Wprowadzenie do baz danych

DBM_1_01_Przygotowanie środowisk

Witaj w pierwszym module szkolenia DB Master. Ten pierwszy moduł poświęcony będzie w zasadzie w całości **zagadnieniom optymalnego modelowania bazy danych**. Jak się wkrótce przekonasz, to jest ta część, gdzie zaczynamy już myśleć o wydajności. Tak więc zachęcam Cię do spędzenia następnych czterech lekcji, w których opowiem ci, jak należy odpowiednio podejść do tego zagadnienia.

Przygotowanie środowisk

Pierwsza lekcja będzie jednak poświęcona przygotowaniu środowisk. Jak wiesz z modułu wstępnego, będziemy pracowali z czterema głównymi silnikami baz danych. Jeśli spojrzysz na portal **db-engines.com**, to faktycznie **Oracle, MySQL, Microsoft SQL Server oraz PostgreSQL są wiodącymi silnikami relacyjnymi na świecie**. Będziemy wobec tego pracować z następującymi wersjami tych silników: SQL Server 2014 lub nowszy, Oracle 12c, MySQL w wersji 8 oraz PostgreSQL w wersji 10 lub nowszej. Dla każdego z tych silników przygotowałem dla Ciebie bazę danych szkoleniową, na której będziesz pracował cały czas w trakcie trwania tego programu. Linki do skryptów tworzących bazę danych znajdziesz jako dodatek do materiałów z tej lekcji.

Jesteś pewnie ciekawy, jakich narzędzi będziemy używali. Dla SQL Servera wybrałem darmowe narzędzie **SQL Server Management Studio**. Ja mam wersję 18.4, ale równie dobrze możesz użyć starszej. Dla Oracle wybrałem darmowe narzędzie **SQL Developer** w wersji 19.2. Dla MySQL to oczywiście **MySQL Workbench** w wersji 8.0. Dla Postgre – **pgAdmin** w wersji 4.16, to jest, jak dobrze wiesz, narzędzie webowe, w więc będziemy dokonywali wszystkich operacji w przeglądarce. Oczywiście pamiętaj – użyj takie narzędzia, w których czujesz się dobrze. Ja wybrałem takie, ponieważ w mojej perspektywie one są bardzo proste w obsłudze i nie wymagają dodatkowej konfiguracji.

Teraz ważna uwaga – pamiętaj, że w trakcie szkolenia w zasadzie nie będziemy poruszali zagadnień administracyjnych. Jedynie takie, które mają bezpośredni wpływ na wydajność zapytań i wydajność bazy danych w ogólności. W module dziesiątym będę opowiadał o optymalnej konfiguracji serwerów bazodanowych dla tych czterech, którymi będziemy się zajmowali. Oczywiście, możesz powiedzieć, że ta konfiguracja ma

bezpośredni wpływ na wydajność, ale pamiętaj, że administracja serwerem to nie tylko jego konfiguracja i administratorzy patrzą troszeczkę szerzej. Ich będą interesowały również zagadnienia o wysokiej dostępności, co mają zrobić w przypadku, kiedy nastąpiła awaria, w jaki sposób odzyskiwać dane oraz oczywiście – czemu poświęcę cały moduł numer 9 – bezpieczeństwo danych.

Schemat bazy danych

Póki co, chciałem ci teraz opowiedzieć, jak wygląda schemat szkoleniowej bazy danych. Będziemy wykonywali zadania na dość prostych strukturach, najczęściej spotykanych strukturach bazodanowych, a więc jeśli popatrzysz na schemat bazy danych, to spójrz, że mamy tam dziesięć tabel. **Tabela Country** będzie zawierała informacje o siedmiu krajach, takie najbardziej podstawowe, czyli CountryId, CountryName oraz kod takiego kraju. Tabela pracowników, czyli **Employee**, będzie miała dziewięciu pracowników, ale struktura tej tabeli jest hierarchiczna, to znaczy, że znajdziesz tam również informacje o przełożonym każdego z tych pracowników. **Tabela Customer** będzie miała czterech klientów. Z kolei **tabela Model** będzie miała model dla pięciu produktów.

I teraz przechodzimy do części tej bazy danych poświęconej produktom. W **tabeli ProductCategory** znajdziesz cztery kategorie produktów. W tabeli **ProductSubcategory** będzie z kolei dziewięć podkategorii. A **tabela Product** będzie miała osiemnaście produktów. Jeśli pomyślisz, w jaki sposób moglibyśmy uzyskać informację o przynależności danego produktu do konkretnej kategorii czy podkategorii, to oczywiście będziemy odpowiednio modelowali pewne połączenia między tymi tabelami i tutaj również będziesz mógł utworzyć sobie pewną hierarchię produktu względem kategorii, podkategorii, ale w zupełnie inny sposób niż ma to miejsce dla pracowników.

Teraz mamy dwie najważniejsze tabele operacyjne, to znaczy **tabelę SalesOrderHeader**, gdzie wygenerowałem 7359 zamówień z różnymi statusami; brakuje nam jeszcze w tym modelu **tabeli Status**, która zawiera cztery statusy. Ostatnia tabela, w naszym przypadku największa, będzie posiadała 22000 wierszy i tu oczywiście mowa o **tabeli SalesOrderDetail**. Moim zdaniem te 22000 wierszy wystarczy nam już do tego, żebyśmy mogli zaobserwować pewne nieoptymalne zachowania zapytań.

Jeśli przyjrzysz się dokładnie schematowi tabel, to stwierdzisz, że brakuje całego mnóstwa elementów, to znaczy między innymi kluczy, relacji między tabelami i być może jakichś wartości domyślnych oraz tak zwanych *check constraints*. Ale nie tylko.

Oczywiście część z nich już zaimplementowałem i jestem ciekaw, czy patrząc na strukturę tabeli Country, potrafisz odnaleźć jeden taki ważny element. Spójrz zatem do skryptu i zastanów się przez chwilkę.

Jeśli spojrzałeś do skryptu, to być może odgadłeś, że chodziło mi o **nullowalność** – tak jak widzisz, zarówno kolumna CountryName oraz CountryCode nie mogą przyjmować wartości pustej, czyli NULL. Myślę, że możesz się w tym momencie zacząć domyślać, jakie będzie jedno z Twoich pierwszych zadań domowych do samodzielnego rozwiązania, o którym opowiem Ci na zakończenie tego modułu. Teraz przejdziemy do kolejnej lekcji, gdzie chciałbym Ci przedstawić, jaka jest różnica w modelowaniu logicznym aplikacyjnym, a fizycznym bazodanowym. Zapraszam!

DBM_1_02_Modelowanie aplikacyjne – opis problemu

Witaj w lekcji drugiej pierwszego modułu, którego głównym tematem jest **modelowanie bazy danych**. Zastanawia Cię na pewno, dlaczego poruszam taki temat. I od razu chciałbym Ci odpowiedzieć na to pytanie. To jest **fundament dobrej wydajności**, warto więc poświęcić więcej niż kilka chwil na ten temat i zachęcam Cię od tej pory do głębszych przemyśleń za każdym razem, kiedy będziesz tworzył nowe tabele w bazie danych. Pamiętaj, że zmiana modelu bazy danych może być bardzo trudna, a czasami bywa na tyle skomplikowana, że taką bazę danych po prostu będziesz wolał zbudować sobie zupełnie od nowa.

