Migration de données par Kettle

Marc Cousin

Outils dédiés

- `Ora2PG`
- `My2PG, my2pg.pl, mysql2pgsql, mp2p`
- `ExportSQL` (Access)
- Méthodes manuelles (scripts «ad hoc»)
- Une méthode différente par moteur?

Outils dédiés

- Migrent schéma et données
- Raisonnablement simples
- Lents
  - Regexp
  - Faiblement typés
  - Scripts
  - Multi-thread ?
- Patcher le programme ?

ETL

- Extract-Transform-Load
- Le T ne nous intéresse pas (ou peu)
- Extraction et chargement rapides
- Nombreux connecteurs, extraction et chargement
- Chargements incrémentaux
- Transformations (normaliser, dénormaliser, modifier le format)
- Marteau pilon pour écraser une mouche

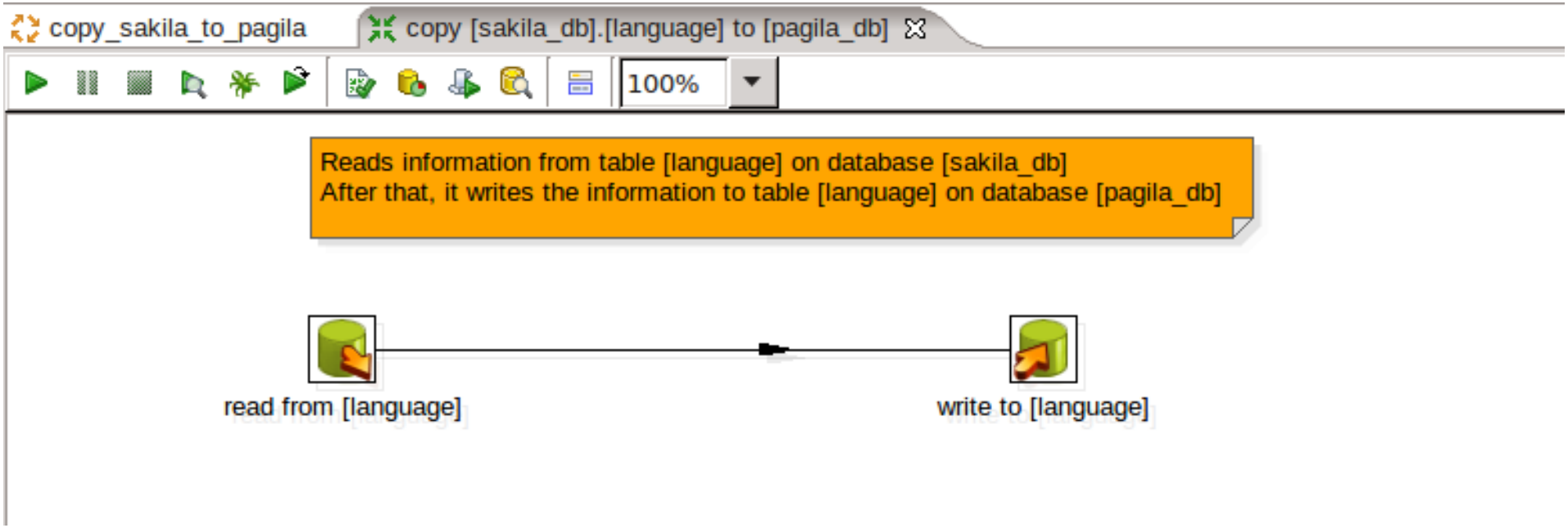
Kettle

- Java
  - JDBC. Se connecte à n'importe quoi
  - Multi-threadé.
  - Fortement typé

Concepts

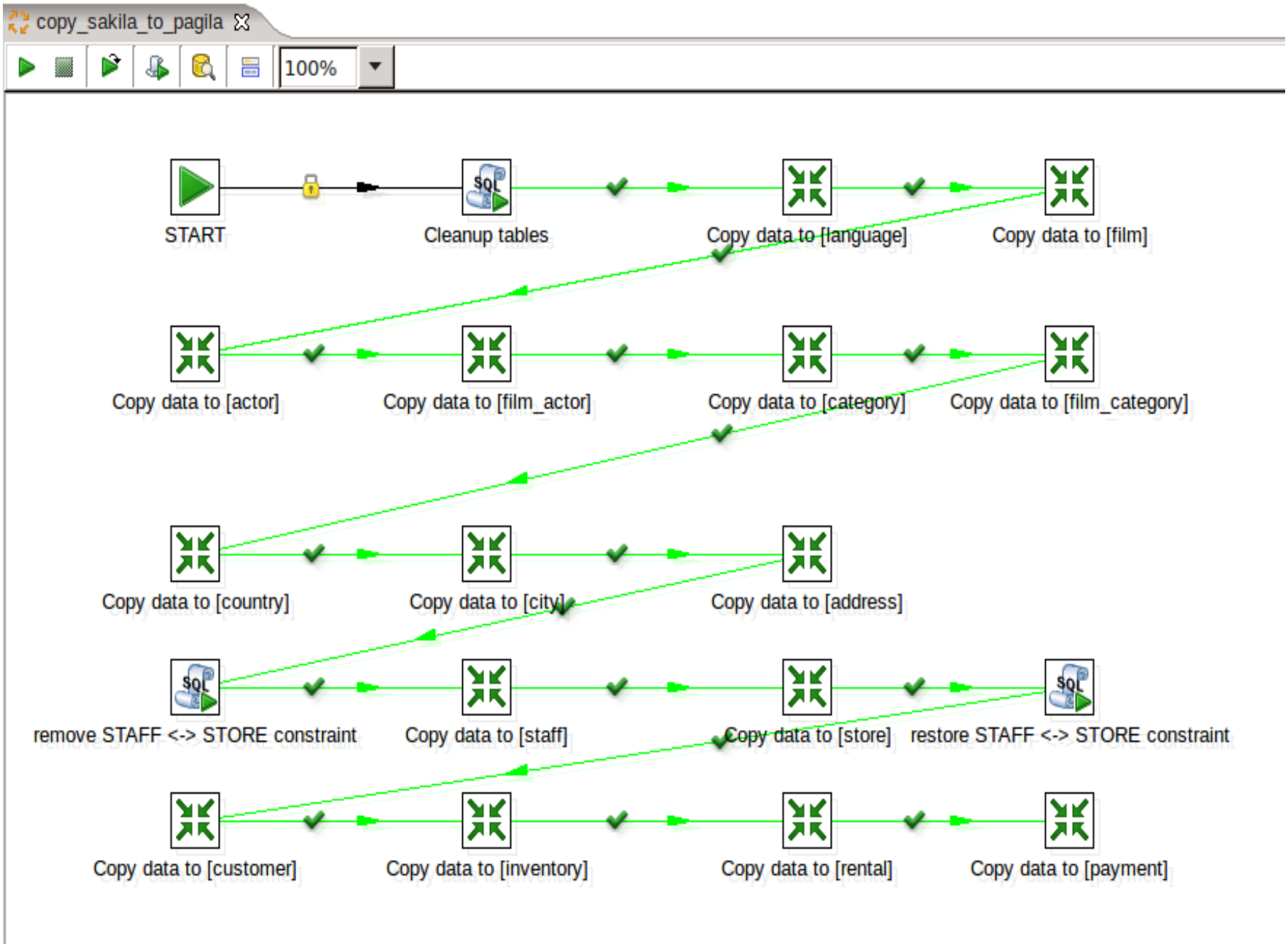
- Transformations
  - Flux : suite d'enregistrements, similaire à des tuples de base (éléments nommés et typés)
  - Étapes traitant un flux (ou le créant)
  - Extraction (input) : récupération depuis base, fichier, webservice…
  - Alimentation (output) : insertion dans base, fichier, webservice, exécution de script…
  - Autres (la plupart) : transformation de flux à la volée

