Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

System informatyczny prowadzenia kursów online

Damian Duy

Projekt z przedmiotu bazy danych na kierunku informatyka profil ogólnoakademicki na Uniwersytecie Gdańskim.

Gdańsk

24 maja 2020

Spis treści

1 Wprowadzenie

Baza danych przeznaczona jest dla systemu informatycznego strony internetowej prowadzącej kursy online. Będzie ona służyć do przechowywania i zarządzania informacjami dotyczącymi jej zawartości. Celem jej utworzenia jest obsługa i zarządzanie użytkownikami, biorącymi udział w kursach, zdobywającymi za nie punkty oraz certyfikaty, a także samymi kursami, podzielonymi na mniejsze encje, takie jak moduły, lekcje i w końcu na pojedyncze zadania.
Pojęcia używane w dalszej części dokumentacji:
Encja- jednostka będaca rozróżnialnym od innych, istniejącym bytem.
Tabela- podstawowy obiekt bazy danych podzielony na wiersze(nazywane też *krotkami*, *rekordami*) i kolumny. W wierszach znajdują się dane, natomiast kolumny odpowiadają za wskazanie atrybutów, których wartości są w nich przechowywane.
Pole- komórka tabeli.
Normalizacja- doprowadzenie schematu bazy danych do zgodności z przyjętymi normami, które mają na celu ograniczenie nadmiarowości danych.
Klucz sztuczny- identyfikator tabeli, który nie jest czymś naturalnie występującym w rzeczywistości (w odróżnieniu od *klucza naturalnego*).
Klucz prosty- klucz główny (identyfikujący tabelę) będący reprezentowany przez jedną kolumnę.
Tabela słownikowa- tabela będąca słownikiem, która zapobiega nadmiernemu wpisywaniu tych samych danych

2 Opis projektu

Organizacja zajmująca się tworzeniem i prowadzeniem kursów online potrzebuje bazy danych, która będzie zawierała informacje o dostępnych kursach (podzielonych na moduły), lekcjach, zadaniach, użytkownikach (oraz danych z nimi związanych), a także uzyskanych przez nich rezultatów w postaci punktów. W projekcie musiała być również przewidziana możliwość uzyskania przez użytkowników certyfikatów przyznawanych za ukończenie danego kursu. Wymagana też jest możliwość zapamiętywania historii wysyłanych przez użytkownika zadań, które mają swój określony typ. Jednym z założeń jest również umożliwienie zapamiętywania historii dokonywanych w kursach zmian.
Założenia projektu dotyczące użytkowników:
- Użytkownik musi być opisywany przez atrybuty takie jak jego pseudonim, imię, nazwisko, nazwa państwa oraz miasta, z którego pochodzi oraz data utworzenia przez niego konta.
- Użytkownik musi posiadać pseudonim (username) oraz data utworzenia przez niego konta musi być znana (pozostałe atrybuty są opcjonalne).
Założenia projektu dotyczące kursów:
- Kursy składają się z modułów, które składają się z lekcji, które składają się z zadań.
- Kursy, moduły, lekcje oraz zadania są identyfikowane po jednoznacznym identyfikatorze.
- Kurs ma właściciela, którym jest jeden, konkretny użytkownik.
- Użytkownik może być zapisanym do wielu kursów.
- W jednym kursie może być też wielu użytkowników.
Dodatkowe założenia dotyczące zadań:
- Każde zadanie ma przyporządkowany do siebie typ, który jest opisany w postaci teksto-

wej.
- Trzymana jest historia przesłanych przez użytkowników zadań, a w niej powinny znajdować się informacje, jaki użytkownik przesłał, które zadanie, a także, jakiego dnia oraz o której godzinie.
Założenia dotyczące historii dokonywania zmian:
- Zapisywana jest osobno, historia dokonywania zmian w kursach, modułach, lekcjach oraz zadaniach.
- Zapisywana jest ona pod postacią opisu tekstowego.
Założenia dotyczące certyfikatów:
- Certyfikat jest identyfikowalny po jednoznacznym identyfikatorze.
- Certyfikat jest przypisany do konkretnego kursu.
- Jeden certyfikat może być uzyskany w jednym kursie.
- Jeden użytkownik może mieć wiele certyfikatów, a także jeden certyfikat może przynależeć do wielu użytkowników (założenie jest, że certyfikaty są przypisane do kursu, istnieją na zasadzie pewnego, konkretnego wzoru, do którego dopiero na końcu dodaje się imię i nazwisko użytkownika, co dzieje się już poza bazą danych. Nie każdy certyfikat jest imienny, a w samym systemie użytownik nie ma obowiązku podawania swoich danych).
- Przy otrzymaniu przez użytkownika certyfikatu, powinna być także widoczna data jego otrzymania.
Założenia dotyczące punktów:
- W kursie użytkownicy zbierają punkty.
- Informacje dotyczące punktów uzyskanych z konkretnego kursu, modułu, lekcji i zadań powinny być przedstawiane osobno.
- W przynależności użytkownika do danego kursu, powinno znajdować się odniesienie do uzyskanych przez niego punktów.