Zacznijmy zatem od czegoś prostego. Spróbujmy razem spojrzeć na modelowanie encji w aplikacji, wiedząc, że taka encja będzie musiała później w jakiś sposób zostać odzwierciedlona w bazie danych. Przykład, który za moment zobaczysz, wyda ci się na pewno mocno przesadzony i być może w pewnym sensie nawet skrajny. Pomyślisz, że nikt tak przecież nie robi, ale właśnie na tym skrajnym przykładzie najlepiej będę mógł Ci pokazać, **jak model aplikacyjny, czyli logiczny różni się od modelu fizycznego, czyli bazodanowego**. I tutaj od razu drobna uwaga, bo w bazie danych również są oczywiście obiekty logiczne, takie jak widoki albo, inaczej mówiąc, perspektywy. Używając terminu model fizyczny, ja zawsze odnoszę się do tabel, czyli tych obiektów, które przechowują dane. Zastanówmy się teraz na przykładzie prostej – ale to niekoniecznie musi być prawda – takiej prostej aplikacji, której zadaniem będzie wysyłanie maili. Czyli będziemy chcieli utworzyć sobie formularz, gdzie przy dość ograniczonej ilości informacji, ze względu na to, że upraszczam ten problem maksymalnie, chciałbym w jakiś sposób mieć możliwość wysyłania maili. Jakie pola mogłyby nas w takim razie interesować?

W tej chwili chciałbym **zaimplementować sobie aplikacyjnie encję, którą nazwę EMAIL**. Pamiętaj, że będą mnie interesowały te najbardziej istotne informacje, a więc na przykład osoba, która ten e-mail wysyła, czyli nasz sender. Od razu staram się napisać, jaka jest liczność takiego obiektu, który wchodzi w skład encji e-mail. Sender, czyli nadawca musi być tylko jeden, odbiorców może być kilku, musi być co najmniej jeden, maksymalnie będzie ich k. Następnie mam osobę, która może być w kopii, i wiemy, że taka osoba nie jest obowiązkowa. Wobec tego może ich być od 0 do jakiejś litery l. Pomijamy w naszym modelu *blind copy*, nie chcemy wysyłać takiej informacji, nazwijmy to niejawnej. Co jeszcze mógłbym posiadać w takim mailu? Na przykład mógłbym posiadać pole, które nazywa się **topic**, czyli temat, no i jak wiesz dobrze, temat może być tylko jeden. I teraz miejsce, w którym samo modelowanie zarówno aplikacyjne, a zwłaszcza bazodanowe, mogłoby nam się mocno skomplikować. A mianowicie chodzi mi o pole, w którym faktycznie napiszemy sobie tego maila, czyli pole **body**. Dlaczego myślę o komplikacji? Tak jest przecież w normalnych klientach poczty, których używamy, że w takim polu body nie tylko będzie przechowywany tekst, możesz również wklejać obrazki. Wobec tego przechowywanie obrazków jako format tekstowy nie wchodzi w grę. Ja upraszczam i mówię, że będziemy przechowywali tutaj tylko pole tekstowe. Dopiszmy do końca model – skoro mamy temat, mamy miejsce, gdzie wpisujemy maila, niech będzie jeszcze stopka, która może mieć podobny stopień skomplikowania, jak body, dlatego że wiesz, że tam również może znajdować się grafika, na przykład informacja o osobie, która wysyła, jakieś logo firmy, i tak dalej. Żeby zakończyć temat maila, dorzucę jeszcze załączniki, to znaczy w takim przypadku – również uproszczonym – będziemy je rozważali, że może ich być od 0 do n.

Teraz, kiedy wiemy, jakie obiekty czy jakie pola nam są potrzebne od strony aplikacyjnej, spróbujmy poczynić jeszcze pewne założenie. Gdyby spojrzeć na te pierwsze trzy obiekty czyli **nadawca, odbiorca oraz osoba w kopii**, w zasadzie mógłbym powiedzieć, że to są osoby, to są pewnego rodzaju obiekty. Wobec tego mógłbym założyć sobie inną encję czy inną klasę, która nazywać się będzie **Person**. W tej innej klasie, powiedziałbym, interesują mnie takie właściwości osoby jak imię, nazwisko i oczywiście e-mail. Jeśli chodzi o odbiorców, to w zasadzie będzie to taka lista osób, zapisana w taki sposób [slajd 5:50]. Podobnie będzie oczywiście z listą osób, które znajdują się w kopii tego maila. Jeśli chodzi o **typy danych**, to nie będzie tutaj żadnej niespodzianki – zarówno imię, nazwisko, jak i e-mail będą typu znakowego, czyli będą to stringi. W przypadku **pola topic** w zasadzie nie mam nic do modelowania, bo będzie to temat, czyli po prostu łańcuch znaków. Podobnie przy uproszczeniu, które przyjąłem ze względu na pole, które będzie nazywało się **body** – również będzie to string. Oczywiście **stopka** również będzie stringiem. W przypadku załącznika sytuacja ponownie może nam się mocno skomplikować, bo gdybyśmy wspólnie zaczęli

rozważać, w jaki sposób i co z tego załącznika tak naprawdę mnie interesuje, to ten model załącznika byłby na pewno niełatwy. Ja postanowiłem zrobić sobie taką **listę obiektów typu attachment**, gdzie pojedynczy obiekt mógłby być opisany w następujący sposób – miałbym **FileName**, **Extension**, czyli pole, które przechowa mi rozszerzenie pliku, tak żeby po kliknięciu w taki załącznik jakiś program automatycznie pozwolił mi go otworzyć, i byłby jeszcze **FilePath** jako ścieżka na dysk czy to na serwerze miejsce, gdzie ten załącznik jest przechowywany. Oczywiście brakuje mi możliwości przechowania tego załącznika w bazie danych. Tak można robić, ale niekoniecznie byłby to dobry wzorzec do takiego podejścia. To jest miejsce, w którym moglibyśmy powiedzieć, że to bardzo mocno zależy od rodzaju tych załączników, które będziemy mieli, ich charakterystyki. Pola takie jak FileName, Extension czy FilePath są typu znakowego.

Mogę więc powiedzieć, że taki model aplikacyjny – bardzo uproszczony, który na pewno dałoby się w jakiś sposób opracować lepiej, ale załóżmy, że mamy pierwszy stopień modelowania – taki model właśnie zrobiłem. Teraz chciałbym Ci pokazać, co się stanie i z jakimi problemami będziesz musiał się zmierzyć, jeśli wziąłbyś ten model i w całości przesunął go do bazy danych jeden do jednego, tak jak widzisz w tej chwili na ekranie [slajd 8:23]. Żebym mógł to zrobić, najłatwiej modeluje mi się takie rzeczy za pomocą arkusza kalkulacyjnego. Tak więc skopiuję sobie do arkusza kalkulacyjnego informacje, które mam, i na ich podstawie będę starał się zbudować taki najprostszy model, który utworzyliśmy aplikacyjnie. Będę chciał również Cię uczulić, że to nie jest dobry model – to jest ten moment, w którym powinieneś naprawdę zastanowić się nad tym, jak takie podejście modelujące strukturę bazy danych zrobić dobrze. Dokonuję pewnego porządkowania, żeby nasze notatki znajdowały się tutaj w odpowiednim formacie.

