

POLITECHNIKA ŁÓDZKA

Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

Projektowanie i Administracja Baz Danych

Semestr 5

---

# Projekt Bazy Danych

Company (CrunchBase)

---

**Autor:** Mateusz Mróz

**Nr indeksu:** 251190

Łódź, styczeń 2026

# Spis treści

<b>1</b>	<b>Podstawowe założenia projektu</b>	<b>3</b>
1.1	Cel tworzenia bazy danych . . . . .	3
1.2	Główne założenia . . . . .	3
1.3	Zakres możliwości systemu . . . . .	3
1.4	Ograniczenia przyjęte podczas projektowania . . . . .	3
<b>2</b>	<b>Schemat bazy danych</b>	<b>4</b>
2.1	Diagram ERD . . . . .	4
2.2	Opis struktury relacyjnej . . . . .	4
2.3	Normalizacja . . . . .	4
<b>3</b>	<b>Obiekty bazy danych i ich opis</b>	<b>6</b>
3.1	Tabele . . . . .	6
3.1.1	1. Tabela crunchbase.Company . . . . .	6
3.1.2	2. Tabela crunchbase.Person . . . . .	6
3.1.3	3. Tabela crunchbase.FinancialOrg . . . . .	6
3.1.4	4. Tabela crunchbase.Product . . . . .	6
3.1.5	5. Tabela crunchbase.Office . . . . .	7
3.1.6	6. Tabela crunchbase.FundingRound . . . . .	7
3.1.7	7. Tabela crunchbase.Investment . . . . .	7
3.1.8	8. Tabela crunchbase.Acquisition . . . . .	7
3.1.9	9. Tabela crunchbase.Milestone . . . . .	8
3.1.10	10. Tabela crunchbase.Competitor . . . . .	8
3.1.11	11. Tabela crunchbase.CompanyRelationship . . . . .	8
3.1.12	12. Tabela crunchbase.ExternalLink . . . . .	8
3.1.13	13. Tabela crunchbase.Screenshot . . . . .	8
3.1.14	14. Tabela crunchbase.ScreenshotSize . . . . .	9
3.1.15	15. Tabela crunchbase.VideoEmbed . . . . .	9
3.1.16	16. Tabela crunchbase.Provider . . . . .	9
3.1.17	17. Tabela crunchbase.CompanyImage . . . . .	9
3.1.18	18. Tabela crunchbase.CompanyIPO . . . . .	9
3.2	Indeksy . . . . .	10
3.3	Triggery . . . . .	10
3.4	Procedury składowane . . . . .	11
3.4.1	Procedura: UpsertCompany . . . . .	11
3.5	Funkcje użytkownika . . . . .	12
3.5.1	Funkcja skalarna: GetTotalFunding . . . . .	12
3.6	Widoki . . . . .	13
3.6.1	Widok: vw_CompanyOverview . . . . .	13
<b>4</b>	<b>Role, uprawnienia i użytkownicy</b>	<b>14</b>
4.1	Konfiguracja bazy dla Contained Users . . . . .	14
4.2	Role w bazie danych . . . . .	14
4.3	Użytkownicy (Contained Users) . . . . .	15
4.4	Sposób logowania jako Contained User . . . . .	15
4.5	Testowanie uprawnień . . . . .	15

<b>5</b>	<b>Uwagi końcowe</b>	<b>17</b>
5.1	Napotkane problemy . . . . .	17
5.2	Elementy zrealizowane . . . . .	17
5.3	Pliki projektu . . . . .	17
5.4	Kolejność uruchamiania skryptów . . . . .	18

# 1. Podstawowe założenia projektu

## 1.1. Cel tworzenia bazy danych

Celem projektu jest stworzenie relacyjnej bazy danych **CompanyDB** na serwerze MS SQL. Baza służy do przechowywania danych o firmach (startupach), ich pracownikach i inwestycjach. Dane wejściowe pochodzą z plików JSON (serwis CrunchBase). Moim zadaniem było zaprojektowanie struktury tabel i napisanie skryptu, który przeniesie te dane z formatu dokumentowego (JSON) do uporządkowanej bazy SQL.

