

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет» РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-1 «Защита информации»

ОТЧЁТ

# по практическому занятию №8

**по дисциплине «Безопасность систем баз данных»**

Студент: Крутов А.М.

Шифр учебной группы:

БББО-05-20

Принял: Кунин Н.Т.

**Задание 1.**

* Создаем роли пользователей:

CREATE ROLE HighLevel;

CREATE ROLE LeadershipLevel;

CREATE ROLE InformationLevel;

CREATE ROLE TechologicalLevel;

CREATE ROLE ServiceLevel;

CREATE ROLE PresentationLevel;

CREATE ROLE IntersectoralLevel;

* Выдаем права созданным ролям:

GRANT SELECT, UPDATE, DELETE ON Магистраль TO HighLevel;

GRANT SELECT, UPDATE, DELETE ON Кабель TO HighLevel;

GRANT SELECT, UPDATE, DELETE ON Тип\_кабеля TO HighLevel;

GRANT SELECT, UPDATE, DELETE ON Пара TO HighLevel;

GRANT SELECT, UPDATE, DELETE ON Магистраль(Тип\_магистрали, Число\_кабелей)

TO LeadershipLevel;

GRANT SELECT, UPDATE, DELETE ON Кабель(Цепь\_ДП, Тип\_кабеля, #Магистрали)

TO LeadershipLevel;

GRANT SELECT, UPDATE, DELETE ON Тип\_кабеля(Число\_пар, Защита)

TO LeadershipLevel;

GRANT SELECT, UPDATE, DELETE ON Пара(#Кабеля, ПРД/ПРМ)

TO LeadershipLevel;

GRANT SELECT, UPDATE, DELETE ON Магистраль(Тип\_магистрали)

TO InformationLevel;

GRANT SELECT, UPDATE, DELETE ON Кабель(Цепь\_ДП, Тип\_кабеля, #Магистрали, #Кабеля)

TO InformationLevel;

GRANT SELECT, UPDATE, DELETE ON Тип\_кабеля(Число\_пар, Защита) TO InformationLevel;

GRANT SELECT, UPDATE, DELETE ON Пара(ПРД/ПРМ, #Кабеля, #Пары) TO InformationLevel;

GRANT SELECT, UPDATE, DELETE ON Магистраль(Тип\_магистрали, Число\_кабелей)

TO TechologicalLevel;

GRANT SELECT, UPDATE ON Кабель(#Магистрали, #Кабеля) TO TechologicalLevel;

GRANT SELECT ON Тип\_кабеля TO TechologicalLevel;

GRANT SELECT, UPDATE, DELETE ON Пара(ПРД/ПРМ, #Кабеля) TO TechologicalLevel;

GRANT SELECT, UPDATE, DELETE ON Магистраль(Тип\_магистрали, Число\_кабелей)

TO ServiceLevel;

GRANT SELECT, UPDATE, DELETE ON Кабель(Цепь\_ДП, Тип\_кабеля, #Магистрали, #Кабеля)

TO ServiceLevel;

GRANT SELECT, UPDATE, DELETE ON Тип\_кабеля TO ServiceLevel;

GRANT SELECT, UPDATE, DELETE ON Пара(ПРД/ПРМ, #Кабеля, #Пары) TO ServiceLevel;

GRANT SELECT ON Магистраль TO PresentationLevel;

GRANT SELECT ON Кабель TO PresentationLevel;

GRANT SELECT ON Тип\_кабеля TO PresentationLevel;

GRANT SELECT ON Пара TO PresentationLevel;

GRANT SELECT ON Магистраль TO IntersectoralLevel;

GRANT SELECT ON Кабель TO IntersectoralLevel;

GRANT SELECT ON Тип\_кабеля TO IntersectoralLevel;

GRANT SELECT ON Пара TO IntersectoralLevel;

**Задание 2.**

* Создание функции add\_task\_data:

CREATE FUNCTION add\_task\_data(@task\_name NVARCHAR(50), @start\_date DATE,

@end\_date DATE, @description NVARCHAR(100),

@responsible\_employee\_id INT, @connected\_employee\_ids NVARCHAR(MAX))

RETURNS INT

AS

BEGIN

DECLARE @result INT = 0

IF EXISTS (SELECT 1 FROM task WHERE task\_name = @task\_name AND start\_date

= @start\_date)

