2022/ 2023

BAZA DANYCH UPROSZCZONEGO SYSTEMU BANKOWEGO

PROJEKT ZALICZENIOWY KURSU BAZY DANYCH

KONRAD BARAN, PAWEŁ MAŁECKI

Spis treści

[CEL I ZAŁOŻENIA PROJEKTU 2](#_Toc124985035)

[Struktura banku 2](#_Toc124985036)

[Ograniczenia przyjęte przy projektowaniu 2](#_Toc124985037)

[Możliwości 2](#_Toc124985038)

[DIAGRAM ER 3](#_Toc124985039)

[SCHEMAT BAZY DANYCH 3](#_Toc124985040)

[WYPEŁNIENIE BAZY PRZYKŁADOWYM ZESTAWEM REKORDÓW 4](#_Toc124985041)

[DODATKOWE WIĘZY INTEGRALNOŚCI DANYCH (NIE ZAPISANE W SCHEMACIE) 10](#_Toc124985042)

[UTWORZONE INDEKSY 10](#_Toc124985043)

[OPIS STWORZONYCH WIDOKÓW 10](#_Toc124985044)

[OPIS PROCEDUR SKŁADOWANYCH 11](#_Toc124985045)

[OPIS FUNKCJI 11](#_Toc124985046)

[OPIS WYZWALACZY 12](#_Toc124985047)

[TYPOWE ZAPYTANIA 12](#_Toc124985048)

[Zestawienie typowych zapytań użytkownika: 12](#_Toc124985049)

[Zestawienie typowych zapytań administratora: 13](#_Toc124985050)

[APLIKACJA OKIENKOWA DLA SYSTEMU 13](#_Toc124985051)

[Wybór roli użytkownika: 13](#_Toc124985052)

[Perspektywa klienta 13](#_Toc124985053)

[Perspektywa administratora 16](#_Toc124985054)

[OPIS SCHEDULED JOBS 17](#_Toc124985055)

[STRATEGIE PIELĘGNACJI BAZY DANYCH (KOPIE ZAPASOWE) 17](#_Toc124985056)

[SKRYPT TWORZĄCY BAZĘ DANYCH 18](#_Toc124985057)

[--Utworzenie tabel 18](#_Toc124985058)

[--Dodanie widoków 20](#_Toc124985059)

[--Dodanie procedur 23](#_Toc124985060)

[--Dodanie funkcji 31](#_Toc124985061)

[--Dodatnie wyzwalaczy 34](#_Toc124985062)

[--Dodanie Job Scheduler 36](#_Toc124985063)

# CEL I ZAŁOŻENIA PROJEKTU

Celem naszego projektu było utworzenie bazy danych systemu bankowego obsługiwanej przez aplikację kliencką dla klienta banku i pracownika nadzorującego bazę tj. administratora bazy. Stworzono i zaimplementowano szereg funkcjonalności służących do jej sprawnego zarządzania z obu perspektyw.

Schematyczność w obsługiwaniu różnych typów kont, klientów, bankomatów i oddziałów pozwala na proste dodawanie nowego podmiotu do bazy, spełniając założenia dla jego typu.

### Struktura banku

System działa dla wyimaginowanego międzynarodowego banku, posiadającego oddziały w różnych krajach. Oddziałom podlegają bankomaty, w których można przeprowadzać wpłaty i wypłaty. Klienci mają ukończone co najmniej 16 lat. Mogą posiadać wiele kont, a każde z nich może posiadać kilka kart kredytowych (w zależności od typu konta). Dozwolone są przelewy na własne konta. Karty posiadają limity tj. maksymalną sumę, którą można jednorazowo wydać.

### Ograniczenia przyjęte przy projektowaniu

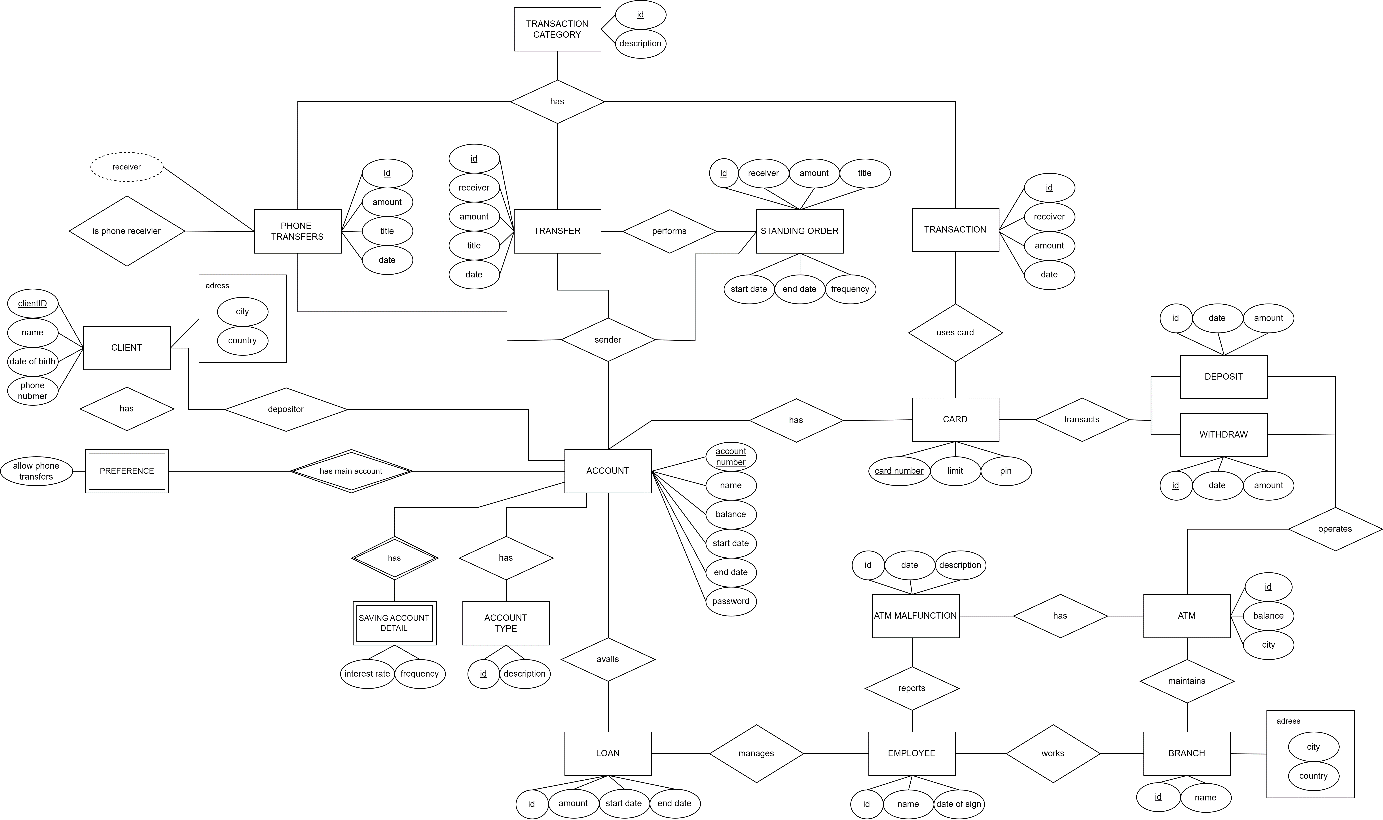
Zakładamy uproszczony system banku, w którym operacje mają naśladować ich główne zamierzenie   
z pominięciem szczegółów technicznych, wymagań wobec klientów (np. zdolność kredytowa)   
i ze znacznym uproszczeniem zabezpieczeń (dowolne hasło 20 znaków), które w prawdziwym systemie bankowym stanowią kluczowy element. Pominiętym zostało również weryfikację poprawności danych do przelewów z kontami zewnętrznymi spoza naszego systemu bankowego.

### Możliwości

Aplikacja dla klienta pozwala na zarządzanie kontem bankowym, wykonywanie prostych przelewów oraz wpłaty i wypłaty w bankomatach lokalnych oddziałów. Możliwości przepływu pieniędzy zostały wzbogacone o przelewy na numer telefonu, przelewy na własne konto oraz zlecenie stałych przelewów. Klient z kontem typu różnym od konta dla młodzieży może również pobrać pożyczkę. Oprócz tego istnieje możliwość zmiany hasła konta lub PINu dla karty kredytowej. Wprowadziliśmy opcję sprawdzania historii konta, z uwzględnieniem uproszczonych filtrów wyszukiwania.

Ze strony administratora bazy istnieje szereg operacji służących zarządzaniem klientami oraz lokalnymi oddziałami banku. Przewidziano funkcjonalności takie jak dodawanie nowego pracownika, oddziału oraz bankomatu. Można utworzyć nowy typ konta i kategorię transakcji. Dodatkowo został zaimplementowany system pożyczek. Pracownik nadzorujący bazę może utworzyć bądź zamknąć konto dla nowego lub obecnego klienta banku, monitorować stan wszystkich kont oraz bankomatów  
 z podziałem na różne statystyki. W razie wykrycia awarii jednego z bankomatów pracownik może zgłosić jego awarię.

# DIAGRAM ER



# SCHEMAT BAZY DANYCH

Obraz zawierający tekst, niebo, mapa

Opis wygenerowany automatycznie

# WYPEŁNIENIE BAZY PRZYKŁADOWYM ZESTAWEM REKORDÓW

Do celów testowania oraz zilustrowania przykładowego działania systemu utworzony został pokaźny zbiór danych losowych, które spełniają założenia oraz wypełniają wszystkie tabele. Poniżej zostały przedstawione fragmenty zapytań wypełniających bazę. A 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11 1 1 1

SET IDENTITY\_INSERT [Branches] ON

INSERT INTO [Branches] (BranchID, Name, City, Country) VALUES

(1,'Branch no. 119 in Balingueo','Balingueo','Philippines'),

(2,'Branch no. 97 in Boyle','Boyle','Ireland'),

(3,'Branch no. 30 in Ribeiro','Ribeiro','Portugal'),

(4,'Branch no. 96 in Manicaragua','Manicaragua','Cuba'),

(5,'Branch no. 83 in Sulang Tengah','Sulang Tengah','Indonesia'),

(6,'Branch no. 265 in Hidalgo','Hidalgo','Mexico'),

(7,'Branch no. 297 in Jhumra','Jhumra','Pakistan'),

(8,'Branch no. 259 in Kraków','Kraków','Poland'),

(9,'Branch no. 193 in Tsiombe','Tsiombe','Madagascar'),

(10,'Branch no. 292 in Baranowo','Baranowo','Poland'),

…

(25,'Branch no. 47 in Hamakita','Hamakita','Japan')

SET IDENTITY\_INSERT [Branches] OFF

SET IDENTITY\_INSERT [Employees] ON

INSERT INTO [Employees] (EmployeeID, Name, DateOfSign, BranchID) VALUES

(1,'Iolanthe Widdocks',CONVERT(DATE,'17.02.2007',105),22),

(2,'Gus Benitti',CONVERT(DATE,'03.03.2006',105),15),

(3,'Thedric Rawood',CONVERT(DATE,'19.09.2011',105),23),

(4,'Xenia Balassa',CONVERT(DATE,'07.04.2019',105),11),

(5,'Corinna Caldaro',CONVERT(DATE,'05.01.2003',105),23),

(6,'Hilton Titmuss',CONVERT(DATE,'02.05.2005',105),4),

(7,'Gibb Gert',CONVERT(DATE,'01.09.2013',105),22),

(8,'Abigael Garside',CONVERT(DATE,'09.01.2005',105),5),

(9,'Rosalinde Ganniclifft',CONVERT(DATE,'07.12.2022',105),16),

(10,'Maurita Ogger',CONVERT(DATE,'14.09.2017',105),11),

…

(112,'Oren Beach',CONVERT(DATE,'16.03.2007',105),6)

SET IDENTITY\_INSERT [Employees] OFF

SET IDENTITY\_INSERT [ATMs] ON

INSERT INTO [ATMs] (ATMID, CurrentBalance, SupervisorDepartment, City) VALUES

(1,5691,5,'Kalipare'),

(2,4546,2,'Cobh'),

(3,7575,11,'Curahpacul Satu'),

(4,11833,3,'Castainço'),

(5,1933,5,'Kuniran'),

(6,4472,21,'Itapemirim'),

(7,6400,23,'La Rochelle'),

(8,9694,11,'Cimara'),

(9,8927,22,'Baylo'),

(10,2672,5,'Muesanaik'),

…

(87,2014,12,'Czerniewice')

