EP2: Dependencias Funcionais, micro serviços e serviços

Entrega: 26/05/2019

Objetivos

Os objetivos do conjunto de EP's dessa disciplina são voltados para a aplicação dos conceitos estudados em sala junto ao desenvolvimento, implementação e deploy de um sistema completo baseado em um problema real.

Neste EP nós aplicaremos alguns dos conceitos que não foram abordados na primeira parte, bem como começaremos a implementação do conjunto de serviços internos do tipo CRUD que serão disponibilizado através de uma extensão simplificada de uma arquitetura de micro serviços.

A principio você pode considerar que essa arquitetura será acessada por alguma API no back-end que dará acesso interno a esse conjunto de micro serviços.

No sistema final, garantidamente haverá no mínimo uma API pensada pelo lado do servidor (numa arquitetura monolítica ou semi monolítica) e eventualmente poderemos estender essa para uma arquitetura completa de micro serviços. Por essa razão, não nos preocuparemos com questões envolvendo REST, SOAP e derivados nessa fase, já que isso eventualmente será tarefa do EP3.

Conhecimentos envolvidos

A solução desse EP envolve o conhecimento/aplicação dos seguintes conceitos/ações:

Dependências funcionais, formas normais e normalização;

Conhecimentos em SQL (voltado para o SGBD PostgreSQL) e suas aplicações em DML.

Conhecimentos em PL/pgSQL e suas aplicações na construção de micro serviços envolvendo operações do tipo CRUD.

Organização do EP

Neste EP vocês deverão trabalhar em grupo de até 4 pessoas.

Apenas um dos integrantes do grupo deverá submeter o arquivo compactado até o dia 26/05/2019 às 23:55 e este será denominado como NUSP1 a partir desse ponto.

Para a entrega, o arquivo deverá conter a seguinte estrutura:

```
EP2_NUSP1_NUSP2_NUSP3_NUSP4.zip
├── EP2_README.txt
├── EP2_NUSP1_REPORT.pdf
├── EP2_NUSP1_DDL_FIXED.sql
├── EP2_NUSP1_DML.sql
├── EP2_NUSP1_DML_CLEAN.sql
├── EP2_NUSP1_DML_CREATE_GROUP.sql
├── EP2_NUSP1_DML_RETRIVAL_GROUP.sql
├── EP2_NUSP1_DML_UPDATE_GROUP.sql
└── EP2_NUSP1_DML_DELETE_GROUP.sql
```

Onde NUSP1, NUSP2, NUSP3 e NUSP4 correspondem ao número USP de cada um dos integrantes, separados por um underscore. (Obs.: Nos casos dos grupos individuais, duplas, ou trios basta omitir os NUSP's correspondentes)

O arquivo EP2_README.txt deve conter no cabeçalho o **nome completo** e NUSP de todos os integrantes e pelo menos um exemplo de chamada/instanciação de todas as funções/métodos implementados na sua API, além de qualquer outra eventual instrução de instalação/execução/uso que se fizerem necessárias.

No arquivo EP2_NUSP1_REPORT.pdf[1] vocês deverão entregar a documentação simplificada[2] de cada uma das functions e triggers desenvolvidos.

Sugerimos que essa apresentação seja feita em um relatório formal contendo:

Capa;

Sumário;

Introdução;

Breve apresentação das dependências funcionais de cada uma das relações da base, sua eventual normalização e as formas normais que elas respeitam;

Breve descrição dos micro serviços dos grupos CREATE, RETRIVAL, UPDATE e DELETE, com atenção especial para os do grupo RETRIVAL que devem especificar claramente a que se propõe a consulta;

Descrição e justificativas de todas as funções e triggers complementares construídos para a base que eventualmente não se enquadrem nos grupos supracitados.

O arquivo *DDL_FIXED.sql deve conter o DDL da base após todas as eventuais alterações que ocorrem por conta da normalização, simplificação. (Enviem esse DDL mesmo que sua base não sofra nenhuma mudança da versão do EP1 para o EP2)

O arquivo *DML.sql deve apresentar o Data Manipulation Language do seu modelo físico capaz de povoar a base com pelo menos 10 instancias em cada uma das relações. Já o *DML_CLEAN.sql deve apresentar o DML para remoção completa de todas as tuplas da base, via psql, sem que a estrutura da base seja afetada. Ou seja, você deve apagar todas as tuplas sem fazer um drop das tabelas que as contem, ficando a seu critério reinicializar ou não eventuais sequencias.

