

dbd27.appspot.com/o/logos%2Flogo_conecsync_dark.svg?alt=media =300x) *Documentação da versão
NodeJs do script de sincronização com projetos da Conecdata, ConecSync v 1.0.

- API
- CLI
- Conecdata
- CSV
- ERP
- Integradores ou integradoras
- Tokens de loja
- Views

- Modo api
- Modo script
- Modo mixto

- Mercadeiro

- Acesso direto ao banco de dados
- Leitura de arquivos CSV

- Instalação NodeJs
- Copiando o script ConecSync
- Estrutura da pasta do ConecSync
- Instalando dependências

- Integração modelo DB
 - Criando o banco de dados
 - Criando as tabelas modelo

- [Tabela grupos](#)
- [Tabela subgrupos](#)
- [Tabela produtos](#)
- [Tabela formas pgto](#)
- [Criando as views](#)
 - [view_conecdata_formas](#)
 - [view_conecdata_produtos](#)
 - [view_conecdata_estoque](#)
- [Configurando modo DB](#)
 - [Configuração geral](#)
 - [Configurando origens](#)
 - [Configurando projetos](#)
- [Integração modelo CSV](#)

Testando sua integração

- [Linux](#)

Executando o script

- [Linux](#)
- [Windows](#)

Glossário

Antes de mais nada, vamos esclarecer alguns termos, necessários, que surgirão constantemente no decorrer dessa documentação:

API

É um mecanismo de comunicação entre dois programas distintos.

CLI

É uma interface que pode ser acionada por meio de um comando em um terminal de seu sistema operacional.

Conecdata

É a softhouse desenvolvedora do **ConecSync** e dos projetos compatíveis com ele, alguns deles, que permitem sincronização (integração) com [ERPs](#) desenvolvidos por softhouses parceiras (aka [integradoras](#)).

CSV

São arquivos texto puro utilizados para compartilhamento de informações, em que os valores em cada linha são separados por um delimitador (normalmente uma vírgula, mas no nosso caso um ponto e vírgula).

ERP

É o programa que o lojista já usa para administrar sua empresa, desenvolvido por uma softhouse (aka [integradora](#)) parceira da [Conecdata](#).

A integração do ERP com algum de nossos projetos evita o retrabalho de realizar modificações de dados em ambas plataformas, uma vez que o ERP comunica automaticamente suas modificações para nossos projetos.

Integradores ou integradoras

Integradores/integradoras são empresas desenvolvedoras de softwares [ERPs](#) de mercados ou supermercados, que se comunicam com alguma de nossas plataformas, por qualquer um dos modos descritos nessa documentação, permitindo a integração de seus dados.

Para que um lojista possa utilizar alguma de nossas plataformas compatíveis com o **ConecSync**, é essencial que a desenvolvedora de seu sistema (integradora) realize a integração de seu ERP conosco.

Tokens de loja

Podemos entender os tokens de loja, como sendo chaves de acesso às [apis](#) de projetos da [Conecdata](#) compatíveis com o **ConecSync**. Esses tokens tem as seguintes características:

- São exclusivos para um [integrador](#), ou seja, tokens para [ERPs](#) do **integrador A** não servem para outros do **integrador B**.
- São exclusivos para uma loja, ou seja, tokens da **loja A** não servem para a **loja B**.
- São exclusivos para um projeto, ou seja, tokens do **projeto A** não servem para o **projeto B**.
- São exclusivos do modo de distribuição, ou seja, tokens de **modo sandbox** não funcionam para **modo produção**.
- São revogáveis, ou seja, o **integrador** pode invalidar os tokens vigentes e gerar outros que os substituam, sempre que quiser.

Para integração via api, o próprio lojista indicará seu token de loja para cada projeto em seu ERP. Em caso de integração via script, o integrador indicará os mesmos tokens, em áreas específicas de configuração do **ConecSync**.

Views

Uma **view** é uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada.

Optamos por utilizar views por diversos motivos, elas podem reunir dados de mais de uma tabela facilmente, por serem uma visão paralela dos dados, podem ter configurações específicas de segurança atribuídas especificamente à ela, entre outros.

Modos de integração

É importante entender quais opções de integração do **ConecSync** estão disponíveis e as vantagens e desvantagens, de cada uma delas.

Modo api

Nesse modo, o **integrador** deve incluir em seu código fonte, chamadas diretamente às **apis** dos projetos da **Conecdata** com os quais deseja integrar, reagindo à ocorrência de eventos específicos, tais como:

- Inclusão e modificação de departamentos, subdepartamentos, produtos, promoções e formas de pagamento.
- Vendas e entradas de estoque.

VANTAGENS

- **Maior velocidade de sincronização:** As sincronizações são realizadas no exato momento em que as modificações ocorrem.

DESVANTAGENS

- **Maior trabalho e testes:** Devido à necessidade de modificação do código fonte, esse modo demandará mais tempo, estudo e testes para ser devidamente implementado.

Como essa documentação trata apenas da integração via o script **ConecSync**, recomendamos a consulta da documentação específica da integração via api, para maiores informações.

Modo script

Nesse modo, basta o download, instalação, configuração e execução regular do script **ConecSync** para realizar as sincronizações desejadas. As **apis** dos projetos configurados, serão chamadas pelo **ConecSync**, com base na leitura direta dos cadastros do sistema **ERP** parceiro, ou de arquivos **CSV**, gerados periodicamente a partir deles.

VANTAGENS

- **Facilidade de implementação:** Por não necessitar de nenhuma modificação em seu código fonte e executar código já amplamente testado, a possibilidade de erros nas chamadas da **api** e necessidades de testes são reduzidas drasticamente.
- **Maior recuperação de falhas:** Caso alguma tentativa de acesso à **api** falhe nesse modo, a próxima execução do **ConecSync** detectará novamente a mudança e repetirá a tentativa.

DESVANTAGENS

- **Menor velocidade de sincronização:** As sincronizações são realizadas a cada execução do **ConecSync** e não imediatamente como ocorre no modo **api**.

Optando por esse modo, o restante da documentação pretende exatamente detalhar e exemplificar cada um dos passos necessários para implementá-la.

Modo mixto

A **integradora** pode monitorar algumas modificações das origens diretamente de seu código, para se beneficiar das sincronizações mais rápidas e ainda assim, executar o **ConecSync** periodicamente para realizar algumas outras tarefas, que poderiam demandar uma modificação mais trabalhosa em seu código, como por exemplo, o monitoramento do estoque.

Agora que conhece as modalidades de integração disponíveis, caso opte por realizar todas integrações diretamente via api e não utilizar o script **ConecSync** em nenhum caso, a leitura do restante dessa documentação se torna desnecessária. Nesse caso é aconselhável que transfira sua leitura para a documentação de integração via api, para maiores detalhes.

Projetos compatíveis

No caso de integração via [api](#), uma vez gerado um objeto das informações modificadas em seus cadastros que devam ser enviadas para sincronização, esse mesmo objeto, pode ser enviado para [apis](#) de mais de um projeto compatível, permitindo assim múltiplas sincronizações de suas modificações, com um mínimo de ajuste no seu código. No caso de integração via script **ConecSync**, cada integração pode ser configurada para ser utilizada ou não, facilitando que se sincronize apenas com os projetos desejados. Como você pode chamar as [apis](#) que quiser de seu código e configurar as que deseja ou não sincronizar pelo **ConecSync**, torna-se muito fácil utilizar ambos modos (api e script), paralelamente. Apenas alguns projetos da Conecdata são compatíveis com a sincronização descrita por essa documentação, são eles:

Mercadeiro

O Marketplace de mercados da Conecdata.

Ok, por enquanto existe apenas um projeto disponível para integração via **ConecSync** (ou por apis), mas aguardem, ele é o primeiro, e já existem alguns outros planejados que serão desenvolvidos em breve e constarão em versões futuras dessa documentação.

Origens

É considerada uma origem, cada fonte de informações disponível para consulta, requerida para sincronização via **ConecSync**.

- Produtos (contendo departamentos e subdepartamentos)
- Estoque
- Formas de pagamento
- Promoções (em desenvolvimento)

Nas integrações via api, não há indicações de origens, uma vez que você mesmo já lê seus dados pelo código (na maioria das vezes as mesmas tabelas e campos) e simplesmente aciona uma chamada de api para sincronizá-los.

São três, os **tipos** possíveis para cada origem disponível, na integração via script:

Tipo	Descrição
"	Não sincronizar (ignorar) essa origem
db	Sincronizar essa origem por leitura direta ao banco de dados
csv	Sincronizar essa origem por leitura de arquivo csv

Acesso direto ao banco de dados

Nesse tipo de origem, os dados são lidos de [views](#) geradas a partir de seus cadastros. Apenas alguns gerenciadores SQL são compatíveis com o **ConecSync**:

- mysql
- mariadb
- postgres
- mssql

Para utilização de gerenciadores não compatíveis ou bancos de dados NoSql, a sincronização pode ser feita pela leitura de arquivos CSV exportados de seus cadastros, antes da execução do script.

O QUUEEEEE deixar um script estranho acessar meus dados, NEM PENSAR. É legítimo que você se preocupe em confiar em um script estranho acessando seus cadastros. Para eliminar seus medos e dúvidas, deve observar o seguinte:

- Todo acesso é feito de maneira indireta em [views](#) de suas tabelas (mais adiante a documentação orienta suas criações) que você mesmo vai criar e podem ser atribuídas a um usuário, com acesso apenas leitura e a apenas essas views. Dessa forma você pode ficar seguro que limitou o acesso apenas às informações relevantes à sincronização e sem possibilidade de modificação tanto delas quanto outras em seus cadastros.
- Todas credenciais de acesso são alimentadas por você mesmo, garantindo que o script não consiga acessar nenhuma informação fora das atribuições desse usuário ou que alguém desconhecido tenha tido acesso à sua senha ou qualquer outra informação confidencial.
- Todo código fonte está disponível para que você com uma consulta rápida (mesmo que não programe em javascript, na verdade typescript, ou conheça o Node Js) verifique facilmente que ele apenas detecta mudanças nas origens e aciona as [apis](#) configuradas quando necessário. Aliás, o código fonte do **ConecSync** é muito pequeno e de fácil análise (mais adiante cada arquivo disponível no script é explicado detalhadamente) .