SET @result = 1

ELSE IF NOT EXISTS (SELECT 1 FROM employee WHERE employee\_id = @responsible\_employee\_id)

SET @result = 2

ELSE IF TRY\_PARSE(CONCAT(@start\_date, ' 00:00:00') AS DATETIME) IS NULL

OR TRY\_PARSE(CONCAT(@end\_date, ' 00:00:00') AS DATETIME) IS NULL

SET @result = 2

ELSE BEGIN

BEGIN TRANSACTION

BEGIN TRY

INSERT INTO task (task\_name, start\_date, end\_date, description,

responsible\_employee\_id)

VALUES (@task\_name, @start\_date, @end\_date, @description, @responsible\_employee\_id)

DECLARE @task\_id INT = SCOPE\_IDENTITY()

DECLARE @connected\_employee\_id INT

DECLARE @delimiter CHAR(1) = ','

DECLARE @start\_position INT = 1

DECLARE @end\_position INT

WHILE CHARINDEX(@delimiter, @connected\_employee\_ids, @start\_position) > 0

BEGIN

SET @end\_position = CHARINDEX(@delimiter, @connected\_employee\_ids, @start\_position)

SET @connected\_employee\_id = SUBSTRING(@connected\_employee\_ids, @start\_position, @end\_position - @start\_position)

INSERT INTO task\_employee (task\_id, employee\_id) VALUES

(@task\_id, @connected\_employee\_id)

SET @start\_position = @end\_position + 1

END

SET @connected\_employee\_id = SUBSTRING(@connected\_employee\_ids,

@start\_position, LEN(@connected\_employee\_ids) - @start\_position + 1)

INSERT INTO task\_employee (task\_id, employee\_id) VALUES

(@task\_id, @connected\_employee\_id)

COMMIT TRANSACTION

END TRY

BEGIN CATCH

SET @result = 3

ROLLBACK TRANSACTION

END CATCH

END

RETURN @result

END

GO

* Создание архивных таблиц:

CREATE TABLE task\_archive

(

task\_id INT PRIMARY KEY,

task\_name NVARCHAR(50) NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

description NVARCHAR(100),

responsible\_employee\_id INT NOT NULL,

CONSTRAINT FK\_responsible\_employee\_id FOREIGN KEY (responsible\_employee\_id) REFERENCES employee(employee\_id)

)

GO

CREATE TABLE task\_employee\_archive

(

task\_id INT NOT NULL,

employee\_id INT NOT NULL,

CONSTRAINT PK\_task\_employee\_archive PRIMARY KEY (task\_id, employee\_id),

CONSTRAINT FK\_task\_id FOREIGN KEY (task\_id) REFERENCES task\_archive(task\_id),

CONSTRAINT FK\_employee\_id FOREIGN KEY (employee\_id) REFERENCES employee(employee\_id)

)

GO

* Создание функции перемещения данных в архивные таблицы:

CREATE FUNCTION archive\_tasks(@archive\_date DATE)

RETURNS INT

AS

BEGIN

DECLARE @result INT = 1

BEGIN TRANSACTION

BEGIN TRY

DECLARE @tasks\_to\_archive TABLE (task\_id INT)

INSERT INTO @tasks\_to\_archive (task\_id)

SELECT task\_id FROM task WHERE end\_date <= @archive\_date

IF @@ROWCOUNT > 0

BEGIN

SET @result = 0

INSERT INTO task\_archive (task\_id, task\_name, start\_date,

end\_date, description, responsible\_employee\_id)

SELECT task\_id, task\_name, start\_date, end\_date, description, responsible\_employee\_id

FROM task WHERE task\_id IN (SELECT task\_id FROM @tasks\_to\_achive)

INSERT INTO task\_employee\_archive (task\_id, employee\_id)

SELECT task\_id, employee\_id FROM task\_employee WHERE task\_id IN

(SELECT task\_id FROM @tasks\_to\_archive)

DELETE FROM task\_employee WHERE task\_id IN (SELECT task\_id FROM

@tasks\_to\_archive)

DELETE FROM task WHERE task\_id IN (SELECT task\_id FROM

@tasks\_to\_archive)

END

COMMIT TRANSACTION

END TRY

BEGIN CATCH

SET @result = 2

ROLLBACK TRANSACTION

END CATCH

RETURN @result

END

GO

* Создание процедуры расчета ежегодной премии:

CREATE PROCEDURE calculate\_annual\_bonus(@hierarchy\_level INT, @num\_tasks INT,

@bonus\_percent INT, @success BIT OUT, @bonus\_table TABLE (employee\_id INT,

bonus\_amount MONEY))

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

INSERT INTO @bonus\_table (employee\_id, bonus\_amount)

SELECT employee.employee\_id, (employee.salary \* @bonus\_percent /

100.0)

FROM employee

INNER JOIN (

SELECT task\_employee.employee\_id, COUNT(DISTINCT task\_employee.task\_id) AS num\_responsible\_tasks

FROM task\_employee

INNER JOIN task ON task.task\_id = task\_employee.task\_id

WHERE task\_employee.employee\_id = task.responsible\_employee\_id

AND task\_employee.employee\_hierarchy\_level = @hierarchy\_level AND

YEAR(task.start\_date) = YEAR(GETDATE())

GROUP BY task\_employee.employee\_id

HAVING COUNT(DISTINCT task\_employee.task\_id) >= @num\_tasks

) AS responsible\_tasks ON responsible\_tasks.employee\_id = employee.employee\_id

SET @success = 1

END TRY

BEGIN CATCH

SET @success = 0

END CATCH

END

GO

CREATE PACKAGE task\_package AS

FUNCTION add\_task\_data(@task\_name NVARCHAR(50), @start\_date DATE,

@end\_date DATE, @description NVARCHAR(100),

@responsible\_employee\_id INT, @connected\_employee\_ids NVARCHAR(MAX)) RETURNS INT

FUNCTION archive\_tasks(@archive\_date DATE) RETURNS INT

PROCEDURE calculate\_annual\_bonus(@hierarchy\_level INT, @num\_tasks INT,

@bonus\_percent INT, @success BIT OUT, @bonus\_table TABLE (employee\_id INT,

bonus\_amount MONEY))

END

GO

**Задание 3.**

* Создание функции add\_flight\_data:

CREATE FUNCTION add\_flight\_data (@departure\_point varchar(50), @destination\_point varchar(50), @departure\_time datetime, @arrival\_time

datetime, @aircraft\_code varchar(10), @captain\_code int, @crew\_codes

varchar(100))

RETURNS int

AS

BEGIN

DECLARE @flight\_id int

SELECT @flight\_id = flight\_id FROM flight WHERE departure\_point =

@departure\_point AND destination\_point = @destination\_point AND departure\_time = @departure\_time

IF @flight\_id IS NULL

BEGIN

INSERT INTO flight (departure\_point, destination\_point, departure\_time, arrival\_time, aircraft\_code) VALUES (@departure\_point,

@destination\_point, @departure\_time, @arrival\_time, @aircraft\_code)

SET @flight\_id = @@IDENTITY

END

ELSE

BEGIN

RETURN 1

END

DECLARE @error int

SET @error = 0

INSERT INTO crew\_flight (crew\_id, flight\_id) VALUES (@captain\_code, @flight\_id)

SET @error = @@ERROR

IF LEN(@crew\_codes) > 0

BEGIN

DECLARE @crew\_table TABLE (code int)

DECLARE @pos int

SET @pos = CHARINDEX(',', @crew\_codes)

WHILE @pos > 0

BEGIN

INSERT INTO @crew\_table (code) VALUES (CAST(SUBSTRING(@crew\_codes, 1, @pos - 1) AS int))

SET @crew\_codes = SUBSTRING(@crew\_codes, @pos + 1,

LEN(@crew\_codes) - @pos)

SET @pos = CHARINDEX(',', @crew\_codes)

END

INSERT INTO @crew\_table (code) VALUES (CAST(@crew\_codes AS

int))

INSERT INTO crew\_flight (crew\_id, flight\_id)

SELECT code, @flight\_id FROM @crew\_table

SET @error = @@ERROR

END

IF @error <> 0

BEGIN

RETURN 2

END

RETURN 0

END

* Создание функции move\_records\_to\_archive:

CREATE FUNCTION move\_records\_to\_archive (@archive\_date datetime)

RETURNS int

AS

BEGIN

DECLARE @error int

SET @error = 0

BEGIN TRAN

INSERT INTO flight\_archive (flight\_id, departure\_point, destination\_point, departure\_time, arrival\_time, aircraft\_code)

