DOSSIER DATA ENGINEERING · GOUVERNANCE

# QUAND UNE DONNÉE CHANGE DE SENS

## Préparer l'évolution d'un flux entre applications

Publié le 10 novembre 2026 · Version 1.3.0 · Mise à jour le 10 novembre 2026
Article en ligne :
https://datanalysis.re/ressources/data-engineering/donnee-change-de-sens-evolution-flux

Une mise à jour peut être correctement déployée, les traitements peuvent se terminer à l'heure, et les montants utilisés par la finance devenir faux. Avant de modifier une donnée partagée, il faut donc vérifier ce que les applications qui la reçoivent vont en comprendre. Voici une méthode pour organiser cette revue, avec une fiche à remplir sur un flux précis.

Exemple fictif : même champ, même type, autre unité

| Avant · invoice_total | Après · invoice_total |
| --- | --- |
| **48.50** euros | **4850** centimes |

Lu en euros : 4 850 €, soit cent fois trop

Normalisé : 4850 ÷ 100 = 48,50 €

## Un nombre valide peut être mal interprété

Prenons un exemple fictif. Une application de facturation transmet le total TTC d'une facture dans le champ `invoice_total`. Pour une facture de 48,50 euros, elle envoie la valeur numérique `48.50`. Après une évolution, elle transmet `4850`, désormais exprimé en centimes. Le nom du champ reste identique et le schéma autorise toujours un nombre.

Un contrôle qui vérifie uniquement la présence du champ et son type peut accepter les deux valeurs. Pourtant, si le traitement suivant continue de lire des euros, il obtient 4 850 euros. Dans cet exemple, le montant interprété est cent fois trop élevé. La devise EUR ne suffit pas à lever l'ambiguïté : il faut aussi connaître l'unité de représentation.

Un rapprochement avec la facture source ou une règle métier bien choisie peut détecter cet écart. Le problème est la portée du contrôle effectivement exécuté. Un statut technique « terminé » nous renseigne sur l'exécution du traitement ; la justesse du montant dépend aussi des vérifications prévues.

La documentation dbt décrit cette difficulté : un calcul peut changer les résultats d'une colonne tout en conservant son nom et son type. L'équipe qui maintient le modèle doit apprécier les conséquences pour ses utilisateurs. [1]

## Retrouver les usages avant de modifier le champ

Pour préparer le changement, nous proposons de partir d'une donnée et de suivre son parcours. Qui la produit ? Quels traitements la transforment ? Quelles applications, quels exports ou quelles décisions en dépendent ? Une vue de dépendances constitue un bon départ, mais sa couverture doit être vérifiée. Un fichier exporté puis repris manuellement peut échapper à la cartographie technique.

Dans notre exemple, imaginons deux consommateurs : un tableau de suivi financier et un export de recouvrement. Le premier affiche les totaux ; le second sert à préparer une liste de factures à traiter. Le changement doit être testé dans les deux usages. Valider uniquement l'arrivée de la donnée dans l'entrepôt laisserait la lecture finale sans vérification.

Pour chaque consommateur, notez un responsable, le traitement à adapter et une preuve attendue. Le responsable de l'application source confirme la nouvelle représentation. L'équipe d'intégration vérifie la transformation. Le propriétaire de chaque usage confirme le résultat obtenu. Une personne pilote la transition et connaît les points qui restent ouverts.

Cette liste doit rester proportionnée au flux. Pour une donnée peu utilisée, quelques lignes peuvent suffire. Pour un montant diffusé à plusieurs systèmes, une dépendance sans responsable mérite d'être résolue avant de fixer la bascule.

## Rendre la définition et les attentes explicites

Un contrat de données est un accord entre ceux qui produisent une donnée et ceux qui l'utilisent. Confluent distingue dans cet accord la structure, les contraintes, les métadonnées, les règles et l'évolution. [2] Pour notre revue, nous retenons les informations qui permettent de décider si le changement est acceptable.