## 1.2. Główne założenia

Przyjąłem następujące zasady przy tworzeniu bazy:

- **Silnik:** Baza działa na Microsoft SQL Server.
- **Struktura:** Baza jest relacyjna i dąży do 3. Postaci Normalnej (3NF), aby nie powielać niepotrzebnie danych.
- **Bezpieczeństwo:** Użyłem opcji *Contained Database*. Dzięki temu użytkownicy są przypisani bezpośrednio do bazy danych, a nie do całego serwera SQL. To ułatwia przenoszenie bazy.
- **Spójność:** Zastosowałem klucze obce (Foreign Keys). Jeśli usuniemy firmę, to automatycznie usuną się jej produkty czy biura (opcja `ON DELETE CASCADE`).
- **Klucze:** Każda tabela ma swój własny, unikalny numer ID (np. `company_id`), który generuje się automatycznie. Nie używam nazw firm jako kluczy, bo mogą się powtarzać lub zmieniać.

## 1.3. Zakres możliwości systemu

W bazie można:

- Przechowywać informacje o firmach, ich produktach i adresach biur.
- Sprawdzać historię finansowania — kto, ile i kiedy zainwestował w daną firmę.
- Widzieć powiązania między ludźmi a firmami (kto gdzie pracuje lub pracował).
- Analizować konkurencję i przejęcia firm.
- Automatycznie importować dane z plików JSON za pomocą przygotowanego skryptu SQL.

## 1.4. Ograniczenia przyjęte podczas projektowania

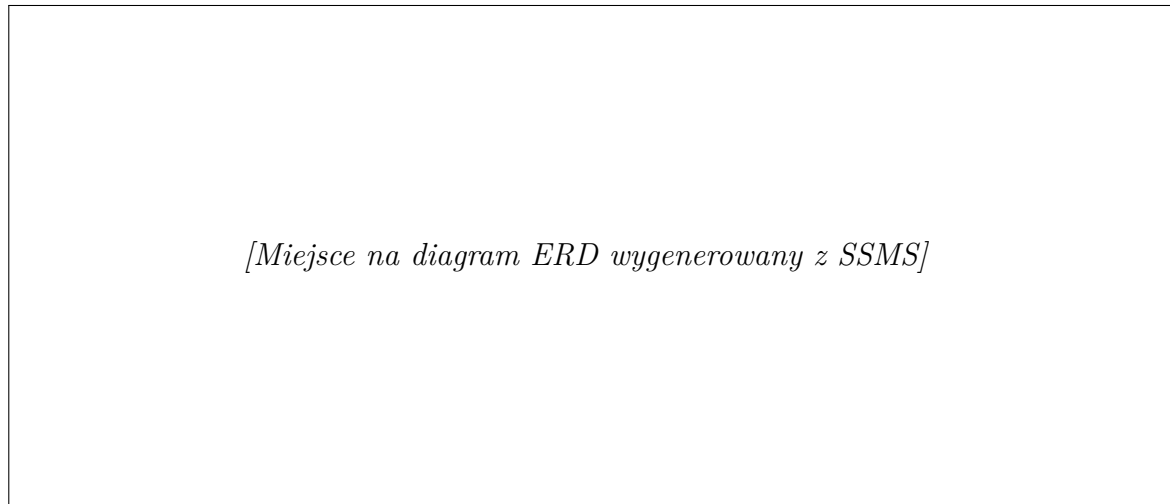
W projekcie zastosowałem kilka uproszczeń:

- **Tagi:** Pola takie jak `tag_list` (lista tagów po przecinku) zostawiłem jako zwykły tekst. Rozbijanie tego na osobne tabele skomplikowałoby import, a w tym projekcie nie jest to kluczowe. To świadome odstępstwo od 1. Postaci Normalnej.
- **Waluty:** Zakładałem, że kwoty są w dolarach (USD) lub są przeliczane, nie robiłem osobnego systemu kursów walut.
- **Braki danych:** Pliki JSON są dziurawe (np. brak roku założenia), więc baza pozwala na wpisywanie wartości pustych (NULL) w wielu miejscach.

## 2. Schemat bazy danych

### 2.1. Diagram ERD

Poniżej przedstawiono diagram relacji bazy danych. Diagram można wygenerować w SQL Server Management Studio poprzez: Database Diagrams → New Database Diagram.



