Question 1 : Différence entre erreur de transformation et ligne invalide métier
Une erreur de transformation est un problème technique lors du traitement des données, comme l'impossibilité de convertir un texte en nombre ou un format de date incorrect. Une ligne invalide métier est une donnée techniquement correcte mais qui ne respecte pas les règles business, comme un montant négatif ou un nombre de passagers à zéro.

Question 2 : Pourquoi est-il dangereux de laisser passer des total_amount < 0 ?
Les montants négatifs faussent tous les calculs financiers et le chiffre d'affaires. Ils conduisent à des décisions stratégiques erronées basées sur des données incorrectes. Ils peuvent créer des problèmes de conformité lors des audits. Enfin, ils corrompent toute la chaîne de traitement en cascade jusqu'aux dashboards de direction.

Question 3 : Différence entre rejet technique et rejet métier
Un rejet technique provient d'un problème de format ou de structure des données, comme une date au format invalide ou une colonne manquante, et relève de la responsabilité du data engineer. Un rejet métier concerne des données techniquement valides mais illogiques pour le business, comme un trajet de 500km en ville ou 15 passagers dans un taxi, et nécessite une validation par l'équipe métier.

Question 4 : Pourquoi un pipeline doit-il être rejouable (idempotent) ?
Un pipeline idempotent garantit que rejouer le même traitement produit exactement le même résultat. C'est critique en cas de crash serveur pour éviter les doublons lors du redémarrage. Par exemple, dans un pipeline de facturation qui crash à 80% de traitement, relancer sans idempotence facturera deux fois les premiers clients, tandis qu'un pipeline idempotent ne traitera que les 20% manquants.

Question 5 : Que se passe-t-il si le schéma du CSV change ?
Une nouvelle colonne peut être ignorée et causer une perte de données. Une colonne supprimée fait planter le pipeline. Un changement d'ordre provoque un mauvais mapping des données. Pour anticiper, il faut utiliser les noms de colonnes plutôt que les positions, ajouter une validation de schéma en début de pipeline, et documenter clairement le schéma attendu.

Question 6 : Pourquoi séparer les flux "clean" et "rejected" ?
La séparation permet une traçabilité complète pour savoir combien et pourquoi des lignes sont rejetées. Elle facilite le debugging en permettant d'analyser les patterns d'erreurs. Elle garantit la conformité en prouvant que les données invalides ne polluent pas le système. Elle permet la réconciliation des volumes et l'amélioration continue en identifiant les sources de problèmes. Supprimer silencieusement empêche toute détection de problème et correction de la source.

Question 7 : Transformation Hop pour filtrer selon condition logique
La transformation principale est "Filter rows" qui permet de définir des conditions logiques avec AND, OR et NOT, et de créer deux flux de sortie true et false.

Question 8 : Qui définit les règles de validation ?
Les deux collaborent : le métier définit les règles business et les seuils acceptables, comme le montant maximum d'une course ou le nombre de passagers autorisés. Le data engineer traduit ces règles en logique technique, propose des validations techniques supplémentaires, et implémente les contrôles dans

le pipeline. La collaboration est essentielle car le métier connaît le domaine mais pas les contraintes techniques, tandis que le data engineer connaît les capacités techniques mais pas toutes les règles métier.

Question 9 : Pourquoi ajouter un processing_timestamp ?
Le processing_timestamp permet de savoir exactement quand chaque donnée a été traitée. Il rend le pipeline idempotent en permettant de détecter les doublons. Il facilite le debugging en traçant le parcours des données. Il permet de reconstruire l'historique des traitements. En cas de retraitement, il permet de distinguer les différentes exécutions du pipeline.

Question 10 : Étape la plus coûteuse en performance sur 10 millions de lignes
L'écriture en base de données est généralement la plus coûteuse car chaque ligne génère une transaction. Les jointures avec d'autres tables sont également très coûteuses car elles nécessitent des index et des recherches. Les transformations simples comme les filtres ou calculs restent rapides. Pour optimiser, il faut utiliser des bulk inserts, indexer correctement les tables, et paralléliser les traitements quand c'est possible.