

# Dokumentacja bazy danych „Collaborate”

## Spis treści

1. Opis ogólny.....	3
2. Schemat i relacje.....	5
3. Tabele.....	8
4. Procedury.....	25
5. Zabezpieczenia.....	35

## Opis ogólny

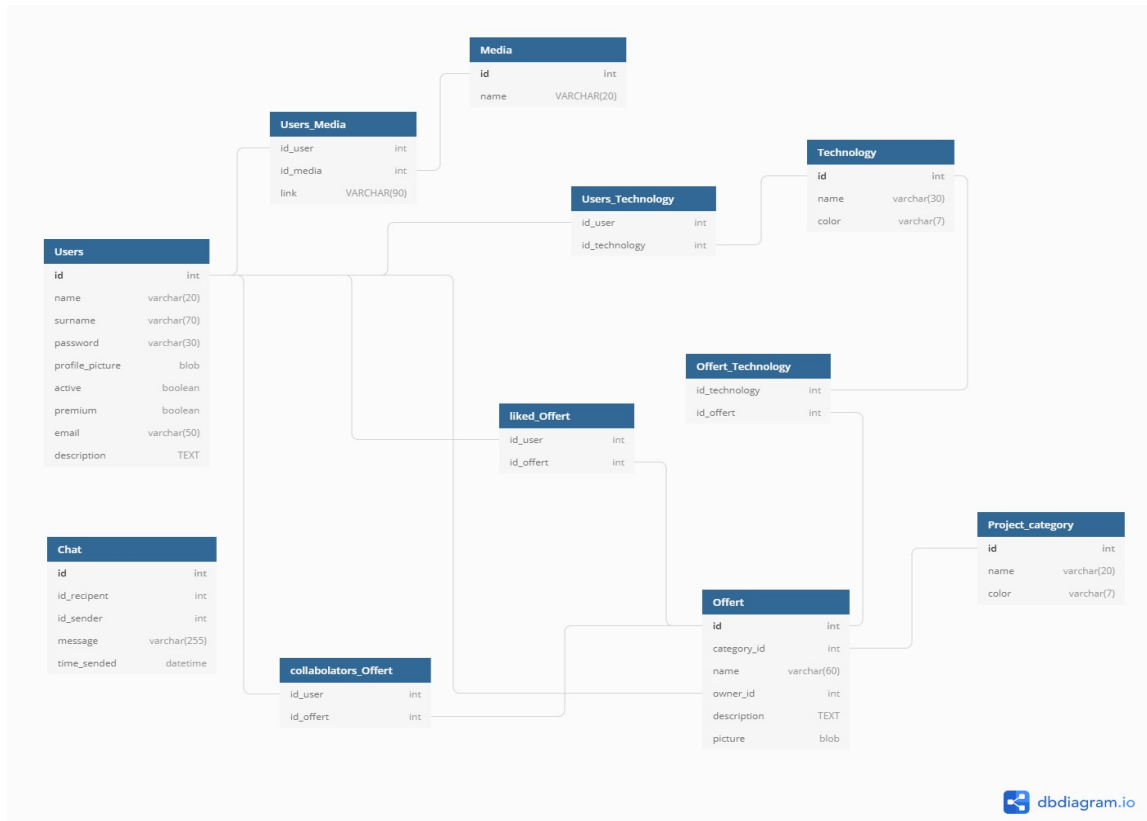
Opisana w niniejszej dokumentacji baza danych o nazwie „collaborate” jest używana do przechowywania danych w aplikacji pod tą samą nazwą. Zalecane jest najpierw zaznajomienie się najpierw z dokumentacją samej aplikacji. Baza jest przygotowana aby obsługiwać wiele użytkowników naraz, a dzięki różnym zabezpieczeniom gwarantuje spójność i bezpieczeństwo danych. Dodatkową jej zaletą są specjalnie zaprojektowane procedury, funkcje dzięki którym znacznie zostaje przyspieszona praca po stronie technologii backendowej. Procedury posiadają konstrukcję, która gwarantuje właściwe wstawianie danych skorelowanych w wielu tabelach, a wymagają jedynie podania odpowiednich parametrów. Zawierają też podstawową obsługę błędów. Aby zainstalować bazę na swoim serwerze należy uruchomić system zarządzania bazami i wykonać plik „init.sql”. Tworzy on całą strukturę bazy danych, jej obiekty oraz wstawia podstawowe,

niezbędne dane do tabel. Baza danych „collaborate” napisana jest w dialekcie mysql i zawiera obsługę polskich znaków diakrytycznych.