As próximas seções apresentam uma breve revisão teórica, bem como a apresentação e descrição detalhada de todos os itens que compõem este EP.

Revisão teórica

Nesta seção faremos uma breve revisão teórica de alguns dos conceitos que serão trabalhados nesse EP.

CRUD

Em ciência da computação, o acrônimo CRUD representa as quatro funções básicas de persistência de dados sobre bases relacionais. Existem algumas diferentes interpretações para a caracterização do acrônimo.

Nossa interpretação e a que vocês devem seguir nesse EP, juntamente com uma noção estendida para algumas outras linguagens e protocolos estão apresentados na tabela abaixo:

Operação	**SQL**	**HTTP**	**REST**
CREATE	INSERT	PUT/POST	POST
RETRIVAL	SELECT	GET	GET
UPDATE	UPDATE	PUT/POST/PATCH	PUT
DELETE	DELETE	DELETE	DELETE

API

O acrônimo API vem de *Application Programming Interface* e consiste de um conjunto de classes/métodos (ou funções) presentes uma applicação e que estabelecem um padrão totalmente documentado e bem estabelecido para o acesso as informações e funcionalidades de alguma outra aplicação.

Assim, quando um usuário (ou máquina) acessa uma API, ele tem um conjunto de métodos que permitem que ele interaja como a aplicação, sem conhecer qualquer detalhe de como a aplicação está implementada. (linguagem, versão, tipo de banco, funções, métodos, nome interno das relações, ...)

Neste EP nós não implementaremos ainda a API, mas é de extrema importância que as funções e triggers que serão acessíveis (diretamente ou via relação Perfil/Serviços) estejam minimamente documentadas no arquivo PDF, uma vez que sua API será baseada inteiramente nessa funções.

[1] Sugerimos o uso de LaTeX e submissões em outro formato que não seja .pdf não serão consideradas

[2] Maiores detalhes nas próximas seções

Micro serviços

A descrição a seguir foi extraída e adaptada diretamente das definições dos próprios proponentes da arquitetura. [2]

A arquitetura de micro serviços é uma abordagem para desenvolver uma única aplicação como uma suíte de serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente através de uma API HTTP. Estes serviços são construídos através de pequenas responsabilidades e publicados em produção de maneira independente através de processos de deploys automatizados. Existe um gerenciamento centralizado mínimo destes serviços, que podem ser escritos em diferentes linguagens e usarem diferentes tecnologias para armazenamento de dados.

Talvez uma das melhores formas de apresentar o padrão micro serviços é através de sua comparação com padrão monolítico. Uma aplicação monolítica é feita como uma única unidade. Aplicações empresariais são geralmente construídas em três partes principais: uma interface para o cliente (tais como páginas HTML e Javascript rodando em um navegador no computador do usuário ou mesmo uma interface gráfica responsável por estabelecer uma conexão direta com a base), um banco de dados (várias tabelas em um mesmo lugar, geralmente um sistema gerenciador de banco de dados relacional) e uma aplicação server-side, algumas vezes conhecida também como back-end. A aplicação server-side irá manipular as requisições (diretas ou via HTTP), executar toda lógica de domínio, receber e atualizar os dados da base de dados e por fim, selecionar e popular os blocos que retornarão para a API (uma página HTML, um arquivo texto ou mesmo um cursor com os resultados de uma consulta). Esta aplicação server-side é monolítica quando uma única unidade lógica é executável. Isso faz com que qualquer mudança no sistema exija a publicação de uma nova versão da aplicação server-side.

Embora aplicações monolíticas sejam a forma natural de se construir um sistema, e eventualmente até permitam alguma escalabilidade, quando estas passam a exigir alta escalabilidade e/ou implementação na nuvem, escalar o sistema exige que toda a estrutura monolítica seja propagada, mesmo quando as alterações ocorrem em um único módulo. A figura 1 extraída de [2] mostra um comparativo gráfico entre os dois modelos.