SELECT flight\_id, departure\_point, destination\_point, departure\_time, arrival\_time, aircraft\_code FROM flight WHERE arrival\_time < @archive\_date

SET @error = @@ERROR

IF @error <> 0

BEGIN

ROLLBACK TRAN

RETURN 2

END

DELETE FROM flight WHERE arrival\_time < @archive\_date

SET @error = @@ERROR

IF @error <> 0

BEGIN

ROLLBACK TRAN

RETURN 2

END

INSERT INTO crew\_flight\_archive (crew\_id, flight\_id)

SELECT crew\_id, flight\_id FROM crew\_flight WHERE flight\_id IN (SELECT

flight\_id FROM flight\_archive)

SET @error = @@ERROR

IF @error <> 0

BEGIN

ROLLBACK TRAN

RETURN 2

END

DELETE FROM crew\_flight WHERE flight\_id IN (SELECT flight\_id FROM

flight\_archive)

SET @error = @@ERROR

IF @error <> 0

BEGIN

ROLLBACK TRAN

RETURN 2

END

COMMIT TRAN

IF @@ROWCOUNT > 0

BEGIN

RETURN 0

END

ELSE

BEGIN

RETURN 1

END

END

* Создание процедуры calculate\_bonus:

CREATE PROCEDURE calculate\_bonus (@hierarchy\_level int, @num\_of\_flights int,

@bonus\_percent int, @result int OUTPUT)

AS

BEGIN

SELECT employee.id, employee.salary \* @bonus\_percent / 100 AS bonus

INTO #employee\_bonus

FROM employee

INNER JOIN crew ON employee.id = crew.employee\_id

INNER JOIN crew\_flight ON crew.id = crew\_flight.crew\_id

INNER JOIN flight ON crew\_flight.flight\_id = flight.flight\_id

WHERE employee.hierarchy\_level = @hierarchy\_level AND YEAR(flight.departure\_time) = YEAR(GETDATE()) AND COUNT(\*) >= @num\_of\_flights

GROUP BY employee.id, employee.salary

IF @@ROWCOUNT > 0

BEGIN

SET @result = 0

END

ELSE

BEGIN

SET @result = 1

END

SELECT \* FROM #employee\_bonus

DROP TABLE #employee\_bonus

END

**Задание 4.**

* Создание функции add\_game\_data:

CREATE FUNCTION add\_game\_data

(

@championship VARCHAR(50),

@location VARCHAR(100),

@game\_date DATETIME,

@opponent\_team\_code INT,

@captain\_code INT,

@player\_codes\_str VARCHAR(MAX)

)

RETURNS INT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @result INT = 0;

IF EXISTS (SELECT 1 FROM games WHERE location = @location AND game\_date =

@game\_date)

BEGIN

SET @result = 1;

END

ELSE

BEGIN

-- Check if captain exists

IF NOT EXISTS (SELECT 1 FROM players WHERE player\_code = @captain\_code)

BEGIN

SET @result = 2;

END

ELSE

BEGIN

DECLARE @player\_codes TABLE (player\_code INT);

DECLARE @delimiter CHAR(1) = ',';

DECLARE @start\_pos INT = 1;

DECLARE @end\_pos INT;

DECLARE @player\_code\_str VARCHAR(20);

WHILE CHARINDEX(@delimiter, @player\_codes\_str, @start\_pos) > 0

BEGIN

SET @end\_pos = CHARINDEX(@delimiter, @player\_codes\_str, @start\_pos);

SET @player\_code\_str = SUBSTRING(@player\_codes\_str, @start\_pos,

@end\_pos - @start\_pos);

SET @start\_pos = @end\_pos + 1;

IF NOT EXISTS (SELECT 1 FROM players WHERE player\_code =

@player\_code\_str)

BEGIN

SET @result = 2;

BREAK;

END

ELSE

BEGIN

INSERT INTO @player\_codes (player\_code) VALUES

(@player\_code\_str);

END

END

SET @player\_code\_str = SUBSTRING(@player\_codes\_str, @start\_pos,

LEN(@player\_codes\_str) - @start\_pos + 1);

IF NOT EXISTS (SELECT 1 FROM players WHERE player\_code =

@player\_code\_str)

BEGIN

SET @result = 2;

END

ELSE

BEGIN

INSERT INTO @player\_codes (player\_code) VALUES (@player\_code\_str);

