Projeto SISCOF
Disciplina de banco de dados – IFSC

Monitoramento de alertas (Crontab e Python)

- **Elaboração:**
  - **André Felippe Weber**
  - **Lucas Gomes Farias**

- **Data: 12/06/2017**

Introdução

O monitoramento da frequência dos alunos do câmpus são josé do IFSC será realizado através da criação de filtros que conterão informações de um aluno, uma turma ou um curso, assim como parâmetros a serem utilizados no monitoramento, como por exemplo a quantidade de dias que um aluno pode faltar.

As aplicações verificacaoFaltas.py e verificacaoHorario.py são responsáveis por verificar quais filtros estão ativos no sistema e então pesquisar, no banco de dados das catracas de entrada do câmpus, se o aluno frequentou a instituição em um dado dia e/ou se ele chegou e saiu no horário esperado. Quando algum dos alertas cadastrados ocorre esse dois *scripts* alerta o usuário. Por exemplo, o usuário do sistema pode cadastrar um alerta para monitorar um aluno que vêm faltando constantemente às aulas (utilizar a interface gŕafica para tal tarefa). As aplicações de verificação irão enviar uma notificação (e-mail) no dia seguinte em que os parâmetros definidos forem atendidos.

Bancos de dados

Três bancos de dados serão utilizado para realizar o monitoramento e registro de ocorrências.

- Controle de Frequência: É um banco implementado utilizando MySQL e possui as seguintes tabelas de interesse:
  - Alertas_Faltas: Contém todos os parâmetros necessário que o *script* verificaFaltas.py necessita para realizar pesquisas no banco de dados da catraca e do sistema acadêmico.
  - Alertas_Horarios: Contém todos os parâmetros necessário que o *script* verificaPontualidade.py necessita para realizar pesquisas no banco de dados da catraca e do sistema acadêmico.
  - Ocorrencia_Faltas: Armazena todas as ocorrências de faltas relacionadas aos alunos cadastrados na tabela Alertas_Faltas.
  - Ocorrencia_Horario: Armazena todas as ocorrências de chegadas tardias ou saídas antecipadas relacionadas aos alunos cadastrados na tabela Alertas_Horarios.
  - Registro_Faltas : Armazena os resultados de eventos ocorridos devido à um ou mais filtros instanciados na tabela Alertas_Faltas.
  - Registro_Horario: Armazena os resultados de eventos ocorridos devido à um ou mais filtros instanciados na tabela Alertas_Horarios.

    - Usuarios_Permitidos: Armazena todos os usuário permitidos no sistemas, isto é, aqueles que podem criar filtros.
- Catraca: É o banco de dados que armazena dados referentes à entrada e saída dos alunos do câmpus são josé. Este banco possui duas tabelas de interesse:
    - IFSC.USUÁRIO: Armazena a matrícula de todos os alunos matriculados no semestre.
    - IFSC.EVENTO: Armazena a data e hora que todo alunos chega e sai do câmpus.
- Sistema Acadêmico: É o banco de dados que armazena dados referentes aos alunos. Há uma tabela de interesse:
    - ALUNOS: Armazena dados pessoais do aluno como matrícula, turno, turma, etc.

Desenvolvimento

Os dois *scripts* são similares em vários pontos pois os dois acessam os mesmos bancos de dados e compartilham da mesma lógica de pesquisa e verificação de frequência. Porém optou-se por separar em dois *scripts* para que fique mais claro o propósito de cada um e para diminuir a complexidade de desenvolvimento e manutenção do código.

verificaFaltas.py:

O arquivo que fará a verificação da presença de um aluno é o verificaFaltas.py. Primeiramente este programa verifica quais filtros na tabela Alertas_Faltas estão ativos, isto é, quais deles possuem o campo dataFim com valor menor que a data do dia atual. O código então pode iniciar três tipos de verificação dependendo de quais campos da tabela Alertas_faltas foram preenchidos.