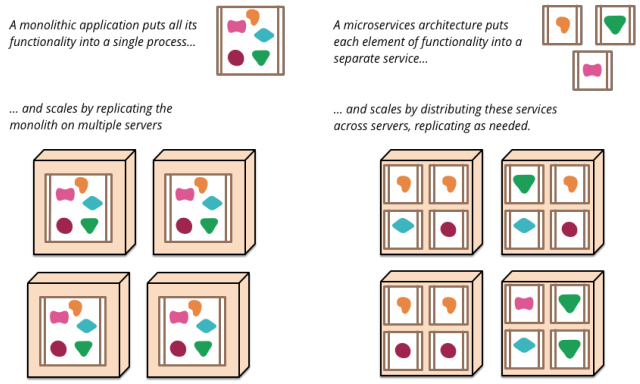

Fig. 1: Sistemas monolíticos vs sistemas micro serviços

Podemos ver que em um sistema totalmente baseado na arquitetura de micro serviços permite uma alta escalabilidade e a correção de erros pontuais de certos módulos, sem que o sistema todo tenha que ser refatorado. Isso gera uma melhor escalabilidade por demanda, ou seja, suponha que após executar o deploy do nosso sistema usando uma arquitetura de micro serviços e uma API HTTP/REST, nós venhamos a perceber que o módulo que controla o serviço que faz o planejamento das disciplinas é acessado 10 vezes mais que qualquer outro módulo.

Na arquitetura de micro serviços, nós poderíamos criar alguns novos servidores que ficariam responsáveis por tratar das requisições apenas desse módulo, permitindo também a escalabilidade temporal sobre demanda. Por exemplo, no período de matriculas esperamos que o sistema seja muito mais acessado que durante o meio do semestre, por isso, poderíamos ter, digamos, 5 servidores em paralelo rodando durante as matriculas, mas apenas 1 no contra período, o que ajudaria na redução do custo operacional do nosso sistema.

Um dos principais conceitos que caracterizam essa arquitetura é a presença de uma maior modularização dos componentes do sistema, ou seja, ao invés de termos um único

serviço que fizesse um select, e com base no retorno deste fizesse um update na relação X, uma inserção no log Y e depois uma deleção na relação Z, nós teremos vários micro serviços, cada qual responsável por executar apenas uma única tarefa.

Nesse EP nós trabalharemos nessa modularização interna da base, como forma de gerar o arcabouço para os micro serviços que serão implementados na API. Embora por questões de simplicidade nossa API eventualmente possa ter uma arquitetura mais voltada a arquitetura monolítica que a de micro serviços, modularizar agora permite que futuras APIs possam ser desenvolvidas usando praticamente qualquer arquitetura[3].

Resumo das atividades

Basicamente, as atividades deste EP resumem-se em:

I) Simplificar e avaliar o modelo desenvolvido no EP1 no que diz respeito as suas dependências funcionais e eventualmente normalizá-lo para atender ao maior número de formas normais que forem possíveis.

II) Para cada relação da base, escrever um script SQL para o PostgreSQL com a DML que insere (pelo menos 10 tuplas) em cada uma das relações e outra DML que "limpa" a base, apagando somente as tuplas inseridas (e eventualmente reinicializando sequencias[4]).

III) Para cada operação envolvendo CREATE, UPDATE e DELETE escrever uma function (e eventuais triggers) e documentá-los.

IV) Para cada operação envolvendo RETRIVAL, escrever uma function (e eventuais triggers), documentá-los e explicitar quais são os objetivos dessa consulta.

As próximas seções trazem um descrição mais detalhada de cada um desses itens.

Descrição detalhada

Parte I

A parte I tem dois itens principais:

1. Consolidação e compreensão do modelo unificado do módulo de acesso;
2. Simplificação, dependências funcionais e normalização.

I.1

Em I.1 você deve modificar o módulo de acesso do seu modelo apresentado no EP1.

Para isso, estamos fornecendo o script `SUPPORT.sql` que contém o modelo físico do módulo de acesso que der ser comum a todos os EP's a partir de agora.

Tudo que você precisa fazer é entender e ajustar a nomenclatura para se adequar ao seu modelo físico. Por exemplo, a relação `Usuário` está nomeada no script como `user` e seus atributos são `us_id`, `us_email`, `us_password`, caso você tenha indexado atribuição de ordem (tipo `tb01_user`) ou escrito em portugues, basta ajustar o script.

Como era esperado a presença de id's em todas as relações do módulo de acesso, além da aplicação do script o máximo que você terá que fazer é ajustar o tipo do id nas relações `us_pf` e `pf_se`.

I.2

Em I.2 e em todos os itens subsequentes, começamos a nos encaminhar para a produção do produto final. Nesse sentido, algumas das estratégias pedagógicas que foram tomadas na versão anterior, precisam ser revistas para essa versão.

Uma delas diz respeito a complexidade do modelo, ou seja, no EP1, estávamos interessados em discutir questões envolvendo a

[3] Além disso, nossa API terá no mínimo 4 conjuntos de serviços (um para cada letra do CRUD) o que a colocaria num patamar intermediário entre as duas arquiteturas citadas.