INSERT INTO games (championship, location, game\_date, opponent\_team\_code)

VALUES (@championship, @location, @game\_date, @opponent\_team\_code);

DECLARE @game\_id INT = SCOPE\_IDENTITY();

INSERT INTO game\_players (game\_id, player\_code, is\_captain)

VALUES (@game\_id, @captain\_code, 1);

INSERT INTO game\_players (game\_id, player\_code, is\_captain)

SELECT @game\_id, player\_code, 0 FROM @player\_codes;

SET @result = 0;

END

END

END

RETURN @result;

END

* Создание архивных таблиц:

CREATE TABLE archive\_games (

game\_id INT PRIMARY KEY,

game\_name VARCHAR(50),

game\_date DATE,

game\_result VARCHAR(10)

);

CREATE TABLE archive\_game\_players (

game\_id INT,

player\_id INT,

score INT,

PRIMARY KEY (game\_id, player\_id)

);

* Создание функции MoveRecordsToArchiveTables:

CREATE FUNCTION MoveRecordsToArchiveTables (@date DATETIME)

RETURNS INT

AS

BEGIN

DECLARE @result INT = 1; -- По умолчанию не было перемещено ни одной записи

IF EXISTS (SELECT \* FROM игра WHERE дата < @date)

BEGIN

INSERT INTO архив\_игры (id, дата, игровое\_поле)

SELECT id, дата, игровое\_поле

FROM игра

WHERE дата < @date;

INSERT INTO архив\_игра\_игрок (id\_игры, id\_игрока, результат)

SELECT id\_игры, id\_игрока, результат

FROM игра\_игрок

WHERE id\_игры IN (SELECT id FROM игра WHERE дата < @date);

DELETE FROM игра WHERE дата < @date;

DELETE FROM игра\_игрок WHERE id\_игры IN (SELECT id FROM игра WHERE дата < @date);

SET @result = 0;

END

ELSE

BEGIN

SET @result = 1;

END

RETURN @result;

END

* Создание процедуры calculate\_annual\_bonus:

CREATE PROCEDURE calculate\_annual\_bonus

@percent\_of\_games float,

@bonus\_percent float,

@success bit OUTPUT,

@result\_table TABLE (player\_code int, bonus\_amount float) OUTPUT

AS

BEGIN

SET NOCOUNT ON;

IF @percent\_of\_games <= 0 OR @bonus\_percent <= 0

BEGIN

SET @success = 0;

RETURN;

END

DECLARE @total\_games int;

SELECT @total\_games = COUNT(\*) FROM club\_games;

IF @total\_games = 0

BEGIN

SET @success = 0;

RETURN;

END

DECLARE @eligible\_players TABLE (player\_code int);

INSERT INTO @eligible\_players

SELECT player\_code

FROM player\_games

GROUP BY player\_code

HAVING COUNT(\*) >= @total\_games \* @percent\_of\_games;

INSERT INTO @result\_table (player\_code, bonus\_amount)

SELECT player\_code, monthly\_salary \* @bonus\_percent

FROM players

WHERE player\_code IN (SELECT player\_code FROM @eligible\_players);

SET @success = 1;

END

**Задание 5.**

* Создание функции заведения сделки:

CREATE FUNCTION [dbo].[AddTransaction] (

@date\_conclusion DATE,

@date\_valuation DATE,

@date\_completion DATE,

@operation\_type VARCHAR(50),

@bank\_code VARCHAR(50),

@transaction\_code VARCHAR(50),

@prolongation\_code VARCHAR(50),

@account\_codes VARCHAR(255)

)

RETURNS INT

AS

BEGIN

DECLARE @bank\_id INT

DECLARE @transaction\_id INT

DECLARE @prolongation\_id INT

DECLARE @account\_count INT

DECLARE @account\_type VARCHAR(50)

DECLARE @account\_codes\_table TABLE (code VARCHAR(50))

SELECT @bank\_id = id

FROM banks

WHERE code = @bank\_code

IF @bank\_id IS NULL

BEGIN

RETURN 1

END

IF @prolongation\_code IS NOT NULL

BEGIN

SELECT @prolongation\_id = id

FROM transactions

WHERE code = @prolongation\_code

IF @prolongation\_id IS NULL

BEGIN

RETURN 1

END

END

INSERT INTO @account\_codes\_table

SELECT value FROM string\_split(@account\_codes, ';')