SET IDENTITY\_INSERT [ATMs] OFF

SET IDENTITY\_INSERT [ATMsMalfunctions] ON

INSERT INTO [ATMsMalfunctions] (ReportID, ATMID, [Description], [Date], ReportingEmployee) VALUES

(1,86,'Faulty dispenser',CONVERT(DATE,'08-09-2013',105),97),

(2,40,'Faulty dispenser',CONVERT(DATE,'03-11-2013',105),59),

(3,75,'Worn out card reader',CONVERT(DATE,'04-09-2018',105),97),

(4,43,'Faulty dispenser',CONVERT(DATE,'08-12-2021',105),41),

(5,23,'Software glitch',CONVERT(DATE,'13-10-2022',105),105),

(6,48,'Broken keypad',CONVERT(DATE,'21-11-2012',105),38),

(7,43,'Faulty dispenser',CONVERT(DATE,'07-04-2012',105),33),

(8,63,'Receipt malfunctions',CONVERT(DATE,'01-01-2012',105),12),

(9,81,'Worn out card reader',CONVERT(DATE,'18-04-2007',105),14),

(10,60,'Broken keypad',CONVERT(DATE,'16-10-2015',105),13),

…

(97,60,'Software glitch',CONVERT(DATE,'11-09-2020',105),13)

SET IDENTITY\_INSERT [ATMsMalfunctions] OFF

SET IDENTITY\_INSERT [Clients] ON

INSERT INTO [Clients] (ClientID, Name, DateOfBirth, City, Country, PhoneNumber) VALUES

(1,'Phil Lofting',CONVERT(DATE,'30-03-1995',105),'Krajan Dua Putukrejo','Indonesia','+62697945465'),

(2,'Filbert Bassingham',CONVERT(DATE,'21-01-1988',105),'Esperanza','Dominican Republic','+1614187413'),

(3,'Osmond Culkin',CONVERT(DATE,'08-06-1977',105),'Tacoma','United States','+1253714047'),

(4,'Trina Ivanyushin',CONVERT(DATE,'26-10-2006',105),'Remas','Albania','+355363144224'),

(5,'Sukey Jack',CONVERT(DATE,'24-08-1967',105),'Raszowa','Poland','+48156764940'),

(6,'Diahann Broz',CONVERT(DATE,'09-05-2000',105),'Zhukovskiy','Russia','+7354293872'),

(7,'Gualterio Castagnaro',CONVERT(DATE,'06-09-1977',105),'Huoqiu Chengguanzhen','China','+86775132185'),

(8,'Jorey Chumley',CONVERT(DATE,'02-12-2003',105),'Xiaohe','China','+86229326632'),

(9,'Chlo Donnett',CONVERT(DATE,'25-11-1973',105),'Hongshan','China','+86368625073'),

(10,'Laryssa Roebottom',CONVERT(DATE,'15-02-1969',105),'Smedjebacken','Sweden','+46600402366'),

…

(250,'Bobbie Rosenthal',CONVERT(DATE,'21-01-1988',105),'Kętrzyn','Poland','+48736787227')

SET IDENTITY\_INSERT [Clients] OFF

SET IDENTITY\_INSERT [AccountTypes] ON

INSERT INTO [AccountTypes] (AccountType, Description) VALUES

(1,'personal'),

(2,'for minors'),

(3,'saving'),

(4,'business')

SET IDENTITY\_INSERT [AccountTypes] OFF

INSERT INTO [Accounts] VALUES

('AD9413090105NCMC3VM1EDKD',1,'Cardguard',1,0,CONVERT(DATE,'22-11-2021',105),CONVERT(DATE,'21-08-2022',105),'AHIMGGAOkpo73zRml6mI'),

('CY9215826814N5WF0EAUVLRJNYEP',2,'Wrapsafe',4,0,CONVERT(DATE,'01-09-2021',105),NULL,'HKsbuz0x818qw52YmZI3'),

('IS467634787117169699889043',3,'Hatity',4,0,CONVERT(DATE,'28-06-2021',105),CONVERT(DATE,'17-11-2021',105),'uF4DGAb7cB6fg30i7rrt'),

('CY12037553938HANCACNARCJXUQ2',4,'Span',3,0,CONVERT(DATE,'15-07-2021',105),NULL,'t9Fci5AZCyWkX45gfQuH'),

('PT07106690676316508649610',4,'Wrapsafe',1,0,CONVERT(DATE,'21-10-2021',105),NULL,'UgGj86O4FBIHvO131y6m'),

('FR45266161274652TMVIVPQ5L01',5,'Bitwolf',1,0,CONVERT(DATE,'03-08-2021',105),CONVERT(DATE,'30-03-2022',105),'E0V0Or1oFGldbh24FNSb'),

('LT696654847411372478',6,'Bitchip',1,0,CONVERT(DATE,'16-06-2021',105),NULL,'tP5489h517q6U8ne2zI7'),

('SI97275105530910376',7,'Lotstring',3,0,CONVERT(DATE,'05-08-2021',105),NULL,'u46Ts48r44gLaOm8Tx27'),

('ES3876567973341776820457',8,'Rank',1,0,CONVERT(DATE,'10-05-2022',105),NULL,'s37kZAVbJ56CUWmF5aWr'),

('PT29499781625205195427425',9,'Zontrax',3,0,CONVERT(DATE,'12-07-2021',105),NULL,'O5bWYmpwi46LjrCabX1B'),

…

('SI90561448213410524',250,'Alphazap',3,0,CONVERT(DATE,'22-02-2021',105),CONVERT(DATE,'24-08-2021',105),'wO1urKZBYjFP3K7vg6Xb')

INSERT INTO [Preferences] VALUES

(1,NULL,0),

(2,'CY9215826814N5WF0EAUVLRJNYEP',1),

(3,NULL,0),

(4,'PT07106690676316508649610',1),

(5,NULL,0),

(6,'LT696654847411372478',1),

(7,'SI97275105530910376',1),

(8,'ES3876567973341776820457',1),

(9,'PT29499781625205195427425',1),

(10,'AE953455736311284747558',1),

…

(250,NULL,0)

INSERT INTO [Cards] VALUES

('3553394739569566','AD9413090105NCMC3VM1EDKD',100000,'4801'),

('5108759701882616','AD9413090105NCMC3VM1EDKD',100000,'3395'),

('5610448283291799','AD9413090105NCMC3VM1EDKD',100000,'7989'),

('6767194131657216','CY9215826814N5WF0EAUVLRJNYEP',100000,'7879'),

('490512906359513746','IS467634787117169699889043',100000,'5203'),

('67629884540120505','CY12037553938HANCACNARCJXUQ2',100000,'8448'),

('5602214913961221','PT07106690676316508649610',100000,'6919'),

('3585389241553134','FR45266161274652TMVIVPQ5L01',100000,'0603'),

('3553021434179678','FR45266161274652TMVIVPQ5L01',100000,'4605'),

('3539594910319199','LT696654847411372478',100000,'1577'),

…

('3531018699110411','SI90561448213410524',100000,'0442')

SET IDENTITY\_INSERT [TransactionCategories] ON

INSERT INTO [TransactionCategories] (CategoryID, Description) VALUES

(1,'Income'),

(2,'Housing'),

(3,'Home Services'),

(4,'Utilities'),

(5,'Household Items'),

(6,'Food'),

(7,'Transportation'),

(8,'Health'),

(9,'Kids'),

(10,'Pets'),

…

(19,'Debt Payments')

SET IDENTITY\_INSERT [TransactionCategories] OFF

SET IDENTITY\_INSERT [Deposits] ON

INSERT INTO [Deposits] (OperationID, Card, Amount, ATMID, [Date]) VALUES

(1,'0604041071005273611',2350,28,CONVERT(DATE,'27-04-2022',105)),

(2,'0604041071005273611',730,77,CONVERT(DATE,'05-04-2022',105)),

(3,'0604208558230577674',1910,18,CONVERT(DATE,'01-07-2021',105)),

(4,'0604621349629116',1170,10,CONVERT(DATE,'05-12-2022',105)),

(5,'0604621349629116',1140,51,CONVERT(DATE,'13-05-2022',105)),

(6,'0604621349629116',870,33,CONVERT(DATE,'11-08-2022',105)),

(7,'0604621349629116',2120,8,CONVERT(DATE,'08-09-2021',105)),

(8,'0604745800328270',370,80,CONVERT(DATE,'09-07-2021',105)),

(9,'0604745800328270',1330,72,CONVERT(DATE,'29-11-2021',105)),

(10,'0604745800328270',860,23,CONVERT(DATE,'12-03-2022',105)),

…

(1365,'6771510211345444174',1750,24,CONVERT(DATE,'31-10-2021',105))

SET IDENTITY\_INSERT [Deposits] OFF

SET IDENTITY\_INSERT [Withdraws] ON

INSERT INTO [Withdraws] (OperationID, Card, Amount, ATMID, [Date]) VALUES

(1,'6767733260832950879',90,13,CONVERT(DATE,'22-05-2022',105)),

(2,'6767733260832950879',1100,23,CONVERT(DATE,'05-06-2022',105)),

(3,'6767733260832950879',460,17,CONVERT(DATE,'20-07-2022',105)),

(4,'6767733260832950879',170,56,CONVERT(DATE,'21-08-2022',105)),

(5,'3576049661406996',570,22,CONVERT(DATE,'27-09-2022',105)),

(6,'3576049661406996',930,84,CONVERT(DATE,'15-11-2022',105)),

(7,'3568688788664829',340,63,CONVERT(DATE,'15-10-2021',105)),

(8,'3568688788664829',1000,43,CONVERT(DATE,'08-07-2022',105)),

(9,'3553394739569566',700,27,CONVERT(DATE,'24-01-2022',105)),

(10,'5610448283291799',190,1,CONVERT(DATE,'02-02-2022',105)),

…

(993,'3560656041219275',130,13,CONVERT(DATE,'20-06-2022',105))

SET IDENTITY\_INSERT [Withdraws] OFF

SET IDENTITY\_INSERT [Loans] ON

INSERT INTO [Loans] (LoanID, AccountID, Amount, StartDate, EndDate, ServingEmployee) VALUES

(1,'AD0357942949XKSMVLBOOIBA',30000,CONVERT(DATE,'14-01-2023',105),CONVERT(DATE,'14-01-2023',105),79),

(2,'AD3436784333NJ8STCMDIPRF',3300,CONVERT(DATE,'25-03-2022',105),CONVERT(DATE,'28-09-2022',105),2),

(3,'AD3436784333NJ8STCMDIPRF',39000,CONVERT(DATE,'20-07-2022',105),CONVERT(DATE,'29-08-2022',105),55),

(4,'AD6546708196DCNFM1FYRYWR',190000,CONVERT(DATE,'11-08-2021',105),CONVERT(DATE,'09-12-2022',105),15),

(5,'AD9413090105NCMC3VM1EDKD',10200,CONVERT(DATE,'23-03-2022',105),CONVERT(DATE,'22-05-2022',105),12),

(6,'AE208442104089089423018',6400,CONVERT(DATE,'08-12-2022',105),CONVERT(DATE,'12-12-2022',105),16),

(7,'AL0870006466BDDYHJJGOUMQB7A6',38000,CONVERT(DATE,'02-05-2022',105),CONVERT(DATE,'05-05-2022',105),76),

(8,'AL33908744009C521XY5QXTPUATG',15000,CONVERT(DATE,'21-07-2022',105),CONVERT(DATE,'14-09-2022',105),60),

(9,'AL5959600739QLBVKSV5UR8L5MCU',9700,CONVERT(DATE,'21-09-2022',105),CONVERT(DATE,'24-10-2022',105),39),

(10,'AL7735553150TGPITLB0SYRDI9PS',43000,CONVERT(DATE,'10-03-2022',105),CONVERT(DATE,'20-10-2022',105),98),

…

(202,'TR7857618UXWCWBI4V4IUMMBD5',3700,CONVERT(DATE,'17-03-2022',105),CONVERT(DATE,'06-10-2022',105),97)

SET IDENTITY\_INSERT [Loans] OFF

SET IDENTITY\_INSERT [PhoneTransfers] ON

INSERT INTO [PhoneTransfers] (TransferID, Sender, PhoneReceiver, Amount, Title, [Date], Category) VALUES