Transformation :



Concepts

- Jobs
  - Exécutés séquentiellement par défaut
  - Lancent d'autres jobs ou des transformations
  - Responsables de l'ordonnancement

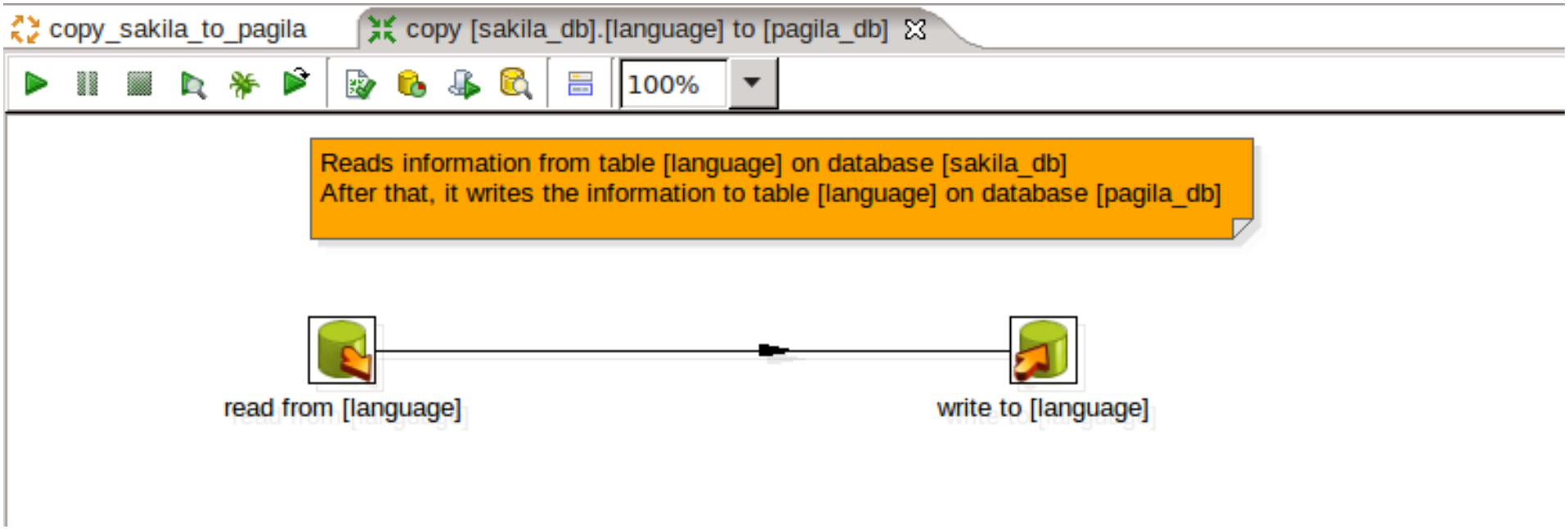


Transformation simple

Recopie de la table language :

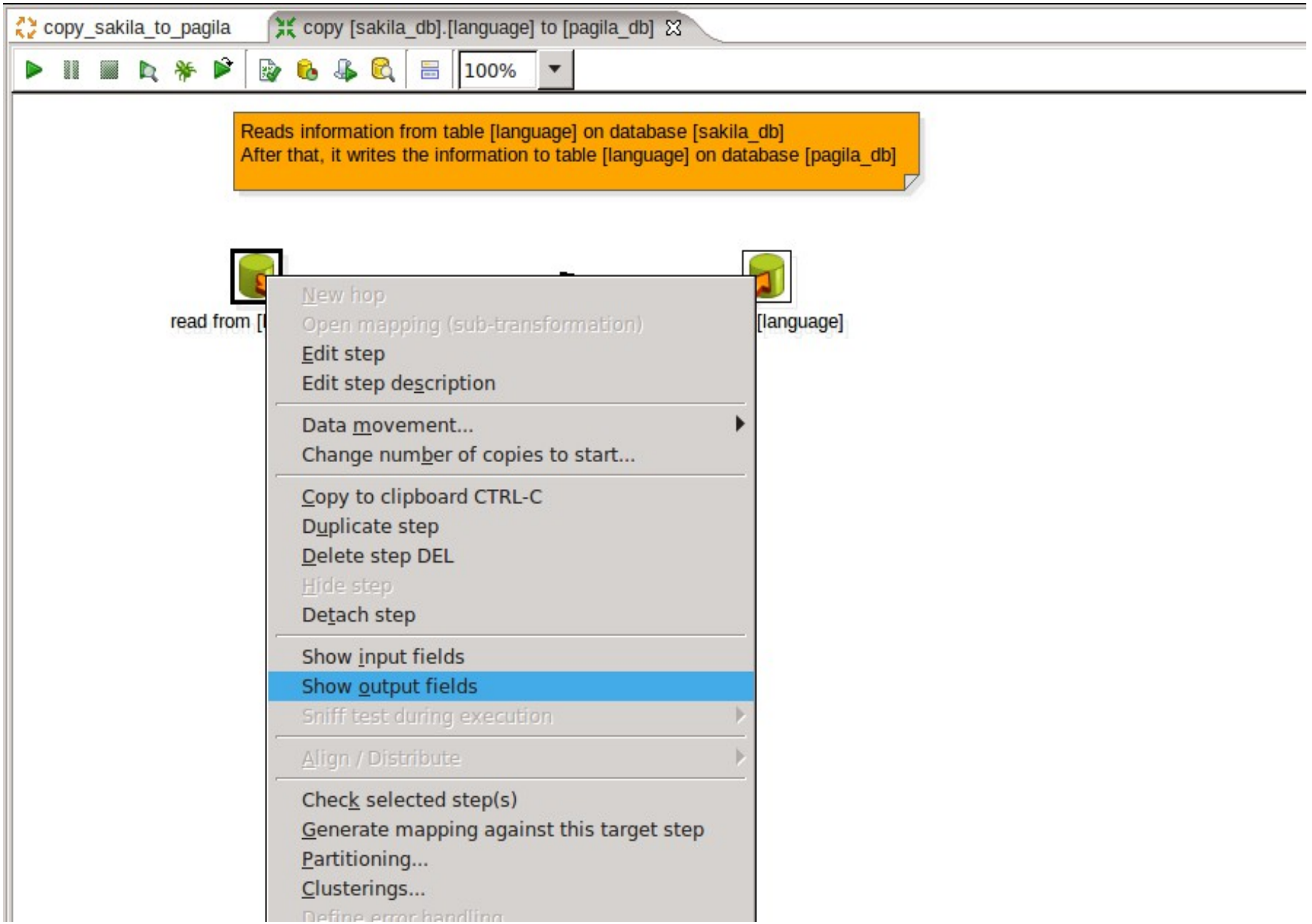
- Table INPUT :
  - `SELECT * FROM language dans sakila`
- Table OUTPUT :
  - Insertion d'enregistrements dans la table language de pagila
- Une flèche («*hop*») entre les deux

