

Projekt wykracza poza podstawowe wymagania, dostarczając:

- **Advanced PL/SQL packages** z enterprise patterns
- **Sophisticated error handling** dla distributed environments
- **Performance optimization** techniques
- **Real-world deployment** considerations

### 13.6.3 Rekomendacje

**Dla środowisk produkcyjnych:**

1. **Security hardening**: Enhanced authentication i encryption
2. **Monitoring tools**: Comprehensive system health monitoring
3. **Backup strategies**: Cross-system consistent backup procedures
4. **Documentation maintenance**: Living documentation dla operational teams

**Dla dalszego rozwoju edukacyjnego:**

1. **NoSQL integration**: MongoDB lub Cassandra jako additional node
2. **Cloud deployment**: Azure/AWS distributed deployment
3. **Microservices architecture**: Service-oriented decomposition
4. **Event-driven patterns**: Message queues i event streaming

System stanowi **solid foundation** dla understanding distributed database concepts i **practical reference** dla implementing podobnych rozwiązań w enterprise environments.