[4] Se você não reinicializar as sequencias, é preciso garantir que múltiplas execuções do tipo DML.sql/DML_CLEAN.sql/DML.sql... gerem os mesmos resultados em uma base limpa.

construção do modelo conceitual e seus mapeamentos. Para isso, era importante que cada relação do modelo contivesse um bom número de atributos e que esses atributos fossem de diferentes tipos (Compostos, multi valorados, simples, etc) e ainda que houvessem múltiplas chaves candidatas em algumas relações.

Agora, nosso foco está na implementação da base que servirá de arcabouço para as futuras API's, nesse sentido, alguns dos atributos que modelamos no EP1 deixam de ter importância para o produto final.

Por exemplo, guardar o telefone de um administrador pode ser muito importante, de um aluno talvez também seja, já que pode permitir que futuras extensões associem o sistema a mídias sociais. Agora, para o nosso sistema, guardar o endereço do aluno não é relevante, assim como também não será guardar informações redundantes de identificação, por exemplo, guardar RG e CPF de uma Pessoa. Em termos práticos, dificilmente uma pessoa que não seja nem Professor, nem Aluno nem Administrador será o alvo da nossa aplicação (talvez vestibulandos, ou país de alunos), mas mesmo nesses casos, o máximo que precisaríamos seria o CPF para garantir unicidade e parcialidade da especialização.

Logo, antes de começar essa parte, simplifique ao máximo o modelo do EP1, mantendo apenas os atributos que forem estritamente necessários.

Após simplificá-lo, escreva as dependências funcionais de todas as relações e veja se é possível normalizar a base ainda mais. Caso seja, normalize-a, caso contrário, apresente em uma tabela todas as dependências funcionais e qual forma normal cada uma delas respeita. Abaixo um pequeno exemplo do que é esperado nessa parte considerando uma base qualquer após a normalização:

Dependências funcionais na base A

```
FDNumber  Domain   Image
#01# ENAME -> {TotalSpending,
   NumEmployees, NroCompet,
   SalarySpending}
#02# CI     -> {RName, RSalary,
   RCompany}
⋮
#19# {EventIn, CountryIn} ->
   {Investiment, DelegationSize,
   PartNroO, PartNroP, PartNroB,
   PartRank}
```

FD Number	Normal Forms	Explanation
01	3NF	-
02	2NF	non prime RSalary and RCompany transitive in FD03
⋮		
19	3NF	-

Finalmente, após a normalização/simplificação, atualizem o DDL desenvolvido no EP1, escrevendo em *DDL_FIXED.sql.

Parte II

A parte II é autoexplicativa, e tudo que você precisa fazer é escrever dois scripts SQL para PostgreSQL: Um que insira pelo menos 10 tuplas[5] em cada uma das relações da base (*DML.sql) e um que "limpe" a base removendo todas as tuplas e eventualmente reinicializando as sequencias (*DML_CLEAN.sql).

Parte III

Para esse item você deve criar o conjunto de operações do tipo CREATE, UPDATE ou DELETE (CUD) usando PL/pgSQL para algumas relações da base. O conjunto de relações é dependente do modelo, mas pode ser pensado da seguinte forma:

Relações que envolvam operações CUD em entidades do módulo/grupo de pessoas, podem ser responsabilidade do próprio usuário/pessoa. Por exemplo, ao fazer o cadastro um usuário "ganha" o perfil de visitante e informa, durante o cadastro, se é ou não aluno. Caso seja aluno, ele informa os dados e a inserção em aluno vira uma função que pode ser acessada do perfil visitante. Nesse caso você deve criar as funções/triggers associados ao serviço `insert_student` e deve incluir na relação `pf_se` uma tupla explicitando a relação entre o perfil `visitante` e o serviço `insert_student`. Como esse serviço (`insert_student`) provavelmente será também acessível dos perfis `professor`, `administrador`, ... Para cada

[5] Algumas poucas relações, como perfil, podem ter menos que 10 tuplas

um deles você deverá inserir a associação na relação `pf_se`.

Já as relações que envolvam operações CUD em entidades "de maior poder", tanto do módulo de pessoas (Professor, Administrador) como do módulo de acesso (Serviço, Perfil), são de responsabilidade do Database administrator (DBA). Nesses casos fica a critério do grupo decidir se irão criar um usuário/-perfil para o(s) DBA(s) e criar as funções/triggers que permitiriam o acesso remoto desse perfil a base via interface ou se vão considerar que nessas relações o DBA fará a gestão diretamente na base, ou seja, logando no servidor e manipulando a base diretamente, sem a necessidade da interface.

