UNIVERSIDADE FEDERAL DE GOIÁS - UFG

Alexandre Oliveira dos Santos

Relatório Técnico: Programação para Banco de Dados

Goiânia, Dezembro de 2017

Alexandre Oliveira dos Santos

Relatório Técnico: Programação para Banco de Dados

Relatório técnico apresentado como requisito parcial do trabalho 2 da disciplina Banco de Dados 2, no Curso de Sistemas de Informação, na Universidade Federal de Goiás.

Prof. Dr. Leonardo Andrade Ribeiro

Goiânia, Dezembro de 2017

RESUMO

Este projeto foi desenvolvido no escopo da disciplina de Banco de Dados II do curso de Sistemas de Informação da Universidade Federal de Goiás, no mês de Dezembro de 2017, com o intuito de realizar programação em um SGBD envolvendo o desenvolvimento de Triggers, funções e/ou procedimentos armazenados, visões e esquema de autorização utilizando um SGBD de código fechado (Microsoft SQL Server 2014).

As solicitações propostas foram desenvolvidos através da utilização de software client (GUI), Microsoft SQL Server - SQL Server Management Studio 17 (SSMS).

A descrição de todo o estudo realizado e resultados e conclusões obtidos são elucidados e compartilhados ao longo deste documento, como informações sobre o esquema do banco de dados, descrições de visões e políticas de atualizações, procedimentos armazenados e esquema de autorização. O campo Apêndice é composto pelas expressões utilizadas (códigos) para a criação das gatilhos (triggers), visões (views) e procedimentos armazenados (procedures).

Palavras-chave: GUI, SGBD, SQL Server, SSMS, Triggers, Views, Procedures

1. INTRODUÇÃO

A grande demanda por informação na conhecida Era digital ilustra-nos de maneira indiscutível a grande importância do termo dado para pessoas, empresas e organizações, independentemente de seus diversos fins ou portes. Para gerenciar essa demanda é necessário a utilização de meios que proporcionem e assegurem diversos princípios, como atomicidade, consistência, independência e também que seja mantida a durabilidade desses dados no meio em que este é armazenado.

Quando utilizado de forma correta, a maioria dos sistemas gerenciadores de banco de dados (SGBDs) oferecem inúmeras funcionalidades que podem melhorar performance e ainda sustentar e proporcionar técnicas para maior garantia de princípios como os descritos acima, como a utilização de visões para a criação e uso de tabelas virtuais pré definidas, de procedimentos armazenados e funções para a criação e execução de comandos SQL, de gatilhos ativados quando eventos ocorrem sobre tabelas e visões, do conceito de autorização para controle e segurança através de permissões, entre outros.

Este relatório visa detalhar o presente trabalho acadêmico, especificando o método de desenvolvimento e resultados obtidos durante a criação e execução da programação proposta no SGBD utilizado: Microsoft SQL Server 2014 Express.

2. BREVE DESCRIÇÃO DOS MECANISMOS DE TRIGGERS E FUNÇÕES/PROCEDIMENTOS ARMAZENADOS IMPLEMENTADOS NO SGBD:

2.1. Mecanismos de Triggers:

Como mecanismo de gatilho (trigger), foi implementado para cada visão (view) três gatilhos, para as operações de inserção (insert), atualização (update) e remoção (delete). As triggers implementadas são do tipo Instead of pois as alterações podem afetar múltiplas tabelas e são disparadas automaticamente quando um evento DML (linguagem de manipulação de dados) como Insert, Update ou Delete ocorre sobre a tabela e no caso deste trabalho a visão (view) para o qual este gatilho foi especificado.

OBSERVAÇÃO: Maiores informações sobre as triggers implementadas neste trabalho podem ser encontradas no item 4 deste documento e ainda no campo Apêndice.

2.2. Funções/Procedimentos armazenados implementados no SGBD.

Neste trabalho foram implementados três mecanismos de procedimento armazenado (procedures), duas destas contendo testes condicionais. A primeira procedure chamada spPisoMinMax, possibilita através do cpf do funcionário verificar o valor do salário via a visão vwFunc_Dep e tomar decisões caso o salário esteja abaixo do piso mínimo ou acima do teto máximo. A segunda procedure chamada spGetFuncByGender é apta a recuperar todos os funcionário de algum sexo específico através da visão vwFunc_Dep. Por fim, a terceira procedure chamada spAgeMinEmployee está apta a, através do cpf do funcionário e a view vwFunc_Dep verificar se esse funcionário, de acordo com a data atual é maior ou menor de idade, e a partir dessa informação tomar a decisão de mantê-lo ou remover seu armazenamento no registro de funcionários.

OBSERVAÇÃO: Maiores informações sobre as visões acima citada e as procedures implementadas neste trabalho podem ser encontradas no item 4.3 deste documento e ainda no campo Apêndice.

3. DESCRIÇÃO RESUMIDA DO ESQUEMA DE BANCO DE DADOS

O esquema disponibilizado no ambiente Moodle da disciplina é chamado companhia. O mesmo possui em seu corpo 6 tabelas (TABLES): funcionario, departamento, dept_locais, projeto, func_proj e dependente. Há na criação das tabelas a definição de chaves primárias, constraints chaves estrangeiras e também referências.
Observação: Código de criação do esquema disponível para visualização no link: *https://goo.gl/TKrTa9*

4. DESCRIÇÃO DOS COMPONENTES IMPLEMENTADOS

4.1. Descrição das Visões (Views)

OBSERVAÇÃO: Todos os códigos para a criação das visões (views) aqui citadas estão disponíveis no campo apêndice deste documento

Conforme a proposta, neste trabalho foi criado 3 diferentes visões. Como convenção de criação todas as views iniciaram seu nome com o padrão ‘vw’. As tabelas atingidas pela view será descrita abaixo com a abreviação ‘TBL’.