Przejdźmy zatem do rozwiązania, które potrzebujemy w tym momencie zaaplikować. Wobec tego ta pierwsza rzecz, która jest nam niezbędnie potrzebna, to **przeniesienie w stosunku jeden do jednego informacji do arkusza**. Biorę sobie zatem obiekt, który nazywa się Sender – składa się z imienia, nazwiska oraz maila. Spróbujmy więc zapisać imię, nazwisko oraz e-mail. To jest obiekt, który nazywać się będzie Sender. Skopiuję to troszeczkę niżej, w tym miejscu będzie mi łatwiej napisać, pokolorować te obiekty, bo jedną z najważniejszych rzeczy, którą robi się przy tego typu modelowaniu – przynajmniej ja szczerze do tego zachęcam – to jest odpowiednie pokolorowanie sobie pól, dlatego że łatwiej będzie widać, w jaki sposób zamieniamy jeden model w drugi. W trakcie tego modelowania, które skończy się w lekcji czwartej tego modułu, zobaczysz, jak po kolei przejdziemy z takiego modelu, który jest w tej chwili jeden do jednego logiczny aplikacyjny, aż do rozwiązania, które będziemy mogli spokojnie nazwać jako

dobre. Nie wiem, czy ono będzie najlepsze, ale ze względu na zapytania, które będziesz tworzył, ono będzie dobre.

Teraz musimy **przenieść strukturę, która nazywa się To**. Czyli musimy mieć odbiorców. I pojawia się pewien problem, bo ja nie wiem, jak wielu odbiorców tak naprawdę w tym silniku bazodanowym mógłbym zaimplementować. Co to znaczy liczba k? Czy powinienem umieć w tej tabeli powiedzieć, że zarezerwuję sobie pewną liczbę pól i na przykład maksymalnie 20 odbiorców. To jest zazwyczaj wystarczająca ilość osób, do których wysyłamy maile. Ja w naszym rozważaniu przyjmę, że k = 2. Podobne założenie przyjmę dla CC, to znaczy l = 2. Podobnie zrobię dla załączników, to znaczy ta lista załączników, która być może miała wartość m, również będzie równa 2. Wobec tego zwróć uwagę, że dla struktury, która nazywa się To, ja również zadeklaruję odpowiednią liczbę pól w schemacie tabeli. Tę tabelę nazwę **E-mailV1** i to będzie wersja pierwsza tego maila. Będziemy tworzyć wersję drugą i trzecią za moment. W podobny sposób zaimplementujmy rozwiązanie dla CC. Oczywiście widzisz, że takie rozwiązanie jest strasznie nieoptymalne. Kiedy od razu na początku w jakiś sposób musimy założyć, że model będzie w stanie przyjąć tylko określoną liczbę informacji, to nie jest dobry model. Nie powinniśmy tak postępować, aczkolwiek na rynku ciągle znajdziesz aplikacje uznanych firm, które z tego typu problemami muszą się borykać. Tak więc pod tym względem nie jest to przesadzone rozwiązanie. **Teraz zamodelujemy Topic, będzie również Body i Footer jako stopka**. Zostają mi dwa załączniki, czyli będzie tutaj Attachment i w tym Attachmencie mam FileName, Extension oraz FilePath. Skopiuję to raz jeszcze, tak żebyśmy mieli przeniesioną jeden do jednego strukturę logiczną aplikacyjną do bazy danych. Ja nawet sam ze sobą mam problem, żeby nazwać to modelem bazodanowym, dlatego że to nie mieści się w moich kanonach, które ja rozumiem jako modelowanie. Natomiast na ten użytek przyjmijmy, że mamy model tabeli EMAIL.

Coś, co musimy w tej chwili zrobić, to powiedzieć, **jak duże są pola tych łańcuchów znakowych**, być może nie wszędzie są tu łańcuchy znakowe, będą może również jakieś inne typy danych. A wszystko po to, żeby teoretycznie powiedzieć, jaka może być maksymalna wielkość takiego wiersza. To nam się przyda. Jak długie może być imię? Tutaj oczywiście można by się zastanawiać i powiedzieć 10, 20, 30 znaków, najpewniej należałoby to skonfrontować z dokumentacją projektową. Ja założyłbym, że 30 znaków. W przypadku nazwiska przyjmijmy, że 50, natomiast na e-mail maksymalnie 100 znaków, co może okazać się za mało – ale zostawmy taką wartość. Skopiuję w tej chwili te informacje, które już mam, do pozostałych komórek.

Teraz kolejne informacje – **jak duży może być temat?** Pytanie jest takie: ile będziemy pozwalali odbiorcom naszej aplikacji wpisywać znaków, na jak długie się zgadzamy. Ja

przyjąłbym, że 50 znaków, byłoby wystarczająco. Jeśli chodzi o pole body, to może 1000 znaków, natomiast stopkę postarajmy się ograniczyć również do 50 znaków. Nazwa pliku załącznika to 50 znaków. Extension załóżmy 5 – dlatego że trzyznakowe rozszerzenia już będą zbyt małe. Ścieżka na dysk niech będzie 255 znaków. Gdybym zsumował w tej chwili, to mój wiersz, teoretycznie gdyby był wypełniony zupełnie w całości, zająłby 2620 znaków. Czyli mógłbym w przybliżeniu powiedzieć, że to jest ta bajtowa wielkość tego rozwiązania, maksymalnie największa wartość wiersza, której mogę się spodziewać.

Żeby pokazać Ci, że **ten model jest zły w każdym możliwym podejściu**, to wstawię pewną przykładową ilość danych do tego modelu. Chciałbym, żebyś zobaczył za chwileczkę – być może w momencie, kiedy zacznę tę informację wprowadzać – to już zwrócisz uwagę, że chyba będziemy mieć za moment problem. Spróbuję napisać nadawców: chciałbym, żeby nadawcą był Piotr Kowalski, który ma jakiś e-mail pk w domenie wp.pl, i wstawię sobie trzy przykładowe wiersze dla Piotra Kowalskiego [slajd 16:16]. Wstawimy teraz kilku odbiorców, jednym z odbiorców będzie pani Janina Nowak, która będzie miała e-mail jn@wp.pl. Moglibyśmy wziąć również Grzegorza Nowaka, miałby e-mail gn@wp.pl, i Jerzy Mrozek jako trzeci odbiorca, e-mail jm@wp.pl. Oczywiście może się zdarzyć, że tych odbiorców jest więcej niż jeden. Wobec tego weźmy pana Jerzego, wstawmy go jako drugiego odbiorcę do drugiego maila. Teraz postanawiam skorzystać z imion czy osób, które już wprowadziłem, na przykład w to miejsce – odbiorcą będzie Grzegorz Nowak. Tu z kolei będzie Janina Nowak – pojawi się w tym miejscu i jeszcze w tym miejscu. Wrzućmy może pana Grzegorza jako CC do ostatniego maila.

Temat – załóżmy, że jest tutaj jakiś temat numer jeden, będzie również temat numer dwa i trzy. Jakieś informacje spróbujmy wprowadzić do samego maila, czyli tu będzie tekst. Uwaga! Z punktu widzenia modelowania, o czym się za chwileczkę przekonasz na koniec tego modułu – te trzy pola, które modelujemy w tej chwili, one nie będą miały możliwości optymalizacji. W zasadzie jedyną optymalizację, jaką będziesz mógł na nich przeprowadzić, to jest zastanowić się odnośnie tego, jaki one mają mieć rozmiar. I to jest wszystko. Dajmy jeszcze stopkę, która będzie się nazywała S1, pamiętajmy, że to jest stopka nadawcy. Przykładowy załącznik: w pierwszym mailu nie ma załącznika, w drugim zróbmy plik, który ma rozszerzenie .txt. I jakaś ścieżka na serwerze do tego pliku, niech będzie w taki sposób zrobiona, w pozostałych niekoniecznie musi się cokolwiek znajdować [slajd 18:40].