Transformation simple

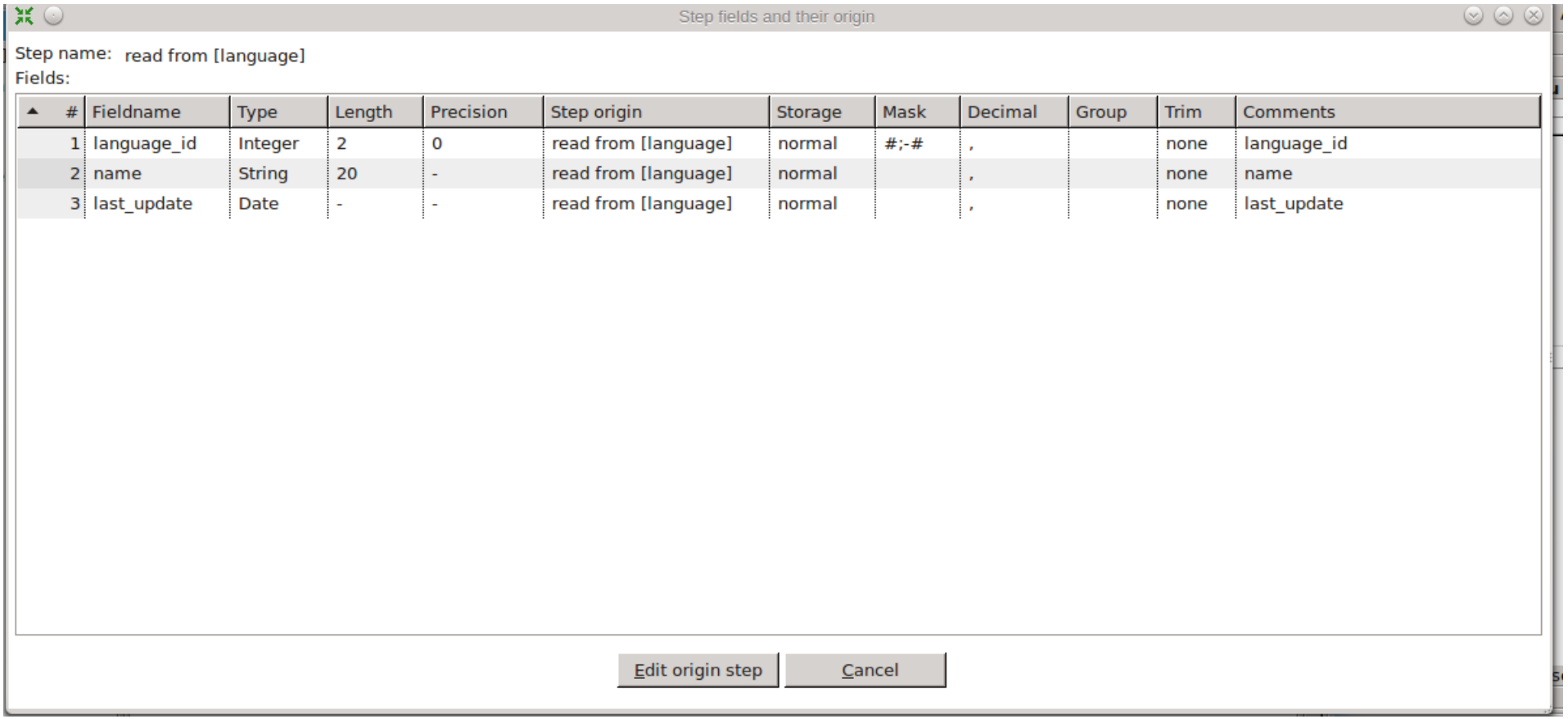


Transformation simple

- «`Show output fields`» (montrer les champs en sortie)
- Tout est typé
- Les informations sont récupérées dynamiquement de la source (au moins quand c'est une base de données)



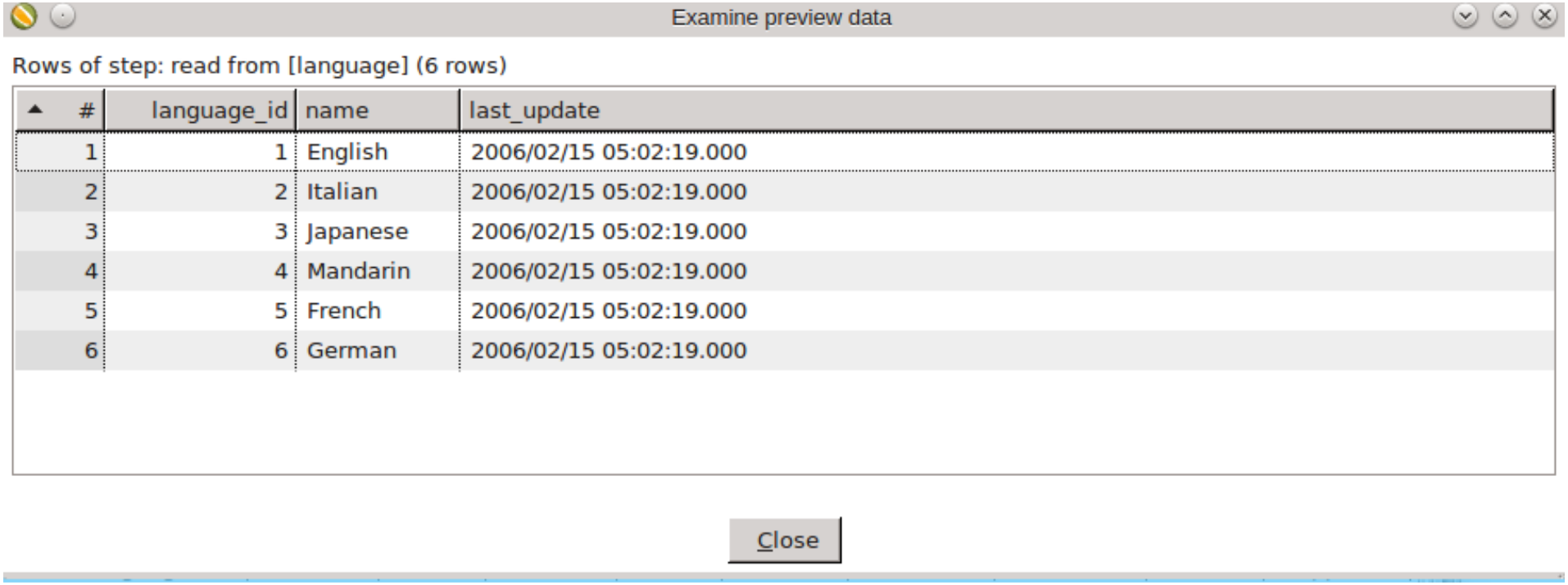
Show output fields



Transformation simple

- On peut avoir un aperçu des données circulant
- «`Show preview`» (menu contextuel)

