# Rapport d'Entrepôt de données

## Talend Open Studio

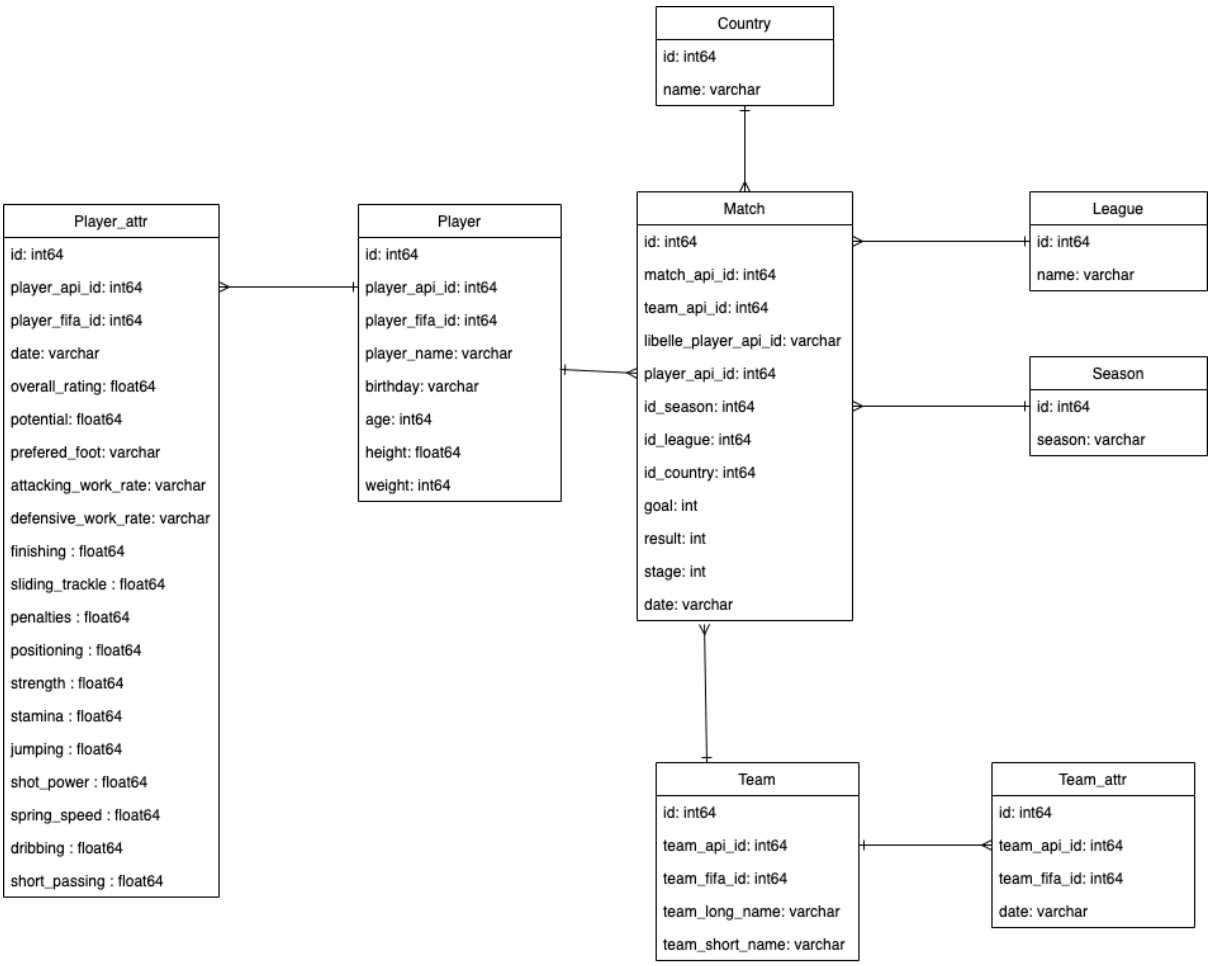
### 1. Démarche

On a suivi 4 étapes dans notre démarche principalement. Nous avons d'abord étudier la source de données pour comprendre les colonnes des tables. Après, on modélise notre datamart. Ensuite, les jobs talend sont faits pour alimenter nos tables de dimension et de fait. Finalement, on utilise notre datamart pour faire le dashboard powerbi qui répond aux questions d'analyse du sujet.