Leitura de arquivos CSV

Caso (apesar dos esclarecimentos do tópico anterior) você não queira ou por qualquer outro motivo, não possa, utilizar o script (gerenciadores db incompatíveis, nosql...), você pode escrever você mesmo um código que exporte as informações necessárias para arquivos [CSVs](#) (mais adiante a documentação orientará como isso deve ser feito em maiores detalhes) e indicá-los como origem para sincronização.

Instalação

Bom, se você chegou até aqui vamos presumir que já está familiarizado com os termos gerais, os tipos disponíveis de sincronização (e optou por alguma que use o **ConecSync**), já sabe que pode optar por tipos de origens **DB** ou **CSV**, que pode escolher quais plataformas (integrações) compatíveis com o script deseja sincronizar. Apesar de que tudo isso será explicado mais detalhadamente e demonstrado no restante da documentação, nada vai acontecer de fato, antes que realizemos a instalação das dependências necessárias.

Instalação NodeJs

O Node Js permite a execução de scripts javascript fora do browser. Ele deverá estar instalado em uma máquina que tenha acesso às origens de dados desejadas para que o script **ConecSync** seja executado, baixe o Node Js [aqui](#) e siga as instruções para instalação em sua plataforma.

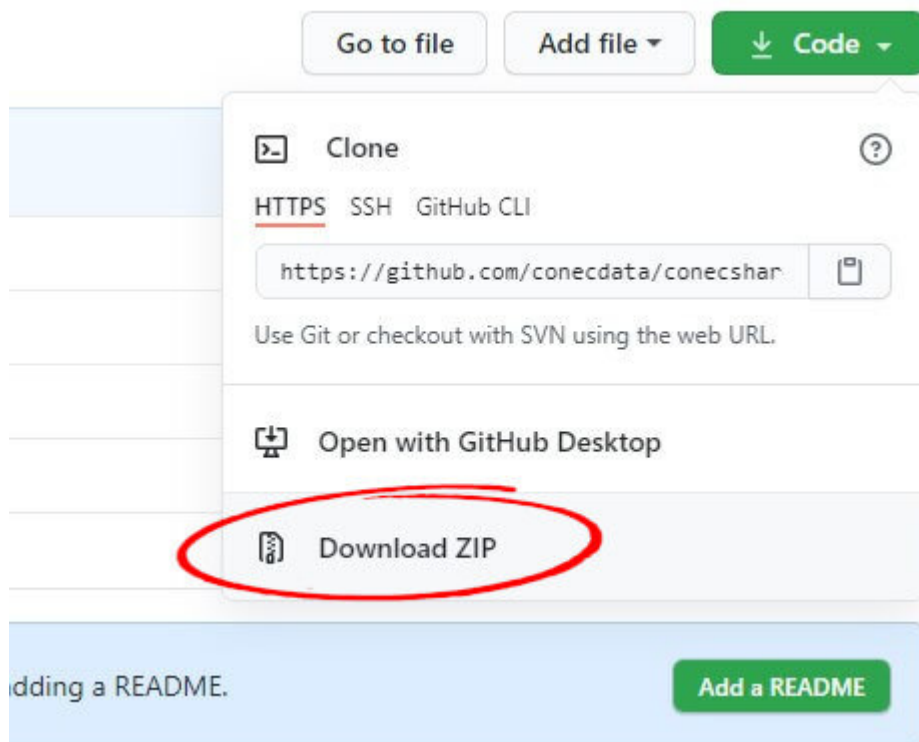
Copiando o script ConecSync

Uma vez instalado o Node Js, agora temos que copiar a pasta do **ConecSync** de seu repositório no github para sua máquina. Existem duas maneiras para fazer isso:

Clonando repositório git Acesse a pasta dentro da qual quer criar a pasta do **ConecSync** (parent), e execute o comando abaixo, para clonar o repositório público (requer o **CLI** do git instalado) do projeto. `git clone https://github.com/conecdata/conecsync`

A pasta **conecsync** será criada automaticamente dentro da pasta em que o comando foi executado. Caso não tenha, saiba ou queira utilizar o cli **git**, utilize o método abaixo.

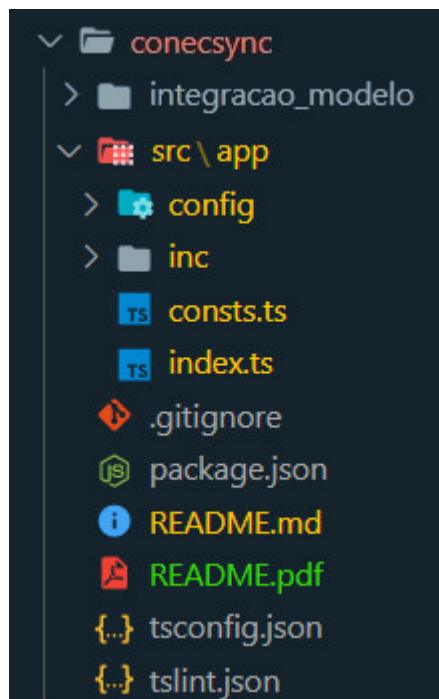
Baixando arquivo zip Acesse o repositório no github por esse [link](#), baixe o arquivo zip pelo link dentro do botão Code e descompacte o arquivo baixado na pasta em que desejada utilizar o **ConecSync**.



Estrutura da pasta do ConecSync

As alterações que você deverá fazer no **ConecSync**, estarão todas dentro da pasta **config**, apesar disso, explicaremos o papel de cada arquivo do projeto.

Você não deve modificar nenhum arquivo fora da pasta **config** sob pena de comprometer o correto funcionamento do script e ter que baixá-lo novamente do repositório para corrigi-lo.



Pasta/arquivo	Descrição
/integracao_modelo	Pasta contendo scripts sql e arquivos csv utilizados nos exemplos da documentação
/src	Pasta contendo códigos fontes e arquivos complementares
/src/app	Pasta contendo códigos fonte
/src/app/config	Pasta contendo os arquivos de configuração que você DEVE modificar
/src/app/inc	Pasta contendo bibliotecas
/src/app/consts.ts	Arquivo com valores gerais diversos
/src/app/index.ts	Arquivo fonte principal do script
/.gitignore	Arquivo de exceções do github (oculto)
/package.json	Arquivo de configuração e dependências do projeto
/README.md	Arquivo contendo esse mesmo documento que você está lendo (formato markdown)
/README.pdf	Arquivo contendo esse mesmo documento que você está lendo (formato PDF)
/tsconfig.json	Arquivo de configuração do typescript
/tslint.json	Arquivo de configuração do typescript

Diversas outras pastas serão criadas posteriormente, o restante da documentação o notificará cada vez que isso acontecer.

Instalando dependências

Uma vez que a pasta do **ConecSync** já tenha sido descompactada ou clonada, entre nela e execute o comando abaixo para instalar suas dependências (pode levar alguns minutos dependendo da velocidade de conexão da sua internet): `npm i` ou `npm install`

Para integração apenas com arquivos csv, as dependências já instaladas pelo comando anterior são suficientes. Entretanto, caso tenha optado por realizar a conexão com um dos gerenciadores de banco de dados compatíveis com o script, é necessário a instalação de uma dependência para esse gerenciador também.

Banco de dados	tipo	comando
MySql	mysql	<code>npm install --save mysql2</code>
Maria DB	mariadb	<code>npm install --save mariadb</code>
Postgres	postgres	<code>npm install --save pg pg-hstore</code>
Microsoft SQL Server	mssql	<code>npm install --save tedious</code>

Mais adiante criaremos um exemplo em MySql para demonstrar a configuração e o funcionamento do script, caso queira acompanhar esse exemplo, execute também o comando correspondente a esse gerenciador na tabela acima.

Pasta das dependências Após o final da instalação das dependências, uma pasta **node_modules** terá sido criada dentro da pasta **conecsync**.

Pasta/arquivo	Descrição
node_modules	Pasta contendo dependências baixadas do projeto

Seguindo a integração de exemplo

Como já vimos, é possível realizar a integração via **ConecSync** por *leitura do banco de dados* ou arquivos [csv](#). Devido a isso, seguiremos um integração exemplo completa para cada uma dessas modalidades.

Os arquivos de exemplo com a estrutura e conteúdo das tabelas utilizados estão disponíveis na pasta **integracao_modelo/db** e os arquivos csv prontos para importação estarão na pasta **integracao_modelo/csv**.

A documentação foi formulada de maneira que você possa acompanhar os exemplos ou utilizá-la como material de consulta. Mesmo que você não queira executar os exemplos passa a passo, é importante analisar a estrutura das tabelas apresentadas, para entender os campos utilizados e como eles são indicados na criação das views, para que sirvam de modelo para criação de suas próprias views (substituindo os campos de exemplo pelos correspondentes em sua base de dados já existente). Os passos necessários para acompanhar o exemplo, sempre estarão destacados como abaixo:

Acompanhando o exemplo DB/CSV

Passo a ser executado no exemplo.