Preview



Ça peut devenir compliqué

- Différences de schéma entre la source et la destination
  - Types non disponibles au départ ou à l'arrivée (`year(4)` sous MySQL…)
  - Schéma différent (contraintes techniques différentes entre les moteurs par exemple)

Ça peut devenir compliqué

- Types pas ou mal gérés par JDBC
  - Enums
  - Arrays
  - Composites
  - Array de composite…
  - Types géométriques…

=> Transformations, SQL dynamique, `CAST` implicite dans PG, `to_char` à la source…

Exemple : chargement de «films»

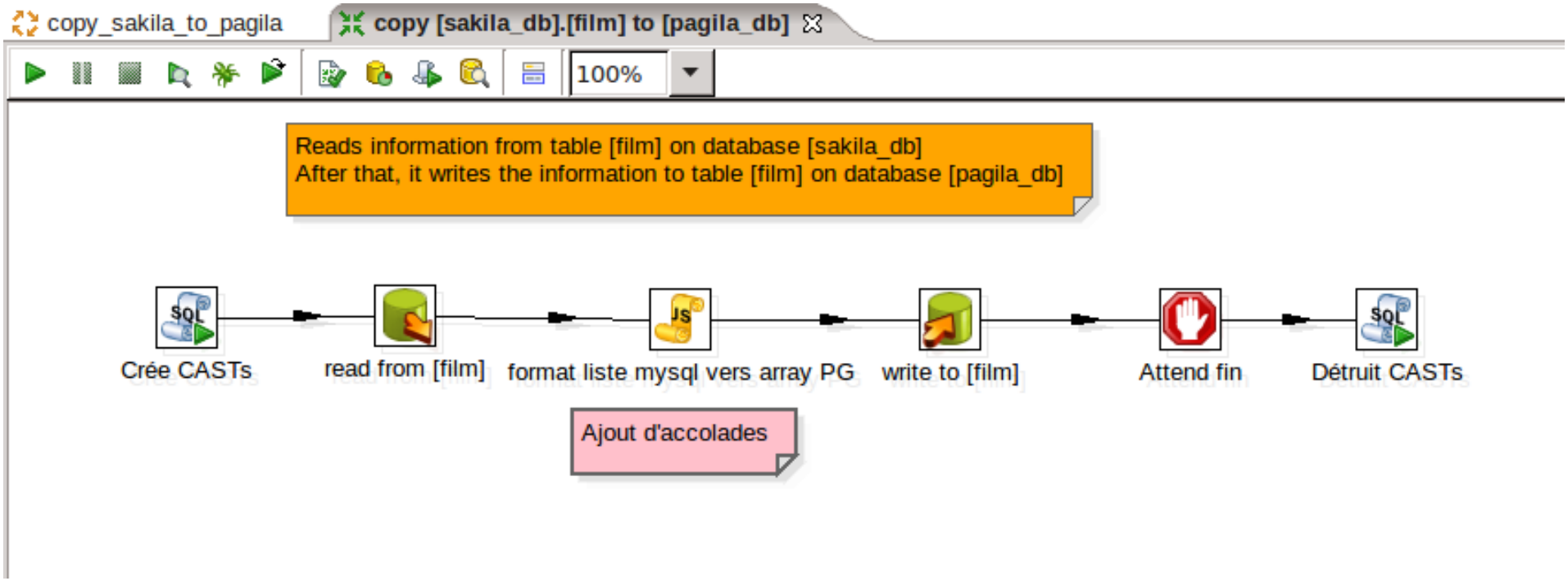
3 champs posent problème :

- special_features:
  - MySQL : «`set('Trailers','Commentaries','Deleted Scenes','Behind the Scenes')`»
  - PG : «text[]»
- rating:
  - MySQL : `enum('G','PG','PG-13','R','NC-17')`
  - PG : `mpaa_rating` (enum aussi)
- release_year :
  - `year(4)` sous MySQL, domaine sur un int sous PG

