Teste de Avaliação de Conhecimento – Curso de Spark – Módulo I

Objetivo:

Preencher tabelas (modeladas com base em modelo relacional) a partir de dados contidos em arquivos CSV.

Organização final dos dados no Hive:

Vide ANEXO I no fim deste documento

Passos para realização da tarefa

1 carga dos dados

1.1 criar esquemas manuais conforme as definições dos dados:

- **base de leitura:**

  PySpark StructType & StructField Explained with Examples - Spark by {Examples} (sparkbyexamples.com)

  Pyspark Data Types — Explained. The ins and outs—Data types… | by Diogo Veloso | BiLD Journal | Medium

  Data Types — PySpark 3.3.1 documentation (apache.org)

  PySpark SQL Types (DataType) with Examples - Spark by {Examples} (sparkbyexamples.com)

- **notebook de referência:**

  tópico_02-fontes_dados.ipynb

1.1.1 Esquema dos dados do arquivo/dataframe referente a compras:

nome no arquivo	nome no schema	tipo dos dados
Número do Contrato	nr_cont	int
Objeto	obj	text
Fundamento Legal	fund_leg	text
Modalidade Compra	mod_comp	text
Situação Contrato	sit_cont	text
Código Órgão Superior	cod_org_sup	int
Nome Órgão Superior	nm_org_sup	text
Código Órgão	cod_org	int
Nome Órgão	nm_org	text
Código UG	cod_ug	int
Nome UG	nm_ug	text
Data Assinatura Contrato	dt_ass_cont	date
Data Publicação DOU	dt_pub_dou	date
Data Início Vigęncia	dt_ini_vig	date
Data Fim Vigęncia	dt_fim_vig	date
CNPJ Contratado	cnpj_contrtd	long
Nome Contratado	nm_contrtd	text
Valor Inicial Compra	vl_ini_comp	text
Valor Final Compra	vl_fim_comp	text

1.1.2 Esquema dos dados do arquivo/dataframe referente a termos aditivos:

nome no arquivo	nome no schema	tipo dos dados
Número Contrato	nr_cont	int
Código Órgão Superior	cod_org_sup	int
Nome Órgão Superior	nm_org_sup	text
Código Órgão	cod_org	int
Nome Órgão	nm_org	text
Código UG	cod_ug	int
Nome UG	nm_ug	text
Número Termo Aditivo	nr_term_adit	int
Data Publicação	dt_pub	date
Objeto	obj	text

1.1.3 resultado esperado:

duas variáveis do tipo pyspark.sql.types.StructType contendo os esquemas criados.

1.2 efetuar as cargas dos dataframes:

- **base de leitura:**

  https://spark.apache.org/docs/2.3.2/sql-programming-guide.html#manually-specifying-options

- **notebook de referência:**

  tópico_02-fontes_dados.ipynb

1.2.1 criar dataframe **compraDF** a partir dos arquivos localizados no caminho:

/home/aluno/_spark/dados/originais/compras_publicas_federal/compras

1.2.2 criar dataframe **termoAditivoDF** a partir dos arquivos localizados no caminho:

/home/aluno/_spark/dados/originais/compras_publicas_federal/temos_aditivo

1.2.3 resultado esperado

- datataframe: **compraDF**
  - schema

    nr_cont:integer
    obj:string
    und_leg:string
    mod_comp:string
    sit_cont:string
    cod_org_sup:integer
    nm_org_sup:string
    cod_org:integer
    nm_org:string
    cod_ug:integer
    nm_ug:string
    dt_ass_cont:date
    dt_pub_dou:date
    dt_ini_vig:date
    dt_fim_vig:date
    cnpj_contrtd:long
    nm_contrtd:string
    vl_ini_comp:string
    vl_fim_comp:string

  - total de registros: **10823**
- datataframe: **termoAditivoDF**
  - schema

    nr_cont:integer
    cod_org_sup:integer
    nm_org_sup:string
    cod_org:integer
    nm_org:string
    cod_ug:integer
    nm_ug:string
    nr_term_adit:integer
    dt_pub:date
    obj:string