### 2. Le modèle du datamart

Puisqu'on a besoin d'analyser par axe saison, on ajoute une nouvelle dimension saison. Alors, notre modèle de datamart a 7 tables de dimension, player, player attributes, team, team attributes, country, league et saison, une seule table de fait MATCH. La table player attributs et la table player sont liés par la player_api_id. De plus, la colonne âge a été ajoutée sur la table de dimension player pour satisfaire les analyses des caractéristiques des joueurs. Le lien entre la table team et team attributs est créé par la colonne team_api_id.

Dans la source de la table MATCH, on a 22 colonnes de player id. Pour faciliter notre relations entre la table de fait et le dimension player, on a pivoté les 22 colonnes, home_player_1_11, away_player_1_11 en lignes. D'ailleurs pour ne pas perdre d'information, on a ajouté une colonne libelle_player_api_id, dont valeurs sont les nom des 22 colonnes, pour clarifier une ligne correspond à quel numéro de joueur à la away ou homme équipe. Alors, les clés étrangères dans la table de fait sont : match_api_id, player_api_id, id_season, id_league et id_country, ce qui nous aident à créer des relations entre les dimensions. De plus, nous devons analyser les matchs gagnés, alors on a ajouté une colonne "result" sur la table de fait pour distinguer si le match est gagné ou pas. (Détaillé logique est dans la partie explication des job Talend.)




## 3. La qualité des données

La qualité des données est relativement bonne. Mais il y a encore des soucis de qualité, surtout sur la table de match.

1) Beaucoup de nulles dans la table Match. Par exemple, shoton, shotoff, home_player_x1_11, home_player_1_11 ( ex: 1224 home_player_1 null sur 25979 de ligne en total, 5% de null), away_player_1_11… Pour certaines colonnes on a presque une moitié de null (goal).

2) Les colonnes, goal, shoton, shotoff, foulcommit, card, cross, cormer et possession sont des scripts html complexes. Ce n'est pas facile à utiliser.

3) Type de données pas correct, surtout les colonnes date et ids sont string.

4) Il y a des colonnes qui ne servent pas beaucoup ou dont on ne connaît pas le sens. Par exemple, home_player_X1_11, home_player_Y1_11, BWH, BWA…

## 4. Problèmes rencontrés

Pendant la modélisation, au début, on ne sait pas comment gérer les relations entre table match et players. Puisque dans la source de données, nous avons 22 colonnes de player id et créer 22 liens entre deux tables n'est pas un bon choix. A la fin, on a décidé de pivoter les colonnes en lignes.

Pour créer des jobs Talend, nous ne pouvons pas lire les tables d'un database SQLite et écrire une autre table du database en même temps. Il y a une exception databases is locked, comme l'image suivante. Donc, au final on crée d'abord un fichier output. Après, on utilise ce fichier pour alimenter nos résultats dans database.

```
Starting job Alimenter_KPI_attr_player_par_team_season at 19:59 12/01/2023.
[statistics] connecting to socket on port 3987
[statistics] connected
Exception in component tDBOutput_4 (Alimenter_KPI_attr_player_par_team_season)
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:288)
    at org.sqlite.DB.exec(DB.java:68)
    at org.sqlite.Conn.commit(Conn.java:172)
    at local_project.alimenter_kpi_attr_player_par_team_season_0_1.Alimenter_KPI_attr_player_par_team_s
    at local_project.alimenter_kpi_attr_player_par_team_season_0_1.Alimenter_KPI_attr_player_par_team_s
    at local_project.alimenter_kpi_attr_player_par_team_season_0_1.Alimenter_KPI_attr_player_par_team_s
[statistics] disconnected

Job Alimenter_KPI_attr_player_par_team_season ended at 20:01 12/01/2023. [Exit code  = 1]
```

