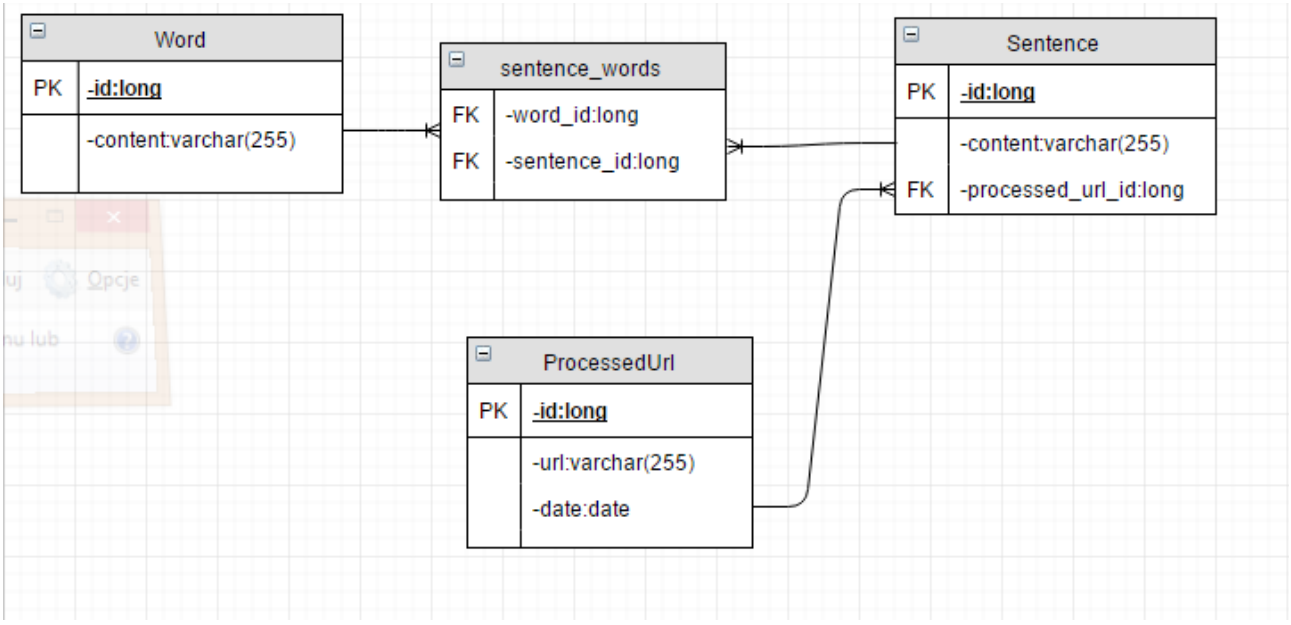
Sprawozdanie z zadania domowego.

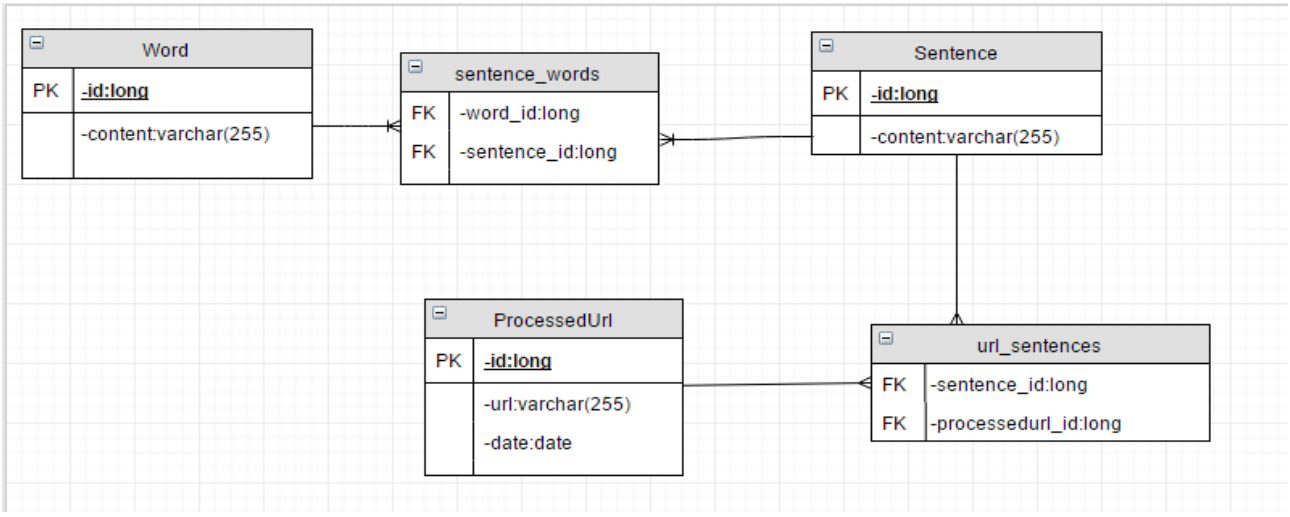
Wstęp:
W ramach ulepszenia projektu, dodałem dwie dodatkowe modyfikacje w kodzie i modelu: zmieniłem schemat bazy danych z zadania, a także dołożyłem starań, by usunąć wszelkiego rodzaju nadmiarowe informacje w niej przechowywane.