  - total de registros: **298**

2 - transformações

2.1 excluir registros de compra que estão duplicados

- **base de leitura:**

  PySpark Window Functions - Spark by {Examples} (sparkbyexamples.com)

  PySpark Window Functions - GeeksforGeeks

- **notebook de referência:** sem referência
- dataframe a ser utilizado: **compraDF**

2.1.1 deverão ser excluídos ambas as cópias e não apenas as duplicatas;

2.1.2 para eliminar as cópias, deverão ser realizadas as etapas:

2.1.2.1 identificar se há duplicidades utilizando a função de agregação **"count**" sobre uma Window **(vide seção 2.1 > base de leitura):**

- o resultado do count sobre uma Window deverá ser armazenado em uma coluna nomeada de **total_por_janela;**

2.1.2.2 realizar uma filtragem dos registros que conterem 1(um) na coluna **total_por_janela;**

2.1.2.3 eliminar a coluna **total_por_janela;**

2.1.2.4 o resultado deverá ser posto em um novo dataframe chamado **compraSemDuplicidadeDF.**

2.1.3 resultado esperado

- datataframe: **compraSemDuplicidadeDF**
  - schema

    nr_cont:integer
    obj:string
    fund_leg:string
    mod_comp:string
    sit_cont:string
    cod_org_sup:integer
    nm_org_sup:string
    cod_org:integer
    nm_org:string
    cod_ug:integer
    nm_ug:string
    dt_ass_cont:date
    dt_pub_dou:date
    dt_ini_vig:date
    dt_fim_vig:date
    cnpj_contrtd:long
    nm_contrtd:string
    vl_ini_comp:string
    vl_fim_comp:string

  - total de registros: **10719**

2.2 converter os campos string (que conterem número decimal) para double

- **base de leitura:**

  PySpark Convert String Type to Double Type - Spark by {Examples}

  PySpark Drop Rows with NULL or None Values

  https://www.geeksforgeeks.org/how-to-change-column-type-in-pyspark-dataframe/

  Data Types — PySpark 3.3.1 documentation (apache.org)

  pyspark.sql.Column.cast - Apache Spark

- **notebook de referência:**

  tópico_02-fontes_dados.ipynb

- **dataframe a ser utilizado: compraSemDuplicidadeDF**

2.2.1 para converter uma string para double deve-se:

2.2.1.1 trocar o divisor decimal, representado por vírgula, para ponto;

2.2.1.2 efetuar a troca do tipo para double;

2.2.1.3 o resultado deverá ser armazenado em uma coluna específica (lembrando que o procedimento deverá ser feito para duas colunas distintas – **vl_ini_comp** e **vl_fim_comp**);

2.2.1.4 as colunas contendo os números com tipo string deverão ser excluídos;

2.2.1.5 os nomes das colunas contendo os números com tipo double deverão ser renomeadas para os nomes das respectivas colunas excluídas no passo anterior (**vl_ini_comp** e **vl_fim_comp**);

2.2.1.6 eliminar registros que contiverem apenas valores nulos;

2.2.1.7 o resultado final deverá ser armazenado no dataframe **compraComCampoDoubleDF**.

2.2.2 resultado esperado

- datataframe: **compraComCampoDoubleDF**
  - schema

    nr_cont:integer
    obj:string
    fund_leg:string
    mod_comp:string
    sit_cont:string
    cod_org_sup:integer
    nm_org_sup:string
    cod_org:integer
    nm_org:string
    cod_ug:integer
    nm_ug:string
    dt_ass_cont:date
    dt_pub_dou:date
    dt_ini_vig:date
    dt_fim_vig:date
    cnpj_contrtd:long
    nm_contrtd:string
    vl_ini_comp:double
    vl_fim_comp:double

  - total de registros: **10687**

-

2.3 criar dataframe domOrgSupDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.3.1 verificar se um **cod_org_sup** está vinculado a mais de um **nm_org_sup**:

2.3.1.1 verificar quantos **cod_org_sup** distintos existem;

2.3.1.2 verificar quantas combinações distintas de **cod_org_sup** com **nm_org_sup** existem;

2.3.1.3 se obter uma quantidade de registros maior no passo **2.3.1.2**, então eliminar um dos registros de toda ocorrência de duplicidade:

2.3.1.3.1 selecione as combinações distintas da combinação **cod_org_sup** com **nm_org_sup;**

2.3.1.3.2 aplicar método de exclusão de duplicidade **(vide seção 2.3 base de leitura);**

2.3.1.3.3 armazenar o resultado no dataframe **domOrgSupDF;**

2.3.1.4 se obter a mesma quantidade de registros nos passos **2.3.1.1** e **2.3.1.2**, então:

2.3.1.4.1 selecione as combinações distintas da combinação **cod_org_sup** com **nm_org_sup;**

2.3.1.4.2 armazene o resultado da seleção no dataframe **domOrgSupDF**.

2.3.2 resultado esperado

- datataframe: **domOrgSupDF**
  - schema

    ```
    cod_org_sup:integer
    nm_org_sup:string
    ```

  - total de registros: **20**

2.4 criar dataframe domOrgDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.4.1 verificar se a combinação **cod_org_sup** com **cod_org** está vinculado a mais de um **nm_org**:

2.4.1.1 verificar quantas combinações distintas de **cod_org_sup** com **cod_org** existem;

2.4.1.2 verificar quantas combinações de **cod_org_sup** com **cod_org** e **nm_org** existem;

2.4.1.3 se obter uma quantidade maior de registros no passo 2.4.1.2, então eliminar um dos registros de toda ocorrência de duplicidade:

2.4.1.3.1 selecione as combinações distintas da combinação **cod_org_sup** com **cod_org** e **nm_org;**

2.4.1.3.2 aplicar método de exclusão de duplicidade **(vide seção 2.4 base de leitura);**

2.4.1.3.3 armazenar o resultado no dataframe **domOrgDF;**

2.4.1.4 se obter a mesma quantidade nos passos 2.4.1.1 e 2.4.1.2, então:

2.4.1.4.1 selecione as combinações distintas da combinação **cod_org_sup** com **cod_org** e **nm_org;**

2.4.1.4.2 armazene o resultado da seleção no dataframe **domOrgDF**.

2.4.2 resultado esperado

- datataframe: **domOrgDF**
  - schema

    ```
    cod_org_sup:integer
    cod_org:integer
    nm_org:string
    ```

  - total de registros: **221**

2.5 criar dataframe domUgDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.5.1 verificar se a combinação **cod_org_sup** com **cod_org** e **cod_ug** está vinculado a mais de um **nm_ug**:

2.5.1.1 verificar quantas combinações distintas das colunas **cod_org_sup** com **cod_org** e **cod_ug** existem;

2.5.1.2 verificar quantas combinações das colunas **cod_org_sup** com **cod_org** e **cod_ug** e **nm_ug** existem;

2.5.1.3 se obter uma quantidade maior de registros no passo 2.5.1.2, então eliminar um dos registros de toda ocorrência de duplicidade:

2.5.1.3.1 selecione as combinações distintas das colunas **cod_org_sup** com **cod_org** e **cod_ug** e **nm_ug;**

2.5.1.3.2 aplicar método de exclusão de duplicidade **(vide seção 2.5 base de leitura);**

2.5.1.3.3 armazenar o resultado no dataframe **domUgDF;**

2.5.1.4 se obter a mesma quantidade nos passos 2.5.1.1 e 2.5.1.2, então:

2.5.2.4.1 selecione as combinações distintas das colunas **od_org_sup** com **cod_org** e **cod_ug** e **nm_ug;**

2.5.1.4.2 armazene o resultado da seleção no dataframe **domUgDF**.