## 5. Les jobs Talend expliqués

### 1) Alimentation tables de dimension

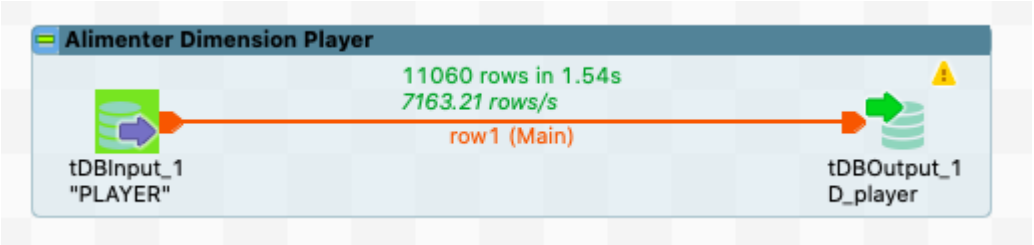
#### a) Dimension player

Nous avons alimenté d'abord des tables de dimension. On a besoin d'ajouter une colonne age dans la dimension player, ce qui va nous aider dans les prochaines étapes d'analyse. L'âge a été calculé par requête sql : SUBSTR(CURRENT_DATE,1,4) - CAST( SUBSTR ( PLAYER.birthday,1,4 )as integer) as age. Requête complète est comme suivante :

```
"SELECT PLAYER.\"id\" as id,
                PLAYER.player_api_id as player_api_id ,
                PLAYER.player_name as player_name,
                PLAYER.player_fifa_api_id as player_fifa_api_id,
                PLAYER.birthday as birthday,
                SUBSTR(CURRENT_DATE,1,4) - CAST(SUBSTR(PLAYER.birthday,1,4)as integer) as age,
                PLAYER.height as height,
                PLAYER.weight as weight
```

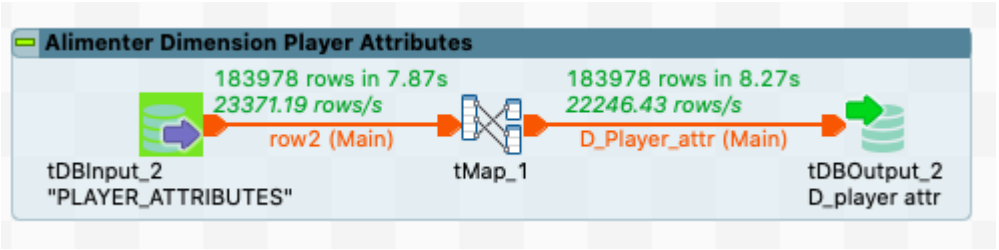
FROM PLAYER"

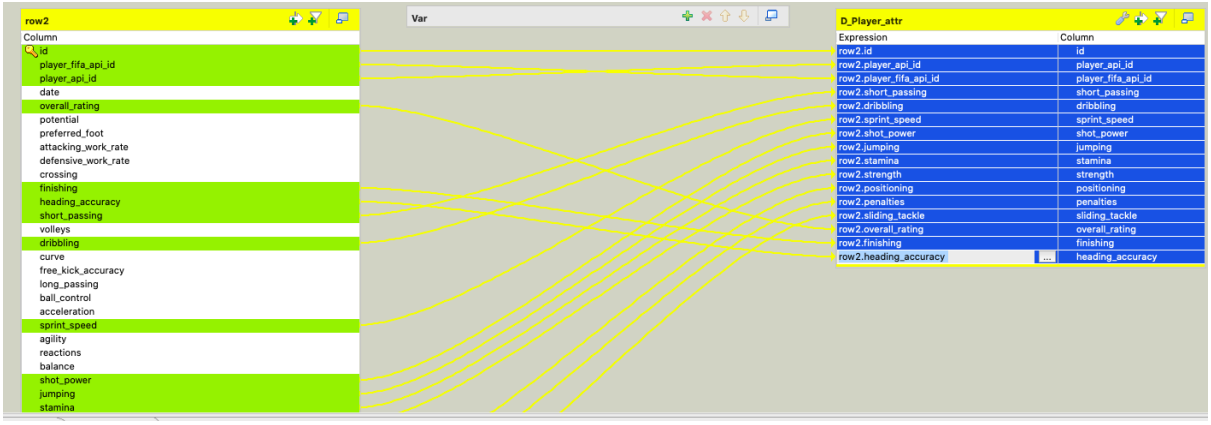
Après, on alimente toutes les données de la source PLAYER avec la nouvelle colonne âge dans notre nouvelle table D_player du nouveau database “Datamart”.




### b) Dimension player attributs

Concernant la dimension player attributs, dans la source, il y a pas mal de colonnes que nous n’utilisons pour ce sujet. Donc, on a utilisé tMap pour choisir que les attributs qu’on utilise. Ensuite, stocker les colonnes choisies dans la nouvelle table D_player-attr.