Jeśli spojrzeć na ten model i dane, które znajdują się w tej chwili w tabeli EMAIL, to powinna zastanowić Cię jedna przynajmniej rzecz – że ta tabela w sobie posiada informacje, które są związane bezpośrednio z mailem, takie jak topic, body i footer, oraz

część informacji, która z tym mailem, niekoniecznie z samym obiektem maila, musi być związana. I tym będziemy zajmowali się w kolejnej lekcji. Natomiast teraz chcę ci opowiedzieć o **aspekcie wydajnościowym tego nietrafionego rozwiązania**. Spójrz na to w ten sposób. Wyobraźmy sobie, że pani Janina Nowak zamierza zmienić stan cywilny, czyli wiemy, że od pewnego momentu ona będzie miała inne nazwisko, na przykład z nazwiska Nowak będzie chciała zamienić nazwisko na Kowalski – tak jak zwyczajowo ma to miejsce w przypadku zawarcia związku małżeńskiego. Jeśli chciałbyś poprawić dane pani Janiny Nowak, tak żeby odzwierciedlić tę zmianę, to zwróć uwagę, że będziesz musiał poprawić w zasadzie każdy wiersz, w którym występuje pani Nowak. Nie wiemy, czy pani Nowak jest nadawcą, odbiorcą czy znajduje się na liście CC i na dodatek na tych listach nie wiemy, czy jest na pierwszej, drugiej czy n-tej pozycji. W związku z tym będziesz musiał przeszukiwać tę tabelę każdy jeden wiersz. Jeśli zaczniesz przeszukiwać tę tabelę od początku do końca i dokonywać takich aktualizacji w wielu miejscach, to będziesz wprowadzał bardzo silne blokowanie. Czyli pojawi ci się nic innego jak tak zwane *locks*. Same ***Locks*, czyli blokady**, nie są niczym złym i to, że one są, raczej jest objawem czy manifestacją tego, że silnik bazodanowy stara się pilnować spójności danych. Po to są właśnie blokady. Natomiast pojawiają się również w momencie, kiedy dokonujemy jakiejś aktualizacji czy zmiany danych, po to właśnie, żeby mieć pewność, że aktualizacja nastąpi od początku do końca dobrze. Jeśli więc spojrzysz na ten model, to widzisz, że poprawianie danych osobowych będzie blokowało również dostęp do samych maili. W związku z tym pojawia się pytanie, czy to jest dobrze, i odpowiedź na to pytanie brzmi: absolutnie nie. W taki sposób nie możemy modelować, dlatego że każda operacja związana z wstawianiem danych, aktualizacją danych będzie powodowała jednak blokowanie. Przy wstawianiu danych nie ma żadnej prostej możliwości, żebyś upewnił się, że wstawiając nazwisko czy osobę, na przykład Grzegorz Nowak, czy taka osoba już w tej tabeli się znajduje, po to żeby sprawdzić, czy nie wykonujesz duplikatów danych. To jest jedna z rzeczy, która oczywiście w takiej tabeli będzie miała znaczenie.

Skoro już widzisz, że model bazodanowy, ten pierwszy, który zrobiliśmy, nie jest dobry, to zapraszam Cię na lekcję trzecią, w której opowiem Ci, jak bardzo diabeł tkwi w szczegółach. Zapraszam.

DBM_1_03_Modelowanie bazodanowe – diabeł tkwi w szczegółach

Zapraszam Cię na trzecią lekcję na temat dobrego modelowania struktur bazodanowych.

Zabrzmiało to bardzo poważnie i takie właśnie jest modelowanie. Dla przypomnienia – będą nas interesowały wyłącznie tabele, czyli takie obiekty bazodanowe, które faktycznie przechowują dane. Tutaj zrobię taką efektowną pauzę, ponieważ nie tylko w tabelach takie dane możesz trzymać. Jest inny rodzaj obiektów, na przykład widoki zmaterializowane, które nie są proste w tworzeniu i najprawdopodobniej z nimi nie będziesz spotykał się na co dzień. Tak więc nie będziemy o nich przynajmniej w tym module mówić.

Teraz chciałbym zadać ci proste pytanie: **ile masz tabel w bazie danych?** Czy jest ich kilka, a może jest ich kilkadziesiąt, czy może raczej myślisz, że twoja baza ma tysiące takich obiektów? W takim razie, które tabele są ważne z punktu widzenia odpowiedniego modelowania? Oczywiście pierwsza odpowiedź, która ciśnie się na usta, jest taka, że to wszystkie tabele, ale tak naprawdę ważne są duże tabele. Natomiast pytanie jest takie: jak należy to rozumieć? Otóż nie jest istotna tylko i wyłącznie ilość wierszy. Liczy się ilość megabajtów, które dana tabela zajmuje na dysku. Taka ilość megabajtów jest to funkcja, zarówno ilości wierszy, ale też – może to być nawet bardziej istotne – ilości kolumn.

Teraz taka krótka dygresja, którą chciałbym, żebyś ją zawsze miał na uwadze. Serwer bazodanowy składa się z trzech podstawowych elementów:

1. **Procesor (CPU)** – interesuje nas tutaj ilość rdzeni ze względu na kwestie licencyjne
2. **Pamięć (RAM)** – im więcej będziesz mógł przydzielić pamięci serwerowi bazodanowemu, tym oczywiście lepiej
3. **Podsystem dyskowy (I/O)**

Generalnie możemy przyjąć, że dyski są póki co wolniejsze od operacji wykonywanych bezpośrednio w pamięci, stąd też obserwujemy dążenie najważniejszych graczy bazodanowych do zbudowania dobrych rozwiązań *in memory*. To nie jest tylko i wyłącznie domena silników relacyjnych. Tak więc pamiętaj, że wydajność będziesz mógł opisać tymi trzema składowymi: CPU, Memory oraz I/O.

Na pierwszy rzut oka CPU nie jest tym elementem, który odpowiada za znaczącą ilość problemów wydajnościowych. Oczywiście to zależy od konkretnej aplikacji, ale to duża rzadkość, aby to procesor stanowił źródło problemów. Wynika z tego, że musimy skupić

się na pozostałych elementach, czyli na pamięci oraz podsystemach dyskowych, a generalnie będą nas tutaj interesowały ilości operacji oraz czas dostępu do dysku.

Teraz chciałbym pokazać Ci najważniejsze zdanie w ramach tej lekcji, które wypowiem. **Dla dużych tabel nawet 8 bajtów zaoszczędzonych w jednym wierszu robi kolosalną różnicę!** Pewnie nie będziesz chciał w to uwierzyć. Spójrz więc na tabelę, którą chcę Ci zaprezentować [slajd 3:37]. Ile jesteś w stanie zaoszczędzić przy tabelach, które mają już 10 albo więcej milionów wierszy? Jeśli oszczędzisz 8 bajtów na 10 milionach wierszy, to znaczy, że zaoszczędzisz aż 76 MB z poziomu całej tabeli. Jeśli masz tych wierszy miliard, to przy 8 bajtach na całym wierszu, zaoszczędzisz 7,6 GB. Natomiast jeśli jesteś w stanie zaoszczędzić 32 bajty, to dla tabeli o wielkości 10 milionów wierszy będziesz miał 305 MB różnicy. Oczywiście dla takiej większej tabeli z jednym miliardem, Twoja oszczędność wyniesie już 30 GB.

Możesz w sumie powiedzieć tak: „zaraz, zaraz, przecież dyski są tanie”, ale nie chodzi tylko o nie. Chodzi między innymi również o zużycie pamięci, dlatego że im więcej masz miejsca zajętego w tabeli, to tym więcej tych danych będziesz do pamięci ładował, a pamięci raczej mają wielkości mniejsze od iluś terabajtów.