2.5.2 resultado esperado

- datataframe: **domUgDF**
  - schema

    cod_org_sup:integer
    cod_org:integer
    cod_ug:integer
    nm_ug:string

  - total de registros: **1700**

2.6 Criar dataframe domContrtdDF

- **base de leitura:**

  PySpark Window Functions - Spark by {Examples} (sparkbyexamples.com)

  PySpark Window Functions - GeeksforGeeks

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.6.1 **Observação**: há alguns CNPJs vinculados há mais de um nome de empresa, tal circunstância e motivada pela alteração do contrato que constitui a empresa, você pode verificar algumas duplicidades consultando pela seguinte lista de CNPJ:

- 14533285000130,
- 15616498000199,
- 29138207000109,
- 904728000490,
- 10875601000100,
- 10278243000140,
- 12591441000194,
- 15219654000188

2.6.2 eliminar os registros duplicados:

2.6.2.1 aplicar ordenação decrescente com base na coluna **dt_ass_cont;**

2.6.2.2 selecionar, forma distintas, a combinação **cnpj_contrtd** com **nm_contrtd;**

2.6.2.3 aplicar método de exclusão de duplicidade **(vide seção 2.6 > base de leitura);**

2.6.2.4 criar um id para cada registros utilizando a função de geração de número de linha **"row_number"** sobre uma Window **(vide seção 2.6 > base de leitura)**:

- o resultado do count sobre uma Window deverá ser armazenado em uma coluna nomeada de **cod_contrtd**.

2.6.2.4.1 o resultado deverá ser armazenado no dataframe **domContrtdDF**.

2.6.3 resultado esperado

- datataframe: **domContrtdDF**
  - schema

    ```
    cod_contrtd:integer
    cnpj_contrtd:long
    nm_contrtd:string
    ```

  - total de registros: **6043**

2.7 criar dataframe domSitContDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

  PySpark Repartition() vs Coalesce() - Spark by {Examples}

  https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.monotonically_increasing_id.html

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.7.1 selecionar valores distintos da coluna **sit_cont**

2.7.2 aplicar redução de número de partições para 1

2.7.3 criar a coluna **cod_sit_cont** e adicionar sequencia numérica utilizando a função **monotonically_increasing_id()**

2.7.4. armazenar o resultado no dataframe **domSitContDF**

2.7.5 resultado esperado

- datataframe: **domSitContDF**
  - schema

    cod_sit_cont:long
    desc_sit_cont:string

  - total de registros: **8**

2.8 criar dataframe domModCompDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}

  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

  PySpark Repartition() vs Coalesce() - Spark by {Examples}

  https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.monotonically_increasing_id.html

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF**

2.8.1 selecionar valores distintos da coluna **mod_comp**

2.8.2 aplicar redução de número de partições para 1

2.8.3 criar a coluna **cod_mod_comp** e adicionar sequencia numérica utilizando a função **monotonically_increasing_id()**

2.8.4. armazenar o resultado no dataframe **domModCompDF**

2.8.5 resultado esperado

- datataframe: **domModCompDF**
  - schema

    cod_mod_comp:long
    desc_mod_comp:string

  - total de registros: **9**

2.9 criar o dataframe compraListDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}
  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

  PySpark Repartition() vs Coalesce() - Spark by {Examples}

  https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.monotonically_increasing_id.html

  PySpark Join Two or Multiple DataFrames

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

  tópico_01-transformações.ipynb

- **dataframe a ser utilizado: compraComCampoDoubleDF, domContrtdDF, domSitContDF, domModCompDF**

2.9.1 criar variável para armazenar expressão de igualdade entre os dataframes **compraComCampoDoubleDF** e **domContrtdDF;**

2.9.2 criar variavel para armazenar expressão de igualdade entre os dataframes **compraComCampoDoubleDF** e **domSitContDF;**

2.9.3 criar variavel para armazenar expressão de igualdade entre os dataframes **compraComCampoDoubleDF** e **domModCompDF;**