(1,'AD9413090105NCMC3VM1EDKD','+86311454257',24,'globe',CONVERT(DATE,'23-03-2022',105),14),

(2,'AD9413090105NCMC3VM1EDKD','+351741434882',255,'painting',CONVERT(DATE,'09-08-2022',105),7),

(3,'CY9215826814N5WF0EAUVLRJNYEP','+86247969462',275,'christmas ornament',CONVERT(DATE,'20-12-2021',105),13),

(4,'CY9215826814N5WF0EAUVLRJNYEP','+63470868442',477,'bowl',CONVERT(DATE,'04-07-2022',105),6),

(5,'CY9215826814N5WF0EAUVLRJNYEP','+261972212399',74,'duffel bag',CONVERT(DATE,'30-06-2022',105),14),

(6,'IS467634787117169699889043','+48999197762',228,'pedestal',CONVERT(DATE,'26-09-2021',105),16),

(7,'IS467634787117169699889043','+86170742330',395,'mace',CONVERT(DATE,'17-11-2021',105),3),

(8,'IS467634787117169699889043','+81388309971',335,'spool of ribbon',CONVERT(DATE,'08-11-2021',105),14),

(9,'CY12037553938HANCACNARCJXUQ2','+261972212399',12,'tiger',CONVERT(DATE,'30-10-2022',105),14),

(10,'CY12037553938HANCACNARCJXUQ2','+81509432607',450,'knitting needles',CONVERT(DATE,'25-07-2021',105),7),

…

(1000,'MU31UEWC1168158191908027236MKS','+62342261629',78,'purse',CONVERT(DATE,'31-01-2022',105),14)

INSERT INTO [PhoneTransfers] (TransferID, Sender, PhoneReceiver, Amount, Title, [Date], Category) VALUES

(1001,'MU31UEWC1168158191908027236MKS','+86965300625',225,'spatula',CONVERT(DATE,'15-01-2022',105),17),

(1002,'MU31UEWC1168158191908027236MKS','+7712613785',326,'box of chocolates',CONVERT(DATE,'03-02-2022',105),1),

(1003,'MU31UEWC1168158191908027236MKS','+967507351165',41,'feather',CONVERT(DATE,'09-03-2022',105),17),

(1004,'CH6569821IBPATSHZIOIH','+264343441963',254,'key chain',CONVERT(DATE,'02-10-2022',105),15),

(1005,'CH6569821IBPATSHZIOIH','+86533742427',440,'rusty nail',CONVERT(DATE,'11-04-2022',105),15),

(1006,'CH6569821IBPATSHZIOIH','+62478307271',480,'case',CONVERT(DATE,'09-05-2022',105),18),

(1007,'CH6569821IBPATSHZIOIH','+7586777818',471,'tree',CONVERT(DATE,'02-09-2022',105),15),

(1008,'FR273037025062NQTUZGL8HIC68','+7121813198',400,'sheep',CONVERT(DATE,'30-05-2022',105),18),

(1009,'FR273037025062NQTUZGL8HIC68','+63820742955',466,'chocolate',CONVERT(DATE,'11-06-2022',105),5),

(1010,'FR273037025062NQTUZGL8HIC68','+351943860770',165,'hand mirror',CONVERT(DATE,'18-03-2022',105),18),

…

(1071,'MR2233184551320834197541398','+420145477570',336,'puddle',CONVERT(DATE,'12-09-2022',105),1)

SET IDENTITY\_INSERT [PhoneTransfers] OFF

INSERT INTO [SavingAccountDetails] (AccountID, InterestRate, Frequency) VALUES

('AE236837503028130721519','yearly',3.8),

('AE953455736311284747558','half year',4.9),

('AT226036253155779986','yearly',1.3),

('AT312563855247280428','monthly',2.0),

('AT678813955725840852','monthly',2.9),

('AZ32WDFXIMHARYKZEWVKS5QG5GQ7','monthly',4.5),

('AZ64HEGTJCSUZBLP4MODQD5TXA2Y','quarter',1.2),

('AZ73BFASXO7TAC1SDXQLYB2RTNGJ','monthly',3.5),

('BG13JEIC509423NAXUKOLW','half year',5.4),

('BG80BGKW2232103ST3GTY6','half year',3.5),

…

('VG35WZDA4882933705963825','quarter',4.3)

SET IDENTITY\_INSERT [Transactions] ON

INSERT INTO [Transactions] (TransactionID, UsedCard, Receiver, Amount, [Date], Category) VALUES

(1,'0604041071005273611','MD5860226340534482710520',21,CONVERT(DATE,'17-04-2022',105),6),

(2,'3545214934208144','NL41MIGS2389939465',18,CONVERT(DATE,'06-04-2021',105),8),

(3,'3545214934208144','SI98397065207785717',573,CONVERT(DATE,'12-04-2021',105),4),

(4,'201694443691883','DE59418084112530380214',10,CONVERT(DATE,'14-05-2021',105),4),

(5,'3545214934208144','IQ86BGCQ605758161158884',742,CONVERT(DATE,'17-05-2021',105),16),

(6,'3545214934208144','JO33CXAK4019486286072200895281',289,CONVERT(DATE,'23-05-2021',105),17),

(7,'201694443691883','BY91686859148228813080886167',346,CONVERT(DATE,'27-06-2021',105),3),

(8,'3574144298800935','LV27TPYW5658482129676',659,CONVERT(DATE,'18-08-2021',105),13),

(9,'0604621349629116','MD3516041554886658371484',473,CONVERT(DATE,'29-08-2021',105),17),

(10,'6333260678108171','MK73620077620369144',475,CONVERT(DATE,'03-12-2021',105),6),

…

(1265,'6771162572428662052','RS07756783329260007681',1018,CONVERT(DATE,'25-06-2022',105),10)

SET IDENTITY\_INSERT [Transactions] OFF

SET IDENTITY\_INSERT [StandingOrders] ON

INSERT INTO [StandingOrders] (StandingOrdersID, Sender, Receiver, Amount, Title, Frequency, StartDate, EndDate) VALUES

(1,'AD9413090105NCMC3VM1EDKD','HU90888651622857489620287790',53,'Security services',2,CONVERT(DATE,'09-04-2022',105),CONVERT(DATE,'17-06-2022',105)),

(2,'CY9215826814N5WF0EAUVLRJNYEP','TN9828137049969786960962',241,'Marketing services',5,CONVERT(DATE,'22-05-2022',105),CONVERT(DATE,'23-11-2022',105)),

(3,'FR45266161274652TMVIVPQ5L01','GR522354528C95EVZDAXCLHWO6W',354,'Construction services',2,CONVERT(DATE,'02-01-2022',105),CONVERT(DATE,'18-03-2022',105)),

(4,'FR45266161274652TMVIVPQ5L01','SE2575710596688421800564',256,'Travel services',5,CONVERT(DATE,'20-08-2021',105),CONVERT(DATE,'04-02-2022',105)),

(5,'SI97275105530910376','LI9371923OXAYHIGPTSWF',447,'Event planning services',14,CONVERT(DATE,'26-08-2021',105),CONVERT(DATE,'06-11-2022',105)),

(6,'ES3876567973341776820457','BR6719619074764247260759846PC',242,'Security services',4,CONVERT(DATE,'30-06-2022',105),CONVERT(DATE,'01-11-2022',105)),

(7,'SM79Q3694121719TOU8ASQA0WSB','FR666537258783X8OMC04VFTC42',37,'Construction services',3,CONVERT(DATE,'05-01-2022',105),CONVERT(DATE,'22-04-2022',105)),

(8,'AE953455736311284747558','FR219161520100MAECISS28QI76',402,'Finance services',2,CONVERT(DATE,'08-03-2022',105),CONVERT(DATE,'28-05-2022',105)),

(9,'FI2219198455116224','DK1296358774419108',166,'Legal services',11,CONVERT(DATE,'11-04-2021',105),CONVERT(DATE,'01-04-2022',105)),

(10,'FR980228933378SS3VLX9UACU28','AT406845176291948939',319,'Insurance services',2,CONVERT(DATE,'22-03-2022',105),CONVERT(DATE,'11-06-2022',105)),

…

(338,'SI90561448213410524','PT13236191257141033561499',233,'Finance services',3,CONVERT(DATE,'05-05-2021',105),CONVERT(DATE,'16-08-2021',105))

SET IDENTITY\_INSERT [StandingOrders] OFF

SET IDENTITY\_INSERT [Transfers] ON

INSERT INTO [Transfers] (TransferID, Sender, Receiver, Amount, Title, [Date], Category, StandingOrder) VALUES

(1,'AD9413090105NCMC3VM1EDKD','FR794576365824OBDHH9XQU3Z48',3302,'Restaurant',CONVERT(DATE,'06-04-2022',105),14,NULL),

(2,'AD9413090105NCMC3VM1EDKD','AE953455736311284747558',1157,'Yacht',CONVERT(DATE,'16-04-2022',105),11,NULL),

(3,'AD9413090105NCMC3VM1EDKD','LB3423188JABVMUC6GSTMJQWP8DR',801,'China',CONVERT(DATE,'07-04-2022',105),15,NULL),

(4,'CY9215826814N5WF0EAUVLRJNYEP','FR569310056713SC8KZOLNQYJ12',562,'Television',CONVERT(DATE,'20-09-2022',105),19,NULL),

(5,'CY9215826814N5WF0EAUVLRJNYEP','AT312563855247280428',278,'Monkey',CONVERT(DATE,'25-05-2022',105),15,NULL),

(6,'CY9215826814N5WF0EAUVLRJNYEP','FR938173448585BAIZQZ14X8G91',925,'Morning',CONVERT(DATE,'07-10-2022',105),18,NULL),

(7,'IS467634787117169699889043','SM21J8226711495E7QWVSFSIA8R',1738,'Napkin',CONVERT(DATE,'16-11-2021',105),19,NULL),

(8,'IS467634787117169699889043','CY9215826814N5WF0EAUVLRJNYEP',832,'Nigeria',CONVERT(DATE,'05-11-2021',105),11,NULL),

(9,'IS467634787117169699889043','FR794576365824OBDHH9XQU3Z48',1516,'Eye',CONVERT(DATE,'17-11-2021',105),9,NULL),

(10,'CY12037553938HANCACNARCJXUQ2','MU31UEWC1168158191908027236MKS',460,'Nail',CONVERT(DATE,'14-11-2021',105),3,NULL),

…

(4374,'SI90561448213410524','PT13236191257141033561499',233,'Finance services',CONVERT(DATE,'06-08-2021',105),1,338)

SET IDENTITY\_INSERT [Transfers] OFF

# DODATKOWE WIĘZY INTEGRALNOŚCI DANYCH (NIE ZAPISANE W SCHEMACIE)

Oprócz oczywistych więzów widocznych w kwerendach tworzących, baza posiada również niezapisane więzy integralności. W skład tych więzów wchodzą:

* odrzucenie wszelkich operacji, które:
  + są o niedodatniej kwocie,
  + wykonują transakcje na to samo konto,
  + wykonują transakcje o kwocie przekraczającej aktualny stan konta,
  + przy zleceniach stałych planowanie operacji o dacie mającej miejsce w przeszłości,
  + przy przelewach na telefon odbiorca musi akceptować ten typ przelewu,
  + przy transakcjach kartą wartość nie może przekroczyć limitu karty,
  + bankomat musi mieć wystarczająco dużo gotówki aby ją wypłacić,
  + konto wykonujące operację nie może być zamknięte,
* przy dodawaniu klientów, kont, kart:
  + minimalna długość hasła dla konta,
  + minimalna długość PINu dla karty,
  + minimalny wiek nowego klienta,
  + kredyt nie może kończyć się w przeszłości,
  + osoba wystawiająca kredyt musi być pracownikiem banku,
  + opisy oddziałów, imiona pracowników, nazwy miast, nazwy typów kont i typów kategorii muszą być dłuższe niż 2 znaki,
* przy zamykaniu konta trzeba podać prawidłowe hasło oraz konto musi być otwarte,
* przy zmianie PINu i hasła konta należy podać poprawne stare hasło/PIN,
* przy dodawaniu raportu o usterce bankomat i pracownik dodający muszą istnieć.

