Asakusa Framework
基本設計ケース・スタディ

改訂履歴

版	日付	内容
1.0	2012/01/17	新規作成

1. はじめに

実際に以下のケース・スタディとして基本設計を行ってみる。参考例は「小売業の POS データの塊を正規化してカテゴリー別に集計するフローパート」を表す。

2. 想定シナリオ

■業務フロー
複雑な業務フローにはなっていない。担当者は特に存在せず、画面処理も想定されない。バッチのみの処理である。POS の売上データを取り込んでクリーニングをして、各カテゴリー別のデータを作成するという内容である。

業務要件としては、以下の事項があげられている。

1. すべてのデータを処理し、エラーデータは報告されること。データロストは一切許容されない。
2. クリーニングしたデータは将来、別のシステムで使う可能性があるので、途中で切り出せる処理を追加した場合にコストが上がらないようにしておくこと。
3. 消費税の計算は内税になっているので、外税に引き直すこと。消費税計算でイレギュラーなケースがあったら報告すること。ただし原則として処理は止めないこと。また、今後、消費税は増税およびカテゴリー別に税率が変わる可能性があるので、対応できるようにしておくこと。
4. 所与のマスターとの付け合わせのクリーニングを行うこと。（マスターは提示。）
5. オープン前の新店のデータがテスト的に流れてくるので、これは処理から除外すること。集配信側で判断することは難しいので、バッチ処理側で棄却する仕組みにして、開店当日から取り込むこと。尚、マスターには有効期限を設定しているので、これを参照すること。
6. マスター未登録の商品もあるので、適切にハンドリングすること。割と頻繁に起こる。

3. バッチの入出力フォーマットと業務概要

提示されている入出力

3.1. *入力モデル*

未チェックの POS データ。ただし、店舗からの入力であり、単品情報から構成されたレシートをまとめた一定の塊になっている。データフォーマットの概要は以下の通り

・店番号：レシートを発行した店舗のユニークコード
・日付：レシートの発行日。売上の集計の日付になる。
・ID：レシートを一定の単位でまとめたデータの束への ID。通常は締め処理のタイミングで発行される。
・承認番号：締め処理で発行される承認コード
・レシート ID：レシートユニーク ID
・行番号：レシート内部での商品の並び順
以上が ID 系になる。投入データの単位は ID 単位＝締め処理単位であり、処理フローは、複数のデータ（すなわち投入データは複数インスタンス）の一斉投入で実行される。

・GTIN コード：商品コード
・商品名：商品名
・売価：単品の売価（売単価）
・数量：販売された数量
・金額：単品の販売金額 売価 x 数量
以上は、レシート＋行番号でのユニークな情報。集計の基礎単位になる。

・合計金額:レシート単位での合計金額(内税)
・消費税:レシート単位での消費税額の合計単位
・受取金額:レシート単位での入金金額
以上はレシート単位での集計数値であり、データユニークではなく、同じレシート ID には同じデータが入る。

3.2. *出力モデル*

正常モデルは 3 種類になっている。

1:売上伝票レイアウト
店別部門別の売上伝票を表す。ヘッダーは、店番・日付・部門コードで構成される。明細は 6 行とする。
各行の記載は、GTIN コード・商品名・販売数量・販売金額・消費税額の結合文字列とする。
改ページごとにフッターとしてページ番号付与される。イメージデータなので、行数はハードコードになる。

モデル・フィールドは以下の通り
・店番号:店舗コード
・日付:売上日付(レシート日付)
・部門コード:部門コードで集計計上する
・行番号 1～行番号 6:GTIN コード・商品名・販売数量・販売金額・消費税額の結合文字列
・ページ番号

2:カテゴリー集計レイアウト
単品データとそれに関連する集計データがアタッチされている形式になる。各単品データに冗長的にすべての集計データを保持する。

モデル・フィールドは以下の通り
・店番号:店舗コード
・日付:レシート日付
・GTIN コード:商品 ID
・商品名:商品マスター上の名称
・部門コード:最上位カテゴリーコードになる
・カテゴリーコード 1:中位層カテゴリー
・カテゴリーコード 2:下層カテゴリー