2.1 Potencjalne grupy użytkowników

- Administrator bazy danych - specjalista z dziedziny IT posiadający pełny dostęp do bazy danych. Jest odpowiedzialny za jej bezpieczeństwo i wydajność. Może on zarzadząć innymi użytkownikami, dodawać ich oraz nadawać i odbierać im uprawnienia.

- Projektant - projektuje schemat bazy danych, ustala jakie dane mają być przechowywane w bazie.

- Programista - tworzy interfejs, aplikację umożliwiający zwyczajnym użytkownikom dostęp do bazy.

- Zwykły użytkownik - korzystając z gotowego interfejsu wprowadza i edytuje on informacje w bazie danych. Ma on ograniczony dostęp do je funkcjonalności. Zwykłych użytkowników można podzielić na użytkowników naiwnych i doświadczonych. Użytkownicy naiwni najczęściej używają formularzy. Użytkownicy doświadczeni są w stanie wykonywać bardziej skomplikowane operacje przy użyciu zapytań.

[1]

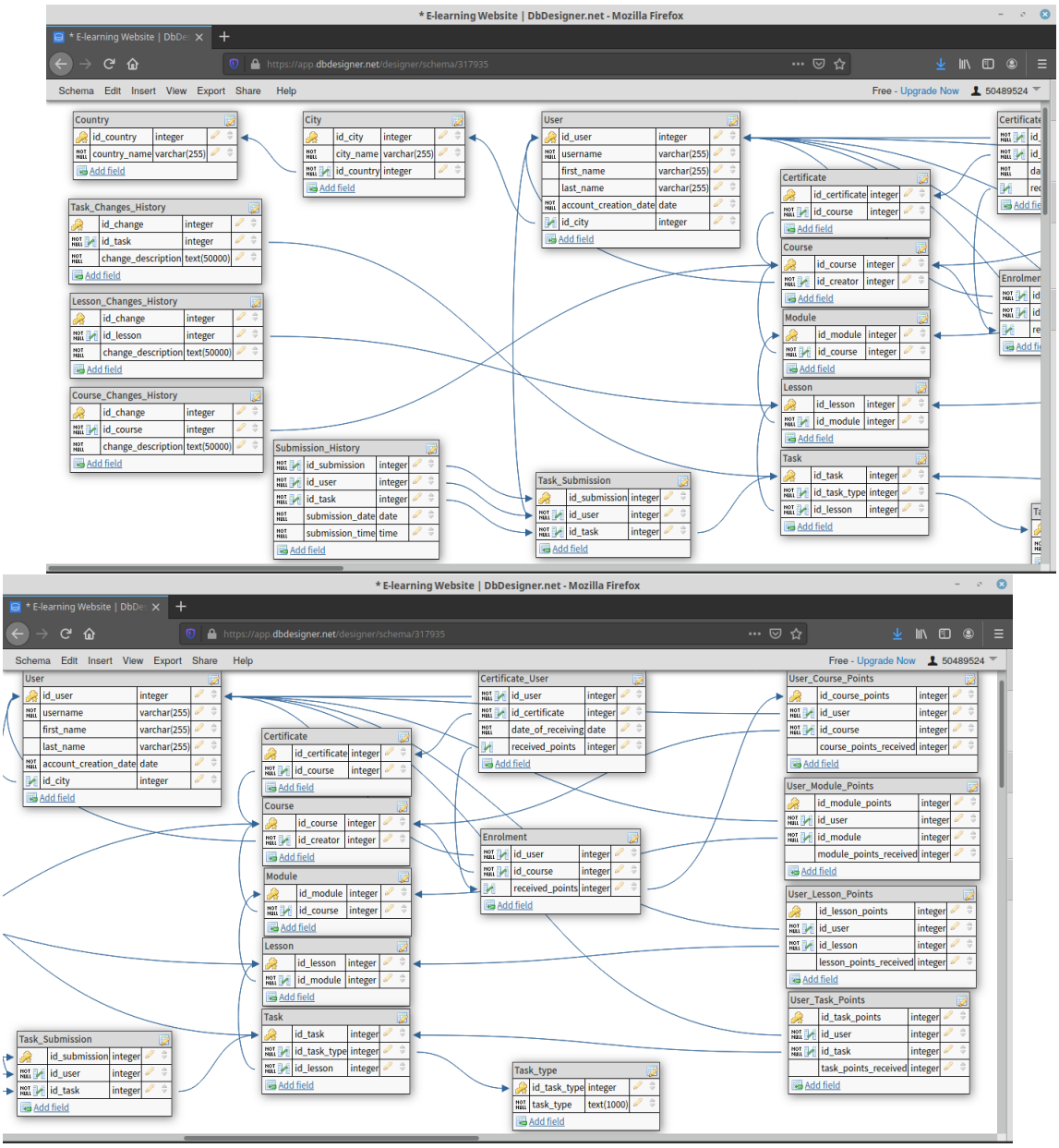
2.2 Wymagania funkcjonalne

Baza danych składa się z 20 tabel.
Rodzaje przechowywanych danych obejmują liczby całkowite, napisy, datę i czas. Z założenia przy projektowaniu bazy klucze podstawowe są kluczami sztucznymi oraz autoinkrementują się. Schemat bazy danych spełnia założenia normalizacji danych. Dane znajdują się w pierwszej postaci normalnej. Wartości w kolumnach są atomowe (niepodzielne). Przykładowo, potencjalne pole *localization* zostało podzielone osobno na *city* oraz *country*.
Tabele znajdują się w drugiej postaci normalnej. Klucze podstawowe są proste. Wynika to również z samego założenia przy projektowaniu bazy danych, że jako klucze podstawowe będą zawsze używane klucze sztuczne, na przykład *id user*.