# UTWORZONE INDEKSY

Więzy integralności oraz duża zależność między tablicami pozwoliła ograniczyć liczbę ręcznie utworzonych indeksów. Indeksy zostały utworzone głównie na kolumnach będących kluczami obcymi odpowiadającym numerom konta, identyfikatorach klienta oraz identyfikatorach karty. Decyzja związana jest z koniecznością częstego sprawdzania tych danych oraz z pewnością, że te dane nie ulegną zmianie. Z podobnych powodów został utworzony indeks w tabeli Clients na kolumnie PhoneNumbers, ta kolumna jako jedyna w tym zestawieniu nie jest kluczem obcym.

# OPIS STWORZONYCH WIDOKÓW

|  |  |
| --- | --- |
| Baza obecnie posiada 11 widoków, część z nich jest wykorzystywana bezpośrednio w celu odpowiedzenia na zapytanie klienta. Pozostałe są pomocniczymi widokami wykorzystywanymi w procedurach, funkcjach itd. Wszystkie widoki skupiają się na zebraniu informacji z wielu tabel na temat konkretnego zagadnienia. Nazwy są intuicyjne, łatwo domyślić się przeznaczenia danego widoku. Przykładowo widok o nazwie „SavingAccountsToUpdate” służy zebraniu kont oszczędnościowych, których termin przyznania należności z oprocentowania wypada na aktualną datę.  Z kolei „CardDetails” gromadzi informacje o operacjach dokonanych kartą (tj. wpłaty, wypłaty oraz transakcje)  i wyświetla je w formacie: numer karty, kwota, data, rodzaj operacji. | Obraz zawierający tekst  Opis wygenerowany automatycznie |

# OPIS PROCEDUR SKŁADOWANYCH

|  |  |
| --- | --- |
| Utworzonych zostało 20 procedur składowanych odpowiadających za dodawanie nowych klientów, kont, kart itd., utworzenie nowych przelewów różnego rodzaju, transakcji, zmianę limitów kart, dezaktywację kont oraz tworzenie raportów. Część z nich bazuje na utworzonych widokach. Procedury weryfikują czy dane spełniają narzucone wymagania. Przykładowe z nich to: nowy klient musi mieć ukończone 16 lat, hasło przy tworzeniu nowego konta musi mieć przynajmniej 20 znaków, PIN 4 cyfry, wartości przelewów nie mogą być ujemne, a konta z których przelewy są wysyłane nie mogą być zamknięte i posiadać odpowiednią ilość pieniędzy itd. Dodatkowo procedury „createBackup”, „checkLoans”, „checkSavingAccounts”  i „checkStandingOrders” odpowiadają za wykonywanie cyklicznych operacji i tworzenie kopii zapasowych po przez automatyzację (Scheduled Jobs). | Obraz zawierający stół  Opis wygenerowany automatycznie |

# OPIS FUNKCJI

|  |  |
| --- | --- |
| Baza obecnie posiada 16 funkcji, w tym 13 funkcji zwracających tablice i 3 zwracające wartości skalarne. Podobnie jak w przypadku widoków nie wszystkie są wykorzystywane bezpośrednio w odpowiedzi na zapytanie użytkownika. Większość funkcji tablicowych służy zwracania zestawu danych konkretnie pod danego klienta, placówkę, kartę itp. Sprawia to, że późniejsze wykorzystanie takiej funkcji jest dużo szybsze i prostsze od wykonania podzapytania z dodatkowym warunkiem. Przykładowo funkcja „AccountOperationsByMonth” przyjmuje numer konta i zwraca tablicę prezentujące liczbę operacji danego konta na przestrzeni miesięcy w formacie: miesiąc, rok, liczba operacji. Z kolei funkcje skalarne wykorzystywane są przykładowo podczas procesu logowania w aplikacji klienckiej. Funkcja „IfAccountExists” zwraca wartość typu bit, w zależności czy podany numer konta faktycznie istnieje w naszej bazie, a funkcja „GetPassword” zwyczajnie zwraca hasło w postaci NVARCHAR dla podanego numeru konta. | Obraz zawierający tekst  Opis wygenerowany automatycznie |

# OPIS WYZWALACZY

Po nastąpieniu operacji takich jak wpłaty, wypłaty, płatności kartą przelewu z konto na konto, przelewu na numer telefonu konieczne jest natychmiastowe zapewnienie spójności bazy po przez zaktualizowanie stanu kont nadawców, odbiorców bądź bankomatów. Jeżeli przelew zachodzi pomiędzy klientem naszego banku oraz banku zewnętrznego, aktualizację konta zewnętrznego pomijamy. Jeżeli klient zaciągnie kredyt na jego konto przesyłana jest kwota, którą pożyczył.

Przykładowe wyzwalacze dla operacji przelewu z konto na konto i utworzenia kredytu:

|  |  |
| --- | --- |
|  | Obraz zawierający tekst  Opis wygenerowany automatycznie |

# TYPOWE ZAPYTANIA

### Zestawienie typowych zapytań użytkownika:

|  |  |
| --- | --- |
| Opis | Kwerenda (przykład) |
| Przelew na konto | EXEC addNewTransfer @sender='CH0777012CHPF0RNYLH5D' , @receiver='CH1618474PWEBI8QAGREI' , @amount=1 , @title = 'test' , @category=1 |
| Przelew na telefon | EXEC addNewPhoneTransfer @sender='CH0777012CHPF0RNYLH5D' , @phoneReceiver='123123123' , @amount=1 , @title = '11' , @category=1 |
| Wypłata | EXEC addNewWithdraw @cardID='0604041071005273611' , @amount=1 , @ATM=1 |
| Wpłata | EXEC addNewDeposit @cardID='0604041071005273611' , @amount=1 , @ATM=1 |
| Dodanie stałego zlecenia | EXEC addStandingOrders @sender='CH0777012CHPF0RNYLH5D', @receiver='CH1618474PWEBI8QAGREI', @amount=1, @title='test', @frequency=5, @startDate='2023-01-19', @endDate='2023-06-18' |
| Zmiana hasła konta | EXEC changeAccountPassword @accountID='AD0357942949XKSMVLBOOIBA', @prev\_password='93q0Kz2gZGE8O9Q37bq4', @new\_password='01234567890123456789' |
| Zmiana limitu na karcie | EXEC changeCardLimit @cardID='0604041071005273611', @limit=123, @pin='3504' |
| Zmiana PINu na karcie | EXEC changeCardPIN @cardID='0604041071005273611', @prevPIN='3504', @newPIN='1234' |
| Zestawienie ilości operacji z podziałem na miesiące | SELECT \* FROM ClientOperationsByMonth(1) |
| Zestawienie ilości transakcji wg karty | SELECT \* FROM ClientOperationsByCard(1) |
| Całkowita liczba transferów | SELECT \* FROM ClientTransfersNumber(1) |
| Całkowita liczba przelewów na telefon | SELECT \* FROM ClientPhoneTransfersNumber(1) |
| Zestawienie ilości operacji z podziałem wg typu operacji | SELECT \* FROM ClientOperationsByOperationType(1) |
| Zestawienie przelewów na własne konta | SELECT \* FROM OnOwnAccountsTransfers(4) |
| Zestawienie wszystkich operacji różnych typów | SELECT \* FROM AccountHistory('AD0357942949XKSMVLBOOIBA') |

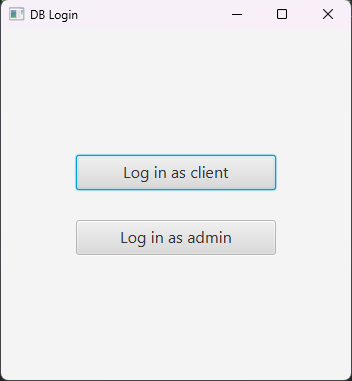
### Zestawienie typowych zapytań administratora:

|  |  |
| --- | --- |
| Opis | Kwerenda (przykład) |
| Dodawanie nowego klienta | EXEC addNewClient @name='A B', @dateOfBirth='2000-01-01', @city='Krakow', @country='PL', @phoneNumber='123123123', @allowPhoneTransfers=1 |
| Dodawanie nowego konta bankowego | EXEC addNewAccount @accountID='PL1234000012340000, @clientID=1, @name='konto', @accountType=1, @password='01234567890123456789' |
| Dodawanie nowej karty debetowe | EXEC addNewCard @cardID='123456789123456789', @accountID='AD0357942949XKSMVLBOOIBA', @limit=1234, @pin='1234' |
| Dodawanie nowego kredytu | EXEC addNewLoan @accountID='AD0357942949XKSMVLBOOIBA', @amount=10000000, @endDate='2050-01-01', @servingEmployee=1 |
| Dodawanie nowego oddziału bankowego | EXEC addNewBranches @name='Branch no. 119 in Krakow', @city='Krakow', @country='PL' |
| Dodanie nowego bankomatu | EXEC addNewATM @currentBalance=123456, @supervisorDepartment=1, @city='ABC' |
| Dezaktywacja konta klienckiego | EXEC disactiveAccount @accountID='AD0357942949XKSMVLBOOIBA', @password='01234567890123456789' |
| Utworzenie nowego zgłoszenia usterki technicznej | EXEC reportATMsMalfunction @ATMID='1', @description='Software glitch', @reportingEmployee=1 |

# APLIKACJA OKIENKOWA DLA SYSTEMU

Zaimplementowano aplikację okienkową pełniącą funkcję zarówno dla klienta banku, jak   
i administratora. Poniżej przedstawiono podstawową obsługę systemu z poziomu aplikacji.

### Wybór roli użytkownika:

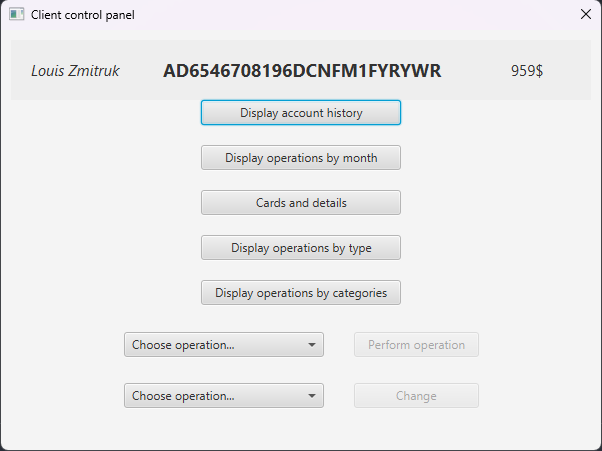


### Perspektywa klienta

1. Uzupełniamy dane logowania na wybrane konto. W przypadku wpisania błędnych danych aplikacja powiadomi o tym użytkownika.

|  |  |
| --- | --- |
| Obraz zawierający tekst  Opis wygenerowany automatycznie |  |

1. Po poprawnym zalogowaniu, ukazuje się panel kontrolny klienta.



1. Po wybraniu typu operacji z rozwijanej listy i kliknięciu przycisku „Perform operation” pojawia się okno dla stosownej operacji. W przykładzie widać operację przelewu. Wpisujemy konto odbiorcy, kwotę i tytuł przelewu, wybieramy kategorię przelewu z rozwijanej listy i zatwierdzamy przyciskiem „Done”.

|  |  |
| --- | --- |
|  |  |

1. Poniżej przykład zestawienia ilości operacji z podziałem wg typu operacji.

|  |  |
| --- | --- |
|  | Obraz zawierający stół  Opis wygenerowany automatycznie |

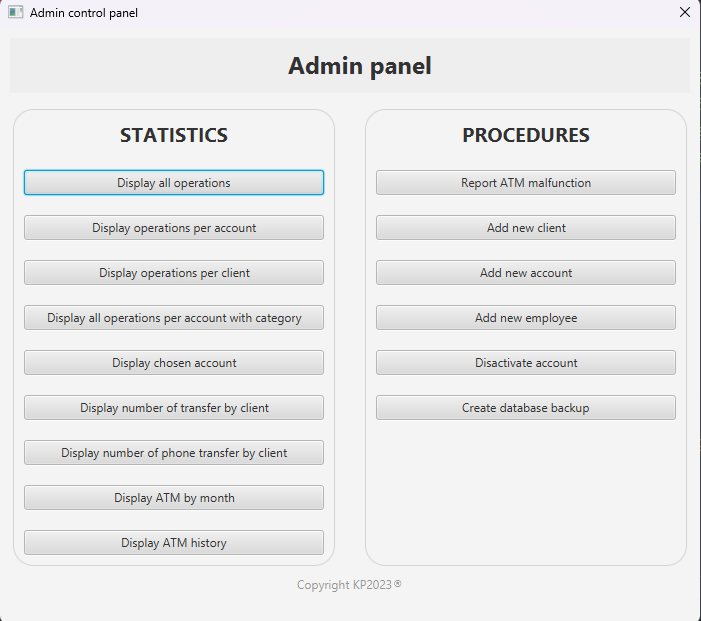
|  |  |
| --- | --- |
|  | Obraz zawierający stół  Opis wygenerowany automatycznie |

1. Poniżej przykład zestawienia ilości operacji z podziałem wg kategorii.
2. Przykład zmiany limitu karty

|  |  |
| --- | --- |
|  | Obraz zawierający tekst  Opis wygenerowany automatycznie |

### Perspektywa administratora

1. Po wyborze strony jako administrator ukazuje się panel administracji podzielony na dwie kategorie: statystyki oraz procedury.