Zdjęcie 1. plik inicjujący

# Schemat i relacje



Zdjęcie 2. schemat bazy danych

Powyższe zdjęcie przedstawia ogólny schemat bazy danych oraz relacje między jej obiektami. Oto dokładne liczby:

- 12 tabel
- 5 połączeń typu „wiele do wielu”
- 4 połączenia typu „jeden do wielu”

Poniżej opisane zostały wszystkie relacje zachodzące w bazie danych. Szczegółowe opisy tabel znajdują się w sekcji „Tabele”

### **Relacje „wiele do wielu”:**

- Users – Technology : Połączenie reprezentowane przez tabelę pośrednią Users\_technology służy do przechowywania informacji o poznanych przez użytkownika technologiach informatycznych, innym słowem w jakim języku umie on programować
- Users – Media : Połączenie reprezentowane przez tabelę pośrednią Users\_Media zawiera informację o linkach do poszczególnych stron internetowych (np. Github, LinkedIn) na których dany użytkownik prowadzi swoje aktywności
- Users – Offert (match) : Połączenie obu tabel za pomocą tabelki Liked\_Offert. Przechowuje ona informacje o „polubieniach” ofert lub użytkowników niezbędne do zidentyfikowania czy zachodzi obustronny „match” (czyli polubienie przez właściciela oferty potencjalnego współpracownika oraz polubienie oferty przez danego użytkownika myślącego o współpracy nad projektem, po dalsze wyjaśnienia zobacz dokumentację aplikacji „Collaborate”)

- Users – Offert (collaborators) : Połączenie przez tabelę Collaborators\_Offert służy do przechowywania informacji o współpracownikach (z wyjątkiem właściciela oferty) pracujących nad danym projektem

### Relacje „jeden do wielu”:

- Users-Offert: Połączenie poprzez pole Offert.owner\_id. Służy do określenia właściciela (innymi słowami założyciela oferty na portalu) projektu
- Offert-Project\_Category: Połączenie możliwe dzięki polu Offert.category\_id. Istnieje by określać kategorię projektu innymi słowami do tego jaka jest docelowa platforma np. Web, Desktop. Dla złożonych projektów istnieje wartość „Mixed”
- Users-Chat (2-krotne) : Połączenie klucza podstawowego tabeli User z polami Chat.id\_recipient i Chat.id\_sender

# Tabele

## Tabela Users

Users	
id	int
name	varchar(20)
surname	varchar(70)
password	varchar(30)
profile_picture	blob
active	boolean
premium	boolean
email	varchar(50)
description	TEXT

Zdjęcie 3. Tabela Users

```
CREATE TABLE Users (  
  id int PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  name varchar(20) NOT NULL,  
  surname varchar(70) NOT NULL,  
  password varchar(255) NOT NULL,  
  profile_picture blob,  
  active boolean DEFAULT false,  
  premium boolean DEFAULT false,  
  email varchar(50) UNIQUE NOT NULL,  
  description TEXT,  
  INDEX LOGIN_INDEX(email,password),  
  INDEX ID_INDEX(id)  
);
```

Zdjęcie 4. Polecenia tworzące tabelę Users

Tabela Users służy do przechowywania części niezbędnych informacji o użytkownikach portalu. Oto szczegółowy opis jej pól:

- **id (int)** : pole klucza podstawowego tabeli
- **name (varchar)** : imię użytkownika
- **surname (varchar)** : nazwisko użytkownika
- **password (varchar)** : hasło konta
- **email (varchar)** : pole na email
- **profile\_picture (blob)** : przechowuje zdjęcie profilowe



- **active (boolean)** : zawiera wartość 1 lub 0 (true lub false) i służy do określania czy użytkownik zalogował się pierwszy raz na swoje konto, jeżeli tak, zostaje przekierowany do formularza w którym musi uzupełnić swoje dane profilowe. Bo jego wypełnieniu pole active zostaje automatycznie wypełnione wartością 1, co jest indykacją, że wszystkie ważne dane zostały wypełnione i można korzystać z serwisu
- **premium (boolean)** : zawiera wartość 1 lub 0 (true lub false) i mówi nam, czy użytkownik ma lub nie ma wykupioną wersję premium aplikacji (nie dostępne w wersji prezentacyjnej)
- **description (text)** : swobodny opis profilu użytkownika, gdzie może on napisać o sobie, swoich pasjach itd.