Rysunek 1: Diagram relacji bazy danych CompanyDB

### 2.2. Opis struktury relacyjnej

Sercem bazy jest tabela **Company** (Firmy). Wszystkie inne tabele są z nią powiązane — np. **Product** (Produkty), **Office** (Biura).

Podczas projektowania musiałem rozwiązać dwa trudniejsze problemy:

#### 1. Relacja Wiele-do-Wielu (Ludzie i Firmy)

Jedna osoba może pracować w wielu firmach, a w firmie pracuje wiele osób. Nie mogłem wpisać pracownika bezpośrednio do tabeli firmy.

- **Rozwiązanie:** Stworzyłem tabelę łączącą **CompanyRelationship**. Łączy ona ID osoby z ID firmy i dodaje informację o stanowisku (np. “CEO”).

#### 2. Kto jest inwestorem? (Różne typy inwestorów)

W firmę może zainwestować: inna osoba, inna firma albo bank/fundusz.

- **Rozwiązanie:** W tabeli **Investment** (Inwestycje) mam trzy kolumny na ID: **person\_id**, **financial\_org\_id** i **investing\_company\_id**. Wypełniam tylko jedną z nich dla danej inwestycji, a reszta zostaje pusta. To proste i skuteczne rozwiązanie problemu polimorfizmu w SQL.

### 2.3. Normalizacja

Starałem się trzymać zasad normalizacji:

- **1NF (Atomowość):** W każdej kratce tabeli jest jedna wartość. Wyjątkiem są wspomniane wcześniej tagi, które zostawiłem jako listę po przecinku dla wygody.

- **2NF:** Każda tabela ma swój klucz główny (ID), więc wszystkie dane w wierszu dotyczą konkretnego obiektu (np. konkretnej firmy). Nie ma sytuacji, że część danych zależy od czegoś innego.
- **3NF:** Usunąłem zależności przechodnie.

*Mały wyjątek:* W tabeli **Acquisition** (Przejęcia) trzymam nazwę przejętej firmy ORAZ jej ID. Robię to celowo — czasem przejmowana firma nie istnieje w naszej bazie (nie ma ID), a musimy wiedzieć, jak się nazywała. To bezpieczniejsza opcja.

## 3. Obiekty bazy danych i ich opis

### 3.1. Tabele

Baza danych składa się z 18 tabel. Poniżej znajduje się szczegółowy opis każdej z nich.

#### 3.1.1. 1. Tabela `crunchbase.Company`

**Opis:** Główna tabela przechowująca podstawowe informacje o firmach (startupach), takie jak nazwa, kategoria, opis, daty założenia i zamknięcia.

**Klucz główny (PK):** `company_id` (INT, IDENTITY).

**Klucze obce (FK):** Brak.

**Ograniczenia:**

- UNIQUE na kolumnach `mongo_id` oraz `permalink`.
- CHECK (`number_of_employees`  $\geq$  0) — liczba pracowników nie może być ujemna.
- CHECK (`founded_month` BETWEEN 1 AND 12) — walidacja miesiąca.
- CHECK (`founded_day` BETWEEN 1 AND 31) — walidacja dnia.
- DEFAULT GETDATE() dla dat utworzenia rekordu.

**Uwagi:** Pole `tag_list` jest przechowywane jako tekst (odstępstwo od 1NF dla uproszczenia).

#### 3.1.2. 2. Tabela `crunchbase.Person`

**Opis:** Słownik osób (pracowników, założycieli, inwestorów).

**Klucz główny (PK):** `person_id` (INT, IDENTITY).

**Klucze obce (FK):** Brak.

**Ograniczenia:** UNIQUE na kolumnie `permalink`.

#### 3.1.3. 3. Tabela `crunchbase.FinancialOrg`

**Opis:** Słownik organizacji finansowych (fundusze VC, banki).

**Klucz główny (PK):** `financial_org_id` (INT, IDENTITY).

**Klucze obce (FK):** Brak.

**Ograniczenia:** UNIQUE na kolumnie `permalink`.

#### 3.1.4. 4. Tabela `crunchbase.Product`

**Opis:** Produkty oferowane przez firmy.

**Klucz główny (PK):** `product_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id`  $\rightarrow$  `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