A primeira visão foi chamada vwFunc_Dep, e se caracteriza por recuperar os campos nome, inicial, sobrenome, cpf, data_nasc, endereco, gen, salario, super_cpf de todos os funcionários (TBL Funcionario) e o nome do departamento, dnome para qual este funcionário trabalha (TBL Departamento). O atributo utilizado para criação da view foi o campo número de departamento presente nas duas tabelas anteriormente citadas.

A segunda visão foi chamada vwSuper_Dep, e se caracteriza por recuperar os campos dnome, dnumero, ger_cpf e ger_inicio_data (TBL departamento) e o nome de seu supervisor nomeado de ger_nome, buscado na (TBL funcionario). O atributo utilizado para a criação da view foi o campo cpf do funcionário presente nas duas tabelas anteriormente citadas.

A terceira visão foi chamada vwFunc_Depend, e se caracteriza por recuperar os campos nome, gen, data_nasc, relacionamento, cpf (TBL dependente) e nome e sobrenome na (TBL funcionario). O atributo utilizado para a criação da view foi o campo cpf do funcionário presente nas duas tabelas anteriormente citadas..

4.2 Política de atualização das Visões (View): TRIGGERS

OBSERVAÇÃO: Todos os códigos para a criação dos gatilhos (triggers) aqui citados estão disponíveis no campo apêndice deste documento.

Como política de atualização das Visões (Views) foram criados 3 gatilhos (Triggers) para cada uma das visões, totalizando 9 triggers, com a finalidade de possibilitar a correta modificação através de operações de inserção (insert), atualização (update) ou remoção (delete). Como convenção de criação todas os gatilhos (triggers) iniciaram seu nome com o padrão 'tr_'.
Abaixo é elucidado com maior detalhamento cada uma das visões e o que foi pensado durante a criação do gatilho (trigger) correspondente:

Operação Insert: View 1 (vwFunc_Dep):
Esse gatilho foi chamado tr_vwFunc_Dep_InsteadOfInsert, e se caracteriza por permitir a inserção de um novo funcionario via view na (TBL funcionario) e associá-lo a um departamento existente através da inserção também do nome deste departamento. Caso o departamento informado não exista na relação da (TBL departamento) é lançada uma mensagem de erro ao usuário informando que o departamento correspondente não foi localizado.

Durante os testes de inserção foram feitas as seguintes operações (SQL):

```
INSERT INTO vwFunc_Dep VALUES('Alexandre', NULL, 'Oliveira', '043842981', '1996-06-16', '759 Centro, Goiânia, GO', 'M', '7000', NULL, 'Software');
INSERT INTO vwFunc_Dep VALUES('Luciana', NULL, 'Rezende', '022666789', '1990-08-22', '112 Centro, Goiânia, GO', 'F', '5000', '043842981', 'Software');
INSERT INTO vwFunc_Dep VALUES('Carla', NULL, 'Morais', '011011011', '1992-08-13', '116 Setor Bueno, Goiânia, GO', 'F', '6000', '043842981', 'Softwarre');
```

Nas duas primeiras inserções acima, o processo de inserção na visão ocorre com sucesso. Na última porém, o campo do nome de departamento está propositalmente incorreto e inexiste na relação de departamento e portanto não ocorre a inserção.

Operação Update: View 1 (vwFunc_Dep):
Esse gatilho foi chamado tr_vwFunc_Dep_InsteadOfUpdate e se caracteriza por permitir a atualização de dados de um funcionário (TBL funcionario) e seu departamento correspondente através de informado o nome deste departamento (TBL departamento). Foi realizado alguns tratamentos para permitir a correta atualização dos campos e manter a consistência do banco. O campo cpf do funcionário não pode ser

atualizado através dessa operação/visão uma vez que ele é utilizado para ligação das duas tabelas. Caso a atualização seja no campo nome do funcionário, inicial, sobrenome, data_nasc, endereço do funcionario, gen, salario a modificação é realizada sem maiores problemas. Contudo, caso a atualização seja nos campos super_cpf ou dnome essa operação necessita de mais atenção.

No caso do campo super_cpf foi necessário consultar a (TBL departamento) e verificar se o novo campo super_cpf corresponde a um supervisor de departamento válido, ou seja, se seu cpf se encontra no campo ger_cpf de algum departamento. Caso negativo, é lançada uma mensagem de erro ao usuário informado que o cpf informado no campo supervisor não corresponde a um gerente de departamento válido. Caso contrário é feito a atualização

No caso do campo nome de departamento (dnome), é verificado se o nome de departamento a ser atualizado na view corresponde a um departamento válido na (TBL departamento) através da comparação do nome de departamento inserido e seus IDs. Caso o departamento seja encontrado é realizada a modificação do nome do departamento na visão e do dnumero na (TBL funcionario). Caso o nome do departamento não seja encontrado é lançada uma mensagem de erro ao usuário informando que o nome do novo departamento informado não foi encontrado.