Points à préciser, appliqués à l'exemple fictif

| Point à préciser | Application à notre exemple fictif |
|---|---|
| Définition métier | Total TTC de la facture, une facture par enregistrement. |
| Unité et devise | Euros avant le changement ; centimes d'euro après. |
| Cas particuliers | Avoirs négatifs, montant nul, annulation et valeur absente à traiter selon des règles explicites. |
| Versions et période | Version reçue, date d'effet et traitement des factures historiques. |
| Consommateurs | Suivi financier et export de recouvrement, avec un responsable pour chacun. |
| Contrôles et réaction | Rapprochement avec les factures sources ; arrêt ou mise à l'écart du flux si l'écart est inexpliqué. |

Ce contenu peut être tenu dans un référentiel, un dépôt de code ou une spécification versionnée. L'emplacement doit permettre aux équipes concernées de le retrouver et de relire le changement. Le document doit ensuite se traduire en contrôles exécutés et en validations attribuées. Une règle écrite qui ne participe jamais à une recette ou à une surveillance ne

détecte aucun écart.

L'Open Data Contract Standard distingue d'ailleurs les règles décrites en texte des règles exécutables, par exemple en SQL ou avec un moteur de qualité. [3] Cette distinction aide à voir ce qui est déjà contrôlé et ce qui reste à mettre en œuvre.

## Tester le résultat dans une unité commune

Le test utile compare des valeurs qui ont le même sens. Si la nouvelle valeur vaut `4850` centimes, son montant normalisé est de `48.50` euros. C'est ce montant qu'il faut comparer à la référence en euros. Comparer directement les deux nombres bruts produirait une alerte attendue, sans vérifier la conversion.

Choisissez un petit jeu de factures dont les résultats attendus sont connus. Incluez les cas autorisés par le métier : une facture ordinaire, un avoir, un montant nul et les situations d'annulation ou d'absence de valeur pertinentes. Pour chaque cas, gardez la donnée d'entrée, la version utilisée, le résultat attendu et le résultat observé dans chaque consommateur.

Vérifications et résultats attendus dans l'exemple fictif

| Vérification | Résultat attendu dans notre exemple |
|---|---|
| Facture de 48,50 euros | `4850` centimes devient `48.50` euros après normalisation. |
| Avoir de 12 euros | Si les avoirs sont négatifs, `-1200` devient `-12.00` euros. |
| Unité absente ou version inconnue | Le traitement applique la règle convenue et signale le cas ; il ne devine pas l'unité. |
| Même lot de factures | Le total normalisé correspond à la référence, avec une règle d'arrondi documentée. |
| Rejeu d'un lot | Le résultat ne comporte pas de doublon supplémentaire. |

Le total est utile, mais il peut masquer des erreurs qui se compensent. Vérifiez aussi les lignes, leur nombre et leur identité. Une tolérance doit avoir une raison métier, par exemple un arrondi défini ; un seuil choisi uniquement pour faire passer la recette rend le contrôle peu instructif.

Pour les montants, précisez également où s'applique l'arrondi et combien de décimales sont conservées. Le même calcul réalisé à la ligne ou après agrégation peut donner des résultats différents. Ce choix doit être partagé par les consommateurs concernés.

## Choisir la transition selon les dépendances

Si tous les consommateurs peuvent être adaptés ensemble, une bascule coordonnée peut convenir. Il faut alors définir la séquence d'intervention, les tests après déploiement et la décision à prendre si l'un d'eux échoue. Une fenêtre courte ne dispense pas de savoir qui peut interrompre le traitement.