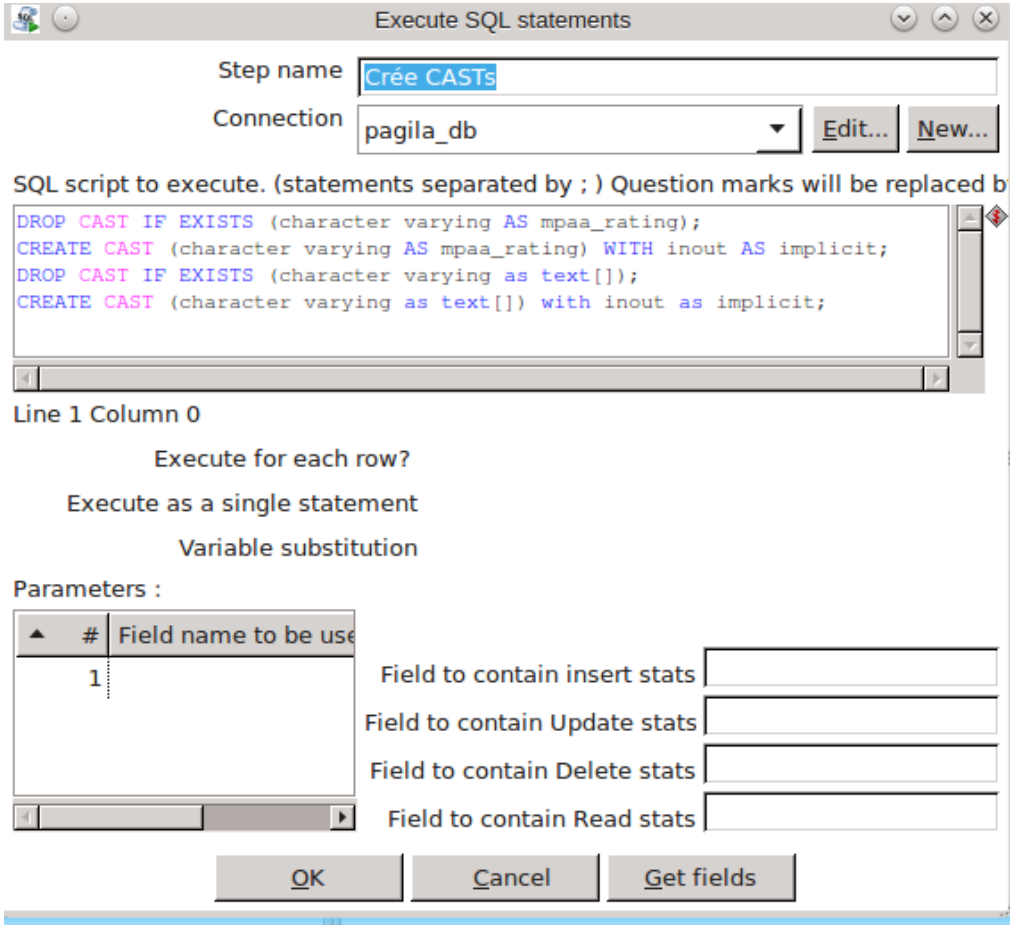
Exemple : chargement de films

- Types non connus par JDBC, qui considère que ce sont des chaînes à l'insertion
  - Création d'un CAST `varchar` -> `mpaa_rating` sous PG pour insertion directe
  - Ajout d'un CAST `varchar` -> `text[]` sous PG
  - Ajout d'un `convert()` : `year(4)` → `int`
  - Conversion du format «`SET`» de MySQL vers `array` PG (ajout d'accolades)
  - Séquencement création CAST -> traitement -> destruction CAST dans la transformation