Tabele spełniają również zasady trzeciej postaci normalnej. Nie występuje w niej redundacja z uwagi na utworzenie tabel łącznikowych, realizujących relację wiele do wielu, którymi są tabele *Task Submission*, *Certificate User*, *Enrolment*, *User Course Points*, *User Module Points*, *User Lesson Points*, *User Task Points*, a także z uwagi na obecność tabel słownikowych: *Country*, *City*,*Task Type*.
Z innych założeń realizujących zadania bazy danych warte wspomnienia jest ograniczenie UNIQUE na *id_course* w tabeli *certificate* umożliwiające realizację zadania dotyczącego tego, że relacja certyfikat-kurs ma być relacją jeden do jednego.
Niektóre atrybuty takie jak *first_name* i *last_name* z tabeli *users* mają cechę ALLOW NULL, co umożliwia to, że nie muszą być one podane przez użytkownika (są opcjonalne).

2.3 Wymagania niefunkcjonalne

Wybranym systemem zarządzania relacyjną bazą danych jest PostgreSQL. Narzędziem do współpracy z bazą danych jest terminalowy klient Postgresa - psql. Zostało ono wybrane z uwagi na mały rozmiar, dużą dostępność i łatwość w użyciu. Wadą wybranego narzędzie jest czasami niewygodny interfejs, który nie jest przyjazny dla osób początkujących. Sam PostgreSQL jest rozwijanym od ponad 30 lat darmowym systemem bazodanowym, typu open source. Jego zaletami jest łatwa rozszerzalność i elastyczność zbudowanych w nim baz danych. Dodatkowym atutem wpływającym na jego wybór jest to, że działa on na wszystkich najważniejszych systemach operacyjnych oraz w odróżnieniu do np. SQL Server posiada wsparcie dla dużej ilości języków programowania. Używa on rozszerzonego języka SQL, którego dużo funkcjonalności zostało przeniesionych z oryginalnego języka zapytań Postgresa jakim był PostQuel. [2]