Tabela na tą chwilę posiada 2 indeksy. Pierwszy z nich ustawiony na pola email i password przyspiesza logowanie, a drugi nałożony na id przyspieszaskorelowane z tabelą zapytania

Odwołania w relacjach:

- Users->Offert
- Users-> Liked\_Offert ->Offert
- Users-> Collaborators\_Offert-> Offert
- Users-> Users\_Technology-> Technology
- Users-> Users\_Media-> Media

## Tabela Offert

Offert	
id	int
category_id	int
name	varchar(60)
owner_id	int
description	TEXT
picture	blob

Zdjęcie 3. Tabela Offert

```
CREATE TABLE Offert (  
  id int PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  category_id int NOT NULL,  
  name varchar(60) NOT NULL,  
  owner_id int NOT NULL,  
  description TEXT NOT NULL,  
  picture blob  
);
```

Zdjęcie 4. Polecenia tworzące tabelę Offert

Tabela Offert służy do przechowywania części niezbędnych informacji o ofercie czyli inaczej projekcie, zamieszczonym w aplikacji celem znalezienia do niego współpracowników. Oto szczegółowy opis jej pól:

- **id (int)** : Pole klucza podstawowego
- **category\_id (int)** : Pole klucza obcego odwołującego się do Project\_category.id niezbędny do określania kategorii (platformy docelowej) projektu
- **owner\_id (int)** : Pole klucza obcego odwołującego się do Users.id służy do określenia właściciela oferty, czyli użytkownika zamieszczającego ogłoszenie
- **name (varchar)** : Przedstawia nazwę projektu

- **description (text)** : Pełny opis ogłoszenia
- **picture (blob)** : Opcjonalne zdjęcie

Odwołania w relacjach:

- Users->Offert
- Users-> Liked\_Offert ->Offert
- Users-> Collaborators\_Offert-> Offert
- Offert-> Offert\_Technology-> Technology
- Offert-> Project\_Category

## Tabela Media

Media	
id	int
name	VARCHAR(20)

Zdjęcie 5. Tabela Media

```
CREATE TABLE Media (  
  id int NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(20) NOT NULL UNIQUE,  
  INDEX ID_INDEX(id)  
);
```

Zdjęcie 6. Polecenia tworzące tabelę Media

Tabela Media służy do przechowywania nazw dostępnych portali społecznościowych, na których użytkownik aplikacji może prowadzić dodatkową działalność (np. swoja strona portfolio, Github). Wartości wpisane w tej tabeli nigdy nie ulegają zmianie, jedyne do czego używana jest ta tabela to skorelowanie linku z odpowiednią stroną dla przykładu „<https://github.com>” → Github. Zapobiega pewnej dowolności wstawiania takich odnośników. Oto jej pola:

- **id (int)** : Pole klucza podstawowego
- **name (varchar)** : nazwa portalu (np. LinkedIn, Github itp.)

Tabela posiada jeden indeks nałożony na jej klucz podstawowy. Przyspiesza to zapytania typu select mające na celu wybór wszystkich szczegółowych informacji o danym użytkowniku

Odwołania w relacjach:

- Users-> Users\_Media-> Media

## Tabela Technology

Technology	
id	int
name	varchar(30)
color	varchar(7)

Zdjęcie 7. Tabela Technology

```
CREATE TABLE Technology (  
  id int PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  name varchar(30) UNIQUE NOT NULL,  
  color varchar(7) UNIQUE,  
  INDEX ID_INDEX(id)  
);
```

Zdjęcie 8. Polecenia tworzące tabelę Technology

Tabela Technology zawiera niezmiennalną informację o wszystkich dostępnych do wymieniania na stronie technologiach programistycznych. Tabela dzięki odpowiednim relacjom służy do przedstawiania w jakich technologiach dany projekt ma być skonstruowany oraz jakie technologie zna użytkownik portalu. Dzięki korzystaniu z jednej finalnej tabeli możliwym jest filtracja pokazywania się proponowanych ofert w oparciu o umiejętności użytkownika. Jej pola to:

- **id (int)** : Pole klucza podstawowego
- **name (varchar)** : Nazwa technologii
- **color (varchar)** : Kolor do malowania miejsc gdzie pojawia się dana technologia. Jest zawsze związany z kolorystyką danego loga języka i zapisany w formacie szesnastkowym z „#” na początku

Odwołania w relacjach:

- Users-> Users\_Technology-> Technology
- Offert-> Offert\_Technology-> Technology

Tabela posiada jeden indeks nałożony na jej klucz podstawowy. Przyspiesza to zapytania typu select mające na celu wybór wszystkich szczegółowych informacji o danym użytkowniku lub ofercie