Durante os testes de atualização foram feitas as seguintes operações (SQL):

```
UPDATE vwFunc_Dep set data_nasc =  '1995-01-11', dnome =  'Research' where cpf = '043842981';
UPDATE vwFunc_Dep set super_cpf = '000000000' where cpf = '043842981';
```

Na primeira operação acima, o processo de atualização na visão ocorre com sucesso. Na última porém, o campo do cpf do supervisor está propositalmente incorreto, inexiste na relação de departamento e portanto não ocorre a inserção.

Operação Delete: View 1 (vwFunc_Dep):

Esse gatilho foi chamado tr_vwFunc_Dep_InsteadOfDelete e se caracteriza por permitir a remoção de um funcionário (TBL funcionario) a partir da informação de seu cpf.

Durante os testes de remoção foram feitas as seguintes operações (SQL):

```
DELETE from vwFunc_Dep where cpf = '022666789';
DELETE from vwFunc_Dep where cpf = '000000000';
DELETE from vwFunc_Dep where cpf = '043842981';
```

Na primeira operação acima, o processo de remoção ocorre com sucesso e o funcionário é removido. Já na segunda o processo finaliza sem nenhuma linha ser afetada uma vez que o cpf inexiste na relação funcionário. Na última operação contudo

a operação não foi concluída pois como o cpf do funcionário informado possui referência/funciona como chave estrangeira em outra tabela acredito que o SGBD impediu a operação para manter consistência/integridade.

Operação Insert: View 2 (vwSuper_Dep):
Esse gatilho foi chamado tr_vwSuper_Dep_InsteadOfInsert, e se caracteriza por permitir a inserção de um novo supervisor a um departamento (TBL departamento). Caso o cpf do funcionário não exista na relação de funcionários é exibida uma mensagem ao usuário. Caso o cpf exista ele é atribuído ao campo ger_cpf e é inserido os demais dados como dnome, dnumero e ger_inicio_data.

Durante os testes de inserção foram feitas as seguintes operações (SQL):
```
INSERT INTO vwSuper_Dep VALUES('Project', 9, NULL,  '043842981', '2017-12-14');
INSERT INTO vwSuper_Dep VALUES('Finances', 10, NULL, '444444444', '2017-12-16');
INSERT INTO vwSuper_Dep VALUES('Finances', 10, NULL, 222222202, '2017-12-16');
```

Nas primeira e terceira inserção acima, o processo de inserção do novo departamento na visão ocorre com sucesso. Na segunda porém, o campo ger_cpf está propositalmente incorreto,inexiste na relação de funcionário e portanto não ocorre a inserção.

Operação Update: View 2 (vwSuper_Dep):
Esse gatilho foi chamado tr_vwSuper_Dep_InsteadOfUpdate e se caracteriza por permitir a atualização de dados de um departamento (TBL departamento) através da view de acordo com o número de departamento informado. É possível alterar o nome do departamento, o nome do gerente localizado na (TBL funcionario) através do seu cpf, a data de início de gerencia. É ainda possível modificar o campo ger_cpf que indica o gerente do departamento. Caso o cpf informado não exista na relação de funcionário é exibida uma mensagem de erro ao usuário. Caso exista o mesmo é atribuído ao campo gerente do departamento. Por fim, caso o usuário deseje atualizar o número do departamento é exibida uma mensagem de erro ao usuário impedindo a operação uma vez que esse é o campo utilizado para conexão das duas tabelas responsáveis pela view.

Durante os testes de atualização foram feitas as seguintes operações (SQL):
```
UPDATE vwSuper_Dep set ger_cpf = '222222202' WHERE dnumero = 9; --Altera o supervisor do departamento
UPDATE vwSuper_Dep set ger_nome = 'Alexandre' WHERE dnumero = 9; --Altera nome na TBL funcionario
UPDATE vwSuper_Dep set ger_cpf = '000000000' WHERE dnumero = 9;
```

Na primeira atualização acima o campo ger_cpf foi alterado na (TBL departamento) com sucesso. A segunda atualização altera o nome do funcionário (TBL funcionário) com sucesso. A terceira operação de atualização contudo não ocorre e é exibida uma mensagem de erro uma vez que o novo ger_cpf de departamento inexiste no campo cpf da (TBL funcionário).

Operação Delete: View 2 (vwSuper_Dep):
Esse gatilho foi chamado tr_vwFunc_Dep_InsteadOfDelete e se caracteriza por permitir através da visão a remoção de um departamento (TBL departamento) a partir da informação de seu número de identificação (dnumero).

Durante os testes de remoção foram feitas as seguintes operações (SQL):
```
DELETE FROM vwSuper_Dep WHERE dnumero = '10';
DELETE FROM vwSuper_Dep WHERE dnumero = '100';
```

Na primeira operação acima, o processo de remoção na visão ocorre com sucesso e o departamento é removido (TBL departamento). Já a segunda operação conclui sem nenhuma linha afetada pois o número de departamento (dnumero) inexiste na (TBL departamento).

Operação Insert: View 3 (vwFunc_Depend):
Esse gatilho foi chamado tr_vwFunc_Depend_InsteadOfInsert, e se caracteriza por permitir a inserção de um novo dependente (TBL dependente) a um funcionario já existente (TBL funcionario). Caso o cpf do funcionario informado durante a inserção do novo dependente não exista na relação de funcionários é exibida uma mensagem de erro ao usuário.