Para o primeiro caso (BDA acessando via interface) vocês precisaram implementar as funções/triggers de todo o CUD dessas relações e precisaram ainda associar cada um dos serviços ao perfil DBA. No segundo caso (BDA acessando diretamente), vocês podem até criar funções/triggers para facilitar a vida do BDA, mas não há necessidade de associação dessas funções a nenhum serviço, já que ninguém irá poder acessá-los remotamente via interface. A segunda alternativa é mais simples de implementar, mas exige um maior conhecimento técnico do BDA, já a primeira é mais complicada de implementar, mas permite que o gestor do banco não seja um especialista.

Indo agora para as relações do módulo currículo, praticamente todas as operações CUD precisam de uma função/trigger e um perfil/-serviço associado. O mesmo ocorre para as relações envolvendo relacionamentos inter módulos (Ministra, Cursa, Planeja, Administra, ...).

Uma boa estratégia para usar em caso de dúvidas é perguntar de quem é a responsabilidade semântica do CUD naquela relação. Exemplo, na relação Planeja, o ato de planejar é de responsabilidade do aluno e portanto precisará funções/triggers e serviços (fts). O ato de inserir, deletar ou atualizar uma disciplina é de responsabilidade do coordenador do curso (administrador) e portanto precisa de fts, já a responsabilidade de criar um novo perfil na base é de competência do DBA e portanto não há obrigatoriedade na criação de fts.

O arquivo `SUPPORT.sql` traz um exemplo envolvendo operações de CREATE e UPDATE sobre a relação usuário.

Por fim, lembre-se que cada uma das funções/triggers devem ser documentados e, na maioria dos casos, ao criá-las, precisamos também criar um serviço e associar esse serviço a um perfil.

O conjunto de funções/triggers desenvolvidos para cada um dos grupos deve estar em seu respectivo *DML. Por exemplo, para operações de CREATE, todos as funções e triggers devem estar no `*DML_CREATE_GROUP.sql`, o mesmo ocorrendo para UPDATE e DELETE.

Parte IV

Basicamente, essa parte é idêntica a anterior, mas envolve agora operações de RETRIVAL.

Apesar disso, nessa parte além do `*DML_RETRIVAL_GROUP.sql`, você também precisa explicitar no relatório PDF qual é o objetivo e o resultado esperado de cada uma das consultas.

Avaliação

Seu EP será avaliado pela qualidade do relatório, pela expressividade da documentação e pelos scripts SQL. Sua nota final será composta por:

```
EP Parte 2 - Nota máxima 10.0
├── Relatório PDF (Item I) (2.0)
│   └── Estrutura do texto e qualidade
│       das discussões bônus de até 0.5
├── Item II (DML e DML_CLEAN) - (1.0)
├── Item II (CUD de CRUD) - (4.0)
└── Item III (R de CRUD) - (3.0)
```

Instruções para entrega

Você deve submeter via PACA os arquivos descritos na organização, compactados em formato .zip e seguindo as especificações de estrutura e nomenclatura previamente estabelecidas, até às 23:55 do dia 26/05/2019.

Para evitar que seu EP seja zerado, certifique-se que o arquivo foi submetido sem problemas (baixando, descompactando e testando todos os arquivos), que o relatório está em formato PDF, que os scripts SQL foram escritos para o PostgreSQL usando PL/pgSQL quando necessário.

Além disso, não deixe de preencher o cabeçalho contendo o nome completo e NUSP de todos os integrantes e atentem para as questões de integridade acadêmica, ou seja, caso você utilize diretamente, ou se baseie fortemente em fontes que não sejam as fornecidas em sala, não se esqueça de referenciá-las no relatório. Lembrem-se que casos de plágio serão tratados com o rigor acadêmico esperado.

Não deixe para a última hora e bom trabalho!

Referências

[1] Ramez Elmasri and Shamkant B. Navathe. *Database Systems, 7th Ed.* Pearson, 2015.

[2] James Lewis and Martin Fowler. Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html. Accessed: 02/05/19.

[3] Décio Lauro Soares, Eduardo Dias Filho, and Bruno Padilha. Tutorial python e postgresql. www.paca.ime.usp.br. Accessed: 20/03/19.

[4] Osvaldo Kotaro Takai, Isabel Cristina Italiano, and João Eduardo Ferreira. Introdução à banco de dados. www.ime.usp.br/~jef/apostila.pdf, 2005. Accessed: 20/03/19.