## Tabela Project\_category

Project_category	
id	int
name	varchar(20)
color	varchar(7)

```
CREATE TABLE Project_category (  
  id int PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  name varchar(20) UNIQUE NOT NULL,  
  color varchar(7) UNIQUE,  
  INDEX ID_INDEX(id)  
);
```

Zdjęcie 9. Tabela Project\_category

Zdjęcie 10. Polecenia tworzące tabelę

Tabela Technology zawiera niezmiennalną informację o wszystkich dostępnych do wybrania kategoriach ogłoszenia, czyli mówiąc inaczej platform docelowych takich jak np. internet, aplikacja okienkowa itp. Oto jej pola:

- **id (int)** : Pole klucza podstawowego
- **name (varchar)** : Nazwa kategorii
- **color (varchar)** : kolor stylistyczny
- 

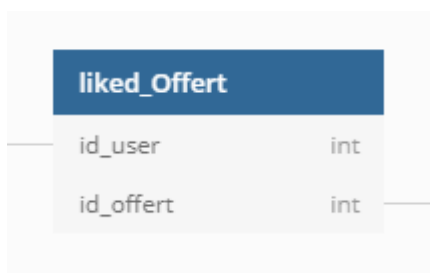
Tabela posiada jeden indeks nałożony na jej klucz podstawowy. Przyspiesza to zapytania typu select mające na celu wybór wszystkich szczegółowych informacji o konkretnej ofercie

Odwołania w relacjach:

- Offert->Project\_Category

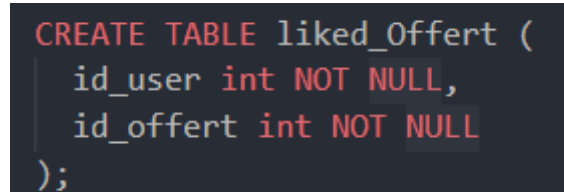


## Tabela liked\_Offert



liked_Offert	
id_user	int
id_offert	int

Zdjęcie 11. Tabela liked\_Offert



```
CREATE TABLE liked_Offert (  
  id_user int NOT NULL,  
  id_offert int NOT NULL  
);
```

Zdjęcie 12. Polecenia tworzące tabelę liked\_Offert

Tabela liked\_Offert przechowuje tymczasowe dane o polubieniach ofert przez użytkowników i polubieniach użytkowników przez właścicieli ogłoszeń. To właśnie tu zachodzi cały system „match”. Odpowiednia procedura do wstawiania rekordów w tą tabelę kontroluje, czy zachodzi takie obustronne polubienie. Jeśli tak, zostaje wysłana automatyczna wiadomość do jednego z zainteresowanych, dzięki czemu można utworzyć na tej podstawie czat pomiędzy nimi. Po takiej akceptacji 2 odpowiednie rekordy są kasowane z tabeli. W przeciwnym razie nic oprócz wstawienia odpowiedniego rekordu się nie dzieje. Oto pola tabeli:

- **id\_user (int)** : jest to zawsze id użytkownika, który nie jest właścicielem projektu (jeśli to właściciel przegląda potencjalnych współpracowników, to wstawiane są ich id, nie jego) a jest zainteresowany współpracą
- **id\_offert (int)** : id polubionej oferty

Odwołania w relacjach:

- Users-> liked\_offert-> Offert

## Tabela Chat

Chat	
id	int
id_recipient	int
id_sender	int
message	varchar(255)
time_sended	datetime

Zdjęcie 13. Tabela Chat

```
CREATE TABLE Chat (  
  id int PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  id_recipient int NOT NULL,  
  id_sender int NOT NULL,  
  message varchar(255) NOT NULL,  
  time_sended datetime NOT NULL,  
  INDEX GET_CHAT_DATA_INDEX(id_recipient, id_sender)  
);
```

Zdjęcie 14. Polecenia tworzące tabelę Chat

Tabela Chat jak sama nazwa wskazuje przechowuje wszystkie wysłane wiadomości w serwisie. Na jej podstawie można odtworzyć całą historię czatów dowolnego użytkownika. Oto jej pola:

- **id (int)** : Pole klucza podstawowego
- **id\_recipient (int)** : Id użytkownika otrzymującego wiadomość
- **id\_sender (int)** : Id użytkownika wysyłającego wiadomość
- **message (varchar)** : treść wiadomości z ograniczeniem do 255 znaków naraz
- **time\_sended (datetime)** : dokładna data wysłania wiadomości z „czułością” do sekundy. To pole jest autouzupełniane przez odpowiednią procedure funkcją now(). Na podstawie wartości określa się kolejność wysyłania