Durante os testes de inserção foram feitas as seguintes operações (SQL):
```
INSERT INTO vwFunc_Depend VALUES('Luiza', 'F', '2015-04-22', 'Daughter', '043842981',
NULL, NULL);
INSERT INTO vwFunc_Depend VALUES(Pedro, 'M', '2014-01-22', 'Son', '043842981', NULL,
NULL);
INSERT INTO vwFunc_Depend VALUES('Lucas', 'M', '2016-05-12', 'Son', '444444402', NULL,
NULL);
INSERT INTO vwFunc_Depend VALUES('Joao', 'F', '2016-05-12', 'Daughter', '444444402',
NULL, NULL);
INSERT INTO vwFunc_Depend VALUES('Josemar', 'M', '2016-08-16', 'Son', '043842981',
NULL, NULL)
```

INSERT INTO vwFunc_Depend VALUES('Ricardo', 'M', '2013-08-12', 'Son', '000000000', NULL, NULL);

Nas primeiras cinco operações acima, o processo de inserção ocorre com sucesso e o dependente é inserido (TBL dependente). Na quarta inserção é inserido propositalmente um dependente com o campo gen e o campo relacionamento incorretos propositalmente pois os mesmos serão atualizados na operação update elucidada a seguir. Já na última operação é tentado propositalmente inserir um dependente à um funcionário ao qual cpf inexiste na (TBL funcionario). A operação é interrompida e uma mensagem de erro exibida ao usuário.

Operação Update: View 3 (vwFunc_Depend):
Esse gatilho foi chamado tr_vwFunc_Depend_InsteadOfUpdate e se caracteriza por permitir a atualização de dados da tabela dependente, como gen e data_nasc. É ainda possível alterar os campos na (TBL funcionario) nome e sobrenome do funcionário. Caso o usuário tente alterar o cpf do funcionário é exibida uma mensagem de erro informando que o campo cpf do funcionário, utilizado para criação da view e presente na (TBL funcionário) e na (TBL dependente) não pode ser alterado.

Durante os testes de atualização foram feitas as seguintes operações (SQL):
UPDATE vwFunc_Depend set relacionamento = 'Son', gen = 'M' WHERE cpf_func = '444444402' AND nome = 'Joao';
UPDATE vwFunc_Depend set sobrenomeFunc = 'Souza' WHERE cpf_func = '043842981';
UPDATE vwFunc_Depend set nomeFunc = 'Josenildo' WHERE cpf_func = '000000000';
UPDATE vwFunc_Depend set cpf_func = '666666666' WHERE cpf_func = '043842981';

Na primeira operação acima, o processo de atualização na visão ocorre com sucesso, os campos inseridos propositalmente incorretos na quarta operação Insert anteriormente são corrigidos e o dependente atualizado na (TBL dependente). A segunda operação altera o nome do funcionário com sucesso. As duas últimas operações contudo são abortadas e uma mensagem de erro é exibida ao usuário pois tenta-se na terceira operação atualizar o nome de um funcionário que o cpf inexiste na (TBL funcionário) e a quarta operação tenta alterar o número do cpf do funcionário sem sucesso.

Operação Delete: View 3 (vwFunc_Depend):
Esse gatilho foi chamado tr_vwFunc_Depend_InsteadOfDelete e se caracteriza por permitir através da visão a remoção de um dependente de funcionário (TBL dependente) a partir da informação do número de cpf do funcionário e nome do dependente a ser removido.

Durante os testes de remoção foram feitas as seguintes operações (SQL):
DELETE FROM vwFunc_Depend WHERE cpf_func = '043842981' AND nome = 'Josemar';

Na operação acima, o processo de remoção na visão ocorre com sucesso e o dependente do funcionario correspondente ao cpf informado apelidado Josemar é removido na (TBL dependente).

4.3. Descrição das funções/procedimentos armazenados (o que a função faz? Quais os parâmetros de entrada? O que é retornado?).

OBSERVAÇÃO: Todos os códigos para a criação dos procedimentos armazenados (procedures) aqui citados estão disponíveis no campo apêndice deste documento.

Para esse trabalho foram desenvolvidos três procedimentos armazenados (Procedures). Todos os procedimentos foram criados utilizando em seu corpo a cláusula 'WITH ENCRYPTION' para aumento de segurança ao não permitir a visualização de seu código e ainda foi convencionado que o nome de todos os procedimentos armazenados devem iniciar seu nome com as letras 'sp' sem a sucessão de underscore.

4.3.1: Procedure 1: spPisoMinMax

Esse procedimento armazenado atua sobre a visão vwFunc_Dep e está apto a atualizar o salário do funcionário TBL(funcionario) quando o salário do funcionário informado através do seu cpf estiver abaixo do piso salarial convencionado em 10000 ou acima do teto máximo, convencionado em 85000. Caso o salário esteja abaixo do piso acima informado, o procedimento armazenado realizará a atualização (aumento) para o valor mínimo. Caso o salário esteja acima do teto acima informado o salário será atualizado (reduzido) para o valor máximo convencionado acima.
Para execução deste procedimento armazenado o parâmetro de entrada é o cpf do funcionário.