以下は集計フィールドになる
・カテゴリーカットの集計～すべてで、金額・数量・消費税・客数を表示
・全店計
・全店部門計
・全店カテゴリー1 計
・全店カテゴリー2 計
・全店 SKU 計

・ロケーション別カットの集計
・店舗計
・店別部門計
・店別カテゴリー1 計
・店別カテゴリー2 計
・店別 SKU 計

提示されているマスター:
以下重要な項目のみ抜粋する。基本的にデータクリーニングのために、最低限の項目のみ選択されており、必要に応じて抽出処理が終わっている。

■商品マスター
商品データ（有効期限が設定されている）

・フィールド一覧
有効開始日付：マスターの有効開始日
無効予定日：この日以降マスターが無効になる
店番：店舗コード
商品コード：商品コード
商品名：商品名称
その他商品属性：割愛

■カテゴリーマスター
カテゴリー分類を管理している
今回事前に結合しやすいようにカテゴリーのツリー構造をフラットにしたものが提供されている。単品に毎に上位のカテゴリーデータが付与されている。（もともとのツリー構造をフラットにする処理も、このバッチ処理の中で行うのだが、ここでは割愛する）

・フィールド一覧
部門コード：最上位カテゴリー
部門名：部門の名称
カテゴリー1 コード：中位カテゴリー
カテゴリー1 名：カテゴリー1 の名称
カテゴリー2 コード：下位カテゴリー
カテゴリー2 名：カテゴリー2 の名称
商品コード：ユニークキー
商品名：商品名称

■店舗マスター
店舗コードを管理している（有効期限が設定されている）

・フィールド一覧
店番有効開始日：マスターの有効開始日
店番無効予定日：この日以降マスターが無効
店舗番号：店舗コード
店舗名：店舗名称

4. 考慮すべき業務ロジック

1 データクリーニング
POS データをマスターデータとぶつけて正当性を確認する。提示されているマスターは店舗・商品マスター・カテゴリーマスターになっている。特に、ストップエラーと処理続行エラーを区別する。エラーファーストの鉄則から考えてエラー検出の順序を考慮する。ストップエラーが先にチェックされる。

2 伝票イメージ作成
もともとの POS のデータは、ヘッダーが各明細行にコピーされているフラットデータで送られてきている。従って、フッターを後付けで作り上げる必要がある。店別部門別で集計伝票を作成して、一定の行数（ここでは 6 行）で改ページ処理を入れる。

3.消費税の処理
POS データはレシート単位で提示される。このレシート単位で、各単品の消費税の合計が内税で表示されている。これを単品に割り付けて、税額を控除した形に表示し直すことが必要である。（POS データの CSV 上は単品にばらし

た形で一ラインが構成されており、かつラインにはそのレシートの"総額"の税額が表示されている。）税額の控除ロジックは消費税率で割り戻し、レシート上の内税総額との差額は、レシート内で最も金額が大きい明細の単品の税額で調整する。また、消費税額がカテゴリー単位で変わる可能性も見越して、先にカテゴリーデータで結合処理をしておく。

4.カテゴリー集計
カテゴリーは基本的にツリー構成になっている。ツリー構成は 2 軸の 5 階層で構成される。すなわち、全店→部門→カテゴリー1→カテゴリー2→単品の軸と、個店→部門→カテゴリー1→カテゴリー2→単品の 2 空間での多次元集計になる。(上記のデータが入手できれば、カテゴリー→店舗の軸は、非常に容易に結合できるので省略している。)