### 3.1.5. 5. Tabela crunchbase.Office

**Opis:** Lokalizacje biur i siedzib firm.

**Klucz główny (PK):** office\_id (INT, IDENTITY).

**Klucz obcy (FK):** company\_id → Company (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

### 3.1.6. 6. Tabela crunchbase.FundingRound

**Opis:** Rundy finansowania firm — typ rundy (seed, Series A, B, C, etc.), kwota, waluta, data.

**Klucz główny (PK):** funding\_round\_id (INT, IDENTITY).

**Klucz obcy (FK):** company\_id → Company (ON DELETE CASCADE).

**Ograniczenia:**

- CHECK (raised\_amount >= 0) — kwota nie może być ujemna.
- DEFAULT 'USD' dla raised\_currency\_code.
- CHECK (funded\_month BETWEEN 1 AND 12) — walidacja miesiąca.
- CHECK (funded\_day BETWEEN 1 AND 31) — walidacja dnia.

### 3.1.7. 7. Tabela crunchbase.Investment

**Opis:** Łączy rundę finansowania z inwestorem. Inwestorem może być osoba, organizacja finansowa lub inna firma.

**Klucz główny (PK):** investment\_id (INT, IDENTITY).

**Klucze obce (FK):**

- funding\_round\_id → FundingRound (ON DELETE CASCADE) — w co zainwestowano.
- person\_id → Person (opcjonalny) — jeśli inwestorem jest osoba.
- financial\_org\_id → FinancialOrg (opcjonalny) — jeśli inwestorem jest organizacja.
- investing\_company\_id → Company (opcjonalny) — jeśli inwestorem jest inna firma.

**Uwagi:** W danym rekordzie wypełniony jest zazwyczaj tylko jeden z trzech kluczy inwestorów.

### 3.1.8. 8. Tabela crunchbase.Acquisition

**Opis:** Informacje o zakupie jednej firmy przez drugą.

**Klucz główny (PK):** acquisition\_id (INT, IDENTITY).

**Klucze obce (FK):**

- acquiring\_company\_id → Company — firma kupująca.
- acquired\_company\_id → Company (opcjonalny) — firma kupowana.



**Uwagi do normalizacji:** Przechowuję nazwę przejmowanej firmy (`acquired_company_name`) jako tekst, nawet jeśli mam do niej ID. To zabezpieczenie na wypadek, gdyby przejmowana firma nie istniała w naszej bazie jako osobny rekord (świadoma redundancja).

### 3.1.9. 9. Tabela `crunchbase.Milestone`

**Opis:** Kamienie milowe w historii firmy (ważne wydarzenia).

**Klucz główny (PK):** `milestone_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

### 3.1.10. 10. Tabela `crunchbase.Competitor`

**Opis:** Informacje o konkurentach firm.

**Klucz główny (PK):** `competitor_id` (INT, IDENTITY).

**Klucze obce (FK):**

- `company_id` → `Company` (ON DELETE CASCADE) — firma.
- `competitor_company_id` → `Company` (opcjonalny) — firma konkurencyjna.

**Ograniczenia:** UNIQUE (`company_id`, `competitor_permalink`) — unikalna para firma-konkurent.

### 3.1.11. 11. Tabela `crunchbase.CompanyRelationship`

**Opis:** Tabela łącznikowa dla relacji wiele-do-wielu między osobami a firmami. Przechowuje historię zatrudnienia.

**Klucz główny (PK):** `relationship_id` (INT, IDENTITY).

**Klucze obce (FK):**

- `company_id` → `Company` (ON DELETE CASCADE).
- `person_id` → `Person` (ON DELETE CASCADE).