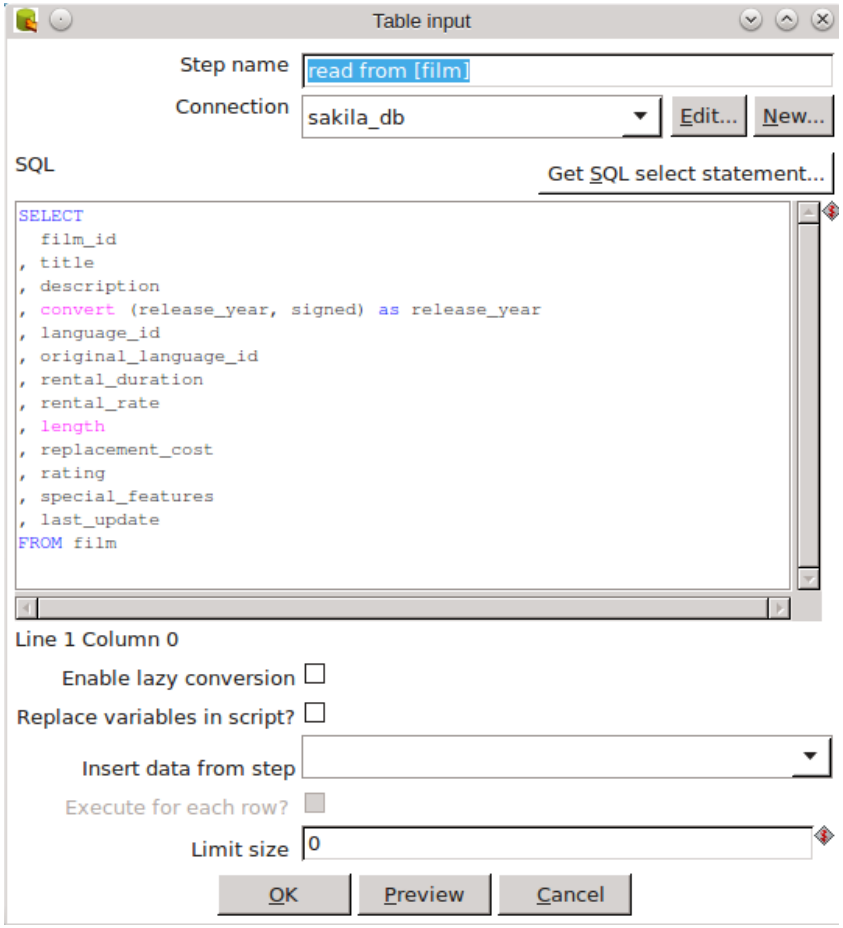
Transformation des films



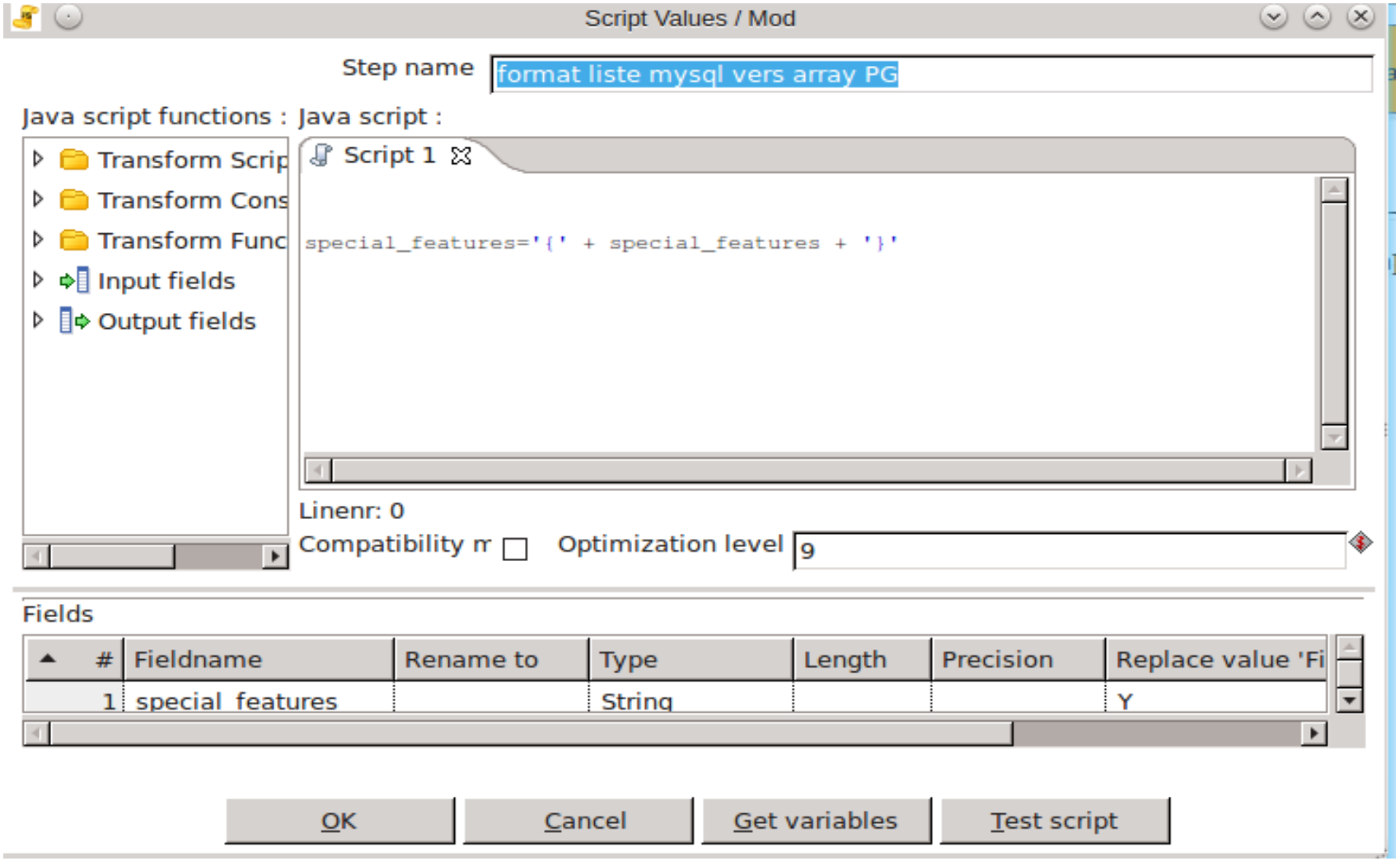
Créer les CASTS



Modifier l'ordre SQL source



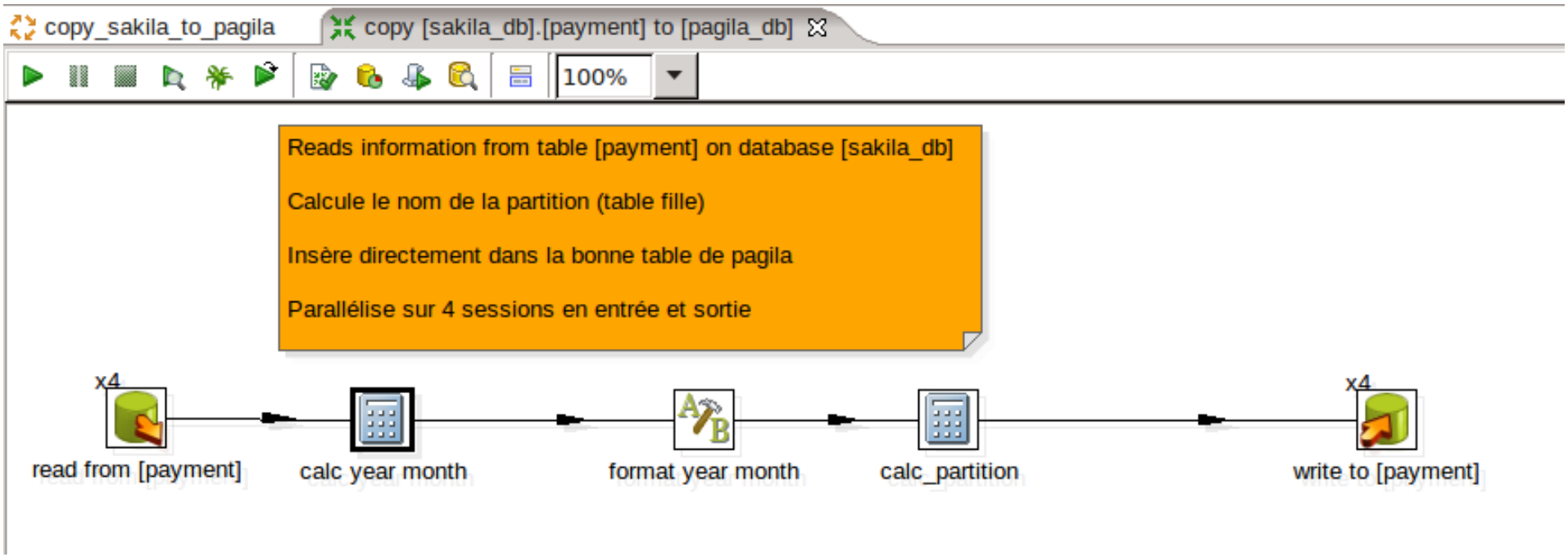
Rajouter des accolades en javascript



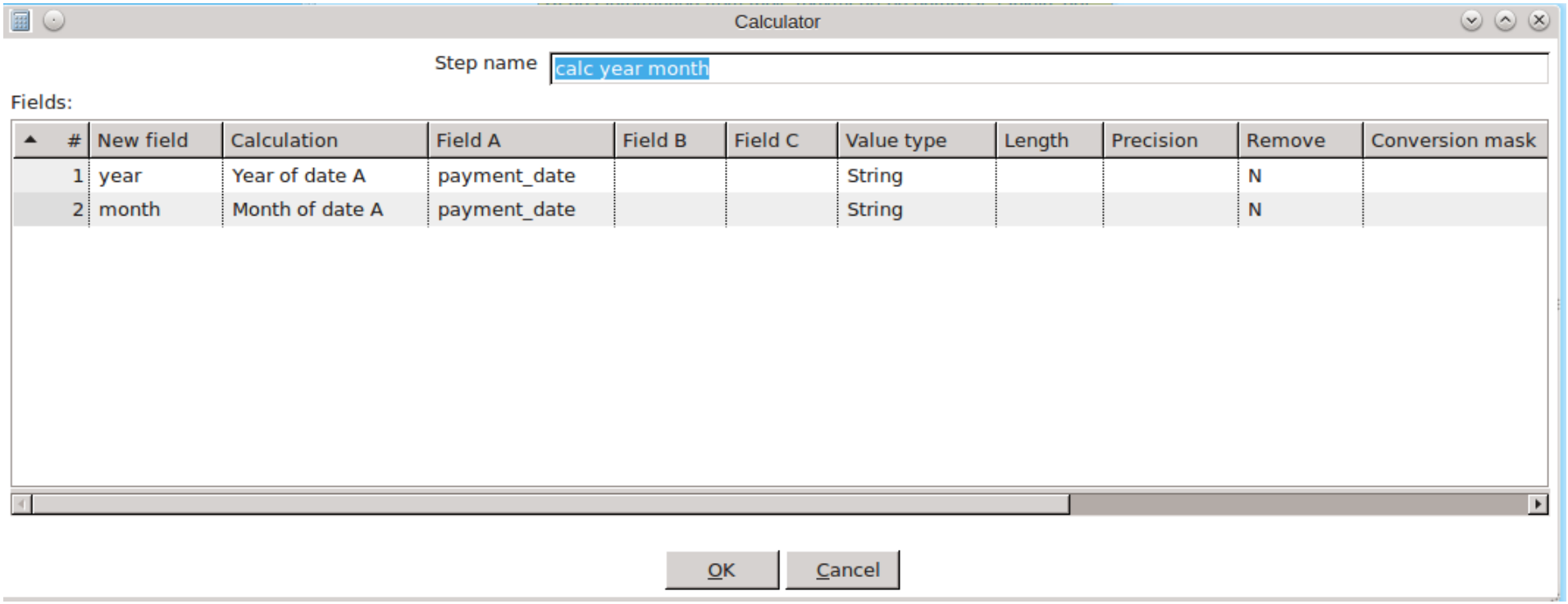
Partitionnement à la volée

- Table payment
  - Non partitionnée dans sakila
  - Partitionnée par mois dans pagila
- Insérer directement dans la bonne partition ?
  - Calculer le nom de la partition pour les enregistrements au fur et à mesure
  - «*Table OUTPUT*» peut utiliser ce champ comme nom de table