1. Po wybraniu przycisku oznaczonego nazwą operacji pokazuje się nowe okno dialogowe, odpowiednie do wybranej operacji. Poniżej zestawienie 3 przykładowych operacji, kolejno: wyświetlenie historii wybranego bankomatu, zgłoszenie usterki bankomat oraz dodanie nowego klienta:

|  |  |  |  |
| --- | --- | --- | --- |
| Obraz zawierający stół  Opis wygenerowany automatycznie | |  | | --- | | Obraz zawierający tekst  Opis wygenerowany automatycznie | |  | |

# OPIS SCHEDULED JOBS

Scheduled Jobs to czynności wykonujące się automatycznie względem ustalonego wcześniej planu (schedule). W naszym projekcie istnieją 4 takie automatyczne procedury. Wiążę się to z koniecznością posiadania faktycznego serwera aktywnego całodobowo, aby Job Agent (menager automatycznych procedur) działał poprawnie. Procedury wykonywane są zgodnie z dwoma planami:

* dziennym – procedury wykonywane każdego dnia o północy,
* miesięcznym – procedury wykonywane każdego pierwszego dnia miesiąca o północy.

Obecnie obsługiwane Scheduled Jobs to:

* wykonywanie stałych zleceń,
* naliczanie odsetek kont oszczędnościowych w postaci przelewu na to konto,
* uproszczone obsługiwanie kredytów (tj. usunięcie kwoty kredytu z konta w dniu jego zakończenia),
* wykonywanie regularnych kopii zapasowych bazy.

Poniżej przykład utworzenia schematów oraz procedury automatycznych backupów.

|  |  |
| --- | --- |
| Obraz zawierający tekst  Opis wygenerowany automatycznie | Obraz zawierający tekst  Opis wygenerowany automatycznie |

# STRATEGIE PIELĘGNACJI BAZY DANYCH (KOPIE ZAPASOWE)

|  |  |  |
| --- | --- | --- |
|  |  | Nasza baza posiada zaimplementowany system automatycznych comiesięcznych backupów. Backupy są wykonywane każdego pierwszego dnia miesiąca  (dokładny opis działania w podpunkcie Scheduled Jobs). Kopie tworzone są w podanej lokalizacji, a w ich nazwie występuje data utworzenia. Backup można też wykonywać ręcznie w dowolnej chwili z poziomu panelu administratora.  Obraz zawierający tekst  Opis wygenerowany automatycznie |

# SKRYPT TWORZĄCY BAZĘ DANYCH

### --Utworzenie tabel

CREATE TABLE [Clients] (

[ClientID] INT IDENTITY PRIMARY KEY,

[Name] NVARCHAR(100),

[DateOfBirth] DATE,

[City] NVARCHAR(100),

[Country] NVARCHAR(100),

[PhoneNumber] NVARCHAR(100) UNIQUE,

INDEX IdxPhoneNumber(PhoneNumber)

)

CREATE TABLE [AccountTypes] (

[AccountType] INT IDENTITY PRIMARY KEY,

[Description] NVARCHAR(100)

)

CREATE TABLE [Accounts] (

[AccountID] NVARCHAR(100) PRIMARY KEY,

[ClientID] INT FOREIGN KEY REFERENCES [Clients] ([ClientID]),

[Name] NVARCHAR(100),

[AccountType] INT FOREIGN KEY REFERENCES [AccountTypes] ([AccountType]),

[CurrentBalance] INT,

[StartDate] DATE,

[EndDate] DATE,

[Password] NVARCHAR(100),

INDEX IdxClientID(ClientID)

)

CREATE TABLE [Cards] (

[CardID] NVARCHAR(100) PRIMARY KEY,

[Account] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[Limit] INT,

[PIN] NVARCHAR(100)

)

CREATE TABLE [Preferences] (

[ClientID] INT FOREIGN KEY REFERENCES [Clients] ([ClientID]),

[MainAccount] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]) ,

[AllowPhoneTransfer] BIT

INDEX IdxClientID(ClientID),

INDEX IdxAccountID(MainAccount)

)

CREATE TABLE [SavingAccountDetails] (

[AccountID] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[InterestRate] NVARCHAR(100),

[Frequency] FLOAT,

INDEX IdxAccountID(AccountID)

)

CREATE TABLE [Branches] (

[BranchID] INT IDENTITY PRIMARY KEY,

[Name] NVARCHAR(100),

[City] NVARCHAR(100),

[Country] NVARCHAR(100)

)

CREATE TABLE [Employees] (

[EmployeeID] INT IDENTITY PRIMARY KEY,

[Name] NVARCHAR(100),

[DateOfSign] DATE,

[BranchID] INT FOREIGN KEY REFERENCES [Branches] ([BranchID])

)

CREATE TABLE [Loans] (

[LoanID] INT IDENTITY PRIMARY KEY,

[AccountID] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[Amount] MONEY,

[StartDate] DATE,

[EndDate] DATE,

[ServingEmployee] INT FOREIGN KEY REFERENCES [Employees] ([EmployeeID]),

INDEX IdxAccountID(AccountID)

)

CREATE TABLE [ATMs] (

[ATMID] INT IDENTITY PRIMARY KEY,

[CurrentBalance] INT,

[SupervisorDepartment] INT REFERENCES [Branches] ([BranchID]),

[City] NVARCHAR(100)

)

CREATE TABLE [ATMsMalfunctions] (

[ReportID] INT IDENTITY PRIMARY KEY,

[ATMID] INT FOREIGN KEY REFERENCES [ATMs] ([ATMID]),

[Description] NVARCHAR(100),

[Date] DATE,

[ReportingEmployee] INT FOREIGN KEY REFERENCES [Employees] ([EmployeeID])

)

CREATE TABLE [Withdraws] (

[OperationID] INT IDENTITY PRIMARY KEY,

[Card] NVARCHAR(100) FOREIGN KEY REFERENCES [Cards] ([CardID]),

[Amount] MONEY,

[ATMID] INT FOREIGN KEY REFERENCES [ATMs] ([ATMID]),

[Date] DATE,

INDEX IdxCardID(Card)

)

CREATE TABLE [Deposits] (

[OperationID] INT IDENTITY PRIMARY KEY,

[Card] NVARCHAR(100) FOREIGN KEY REFERENCES [Cards] ([CardID]),

[Amount] MONEY,

[ATMID] INT FOREIGN KEY REFERENCES [ATMs] ([ATMID]),

[Date] DATE,

INDEX IdxCardID(Card)

)

CREATE TABLE [TransactionCategories] (

[CategoryID] INT IDENTITY PRIMARY KEY,

[Description] NVARCHAR(100)

)

CREATE TABLE [StandingOrders] (

[StandingOrdersID] INT IDENTITY PRIMARY KEY,

[Sender] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[Receiver] NVARCHAR(100),

[Amount] MONEY,

[Title] NVARCHAR(100),

[Frequency] INT,

[StartDate] DATE,

[EndDate] DATE,

INDEX IdxAccountID(Sender)

)

CREATE TABLE [Transfers] (

[TransferID] INT IDENTITY PRIMARY KEY,

[Sender] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[Receiver] NVARCHAR(100),

[Amount] MONEY,

[Title] NVARCHAR(100),

[Date] DATE,

[Category] INT FOREIGN KEY REFERENCES [TransactionCategories] ([CategoryID]),

[StandingOrder] INT FOREIGN KEY REFERENCES [StandingOrders] ([StandingOrdersID]),

INDEX IdxAccountID(Sender)

)

CREATE TABLE [Transactions] (

[TransactionID] INT IDENTITY PRIMARY KEY,

[UsedCard] NVARCHAR(100) FOREIGN KEY REFERENCES [Cards] ([CardID]),

[Receiver] NVARCHAR(100),

[Amount] MONEY,

[Date] DATE,

[Category] INT FOREIGN KEY REFERENCES [TransactionCategories] ([CategoryID]),

INDEX IdxCardID(UsedCard)

)

CREATE TABLE [PhoneTransfers] (

[TransferID] INT IDENTITY PRIMARY KEY,

[Sender] NVARCHAR(100) FOREIGN KEY REFERENCES [Accounts] ([AccountID]),

[PhoneReceiver] NVARCHAR(100),

[Amount] MONEY,

[Title] NVARCHAR(100),

[Date] DATE,

[Category] INT FOREIGN KEY REFERENCES [TransactionCategories] ([CategoryID])

)

### --Dodanie widoków

IF OBJECT\_ID('SavingAccountsToUpdate', 'V') IS NOT NULL

DROP VIEW SavingAccountsToUpdate

GO

CREATE VIEW SavingAccountsToUpdate AS(

SELECT \*

FROM(

SELECT SAD.\*, A.CurrentBalance,

CASE

WHEN InterestRate = 'half year' THEN IIF(MONTH(A.StartDate) % 6 IN (MONTH(GETDATE()), 0), 0, 1)

WHEN InterestRate = 'quarter' THEN IIF(MONTH(A.StartDate) % 3 IN (MONTH(GETDATE()), 0), 0, 1)

WHEN InterestRate = 'yearly' THEN IIF(MONTH(A.StartDate) = MONTH(GETDATE()), 0, 1)

ELSE 0

END 'mod'

FROM SavingAccountDetails SAD

JOIN Accounts A ON A.AccountID = SAD.AccountID

WHERE A.EndDate IS NULL

) SUB

WHERE mod = 0

)

GO

IF OBJECT\_ID('StandingOrdersToSend', 'V') IS NOT NULL

DROP VIEW StandingOrdersToSend

GO

CREATE VIEW StandingOrdersToSend AS(

SELECT \*

FROM StandingOrders

WHERE StartDate <= CAST(GETDATE() AS Date) AND CAST(GETDATE() AS Date) <= EndDate

AND DAY(StartDate) = DAY(GETDATE())

)

GO

IF OBJECT\_ID('CardDetails', 'V') IS NOT NULL

DROP VIEW CardDetails

GO

CREATE VIEW CardDetails AS(

SELECT DISTINCT Card,

COUNT(Amount) OVER(PARTITION BY Card) 'Operations',

SUM(Amount) OVER(PARTITION BY Card) 'Value'

FROM(

SELECT Card, Amount, [Date]

FROM Withdraws

UNION ALL

SELECT Card, Amount, [Date]

FROM Deposits

UNION ALL

SELECT UsedCard, Amount, [Date]

FROM Transactions

) CardOperations

)

GO

IF OBJECT\_ID('AllOperations', 'V') IS NOT NULL

DROP VIEW AllOperations

GO

CREATE VIEW AllOperations AS(

SELECT AccountID, Date, -Amount 'Amount', 'Withdraw' 'Operation'

FROM Withdraws W

JOIN Cards C ON C.CardID = W.Card

JOIN Accounts A ON A.AccountID = C.Account

UNION ALL

SELECT AccountID, Date, Amount, 'Deposit' 'Operation'

FROM Deposits D

JOIN Cards C ON C.CardID = D.Card

JOIN Accounts A ON A.AccountID = C.Account

UNION ALL

SELECT AccountID, Date, -Amount 'Amount', 'Made transaction' 'Operation'

FROM Transactions T

JOIN Cards C ON C.CardID = T.UsedCard

JOIN Accounts A ON A.AccountID = C.Account

UNION ALL

SELECT AccountID, Date, Amount, 'Recived transaction' 'Operation'

FROM Transactions T

JOIN Accounts A ON A.AccountID = T.Receiver

UNION ALL

SELECT AccountID, Date, -Amount 'Amount', 'Made transfer' 'Operation'

FROM Transfers T

JOIN Accounts A ON A.AccountID = T.Sender

UNION ALL

SELECT AccountID, Date, Amount, 'Recived transfer' 'Operation'