SELECT @account\_count = COUNT(\*)

FROM @account\_codes\_table

IF @account\_count = 0

BEGIN

RETURN 1

END

DECLARE @accounts\_table TABLE (id INT)

INSERT INTO @accounts\_table

SELECT id

FROM accounts

WHERE code IN (SELECT code FROM @account\_codes\_table)

AND type = @operation\_type

SELECT @account\_count = COUNT(\*)

FROM @accounts\_table

IF @account\_count <> @account\_codes\_table

BEGIN

RETURN 1

END

IF NOT EXISTS (

SELECT \*

FROM agreements

WHERE bank\_id = @bank\_id

AND start\_date <= @date\_conclusion

AND end\_date >= @date\_completion

)

BEGIN

RETURN 1

END

INSERT INTO transactions (code, date\_conclusion, date\_valuation, date\_completion, operation\_type, prolongation\_id)

VALUES (@transaction\_code, @date\_conclusion, @date\_valuation, @date\_completion,

@operation\_type, @prolongation\_id)

SELECT @transaction\_id = SCOPE\_IDENTITY()

INSERT INTO transaction\_accounts (transaction\_id, account\_id)

SELECT @transaction\_id, id

FROM @accounts\_table

RETURN 0

END

* Создание функции перемещения записей в архивные таблицы:

CREATE FUNCTION moveTransactionsToArchive (@archiveDate DATE)

RETURNS INT

AS

BEGIN

DECLARE @result INT;

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO Archive\_Transaction (transaction\_id, transaction\_date, transaction\_amount)

SELECT transaction\_id, transaction\_date, transaction\_amount

FROM Transaction

WHERE transaction\_status = 'completed' AND transaction\_date < @archiveDate;

INSERT INTO Archive\_Transaction\_Account (transaction\_id, account\_id, transaction\_amount)

SELECT transaction\_id, account\_id, transaction\_amount

FROM Transaction\_Account

WHERE transaction\_status = 'completed' AND transaction\_date < @archiveDate;

DELETE FROM Transaction

WHERE transaction\_status = 'completed' AND transaction\_date < @archiveDate;

DELETE FROM Transaction\_Account

WHERE transaction\_status = 'completed' AND transaction\_date < @archiveDate;

SET @result = 0

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SET @result = 2;

END CATCH

IF NOT EXISTS (SELECT \* FROM Archive\_Transaction) AND NOT EXISTS (SELECT \* FROM

Archive\_Transaction\_Account)

BEGIN

SET @result = 1;

END

RETURN @result;

END

* Создание процедуры поиска крупнейших кредиторов и дебиторов:

CREATE PROCEDURE dbo.find\_largest\_creditors\_and\_debtors

@percent\_credit\_given INT,

@percent\_credit\_taken INT,

@success BIT OUTPUT,

@largest\_credit\_given TABLE (bank\_code INT, credit\_amount DECIMAL(18,2)) OUTPUT,

@largest\_credit\_taken TABLE (bank\_code INT, credit\_amount DECIMAL(18,2)) OUTPUT

AS

BEGIN

SET NOCOUNT ON

DECLARE @avg\_credit\_given DECIMAL(18,2)

DECLARE @avg\_credit\_taken DECIMAL(18,2)

SELECT @avg\_credit\_given = AVG(credit\_amount) FROM loans WHERE YEAR(loan\_date) =

YEAR(GETDATE())

SELECT @avg\_credit\_taken = AVG(credit\_amount) FROM loans WHERE YEAR(loan\_date) =

YEAR(GETDATE())

INSERT INTO @largest\_credit\_given

SELECT bank\_code, credit\_amount FROM loans WHERE YEAR(loan\_date) = YEAR(GETDATE()) AND credit\_amount > (@avg\_credit\_given \* (1 + (@percent\_credit\_given /

100)))

INSERT INTO @largest\_credit\_taken

SELECT bank\_code, credit\_amount FROM loans WHERE YEAR(loan\_date) = YEAR(GETDATE()) AND credit\_amount > (@avg\_credit\_taken \* (1 + (@percent\_credit\_taken /

100)))

IF ((SELECT COUNT(\*) FROM @largest\_credit\_given) = 0 OR (SELECT COUNT(\*) FROM

@largest\_credit\_taken) = 0)

SET @success = 0

ELSE

SET @success = 1

RETURN

END