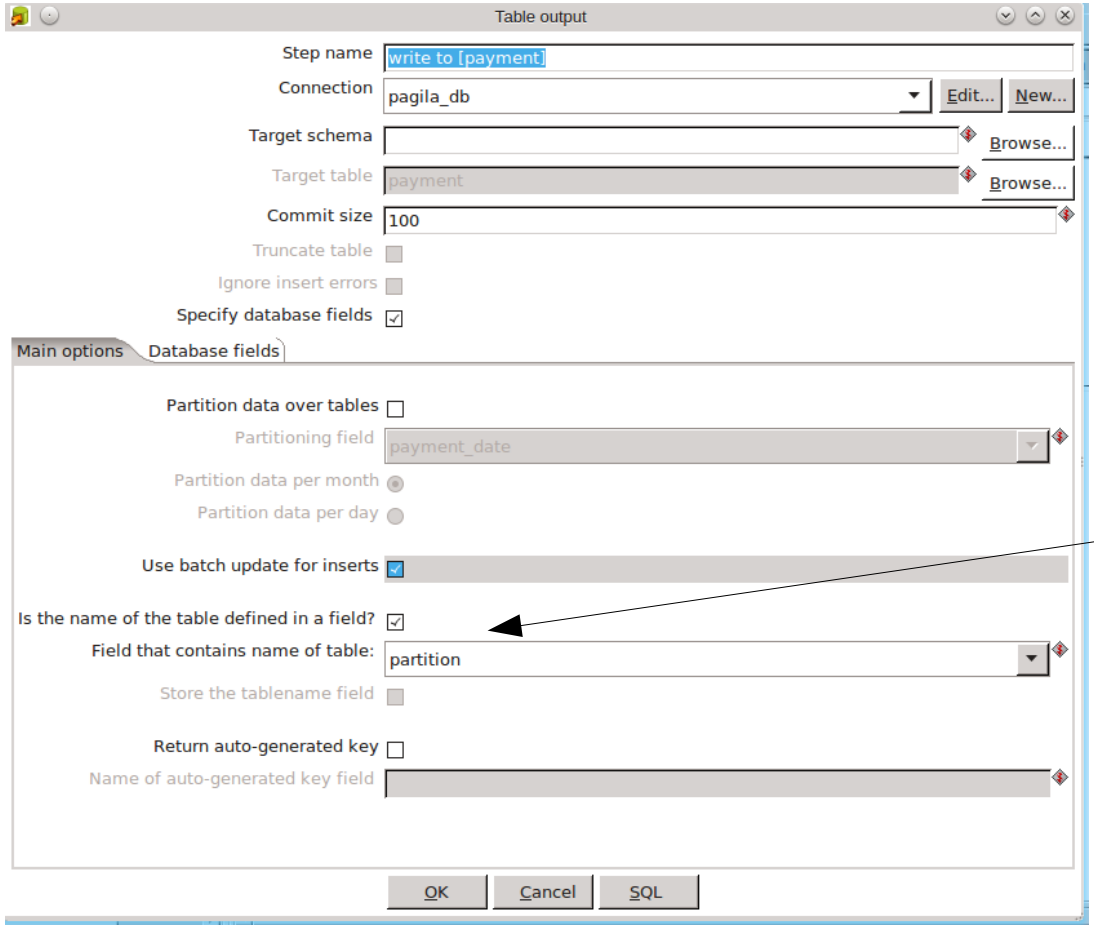
Transformation payments



Calcul de l'année et du mois

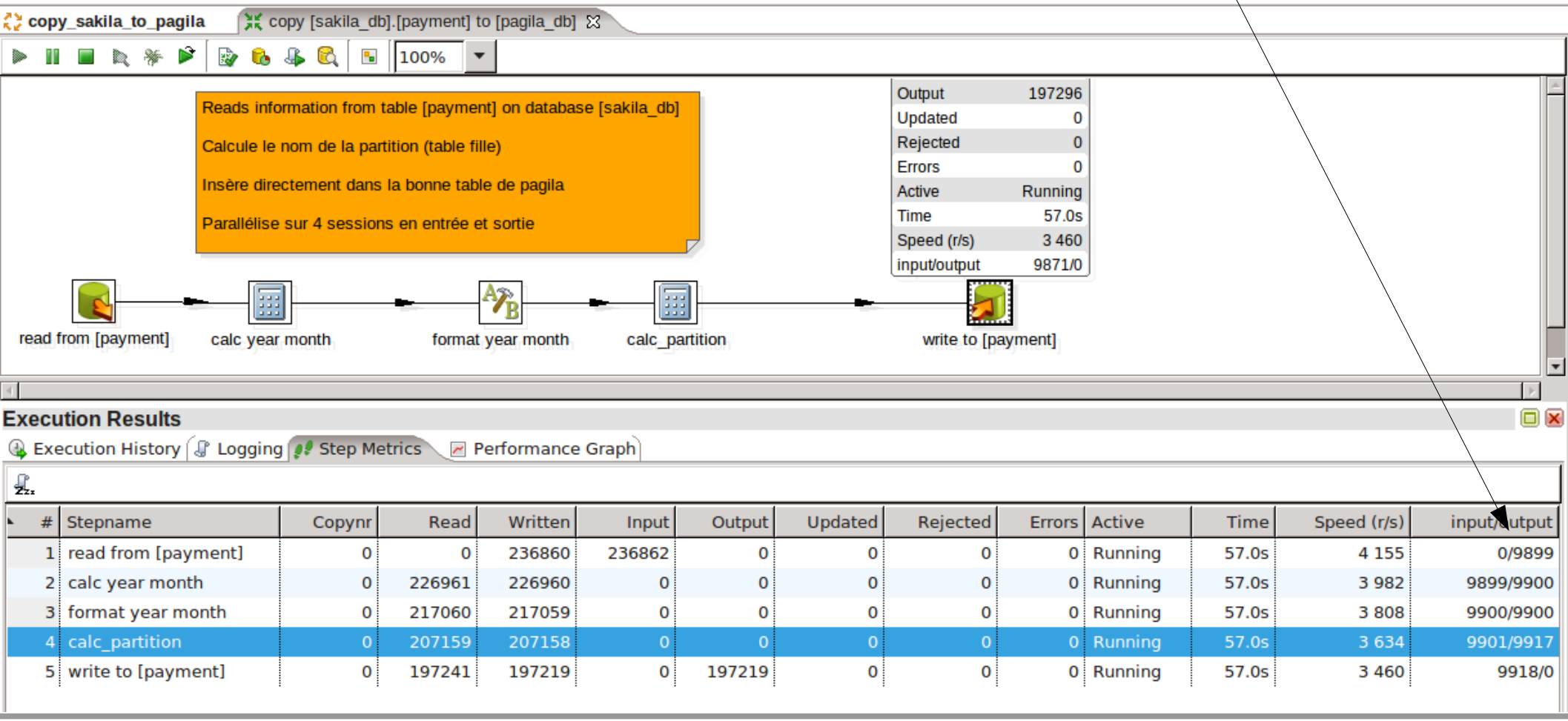


Insertion dans la partition



Et si c'est trop lent ?