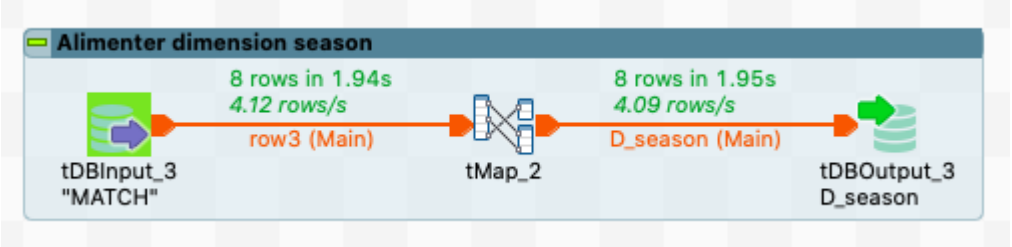



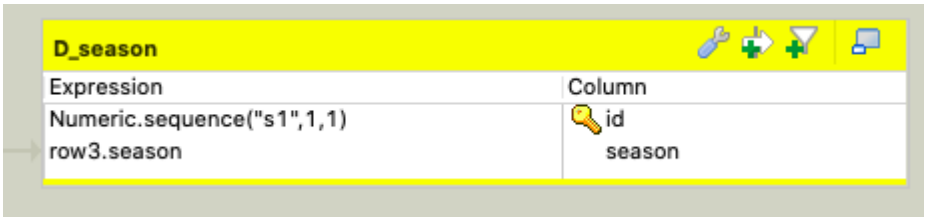

*Tmap - select les colonnes*

### c) Dimension season

Nous avons ajouté une nouvelle dimension saison. Cette dimension doit contenir les saisons uniques avec les ids correspondant. Alors, on a utilisé la requête sql pour sélectionner les valeurs de saison unique à partir de la source MATCH(SELECT distinct(MATCH.season) as

season FROM MATCH). Après, le tMap a été utilisé pour ajouter la colonne id auto increment, ce qui a généré par la fonction Numeric.sequence dans Talend.