2.9.4 efetuar **left join** entre os dataframes **compraComCampoDoubleDF** e **domContrtdDF;**

2.9.5 efetuar **left join** entre os dataframes **compraComCampoDoubleDF** e **domSitContDF;**

2.9.6 efetuar **left join** entre os dataframes **compraComCampoDoubleDF** e **domModCompDF;**

2.9.7 aplicar redução de número de partições para 1;

2.9.8 criar a coluna **cod_comp** e adicionar sequencia numérica utilizando a função **monotonically_increasing_id();**

2.9.9 armazenar o resultado no dataframe **compraListDF;**

2.9.10 **OPCIONAL**: retirar os termos "objeto: " (dos conteúdos presentes na coluna **obj**) e "fundamento legal: " (dos conteúdos presentes na coluna **fund_leg**).

2.9.11 resultado esperado

- datataframe: **compraListDF**
  - schema

    cod_comp:integer
    cod_org_sup:integer
    cod_org:integer
    cod_ug:integer
    nr_cont:integer
    obj:string
    fund_leg:string
    dt_ass_cont:date
    dt_pub_dou:date
    dt_ini_vig:date
    dt_fim_vig:date
    vl_ini_comp:double
    vl_fim_comp:double
    cod_mod_comp:long

cod_sit_cont:long
cod_contrtd:integer

- total de registros: **10687**

2.10 criar dataframe termoAditListDF

- **base de leitura:**

  PySpark Count Distinct from DataFrame - Spark by {Examples}
  PySpark Distinct to Drop Duplicate Rows - Spark by {Examples}

  PySpark Repartition() vs Coalesce() - Spark by {Examples}

  https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.monotonically_increasing_id.html

  PySpark Join Two or Multiple DataFrames

- **notebook de referência:**

  tópico_04-agregação_dados.ipynb

  tópico_01-transformações.ipynb

- **dataframes a serem utilizados: termoAditivoDF e compraListDF**

2.10.1 criar variavel para armazenar expressão de igualdade entre os dataframes **compraListDF** e **termoAditivoDF**

2.10.2 efetuar **inner join** entre os dataframes **termoAditivoDF** e **compraListDF**

2.10.3 aplicar redução de número de partições para 1

2.10.4 criar a coluna **cod_term_adit** e adicionar sequencia numérica utilizando a função **monotonically_increasing_id()**

2.10.3 armazenar resultado no dataframe **termoAditListDF**

2.10.4 **OPCIONAL**: retirar o termo "objeto: " (dos conteúdos presentes na coluna **obj**)

2.9.11 resultado esperado

- datataframe: **termoAditListDF**
  - schema

    cod_term_adit:integer
    cod_comp:integer
    nr_term_adit:integer
    dt_pub:date
    obj:string

  - total de registros: **298**

3 Persistência dos dados

- **base de leitura:**

  PySpark Repartition() vs Coalesce() - Spark by {Examples}

  https://spark.apache.org/docs/2.3.3/sql-programming-guide.html#saving-to-persistent-tables

- **notebook de referência:**

  tópico_02-fontes_dados.ipynb

- **dataframes a serem utilizados: domOrgSupDF, domOrgDF, domUgDF, domContrtdDF, domSitContDF, domModCompDF, compraListDF e termoAditivoDF**

3.1 todos os dataframes deverão ser persistidos em forma de tabelas gerenciadas/persistidas, ou seja, registradas no metastore do hive.