Durante os testes foram feitas as seguintes operações (SQL):
EXEC spPisoMinMax @cpfProc = '000000000'
EXEC spPisoMinMax @cpfProc = '043842981'

Na primeira operação realizada acima é exibida uma mensagem de erro ao usuário informando que o cpf informado inexiste na relação de funcionários. Na segunda operação o cpf informado é localizado na tabela funcionário através da vwFunc_Dep e

foi verificado que o salário do funcionário era inferior ao piso salarial convencionado. Ao final o salario do funcionario anteriormente equivalente a 7000 foi atualizado na TBL(funcionário) para o piso salarial: 10000.

4.3.2: Procedure 2: spGetFuncByGender

Esse procedimento armazenado atua sobre e visão vwFunc_Dep e está apto a retornar todos os funcionários de determinado gênero (masculino/feminino). Para execução deste procedimento o parâmetro de entrada é o caracter ‘M’ para obter os funcionários do sexo masculinos da visão, e ‘F’ para os de sexo femino.

Durante os testes foram feitas as seguintes operações (SQL):

```
EXEC spGetFuncByGender @Gender =  'F';
EXEC spGetFuncByGender @Gender =  'M;
```

4.3.3: Procedure 3: spAgeMinEmployee

Esse procedimento armazenado atua sobre a visão vwFunc_Dep e está apto a verificar se o funcionário é maior ou menor de idade, ou seja, se o cálculo de data atual - o campo data_nasc corresponde a mais ou menos de 18 anos. Caso seja inserido um cpf que inexista na (TBL funcionario) é exibida uma mensagem de erro ao usuário informando tal fato. Caso o funcionário seja maior de idade é exibida uma mensagem ao usuário informando que o funcionário é maior de idade. Contudo, caso o funcionário seja menor de idade seu registro é excluído na (TBL funcionário).
Para execução deste procedimento armazenado o parâmetro de entrada é o cpf do funcionário.

Durante os testes foram feitas as seguintes operações (SQL):

```
INSERT INTO vwFunc_Dep VALUES('Lara', NULL, 'Chediak', '022888144', '2005-02-16', '888 Centro, Goiânia, GO', 'F', '10000', NULL, 'Hardware');

EXEC spAgeMinEmployee @cpfProc = '000000000'
EXEC spAgeMinEmployee @cpfProc = '022888144'
EXEC spAgeMinEmployee @cpfProc = '444444402'
```

Na primeira operação sobre o procedimento armazenado acima não é retornada nenhuma linha, vez que o funcionário informado não existe na (TBL funcionário). Já a segunda operação realiza a exclusão do funcionário menor de idade previamente inserida na primeira linha das operações. A última operação por fim retorna uma mensagem informando que o funcionário localizado é maior de idade.

4.4. Descrição do esquema de autorização (Roles e Política de Segurança)

OBSERVAÇÃO: Todos os códigos para a criação dos papéis (Roles) e concessões de autorização (Grant) aqui citados estão disponíveis no campo apêndice deste documento.

Para o esquema de autorização foi utilizado o conceito de papéis (Roles).Foram definidos três tipos de usuários, atribuido-lhes diferentes níveis de autorização sobre as visões criadas, conforme descrição a seguir:

4.4.1- Diretor

Diretor possui permissão completa (leitura, inserção, atualização e exclusão) de dados de qualquer umas das 3 views (vwFunc_Dep, vwSuper_Dep e vwFunc_Depend). Ele poderá repassar seus privilégios a outros usuários.

4.4.2- Gerente

Gerente possui permissão completa (leitura, inserção, atualização e exclusão) de dados das views (vwFunc_Dep e vwFunc_Depend) e de repasse desses privilégios, todavia apenas permissão de leitura na view (vwSuper_Dep).

4.4.3- TimeRh

TimeRh possui permissão completa (leitura, inserção, atualização e exclusão) de dados da view (vwFunc_Depend) e apenas leitura na view (vwFunc_Dep). Não há permissão para repasse em nenhuma view ou para leitura, inserção atualização ou exclusão na view (vwSuper_Dep).

5. METODOLOGIA DE TRABALHO

Este trabalho foi desenvolvido individualmente por mim, Alexandre Oliveira dos Santos.

Para a realização, foram utilizadas algumas soluções para o desenvolvimento e ainda o gerenciamento da realização do trabalho: Foi realizado a análise e estudo de documentação do sistema gerenciador de banco de dados (SQL Server 2012 Express) para criação e manuseio e também para o solucionamento de erros gerais que ocorriam durante o desenvolvimento do trabalho, como durante à carga das tabelas em que o formato de data aceito pelo SGBD diferia do presente no arquivo disponibilizado (companhia-dados.sql), entre outros erros que foram solucionados e dúvidas sanadas através da análise de documentações em outros fóruns diversos.

Para criação deste documento foi utilizado a plataforma Google Drive e a ferramenta Google Documents para edição e gerenciamento de versões.

6. CONCLUSÃO

Através do estudo foi possível realizar na prática, grande parte da teoria visualizada em sala de aula sobre os diversos conceitos no campo da programação em banco de dados aqui abordados, através da bastante difundida linguagem SQL e o uso do SGBD SQL Server 2012 conjuntamente ao SQL Server Management Studio 17.

Foi visualizado e absorvido que através da utilização de algumas técnicas é possível aumentar a confiabilidade e segurança do banco de dados através de implementação de técnicas de restrições e controle sobre o banco e os objetos nele contido, além ainda de possibilitar a automatização de algumas tarefas rotineiras e proporcionar além dos benefícios anteriormente, o aumento da produtividade de funcionários e a melhoria na performance do banco correspondente.