**Dodatkowe pola:** `title` (stanowisko), `is_past` (czy to była praca w przeszłości).

### 3.1.12. 12. Tabela `crunchbase.ExternalLink`

**Opis:** Linki zewnętrzne do artykułów o firmie.

**Klucz główny (PK):** `external_link_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

### 3.1.13. 13. Tabela `crunchbase.Screenshot`

**Opis:** Zrzuty ekranu produktów.

**Klucz główny (PK):** `screenshot_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

#### 3.1.14. 14. Tabela `crunchbase.ScreenshotSize`

**Opis:** Różne rozmiary zrzutów ekranu (thumbnails).

**Klucz główny (PK):** `screenshot_size_id` (INT, IDENTITY).

**Klucz obcy (FK):** `screenshot_id` → `Screenshot` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

#### 3.1.15. 15. Tabela `crunchbase.VideoEmbed`

**Opis:** Osadzone filmy (kod embed) dotyczące firm.

**Klucz główny (PK):** `video_embed_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

#### 3.1.16. 16. Tabela `crunchbase.Provider`

**Opis:** Dostawcy usług dla firm.

**Klucz główny (PK):** `provider_id` (INT, IDENTITY).

**Klucze obce (FK):**

- `company_id` → `Company` (ON DELETE CASCADE) — firma.
- `provider_company_id` → `Company` (opcjonalny) — firma dostawcy.

**Ograniczenia:** Brak dodatkowych.

#### 3.1.17. 17. Tabela `crunchbase.CompanyImage`

**Opis:** Obrazy i logo firm.

**Klucz główny (PK):** `image_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE).

**Ograniczenia:** Brak dodatkowych.

#### 3.1.18. 18. Tabela `crunchbase.CompanyIPO`

**Opis:** Informacje o IPO (Initial Public Offering) firmy.

**Klucz główny (PK):** `ipo_id` (INT, IDENTITY).

**Klucz obcy (FK):** `company_id` → `Company` (ON DELETE CASCADE, UNIQUE).

**Ograniczenia:** Firma może mieć tylko jedno IPO (relacja 1:1).

### 3.2. Indeksy

W bazie danych utworzono następujące indeksy nieklastrowe (poza automatycznymi dla PK i UNIQUE):

Tabela 1: Indeksy utworzone w bazie danych

Nazwa indeksu	Tabela	Kolumna
IX_Company_Name	Company	name
IX_Company_CategoryCode	Company	category_code
IX_FundingRound_RoundCode	FundingRound	round_code

#### Uzasadnienie utworzenia indeksów:

1. **IX\_Company\_Name** — przyspiesza wyszukiwanie firm po nazwie, często używane w zapytaniach.
2. **IX\_Company\_CategoryCode** — przyspiesza filtrowanie firm według kategorii działalności.
3. **IX\_FundingRound\_RoundCode** — przyspiesza grupowanie i filtrowanie rund finansowania według typu (seed, series-a, series-b, etc.).

### 3.3. Triggery

W obecnej wersji bazy danych **nie zdefiniowano triggerów**. Zrezygnowano z triggerów na rzecz więzów integralności (FOREIGN KEY CASCADE) oraz procedur składowanych, co zapewnia lepszą wydajność i przejrzystość logiki bazy.

Integralność danych jest zapewniona przez:

- więzy klucza obcego (FK) z opcją ON DELETE CASCADE,
- ograniczenia CHECK walidujące wartości,
- ograniczenia DEFAULT dla wartości domyślnych,
- procedurę składowaną do bezpiecznego wstawiania/aktualizacji danych.

## 3.4. Procedury składowane

### 3.4.1. Procedura: UpsertCompany

**Opis:** Procedura służąca do wstawiania nowej firmy lub aktualizacji istniejącej na podstawie unikalnego identyfikatora `mongo_id`. Implementuje wzorzec UPSERT (UPDATE or INSERT).

**Parametry wejściowe:**

- `@mongo_id` (NVARCHAR(50)) — unikalny identyfikator z MongoDB (wymagany)
- `@name` (NVARCHAR(255)) — nazwa firmy (wymagany)
- `@permalink` (NVARCHAR(255)) — unikalny identyfikator URL (wymagany)
- `@category_code` (NVARCHAR(100)) — kategoria działalności (opcjonalny)
- `@description` (NVARCHAR(MAX)) — opis firmy (opcjonalny)
- `@number_of_employees` (INT) — liczba pracowników (opcjonalny)
- `@founded_year` (INT) — rok założenia (opcjonalny)