Se o campo matrícula estiver preenchido, então apenas um aluno (que possui a matrícula) será procurado. O programa verifica se o aluno em questão possui registro na tabela IFSC.EVENTO na data em que ele é executado. Se não consta nada referente àquele aluno na tabela IFSC.EVENTO então um registro de falta é feito na tabela Registro_Faltas, caso contrário o filtro é ignorado para aquela data. Se há um novo registro de falta o programa verifica se o aluno faltou o número máximo de vezes especificado na tabela Alertas_Faltas, e em caso positivo uma notificação (email) para o usuário que criou o alerta é gerado e esta ocorrência é registrada na tabela Ocorrência Faltas.

Já se o campo matrícula estiver vazio porém o campo turma foi preenchido com o código de uma das turmas do IFSC câmpus São José o programa busca no banco de dados do sistema acadêmico as matrículas de todos os alunos que estão cadastrados naquela turma. O código realiza, então, o mesmo processo explicado anteriormente de verificação da presença utilizando a matrícula de um aluno para todos os alunos matriculado naquela turma.

Para o caso dos campos matrícula e turma serem vazios, o campo curso deve ser preenchido com os três primeiros dígitos que identificam um curso do IFSC câmpus São José, por exemplo: Os três primeiro dígitos do curso de Engenharia de Telecomunicações é 290. O programa verifica no sistema acadêmico todos os alunos matriculados naquele curso, os separa por turma e então realiza a verificação da presença de cada aluno individualmente.

verificaPontualidade.py:

O comportamento do verificaPontualidade.py se assemelha bastante ao verificaFaltas.py pois os dois devem acessar os mesmos bancos para verificar somente um aluno, uma turma ou todos os alunos de um curso. Porém o verificaPontualidade.py deverá verificar se o aluno chegou ou saiu do câmpus São José dentro do período de tempo definido pelo usuário criador do alerta.

Assim sendo, ao instanciar um novo alerta o usuário deverá especificar uma quantidade de tempo, em minutos, que o aluno pode se atrasar em relação ao horário de início de suas aulas ou sair mais cedo em relação ao horário de fim de suas aulas. Os horários de início e fim das aulas de cada aluno se referem aos horários de cada turno de aulas do câmpus são josé, isto é, para os alunos matriculados no turno matutino os horários definidos para início e fim das aulas são, respectivamente, 7:30 e 11:30, para o turno vespertino os horários de início e fim são, respectivamente, 13:30 e 17:30 e para o turno noturno os horários são 18:30 e 22:30.

Conclusões:

Instalação das dependências:

1 - Instalar o python pip:

$ apt-get install python-pip

2 - Instalação do mysql-connector-python:

$ pip install mysql-connector-python

3 - Instalação do cx_Oracle:

Faça o download dos seguintes pacotes:

- Para um PC de 64 bits o download pode ser feito aqui
  - Oracle-instantclient12.1-devel-12.2.0.1.0-1.x86_64.rpm
  - oracle-instantclient12.1-basic-12.2.0.1.0-1.x86_64.rpm

- Para um PC de 64 bits o download pode ser feito aqui
  - Oracle-instantclient12.2-devel-12.2.0.1.0-1.i386.rpm
  - oracle-instantclient12.2-basic-12.2.0.1.0-1.i386.rpm
-

Converta todos os pacotes rpm para deb utilizando *alien*:

$ sudo alien -d *.rpm

Instale todos os pacotes deb:

$ sudo dpkg -i *.deb

Crie um ambiente ORACLE_HOME:

Crie um arquivo: **/etc/ld.so.conf.d/oracle.conf** e na primeira linha do *script* escreva:
Para instalação **64bits**:
/usr/lib/oracle/12.2/client64/lib

Para instalação **32bits**:
/usr/lib/oracle/12.2/client/lib

Rode os seguintes comandos:

```
$       export ORACLE_HOME=/usr/lib/oracle/12.1/client64
$       export LD_LIBRARY_PATH=$ORACLE_HOME/lib
$       sudo ldconfig
```