Kłania się tutaj również administracja serwerem. Generalnie chodzi mi na przykład o wykonywanie kopii zapasowych, o budowanie odpowiedniej strategii wysokiej dostępności, czyli *high availability* (HA), budowanie strategii odzyskiwania danych w momencie wystąpienia awarii, czyli tak zwana *disaster recovery* (DR). Pozostaje również kwestia skalowalności Twojego modelu.

Wróćmy zatem do modelu bazodanowego, który zrobiłem do tej pory. W kolejnej lekcji pokażę Ci, jak go naprawić. Zrobię to dwustopniowo. Najpierw wykonam jeden model, który pokaże Ci ideę zamiany modelu logicznego w model bazodanowy oraz przy okazji pokażę Ci najlepsze praktyki z zakresu modelowania, a następnie wykonam już taki model optymalny, który będzie miał tę zaletę, że będzie się fantastycznie skalował. Zapraszam Cię na lekcję czwartą. Do zobaczenia!

DBM_1_04_Modelowanie bazodanowe – dobry model = klucz do dobrej wydajności

Witam Cię w lekcji czwartej w module poświęconym modelowaniu baz danych. Wreszcie nadszedł czas, aby pokazać ci dobre praktyki związane z modelowaniem tabel w bazie danych.

Do dobrych praktyk będziemy zaliczali między innymi wybór odpowiednich typów danych, normalizację tabel, zwłaszcza tych dużych, wybór odpowiedniej kolumny jako kandydata na klucz główny, używanie kluczy obcych, tak zwane *referential integrity* oraz nazewnictwo tabel czy kolumn. Oczywiście nie są to wszystkie dobre praktyki związane z ogólnie przyjętymi relacyjnymi bazami danych. Mógłbym wymienić na przykład kwestie dokumentacyjne, testowanie bazy danych oraz używanie widoków, funkcji czy procedur składowanych. Więcej na ten temat przekażę Ci w kolejnych modułach.

Niestety, bo muszę powiedzieć niestety, każdy z omawianych na tym szkoleniu silników bazodanowych jest świetny. Możesz więc zbudować w zasadzie dowolną bazę danych bez zwracania uwagi na szczegóły, o których będę Ci opowiadał, i przez pewien czas nie zauważysz żadnych problemów. Tak jak powiedziałem, niestety, a później w momencie, kiedy te problemy zostaną zauważone, będzie już za późno. Ale to pewnie wiesz, bo ja miałem okazję się o tym przekonać wielokrotnie. Teraz spróbuję Ci pokazać, jak powinienem naprawić to, co było popsute do tej pory. Zapraszam Cię zatem do wykonania dobrego modelu bazodanowego.

Kiedy dokończyliśmy model aplikacyjny i przełożyłem go w stosunku jeden do jednego do bazy danych, to zwróciliśmy uwagę, że pierwszy problem, który nam wystąpił, taki wydajnościowy, był związany z tym, że żeby dokonać aktualizacji danych osobowych jednej z osób, która może maila wysyłać, odbierać bądź być na liście CC, jest to związane z nadmiernym blokowaniem tabeli. Jak rozwiązać ten pierwszy problem? Tutaj z pomocą przychodzi nam **normalizacja tabeli**. Powinniśmy się zastanowić i powiedzieć tak: „chciałbym, żeby w tabeli EMAIL pozostały tylko i wyłącznie te rzeczy, które faktycznie z tym mailem są związane; każde inne obiekty powinny zostać przeniesione do innych tabel”. Spójrzmy na model, który mamy [slajd 2:49]. Gdybym posiłkować się tym, co przygotowaliśmy czy co otrzymaliśmy od strony aplikacji, to mógłbym powiedzieć, że to co jest związane z mailem, to będzie tylko temat, body oraz stopka. Pozostałe obiekty, takie jak nadawca, odbiorca czy lista osób będących w kopii, to są osoby, wobec tego mogę utworzyć osobną tabelę, która będzie zawierała wszystkie dane związane z osobami. Podobnie postąpię z załącznikiem. Wszystkie załączniki przeniosę do osobnej tabeli, tak więc w mailu zostawię tylko te trzy property: topic, body oraz footer, a do pozostałych obiektów przekażę wskaźniki, czyli tak zwane klucze.

Jak zrobić takie modelowanie? Załóżmy sobie tutaj tabelę, którą nazwę **E-mailV2**. Na marginesie – ta tabela czy to nazewnictwo na pewno nie jest zgodne z dobrymi praktykami czy ze standardami nazewnictwa, dlatego że nie powinniśmy w bazie danych trzymać tabel z konkretnymi wersjami. Utrzymanie tego typu obiektów, zarządzanie nimi z punktu widzenia logicznego będzie w pewnym momencie powodowało problemy. Zapiszę sobie, że chciałbym przechować właściwości takie jak topic, body oraz na zakończenie był to footer – i zostawmy na razie tabelę E-mail w taki sposób. Natomiast chciałbym w tej chwili przejść do tabeli Person, którą założę. Pierwszą kolumnę, którą zakładam w takiej tablice, to w tym momencie będzie kolumna, która będzie nazywała się PersonId. Moje pierwsze pytanie do Ciebie byłoby takie: jaki jest cel założenia tej kolumny? Oczywiście jeśli twoje myśli idą w kierunku takim, że jest to klucz podstawowy, klucz główny, tak zwany ***primary key*** – to tak, taki jest mój cel, każda tabela powinna taki klucz mieć zdefiniowany. Jest on nam potrzebny po to, żebym mógł zidentyfikować jednoznacznie każdy wiersz w tabeli.

Jakiego typu danych powinien być taki klucz główny? To zależy od tego, z jakim silnikiem danych pracujesz. Generalnie chodzi mi w tym momencie o jakiś typ numeryczny, taki który nie będzie miał części dziesiętnej. Chciałbym Ci pokazać w tej chwili taką ściągę, którą przygotowałem sobie, jeśli chodzi o typy danych, które są w różnych silnikach bazodanowych [slajd 5:41]. Gdybyśmy zaczęli od Oracle, to okaże się że w Oracle jest tylko jeden taki typ numeryczny, który będzie nazywał się NUMBER. **Typ numeryczny NUMBER** przyjmuje dwa parametry: parametr P jako *Precision* oraz S jako *Scale*. One oznaczają coś takiego: *Precision* mówi, jak wiele liczb chcesz przechować, a *Scale* mówi, ile z nich jest po przecinku. Uwaga, samego przecinka nie wliczamy do ilości liczb. Każdy z innych typów, na przykład typ **INTEGER**, **SMALL INTEGER**, **DECIMAL** jako alias na NUMBER, ale głównie INTEGER i SMALL INTEGER – one są tylko i wyłącznie aliasami na typ NUMBER. INTEGER i SMALL INTEGER to jest NUMBER od 38, czyli Precision jest równe 38, natomiast Scale równa się zero. Gdybyś chciał zrobić sobie typ SMALL INTEGER, który ma odpowiednik w SQL-u, gdzie może przechować liczby z zakresu od -32 000 do +32 000 bądź od 0 do 65 000, w tym momencie musiałbyś sobie w odpowiedni sposób SMALL INTEGERA zaprojektować w Oracle'u i powiedzieć, że byłby to NUMBER od 5. Oracle ma jeszcze jeden typ danych, który nazywa się **SIMPLE INTEGER**, który odpowiada takiemu integerowi, którego znasz z innych silników, takich jak SQL Server, MySQL czy Postgre, on ma ten sam zakres, ale posiada jedno ograniczenie, a mianowicie kolumna, która taki typ danych będzie używała, musi być zawsze oznaczona jako **NOT NULL**.