**Działanie:**

1. Sprawdza, czy firma o podanym `mongo_id` już istnieje w bazie.
2. Jeśli istnieje — aktualizuje dane (UPDATE) i ustawia `updated_at = GETDATE()`.
3. Jeśli nie istnieje — wstawia nowy rekord (INSERT).
4. Wyświetla komunikat o wykonanej operacji.

```
1 CREATE OR ALTER PROCEDURE crunchbase.UpsertCompany
2     @mongo_id NVARCHAR(50),
3     @name NVARCHAR(255),
4     @permalink NVARCHAR(255),
5     @category_code NVARCHAR(100) = NULL,
6     @description NVARCHAR(MAX) = NULL,
7     @number_of_employees INT = NULL,
8     @founded_year INT = NULL
9 AS
10 BEGIN
11     SET NOCOUNT ON;
12
13     IF EXISTS (SELECT 1 FROM crunchbase.Company
14                WHERE mongo_id = @mongo_id)
15     BEGIN
16         UPDATE crunchbase.Company
17         SET name = @name,
18             category_code = @category_code,
19             description = @description,
20             number_of_employees = @number_of_employees,
21             founded_year = @founded_year,
22             updated_at = GETDATE()
23         WHERE mongo_id = @mongo_id;
24
25         PRINT 'Firma zaktualizowana: ' + @name;
26     END
27 ELSE
28 BEGIN
```

```
29      INSERT INTO crunchbase.Company
30          (mongo_id, name, permalink, category_code,
31           description, number_of_employees, founded_year)
32      VALUES
33          (@mongo_id, @name, @permalink, @category_code,
34           @description, @number_of_employees, @founded_year);
35
36      PRINT 'Firma dodana: ' + @name;
37  END
38 END
```

Listing 1: Procedura UpsertCompany

**Zastosowanie:** Procedura umożliwia bezpieczne dodawanie i aktualizowanie firm bez ryzyka duplikacji danych. Jest wykorzystywana przez użytkownika Emp do zarządzania danymi.

## 3.5. Funkcje użytkownika

### 3.5.1. Funkcja skalarna: GetTotalFunding

**Opis:** Funkcja skalarna zwracająca łączną sumę finansowania pozyskanego przez firmę we wszystkich rundach finansowania.

**Parametry wejściowe:**

- @company\_id (INT) — identyfikator firmy

**Zwracana wartość:** DECIMAL(18,2) — suma finansowania (0 jeśli brak danych)

```
1 CREATE OR ALTER FUNCTION crunchbase.GetTotalFunding(
2     @company_id INT
3 )
4 RETURNS DECIMAL(18,2)
5 AS
6 BEGIN
7     DECLARE @total DECIMAL(18,2);
8
9     SELECT @total = ISNULL(SUM(raised_amount), 0)
10    FROM crunchbase.FundingRound
11   WHERE company_id = @company_id;
12
13     RETURN @total;
14 END
```

Listing 2: Funkcja GetTotalFunding

**Zastosowanie:** Funkcja jest wykorzystywana w widoku vw\_CompanyOverview do prezentacji podsumowania finansowania każdej firmy. Może być również używana bezpośrednio w zapytaniach SELECT.

**Przykład użycia:**

```
1 SELECT name, crunchbase.GetTotalFunding(company_id) AS funding
```

```
2 FROM crunchbase.Company
3 ORDER BY funding DESC;
```

## 3.6. Widoki

### 3.6.1. Widok: vw\_\_CompanyOverview

**Opis:** Widok prezentujący podsumowanie informacji o firmach wraz z obliczonym łącznym finansowaniem i liczbą produktów oraz rund finansowania.

```
1 CREATE OR ALTER VIEW crunchbase.vw_CompanyOverview
2 AS
3 SELECT
4     c.company_id,
5     c.name,
6     c.category_code,
7     c.founded_year,
8     c.number_of_employees,
9     c.homepage_url,
10    crunchbase.GetTotalFunding(c.company_id) AS total_funding,
11    (SELECT COUNT(*) FROM crunchbase.FundingRound fr
12     WHERE fr.company_id = c.company_id) AS funding_rounds_count,
13    (SELECT COUNT(*) FROM crunchbase.Product p
14     WHERE p.company_id = c.company_id) AS products_count
15 FROM crunchbase.Company c;
```

Listing 3: Widok vw\_\_CompanyOverview

#### Kolumny widoku:

- `company_id` — identyfikator firmy
- `name` — nazwa firmy
- `category_code` — kategoria działalności
- `founded_year` — rok założenia
- `number_of_employees` — liczba pracowników
- `homepage_url` — strona główna firmy
- `total_funding` — łączne finansowanie (obliczone przez funkcję)
- `funding_rounds_count` — liczba rund finansowania
- `products_count` — liczba produktów

**Zastosowanie:** Widok umożliwia szybki przegląd najważniejszych informacji o firmach bez konieczności pisania złożonych zapytań. Jest udostępniony użytkownikowi `Guest` do przeglądania danych.