1. Stan przed rozpoczęciem:
Po zrobieniu wszystkich punktów obowiązkowych konspektu, napisałem także kod realizujący podpunkty 6 i 7 (zadania "z gwiazdkami"). Schemat bazy wyglądał w nstępujący sposób:



Zauważyłem jednak, że takie podejście ma jedną istotną wadę logiczną: może istnotnie zajść sytuacja, w której jedno zdanie będzie należeć do więcej niż jednego url-a. Przykładowo, w przypadku stron Wikipedii, w ciągach słów akceptowanych jako zdania przez "algorytm" w projekcie, mogą się pojawić (i generalnie pojawiają się) sformułowania w stylu "o wikipedii", "wersja dla urządzeń mobilnych" itp., które są wspólne dla wszystkich artykułów.

2. Stan po zrobieniu zadania:
W celu rozwiązania problemu , zmodyfikowałem schemat bazy w następujący sposób:



Z punktu widzenia relacyjnego problem rozwiązuje tabelka łącznikowa url_sentences pomiędzy ProcessUrl a Sentence.
ProcessedUrl dalej agreguje instancje typu Sentence, ale dodatkowo instancje Sentence mogą przynależeć do różnych (więcej niż jednego) ProcessedUrl.

3. Usunięcie problemu reundancji:
Dodatkową funkcjonalnością jaką dodałem było usunięcie wspomnianej nadmiarowości przy wstawianiu informacji do bazy. Początkowo posiłkowałem się w tym celu strukturami danych typu HashMap (dodanie pary(tekst,obiekt) i sprawdzenie, czy klucz nie pojawił się już wcześniej). Miało to jednak oczywiste ograniczenie: o ile w przypadku wstawiania wszystkich stron do bazy tylko przy jednym uruchomieniu aplikacji wszystko działało poprawnie, o tyle operacje dodawania w n_tym odpaleniu HMTLIndexera nie korzystają z wcześniej utworzonych struktur, tworząc nowe. Istnieje więc ryzyko, że dwie instancje słowa pojawią się w naszej bazie.
By tego uniknąć, dodałem nieco wolniejszą, ale bezpieczniejszą, operacjęwstawiania do bazy: przy każdej próbie persystowania obiektu wykonywane jest zapytanie sql-owe czy instancja nie istnieje już w bazie. Dopiero przy odpowiedzi przeczącej tworzony jest nowy obiekt, następnie persystowany i zwracany do aplikacji. Jest to ukryte za "interfejsem" get*() w głównej klasie.
To rozwiązanie realizuje w pewnym stopniu wzorzec Factory. Wpływa też niestety na szybkość, gdyż przy każdym wywołaniu get*() następuje przeszukiwanie bazy, co w porównianiu do sprawdzania klucza w strukturze typu HashMap nie posiada czasu jednostkowego.