Tabela ma założony jeden indeks na dwa jej pola (id\_recipient, id\_sender), który znacznie przyspiesza wygenerowanie historii czatów użytkownika

Odwołania w relacjach:

- Chat->Users (2-krotne)

## Tabela collaborators\_offert

collabolators_Offert	
id_user	int
id_offert	int

Zdjęcie 15. Tabela collaborators\_Offert

```
CREATE TABLE collabolators_Offert (  
  id_user int NOT NULL,  
  id_offert int NOT NULL  
);
```

Zdjęcie 16. Polecenia tworzące tabelę

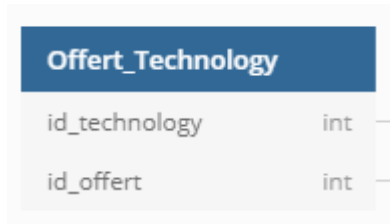
Tabela collabolators\_offert jest tabelą pośrednią, służącą do przechowywania informacji o współpracownikach pracujących razem nad jednym projektem. W tabeli nie znajdują się informacje o właścicielach ofert. Jej pola to:

- **id\_user (int)** : id współpracownika
- **id\_offert (int)** : id projektu

Odwołania w relacjach:

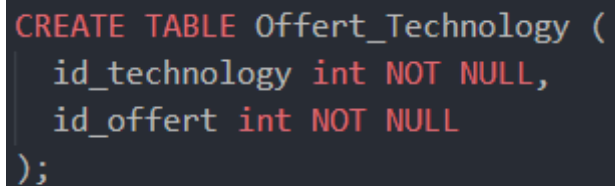
- Users->collaborators\_offert-> Offert

## Tabela Offert\_technology



Offert_Technology	
id_technology	int
id_offert	int

Zdjęcie 17. Tabela Offert\_technology



```
CREATE TABLE Offert_Technology (  
  id_technology int NOT NULL,  
  id_offert int NOT NULL  
);
```

Zdjęcie 18. Polecenia tworzące tabelę

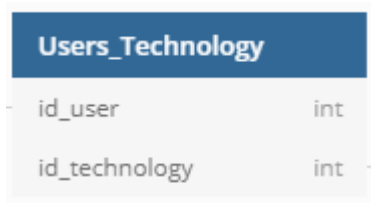
Tabela Offert\_technology to tabela pośrednia za pomocą której odczytywane są technologie wykorzystywane w danym projekcie. Jej pola to odpowiednio:

- **id\_technology (int)** : id technologii (języka programowania)
- **id\_offert (int)** : id oferty

Odwołania w relacjach:

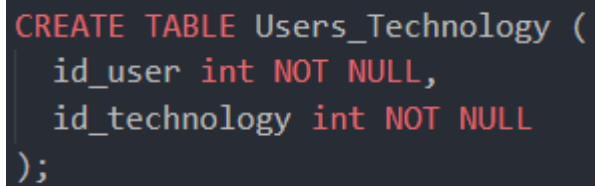
- Offert-> Offert\_Technology-> Technology

## Tabela Users\_technology



Users_Technology	
id_user	int
id_technology	int

Zdjęcie 19. Tabela Users\_technology



```
CREATE TABLE Users_Technology (  
  id_user int NOT NULL,  
  id_technology int NOT NULL  
);
```

Zdjęcie 20. Polecenia tworzące tabelę

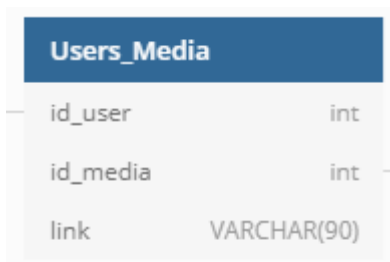
Tabela Users\_technology to tabela pośrednia za pomocą której odczytywane są języki programowania poznane przez danego użytkownika. Jej pola to:

- **id\_technology (int)** : id technologii (języka programowania)
- **id\_user (int)** : id użytkownika

Odwołania w relacjach:

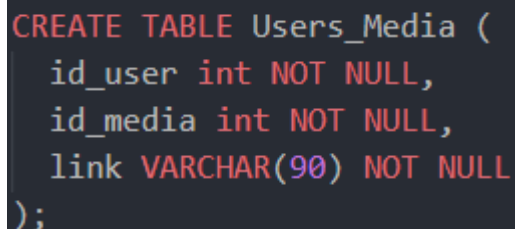
- Offert-> Offert\_Technology-> Technology