#### Przykład użycia:

```
1 SELECT * FROM crunchbase.vw_CompanyOverview
2 ORDER BY total_funding DESC;
```

## 4. Role, uprawnienia i użytkownicy

### 4.1. Konfiguracja bazy dla Contained Users

Baza danych została skonfigurowana z opcją `CONTAINMENT = PARTIAL`, co umożliwia tworzenie użytkowników przechowywanych lokalnie w bazie (contained users). Użytkownicy ci nie wymagają loginów na poziomie serwera.

```

1  -- Włączenie zaawansowanych opcji
2  EXEC sp_configure 'show advanced options', 1;
3  RECONFIGURE;
4
5  -- Włączenie contained database authentication
6  EXEC sp_configure 'contained database authentication', 1;
7  RECONFIGURE;
8
9  -- Utworzenie bazy z CONTAINMENT = PARTIAL
10 CREATE DATABASE CompanyDB CONTAINMENT = PARTIAL;
```

Listing 4: Konfiguracja Contained Database Authentication

### 4.2. Role w bazie danych

W bazie zdefiniowano trzy role odpowiadające różnym poziomom dostępu:

Tabela 2: Role zdefiniowane w bazie danych

Rola	Poziom	Uprawnienia
AdminRole	Pełny	Członek roli <code>db_owner</code> — pełne uprawnienia do wszystkich obiektów bazy
EmpRole	Ograniczony	Uprawnienie <code>EXECUTE</code> na schemacie <code>crunchbase</code> — może wykonywać procedury składowane
GuestRole	Minimalny	Uprawnienie <code>SELECT</code> na widoku <code>vw_CompanyOverview</code> — tylko odczyt przez widok

```

1  -- Rola administratora
2  CREATE ROLE AdminRole;
3  ALTER ROLE db_owner ADD MEMBER AdminRole;
4
5  -- Rola pracownika
6  CREATE ROLE EmpRole;
7  GRANT EXECUTE ON SCHEMA::crunchbase TO EmpRole;
8
9  -- Rola gościa
10 CREATE ROLE GuestRole;
11 GRANT SELECT ON crunchbase.vw_CompanyOverview TO GuestRole;
```

Listing 5: Tworzenie ról i nadawanie uprawnień

### 4.3. Użytkownicy (Contained Users)

W bazie utworzono trzech użytkowników lokalnych (contained users):

Tabela 3: Użytkownicy w bazie danych

Użytkownik	Hasło	Rola	Możliwości
Admin	Admin	AdminRole	Pełny dostęp do bazy, tworzenie/modyfikacja/usuwanie obiektów
Emp	Emp	EmpRole	Wykonywanie procedur składowanych (np. Upsert-Company)
Guest	Guest	GuestRole	Przeglądanie danych tylko przez widok

```

1  -- Administrator
2  CREATE USER Admin WITH PASSWORD = 'Admin';
3  ALTER ROLE AdminRole ADD MEMBER Admin;
4
5  -- Pracownik
6  CREATE USER Emp WITH PASSWORD = 'Emp';
7  ALTER ROLE EmpRole ADD MEMBER Emp;
8
9  -- Gosc
10 CREATE USER Guest WITH PASSWORD = 'Guest';
11 ALTER ROLE GuestRole ADD MEMBER Guest;

```

Listing 6: Tworzenie użytkowników contained

### 4.4. Sposób logowania jako Contained User

Aby zalogować się jako contained user w SQL Server Management Studio:

1. Server name: localhost (lub nazwa serwera)
2. Authentication: SQL Server Authentication
3. Login: Admin / Emp / Guest
4. Password: Admin / Emp / Guest
5. Options → Connect to database: CompanyDB