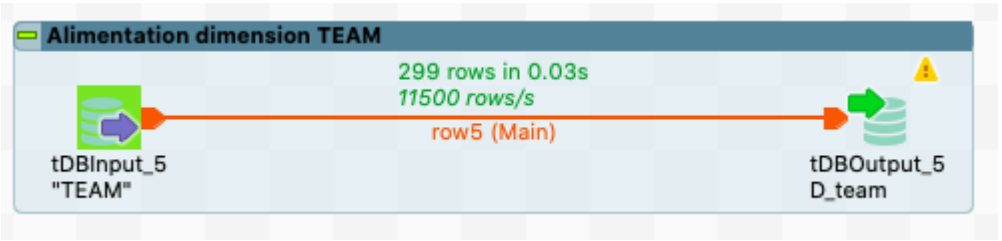



*Tmap - génère la colonne id auto_increment*

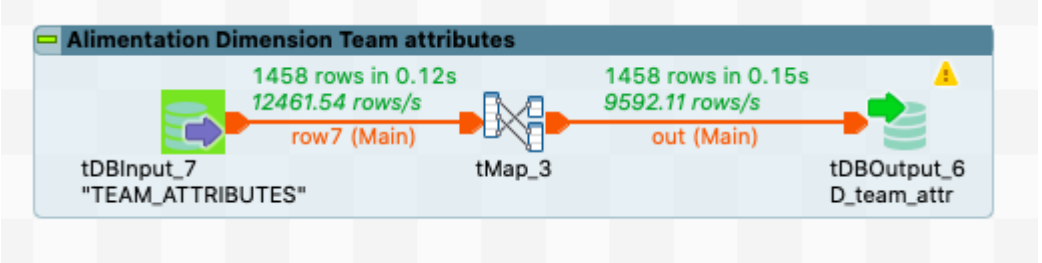
### d) Dimension team

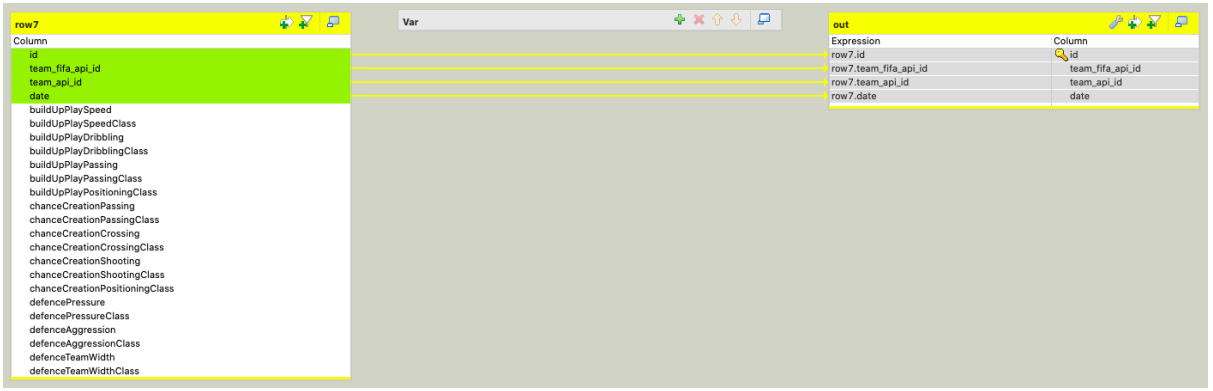
Par rapport à la dimension team, nous n'avons pas modifié la source. Donc, on alimente directement la table TEAM dans la nouvelle table D_team de notre datamart.




### e) Dimension team attributes

Le traitement pour la dimension team attributs est le même que celui de la dimension player attributs. On utilise Tmap pour sélectionner les colonnes qu'on va utiliser dans ce sujet.

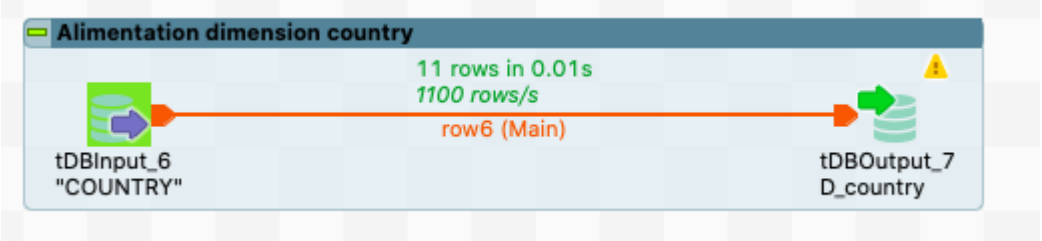





*Tmap - select les colonnes*

### f) Dimension country

Par rapport à la dimension country, nous n'avons pas modifié la source. Donc, on alimente directement la table COUNTRY dans la nouvelle table D_country de notre datamart.

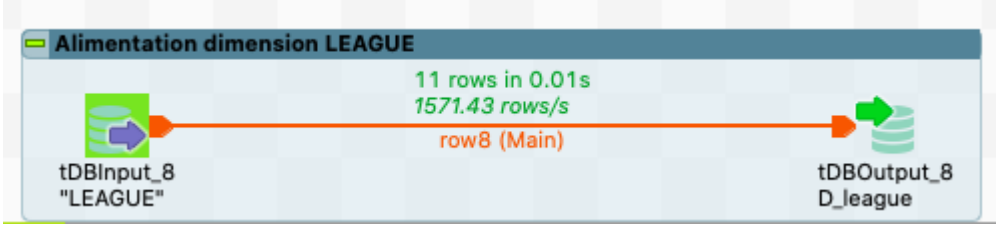



### g) Dimension league

Puisqu'on a déjà la dimension country, on n'a plus besoin du id country dans la table league. Alors, nous utilisons la requête SQL pour supprimer cette colonne, comme suivant :

```
"SELECT LEAGUE.\"id\",LEAGUE.name  FROM LEAGUE"
```

On alimente la dimension league avec toutes les informations dans la source LEAGUE, sauf la colonne country id.