Como citado na introdução desse trabalho, em um cenário em que o uso de informação é cada vez mais crescente se faz de extrema importância o estudo e conhecimento de técnicas de programação SQL para que a transformação de dados em informação e ainda o gerenciamento desta ocorra da melhor maneira e sem prejuízos aos interessados.

.

7. REFERÊNCIAS

1. - ELMASRI, R. e NAVATHE, S.B., Sistemas de Banco de Dados, 6ª ed., Pearson – Addison Wesley, 2011;
2. MANNINO, M. V., Projeto, Desenvolvimento de Aplicações e Administração de Banco de Dados, tradução da 3ª edição, São Paulo, McGraw-Hill, 2008;
3. DATE, C. J., Introdução a Sistemas de Banco de Dados, tradução da 8º edição americana, Ed. Campus, Rio de Janeiro, 2004;
4. Slides Disciplina Banco de Dados II - 2017-2, Universidade Federal de Goiás, Prof. Dr.-Ing. Leonardo Andrade Ribeiro.
5. Youtube.com/user/kudvenkat - Canal utilizado para estudos sobre Triggers.
6. https://docs.microsoft.com/pt-br/sql/sql-server/sql-server-technical-documentation - Documentação do SQL Server, Microsoft.

8. APÊNDICE

VIEWS e TRIGGERS

As expressões SQL do campo Views e Triggers estão organizadas da seguinte forma:

8.x: View y
8.x.1 - Expressões para criação da View y;
8.x.2 - Expressões para criação da Trigger (Insert) para a View y;
8.x.3 - Expressões para criação da Trigger (Update) para a View y;
8.x.4 - Expressões para criação da Trigger (Delete) para a View y;

8.1: View 1 - (vwFunc_Dep):

8.1.1: Expressão para a criação da View 1:

```
CREATE VIEW  vwFunc_Dep AS
SELECT DISTINCT
     nome, inicial, sobrenome, cpf, data_nasc, endereco, gen, salario, super_cpf, dnome
FROM funcionario AS FUNC INNER JOIN
departamento AS DEPT ON FUNC.dno= DEPT.dnumero
```

8.1.2: Expressão para criação da Trigger (Insert) para a View 1:

```
CREATE TRIGGER tr_vwFunc_Dep_InsteadOfInsert
ON vwFunc_Dep
INSTEAD OF INSERT
AS
BEGIN
        DECLARE @DeptID int

        SELECT @DeptID = dnumero
        FROM departamento
        JOIN inserted
        ON inserted.dnome = departamento.dnome

        IF(@DeptID IS NULL)
        BEGIN
                RAISERROR('Departamento nao encontrado!', 16, 1)
                RETURN
        END
```

```
	INSERT INTO funcionario(nome, inicial, sobrenome, cpf, data_nasc, endereco, gen,
salario, super_cpf, dno)
	SELECT nome, inicial, sobrenome, cpf, data_nasc, endereco, gen, salario, super_cpf,
@DeptID
	FROM inserted
END
```

8.1.3: Expressão para criação da Trigger (Update) para a View 1:

```
CREATE TRIGGER tr_vwFunc_Dep_InsteadOfUpdate
ON vwFunc_Dep
INSTEAD OF UPDATE
AS
BEGIN
-- Caso a atualizacao ocorrer sob o CPF do funcionario (Atencao)
IF(UPDATE(cpf))
BEGIN
	RAISERROR('O campo Cpf nao pode ser modificado!', 16, 1)
	RETURN
END

IF (UPDATE(nome))
BEGIN
	UPDATE funcionario SET nome = INSERTED.nome
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

IF (UPDATE(inicial))
BEGIN
	UPDATE funcionario SET inicial= INSERTED.inicial
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

IF (UPDATE(sobrenome))
BEGIN
	UPDATE funcionario SET sobrenome = INSERTED.sobrenome
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
```

```
END

IF (UPDATE(data_nasc))
BEGIN
	UPDATE funcionario SET data_nasc = INSERTED.data_nasc
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

IF (UPDATE(endereco))
BEGIN
	UPDATE funcionario SET endereco = INSERTED.endereco
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

IF (UPDATE(gen))
BEGIN
	UPDATE funcionario SET gen = INSERTED.gen
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

IF (UPDATE(salario))
BEGIN
	UPDATE funcionario SET salario = INSERTED.salario
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

--Verificar se novo campo super_cpf é um Supervisor valido. Caso este nao seja exibir Erro.
IF (UPDATE(super_cpf))
BEGIN
	DECLARE @Supercpf character(9)

	SELECT @Supercpf = departamento.ger_cpf
	FROM departamento
	JOIN inserted
	ON inserted.super_cpf = departamento.ger_cpf
```

```
IF(@Supercpf IS NULL)
BEGIN
	RAISERROR('Campo super_cpf nao corresponde a um supervisor valido!', 16, 1)
	RETURN
END

	UPDATE funcionario SET super_cpf = @Supercpf
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

-- Caso deseje alterar o nome do Departamento (dnome). Caso novo departamento nao exista na relação de departamento existente exibir mensagem de erro.
IF (UPDATE(dnome))
BEGIN
	DECLARE @DeptID int

	SELECT @DeptID = dnumero
	FROM departamento
	JOIN inserted
	ON inserted.dnome = departamento.dnome

	IF(@DeptID IS NULL)
	BEGIN
		RAISERROR('Nome de departamento nao encontrado!', 16, 1)
		RETURN
	END

	UPDATE funcionario SET dno = @DeptID
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf
END

END
```