Integração modelo DB

Utilizaremos um banco de dados MySQL para nossa integração de modelo. Presumimos que você já tenha um servidor compatível com esse gerenciador de banco de dados em que a pasta do **ConecSync** esteja copiada.

Mesmo que não esteja usando o MySQL, os passos descritos serão praticamente os mesmos para os demais gerenciadores compatíveis.

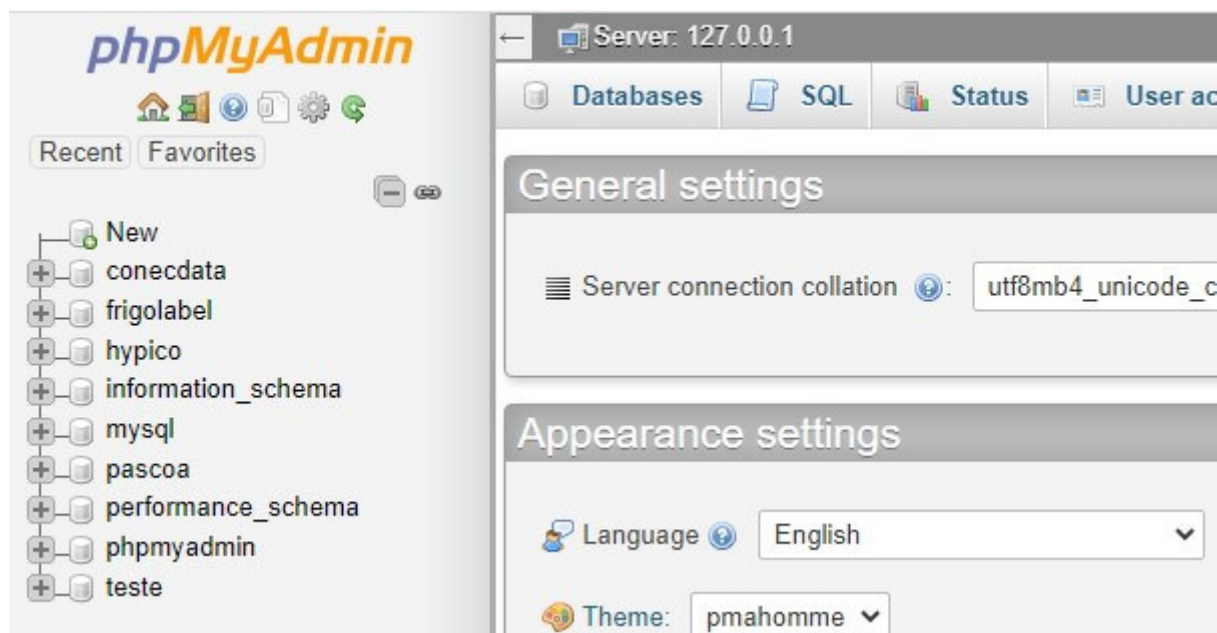
Criando o banco de dados

Utilizaremos a **gui** (interface gráfica) **phpmyadmin** para os exemplos, mas você pode criar as tabelas e preenchê-las da maneira que preferir.

Acompanhando o exemplo DB Com o servidor e o gerenciador Mysql funcionando, basta acionar o **phpmyadmin** em seu browser pelo seguinte comando.

```
localhost/phpmyadmin
```

Para que a interface gráfica funcione corretamente, você pode ter que logar com um usuário e senha válidos do mysql ou estar com essas credenciais gravadas em arquivos de configuração do phpmyadmin. De qualquer maneira essas informações de acesso serão também necessárias para configuração de conexão do script com seu gerenciador de banco de dados e dependem de detalhes da instalação do MySQL no seu sistema.



Se for utilizar o phpmyadmin para executar os comandos ou scripts (múltiplos comandos separados por ponto e vírgula), basta selecionar a guia SQL para acionar a textarea para digitação de scripts.

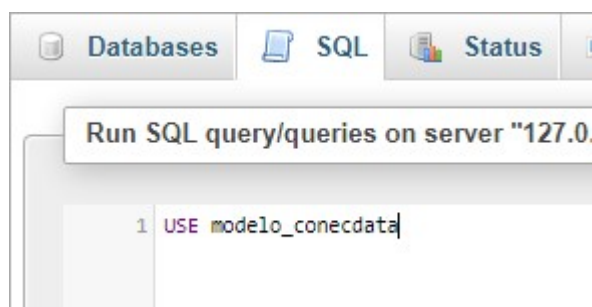
Acompanhando o exemplo DB

Primeiro, vamos criar um novo database chamado **modelo_conecdata** pelo comando abaixo na guia SQL. `CREATE DATABASE modelo_conecdata`



Acompanhando o exemplo DB

Depois execute o comando abaixo para selecionar o database gerado como default, para as próximas operações: `USE modelo_conecdata`



Os scripts de criação das tabelas que serão executados a seguir, preventivamente, já selecionam esse database para uso, mas não custa prevenir fazendo isso aqui também, isso permitirá que você execute outros comandos para fazer testes, fora da execução dos scripts.

Criando as tabelas modelo

Acompanhando o exemplo DB Certifique-se que o database **modelo_conecdata** esteja criado e execute os scripts listados abaixo (em qualquer ordem), presentes na pasta **integracao_modelo/db** do projeto **ConecSync**. Copie seus conteúdos (um por vez) e os cole na textarea da guia **SQL** do **phpmyadmin** conforme já demonstrado anteriormente.

- grupos.sql
- subgrupos.sql
- produtos.sql
- formas_pgto.sql

Note que, em cada script, o database foi selecionado (por garantia), as tabelas foram criadas, seus campos de chave primária e autoincremento definidos e alguns valores lançados. Normalmente, criaríamos também chaves estrangeiras para interligar as tabelas, entretanto, esse passo foi intencionalmente ignorado aqui para tornar o exemplo mais simples, uma vez que os relacionamentos não comprometem em nada o que pretendemos demonstrar.

Caso as chaves estrangeiras fossem definidas nas tabelas, o ideal seria criar todas estruturas primeiro e posteriormente executar os comandos de entradas de dados, uma vez que indicar uma chave correspondente a uma tabela ou linha inexistente, resultaria em um erro.

Tabela grupos

Campo	Tipo	Tamanho	Descrição
gru_pk	chave primária	-	-
gru_b_ativo	boolean	-	Indica se grupo está ativo ou não
gru_c_grupo	string	40	Nome do grupo

Note que os campos indicados como boolean no MySQL, são de fato, números inteiros, cujo valor 0 significa FALSE, e qualquer outro valor significa TRUE.

Tabela subgrupos

Campo	Tipo	Tamanho	Descrição
sub_pk	chave primária	-	-
sub_b_ativo	boolean	-	Indica se subgrupo está ativo ou não
sub_c_subgrupo	string	40	Nome do subgrupo

Tabela produtos

Campo	Tipo	Tamanho	Descrição
pro_pk	chave primária	-	-
pro_fk_grupo	chave estrangeira	-	Chave de relação com gru_pk na tabela grupos
pro_fk_subgrupo	chave estrangeira	-	Chave de relação com sub_pk na tabela subgrupos
pro_b_ativo	boolean	-	Indica se produto está ativo ou não
pro_b_balanca	boolean	-	Indica se produto gera etiqueta para balança ou não (NÃO UTILIZADO)
pro_b_destaque	boolean	-	Indica se produto está com exibição privilegiada ou não
pro_b_estoque	boolean	-	Indica se produto tem estoque controlado ou não
pro_b_favorito	boolean	-	Indica se produto é favorito ou não (NÃO UTILIZADO)
pro_b_fracionado	boolean	-	Indica se produto pode ser vendido a granel ou não
pro_b_industrializado	boolean	-	Indica se produto é de fabricação industrial ou própria
pro_c_barcode	string	20	Código de barras do produto

Campo	Tipo	Tamanho	Descrição
pro_c_descricao	string	60	Descrição do produto
pro_c_img	string	60	Nome do arquivo de imagem do produto (NÃO UTILIZADO)
pro_c_pesavel_tipo	string	5	Tipo de gramatura do produto pesável
pro_c_produto	string	60	Nome do produto
pro_f_perc_limite_venda	decimal	10,2	Percentual do estoque atual base para cálculo de limite de venda online
pro_f_pesavel_fracao	decimal	10,2	Fração de aumento/redução de produtos pesáveis
pro_f_preco	decimal	10,2	Preço de venda do produto
pro_f_qtde_atacado	decimal	10,2	Qtde mínima do produto para ativar preço atacado
pro_f_qtde_estoque_loja	decimal	10,2	Quantidade atualmente em estoque do produto
pro_f_qtde_estoque_min	decimal	10,2	Quantidade mínima de estoque regular na loja
pro_f_qtde_limite_venda	decimal	10,2	Qtde máxima para venda do produto online
pro_f_valor_atacado	decimal	10,2	Preço de venda do produto por atacado
pro_i_cod	integer	10	Código do produto (NÃO UTILIZADO)

Note que apesar das colunas correspondentes às chaves estrangeiras estarem presentes, elas não estão interligadas com as outras tabelas, conforme explicado anteriormente.

Alguns campos indicados como NÃO UTILIZADOS, foram mantidos intencionalmente para demonstrar que nem todos campos presentes na tabela necessitam estar presentes na view. Como proceder no caso inverso, será demonstrado também mais tarde, como lidar com campos que não estejam presentes nas tabelas origem mas são requeridos na view.

