Lidia Sobońska
Aleksandra Sypuła
Arkadiusz Kojtek
Patryk Zdziech

BAZA DANYCH BD1/PAP

Struktura bazy danych

Nasza baza danych składa się z osiemnastu tabel.
Pięć tabel przechowujących informacje dla których nie przewidujemy większych zmian, będą jedynie służyć jako źródło referencji dla pozostałych informacji w bazie, są to:
CARD_TYPES - przechowuje wybrane przez nas typy dostępnych kart
ACCOUNT_TYPES - przechowuje zbiór dostępnych w naszej aplikacji kont, które mogą zostać założone przez klienta, na przykład osobiste, studenckie, dla seniora etc.
TRANSACTION_TYPE - przechowuje informacje o dostępnych rodzajach transakcji czyli wpłatach, wypłatach oraz przelewach z i do konta.
CURRENCIES - przechowuje listę wykorzystywanych przez nas walut wraz z kursami, na ten moment korzystamy tylko z dziesięciu.
PROFESSION - przechowuje informacje o dostępnych na ten moment w naszej bazie danych zawodach.
Pozostałe trzynaście tabel będzie zmieniało swoją zawartość wraz z działaniem bazy danych. Wśród tych tabel znajdują się między innymi tabele z informacjami o klientach, pracownikach czy danymi osobistymi

Dodatkowo aby ułatwić pracę z danymi utworzyliśmy sześć różnych widoków zawierających informacje, które służą nam do ułatwienia pracy z danymi. Np widok **v_loans** zawiera połączone informacje z tabel **services_info** oraz **loans** co znacząco zwiększa przejrzystość odczytywanych danych w przypadku korzystania z procedur, które polegają na informacjach rozrzuconych pomiędzy wieloma tabelami.

W trakcie tworzenia bazy danych zdecydowaliśmy się wydzielenie danych wspólnych pomiędzy tabelami do oddzielnych tabel. W ten sposób zostały utworzone dwie dodatkowe.
PERSONAL_DATA zawiera informacje występujące zarówno dla klienta jak i pracownika
SERVICES_INFO zawiera informacje wspólne dla obu usług dostępnych w naszej aplikacji to jest kredytów oraz kont bankowych.
Przy czym, aby podkreślić integralność wspomnianych danych z tymi tabela użyliśmy relacji 1-1.

Jednak ostatecznie koncept korzystania z tych relacji musiał zostać zmieniony gdy zaczęliśmy zastanawiać się nad usuwaniem danych z tabeli - szczególnie klientów i powiązanych z nimi tabel. Po analizie problemu zdecydowaliśmy, że zwykły usuwanie danych z tabeli może być niekorzystne z dwóch powodów:

1. Może istnieć potrzeba przywrócenia danych lub odniesienia się do nich w procesie archiwizacji.
2. Przez istnienie powiązania pomiędzy różnymi kontami podczas transferu pieniędzy pomiędzy kontami, zwykłe usunięcie klienta i powiązanych z nim rekordów doprowadzi do powstania błędnych informacji jak np transfer przychodzący pieniędzy z konta o id null.

Z uwagi na to zdecydowaliśmy że usunięcie klienta będzie wiązało się z przepisaniem starych kont klienta do specjalnego pustego rekordu w tabeli klientów, który będzie istniał wyłącznie jako zaczep dla ‘usuniętych’ kont. Konta przejdą przez tak zwany **soft delete** gdzie w specjalnej kolumnie w **SERVICES_INFO** zostaną one oznaczone jako ‘Closed’. Natomiast dane osobiste o usuwanym kliencie zostaną permanentnie usunięte z bazy danych.

W związku z tym że wiele usuniętych klientów będzie odwoływało się do tego samego pustego rekordu służącego jako zaczep relacja **CLIENTS -> PERSONAL_DATA** została zmieniona z 1 do 1 na 1 do N.

Procedury, funkcje, sekwencje oraz wyzwalacze

W nasze bazie danych umieściliśmy:
Dwie procedury odpowiedzialne za zmianę wartości kursów walut
Cztery funkcje o różnych zastosowaniach, gdzie **F_CONVERT_ACC_CURRENCY** wykorzystywana jest we wcześniej wspomnianych procedurach a pozostałe służą do obliczania niektórych przydatnych informacji związanych z kredytami.
Oraz **siedem wyzwalaczy** o różnych zastosowaniach od sprawdzania poprawności danych po manipulacje danymi w bazie danych w reakcji na niektóre polecenia DML.

W celu wyznaczania wartości dla kolumn klucza głównego poszczególnych tabel zdecydowaliśmy się na wykorzystanie dostępnego w Oracle od wersji 12c słowa kluczowego **IDENTITY.** Tworzy ono sekwencje na własną rękę co zdecydowanie ułatwiło implementacje autoinkrementacji gdyż wykluczyło to konieczność tworzenia specjalnych wyzwalaczy z przypisanymi im sekwencjami.