## Tabela Users\_Media



Users_Media	
id_user	int
id_media	int
link	VARCHAR(90)

Zdjęcie 21. Tabela Users\_Media



```
CREATE TABLE Users_Media (  
  id_user int NOT NULL,  
  id_media int NOT NULL,  
  link VARCHAR(90) NOT NULL  
);
```

Zdjęcie 22. Polecenia tworzące tabelę Users\_Media

Tabela Users\_Media to tabela pośrednia przechowująca informację o linkach użytkownika do poszczególnych portali społecznościowych np. Facebook. Te dane są widoczne na profilu. Oto pola tej tabeli:

- **id\_user (int)** : id użytkownika
- **id\_media (int)** : id portalu społecznościowego
- **link (varchar)** : bezpośredni link

Odwołania w relacjach:

- Users-> Users\_Media-> Media



## Procedury

Procedury są nazwanym blokiem kodu w SQL, który można używać wiele razy, podając tylko ich nazwy oraz odpowiednie parametry. W przeciwieństwie do funkcji, nie muszą zwracać żadnego wyniku. W tej bazie danych, są one głównie używane do ułatwiania wstawiania danych ze strony backendu. Zapewniają bezpieczeństwo danych oraz gwarantują ich spójność, a czasem zawierają dodatkową logikę. Na tą chwilę w bazie Collaborate istnieje 7 odmiennych procedur. W przyszłości może powstać ich więcej. Ciągi poleceń w każdej procedurze są dodatkowo opięte transakcjami

## Procedura login\_user

```
delimiter //  
CREATE PROCEDURE login_user (IN email1 varchar(50), IN password1 varchar(30))  
BEGIN  
  
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
    START TRANSACTION;  
  
    SELECT id,name,surname,active,premium,email,description FROM users WHERE email=email1 AND p  
    COMMIT;  
END//  
delimiter ;
```

Zdjęcie 23. Polecenia tworzące procedurę login\_user

Prosty kod mający na celu sprawdzenie, czy użytkownik o podanym haśle i emailu istnieje w bazie danych. Jeżeli tak, jego id zostaje wyciągnięte Selectem i zapisane w sesji PHP. Ustawiono tutaj jeden z najniższych poziomów transakcji bo nie zachodzi potrzeba większego bezpieczeństwa zapytań

Parametry:

- **email1 (varchar)** : Wpisany w formularz email
- **password1 (varchar)** : Wpisane w formularz hasło

## Procedura insert\_new\_user

```
delimiter //  
CREATE PROCEDURE insert_new_user (IN email1 varchar(50), IN name1 varchar(20), IN surname1 varchar(70), IN password1 varchar(30))  
BEGIN  
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
    START TRANSACTION;  
    INSERT INTO users(email,name,surname,password) VALUES  
    (email1, name1, surname1,password1);  
    COMMIT;  
END//  
delimiter ;
```

Zdjęcie 24. Polecenia tworzące procedurę insert\_new\_user

Procedura insert\_new\_user jest jedną z 2 podstawowych funkcji wstawiających dane o użytkowniku. W tym przypadku, są to absolutnie niezbędne informacje, wymagane aby wejść do aplikacji. Są one podawane przy formularzu rejestracji. Dla bezpieczeństwa, jest tutaj zastosowany najwyższy poziom izolacji transakcji żeby zagwarantować poprawne dodawanie auto inkrementującego się id użytkownika

Parametry:

- email1 (varchar) : Email nowego użytkownika
- name1 (varchar) : imię użytkownika
- surname1 (varchar) : nazwisko użytkownika
- password1 (varchar) : hasło nowego użytkownika

## Procedura insert\_further\_user\_data

```
delimiter //
CREATE PROCEDURE insert_further_user_data(

    IN id_user_inserting INT, IN description1 TEXT,
    IN email_changed varchar(50), IN password_changed varchar(30),
    IN media_names TEXT, IN links_to_media TEXT, IN technology_list TEXT

)
BEGIN
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;

    UPDATE users SET description=description1 WHERE id=id_user_inserting;

    IF email_changed IS NOT NULL THEN
        UPDATE users set email=email_changed WHERE id=id_user_inserting;
    END IF;

    IF password_changed IS NOT NULL THEN
        UPDATE users set password=password_changed WHERE id=id_user_inserting;
    END IF;

    IF description1 IS NOT NULL THEN
        UPDATE users set description=description1 WHERE id=id_user_inserting;
    END IF;

    DELETE FROM users_technology WHERE id_user=id_user_inserting;

    iterator:LOOP
```