Si les équipes ont des calendriers différents, une période de coexistence peut être nécessaire. Dans notre exemple, la source pourrait exposer une nouvelle version ou un nouveau champ dont l'unité est explicite. Les consommateurs migrent alors selon un calendrier connu. Il faut éviter

qu'une même donnée devienne successivement « euros » puis « centimes » sans moyen fiable de distinguer les deux représentations.

dbt permet, pour ses modèles versionnés, de conserver des versions parallèles et de fixer une date de dépréciation. Sa documentation rappelle aussi le coût de cette coexistence. [1] Le principe est utile au-delà de cet outil : chaque version conservée doit avoir des utilisateurs identifiés et une condition de retrait. Multiplier les copies sans calendrier déplace simplement le problème.

La transition doit aussi traiter l'historique. Les factures déjà enregistrées restent-elles dans l'ancienne représentation ? Seront-elles converties ? Comment reconnaître un lot ancien rejoué après la bascule ? Une date de traitement seule peut être insuffisante si le lot contient des données plus anciennes. Une version explicite, ou une autre règle non ambiguë, doit guider l'interprétation.

## Préparer la reprise avant de donner le feu vert

Le retour arrière doit couvrir le code et les données déjà traitées. Remettre un ancien traitement en service ne corrige pas automatiquement les montants diffusés pendant l'incident. Définissez le lot concerné, les sorties à suspendre et la façon de recalculer ou de corriger les résultats sans créer de doublons.

Avant la bascule, l'équipe doit pouvoir répondre à une question concrète : si le contrôle sur une facture échoue, quelle action arrête la propagation et qui la déclenche ? Selon le flux, cela peut être une mise en attente, une quarantaine de données ou un arrêt du consommateur. La réponse dépend du coût d'une donnée incorrecte et de celui d'une interruption.

Fixez aussi le point de contrôle après déploiement. Les responsables des usages doivent confirmer leurs résultats sur le même périmètre de données. La fermeture du changement peut alors s'appuyer sur des preuves identifiées, et le retrait de l'ancienne version sur l'absence de consommateurs qui en dépendent encore.

## Préparer votre prochaine revue de changement

La fiche pratique associée reprend cette démarche : un exemple fictif rempli, puis une grille pour documenter votre flux. Elle ne remplace pas les résultats de tests : elle indique ce qui doit être vérifié, par qui, et où retrouver la preuve.

Commencez par une donnée dont l'évolution est déjà prévue. Réunissez l'équipe source et les responsables des usages. Si vous ne pouvez pas encore nommer un consommateur, son unité attendue ou son responsable, vous avez trouvé un point à traiter avant la livraison.

Vous faites évoluer une donnée utilisée par plusieurs applications ? Parlons d'un flux précis pour cadrer ses dépendances, les contrôles à prévoir et la transition.

→ Cadrer mon flux avec DATANALYSIS : https://datanalysis.re/contact?lang=fr

→ Fiche remplissable (PDF) : https://datanalysis.re/downloads/data-n2-fiche-remplissable.fr.pdf

→ Fiche HTML : https://datanalysis.re/ressources/data-engineering/donnee-change-de-sens-evolution-flux/fiche

## Expertises associées

Data Engineering : https://datanalysis.re/expertises/data-engineering?lang=fr

Data Governance : https://datanalysis.re/expertises/data-governance?lang=fr

## Sources

Sources revérifiées le 10 novembre 2026. La méthode et les exemples sont pédagogiques ; aucun résultat client n'est présenté.

[1] dbt Labs | Model versions (changement de calcul sans changement de nom ni de type ; versions et dépréciation)
https://docs.getdbt.com/docs/mesh/govern/model-versions

[2] Confluent | Data Contracts for Schema Registry (structure, règles, métadonnées, évolution)
https://docs.confluent.io/cloud/current/sr/fundamentals/data-contracts.html

[3] Bitol | Open Data Contract Standard, Data Quality, version 3.2.0 (description textuelle et contrôle exécutable)
https://bitol-io.github.io/open-data-contract-standard/v3.2.0/data-quality/

Exemple fictif à visée pédagogique ; aucun résultat client.