8.1.4: Expressão para criação da Trigger (Delete) para a View 1:

```
CREATE TRIGGER tr_vwFunc_Dep_InsteadOfDelete
ON vwFunc_Dep
INSTEAD OF DELETE
```

```
AS
BEGIN
	DELETE funcionario
	FROM funcionario
	JOIN deleted
	ON funcionario.cpf = deleted.cpf
END
```

8.2: View 2 - (vwSuper_Dep):

8.2.1: Expressão para a criação da View 2:

```
CREATE VIEW  vwSuper_Dep AS
SELECT DISTINCT
    dnome,
    dnumero,
    FUNC.nome AS ger_nome,
    ger_cpf,
    ger_inicio_data
FROM funcionario AS FUNC INNER JOIN
departamento AS DEPT ON FUNC.cpf =  DEPT.ger_cpf
```

8.2.2: Expressão para criação da Trigger (Insert) para a View 2:

```
CREATE TRIGGER tr_vwSuper_Dep_InsteadOfInsert
ON vwSuper_Dep
INSTEAD OF INSERT
AS
BEGIN
	DECLARE @Trcpf character(9)

	SELECT @Trcpf = cpf
	FROM funcionario
	JOIN inserted
	ON inserted.ger_cpf = funcionario.cpf

	IF(@Trcpf IS NULL)
	BEGIN
		RAISERROR('CPF nao encontrado na relacao de Funcionarios!', 16, 1)
		RETURN
	END

	INSERT INTO departamento(dnome, dnumero, ger_cpf, ger_inicio_data)
	SELECT dnome, dnumero, @Trcpf, ger_inicio_data
	FROM inserted
END
```

8.2.3: Expressão para criação da Trigger (Update) para a View 2:

```
CREATE TRIGGER tr_vwSuper_Dep_InsteadOfUpdate
ON vwSuper_Dep
INSTEAD OF UPDATE
AS
BEGIN

IF (UPDATE(dnome))
BEGIN
	UPDATE departamento SET dnome = INSERTED.dnome
	FROM inserted
	JOIN departamento
	ON departamento.dnumero = inserted.dnumero
END

-- Caso a atualizacao ocorrer sob o numero do departamento (Atencao)
IF(UPDATE(dnumero))
BEGIN
	RAISERROR('O campo dnumero nao pode ser modificado!', 16, 1)
	RETURN
END

IF (UPDATE(ger_nome))
BEGIN
	UPDATE funcionario SET nome = ger_nome
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.ger_cpf
END

IF (UPDATE(ger_cpf))
BEGIN
	DECLARE @cpfSuperDep character(9)

	SELECT @cpfSuperDep = cpf
	FROM funcionario
	JOIN inserted
	ON funcionario.cpf = inserted.ger_cpf

	IF(@cpfSuperDep IS NULL)
	BEGIN
```

```
            RAISERROR('Cpf nao localizado na relacao de funcionarios!', 16, 1)
            RETURN
        END

        UPDATE departamento SET ger_cpf = inserted.ger_cpf
        FROM inserted
        JOIN departamento
        ON departamento.dnumero = inserted.dnumero
END

IF (UPDATE(ger_inicio_data))
BEGIN
        UPDATE departamento SET ger_inicio_data = inserted.ger_inicio_data
        FROM inserted
        JOIN departamento AS DEPT
        ON DEPT.dnumero = inserted.dnumero
END

END
```

8.2.4: Expressão para criação da Trigger (Delete) para a View 2:

```
CREATE TRIGGER tr_vwSuper_Dep_InsteadOfDelete
ON vwSuper_Dep
INSTEAD OF DELETE
AS
BEGIN
        DELETE departamento
        FROM departamento
        JOIN deleted
        ON departamento.dnumero = deleted.dnumero
END
```

8.3: View 3:

8.3.1: Expressão para a criação da View 3:

```
CREATE VIEW  vwFunc_Depend AS
SELECT DISTINCT
     DEPEND.nome, DEPEND.gen, DEPEND.data_nasc, relacionamento, FUNC.cpf AS
cpf_func, FUNC.nome AS nomeFunc, Sobrenome AS sobrenomeFunc
FROM funcionario AS FUNC INNER JOIN
dependente AS DEPEND ON FUNC.cpf =  DEPEND.cpf
```

8.3.2: Expressão para criação da Trigger (Insert) para a View 3:

```
CREATE TRIGGER tr_vwFunc_Depend_InsteadOfInsert
ON vwFunc_Depend
INSTEAD OF INSERT
AS
BEGIN
	DECLARE @Trcpf character(9)
	DECLARE @TrnomeFunc character varying(15)
	DECLARE @TrsobrenomeFunc character varying(15)

	SELECT @Trcpf = cpf_func, @TrsobrenomeFunc = sobrenomeFunc, @TrnomeFunc=
nomeFunc
	FROM funcionario
	JOIN inserted
	ON inserted.cpf_func = funcionario.cpf

	IF(@Trcpf IS NULL)
	BEGIN
		RAISERROR('Cpf nao localizado na relacao de Funcionarios!', 16, 1)
		RETURN
	END

	INSERT INTO dependente(nome, gen, data_nasc, relacionamento, cpf)
	SELECT nome, gen, data_nasc, relacionamento, @Trcpf
	FROM inserted
END
```