5. 実際の設計

■考え方
以下に実際のデータフロー・データモデルの設計を行う。実際に DAG の設計を行う。

□データフロー

5.1. *まず入力と出力をつなぐことを意図する。*

まず出力を意識する。与えられている出力データは集計データと伝票確定データである。
出力は集計別のデータと伝票イメージになる。両者とも**集計結果**から構成されていることがわかる。対応する入力は POS データであり、フラットデータで構成されている。レシートデータがヘッダー的に各ラインに冗長的に展開されていることを確認する。また、業務要件から**データのクリーニング**も要請されていることがわかる。ラフ・スケッチとしては POS データを入力として、必要な処理は、データのクリーニング、集計、出力のフォーマッティングから構成されそうだという当たりはつく。出力項目をレビューして、過不足がないかどうか確認する。クリーニングは所与のマスターで処理ができそうなことが確認できる。細かいところだと、消費税の計算と客数の計算のタイミングが面倒だな、という印象が持てればよい。

5.2. *どのような処理が必要になるか、検討する。*

入口と出口を確認した後は、どのようなフィールドが必要になるのか、形成の過程を精査していく。出力のフィールドは、大きくは単品のデータと集計データで構成されている。これらは入力の POS データのクリーニング、集計、出力のフォーマッティングの処理で生成できそうな気配ではある。そのまま STS 分割を適用する。この場合、入力＋処理＋出力の形になる。この形を念頭に置きながら、それぞれに処理を割り当てることを考える。取り込み処理→処理→出力のフォーマット処理の順になることを考えると、以下のパターンを検討することができる。

入力処理：POS データの取り込み処理とクリーニング
本処理：データの展開/集計
出力処理：集計データと伝票データへの展開

出力要件が二本あるので、どこかで処理の結果を分岐させるオプションを確保しておく必要がある。出力はカテゴリーデータを結合した単品である必要があるため、結合してからの分岐になる。分岐してからの結合ではない。

ここでクリーニングの位置づけを検討してみる。まず要件を見ればわかるとおり、クリーニングは各種マスターと結合処理が必要になっている。データクリーニングを形式チェックと意味チェックから見てみる。形式チェックでエラーがあった時点で続行処理はできないので、取込時点に形式チェックをしてしまうのは特に問題がない。一方、意味チェックは様々な情報との結合処理が必要になる。すなわち、POS データ物理の取り込みの処理時に形式チェックを行ってしまうのは問題はないが、マスターとの結合を必要とする意味チェックまで同じ処理に統合してしまうのはちょっとどうか？ということになる。
とはいえ、「クリーニング処理」という意味では処理として統一感はある。さてどうしたものか？要件からも「クリーニングしたデータは別で使う可能性があるので、途中で切り出せる処理を追加した場合にコストが上がらないようにしておくこと。」とある。

案 1

入力処理:POS データの物理的取り込み処理(形式チェック含む)
本処理:クリーニング～マスター結合等の重い処理
本処理:データの展開/集計
出力処理:集計データと伝票データへの展開

案 2
入力処理:POS データの取り込み処理
　　入力-下位前処理:物理取込
　　入力-下位本処理 1:形式チェック
　　入力-下位本処理 2:意味チェック
　　入力-下位後処理:POS クリーニング終了データ書き込み
本処理:データの展開/集計
出力処理:集計データと伝票データへの展開

どちらがよいかは、業務要請と処理の重さによる。POS データのクリーニング後のデータを他で転用するような場合は、案 2 が妥当だろう。他方、クリーニングの処理が、かなり重いことが想定される場合は、案 1 が妥当だと言える。業務要件としては、案 2 が妥当だろう。ただし、ここではチュートリアルであるので、見通しの良さ等を考慮して案 1 で検討を進める。尚、案2へ要請を考慮して、マスター結合によるクリーニングとデータ集計は明確に切り分ける方針にする。

次にデータの展開・集計を検討する。消費税の計算は、これはレシート単位でグルーピングが必要になるのは、すぐにわかる。と同時に客数のカウントを行った方がよいことも推測できる。(客数はレシート ID のカウントになる)・・よって必要な結合はその前にやっておいた方がよいだろうということも推測できる。