Popatrzmy na to, co oferuje nam **SQL Server**. Tutaj tych typów numerycznych jest zdefiniowanych troszkę więcej, poczynając od **TINY INTEGERA**, który zajmuje jeden

bajt, aż do **BIG INTEGERA**, który zajmuje tych bajtów 8 i może przechować wartość aż 2 do potęgi 63. Pytanie jest takie: gdybym pomyślał o PersonId, to który z tych typów TINY INTEGER, SMALL INTEGER, INTEGER bądź BIG INTEGER, powinienem wybrać? Tu musisz spojrzeć na temat z punktu widzenia biznesowego: ile masz osób, które są nadawcami bądź odbiorcami maili? Jeśli tych osób w Twojej firmie jest mniej niż 50 tysięcy, możesz wybrać sobie typ SMALL INTEGER. Natomiast jeśli masz jakieś rozwiązanie globalne i wiesz, że już w danej chwili Twoja korporacja, Twoja firma będzie wymagała, żeby w ramach projektu można było wysyłać maile do 100 czy 200 tysięcy osób, wtedy nie zastanawiaj się, wybierz typ INTEGER. Zawsze staraj się dopasowywać najmniejszy pasujący typ danych, bo to jest to miejsce, gdzie będziesz oszczędzał bajty, o których wspominałem Ci na poprzedniej lekcji. Z tego punktu widzenia, o którym w tej chwili mówimy dla tabeli PersonId, mógłbym przyjąć, że interesuje mnie typ SMALL INTEGER.

Popatrzmy na **MySQL-a**, tutaj również mamy TINY INTEGER, SMALL INTEGER i pojawił się także taki jeden taki typ danych, który nazywa się **MEDIUM INTEGER**, który jest pomiędzy SMALL oraz INTEGEREM i zajmuje odpowiednio 3 bajty. W **PostgreSQL** mamy podobną sytuację: SMALL INTEGER, INTEGER, BIG INTEGER.

Chciałbym zwrócić Twoją uwagę na to, że te typy danych są typami danych precyzyjnymi, to znaczy liczba jest przechowywana dokładnie w takim formacie, w jakim ją wpiszesz. Są również typy przybliżone, takie jak **FLOAT, REAL oraz DOUBLE**, które będą zachowywały się w taki sposób, że to jak dana liczba będzie przechowywana, będzie zależało o precyzji, którą dany typ będzie posiadał.

Powrócę w tej chwili do modelu i zapiszę, że taki PersonId powinien być typu SMALL INTEGER. Wobec tego zajmie dwa bajty. Pozostałe trzy kolumny, takie jak imię, nazwisko oraz oczywiście e-mail, one nie ulegają zmianie. Ich wielkości są takie jak zaproponowaliśmy wcześniej, czyli dla imienia będzie to 30 znaków, dla nazwiska 50, a dla e-maila 100. W podobny sposób postąpimy dla attachementu. Ja sobie zaznaczę tutaj tabele, odpowiednio pogrubiając ich nazwy [slajd 10:53]. Będzie mi za moment łatwiej zauważyć pewne rzeczy. Chciałbym teraz wygenerować sobie nową tabelę Attachement, która będzie miała kolumnę AttachementId i zwróć uwagę, że to akurat jest zgodne z takimi dobrymi praktykami odnośnie nazewnictwa, że klucz główny powinien składać się z nazwy tabeli i dodajemy mu końcówkę ID, z tym że to oczywiście jest kwestia umowy projektowej. Jeśli w twoim projekcie jest inaczej, to trudno. Generalnie chodzi o to, że jeśli w ramach projektu ustaliłeś czy ustaliliście w swojej grupie pewne reguły, to musicie się trzymać.

Tabela Attachement z kolumną AttachementId – tutaj typem danych, który bym zaproponował, byłoby INTEGER na czterech bajtach, dlatego że wydaje mi się, że miałbym więcej załączników docelowo niż sześćdziesiąt kilka tysięcy, natomiast zdecydowanie mniej niż dwa miliardy. Mieliśmy tutaj kolumnę, która nazywała się FileName, kolumnę Extension, no i ostatnia kolumna, która nazywa się FilePath. One mają długość tak jak poprzednio, czyli FileName oznaczyliśmy, że powinien mieć, o ile dobrze pamiętam, 50 znaków; dla Extension było to 5, a dla FilePath, czyli ścieżki na dysk 255. No i teraz pytanie, co dalej powinniśmy zrobić? W jaki sposób tabela E-mail będzie miała połączenia do tabeli Person oraz do tabeli Attachement.

Można to zrobić w bardzo prosty sposób za pomocą relacji. Spróbujmy zatem to zrobić. Ja przeniosę te trzy kolumny, które miałem do tej pory przygotowane, troszkę niżej, dlatego że chciałbym Ci pokazać dobry sposób utworzenia definicji tabeli, tak żeby ona była bardziej czytelna. Wobec tego, idąc za definicją i kierując się regułą, którą omówiłem już do tej pory, chciałbym założyć również tutaj kolumnę, która będzie nazywała się EmailId i przyjąłbym, że ta kolumna jest typu BIG INTEGER, więc ona będzie przechowywała 8 bajtów. Następnie z modelu bazodanowego wiesz, że mamy jednego odbiorcę, wobec tego byłby to SenderId i ten SenderId to byłaby kolumna PersonId z tabeli Person. Następnie mamy dwóch odbiorców, czyli mamy ToId1 i to również będzie kolumna Person. Będziemy mieć jeszcze dwie osoby na liście CC. Wobec tego mogę napisać CCId1 oraz CCId2. I zostały koniec końców dwa Attachementy, czyli AttachementId1 oraz AttachementId2 i to oczywiście są odpowiadające kolumny AttachementId z tabeli Attachement. Jestem w stanie to oczywiście skopiować, żeby oznaczyć, skąd dana kolumna się bierze.

Gdybym teraz popatrzył na to, ile każda z tych kolumn zajmuje bajtów, to mógłbym powiedzieć tak: ta pierwsza zajmuje 8, druga zajmuje 2, tu 2 i 2 oraz 2, 4, 4, te pozostałe się nie zmieniają, na temat zapisaliśmy 50 znaków, na body 1000, na stopkę również, o ile pamiętam, 50-znakową. Nie chcieliśmy, aby w ramach tej stopki zbyt wiele informacji było można przekazać. Oznaczam EmailId jako tak zwane *primary key*, a pozostałe klucze oznaczane jako tak zwane klucze obce, co powodują utworzenie relacji. Oznaczę teraz kolorem kolumny [slajd 15:20], które są kolumnami klucza, które będą odnosiły się do tabeli Person. Wobec tego oznaczę je sobie kolorem żółtym, tabele Person również oznaczę kolorem żółtym, będzie nam się łatwiej za moment rozeznać, w jaki sposób wykonane są połączenia. Jeśli chodzi o Attachement, możemy wybrać inny kolor niebieski i również tutaj oznaczyć Attachement na kolor niebieski. Zwróć uwagę, że teoretyczna wielkość wiersza, który w tej chwili mamy, na pewno będzie znacznie mniejsza od tego, co było wcześniej, dlatego że ten wiersz w tej chwili zajmuje 1126 bajtów. Oczywiście muszę policzyć analogicznie wiersze dla osoby

Person oraz odpowiednio policzyć również dla załącznika. I okazuje się, że w tej chwili zamiast jednej tabeli będę tych tabel miał trzy. Natomiast najbardziej interesowało mnie, żebym był w stanie zmniejszyć tabelę, która do tej pory była duża, miała dużą ilość kolumn. Wobec tego dzięki odpowiedniemu modelowaniu ta liczba kolumn się zmniejszyła i dramatycznie zmniejszył się również rozmiar tej tabelki. Nie wiem, czy pamiętasz, jak mówiłem, że kolumny, topic, body oraz footer to będą te jedyne, z którymi ja nie będę w stanie nic zrobić. Gdybyś chciał pomyśleć, co tutaj się da zrobić, to należałoby się zastanowić, czy rozmiar danych jest odpowiedni: 50, 1000 oraz 50 znaków.