Tabela grupos

tabela formas_pgto

Campo	Tipo	Tamanho	Descrição
fpg_pk	chave primária	-	-
fpg_c_forma	string	40	Nome completo da forma de pagamento
fpg_c_legenda	string	40	Nome reduzido da forma de pagamento para acompanhar sua imagem (NÃO UTILIZADO)
fpg_c_id_externo	string	40	Identificador dessa forma de pagamento na lista da Conecdata

Nessa tabela, a coluna de id externo, relaciona suas formas de pagamento com seus correspondentes [nessa lista](#). Apenas formas constantes nessa lista podem ser sincronizadas, caso necessite de novas formas, entre em contato com a Conecdata para que elas sejam implementadas e você tenha novos identificadores para lançar em seu cadastro.

Optamos por utilizar campos com uma nomenclatura estranha (gru_pk, pro_fk_grupo, fpg_c_forma, ...), intencionalmente, para deixar mais claro nos exemplos que virão de criação das [views](#), quais campos pertencem às tabelas origem (e portanto serão substituídos por nomes de campos dos seus próprios cadastros) e quais são os apelidos com nomes específicos (e que devem ser mantidos exatamente com exibidos na view).

Todos nomes de tabelas e também os nomes de campos seguindo essa nomenclatura exótica, utilizados em nossos exemplos, devem ser substituídos pelos correspondentes em sua base de dados, quando for fazer sua integração real.

Criando as views

Além dos motivos já citados anteriormente para a utilização de [views](#), um que é essencial, é a possibilidade de se agrupar diversas tabelas em uma só (JOINS) facilitando sua consulta pelo **ConecSync**, as seguintes views podem ser criadas para servir de origem de dados para sincronização:

As origens de dados geradas pelas views não dependem umas das outras, e mais tarde veremos como podemos indicar quais origens desejamos integrar especificamente. Obviamente, apenas as views das origens de dados que você deseja integrar necessitam ser criadas. Algumas integrações podem ser feitas por arquivos CSV ou até mesmo diretamente por seu código chamando diretamente a api em vez de executar o ConecSync.

View	Origem
view_conecdata_produtos	Tabela de produtos integrada com a de grupos e subgrupos
view_conecdata_estoque	Parte da tabela de produtos correspondente às informações de estoque do produto
view_conecdata_formas	Tabela de formas de pagamento

Note que apesar de estarmos utilizando os termos grupos e subgrupos, os termos correspondentes utilizados nas plataformas que vão receber as modificações são departamentos e subdepartamentos. Os nomes das tabelas foram mantidos divergentes intencionalmente, exatamente para demonstrar que eles podem ser facilmente convertidos para os nomes requeridos pelo **ConecSync**.

A parte mais complexa da integração pelo **ConecSync** provavelmente será a geração das views. Não muito complexa na verdade, uma vez que consistem basicamente em trocar alguns valores nos scripts SQL apresentados como exemplo por outros de sua base de dados, mas explicaremos esse tópico mais detalhadamente para que você saiba como substituir esses valores (tabelas, campos e constantes) corretamente.

view_conecdata_formas

Acompanhando o exemplo DB Para criar a view de baseada em nossa tabela **formas_pgto** exemplo, execute o script abaixo:

```
DROP VIEW IF EXISTS view_conecdata_formas;

CREATE VIEW
    view_conecdata_formas
AS SELECT
    fpg_pk AS idInterno,
    fpg_c_forma AS nomeForma,
    fpg_c_id_externo AS idExterno,
    '1' AS idLoja,
    1 AS formaAtiva
FROM
    formas_pgto
WHERE
    fpg_c_id_externo IS NOT NULL;
```

Esses mesmos scripts de criação das views modelos, constarão como comentários nos arquivos de configuração de origens de dados.

Vamos analisar cada comando da view para entendê-la melhor:

DROP VIEW Esse comando apaga uma eventual versão antiga da view, para que possamos criar uma nova versão, sem problemas.

Como veremos no próximo comando, você pode utilizar o nome que quiser para suas views, e obviamente, o nome da view sendo excluída aqui, deve corresponder ao que vai ser criada pelo comando seguinte.

CREATE VIEW Esse comando indica o nome que a view terá para essa origem de dados, **view_conecdata_formas** em nosso exemplo, você pode substituir esse nome por um de sua preferência, mais tarde veremos onde indicar o nome dessa view na configuração.

AS SELECT É aqui que a mágica acontece, e para entendê-la, precisamos entender o papel de valores à esquerda (antes) de cada comando **AS** e os valores à direita (depois) deles.

- **ESQUERDA/ANTES:** Devem ser campos correspondentes em suas tabelas de origem de dados (que variam de linha para linha no cadastro) ou valores constantes (que valem para todas linhas). **Você sempre DEVE modificar esses nomes por correspondentes na sua base de dados ou por valores constantes.**
- **DIREITA/DEPOIS:** São apelidos de valores requeridos pela plataforma. **Você NÃO DEVE modificar ou omitir qualquer um desses nomes.**

Deixar de indicar algum dos apelidos requeridos na geração da view, causa um erro que interrompe o script de integração.

Três situações possíveis, ocorrerão na composição de suas views:

- Campos na tabela de origem de dados que POSSUEM um valor correspondente a um apelido.

- Campos na tabela de dados que NÃO POSSUEM um valor correspondente a um apelido.
- Apelidos que NÃO POSSUEM campos correspondentes na tabela de origem de dados.

Vamos verificar todos campos de nossa tabela de origem exemplo e todos apelidos listados na view para ver a correspondência entre eles.

Campo	Apelido	Situação
fpg_pk	idInterno	ENCONTRADO
fpg_c_forma	nomeForma	ENCONTRADO
fpg_c_id_externo	idExterno	ENCONTRADO
fpg_c_legenda	-	NÃO UTILIZADO
-	formaAtiva	NÃO ENCONTRADO
-	idLoja	NÃO ENCONTRADO

Caso 1: Campos na tabela origem que correspondem a apelidos (situação **ENCONTRADO**)

```
fpg_pk
fpg_c_forma
fpg_c_id_externo
```

Basta indicar o nome do campo antes do comando AS que o renomeia para o apelido correspondente.

Campos indicados que correspondam a identificadores (chaves primárias ou estrangeiras em SQL e uids em NoSqls) podem ser tanto do tipo número como string.

Caso 2: Campos na tabela origem que NÃO correspondem a apelidos (situação **NÃO UTILIZADO**)

```
fpg_c_legenda
```

Basta ignorar esses campos na composição da view, note que não existe referência a **fpg_c_legenda** no comando de geração da view **view_conecdata_formas** que apresentamos como modelo.

Caso 3: Apelidos na view que NÃO tem campos correspondentes na tabela de origem (situação **NÃO ENCONTRADO**)

```
formaAtiva
idLoja
```

Nesses casos devemos indicar um valor constante no lugar do nome de nosso campo de origem. Em nosso exemplo, o valor constante **1** (number) foi atribuído ao apelido **formaAtiva** (boolean), indicando que TODAS formas de pagamento estão ativas.

Lembre-se que para o MySQL (gerenciador DB utilizado no exemplo), booleanos são números.

Da mesma forma uma constante '1' (string) foi atribuída ao apelido **idLoja** indicando que TODAS essas formas pertencem à loja 1.

Todas lojas que geram origens de dados para integração, devem ter seus tokens de loja indicados (como veremos mais adiante na documentação) em configurações dos projetos utilizados.

Você pode estar estranhando que o **idLoja** seja indicado como string e não número, uma vez que normalmente chaves primárias (identificadores) em bancos de dados relacionais, normalmente, são valores numéricos gerados automaticamente por autoincremento. Na verdade, você pode indicar todos ids para o **ConecSync** como números ou strings, pois eles serão convertidos para strings antes de serem utilizados, o motivo disso é que o script também permite integração com bancos de dados NoSql (normalmente importando seus dados de arquivos CSV), e nesses bancos de dados, normalmente os identificadores são strings geradas aleatoriamente.

E SE eu tivesse uma coluna **fpg_fk_loja** para cada linha da tabela? Bastaria indicar essa coluna no lugar da constante na geração da view e repetir seu comando (lembre-se que ele apaga a view existente com seu primeiro comando). Dessa maneira, cada forma de pagamento poderia corresponder a uma loja específica e não todas à mesma loja como acontece se utilizarmos um valor constante. '1' AS **idLoja** vira **fpg_fk_loja AS idLoja**

E SE eu tivesse uma coluna **fpg_b_ativo** para cada linha da tabela? Da mesma forma, bastaria indicar essa coluna no lugar da constante na view (lembre-se de precisa recriar a view após fazer isso). Dessa maneira, cada forma de pagamento poderia estar ou não ativa, e não todas ativas como acontece se utilizarmos um valor constante. '1' AS **idLoja** vira **fpg_fk_loja AS idLoja**

É claro que os nomes dos campos indicados devem existir nas tabelas, não é o caso do **idLoja**, uma vez que ele não existe na tabela modelo que criamos.

FROM Esse comando simplesmente indica o nome da tabela de origem de dados sendo lida (e obviamente deve ser trocado pelo nome de tabela correspondente em seu cadastro).

WHERE Nesse exemplo específico, esse comando descarta linhas que não tenham um valor correspondente para um identificador em nossa plataforma, uma vez que se fossem passados como parâmetros para a **api**, gerariam erros (obviamente, o nome do campo **fpg_c_id_externo** também deveria ser substituído pelo correspondente em próprio seu cadastro).

view_conecdata_produtos

A **view** produtos é com certeza a mais complexa, os cadastros de produtos de diferentes empresas podem variar bastante entre si, na quantidade e tipo dos campos, e ainda depender ou não, de outros cadastros. Decidimos então, criar essa **view** de maneira progressiva, ou seja, vamos começar com uma versão bastante simples (utilizando constantes no lugar da maioria dos campos presentes em nossas tabelas de exemplo). Posteriormente, vamos implementar mudanças pontuais na view básica, para explicar ações comuns, que envolvem campos que têm relação entre si, e devem ser modificados conjuntamente.