したがって、必要データの結合→消費税計算→集計が流れとしてよさそうだということが分かる。

次に最終のアウトプットの、売上伝票レイアウトとカテゴリー集計レイアウトについて検討する。
売上伝票レイアウトでは、店舗別-部門以下で、各単品の GTIN コード・商品名・販売数量・販売金額・消費税額の結合文字列になっている。また、カテゴリーカテゴリー集計レイアウトでは、これに相当する集計は店別単品の販売数量・販売金額・消費税額・客数の部門別の集計になる。一部データの流れの共用を検討すべきこともわかる。

方針として以下の方向で設計を進める。

入力処理:POS データの物理的取り込み処理(形式チェック含む)
これはデータの取り込みとして設計する。このときに形式チェックのクリーニングのみを行う。内容は POS データのデータフォーマットの形式確認になる。

本処理:クリーニング～マスター結合
クリーニングと必要なデータの結合処理を行う

本処理:データの展開/集計
消費税処理等の処理、カテゴリーデータの集計、から構成される。

出力処理:集計データと伝票データへの展開
出力処理を行う集計データの結合処理が必要になる。データを分岐させて、集計データと伝票データのそれぞれの作成処理に分流する。

まず深さについては、全部の処理をフラットに並べると、処理が複雑になったときに見通しが悪くなるので、一階層準備する。

5.3. *順に処理を詳細に検討していく。特にエラー処理を設計する。*

さらに処理を詳細にブレイクダウンしていく、と同時に例外処理を設計していく。

●入力処理：POS データの物理的取り込み処理（形式チェック含む）
取り込んだ POS データのバリデーションになる。前述のように意味チェックは本処理に回すので、形式チェックを行う。
文字コード・型・桁数・存在等を確認する。
失敗した場合は、エラー処理をして、データ・インスタンスの単位で、処理は即時に終了処理になる。

●本処理：クリーニング～マスター結合
クリーニングと必要なデータの結合処理を行う。
データクリーニングは、POS データの各項目の意味の整合性の確認になる。基本的には各マスターの存在チェックが主要になる。マスターデータで、各 POS データの記録されている項目が適当か確認する。各マスターとのチェックごとに処理を設計する。

マスターチェックに失敗したデータは、そのままエラーデータとして、処理を分岐させる。成功したデータだけで処理を続行させることが基本になる。ただし、失敗したデータであっても、そのまま処理を続行させていくケースもある。処理を続行しないものから失敗させることが望ましい。これらはマスターチェックの内容で個々に判断していく。

各マスターのチェックを以下のように設計する。

・店舗マスターとの照合
店舗マスターチェック。ここでマスターがない場合は、基本的にデータとして後続処理ができないので失敗させる。処理するデータが間違っている可能性が高い。ただし、要件の「開店前の新店のデータがテスト的に流れてくるので、これは処理から除外してほしい。」という課題がある。結合処理をするときに、マスターの有効期限を確認して対応する。

・商品マスターとの照合
いわゆる「未登録エラー」にあたる。業務要件より、わりと頻繁に起こることを想定する。マスターにない商品が売られているということになるが、マスターの登録忘れ等の登録ミスや削除ミスにより発生することがあるのだろう。・・なので、実は POS データが正しくて、マスターが間違っていることにもなる。ここで考慮すべきは”チェック順”になる。この処理まで失敗しなかったデータは、日付等形式チェック→店番チェックの順に、それらをクリアしてきている。したがってこの商品マスターで引っかかっている場合は、商品マスターにも問題がある可能性もあると判断する。