2.4 Diagram związków encji



3 Przykłady realizacji bazy danych

Przykłady realizacji bazy danych znajdują się w postaci przedstawienia w jaki sposób są w niej reprezentowane encje oraz ich atrybuty. Poniżej pokazane są, kluczowe do odpowiedniego realizowania założeń jakie miała spełniać baza, tabele. Są to dwie tabele opisujące najważniejsze encje w projekcie, jakimi są użytkownicy (*users table*) oraz kursy (*course table*). Inne tabele opisujące pozostałe encje, ważne przy realizacji założeń (takie jak zadania, lekcje czy moduły) tworzone były w podobny sposób. Pozostałe dwie przedstawione jako przykłady poniżej tabele realizują zależność wiele do wielu. Jest to tabela *Enrolment table* obsługująca przynależność uzytkowników do kursów, oraz tabela *User Course Points table* pokazująca zdobyte przez użytkowników punkty. Są one kluczowe do realizacji założeń jakie postawiła organizacja zlecająca wykonanie projektu.
Przykładowe zapytania realizujące bazę danych pisane są w zgodnym z Postgresem języku, będącym wariancją języka SQL. Ich dokładny opis znajduje się poniżej. Oprócz standardowych zapytań, znajdują się tam również przykłady stworzenia i użycia view, procedury czy wyzwalacza.
Inne przykłady zapytań realizujących funkcjonalność bazy danych:
- Pokazywanie rankingów/list
- Sortowanie użytkowników według różnych kryteriów.
- Przynależność użytkownika do danych kursów.
- Certyfikaty posiadane przez użytkownika.
- Historia wysyłania zadań przez użytkownika.
- Historia zmian w danym kurse.
- Wykaz modułów/lekcji/zadań danego kursu.

3.1 Przykłady zawartości najważniejszych tabel

```
Users Table
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id_user         | integer      | NO   | PK  | None    |       |
| first_name      | varchar(255) | YES  |     | None    |       |
| last_name       | varchar(255) | YES  |     | None    |       |
| account_cre_date| date         | NO   |     | None    |       |
| id_city         | integer      | YES  | FK  | None    |       |
+-----------------+--------------+------+-----+---------+-------+
```

```
Enrolment Table
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| id_user        | integer      | NO   | PK  | None    |       |
| id_course      | integer      | NO   | FK  | None    |       |
| received_points| integer      | YES  | FK  | None    |       |
+----------------+--------------+------+-----+---------+-------+
```

```
Course Table
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| id_course     | integer     | NO   | PK  | None    |       |
| id_creator    | integer     | NO   | FK  | None    |       |
+---------------+-------------+------+-----+---------+-------+
```

```
User_Course_Points
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| id_course_points| integer    | NO   | PK  | None    |       |
| id_user        | integer     | NO   | FK  | None    |       |
| id_course      | integer     | NO   | FK  | None    |       |
| course_points_re| integer    | YES  |     | None    |       |
+----------------+-------------+------+-----+---------+-------+
```

3.2 Przykłady kilku zapytań i ich wyników

1. Zapytanie wyświetlające użytkowników po ich username, wraz z identyfikatorem kursu oraz uzyskanymi w danym kursie punktami, w kolejności malejącej. Zapytanie jest negatywne (sprawdza warunek IS NOT NULL dotyczący uzyskanych punktów). Przedstawia również użycie INNER JOIN, aby połączyć wyniki dwóch tabel, oraz użycie klauzuli ORDER BY DESC.

```
01 | SELECT username, id_course, course_points_received FROM
     user_course_points AS P INNER JOIN users AS U ON U.id_user = P.
     id_user WHERE course_points_received IS NOT NULL ORDER BY
     course_points_received DESC;
```

Wynik zapytania:

```
 username | id_course | course_points_received
----------+-----------+------------------------
 Jer23    |         1 |                     20
 kiki     |         1 |                     15
 mike     |         1 |                     10
 maro     |         2 |                      5
```

2. Zapytanie wyświetlające jeden (pierwszy) kurs, którego właścicielem jest użytkownik z przykładowym *id_ user* równym 1. Użyte zostały w nim klauzule JOIN oraz LIMIT.

```
01 | SELECT id_course, username, first_name, last_name FROM users JOIN course
       ON id_user = id_creator WHERE id_user = 1 ORDER BY id_course LIMIT
     1;
```

Wynik zapytnia:

```
 id_course | username | first_name | last_name
-----------+----------+------------+-----------
         1 | kiki     | Julian     | Kowal
```

3. Zapytanie zwracające username, imię, nazwisko, nazwę państwa oraz nazwę miasta użytkowników będących zapisanymi do jakiegokolwiek kursu oraz mieszkających w mieście zaczynającym się na literę 'L'. W zapytaniu występuje porównanie do wzorcu napisu. W zapytaniu użyta została również klauzula INNER JOIN oraz zapytanie zagnieżdżone.

```
01 | SELECT username, first_name, last_name, country_name, city_name FROM
     users as U INNER JOIN city as C ON U.id_city = C.id_city INNER JOIN
     country as Co ON Co.id_country = C.id_country WHERE C.city_name LIKE
     'L%' AND U.id_user IN (SELECT id_user FROM Enrolment);
```