Faça o download do cx_Oracle *tar ball* do pypi e extraia (link para o download):

```
$       tar -xzvf cx_Oracle-5.1.3.tar.gz
$       cd cx_Oracle-5.1.3
```

Modifique as seguintes linhas do arquivo cx_Oracle-5.1.3/setup.py, da linha 123 (as linhas em vermelho):

```
# try to determine the Oracle home
#userOracleHome = os.environ.get("ORACLE_HOME")
userOracleHome = "/usr/lib/oracle/12.2/client64"
if userOracleHome is not None:
    if not CheckOracleHome(userOracleHome):
        messageFormat = "Oracle home (%s) does not refer to an " \
                "9i, 10g, 11g or 12c installation."
        raise DistutilsSetupError(messageFormat % userOracleHome)
else:
    for path in os.environ["PATH"].split(os.pathsep):
        if CheckOracleHome(path):
            break
    if oracleHome is None:
        #~ raise DistutilsSetupError("cannot locate an Oracle software " \
                #~ "installation
        oracleHome = "/usr/lib/oracle/12.2/client64"
```

OBS: Para instalações em **32 bits** as linhas vermelhas acima assumem o seguinte endereço /usr/lib/oracle/12.2/client

Faça um *Build* e instale o cx_Oracle 5.1.3:

```
$       python setup.py build
$       sudo python setup.py install
```

Configuração

Modificar arquivo configuracoes.cfg

O arquivo configuracoes.cfg define os valores para acesso aos bancos de dados (mysql e Oracle), bem como os valores para configuração do servidor SMPT (google).

```
[mysqlSection]
user = usuario_mysql
password = senha_mysql
host = servidor_mysql
database = banco_mysql

[smtpSection]
servidor = servidor_smtp
login = login_smtp
senha = senha_smtp
enviador = email_origem
mensagem = definir_mensagem

[oracleSection]
conexaoOracle1 = login/senha@servidor:porta/banco_dados
conexaoOracle2 = login/senha@servidor:porta/banco_dados
```

O arquivo já está configurado, porém qualquer modificação deve ser executada neste arquivo e não nas aplicações de verificação!

Configurar crontab

O cron é um programa de agendamento de tarefas. Utilizando dos seus recursos é possível programar qualquer coisa a ser executada em uma periodicidade ou até mesmo em um exato dia, numa exata hora.

Para o projeto em questão, queremos que os scripts verificacaoFaltas.py e verificacaoHorarios.py sejam execetudos às 3:00hs de terça-feira à sábado. A intenção de executá-los neste horário é impedir que o sistema da catraca esteja recebendo registros durante o período de verificação. Vale ressaltar que os programas de verificações são executados em relação ao dia anterior, por exemplo: na quarta-feira, será feita a verificação referente a terça-feira.

Para configurar o crontab com o agendamento desta tarefa é necessário editar o arquivo, /etc/crontab. Este arquivo só pode ser manipulado pelo root

```
$ sudo gedit /etc/crontab
```

Utilizamos o editor de texto gedit, utilize o da sua preferência.

O arquivo provavelmente terá alguns agendamentos pré-definidos, portanto será semelhante ao seguinte:

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user      command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root     test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root     test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root     test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
```

Podemos observar que o crontab tem o seguinte formato:

[minutos] [horas] [dias do mês] [mês] [dias da semana] [usuário] [comando]

O preenchimento de cada campo deve ser feito da seguinte maneira:

- Minutos: números de 0 a 59;
- Horas: números de 0 a 23;
- Dias do mês: números de 0 a 31;
- Mês: números de 1 a 12;
- Dias da semana: informe números de 0 a 7;
- Usuário: usuário que vai executar o comando;
- Comando: a tarefa que deve ser executada.

Desta maneira, para a configuração desejada, devemos incluir a seguinte instrução no arquivo:

Para que o crontab seja executado de terça-feira à sábado às 2 horas da manhã:

```
00 02 * * 2-6 root /path/to/python_file/verificaFaltas.py
00 02 * * 2-6 root /path/to/python_file/verificaPontualidade.py
```