ここでは、集計自体の業務的な優先度を考慮して、このチェックでのエラーデータは処理を続行させる。ただし、エラー処理はレポートし、警告エラーとして扱う。

この種類のチェックは取り扱いが微妙である。このケースでは、POS データではなくて、むしろ商品マスターデータの制御が間違っている可能性があることと、データの集計処理の整合性を合わせることを考慮している。日付管理等の形式マスターや店舗マスターと、商品マスターとの扱いでは、相対的に後者の方が更新頻度が高く、従って間違っている可能性がある。逆に言うと、日付管理等の形式マスターや店舗マスターと合致しないようなデータは、致命的なエラーがあると考えられ、そもそも合計処理等の続行処理は意味がない。勿論、商品マスターの整合性確保が最優先であり、異常データは即時停止というケースもある。ここでは、そのようなケースは想定していない。

結合が成功したデータについては、適時必要なフィールドの更新を行う。ここで商品名を商品マスターの表示名に上書きして更新する。通常、レシートの表示商品名称は表示用に簡略化されているので、正式な商品名に名称を更新する。尚、失敗した場合はレシート名称でそのまま処理を続行する。

●本処理：データの展開/集計
マスターチェックの終了した POS データに集計に必要な情報の付加を行い、集計処理を行う。その後のデータの分岐を行う。

・カテゴリー結合
クリーニングが終わったデータにカテゴリーデータを結合する。商品 ID から、カテゴリーコードを取得する。カテゴリーコードが見つからない場合はエラー処理にするが、「その他コード」で処理を続行する。ここでのカテゴリーマスターとの結合も商品マスターと照合と同様の扱いを行い、エラーであっても処理を続行する。

・消費税計算処理

消費税額はレシート単位になっている。従って、一度全部の単品データをレシート単位でグルーピングする。(POS レジでその位の処理はやっておけ的な話は、とりあえずおいておく)、グループされた各単品で消費税を計算し、レシート計の税額と比較して差額を一番税額の大きな単品に付け替える。ここで要件である「消費税計算でイレギュラーなケースがあったら報告してほしい。ただし原則として処理は止めないこと」を考慮する必要がある。イレギュラーなケースとして容易に想定できるケースが、レシート単位のグルーピングに失敗するケースがある。この場合単品に割りつける消費税とレシートデータの消費税額の差額が計算できない。とはいえ、消費税額の総額には変更はないので、特定の単品に差額を負担させて、処理は続行させる必要がある。処理上は差額がヒットした件数 x1 円を超えた場合に異常としてログを吐く形にする。

また、カテゴリーによって消費税率が変わることにも対応できるに留意する。2011 年現在はまだ税率は一律なので、ここでは処理は行わない。

・客数計算
基本的にはレシート ID 一つにつき、客数は 1 のカウントになる。カテゴリー計算を行う場合に、階層積み上げ計算が行うことができない。階層に関係なくカテゴリー単位で、distinct を取らなければいけない。たとえば、親カテゴリーX の所属に、サブカテゴリーA と B があり、それぞれに属する商品を一回で買った客数が一人だった場合、それぞれカテゴリーの客数はすべて 1 にカウントされなければならない。実装上工夫が必要になる。

・カテゴリーデータ集計
店舗単位・単品単位・カテゴリー単位で集計する。カテゴリーツリーは階層構造になっているため、下位の SKU である単品単位での集計から順にカテゴリーツリーを集計してさかのぼる形の処理では時間がかかってしまう。そこで、それぞれ集計が必要なキーでカテゴリーコード・単品コードで集計処理を行う方針をとる。分散並列集計を行う。

出力処理:集計データと伝票データへの展開
データ処理の分岐させ、それぞれの後続処理を並列に走らせる。カテゴリー集計のアウトプットがキー単位で生成されてくることがわかるので、それぞれを後続のフォーマッティングの処理につなぐ。

●カテゴリーデータへの展開
集計済みのデータを必要なデータフォーマットに変形する。集計結果から入力されるデータは、
{金額・数量・消費税・客数} x
{全店計/全店部門計/全店カテゴリー1 計/全店カテゴリー2 計/全店 SKU 計}
および
{金額・数量・消費税・客数} x
{店舗計/店別部門計/店舗カテゴリー1 計/店舗カテゴリー2 計/店舗 SKU 計}
となり、4 つの集計数値をもつ集計データのフローが 10 本生成されてくる。これを単品 SKU で結合処理する必要がある。