Wynik zapytania:

```
username | first_name | last_name |  country_name   | city_name
---------+------------+-----------+-----------------+-----------
 Jer23   | Jerry      | Blue      | Wielka Brytania | Londyn
 mike    | Micheal    | Gray      | Wielka Brytania | Londyn
```

4. Zapytanie wyświetlające moduły/lekcje/zadania danego kursu. Użyto klauzuli INNER JOIN.

```
01 | SELECT C.id_course, M.id_module, L.id_lesson, T.id_task FROM course AS C
      INNER JOIN module AS M ON C.id_course = M.id_course INNER JOIN
     lesson AS L ON M.id_module = L.id_module INNER JOIN task AS T ON L.
     id_lesson = T.id_lesson;
```

Wynik zapytania:

```
 id_course | id_module | id_lesson | id_task
-----------+-----------+-----------+---------
         1 |         1 |         1 |       1
         2 |         2 |         2 |       2
         3 |         3 |         3 |       3
         4 |         4 |         4 |       4
```

5. Utworzenie VIEW *tasks_before_quarantine* pokazującego zadania przesłane przez użytkowników przed datą 15 marca 2020 roku.

```
01 | CREATE VIEW tasks_before_quarantine AS
02 | SELECT * FROM Submission_History
03 | WHERE submission_date < '2020-03-15';
```

VIEW jest usuwany za pomocą polecenia:

```
01 | DROP VIEW IF EXISTS tasks_before_quarantine;
```

6. Utworzenie procedury usuwającej nazwiska dwóch użytkowników.

```
CREATE OR REPLACE PROCEDURE usun_nazw(INT, INT)
LANGUAGE plpgsql
AS $$
BEGIN

    UPDATE users
    SET last_name = NULL
    WHERE id_user = $1;


    UPDATE users
    SET last_name = NULL
    WHERE id = $2;

    COMMIT;
END;
$$;
```

Wywołanie procedury następuje za pomocą polecenia:

```
CALL usun_nazw(1,2);
```

7. Utworzenie i wywołanie tiggera. Trigger jest wywoływany, gdy nastąpi dodanie wartości do tabeli *Task_Submission*. Wtedy też zostanie automatycznie dodany wpis do tabeli *Submission_History* zawierający dodane zadanie oraz date i czas, w którym zostało ono dodane.

```
CREATE OR REPLACE FUNCTION sub_his() RETURNS TRIGGER AS $example_table$
   BEGIN
      INSERT INTO Submission_History(id_submission, id_user, id_task,
   submission_date, submission_time) VALUES (new.id_submission, new.
   id_user, new.id_task, current_date, current_timestamp);
      RETURN NEW;
   END;
$example_table$ LANGUAGE plpgsql;

CREATE TRIGGER his_trigger AFTER INSERT ON Task_Submission
FOR EACH ROW EXECUTE PROCEDURE sub_his();
```

Wywoływanie triggera następuje poprzez przykładowo:

```
INSERT INTO Task_Submission(id_user, id_task) VALUES (1,4);
```

Tabela *Submission_History* przed powyższym poleceniem:

```
 id_submission | id_user | id_task | submission_date | submission_time
---------------+---------+---------+-----------------+-----------------
             1 |       1 |       1 | 2021-05-06      | 20:34:00
             2 |       2 |       1 | 2021-05-06      | 23:34:00
             3 |       3 |       2 | 2021-08-06      | 17:28:00
             4 |       4 |       1 | 2022-08-06      | 19:28:00
```

Tabela *Submission_History* po powyższym poleceniu:

```
 id_submission | id_user | id_task | submission_date | submission_time
---------------+---------+---------+-----------------+-----------------
             1 |       1 |       1 | 2021-05-06      | 20:34:00
             2 |       2 |       1 | 2021-05-06      | 23:34:00
             3 |       3 |       2 | 2021-08-06      | 17:28:00
             4 |       4 |       1 | 2022-08-06      | 19:28:00
             5 |       1 |       4 | 2020-05-20      | 19:22:23.423605
```

[3]

Literatura

[1] Doktor Andrzej Borzyszkowski, *Wykłady z baz danych*, 2020.

[2] The PostgreSQL Global Development Group, *Oficjalna dokumentacja postgresql*, 2020.

[3] PostgreSQL Tutorial Website, *Postgresql tutorial*, 2020.

[2] [3] [1]