FROM Transfers T

JOIN Accounts A ON A.AccountID = T.Receiver

UNION ALL

SELECT AccountID, Date, -Amount 'Amount', 'Made phone transfer' 'Operation'

FROM PhoneTransfers PT

JOIN Accounts A ON A.AccountID = PT.Sender

UNION ALL

SELECT MainAccount, Date, Amount, 'Recived phone transfer' 'Operation'

FROM PhoneTransfers PT

JOIN Clients C ON C.PhoneNumber = PT.PhoneReceiver

JOIN Preferences P ON P.ClientID = C.ClientID

)

GO

IF OBJECT\_ID('NumberOfOperationsByAccount', 'V') IS NOT NULL

DROP VIEW NumberOfOperationsByAccount

GO

CREATE VIEW NumberOfOperationsByAccount AS

SELECT Account, COUNT(\*) 'Operations'

FROM(

SELECT Amount, [Date], Sender AS [Account]

FROM Transfers

UNION ALL

SELECT Amount, [Date], Receiver AS [Account]

FROM Transfers

UNION ALL

SELECT Amount, [Date], (SELECT Account FROM Cards WHERE CardID = T.UsedCard) AS [Account]

FROM Transactions T

UNION ALL

SELECT Amount, [Date], Receiver AS [Account]

FROM Transactions T

UNION ALL

SELECT Amount, [Date], Sender AS [Account]

FROM PhoneTransfers PT

UNION ALL

SELECT Amount, [Date], (SELECT MainAccount FROM Preferences P

JOIN Clients C ON C.ClientID = P.ClientID

WHERE C.PhoneNumber = PT.PhoneReceiver) AS [Account]

FROM PhoneTransfers PT

) AccountOperations

WHERE Account IS NOT NULL

GROUP BY Account

GO

IF OBJECT\_ID('NumberOfOperationsByClient', 'V') IS NOT NULL

DROP VIEW NumberOfOperationsByClient

GO

CREATE VIEW NumberOfOperationsByClient AS

SELECT A.ClientID, SUM(N.Operations) AS 'Operations'

FROM NumberOfOperationsByAccount N

JOIN Accounts A ON A.AccountID = N.Account

GROUP BY A.ClientID

GO

IF OBJECT\_ID('NumberOfOperationsByAccountsAndCategories', 'V') IS NOT NULL

DROP VIEW NumberOfOperationsByAccountsAndCategories

GO

CREATE VIEW NumberOfOperationsByAccountsAndCategories AS

SELECT Account, Category, COUNT(\*) 'Operations'

FROM(

SELECT Category, Amount, [Date], Sender AS [Account]

FROM Transfers

UNION ALL

SELECT Category, Amount, [Date], Receiver AS [Account]

FROM Transfers

UNION ALL

SELECT Category, Amount, [Date], (SELECT Account FROM Cards WHERE CardID = T.UsedCard) AS [Account]

FROM Transactions T

UNION ALL

SELECT Category, Amount, [Date], Receiver AS [Account]

FROM Transactions T

UNION ALL

SELECT Category, Amount, [Date], Sender AS [Account]

FROM PhoneTransfers PT

UNION ALL

SELECT Category, Amount, [Date], (SELECT MainAccount FROM Preferences P

JOIN Clients C ON C.ClientID = P.ClientID

WHERE C.PhoneNumber = PT.PhoneReceiver) AS [Account]

FROM PhoneTransfers PT

) AccountOperations

WHERE Account IS NOT NULL

GROUP BY Category, Account

GO

IF OBJECT\_ID('NumberOfTransfersByClient', 'V') IS NOT NULL

DROP VIEW NumberOfTransfersByClient

GO

CREATE VIEW NumberOfTransfersByClient AS

SELECT A.ClientID, COUNT(T.[Account]) AS 'Operations'

FROM (SELECT Sender AS [Account]

FROM Transfers

UNION ALL

SELECT Receiver AS [Account]

FROM Transfers) T

JOIN Accounts A ON A.AccountID = T.Account

GROUP BY A.ClientID

GO

IF OBJECT\_ID('NumberOfPhoneTransfersByClient', 'V') IS NOT NULL

DROP VIEW NumberOfPhoneTransfersByClient

GO

CREATE VIEW NumberOfPhoneTransfersByClient AS

SELECT ClientID, COUNT(PT.ClientID) AS 'Operations'

FROM (SELECT ClientID

FROM PhoneTransfers tmpPT

JOIN Accounts A ON A.AccountID = tmpPT.Sender

UNION ALL

SELECT ClientID

FROM PhoneTransfers tmpPT

JOIN Clients C ON C.PhoneNumber = tmpPT.PhoneReceiver

) PT

GROUP BY ClientID

GO

IF OBJECT\_ID('NumberOfOperationsByCard', 'V') IS NOT NULL

DROP VIEW NumberOfOperationsByCard

GO

CREATE VIEW NumberOfOperationsByCard AS(

SELECT Card,

COUNT(\*) 'Operations'

FROM(

SELECT Card, Amount, [Date]

FROM Withdraws

UNION ALL

SELECT Card, Amount, [Date]

FROM Withdraws

UNION ALL

SELECT UsedCard, Amount, [Date]

FROM Transactions

) CardOperations

GROUP BY Card

)

GO

### --Dodanie procedur

DROP PROCEDURE addNewClient

GO

CREATE PROCEDURE addNewClient

@name NVARCHAR(100),

@dateOfBirth DATE,

@city NVARCHAR(100),

@country NVARCHAR(100),

@phoneNumber NVARCHAR(100),

@allowPhoneTransfers BIT

AS

BEGIN

IF @dateOfBirth > GETDATE()

RAISERROR('Date of Birth can not be in the future', 17, 1)

ELSE IF @dateOfBirth > CAST(DATEADD(YEAR, -16, GETDATE()) AS DATE)

RAISERROR('Client must be older than the age of 16 ', 17, 1)

ELSE

BEGIN

INSERT INTO Clients VALUES

(@name, @dateOfBirth, @city, @country, @phoneNumber);

INSERT INTO Preferences VALUES

(@@IDENTITY , NULL, @allowPhoneTransfers)

END

END

GO

DROP PROCEDURE IF EXISTS addNewAccount

GO

CREATE PROCEDURE addNewAccount

@accountID NVARCHAR(100),

@clientID INT,

@name NVARCHAR(100),

@accountType INT,

@password NVARCHAR(100),

@interestRate FLOAT = 1.0,

@frequency NVARCHAR(100) = 'monthly'

AS

BEGIN

IF LEN(@password) < 20

RAISERROR('Password not strong enough',17,1);

ELSE

BEGIN

INSERT INTO Accounts VALUES

(@accountID, @clientID, @name, @accountType, 0, CAST(GETDATE() AS Date), NULL, @password)

IF (SELECT MainAccount FROM Preferences WHERE ClientID = @clientID) IS NULL

BEGIN

UPDATE Preferences

SET MainAccount = @accountID

WHERE ClientID = @clientID

END

IF @accountType = 3

BEGIN

INSERT INTO SavingAccountDetails VALUES

(@accountID, @interestRate, @frequency)

END

END

END

GO

DROP PROCEDURE IF EXISTS addNewCard

GO

CREATE PROCEDURE addNewCard

@cardID NVARCHAR(100),

@accountID NVARCHAR(100),

@limit INT,

@pin NVARCHAR(100)

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @accountID) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF LEN(@pin) <> 4

RAISERROR('Incorrect pin',17,1);

ELSE IF @limit < 0

RAISERROR('Incorrect limit',17,1);

ELSE

INSERT INTO Cards VALUES

(@cardID, @accountID, @limit, @pin);

END

GO

DROP PROCEDURE IF EXISTS addNewTransfer

GO

CREATE PROCEDURE addNewTransfer

@sender NVARCHAR(100),

@receiver NVARCHAR(100),

@amount MONEY,

@title NVARCHAR(100),

@category INT

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @sender)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @sender) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF @amount <= 0

RAISERROR('Incorrect amount',17,1)

ELSE IF ( SELECT CurrentBalance

FROM Accounts

WHERE AccountID = @sender ) < @amount

RAISERROR('Not enough funds',17,1)

ELSE IF @sender = @receiver

RAISERROR('Incorrect operation',17,1)

ELSE

INSERT INTO Transfers VALUES

(@sender, @receiver, @amount, @title, CAST(GETDATE() AS Date), @category, NULL)

END

GO

DROP PROCEDURE IF EXISTS addNewPhoneTransfer

GO

CREATE PROCEDURE addNewPhoneTransfer

@sender NVARCHAR(100),

@phoneReceiver NVARCHAR(100),

@amount MONEY,

@title NVARCHAR(100),

@category INT

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @sender)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @sender) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF @amount <= 0

RAISERROR('Incorrect amount',17,1)

ELSE IF ( SELECT CurrentBalance

FROM Accounts

WHERE AccountID = @sender ) < @amount

RAISERROR('Not enough funds',17,1)

ELSE IF ( SELECT AllowPhoneTransfer

FROM Preferences P

JOIN Clients C ON C.ClientID = P.ClientID

WHERE C.PhoneNumber = @phoneReceiver ) = 0

RAISERROR('Receiver does not accept phoneTransfers',17,1)

ELSE IF ( SELECT MainAccount

FROM Preferences P

JOIN Clients C ON C.ClientID = P.ClientID

WHERE C.PhoneNumber = @phoneReceiver ) = @sender

RAISERROR('Incorrect operation',17,1)

ELSE

INSERT INTO PhoneTransfers VALUES

(@sender, @phoneReceiver, @amount, @title, CAST(GETDATE() AS Date), @category)

END

GO

DROP PROCEDURE IF EXISTS addNewTransaction

GO

CREATE PROCEDURE addNewTransaction

@cardID NVARCHAR(100),

@receiver NVARCHAR(100),

@amount MONEY,

@title NVARCHAR(100),

@category INT

AS

BEGIN

IF @amount <= 0

RAISERROR('Incorrect amount', 17 ,1)

ELSE IF (SELECT EndDate FROM Accounts JOIN Cards ON Account = AccountID WHERE CardID = @cardID) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF @amount > (

SELECT CurrentBalance

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

WHERE C.CardID = @cardID)

RAISERROR('Not enough funds', 17, 1)

ELSE IF @amount > (SELECT Limit FROM Cards WHERE CardID = @cardID)

RAISERROR('Amount greater then card limit',17,1)

ELSE IF @receiver = (SELECT Account FROM Cards WHERE CardID = @cardID)

RAISERROR('Incorrect operation', 17, 1)

ELSE

INSERT INTO Transactions VALUES

(@cardID, @receiver, @amount, GETDATE(), @category)

END

GO

DROP PROCEDURE IF EXISTS addNewWithdraw

GO

CREATE PROCEDURE addNewWithdraw

@cardID NVARCHAR(100),

@amount MONEY,

@ATM INT

AS

BEGIN

IF @amount <= 0

RAISERROR('Incorrect amount',17,1)

ELSE IF (SELECT EndDate FROM Accounts JOIN Cards ON Account = AccountID WHERE CardID = @cardID) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF NOT EXISTS(SELECT \* FROM ATMs WHERE ATMID = @ATM)

RAISERROR('ATM does not exist',17,1)

ELSE IF @amount > (SELECT CurrentBalance FROM ATMs WHERE ATMID = @ATM)

RAISERROR('ATM does not have enough funds',17,1)

ELSE IF @amount > (

SELECT CurrentBalance

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

WHERE C.CardID = @cardID)

RAISERROR('Not enough funds',17,1)

ELSE

INSERT INTO Withdraws VALUES

(@cardID, @amount, @ATM, GETDATE())

END

GO

DROP PROCEDURE IF EXISTS addNewDeposit

GO

CREATE PROCEDURE addNewDeposit

@cardID NVARCHAR(100),

@amount MONEY,

@ATM INT

AS

BEGIN

IF @amount <= 0

RAISERROR('Incorrect amount',17,1)

ELSE IF (SELECT EndDate FROM Accounts JOIN Cards ON Account = AccountID WHERE CardID = @cardID) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF NOT EXISTS(SELECT \* FROM ATMs WHERE ATMID = @ATM)

RAISERROR('ATM does not exist',17,1)

ELSE

INSERT INTO Deposits VALUES

(@cardID, @amount, @ATM, GETDATE())