●伝票イメージ作成
印刷用の伝票レイアウトデータを保存する。レイアウトに沿った CSV を生成する。
集計データとして必要なのは、店別・単品の集計結果のデータになる。
{金額・数量・消費税・客数} x {店舗 SKU 計}
これを部門で集計すればよい。

5.4. *さらに順序性について確認する。*

ここでもう一度処理の順序性を確認する。効率の悪い処理の並びに順になっていないか等の検証を行う。

クリーニングがないと各種集計ができないので、まず「クリーニング→集計処理」の順序で問題はない。消費税の計算の順序位置を検討してみる。検討している「カテゴリーの結合→消費税計算→カテゴリー集計」のフロー順序は、カテゴリーの結合→カテゴリー集計→消費税計算や、消費税計算→カテゴリーの結合→カテゴリー集計の順序も検討できるが、消費税の計算にはレシート単位でのサマリーが必要なので、完全に単品にばらしてしまうカテゴリー集計のあとでは、具合が悪い。また、「今後、消費税は増税およびカテゴリー別に税率が変わる可能性があるので、対応できるようにしておくこと。」という要件もあるため、消費税計算の前には必ずカテゴリー結合が必要である。

よって、本処理は　検討している、データクリーニング→集計データ作成(カテゴリー結合→消費税計算→カテゴリーデータ集計)の順で構成できる。
尚、階層としては、データクリーニングと集計元データ作成の二つにしているケースにする。

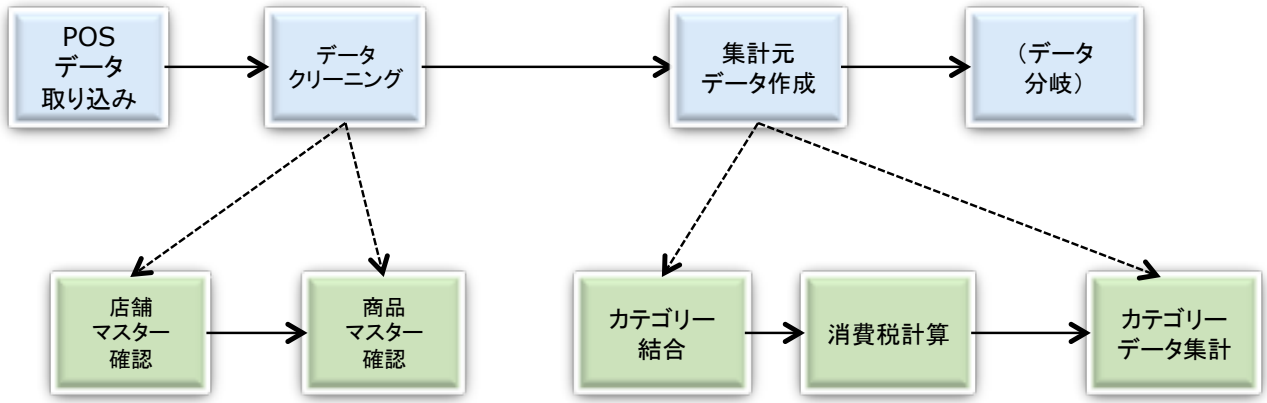

図 1

（ただし、別掲の詳細実装ガイドでは、便宜的なフラットな構造を採用する。）

図 2

以上で、基本的な設計は一段落する。

6. 附随的に行うべき事項を整理する

6.1. *TX 処理*

複数の業務間のデータのやりとりは発生していないので、強制的に TX を発生させる必要はない。ただし、クリーニング以降と以前ではデータの練度が違うので、クリーニングまでを一つの TX として境界を設定しておく。また、これは業務要件の「クリーニングしたデータを、途中で切り出せる処理を追加した場合に、コストが上がらないようにしておくこと」にもつながる。これにより、何らかの事情でカテゴリー集計が失敗したとしても、クリーニングまでやり直す必要がなくなる。