View básica Aproveitando poucos valores da origem de dados, e ignorando os demais (por enquanto).

Acompanhando o exemplo DB Para criar a view de baseada em nossa tabela **produtos** exemplo, execute o script abaixo:

```
DROP VIEW IF EXISTS view_conecdata_produtos;

CREATE VIEW
    view_conecdata_produtos
AS SELECT
    pro_pk AS idProduto,
    pro_c_barcode AS barcodeProduto,
    pro_f_preco AS precoVenda,

    pro_fk_grupo AS idDepartamento,
    gru_c_grupo AS nomeDepartamento,
    gru_b_ativo AS ativoDepartamento,

    0 AS idSubdepartamento,
    '' AS nomeSubdepartamento,
    0 AS ativoSubdepartamento,

    1 AS industrializado,
    '' AS nomeProduto,

    0 AS estoqueControlado,
    0 AS qtdeEstoqueMinimo,
    0 AS qtdeEstoqueAtual,

    0 AS atacadoQtde,
    0 AS atacadoValor,

    0 AS percentualLimiteVenda,
    0 AS qtdeLimiteVenda,

    0 AS pesavelStatus,
    0 AS pesavelFracao,
    '' AS pesavelTipo,

    1 AS produtoAtivo,

    '' AS descricaoProduto,

    0 AS destaque,

    '1' AS idLoja
FROM
    produtos
LEFT JOIN
    grupos AS departamentos ON produtos.pro_fk_grupo = departamentos.gru_pk;
```

Nessa versão, praticamente passamos apenas as informações necessárias para uma sincronização básica, com poucos campos do cadastro e muitas constantes. Não indicamos nenhum subdepartamento para os

produtos, indicamos todos para uma mesma loja, todos ativos, todos industrializados, todos SEM NOME, todos sem controle de estoque, todos sem promoção atacado, todos sem serem pesáveis, todos sem limite venda, todos sem descrição, etc...

Observe que nem mesmo os nomes dos produtos presentes no cadastro foram indicados (uma string vazia foi indicada em seu lugar), nesse exemplo isso funcionará, devido a todos produtos terem sido indicados como industrializados. Produtos industrializados nas plataformas compatíveis, têm seu barcode usado como referência para acessar uma base de dados da Conecdata e buscar os nomes e as imagens desses produtos.

Já fomos apresentados aos comandos que compõem a [view](#) no exemplo anterior, mas dessa vez existe um novo comando o LEFT JOIN, que, nesse exemplo, liga a tabela produtos com a de grupos (inclusive, renomeando-a para departamentos, o que na verdade nem é necessário) e permitindo dessa forma que campos de ambas componham o SELECT, gerando um resultado combinado das duas. No caso, os campos **gru_c_grupo** e **gru_b_ativo** não existem na tabela produtos, mas por sua chave de interligação com grupos **pro_fk_grupo** (veja que ele consta no comando JOIN junto da chave primária da tabela grupos **gru_pk**) foi possível utilizá-los como parte do SELECT final que queremos.

A partir de agora, vamos implementar algumas modificações conjuntas na geração da [view](#), cada uma com uma finalidade específica.

Incluindo subdepartamentos Em nossa versão simplificada da view produtos, todos produtos indicam apenas seus departamentos (aqui chamados de grupos) e todos subdepartamentos (ou subgrupos, temos até uma tabela para eles) ignorados, pois no lugar de seus campos, foram indicados strings vazias, indicando dessa forma, que esses produtos devem ser atribuídos diretamente a um departamento e não a um subdepartamento dentro deles. Agora vamos implementar as mudanças necessárias para que os produtos possam, cada um deles, indicar também um subdepartamento.

Acompanhando o exemplo DB Substitua as constantes do seguinte trecho:

```
...
0 AS idSubdepartamento,
'' AS nomeSubdepartamento,
0 AS ativoSubdepartamento,
...
```

Por seus nomes de campos presentes nas tabelas:

```
...
pro_fk_subgrupo AS idSubdepartamento,
sub_c_subgrupo AS nomeSubdepartamento,
sub_b_ativo AS ativoSubdepartamento,
...
```

E execute o novo comando, para substituir a view.

Ao executar o comando, provavelmente ocorrerá um erro, o problema é que, além de campos da tabela produtos (que está indicada na composição da view), estamos também indicando campos da tabela **subgrupos** e nada, até então, faz referência a ela em nossa nova view. Para resolvermos isso temos que incluir um segundo comando JOIN unindo a tabela produtos também com a de subgrupos, o que permitirá que também possamos referenciar seus campos no select da view.

Acompanhando o exemplo DB Para corrigir o problema, logo abaixo do trecho da view atual:

```
FROM
  produtos
LEFT JOIN
  grupos AS departamentos ON produtos.pro_fk_grupo = departamentos.gru_pk
```

Inclua um segundo comando JOIN:

```
LEFT JOIN
  subgrupos ON produtos.pro_fk_grupo = subgrupos.gru_pk
```

E execute o novo comando de substituir a view. Dessa vez o comando deve ser executado com sucesso.

Note que dessa vez, não fizemos como no JOIN anterior, renomeando a tabela subgrupos para subdepartamentos, já havíamos citado anteriormente que esse não era um passo necessário, dessa forma, o comando abaixo surtiria o mesmo efeito do acima.

NÃO É NECESSÁRIO EXECUTAR ESSE COMANDO

```
LEFT JOIN
  subgrupos AS subdepartamentos ON produtos.pro_fk_grupo =
  subdepartamentos.gru_pk
```

Com essas mudanças, os valores constantes no campo **pro_fk_subgrupo** de cada produto, são utilizados para indicar a qual linha da tabela **subgrupos** eles correspondem. Agora, com as tabelas devidamente interligadas, valores de ambas podem ser utilizados para composição da view, no caso **sub_c_subgrupo** e **sub_b_ativo** vieram da tabela subgrupos e não da de produtos.

Indicando produtos não industrializados Já foi citado anteriormente que produtos indicados como industrializados, têm seu nome e imagens buscados pela plataforma em uma base própria dela, dessa forma pudemos omitir o nome do produto na versão inicial da view, uma vez que ele seria ignorado de qualquer forma. Agora, caso indiquemos em cada produto, se ele é ou não industrializado, os que indicarem esse valor como FALSE (tornando-os digamos "manufaturados"), devem passar também seus nomes para plataforma, pois produtos não industrializados passados sem nome para as apis, resultam em erros.

Acompanhando o exemplo DB Para que possamos indicar para cada produto quais são industrializados e quais não, basta modificar o trecho abaixo da [view](#).

```
1 AS industrializado,  
' ' AS nomeProduto,
```

Por este:

```
pro_b_industrializado AS industrializado,  
pro_c_produto AS nomeProduto,
```

E execute o novo comando, para substituir a view.

Como todos campos de tabelas indicados já estão presentes na tabela produtos, que é a tabela base da view, nenhum outro comando JOIN é necessário.

Controlando o estoque dos produtos Se você analisar a documentação das apis dos projetos compatíveis com o **ConecSync**, perceberá que ele não possui um valor para indicação de estoque atual do produto (para ser atualizada a cada venda do produto), pois essas apis pretendem ser comunicadas apenas sobre modificações eventuais dos produtos, como modificações de preço, departamentos, status ativo, etc... Entretanto, existe uma flag **estoqueMinimo** relacionada ao controle de estoque nelas, quando esse valor é passado como TRUE, isso quer dizer que o produto se encontra com quantidade crítica de estoque e nesse caso, sua venda online deve ser suspensa, o inverso ocorre caso seu valor seja indicado como FALSE, ou seja, a venda online volta a ser liberada, pois o produto já não se encontra mais com estoque crítico. É claro que você poderia controlar isso via api em seu próprio código, entretanto para que a integração seja aprovada, tem que controlar acionamentos às [APIs](#) apenas quando esses valores sofrerem modificações (ou seja, não deve acionar a api a cada venda que reduza a quantidade do produto e cuja situação de estoque crítico já tenha sido comunicada em chamadas anteriores, nem quando uma nova entrada de estoque de um produto com estoque regular não tenha entrado no estado crítico nesse cálculo) sob pena da integração ser recusada. Caso queira controlar a flag estoqueMinimo de em seu código, deve solicitar maiores informações à [Conecdata](#) da forma correta de fazê-lo, felizmente é muito mais fácil e seguro você controlar o status da venda online dos produtos, usando o **ConecSync** como demonstraremos a seguir.

Acompanhando o exemplo DB Seguindo os exemplos anteriores, basta substituir o trecho da view:

```
Ø AS estoqueControlado,  
Ø AS qtdeEstoqueMinimo,  
Ø AS qtdeEstoqueAtual
```

Por este:

```
pro_b_estoque AS estoqueControlado,  
pro_f_qtde_estoque_min AS qtdeEstoqueMinimo,  
pro_f_qtde_estoque_loja AS qtdeEstoqueAtual
```

E executar o novo comando, para substituir a view.

Nenhum desses valores é de fato, repassado para as chamadas das apis para sincronização, eles são apenas utilizados para calcular o status da flag **estoqueMinimo** de acordo com as seguintes regras:

estoqueControlado	estoqueMinimo resultante	Explicação
FALSE	FALSE	Se o estoque desse produto não é controlado, sua venda online nunca será suspensa devido à situação do estoque (já que ela não é controlada).
TRUE	qtdeEstoqueAtual < qtdeEstoqueMinimo	A venda online será suspensa caso a qtde atual do estoque esteja abaixo do valor mínimo indicado e reativada em caso contrário.