### 4.5. Testowanie uprawnień

Test użytkownika Admin:

```

1  -- Logowanie jako Admin - moze wszystko
2  SELECT * FROM crunchbase.Company;
3  INSERT INTO crunchbase.Company (mongo_id, name, permalink)

```



```
4 VALUES ('test123', 'Test', 'test');
5 EXEC crunchbase.UpsertCompany 'test456', 'Test2', 'test2';
```

### Test użytkownika Emp:

```
1 -- Logowanie jako Emp - moze wykonac procedure
2 EXEC crunchbase.UpsertCompany 'test789', 'Test3', 'test3';
3
4 -- Nie moze bezposrednio modyfikowac tabel
5 SELECT * FROM crunchbase.Company; -- BLAD!
```

### Test użytkownika Guest:

```
1 -- Logowanie jako Guest - moze tylko przegladac widok
2 SELECT * FROM crunchbase.vw_CompanyOverview;
3
4 -- Nie moze nic wiecej
5 SELECT * FROM crunchbase.Company; -- BLAD!
6 EXEC crunchbase.UpsertCompany 'x', 'X', 'x'; -- BLAD!
```

## 5. Uwagi końcowe

### 5.1. Napotkane problemy

Podczas realizacji projektu napotkano następujące wyzwania:

1. **Struktura zagnieżdżona JSON** — dokumenty JSON zawierały głęboko zagnieżdżone obiekty (np. `investments` wewnątrz `funding_rounds`). Wymagało to użycia wielokrotnego `CROSS APPLY OPENJSON`.
2. **Polimorficzne inwestycje** — inwestorem może być osoba (`person`), organizacja finansowa (`financial_org`) lub firma (`company`). Rozwiązano przez trzy opcjonalne klucze obce w tabeli `Investment`.
3. **Referencje do nieistniejących firm** — niektórzy konkurenci i przejęte firmy nie istnieją w danych źródłowych (6 firm). Zachowano ich nazwy i permalinki jako tekst, z opcjonalnym kluczem obcym jeśli firma istnieje w bazie.
4. **Contained users** — wymagana była dodatkowa konfiguracja serwera (`sp_configure`) przed utworzeniem użytkowników lokalnych.

### 5.2. Elementy zrealizowane

- ✓ 18 tabel w schemacie `crunchbase` z pełną strukturą więzów integralności
- ✓ Klucze główne (PK) i obce (FK) dla wszystkich tabel
- ✓ Ograniczenia `CHECK`, `UNIQUE` i `DEFAULT`
- ✓ 3 indeksy nieklastrowe na często używanych kolumnach
- ✓ 1 procedura składowana (`UpsertCompany`)
- ✓ 1 funkcja skalarna (`GetTotalFunding`)
- ✓ 1 widok (`vw_CompanyOverview`)
- ✓ 3 role z różnymi uprawnieniami
- ✓ 3 contained users (`Admin`, `Emp`, `Guest`)
- ✓ Skrypt importu danych z JSON (`OPENJSON`)
- ✓ Normalizacja do 3NF

### 5.3. Pliki projektu

Tabela 4: Pliki wchodzące w skład projektu

Plik	Opis
01_struktura.sql	Tworzenie bazy danych, schematu i wszystkich 18 tabel z ograniczeniami
02_obiekty.sql	Procedura składowana, funkcja skalarna i widok
03_uzytkownicy.sql	Role i contained users z uprawnieniami
04_import.sql	Import danych z pliku JSON do wszystkich tabel
05_testy.sql	Zapytania testowe i demonstracyjne ( <code>SELECT</code> , <code>JOIN</code> , agregacje)
Raport_CompanyDB.pdf	Niniejszy dokument

## 5.4. Kolejność uruchamiania skryptów

1. 01\_struktura.sql — tworzenie bazy i tabel
2. 02\_obiekty.sql — procedura, funkcja, widok
3. 03\_uzytkownicy.sql — role i użytkownicy
4. 04\_import.sql — import danych z JSON
5. 05\_testy.sql — zapytania testowe (opcjonalnie, do demonstracji)

**Uwaga:** W pliku 04\_import.sql należy zmodyfikować ścieżkę do pliku JSON odpowiednio do lokalizacji na dysku.

---

*Koniec dokumentu*