END

GO

DROP PROCEDURE IF EXISTS addStandingOrders

GO

CREATE PROCEDURE addStandingOrders

@sender NVARCHAR(100),

@receiver NVARCHAR(100),

@amount MONEY,

@title NVARCHAR(100),

@frequency INT,

@startDate DATE,

@endDate DATE

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @sender)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @sender) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF @sender = @receiver OR @endDate = @startDate

RAISERROR('Incorrect operation',17,1)

ELSE IF @amount <= 0

RAISERROR('Incorrect amount',17,1)

ELSE IF @frequency <= 0 OR @frequency > (DATEDIFF(day, @startDate, @endDate))

RAISERROR('Incorrect frequency',17,1)

ELSE IF @startDate < GETDATE()

RAISERROR('Start date can not be in the past', 17, 1)

ELSE IF @endDate < GETDATE()

RAISERROR('End date can not be in the past', 17, 1)

ELSE

INSERT INTO StandingOrders VALUES

(@sender, @receiver, @amount, @title, @frequency, @startDate, @endDate)

END

GO

DROP PROCEDURE IF EXISTS addNewLoan

GO

CREATE PROCEDURE addNewLoan

@accountID NVARCHAR(100),

@amount MONEY,

@endDate DATE,

@servingEmployee INT

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @accountID) IS NOT NULL

RAISERROR('Account has been closed', 17 ,1)

ELSE IF (SELECT AccountType FROM Accounts WHERE AccountID = @accountID) = 2

RAISERROR('Incorrect account type', 17 ,1)

ELSE IF @amount <= 0

RAISERROR('Incorrect amount', 17 ,1)

ELSE IF NOT EXISTS(SELECT \* FROM Employees WHERE EmployeeID = @servingEmployee)

RAISERROR('Employee does not exist',17,1)

ELSE IF @endDate < GETDATE()

RAISERROR('Date can not be in the past', 17, 1)

ELSE

INSERT INTO Loans VALUES

(@accountID, @amount, GETDATE(), @endDate, @servingEmployee)

END

GO

DROP PROCEDURE IF EXISTS addNewBranches

GO

CREATE PROCEDURE addNewBranches

@name NVARCHAR(100),

@city NVARCHAR(100),

@country NVARCHAR(100)

AS

BEGIN

IF LEN(@name) < 2 OR LEN(@city) < 2 OR LEN(@city) < 2

RAISERROR('To short parameters', 17, 1)

ELSE

INSERT INTO Branches VALUES

(@name, @city, @country)

END

GO

DROP PROCEDURE IF EXISTS addNewEmployee

GO

CREATE PROCEDURE addNewEmployee

@name NVARCHAR(100),

@dateOfSign DATE,

@BranchID INT

AS

BEGIN

IF LEN(@name) < 2

RAISERROR('To short name', 17, 1)

ELSE IF @dateOfSign > GETDATE()

RAISERROR('Date can not be in the future', 17, 1)

ELSE

INSERT INTO Employees VALUES

(@name, @dateOfSign, @BranchID)

END

GO

DROP PROCEDURE IF EXISTS addNewATM

GO

CREATE PROCEDURE addNewATM

@currentBalance INT,

@supervisorDepartment INT,

@city NVARCHAR(100)

AS

BEGIN

IF LEN(@city) < 2

RAISERROR('To short city name', 17, 1)

ELSE IF @currentBalance < 0

RAISERROR('Incorrect current balance', 17, 1)

ELSE

INSERT INTO ATMs VALUES

(@currentBalance, @supervisorDepartment, @city)

END

GO

DROP PROCEDURE IF EXISTS addNewAccountType

GO

CREATE PROCEDURE addNewAccountType

@description NVARCHAR(100)

AS

BEGIN

IF LEN(@description) >= 2

INSERT INTO AccountTypes VALUES

(@description)

ELSE

RAISERROR('To short description', 17, 1)

END

GO

DROP PROCEDURE IF EXISTS addNewTransactionCategory

GO

CREATE PROCEDURE addNewTransactionCategory

@description NVARCHAR(100)

AS

BEGIN

IF LEN(@description) >= 2

INSERT INTO TransactionCategories VALUES

(@description)

ELSE

RAISERROR('To short description', 17, 1)

END

GO

DROP PROCEDURE IF EXISTS disactiveAccount

GO

CREATE PROCEDURE disactiveAccount

@accountID NVARCHAR(100),

@password NVARCHAR(100)

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Account does not exist',17,1);

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @accountID) IS NOT NULL

RAISERROR('Account is disactived',17,1);

ELSE IF @password <> (SELECT Password FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Password is not correct',17,1);

ELSE

BEGIN

DECLARE @clientID INT

SET @clientID = (SELECT A.ClientID FROM Accounts A WHERE A.AccountID = @accountID)

IF ( SELECT MainAccount FROM Preferences WHERE ClientID = @clientID) = @accountID

BEGIN

IF (SELECT COUNT(\*) FROM Accounts WHERE ClientID = @clientID and AccountID <> @accountID and EndDate IS NULL) > 0

UPDATE Preferences

SET MainAccount = ( SELECT TOP 1 AccountID FROM Accounts

WHERE ClientID = @clientID and AccountID <> @accountID and EndDate IS NULL)

WHERE ClientID = @clientID

ELSE

UPDATE Preferences

SET MainAccount = NULL, AllowPhoneTransfer = 0

WHERE ClientID = @clientID

END

UPDATE Accounts

SET EndDate = CAST(GETDATE() AS Date)

WHERE AccountID = @accountID

END

END

GO

DROP PROCEDURE IF EXISTS changeAccountPassword

GO

CREATE PROCEDURE changeAccountPassword

@accountID NVARCHAR(100),

@prev\_password NVARCHAR(100),

@new\_password NVARCHAR(100)

AS

BEGIN

IF NOT EXISTS (SELECT AccountID FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Account does not exist',17,1)

ELSE IF (SELECT EndDate FROM Accounts WHERE AccountID = @accountID) IS NOT NULL

RAISERROR('Account is disactived',17,1)

ELSE IF @prev\_password <> (SELECT Password FROM Accounts WHERE AccountID = @accountID)

RAISERROR('Previous password is not correct',17,1)

ELSE

BEGIN

UPDATE Accounts

SET Password = @new\_password

WHERE AccountID = @accountID

END

END

GO

DROP PROCEDURE IF EXISTS changeCardLimit

GO

CREATE PROCEDURE changeCardLimit

@cardID NVARCHAR(100),

@limit INT,

@pin NVARCHAR(100)

AS

BEGIN

IF @pin <> (SELECT PIN FROM Cards WHERE CardID = @cardID)

RAISERROR('PIN is not correct',17,1)

ELSE IF @limit <= 0

RAISERROR('Incorrect limit',17,1)

ELSE

UPDATE Cards

SET Limit = @limit

WHERE CardID = @cardID

END

GO

DROP PROCEDURE IF EXISTS changeCardPIN

GO

CREATE PROCEDURE changeCardPIN

@cardID NVARCHAR(100),

@prevPIN NVARCHAR(100),

@newPIN NVARCHAR(100)

AS

BEGIN

IF @prevPIN <> (SELECT PIN FROM Cards WHERE CardID = @cardID)

RAISERROR('Previous PIN is not correct',17,1)

ELSE

UPDATE Cards

SET PIN = @newPIN

WHERE CardID = @cardID

END

GO

DROP PROCEDURE IF EXISTS reportATMsMalfunction

GO

CREATE PROCEDURE reportATMsMalfunction

@ATMID INT,

@description NVARCHAR(100),

@reportingEmployee INT

AS

BEGIN

IF NOT EXISTS(SELECT \* FROM ATMs WHERE ATMID = @ATMID)

RAISERROR('ATM does not exist',17,1)

ELSE IF NOT EXISTS(SELECT \* FROM Employees WHERE EmployeeID = @reportingEmployee)

RAISERROR('Employee does not exist',17,1)

ELSE

INSERT INTO ATMsMalfunctions VALUES

(@ATMID, @description, GETDATE(), @reportingEmployee)

END

GO

### --Dodanie funkcji

IF OBJECT\_ID('ClientOperationsByMonth', 'IF') IS NOT NULL

DROP FUNCTION ClientOperationsByMonth

GO

CREATE FUNCTION ClientOperationsByMonth(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT MONTH([Date]) 'Month', YEAR([Date]) 'Year',

COUNT(\*) 'Operations'

FROM Accounts A

JOIN AllOperations AO ON AO.AccountID = A.AccountID

WHERE ClientID = @clientID

GROUP BY MONTH([Date]), YEAR([Date])

)

GO

IF OBJECT\_ID('ClientOperationsByCard', 'IF') IS NOT NULL

DROP FUNCTION ClientOperationsByCard

GO

CREATE FUNCTION ClientOperationsByCard(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT A.ClientID, C.CardID, N.Operations

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

JOIN NumberOfOperationsByCard N ON N.Card = C.CardID

WHERE A.ClientID = @clientID

)

GO

IF OBJECT\_ID('ClientTransfersNumber', 'IF') IS NOT NULL

DROP FUNCTION ClientTransfersNumber

GO

CREATE FUNCTION ClientTransfersNumber(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT ClientID, N.Operations

FROM NumberOfTransfersByClient N

WHERE ClientID = @clientID

)

GO

IF OBJECT\_ID('ClientPhoneTransfersNumber', 'IF') IS NOT NULL

DROP FUNCTION ClientPhoneTransfersNumber

GO

CREATE FUNCTION ClientPhoneTransfersNumber(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT ClientID, N.Operations

FROM NumberOfPhoneTransfersByClient N

WHERE ClientID = @clientID

)

GO

IF OBJECT\_ID('ClientOperationsByOperationType', 'IF') IS NOT NULL

DROP FUNCTION ClientOperationsByOperationType

GO

CREATE FUNCTION ClientOperationsByOperationType(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT 'Transfers: ' AS 'Type', Operations

FROM ClientTransfersNumber(@clientID)

UNION ALL

SELECT 'Card Operations: ' AS 'Type', SUM(Operations)

FROM ClientOperationsByCard(@clientID)

UNION ALL

SELECT 'Transactions: ' AS 'Type', Operations

FROM ClientPhoneTransfersNumber(@clientID)

)

GO

IF OBJECT\_ID('OnOwnAccountsTransfers', 'IF') IS NOT NULL

DROP FUNCTION OnOwnAccountsTransfers

GO

CREATE FUNCTION OnOwnAccountsTransfers(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT \*

FROM Transfers

WHERE Sender IN (SELECT AccountID FROM Accounts WHERE ClientID = @clientID)

AND Receiver IN (SELECT AccountID FROM Accounts WHERE ClientID = @clientID)

)

GO

IF OBJECT\_ID('NumberOfOperationsByCategories', 'IF') IS NOT NULL

DROP FUNCTION NumberOfOperationsByCategories

GO

CREATE FUNCTION NumberOfOperationsByCategories()

RETURNS TABLE

AS

RETURN(

SELECT A.ClientID, N.Category, SUM(N.Operations) AS 'Operations'

FROM NumberOfOperationsByAccountsAndCategories N

JOIN Accounts A ON A.AccountID = N.Account

GROUP BY A.ClientID, N.Category

)

GO

IF OBJECT\_ID('ClientOperationsByCategories', 'IF') IS NOT NULL

DROP FUNCTION ClientOperationsByCategories

GO

CREATE FUNCTION ClientOperationsByCategories(@clientID INT)

RETURNS TABLE

AS

RETURN(

SELECT N.Category, N.Operations

FROM NumberOfOperationsByCategories() N

WHERE ClientID = @clientID

)

GO

IF OBJECT\_ID('AccountHistory', 'IF') IS NOT NULL

DROP FUNCTION AccountHistory

GO

CREATE FUNCTION AccountHistory(@account NVARCHAR(100))

RETURNS TABLE

AS

RETURN(

SELECT ROW\_NUMBER() OVER(ORDER BY Date) 'Id',\*

FROM AllOperations

WHERE AccountID = @account

)

GO

IF OBJECT\_ID('AccountOperationsByMonth', 'IF') IS NOT NULL

DROP FUNCTION AccountOperationsByMonth

GO

CREATE FUNCTION AccountOperationsByMonth(@account NVARCHAR(100))

RETURNS TABLE

AS

RETURN(

SELECT MONTH([Date]) 'Month',

YEAR([Date]) 'Year',

COUNT(\*) 'Operations'

FROM AccountHistory(@account)