A comunicação para as apis só acontecem, caso algum valor relevante do cadastro de produtos (incluindo estoqueMinimo) esteja diferente do que foi emitido anteriormente.

Existe uma origem de dados "estoque" para ser utilizada no lugar da origem "produtos", caso você não use o **ConecSync** para sincronizar os produtos (fazendo-o por exemplo por chamadas às apis de seu código), mas queira continuar utilizando o script para monitorar o estoque dos produtos. Caso já sincronize a origem produtos, mesmo que a origem estoque seja indicada, ela será ignorada, uma vez que o estoque já foi controlado pela integração dos produtos.

Produtos com venda por atacado

Acompanhando o exemplo DB Para que você possa indicar uma quantidade para cada produto, que reduza seu preço quando essa quantidade for atingida, basta substituir o trecho da view:

```
0 AS atacadoQtde,
0 AS atacadoValor,
```

Por esse:

```
pro_f_qtde_atacado AS atacadoQtde,
pro_f_valor_atacado AS atacadoValor,
```

E executar o novo comando, para substituir a view.

Produtos com limite de quantidade de venda Dois campos podem controlar a quantidade disponível para venda online de um produto:

- **pro_f_perc_limite_venda** Só é utilizada caso o estoque desse produto seja controlado (campo **pro_b_estoque** em nosso exemplo). O valor percentual aqui indicado, é calculado sobre o valor do campo **pro_f_qtde_estoque_loja**, gerando o valor limite para venda online desse produto. Vejamos alguns exemplos:

pro_b_estoque	pro_f_perc_limite_venda	pro_f_qtde_estoque_loja	Resultado	Motivo
----------------------	--------------------------------	--------------------------------	------------------	---------------

pro_b_estoque	pro_f_perc_limite_venda	pro_f_qtde_estoque_loja	Resultado	Motivo
FALSE	10	200	0 (sem limite)	Se estoque não é controlado, não há limite para venda online
TRUE	10	200	20 no máx	$200 * 10\% = 20$

- **pro_f_qtde_limite_venda** Indicação direta de valor mínimo para venda online. Caso **pro_f_perc_limite_venda** resulte em algum valor (> 0), ele é comparado com **pro_f_qtde_limite_venda** e o MENOR deles é repassado para as apis para ser utilizado na limitação da quantidade de venda online do produto.

Acompanhando o exemplo DB Para definir um limite para venda online de cada produto baseado no estoque atual, basta substituir o trecho da view:

```
0 AS percentualLimiteVenda
```

Por esse:

```
pro_f_perc_limite_venda AS percentualLimiteVenda
```

E executar o novo comando, para substituir a view. Lembre-se que esse valor só tem efeito caso esse produto tenha a flag de estoque controlado habilitada.

Acompanhando o exemplo DB Para definir um limite de quantidade de venda fixo para cada produto, basta substituir o trecho da view:

```
0 AS qtdeLimiteVenda,
```

Por esse:

```
pro_f_qtde_limite_venda AS qtdeLimiteVenda,
```

E executar o novo comando, para substituir a view.

O menor desses dois valores (o baseado no percentual e o fixo) será passado para API como sendo o limite de venda desse produto por pedido.

Produtos pesáveis

Acompanhando o exemplo DB Caso indique um produto como pesável (venda a granel), tem também que indicar dois outros valores, o tipo de sua medida e a fração para seu acréscimo/decrécimo. Basta substituir o trecho da view:

```
0 AS pesavelStatus,  
0 AS pesavelFracao,  
' ' AS pesavelTipo,
```

Por esse:

```
pro_b_fracionado AS pesavelStatus,  
pro_f_pesavel_fracao AS pesavelFracao,  
pro_c_pesavel_tipo AS pesavelTipo,
```

E executar o novo comando, para substituir a view. Tipos de unidade de medida válidas: (K)ilograma, (G)rama, (L)itro, (ML) mililitro, (M)etro ou (CM) centímetro.

View completa Os demais valores não possuem outros relacionados e seus os demais campos que podem substituir suas constantes podem ser observados na versão completa da view produtos abaixo.

```
DROP VIEW IF EXISTS view_conecdata_produtos;  
  
CREATE VIEW  
view_conecdata_produtos  
AS SELECT  
pro_pk AS idProduto,  
pro_c_barcode AS barcodeProduto,  
pro_f_preco AS precoVenda,  
  
pro_fk_grupo AS idDepartamento,  
gru_c_grupo AS nomeDepartamento,  
gru_b_ativo AS ativoDepartamento,  
  
pro_fk_subgrupo AS idSubdepartamento,  
sub_c_subgrupo AS nomeSubdepartamento,  
sub_b_ativo AS ativoSubdepartamento,  
  
pro_b_industrializado AS industrializado,  
pro_c_produto AS nomeProduto,  
  
pro_b_estoque AS estoqueControlado,  
pro_f_qtde_estoque_min AS qtdeEstoqueMinimo,  
pro_f_qtde_estoque_loja AS qtdeEstoqueAtual,  
  
pro_f_qtde_atacado AS atacadoQtde,  
pro_f_valor_atacado AS atacadoValor,  
  
pro_f_perc_limite_venda AS percentualLimiteVenda,
```



```

    pro_f_qtde_limite_venda AS qtdeLimiteVenda,

    pro_b_fracionado AS pesavelStatus,
    pro_f_pesavel_fracao AS pesavelFracao,
    pro_c_pesavel_tipo AS pesavelTipo,

    pro_b_ativo AS produtoAtivo,

    pro_c_descricao AS descricaoProduto,

    pro_b_destaque AS destaque,

    '1' AS idLoja
FROM
    produtos
LEFT JOIN
    grupos AS departamentos ON produtos.pro_fk_grupo = departamentos.gru_pk
LEFT JOIN
    subgrupos AS subdepartamentos ON produtos.pro_fk_subgrupo =
subdepartamentos.sub_pk;

```

Como nosso exemplo, a tabela produtos não possui uma coluna (pro_fk_loja ou similar) indicando a qual loja pertence cada produto, mantivemos a constante (representando uma eventual loja 1), uma vez que todos apelidos devem ser indicados.

Campos sem prefixos da tabela Caso sua nomenclatura de campos não referencie a qual tabela pertencem (tipo com os prefixos pro_, gru_, sub_ e fpg_ em nossos exemplos), algumas modificações na composição das view serão necessárias quando ela precisar agrupar (via join) duas (ou mais) tabelas. Vamos supor duas tabelas simples nessa situação e como ficaria a view gerada corretamente a partir delas:

Tabela Grupos

Campo	Tipo	Tamanho	Descrição
id	chave primária	-	-
ativo	boolean	-	Indica se grupo está ativo ou não
nome	string	40	Nome do grupo

Tabela Produtos

Campo	Tipo	Tamanho	Descrição
id	chave primária	-	-
idGrupo	chave estrangeira	-	Campo de ligação com a tabela grupos
ativo	boolean	-	Indica se produto está ativo ou não
nome	string	40	Nome do produto

NÃO É NECESSÁRIO EXECUTAR O RESTANTE DE COMANDOS DESSE TÓPICO

Uma visão reduzida da view dessas tabelas também reduzidas ficaria mais ou menos da seguinte maneira

VERSÃO COM PROBLEMAS:

```
DROP VIEW IF EXISTS view_conecdata_produtos;

CREATE VIEW
    view_conecdata_produtos
AS SELECT
    id AS idProduto,
    produto AS nomeProduto,
    ativo AS produtoAtivo,

    id AS idDepartamento,
    nome AS nomeDepartamento,
    ativo AS ativoDepartamento,

    '1' AS idLoja
FROM
    produtos
LEFT JOIN
    grupos ON idGrupo = id;
```

Esse comando é claramente confuso, uma vez que a maioria dos campos das duas tabelas tem exatamente os mesmos nomes (coisa que os prefixos evitariam), a criação dessa view deveria ser corrigida para o seguinte

VERSÃO SEM PROBLEMAS:

```
DROP VIEW IF EXISTS view_conecdata_produtos;

CREATE VIEW
    view_conecdata_produtos
AS SELECT
    produtos.id AS idProduto,
    produtos.produto AS nomeProduto,
    produtos.ativo AS produtoAtivo,

    grupos.id AS idDepartamento,
    grupos.nome AS nomeDepartamento,
    grupos.ativo AS ativoDepartamento,

    '1' AS idLoja
FROM
    produtos
LEFT JOIN
    grupos ON idGrupo = produtos.id;
```

Note que todas as vezes que os nomes dos campos conflitaram entre alguma das tabelas componentes da view, foram prefixados pelo nome da tabela para eliminar qualquer confusão.

view_conecdata_estoque

A [view](#) de estoque é uma versão extremamente simplificada da produtos e só deve ser utilizada em sua ausência na integração via **ConecSync** (uma vez que a view produtos além de sincronizar os produtos, já sincroniza seu estoque também). Caso tanto a integração de produtos quanto a de estoque seja configurada, a de produtos será executada e a de estoque ignorada.

Acompanhando o exemplo DB Para criar a view de estoque, execute o seguinte comando:

```
DROP VIEW IF EXISTS view_conecdata_estoque;

CREATE VIEW
  view_conecdata_estoque
AS SELECT
  pro_pk AS idProduto,
  pro_b_estoque AS estoqueControlado,
  pro_c_barcode AS barcodeProduto,
  pro_c_produto AS nomeProduto,
  '1' AS idLoja,
  pro_f_qtde_estoque_min AS qtdeEstoqueMinimo,
  pro_f_qtde_estoque_loja AS qtdeEstoqueAtual
FROM
  produtos
WHERE
  pro_b_estoque > 0;
```

Na verdade, os campos `pro_c_barcode` e `pro_c_produto` não seriam necessários na composição da view (uma vez que a api não os utiliza) e apenas foram incluídos para facilitar a identificação de cada produto na view resultante.