8.3.3: Expressão para criação da Trigger (Update) para a View 3:

```
CREATE TRIGGER tr_vwFunc_Depend_InsteadOfUpdate
ON vwFunc_Depend
INSTEAD OF UPDATE
AS
BEGIN
-- Caso a atualizacao ocorrer sob o CPF do funcionario (Atencao)
IF(UPDATE(cpf_func))
BEGIN
	RAISERROR('O campo Cpf nao pode ser modificado!', 16, 1)
	RETURN
END
```

```
IF (UPDATE(nome))
BEGIN
	UPDATE dependente SET nome = INSERTED.nome
	FROM inserted
	JOIN dependente
	ON dependente.cpf = inserted.cpf_func
END

IF (UPDATE(gen))
BEGIN
	UPDATE dependente SET gen = INSERTED.gen
	FROM inserted
	JOIN dependente
	ON dependente.cpf = inserted.cpf_func
END

IF (UPDATE(data_nasc))
BEGIN
	UPDATE dependente SET data_nasc = INSERTED.data_nasc
	FROM inserted
	JOIN dependente
	ON dependente.cpf = inserted.cpf_func
END

IF (UPDATE(relacionamento))
BEGIN
	UPDATE dependente SET relacionamento = INSERTED.relacionamento
	FROM inserted
	JOIN dependente
	ON dependente.cpf = inserted.cpf_func
END

IF (UPDATE(nomeFunc))
BEGIN
	UPDATE funcionario SET nome = nomeFunc
	FROM inserted
	JOIN funcionario
	ON funcionario.cpf = inserted.cpf_func
END

IF (UPDATE(sobrenomeFunc))
BEGIN
```

```
        UPDATE funcionario SET sobrenome = sobrenomeFunc
        FROM inserted
        JOIN funcionario
        ON funcionario.cpf = inserted.cpf_func
END

END
```

8.3.4: Expressão para criação da Trigger (Delete) para a View 3:

```
CREATE TRIGGER tr_vwFunc_Dependente_InsteadOfDelete
ON vwFunc_Depend
INSTEAD OF DELETE
AS
BEGIN
        DELETE dependente
        FROM dependente
        JOIN deleted
        ON dependente.cpf = deleted.cpf_func AND dependente.nome = deleted.nome
END
```

PROCEDURES

Expressões para criação das Procedures:

Procedure 1: spPisoMinMax

```
CREATE PROCEDURE spPisoMinMax
 @cpfProc character(9)
AS
BEGIN
        DECLARE @salFunc int = NULL

        SELECT @salFunc = salario
        FROM vwFunc_Dep
        WHERE cpf = @cpfProc

IF @salFunc < 10000
BEGIN
   UPDATE vwFunc_Dep SET salario = 10000
        FROM vwFunc_Dep
        WHERE cpf = @cpfProc
END

IF @salFunc > 85000
BEGIN
   UPDATE vwFunc_Dep SET salario = 85000
        FROM vwFunc_Dep
        WHERE cpf = @cpfProc
END

IF @salFunc >= 10000 AND @salFunc <= 85000
BEGIN
   PRINT 'O salario do funcionario esta dentro do limite convencionado!'
END

IF @salFunc IS NULL
BEGIN
   PRINT 'O cpf informado nao existe na relacao de funcionarios!'
END

END
```

Procedure 2: spGetFuncByGender

```
CREATE PROCEDURE spGetFuncByGender
@Gender character(1)
WITH ENCRYPTION
AS
BEGIN
      SELECT * FROM vwFunc_Dep
WHERE gen = @Gender
ORDER BY nome;
END
```

Procedure 3: spAgeMinEmployee

```
CREATE PROCEDURE spAgeMinEmployee
 @cpfProc character(9)
AS
BEGIN
        DECLARE @data_nasc date = NULL
        DECLARE @age int = NULL
        DECLARE @nomeDepend character varying(15)
        DECLARE @Today DATETIME = GETDATE()

        SELECT @data_nasc = data_nasc
        FROM vwFunc_Dep
        WHERE cpf = @cpfProc

        SELECT @age = FLOOR(DATEDIFF(DAY, @data_nasc, @Today) / 365.25)

IF @data_nasc IS NULL
BEGIN
   PRINT 'Nenhum funcionario encontrado para o cpf informado!'
END

IF @age < 18
BEGIN
        DELETE funcionario
        FROM funcionario
        WHERE cpf = @cpfProc
END

IF @age > 18
BEGIN
   PRINT 'Funcionario maior de idade!'
END

END
```

ROLES e AUTORIZAÇÕES

Expressões para criação das Roles:

```
CREATE ROLE diretor;
CREATE ROLE gerente;
CREATE ROLE TimeRh;
```

Expressões para concessão de autorização às Roles:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON vwSuper_Dep to diretor WITH GRANT
OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON vwFunc_Dep to diretor WITH GRANT
OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON vwFunc_Depend to diretor WITH GRANT
OPTION;

GRANT SELECT, INSERT, UPDATE, DELETE ON vwFunc_Dep to gerente WITH GRANT
OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON vwFunc_Depend to gerente WITH GRANT
OPTION;
GRANT SELECT ON vwSuper_Dep to gerente;

GRANT SELECT ON vwFunc_Dep to TimeRh;
GRANT SELECT, INSERT, UPDATE, DELETE ON vwFunc_Depend to TimeRh;
```