Zdjęcie 25. Część poleceń tworzących procedurę insert\_further\_user\_data

insert\_further\_user\_data jest jedną z najbardziej skomplikowanych procedur bazy. Odpowiada za wstawianie bardziej szczegółowych, opcjonalnych dla użytkownika aplikacji danych użytkownika. Wstawiają się tutaj takie dane jak : opis konta, zdjęcie profilowe, poznane technologie, linki do kont społecznościowych. Oprócz tego, jest tutaj możliwość zmiany swojego hasła jak i emaila. insert\_further\_user\_data wstawia dane nie tylko do tabeli users, ale też do skorelowanych z nią tabel wiele do wielu, dzięki czemu nie istnieje ryzyko niespójności danych

### Parametry:

- **id\_user\_inserting (int)** : id obecne zalogowanego w sesji użytkownika
- **description1 (text)** : opis konta, gdy wstawi się tutaj wartość NULL, procedura zrozumie, że nic w nim nie zmieniono
- **email\_changed (varchar)** : email konta, gdy wstawi się tutaj wartość NULL, procedura zrozumie, że nic w nim nie zmieniono
- **password\_changed (varchar)** : hasło konta, gdy wstawi się tutaj wartość NULL, procedura zrozumie, że nic w nim nie zmieniono
- **media\_names (text)** : ciąg znaków reprezentujący możliwe do odwołania się witryny społecznościowe. Należy je podać, oddzielając przecinkiem bez żadnych spacji np. „Github,Facebook”
- **link\_to\_media (text)** : ciąg znaków zawierający rzeczywiste linki do wybranych witryn. Należy je podawać analogicznie dla media\_names z odpowiednią kolejnością np.  
„<https://github.com/123>,<https://facebook.com/123>”
- **technology\_list (text)** : lista technologii znanych przez użytkownika. Podaje się je w jednym ciągu znaków, oddzielając przecinkiem bez spacji np.  
„Python,HTML,CSS”

## Procedura insert\_collaborator

```
delimiter //
CREATE PROCEDURE insert_collaborator (IN id_user_inserting INT, IN id_user_inserted INT, IN id_offert_destination INT)
BEGIN
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;

    SET @check_ownership = (SELECT count(offert.id) FROM offert INNER JOIN users ON offert.owner_id = users.id WHERE offert.owner_id = id_user_inserting);

    IF @check_ownership = 1 THEN
        INSERT INTO collaborators_Offert(id_user, id_offert) VALUES (id_user_inserted, id_offert_destination);
        SELECT 'Done' AS 'message';
    ELSE
        SELECT 'Error : Not an owner' AS 'message';
    END IF;

    COMMIT;
END//
delimiter ;
```

Zdjęcie 26. Polecenia tworzące insert\_collaborator

Ten blok kodu wstawia nowego współnika do projektu. Dodatkowo, mamy tutaj sprawdzenie, czy proszący o dołączenie kogoś nowego użytkownik jest do tego uprawniony, czyli czy jest właścicielem danej oferty. Jeśli nie, nie może dodać do niego współpracownika. O takim błędzie informuje nas treść końcowego selecta

Parametry:

- **id\_user\_inserting (int)** : id obecnie zalogowanego w sesji użytkownika, domyślnie właściciela danej oferty
- **id\_user\_inserted (int)** : id współpracownika
- **id\_offert\_destination (int)** : id oferty

## Procedura insert\_match

```
delimiter //
CREATE PROCEDURE insert_match (IN id_user_inserted INT, IN id_offert1 INT)
BEGIN
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

    SET @setmatch = (SELECT count(*) FROM liked_offert WHERE id_offert=id_offert1 AND id_user=id_user_inserted);

    IF @setmatch < 2 THEN
        INSERT INTO liked_offert(id_user, id_offert) VALUES(id_user_inserted,id_offert1);
    END IF;

    IF @setmatch + 1 = 2 THEN
        SET @owner = (SELECT id FROM users INNER JOIN offert ON users.id = offert.owner_id WHERE offert.id = id_offert1);
        CALL insert_message(id_user_inserted, @owner, "Właśnie dostaliście matcha! Super! Teraz możecie do siebie pisać. Ta wiadomość została wygenerowana");
        SELECT 'Match' AS 'message';
    ELSE
        SELECT 'Not match' AS 'message';
    END IF;

    COMMIT;
END//
delimiter ;
```