Configurando modo DB

Tendo seguido ou não os exemplos apresentados, você agora deve estar apto a criar as views para as tabelas que serão integradas, agora bastam algumas configurações e todo processo estará completo e pronto para rodar.

Configuração geral

Como vimos, criamos um database (repositório de banco de dados) e diversas tabelas e views dentro dele. Agora temos que indicar esse database, bem como, diversas outras informações de conexão para o **ConecSync**.

[config/config.ts](#)

```
export const CONFIG = {
  db: {
    conexao: {
      host: 'localhost',
      tabela: 'modelo_conecdata',
```

```

    usuario: 'root',
    senha: 'senhasecreta',
    tipo: 'mysql', /* 'mysql' | 'mariadb' | 'postgres' | 'mssql' */
  /*
  $ npm install --save pg pg-hstore # Postgres
  $ npm install --save mysql2
  $ npm install --save mariadb
  $ npm install --save sqlite3
  $ npm install --save tedious # Microsoft SQL Server
  */
},
},
csv: {
  path: 'C:\\csvs'
},
/*
TRUE = plataforma de testes
FALSE = plataforma definitiva ( CUIDADO )
*/
sandbox: true,
/*
TRUE = Envia mensagens para terminal (se disponível)
FALSE = Não envia mensagens, apenas grava em arquivos de log
*/
verbose: true
}

```

Observe que os arquivos de configuração estão repletos de comentários explicando suas propriedades.

Propriedade	Descrição	Requerido
db.conexao	Credenciais de acesso a seu database de origem	Apenas se houver alguma origem de dados tipo db
csv	Pasta contendo arquivos csv de origem	Apenas se houver alguma origem de dados do tipo csv
sandbox	Flag alternando entre modo desenvolvimento/produção	Sim
verbose	Habilita exibição de comandos no terminal (se disponível)	Sim

Quando você se habilita para integração, receberá um token de loja sandbox (modo teste), posteriormente, com sua integração aprovada, você passará a receber tokens de lojas do modo produção (que realmente modificam os valores de lojas reais em funcionamento na plataforma utilizando seu ERP). Essa flag, deve refletir o tipo de token que você vai indicar na configuração específica que será vista mais adiante na documentação, uma vez que os de um modo não funcionam para outro.

Acompanhando o exemplo DB Modifique os seguintes valores em **config.ts**:

- Em *db.conexao*, defina a propriedade *db.conexao.tabela* para **modelo_conecdata** (que nós criamos em nosso exemplo) e *db.conexao.tipo* para **mysql**, os demais valores da conexão dependerão de detalhes da instalação de seu gerenciador MySQL (ou outro).
- Deixe a propriedade *csv* vazia, para desativar a integração via arquivos csv (um exemplo desse modo será explicado mais adiante).
- Defina a propriedade *sandbox* para **TRUE**, indicando estarmos utilizando uma plataforma de testes (depende do tipo de token que você tem para indicar mais adiante).
- Defina a propriedade *verbose* para **TRUE**, enviando os resultados e erros também para tela do terminal (mensagens e erros sempre serão gravados em arquivos para esse fim como veremos em breve).

Configurando origens

Dentro da pasta **config/origens**, cada arquivo representa uma origem de dados disponível que pode ser configurada individualmente.

- *config/origens/config-produtos.ts*
- *config/origens/config-estoque.ts*
- *config/origens/config-formas-pgto.ts*

Todos arquivos dessa pasta possuem um formato similar a esse:

```
// Origens de dados podem ser "views no DB" ou "paths de arquivos CSV"
export const CONFIG_[NOME_ORIGEM] = {
  /* Tipo de origem */
  // Se '' ignora essa origem de dados (não sincroniza).
  tipo: '', // 'db' | 'csv' | ''

  // Nome da view do cadastro de produtos
  nomeView: '', // db
}
// Aqui vem um exemplo de criação da view dessa [ORIGEM DE DADOS]
```

Acompanhando o exemplo DB Como já vimos anteriormente, quando a integração de *produtos* é configurada, a de *estoque* se torna redundante e desnecessária, logo:

- No arquivo *config-produtos.ts*, defina a propriedade *tipo* para **db** e o *nomeView* para **view_conecdata_produtos**.
- No arquivo *config-formas-pgto.ts*, defina a propriedade *tipo* para **db** e o *nomeView* para **view_conecdata_formas**.
- No arquivo *config-produtos.ts*, defina as propriedades *tipo* e *nomeView* para **strings vazias** (mesmo que as propriedades fossem indicadas, elas seriam ignoradas uma vez que a origem produtos tem prioridade mais alta, se presente).

Configurando projetos

Agora que já indicamos as configurações gerais de conexão com o cadastro, já indicamos para cada origem possível, qual view deve ser utilizada (e por omissão, quais não queremos integrar), falta indicar em quais

projetos e em quais lojas dentro deles, as modificações detectadas devem ser aplicadas, e como sabemos que isso vai ser feito chamando-se a api de cada projeto, vamos precisar indicar os tokens dessas lojas, pois como vimos, são eles que nos permitem esse tipo de acesso a elas.

Dentro da pasta **config/projetos**, cada arquivo representa um projeto compatível com a integração via **ConecSync** e pode ser configurado individualmente.

- config/projetos/config-mercadeiro.ts

Como já foi dito anteriormente, o Mercadeiro é o primeiro projeto da Conecdata compatível com o ConecSync, e à medida que surgirem outros, seus arquivos de configuração serão incluídos na lista acima.

Todos arquivos dessa pasta possuem um formato similar a esse:

```
export const CONFIG_[NOME_PROJETO] = {
  lojas: [
    /*
     * Você deve indicar quantas lojas quiser sincronizar conforme o exemplo abaixo:
     */
    {
      id: ID_LOJA_NO_SEU_CADASTRO,
      token: TOKEN_PROJETO_LOJA_CORRESPONDENTE
    },
    ...
    OBS: Uma array vazia ignora sincronização com esse projeto.
    /*
     */
    {
      id: '1',
      token: 'ZX1KaGJHY2lPaUpJVXpJMU5pSXNJUN0Ylo2WD1tQ3NzVUNhYUM1Yk9ZbngxMW5v...'
    }
  ]
}
```

Conforme explicado pelos comentários, para ignorar um projeto, basta deixar sua array de lojas vazia.

Acompanhando o exemplo DB Ao iniciar o processo de integração com a conecdata você terá recebido um token de loja para testes.

Indique na array *lojas* o id da loja em seu cadastro junto desse token para relacionar as origens que serão lidas dela, com a loja na plataforma que ele representa.

Posteriormente, após todos os testes realizados, quando receber tokens de lojas de produção, basta relacioná-los aqui, juntamente com os ids de suas respectivas lojas e mudar a flag *sandbox* em *config.ts* para FALSE.

Bom, é isso, nosso exemplo de banco de dados foi criado, geramos suas tabelas, views, configuramos o **ConecSync** corretamente, e agora repetiremos todo processo de exemplo, só que dessa vez para arquivos .csv. Antes de aprendermos como eles funcionam, é recomendável testarmos o que foi feito até agora, para isso, salte para o tópico Testando sua integração, e depois volte e continue no tópico abaixo caso se interesse em aprender a lidar com a sincronização via arquivos, que permitirão que você utilize gerenciadores SQL não

compatíveis com o **ConecSync** bem como, bancos de dados NoSql, cujos cadastros ele não lê diretamente (pelo menos por enquanto).

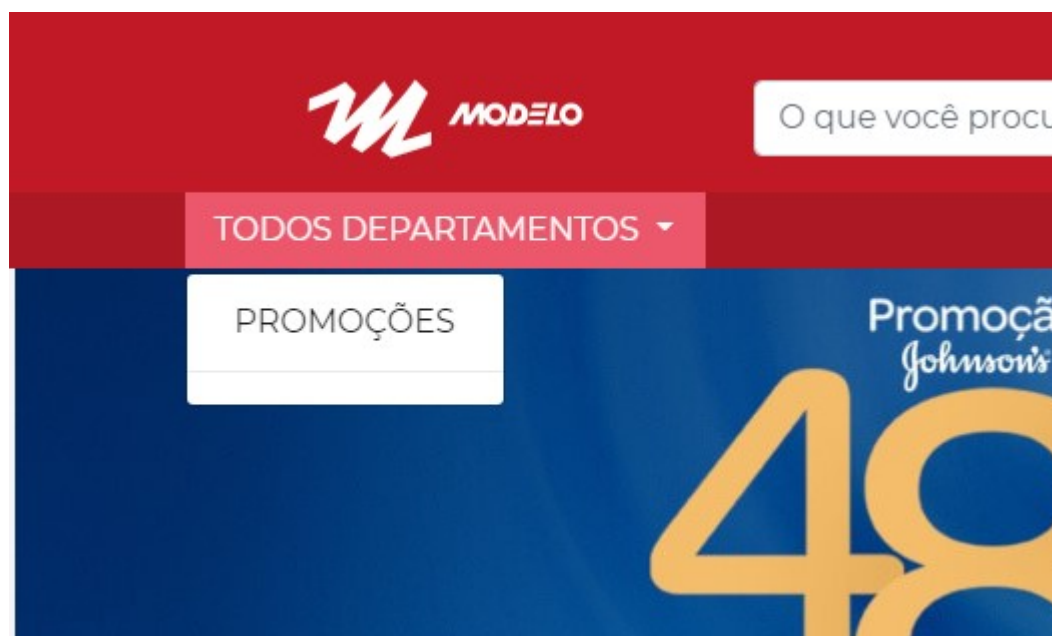
Integração modelo CSV

Testando sua integração

Ao iniciar o processo de integração de seu [ERP](#) com a [Conecdata](#), você terá acesso a uma loja exclusiva na versão sandbox de cada plataforma de testes que queira integrar. Até aqui, aprendemos a configurar as origens de dados e as plataformas que desejamos, agora falta ver isso na prática, vamos lá.