Dlaczego ten model jest lepszy? Zwróć uwagę, że w poprzednim przypadku, w momencie kiedy chcieliśmy zmienić nazwisko osoby, czyli zmienić nazwisko w tym wypadku pani Janiny Nowak na inne, to dokonywaliśmy blokowania znacznej ilości, jeśli nawet nie całej tabeli. W podejściu tym, które teraz mamy, jeśli zmienię imię bądź nazwisko osoby, to zwróć uwagę, że nie ma to żadnego wpływu na tabelę z mailami, dlatego że tabela E-mail posiada jedynie wskaźnik do osoby. Dopóki nie zmienisz tego wskaźnika, czyli nie wygenerujesz nowego klucza, to można powiedzieć, że taka operacja jest bezpieczna i na pewno nie będzie blokowała tabeli E-mail. Tak więc odpowiedni model i zastosowanie kluczy obcych pomogło Ci zredukować blokowanie.

Te klucze obce pozwolą ci również zrobić inną ważną rzecz, a mianowicie zapewnią Cię, że twoje dane będą spójne. Pytanie, które mógłbym sobie w tej chwili zadać, jest takie: czy mógłbym użyć na przykład jako nadawcy, identyfikatora nadawcy, taki numer klucza, który nie istnieje w tej tabeli. Na przykład moja tabela ma w tej chwili 10 osób z kluczami w kolumnie PersonId od 1 do 10, i pytanie jest takie, czy Sender w tabeli E-mail mógłby mieć numer 11. Oczywiście, odpowiedź brzmi, że nie. W momencie kiedy założysz taką relację *primary key – foreign key*, ta relacja będzie pilnowała spójności danych, a więc będziesz najpierw musiał zarejestrować osobę w tabeli Person i dopiero wtedy będziesz mógł jej użyć na jednym z tych pięciu miejsc w tabeli E-mail. To jest zadanie kluczy obcych.

Chciałbym powrócić jeszcze na chwilę do typów danych, a mianowicie opowiedzieć ci o typach, które związane są z długością znaków, czyli z **typami znakowymi**. Generalnie mamy dwa podejścia w tych silnikach. W każdym z nich jest to zrobione identyczne. Po pierwsze mamy typy stało znakowe, czyli takie typy jak **CHAR** oraz **NCHAR** w przypadku SQL-a czy Oracle'a, oraz CHAR w przypadku MySQL i PostgreSQL. Co to oznacza?

Jeśli zadeklarujesz sobie CHAR(20), taki typ danych i będziesz chciał przechować tam taki string jak DBMASTER, który ma znaków 8, a zadeklarowałeś zmienną o długości

20 znaków, to pozostałe 12 znaków będziesz miał wypełnione spacjami. Wobec tego już powinieneś umieć ocenić, że ten typ danych będzie idealnie sprawdzał się wtedy, kiedy masz pewność, że zawsze albo prawie zawsze zajmiesz całą jego pojemność.

One nadają się na przykład do kolumn typu kod waluty, na przykład kod waluty jest 3-znakowy: PLN, EUR i USD. Wobec tego taki typ danych CHAR(3) fantastycznie spełni swoją funkcję dla takiej kolumny. Dla każdej innej kolumny, która będzie zawierała na przykład imię, nazwisko, nazwę kraju, gdzie wielkość czy ilość znaków jest różna, powinieneś raczej skoncentrować się na typie, który będzie nazywał się **VARCHAR** albo **NVARCHAR**. W przypadku VARCHAR-a czy NVARCHAR-a w momencie kiedy zadeklarujesz sobie na przykład kolumnę o długości 20 znaków i będziesz chciał wstawić napis DBMASTER, który zajmuje 8 znaków, pozostałe 12 znaków zostanie odzyskanych, czyli one nie będą już wypełnione spacjami, ale kolumna dopasuje się do tego rozmiaru. Oczywiście nic za darmo. Pamiętaj, że w jednym wierszu w tej kolumnie znaków możesz mieć 8, w drugim wierszu 19, więc ta kolumna rzeczywiście ma zmienną długość. Sposób, w jaki każdy z silników sobie z tym radzi, jest taki, że dokłada jeszcze pewien 4-bajtowy wskaźnik, który mówi, gdzie każda z takich kolumn się zaczyna. Wobec tego, jeśli mówimy VARCHAR(20), to pamiętaj, że gdzieś tam z tyłu są jeszcze 4 bity użyte na operacje wewnętrzne, które są potrzebne do identyfikacji miejsca na dysku, gdzie dana kolumna w danym wierszu się zaczyna.

Te typy VARCHAR mają różną pojemność w zależności od silnika bazodanowego. Typowo w Oracle'u możesz mieć tam do 32 KB, w SQL-u jest to rozmiar 8 KB, z kolei w MySQL-u możesz mieć 64 KB, ale te 64 KB jest współdzielone z innymi kolumnami tego typu w danej tabeli. W przypadku PostgreSQL jest taka liczba, która jest wielkością 10 MB w ramach jednej kolumny. Oczywiście na przykład w SQL-u jest jeszcze typ **MAX**, który mówi, że pojedynczy wiersz może zajmować do 2 GB. Analogiczne rozwiązania znajdują się w innych silnikach bazodanowych.

Przy okazji wspomnę jeszcze o typach, które związane są z datą i czasem, dlatego że one bardzo często w wielu tabelach występują. Jest to również bardzo ważne, żebyś wybrał odpowiedni typ danych do tego, co dana kolumna powinna robić. Jeśli więc masz na przykład zapisać datę urodzenia pracownika, powinieneś wybrać taki typ, który oferuje Ci przechowanie tylko i wyłącznie daty. Jeśli natomiast masz zapisać z bardzo dużą dokładnością jakieś zdarzenie, no to każdy z silników oferuje taką możliwość. Na przykład w Oracle'u jest typ **DATE**, który na 7 bajtach zapisuje datę i czas z rozdzielczością do sekundy. Jeśli potrzebujesz większą rozdzielczość, masz typ **TIMESTAMP**. Jest również możliwość posiadania takiego typu ze strefą czasową, to wtedy on się nazywa **TIMESTAMP WITH TIME ZONE** albo **TIMESTAMP WITH LOCAL TIME ZONE**. Po stronie SQL Servera masz typ, który nazywa się DATE, który na 3

bajtach przechowuje datę. Jeśli potrzebujesz datę i czas z taką rozdzielczością do trzech i jednej trzeciej milisekundy, powinieneś wybrać typ **DATETIME**. Potem masz typ danych **SMALLDATETIME**, który ma rozdzielczość jednej minuty. No i dużo bardziej precyzyjne typy, takie jak DATETIME2, który ma rozdzielczość do stu nanosekund oraz **DATETIMEOFFSET**, który jest analogiem tego, co ma Oracle w typie TIMESTAMP WITH TIME ZONE, czyli masz możliwość określenia również strefy czasowej. Osobny typ **TIME**, który przechowuje na 5 bajtach wartość czasu. W MySQL-u będą to typy DATE, DATETIME, TIMESTAMP, TIME oraz YEAR. Po stronie Postgre bardzo podobne: DATE 4-bajtowy, TIME 8-bajtowy, TIME ze strefą czasową 12 bajtów, TIMESTAMP ze strefą czasową 8 bajtów.