尚、集計処理前に、結合が終わった段階で一度 TX を切っておくという判断もある。集計処理が重い場合は、失敗した場合にその直前まで戻ることができれば、集計失敗時点でのリトライがしやすいというメリットがあるためである。

6.2. *並列性確認*

ここで一旦、この処理自体の並列性が確保できるかどうか、業務的に検証してみる。
POS データ取込～人手を増やせば処理可能。
店舗マスター確認～店舗単位で人を割り当てて、伝票のコピーを渡して、自分の分だけ抜く作業で処理可能。
商品マスター確認～商品台帳のコピーと伝票のコピーを分割して渡して人海戦術で処理可能。
カテゴリー結合～商品マスターと同じ手法で処理可能
消費税計算～レシートデータ単位で人を分ければ対応可能
カテゴリーデータ集計～集計すべきキーの単位で人をアサインして、伝票のコピーを回す。

伝票イメージ作成～人手で分散可能
カテゴリーデータ展開～人手で分散可能

概ね、並列性が確保できそうな気配ではある。
尚、このように業務系での人手での展開を想定検証したときにポイントになりそうだなと推定できるボトルネックが、実際の分散実装での勘所と同じポイントになることがある。例えば、カテゴリーデータの集計を人手でやる場合、カテゴリー毎に担当者を決めてそこの伝票を流せばよい、というやり方は、実際の実装でもカテゴリーをキーにして、如何に分散集計をさせるか、という点と一致する。机上でのデザインが実装手法へのアナロジーになっている。基本設計の段階で、簡単にシミュレートしておく。

6.3. *業務エラー処理*

さらに、前述の詳細内容で検査された、業務エラー処理を整理しておく。
大きく以下の 3 つに分けられる。それぞれのエラー処理は、エラーデータを作成して、それぞれの種別にマージして、最終のエラーデータを作成する。

・ストップエラー
そもそも完全に問題があり、後続の処理自体に意味がない場合のエラー。ここでは存在しない項目やあり得ない日付のデータが発生した場合のエラーであり、この場合はこのデータについてのみ処理を続行させない。それ以外のデータでジョブを続行させる。POS データエラー・店舗マスターエラーが該当。

・警告エラー
本来はあってはいけないが、処理を続行することが必要なケース。例外処理をして、例外データを正常系に流す形の処理になる。非常にたびたび起こるケースでのエラー。商品マスターエラー・カテゴリー結合エラーが該当する。

・軽微なエラーだが、取扱が微妙なエラー
ここでは消費税のグルーピングの失敗の例。エラー系に流すデータも存在しない上、発生件数も限られると思われる。警告エラーとして処理は続行する。ただし、外部のシステムに根本的にエラーが発生している可能性もあるため、検証が必要である。