Niestety w trakcie eksportowania i importowania danych zauważyliśmy pewien problem. Mianowicie jeżeli rekord zostanie utworzony z podaniem już jakiejś wartości dla klucza głównego to sekwencje utworzone przez IDENTITY nie zmienią swojej wartości. Pojawiała się wtedy konieczność ręcznego zwiększenia wartości dla tych sekwencji przed dalszą pracą ponieważ w innym wypadku polecenia INSERT rzucały błędy naruszenia więzów unikatowości. W celu radzenia sobie z tym problemem utworzyliśmy skrypt którego wywołanie przez rozpoczęciem pracy zwiększa wartość z każdej sekwencji do momentu aż przekroczony zostanie próg największego zapisanego klucza głównego.

Import oraz eksport danych.

Do eksportowania danych oraz struktury bazy danych korzystaliśmy z wbudowanego w SQL Developera narzędzia. Ułatwiło to znacznie pracę umożliwiając nam sprawne eksportowanie danych z naszych prywatnych wersji oraz ich import do wspólnej bazy danych. Niestety również tutaj nie obyło się bez komplikacji. Import danych wyeksportowanych przez SQL Developera powoduje pojawienie się licznych powiadomień o błędach związanych ze zbędnymi operacjami ustawienia typu kolumn na '*niepuste*' gdy te już miały takie ustawienie. O ile nie są to błędy wpływające w jakikolwiek sposób na zaimportowane dane, sama baza danych jest spójna pomiędzy importami i nie traci informacji to sam fakt wystąpienia tego typu komunikatów jest warty odnotowania.

Relacje w bazie danych.

Relacje w bazie danych były utworzone z wykorzystaniem schematu logicznego bazy danych. Samo narzędzie pozwoliło na bardzo precyzyjne dostosowanie typu relacji do naszych potrzeb.

Problematyczna jednak okazała się edycja już istniejących relacji. Brak wiedzy jak poprawnie określić opcjonalność danych przy użyciu wyłącznie kodu SQL sprawił, że relacje tworzone w trakcie późniejszej pracy mogą różnić się ustawieniami opcjonalności w zależności od momentu na jakim etapie tworzenia projektu były tworzone.

Jednak korzystanie z wbudowanego narzędzia również przełożyło się na problemy i pewnie niespójności w ostatecznej wersji naszej bazy danych. Klucze obce utworzone poprzez schemat logiczny potrafią mieć bardzo długie nazwy czego nie zauważyliśmy odpowiednio wcześnie. W związku z tym w naszej bazie danych pojawiły się takie nazwy jak: bank_accounts_account_id, transaction_type_type_id itd. które powinny jednak być dużo krótsze gdyż tak długie nazwy wiązać się mogą z licznymi literówkami podczas tworzenia skryptów.
Późniejsze klucze obce, by uniknąć tego ryzyka miały już prostsze nazwy z pominięciem nazwy tabeli do której się odwołują jak np kolumna **Loan_ID** w tabeli **PAYMENT_HISTORY**. Jednak w ten sposób zostaliśmy z dwiema różnymi konwencjami nazw kolumn jednocześnie nie mogąc zdecydować się na zmianę nazw wcześniej utworzonych kolumn.
Głównym powodem pozostawienia dwóch różnych sposobów nazywania kolumn był nieznaczny zysk w porównaniu z czasem jaki trzeba by poświęcić na edycję kodu naszej aplikacji, która tworzona była równolegle z bazą danych.

Technologie

Z wartych odnotowania technologii z jakich korzystaliśmy warto nadmienić sposób szyfrowania. Dane wrażliwe jak hasła, kody pin czy kody CCV szyfrowane są przez nas z wykorzystaniem algorytmu SHA-256 oraz ‘saltingu’. Do danych, które mają zostać zaszyfrowane dodajemy najpierw ustalony przez nas sufiks - w naszym wypadku ‘bd1’. Po połączeniu tych stringów poddajemy jest szyfrowaniu z użyciem algorytmu SHA-256 po czym te dane dopiero umieszczamy lub porównujemy z odpowiednimi informacjami w tabeli w bazie danych.

Testy

W ramach testów utworzony został skrypt (tests.sql) z różnego rodzaju zapytaniami, od tych testujących działanie utworzonych przez nas procedur, po testowanie funkcji oraz wyzwalaczy.

W testach umieściliśmy również przedstawienie działania utworzonych przez nas kursorów. Jednak jest to jedyne wykorzystanie kursorów przez nas gdyż nie mogliśmy dla nich znaleźć żadnego zastosowania, w którym miałoby ono szansę osiągać lepsze wyniki niż zwykłe zapytania niekorzystające z odczytywania danych wiersz po wierszu.

Ponadto w skrypcie tym znajdują się zapytania testujące działanie grupowania, łączenia czy filtrowania danych.