Nie jest ważne, żebyś te wszystkie typy pamiętał. Znajdziesz sobie opis do nich w dokumentacji każdego z tych produktów z każdego z tych silników. Ważne jest, żebyś miał na uwadze, że jeśli będziesz modelował każdą z tabel, to wybieraj odpowiednie typy danych do każdej kolumny, żebyś starał się nie napisać tabeli, która ma dwadzieścia kolumn znakowych i wszędzie wpiszesz VARCHAR albo NVARCHAR(255), kiedy wiesz, że w jednej z tych kolumn będzie to tylko 5 znaków, a w drugiej 7. Masz kolumnę, która przechowuje DATE, czyli datę, na wszelki wypadek wpisujesz DATETIMEOFFSET albo TIMESTAMP, które są znacznie większe. To jest tylko i wyłącznie w kwestii przypomnienia, że 8 bajtów naprawdę ma znaczenie. Wobec tego powinieneś szukać możliwości rozwiązania, takiego modelowania przy wykorzystaniu optymalnych typów danych.

Powróćmy do modelu, dlatego że na poprzedniej lekcji wspomniałem Ci, że będą dwa modele. Ten model pierwszy, który w tej chwili wykonaliśmy, on się jeszcze nie skaluje. Ciągle masz ograniczenie, że masz jednego nadawcę – co akurat jest dobre – ale odbiorców możesz mieć co najwyżej dwóch i dwie osoby z listy CC. Prawdziwe profesjonalne rozwiązanie, które chciałbyś wytworzyć, nie powinno mieć tego ograniczenia. Wobec tego jak to zrobić? Jak powinniśmy podejść do tej sytuacji? Wynika z tego, że gdybyśmy tylko popatrzyli na tabelę E-mail, wystarczyłoby pozbyć się tych elementów, które są w tej chwili wskaźnikami. Czyli gdyby tabela E-mail posiadała tylko i wyłącznie klucz główny EmailId plus swoje kolumny biznesowe, natomiast wszystkie inne klucze zostały przeniesione do jeszcze jednej tabeli, taką tabelę będziemy nazywali *bridge table* albo *link table*. Jeśli zaprojektujemy taką strukturę, to dojdziemy do sytuacji, w której nasz model będzie się skalował bez żadnych ograniczeń. Spróbujmy zatem to zrobić.

Utworzę sobie tabelę **E-mailV3**. Pozostawiam w niej kolumnę Email na 8 bajtach oraz te kolumny, które miałem do tej pory, czyli topic, body oraz footer, czyli stopka. Obok pozostawiam te dwie tabele, które miałem do tej pory, i z nimi nic nie będę robił. W

sumie jestem zadowolony z tego, w jaki sposób udało nam się je przygotować. Zrobię odpowiedni kolor, w tej chwili będzie to już ważne, żeby kolor tabeli E-mail był również jakiś – pozwoli nam to za moment zauważyć, w jaki sposób zamodelujemy tę tabelę, którą nazwałem *bridge table*. Ona również się nazywa *link table*, tabela linkująca, tego typu stwierdzenia znajdziesz w literaturze fachowej.

Jak ją zrobić? Mogłaby to być tabela o takiej postaci, która nazywa się EmailPerson. Wobec tego miałaby kolumny EmailId oraz PersonId. W takim wypadku zwróć uwagę, że masz EmailId z tabeli Email, PersonId z tabeli Person, ale zgubiliśmy w tej chwili informacje na temat, jaką funkcję ta osoba pełni. Person – nie wiemy, czy jest nadawcą, odbiorcą czy znajduje się na liście CC. Wobec tego będzie potrzebna jakaś dodatkowa informacja, mogłaby to być jednoznakowa kolumna, która przechowywałaby na przykład jedną z takich wartości, jak S czyli Sender, T jako To oraz C jako CC. Albo być może inne rozwiązanie?

I to jest zadanie, nad którym chciałbym, żebyś pomyślał w ramach zadania domowego. Czy mogę zostawić to w taki sposób jak jest? A może mogę zrobić to inaczej? Może osobna tabela? Może jednak gdzieś indziej tę kolumnę PersonId powinienem przenieść jeszcze? Tych rozwiązań ja widzę co najmniej dwa i będę chciał o nich z Tobą podyskutować na naszej stronie na Facebooku.

Natomiast mam jeszcze inne pytanie odnośnie tabeli EmailPerson. Czy ona już rzeczywiście rozwiązuje wszystkie problemy, które mamy? Spróbujmy ją najpierw zmierzyć, to znaczy podajemy 8 bajtów +2, tutaj załóżmy, że jest jeden bajt, czyli bajtów mamy w tym momencie 11. Zmierzę jeszcze tę pierwszą tabelę – 1108 bajtów na jeden wpis. Ten problem, o którym wspomniałem, w momencie kiedy opisywałem tabele EmailPerson, jest taki: w jaki sposób miałbym biznesowo upewnić się, że kolumna Person dla nadawcy będzie występowała tylko raz, tylko jedną mamy taką informację per e-mail; mamy co najmniej jednego odbiorcę i 0 bądź więcej osób na liście CC. Nie jesteś w stanie zapewnić za pomocą designu w łatwy sposób, wobec tego w tym momencie na scenę wchodzą tak zwane ***check constraints***, które pozwolą Ci zrealizować to zadanie.

To już wszystko w lekcji czwartej. Zapraszam Cię teraz na zadanie domowe, ale chciałbym Cię upewnić, że temat modelowania nie zostanie zapomniany w module pierwszym i to nie znaczy, że nie spotkasz się już z nim w kolejnych modułach. Chciałbym Ci powiedzieć, że w jednej lekcji w miejscu, w którym się tego zupełnie nie spodziewasz, temat modelowania wróci do nas jak bumerang. A tymczasem wysłuchaj co dla Ciebie przygotowałem w ramach zadania domowego. Zapraszam.

Zadanie

Nadszedł czas na zadanie domowe:

1. Przeanalizuj skrypt tworzący szkoleniową bazę danych.
2. Zaproponuj zmiany, które powinny zostać zrobione, aby ten model bazodanowy był poprawny. Zwróć uwagę nie tylko na typy danych, powinieneś również zidentyfikować inne brakujące obiekty. Spróbuj zaproponować również pewne obiekty biznesowe, czyli nie tylko takie, dzięki którym jesteśmy pewni co do logicznej spójności danych. W tej lekcji nie koncentruj się na indeksach, ponieważ na ten moment nie wiemy, jakie będą potrzebne.
3. Przypomnij sobie również to, co mówiłem ci w lekcji czwartej, kiedy poprosiłem Cię o być może jeden jeszcze obiekt dla tabeli EmailPerson. To wykracza poza skrypty, które mamy, ale zaproponuj, co można byłoby w tym modelu jeszcze poprawić. Ja znalazłem tam dwa możliwe podejścia, o których chciałbym z tobą podyskutować na Facebooku.

Zapraszam Cię w takim razie do drugiego modułu, które w całości poświęcę indeksowane. Do zobaczenia.