エラー復帰検討
特にストップエラーについては、前もって運用を検討しておく必要がある。ここでは POS データの形式チェックのエラーと店舗マスターのエラーである。

以上のフローをまとめると以下の DAG が形成される

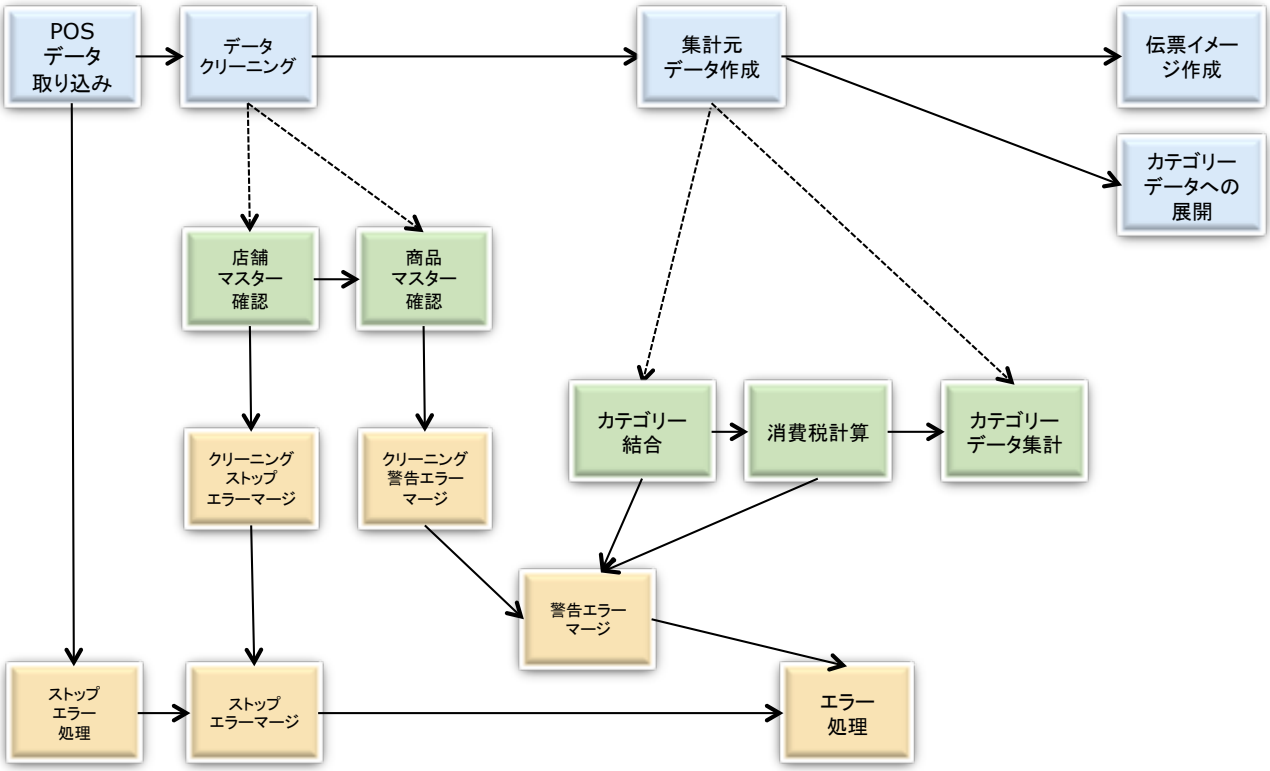

図 3

□データモデル
今回は与えられているデータフォーマットが比較的単純であるので、明示的に DDL を記述できる。

入力:POS データ(トランザクションデータ)
注意すべきは、レシート ID 等の当該データインスタンスではなくその上位の集合にユニークなデータを、当該データインスタンスにユニークなデータと区別して記述しておくこと。

入力:店舗マスター・商品マスター・カテゴリーマスター(マスターデータ)
それぞれ、POS のデータクリーニング用に抽出処理がされている。

出力:売上伝票レイアウトとカテゴリー集計データ
6 行の改ページが印刷のイメージデータなので、行数がハードコードされる。カテゴリー集計レイアウトは、単品データに所属カテゴリーの集計データが最上位まで結合されている。

出力:エラーデータ
エラー処理されるデータは、POS 取込エラー・店舗マスター結合失敗エラー・商品マスター結合失敗エラー・カテゴリーマスター結合失敗エラーおよび消費税計算警告エラーになる。前者の 4 っつは処理すべきデータインスタンスにエラーコードを附記して、エラー処理に流す。したがってデータモデルは基本的に入力モデルの POS データを拡張したものとなる。最後の消費税計算警告エラーはエラーデータを作成して、エラー処理に流す。

TX データの整理
フローモデルの TX 処理で検討したように、データクリーニングが終了した段階で、TX 処理をコミットする。
以上で基本設計がほぼ終了する。この後、この基本設計を出力として、詳細設計に入っていく。詳細設計では、より詳細な演算子の設計を行う。