Trouver l'étape limitante: «*Step Metrics*»



#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/output
1	read from [payment]	0	0	236860	236862	0	0	0	0	Running	57.0s	4 155	0/9899
2	calc year month	0	226961	226960	0	0	0	0	0	Running	57.0s	3 982	9899/9900
3	format year month	0	217060	217059	0	0	0	0	0	Running	57.0s	3 808	9900/9900
4	calc_partition	0	207159	207158	0	0	0	0	0	Running	57.0s	3 634	9901/9917
5	write to [payment]	0	197241	197219	0	197219	0	0	0	Running	57.0s	3 460	9918/0

Et si c'est trop lent ?

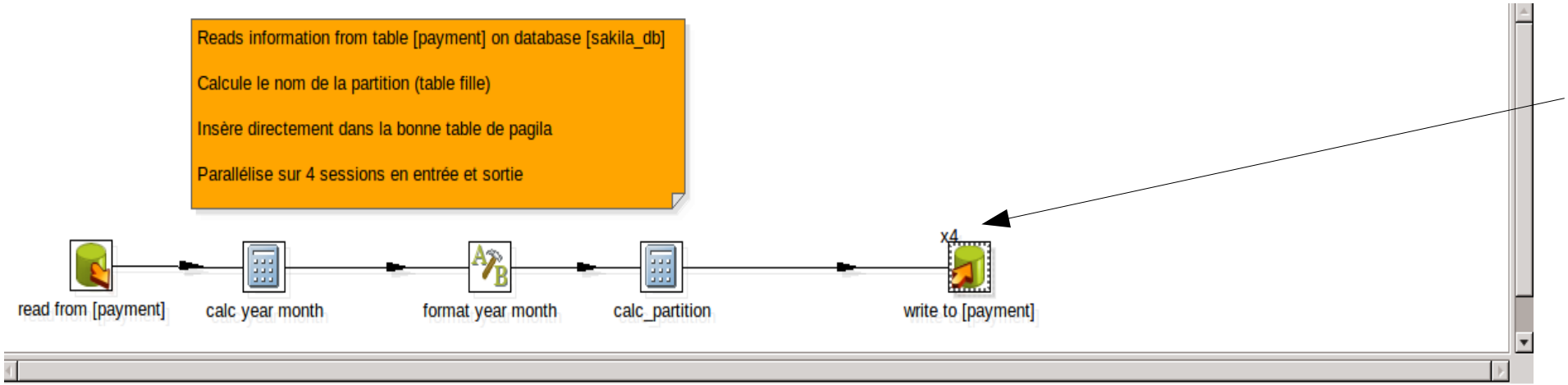
- Optimiser l'étape :
  - Éviter le step javascript, utiliser formula, calculator, ou Java, plus rapides
  - Préférer faire les tris dès l'input (`ORDER BY`)
  - Transtyper dès l'input (`CAST`, `to_char`)
  - Détruire/recréer les index, PK, FK
  - Utiliser `COPY` (ou autre suivant SGBD)
  - Augmenter la «taille des commit» et des «buffers»

Et si c'est trop lent ?

- Paralléliser le nœud limitant
  - «*Change number of copies to start*», disponible pour presque tous les nœuds
  - Plus compliqué pour un nœud INPUT

```
SELECT * FROM ma_table
   WHERE pk%${Internal.Step.Unique.Count}=${Internal.Step.Unique.Number}
```

    - «*Replace variables in script*»
    - Utile si c'est la récupération de l'enregistrement qui est longue (gros blob par exemple)
  - Paralléliser plusieurs opérations longues (chargement de plusieurs tables)



Execution Results

Execution History | Logging | Step Metrics | Performance Graph

#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/outp
1	read from [payment]	0	0	802543	802545	0	0	0	0	Running	38.0s	21 147	0/98
2	calc year month	0	792644	792643	0	0	0	0	0	Running	38.0s	20 886	9899/99
3	format year month	0	782743	782742	0	0	0	0	0	Running	38.0s	20 625	9900/99
4	calc_partition	0	772787	772786	0	0	0	0	0	Running	38.0s	20 363	9955/337
5	write to [payment]	0	185637	185395	0	185395	0	0	0	Running	38.0s	4 891	7560
6	write to [payment]	1	184403	184320	0	184320	0	0	0	Running	38.0s	4 859	8794
7	write to [payment]	2	183196	183159	0	183159	0	0	0	Running	38.0s	4 827	10000
8	write to [payment]	3	185814	185662	0	185662	0	0	0	Running	38.0s	4 896	7382

Passage de 6m20s à 1m45s

Il faut tout faire à la main ?

- Wizard «*COPY TABLES*»
  - Crée job recopiant toutes les tables sélectionnées
  - Et ordres SQL pour créer/modifier les tables pour les rendre conformes (préférer un script de schéma personnalisé)
  - Pas d'index, de contrainte, de séquence, de trigger, de PL…
  - Ne gère pas l'ordre des contraintes. Préférer les créer après le chargement des données
  - Crée un job initial, à affiner

Les erreurs ?

- Kettle trace les erreurs, au format Java (pile d'exceptions)
  - Différents niveaux de log
  - Plus rapide et naturel pour un DBA de regarder la log de la base
- Analyse simple de ce qui entre ou sort d'un nœud (show input/output fields, preview…)
- Attention aux types. On peut forcer les types avec le nœud 'SELECT values'

Les erreurs ?

- Une erreur dans un nœud déclenche l'erreur de la transformation
  - Sauf si on crée un flux d'erreur pour ce nœud
- Une erreur dans un job déclenche la remontée au job parent, ou arrête le job s'il est le parent
  - Sauf si on crée un chemin pour gérer l'erreur de ce nœud

Quelques pièges

- Dans «kettle.properties», positionner `KETTLE_EMPTY_STRING_DIFFERS_FROM_NULL=Y`
- Dans les paramètres des connexions, cocher «*supports boolean*» (pas par défaut)
- Ne pas oublier de positionner les valeurs des séquences