GROUP BY MONTH([Date]), YEAR([Date])

)

GO

IF OBJECT\_ID('GetPassword', 'FN') IS NOT NULL

DROP FUNCTION GetPassword

GO

CREATE FUNCTION GetPassword(@account NVARCHAR(100))

RETURNS NVARCHAR(100)

AS

BEGIN

RETURN(

SELECT [Password]

FROM Accounts

WHERE AccountID = @account

)

END

GO

IF OBJECT\_ID('GetPIN', 'FN') IS NOT NULL

DROP FUNCTION GetPIN

GO

CREATE FUNCTION GetPIN(@cardID NVARCHAR(100))

RETURNS NVARCHAR(100)

AS

BEGIN

RETURN(

SELECT PIN

FROM Cards

WHERE CardID = @cardID

)

END

GO

IF OBJECT\_ID('IfAccountExists', 'FN') IS NOT NULL

DROP FUNCTION IfAccountExists

GO

CREATE FUNCTION IfAccountExists(@account NVARCHAR(100))

RETURNS BIT

AS

BEGIN

RETURN(

SELECT IIF(@account IN (

SELECT AccountID

FROM Accounts), 1, 0)

)

END

GO

IF OBJECT\_ID('DepartmentATMsBalance', 'IF') IS NOT NULL

DROP FUNCTION DepartmentATMsBalance

GO

CREATE FUNCTION DepartmentATMsBalance(@departmentID INT)

RETURNS TABLE

AS

RETURN(

SELECT SUM(CurrentBalance) as balancesSUM

FROM ATMs

WHERE SupervisorDepartment = @departmentID

GROUP BY SupervisorDepartment

)

GO

IF OBJECT\_ID('ATMOperationsByMonth', 'IF') IS NOT NULL

DROP FUNCTION ATMOperationsByMonth

GO

CREATE FUNCTION ATMOperationsByMonth(@atm INT)

RETURNS TABLE

AS

RETURN(

SELECT MONTH([Date]) 'Month', YEAR([Date]) 'Year',

COUNT(\*) 'Operations'

FROM(

SELECT \*

FROM Withdraws

UNION ALL

SELECT \*

FROM Deposits

) Operations

WHERE ATMID = @atm

GROUP BY MONTH([Date]), YEAR([Date])

)

GO

IF OBJECT\_ID('ATM\_MalfunctionsHistory', 'IF') IS NOT NULL

DROP FUNCTION ATM\_MalfunctionsHistory

GO

CREATE FUNCTION ATM\_MalfunctionsHistory(@atmID INT)

RETURNS TABLE

AS

RETURN (

SELECT \*

FROM ATMsMalfunctions

WHERE ATMID = @atmID

)

GO

### --Dodatnie wyzwalaczy

CREATE TRIGGER newWithdraw

ON dbo.Withdraws

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance - i.Amount

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

JOIN inserted i ON i.Card = C.CardID

UPDATE ATMs

SET CurrentBalance = A.CurrentBalance - i.Amount

FROM ATMs A

JOIN inserted i ON i.ATMID = A.ATMID

END

GO

CREATE TRIGGER newDeposit

ON dbo.Deposits

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

JOIN inserted i ON i.Card = C.CardID

UPDATE ATMs

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM ATMs A

JOIN inserted i ON i.ATMID = A.ATMID

END

GO

CREATE TRIGGER newTransaction

ON dbo.Transactions

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance - i.Amount

FROM Accounts A

JOIN Cards C ON C.Account = A.AccountID

JOIN inserted i ON i.UsedCard = C.CardID

IF (SELECT TOP 1 Receiver FROM inserted ORDER BY TransactionID) IN (SELECT AccountID FROM Accounts)

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM Accounts A

JOIN inserted i ON i.Receiver = A.AccountID

END

END

GO

CREATE TRIGGER newTransfer

ON dbo.Transfers

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance - i.Amount

FROM Accounts A

JOIN inserted i ON i.Sender = A.AccountID

IF (SELECT TOP 1 Receiver FROM inserted ORDER BY TransferID) IN (SELECT AccountID FROM Accounts)

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM Accounts A

JOIN inserted i ON i.Receiver = A.AccountID

END

END

GO

CREATE TRIGGER newPhoneTransfer

ON dbo.PhoneTransfers

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance - i.Amount

FROM Accounts A

JOIN inserted i ON i.Sender = A.AccountID

IF (SELECT TOP 1 PhoneReceiver FROM inserted ORDER BY TransferID) IN (SELECT PhoneNumber FROM Clients)

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM Accounts A

JOIN Preferences P ON P.MainAccount = A.AccountID

JOIN Clients C ON C.ClientID = P.ClientID

JOIN inserted i ON i.PhoneReceiver = C.PhoneNumber

END

END

GO

CREATE TRIGGER newLoan

ON dbo.Loans

AFTER INSERT

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance + i.Amount

FROM Accounts A

JOIN inserted i ON i.AccountID = A.AccountID

WHERE i.EndDate > GETDATE()

END

GO

### --Dodanie Job Scheduler

USE DBproject

DROP PROCEDURE IF EXISTS createBackup

GO

CREATE PROCEDURE createBackup

AS

BEGIN

DECLARE @path NVARCHAR(100) = 'C:\Users\Konrad\Desktop\BAZA BACKUP\'

DECLARE @filename NVARCHAR(100) = 'backup' + CONVERT(NVARCHAR, GETDATE(), 5) + '.bak'

DECLARE @final NVARCHAR(100) = @path + @filename

BACKUP DATABASE DBproject

TO DISK = @final

END

GO

DROP PROCEDURE IF EXISTS checkLoans

GO

CREATE PROCEDURE checkLoans

AS

BEGIN

UPDATE Accounts

SET CurrentBalance = A.CurrentBalance - L.Amount

FROM Accounts A

JOIN Loans L ON L.AccountID = A.AccountID

WHERE L.EndDate = GETDATE()

END

GO

DROP PROCEDURE IF EXISTS checkSavingAccounts

GO

CREATE PROCEDURE checkSavingAccounts

AS

BEGIN

--DECLARE @month INT = MONTH(GETDATE())

DECLARE @rowCount INT = (SELECT COUNT(\*) FROM SavingAccountsToUpdate)

DECLARE @temp TABLE(ID INT IDENTITY, Account NVARCHAR(100), Frequency FLOAT, CurrentBalance MONEY)

DECLARE @id INT = 1

INSERT INTO @temp (Account, Frequency, CurrentBalance)

SELECT AccountID, Frequency, CurrentBalance FROM SavingAccountsToUpdate

DECLARE @account NVARCHAR(100)

DECLARE @frequency FLOAT

DECLARE @balance MONEY

WHILE @rowCount > 0

BEGIN

SELECT @id = ID,

@account = Account,

@frequency = Frequency,

@balance = CurrentBalance

FROM @temp

ORDER BY ID DESC OFFSET @rowCount - 1 ROWS FETCH NEXT 1 ROWS ONLY

DECLARE @amount MONEY = CAST(@balance \* @frequency AS MONEY)

INSERT INTO Transfers VALUES

('BANK',@account,@amount,'Saving Account Income',GETDATE(),1,NULL)

SET @rowCount = @rowCount - 1

END

END

GO

DROP PROCEDURE IF EXISTS checkStandingOrders

GO

CREATE PROCEDURE checkStandingOrders

AS

BEGIN

DECLARE @rowCount INT = (SELECT COUNT(\*) FROM StandingOrdersToSend)

DECLARE @temp TABLE(ID INT IDENTITY, Sender NVARCHAR(100), Receiver NVARCHAR(100), Amount MONEY, Title NVARCHAR(100))

DECLARE @id INT = 1

INSERT INTO @temp (Sender, Receiver, Amount, Title)

SELECT SOS.Sender, SOS.Receiver, SOS.Amount, SOS.Title FROM StandingOrdersToSend SOS

DECLARE @tmp\_sender NVARCHAR(100)

DECLARE @tmp\_receiver NVARCHAR(100)

DECLARE @tmp\_amount MONEY

DECLARE @tmp\_title NVARCHAR(100)

WHILE @rowCount > 0

BEGIN

SELECT @id = ID,

@tmp\_sender = Sender,

@tmp\_receiver = Receiver,

@tmp\_amount = Amount,

@tmp\_title = Title

FROM @temp

ORDER BY ID DESC OFFSET @rowCount - 1 ROWS FETCH NEXT 1 ROWS ONLY

EXEC [dbo].[addNewTransfer] @sender = @tmp\_sender, @receiver = @tmp\_receiver, @amount = @tmp\_amount, @title = @tmp\_title, @category=11

SET @rowCount = @rowCount - 1

END

END

GO

USE msdb

GO

-- DELETING

IF EXISTS(SELECT \* FROM dbo.sysjobs WHERE name = 'checkSavingAcoounts')

EXEC sp\_delete\_job @job\_name = 'checkSavingAcoounts'

GO

IF EXISTS(SELECT \* FROM dbo.sysjobs WHERE name = 'checkLoans')

EXEC sp\_delete\_job @job\_name = 'checkLoans'

GO

IF EXISTS(SELECT \* FROM dbo.sysjobs WHERE name = 'checkStandingOrders')

EXEC sp\_delete\_job @job\_name = 'checkStandingOrders'

GO

IF EXISTS(SELECT \* FROM dbo.sysjobs WHERE name = 'monthlyBackup')

EXEC sp\_delete\_job @job\_name = 'monthlyBackup'

GO

IF EXISTS(SELECT \* FROM dbo.sysschedules WHERE name = 'Daily')

EXEC sp\_delete\_schedule @schedule\_name = 'Daily'

GO

IF EXISTS(SELECT \* FROM dbo.sysschedules WHERE name = 'Monthly')

EXEC sp\_delete\_schedule @schedule\_name = 'Monthly'

GO

-- CREATING SCHEDULES

EXEC sp\_add\_schedule

@schedule\_name = N'Daily',

@freq\_type = 4, --daily

@freq\_interval = 1,

@active\_start\_time = 000000 ; --every midnight

GO

EXEC sp\_add\_schedule

@schedule\_name = N'Monthly',

@freq\_type = 16, --monthly

@freq\_interval = 1, --on the 1st day of the month

@freq\_recurrence\_factor = 1,

@active\_start\_time = 000000 ; --every midnight

GO

-- LOANS CHECK

EXEC sp\_add\_job

@job\_name = 'checkLoans'

GO

EXEC sp\_add\_jobstep

@job\_name = N'checkLoans',

@step\_name = N'CheckLoans',

@subsystem = N'TSQL',

@command = N'EXEC checkLoans',

@retry\_attempts = 5,

@retry\_interval = 5;

GO

EXEC sp\_attach\_schedule

@job\_name = N'checkLoans',

@schedule\_name = N'Daily'

GO

-- SAVING ACCOUNTS CHECK

EXEC sp\_add\_job

@job\_name = N'checkSavingAcoounts'

GO

EXEC sp\_add\_jobstep

@job\_name = N'checkSavingAcoounts',

@step\_name = N'CheckSavingAcoounts',

@subsystem = N'TSQL',

@command = N'EXEC checkSavingAccounts',

@retry\_attempts = 5,

@retry\_interval = 5;

GO

EXEC sp\_attach\_schedule

@job\_name = N'checkSavingAcoounts',

@schedule\_name = N'Monthly'

GO

-- STANDING ORDERS CHECK

EXEC sp\_add\_job

@job\_name = N'checkStandingOrders'

GO

EXEC sp\_add\_jobstep

@job\_name = N'checkStandingOrders',

@step\_name = N'CheckStandingOrders',

@subsystem = N'TSQL',

@command = N'EXEC checkStandingOrders',

@retry\_attempts = 5,

@retry\_interval = 5;

GO

GO

EXEC sp\_attach\_schedule

@job\_name = N'checkStandingOrders',

@schedule\_name = N'Daily'

GO

-- DATABASE BACKUP

EXEC sp\_add\_job

@job\_name = N'monthlyBackup'

GO

EXEC sp\_add\_jobstep

@job\_name = N'monthlyBackup',

@step\_name = N'CreateMonthlyBackupOfDB',

@subsystem = N'TSQL',

@command = N'EXEC createBackup',

@retry\_attempts = 5,

@retry\_interval = 5;

GO

GO

EXEC sp\_attach\_schedule

@job\_name = N'monthlyBackup',

@schedule\_name = N'Monthly'

GO