Zdjęcie 27. Polecenia tworzące insert\_match

Procedura insert\_match jest odpowiedzialna za wstawianie polubień ofert lub polubień potencjalnych współpracowników, których chcielibyśmy zaprosić do swojego projektu. Wywołanie tego kodu wstawia rekord do tabelki liked\_offert i od razu sprawdza, czy nastąpiło obopólne polubienie. Jeśli tak, system wysyła automatyczną wiadomość do zainteresowanych, tak aby mogli między sobą pisać na czacie. Z powodu potrzeby użycia funkcji count i w celu uniknięcia nieprawidłowych odczytów, użyto tutaj najwyższego poziomu izolacji transakcji

Parametry:

- **id\_user\_inserted (int)** : id użytkownika, który polubił ofertę, lub id użytkownika upatrzonego na współpracownika
- **id\_offert1 (int)** : id oferty docelowej

## Procedura insert\_new\_offert

```
delimiter //
CREATE PROCEDURE insert_new_offert (
    IN id_user_inserting INT, IN category_name VARCHAR(20),
    IN offert_name VARCHAR(60), IN offert_picture1 blob,
    IN offert_description TEXT, IN technology_list TEXT
)
BEGIN

    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;

    SET @category_id = (SELECT id FROM project_category WHERE name=category_name);

    IF @category_id IS NULL THEN
        INSERT INTO offert(name, description, owner_id, category_id) VALUES
            (offert_name, offert_description, id_user_inserting, 6);
    ELSE
        INSERT INTO offert(name, description, owner_id, category_id, picture) VALUES
            (offert_name, offert_description, id_user_inserting, @category_id, offert_picture1);
    END IF;

    SET @current_id_offert = (SELECT LAST_INSERT_ID());
```

Zdjęcie 28. Część poleceń tworzących insert\_new\_offert

Ta procedura wstawia do bazy danych nową ofertę (projekt). Oprócz zapełniania danych w tabeli offert, dodaje też wybrane przez użytkownika technologie używane w ofercie. Z tego powodu, niezbędne jest użycie funkcji SQL last\_inserted\_id, a to wymaga dla bezpieczeństwa ustawienia najwyższego poziomu izolacji transakcji



Parametry:

- `id_user_inserting (int)` : id użytkownika wstawiającego ofertę
- `category_name (varchar)` : nazwa kategorii projektu np. Web, Desktop itp.
- `offert_name (varchar)` : nazwa oferty
- `offert_picture1 (blob)` : obrazek lub grafika projektu
- `offert_description (text)` : opis oferty
- `technology_list (text)` : ciąg znaków zawierający używane przez projekt technologie, oddzielone przecinkiem bez spacji np. „Python,HTML,CSS”

## Procedura insert\_message

```
delimiter //
CREATE PROCEDURE insert_message(IN id_sender1 INT, IN id_recipient1 INT, IN message1 VARCHAR(255))
BEGIN
    START TRANSACTION;
    INSERT INTO chat(id_recipient, id_sender, message, time_sended) VALUES (id_recipient1, id_sender1, message1, NOW());
    COMMIT;
END//
delimiter ;
```

Zdjęcie 29. Polecenia tworzące procedure insert\_message

insert\_message jest wykorzystywany do wstawiania rekordów, przedstawiających wiadomości użytkowników aplikacji. Wystarczy podać id nadawcy i id odbiorcy. Serwer sam doda do rekordu automatycznie datę wysłania wiadomości, aby uniknąć niezgodności formatów daty i ułatwić wszystko na backendzie

Parametry:

- **id\_sender1 (int)** : id nadawcy
- **id\_recipient1 (int)** : id odbiorcy

# Zabezpieczenia

Baza danych Collaborate zostają zaprojektowana tak żeby obsługiwać wiele użytkowników naraz. To i wiele innych czynników wymusza nakładanie pewnych zabezpieczeń w celu uniknięcia np. usunięcia niektórych rekordów lub naruszenia spójności danych. Do takich celów wykorzystano:

- **Klucze podstawowe i obce** : zapewniają podstawową formę kontroli spójności danych
- **Odpowiednio nałożone klauzule on delete** : w przypadku usunięcia np. oferty lub użytkownika automatycznie czyszczą skorelowane z usuniętym rekordem dane w innych tabelach
- **Dodatkowa logika procedur** : niektóre procedury posiadają dodatkowe sprawdzenia poprawności wpisanych parametrów, co może w skrajnych warunkach ochronić przed niespodziewalnymi działaniami

- **Stosowanie poziomów izolacji w transakcjach** : w zależności od poziomu niwelują one problemy budnych odczytów, odczytów widmo, zduplikowanych odczytów itp. Niezbędne, kiedy wielu użytkowników działa na tych samych danych