Vamos utilizar a loja *Modelo* do projeto *Mercadeiro* como exemplo, você deve utilizar a loja e a plataforma a que tem acesso e que tenha colocado nas configurações.

Acesse [a versão sandbox do site Mercadeiro](#) e vá até a loja que esteja usando como teste de integração.



Como você pode ver, inicialmente, não temos nenhum departamento (e consequente nenhum subdepartamento ou produto) na loja.

No ambiente de testes (modo sandbox), de posse do token de loja, você pode resetar qualquer origem de dados (eliminar todos dados relativos à essa origem) em sua loja de testes para praticar a integração. Para ver como fazer isso, procure o endpoint *reset* na documentação da api da plataforma que esteja integrando (Mercadeiro no nosso exemplo).

Para executar o **ConecSync**, basta entrar na pasta do projeto e executar o seguinte comando: `npm start`

Quando está rodando no terminal e com a flag *verbose* TRUE no arquivo *config.ts*, os resultados dos comandos (sucessos ou erros) serão exibidos no terminal.


```

$ npm start

> conecsync@1.0.0 start D:\conecdata\conecsync
> tsc && node dist/index.js

2020-11-10 17:27:54 Verificando configurações de lojas em integrações: {"mercadeiro":1}
2020-11-10 17:27:54 Verificando configurações de pasta csv.
2020-11-10 17:27:54 Nenhuma integração csv encontrada
2020-11-10 17:27:54 Verificando configurações de conexão com banco de dados.
(node:7868) ExperimentalWarning: The fs.promises API is experimental
Executing (default): SELECT 1+1 AS result
2020-11-10 17:27:54 Conexão com banco de dados estabelecida com sucesso.
2020-11-10 17:27:54 Verificando integração PRODUTOS.
2020-11-10 17:27:54 Encontrado: view_conecdata_produtos
2020-11-10 17:27:54 Buscando produtos do DB.
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT `idProduto`, `idLoja`, `estoqueControlado`, `barcodeProduto`, `i
alor`, `descricaoProduto`, `industrializado`, `idSubdepartamento`, `nomeSubdepartamento`, `a
ercentualLimiteVenda`, `qtdeLimiteVenda` FROM `view_conecdata_produtos` AS `Produto` WHERE `
2020-11-10 17:27:54 Buscando departamentos e subdepartamentos.
2020-11-10 17:27:54 4 departamento(s) encontrado(s).
2020-11-10 17:27:54 Sincronizando departamentos.
2020-11-10 17:27:57 0 subdepartamento(s) encontrado(s).
2020-11-10 17:27:57 30 produto(s) encontrado(s).
2020-11-10 17:27:57 Sincronizando produtos.
2020-11-10 17:28:16 {"departamentos":{"total":4,"sincronizados":4},"subdepartamentos":{"tot
2020-11-10 17:28:16 Verificando integração FORMAS PGTO.
2020-11-10 17:28:16 Encontrado: view_conecdata_formas
2020-11-10 17:28:16 Buscando formas pgto do DB.
Executing (default): CREATE TABLE IF NOT EXISTS `view_conecdata_produtos` (`idProduto` VARCH
`nomeDepartamento` VARCHAR(255), `ativoDepartamento` TINYINT(1), `nomeProduto` VARCHAR(255)
55), `industrializado` TINYINT(1), `idSubdepartamento` VARCHAR(255), `nomeSubdepartamento` V
), `destaque` TINYINT(1), `qtdeEstoqueMinimo` DECIMAL, `qtdeEstoqueAtual` DECIMAL, `percentu
Executing (default): SHOW INDEX FROM `view_conecdata_produtos`
Executing (default): SELECT `idInterno`, `idExterno`, `nomeForma`, `idLoja` FROM `view_conec
2020-11-10 17:28:16 21 formas(s) pgto encontrada(s).
2020-11-10 17:28:16 Sincronizando formas pgto.
2020-11-10 17:28:30 {"formas":{"total":21,"sincronizados":21}}
2020-11-10 17:28:30 INTEGRAÇÃO DE PRODUTOS ENCONTRADA. IGNORANDO INTEGRAÇÃO DE ESTOQUE: cor
2020-11-10 17:28:30 Verificando integração ESTOQUE.
2020-11-10 17:28:30 {"produtos":{"total":30,"sincronizados":30},"departamentos":{"total":4,
":0,"sincronizados":0}}

```

A exibição dos comandos no terminal é opcional, entretanto todo resultado do processamento de uma execução do script SEMPRE é gravado nos seguintes arquivos.

Arquivo	Descrição
ok.log	Arquivo contendo resultados da execução do script.
errors.log	Arquivo contendo erros ocorridos durante execução do script.

ok.log

```
2020-10-30 18:46:36 Verificando configurações de lojas em integrações: {"mercadeiro":1}
2020-10-30 18:46:36 Verificando configurações de pasta csv.
2020-10-30 18:46:36 Encontrado: D:\conecdata\produtos\conecsync\src\assets
2020-10-30 18:46:36 Verificando configurações de conexão com banco de dados.
2020-10-30 18:46:37 Conexão com banco de dados estabelecida com sucesso.
2020-10-30 18:46:37 Buscando produtos do DB.
2020-10-30 18:46:37 Encontrado: conecdata_produtos
2020-10-30 18:46:37 Verificando integração PRODUTOS.
2020-10-30 18:46:37 {"departamentos":{"total":0,"sincronizados":0},"subdepartamentos":
{"total":0,"sincronizados":0},"produtos":{"total":0,"sincronizados":0}}
2020-10-30 18:46:37 Verificando integração FORMAS PGTO.
2020-10-30 18:46:37 Lendo D:\conecdata\produtos\conecsync\src\assets\formas-pgto\1.csv
2020-10-30 18:46:37 Removendo linhas vazias ou comentadas.
2020-10-30 18:46:37 Validando campos obrigatórios.
2020-10-30 18:46:37 21 forma(s) pgto encontrada(s).
2020-10-30 18:46:37 Verificando largura das linhas.
2020-10-30 18:46:37 Convertendo linhas texto para formas pgto.
2020-10-30 18:46:37 21 formas(s) pgto encontrada(s).
2020-10-30 18:46:37 Sincronizando formas pgto.
2020-10-30 18:46:57 {"formas":{"total":21,"sincronizados":21}}
```

errors.log

```
2020-10-30 18:46:37 Loja 1: Unknown column 'idProduto' in 'field list'
2020-11-10 17:04:21 Loja 1: Unknown column 'vitrine' in 'field list'
2020-11-10 17:04:21 Loja 1: Table 'modelo_conecdata.formas' doesn't exist
2020-11-10 17:06:58 Produto 416: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
2020-11-10 17:06:58 Produto 1004: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
2020-11-10 17:06:58 Produto 1065: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
2020-11-10 17:06:59 Produto 1423: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
2020-11-10 17:06:59 Produto 1757: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
2020-11-10 17:07:00 Produto 1761: 400 - {"erros":{"idDepartamento":"Departamento não
encontrado."}}
```


Esses erros não são dessa execução do script e só foram exibidos como exemplo para seu formato.

Após uma execução bem sucedida do script, as informações lidas de suas origens (sejam db ou csv) estarão



implementadas no site.


Os departamentos foram sincronizados com os grupos na origem produtos. Note que não atribuímos nenhum subgrupo aos produtos e consequentemente nenhum subdepartamento foi gerado.




O que você procura?

TODOS DEPARTAMENTOS ▾

ENTREGA


 / Departamentos / Bebidas



REFRIGERANTE ANTARCTICA
GUARANÁ GARRAFA 237ML

R\$ 1,50


COMPRAR



REFRIGERANTE COCA-COLA
LATA 350ML

R\$ 2,50




COMPRAR



SUCO CONCENTRADO
MAGUARY CAJÁ 500ML

R\$ 3,00

COMPRAR



Os produtos foram distribuídos dentro dos departamentos correspondentes em seus cadastros.

Formas de pagamento [ver todas](#)

00

MERCADEIRO

35 / 36

As formas de pagamento também foram importadas, entretanto elas devem ser atribuídas a tipos específicos de recebimento (retirada, entrega e/ou online) para serem exibidas no site.

Após a primeira execução do script, algumas novas pastas serão criadas, são elas:

Pasta	Descrição
/dist	Pasta contendo versão "compilada" do ConecSync .
/lojas	Pasta contendo hashes para monitoramento de modificações nos cadastros.

Os códigos fontes (na pasta /src) em linguagem typescript utilizada para criação do script, são transpilados (algo como interpretados), o que quer dizer que são convertidos para outros códigos fontes (dessa vez em javascript) para que estejam prontos para execução. O resultado desse processo é gravado na pasta /dist.

A pasta /lojas contém os hashes de todos registros de todas lojas da última execução do script, e serão utilizados na próxima vez que ele rodar, para determinar quais registros foram modificados e só executar chamadas às apis dos projetos dos que precisam ser sincronizados, dessa forma as execuções do **ConecSync** se tornam extremamente rápidas uma vez que só comunicam as modificações ocorridas, o que reduz o tempo e ocupação da rede sem prejuízo ao processo. Se você apagar essa pasta (ou algo dentro dela), os hashes que não forem encontrados serão criados e suas chamadas de api correspondentes executadas.