3.2 todas as tabelas deverão ficar dentro da base de dados **aula_fd** (no Hive)

3.3 as tabelas deverão ter como padrão o nome: "<nome do dataframe>", exemplo:

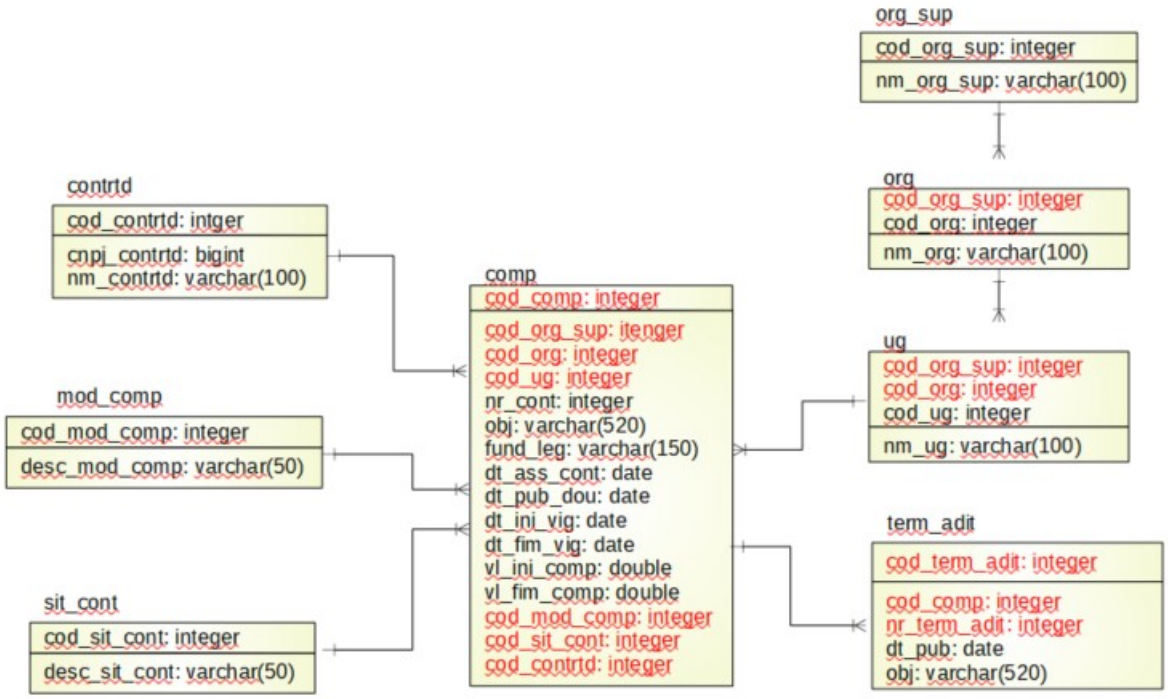
aula_fd.domContrtdDF

3.4 antes de gravar qualquer dataframe deverá conter apenas uma partição

3.5 resultado esperado

- tabelas:
  - **aula_fd.domOrgSupDF**
  - **aula_fd.domOrgDF**
  - **aula_fd.domUgDF**
  - **aula_fd.domContrtdDF**
  - **aula_fd.domSitContDF**
  - **aula_fd.domModCompDF**
  - **aula_fd.compraListDF**
  - **aula_fd.ermoAditivoDF**
- total de tabelas: **8**

ANEXO I


org_sup
cod_org_sup: integer
nm_org_sup: varchar(100)
org
cod_org_sup: integer
cod_org: integer
nm_org: varchar(100)
contrtd
cod_contrtd: intger
cnpj_contrtd: bigint
nm_contrtd: varchar(100)
comp
cod_comp: integer
cod_org_sup: itenger
cod_org: integer
cod_ug: integer
nr_cont: integer
obj: varchar(520)
fund_leg: varchar(150)
dt_ass_cont: date
dt_pub_dou: date
dt_ini_vig: date
dt_fim_vig: date
vl_ini_comp: double
vl_fim_comp: double
cod_mod_comp: integer
cod_sit_cont: integer
cod_contrtd: integer
ug
cod_org_sup: integer
cod_org: integer
cod_ug: integer
nm_ug: varchar(100)
mod_comp
cod_mod_comp: integer
desc_mod_comp: varchar(50)
term_adit
cod_term_adit: integer
cod_comp: integer
nr_term_adit: integer
dt_pub: date
obj: varchar(520)
sit_cont
cod_sit_cont: integer
desc_sit_cont: varchar(50)