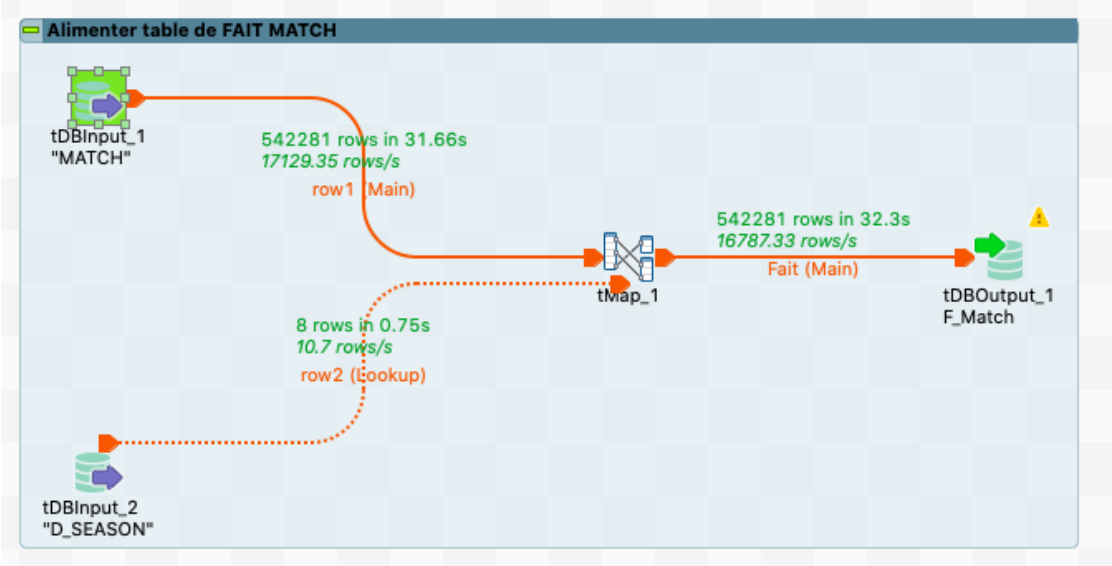



## 2) Alimentation tables de fait

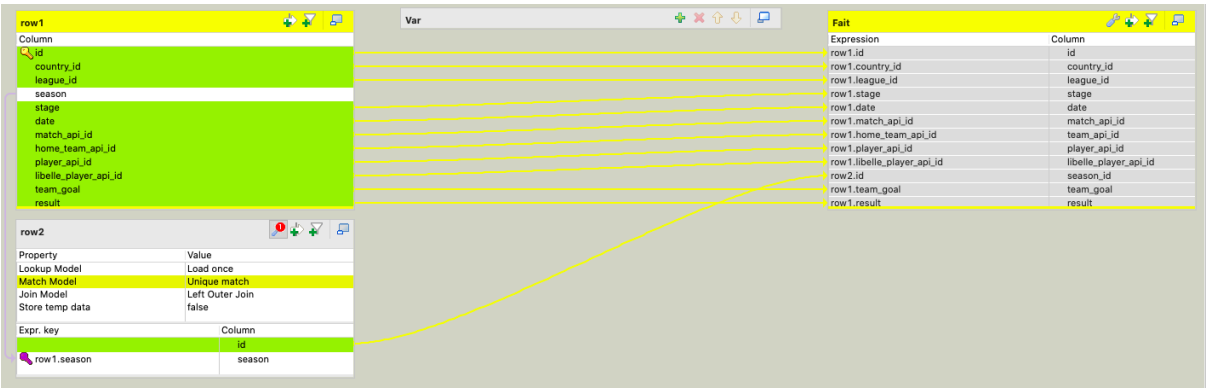
Nous n'avons qu'une seule table de fait match, ce qui est alimenté par la source MATCH. Dans la source, il y a 22 colonnes de player id, 11 pour home player, 11 pour away player.

Pour faciliter notre analyse de players dans les prochaines étapes, on a pivoté les colonnes players vers des lignes en utilisant requête SQL. D'ailleurs une colonne libellé player_id a été ajoutée pour ne pas perdre d'information. De plus, dans la source, on a plusieurs colonnes qu'on utilisera pas, alors dans la requête on ne choisit que les informations dont on a besoin : id, country_id, league_id, season, stage, date, match_api_id, team_api_id, player_api_id et team_goal…

Ensuite, la colonne résultat a été ajoutée, si la ligne est home team et avec home team goal plus que le ways team goal, on donne le résultat en 1 sinon en 0. Les lignes away team ont la même logique. (Exemple code de home team line : CASE WHEN MATCH.home_team_goal > MATCH.away_team_goal THEN 1 ELSE 0 END as result)



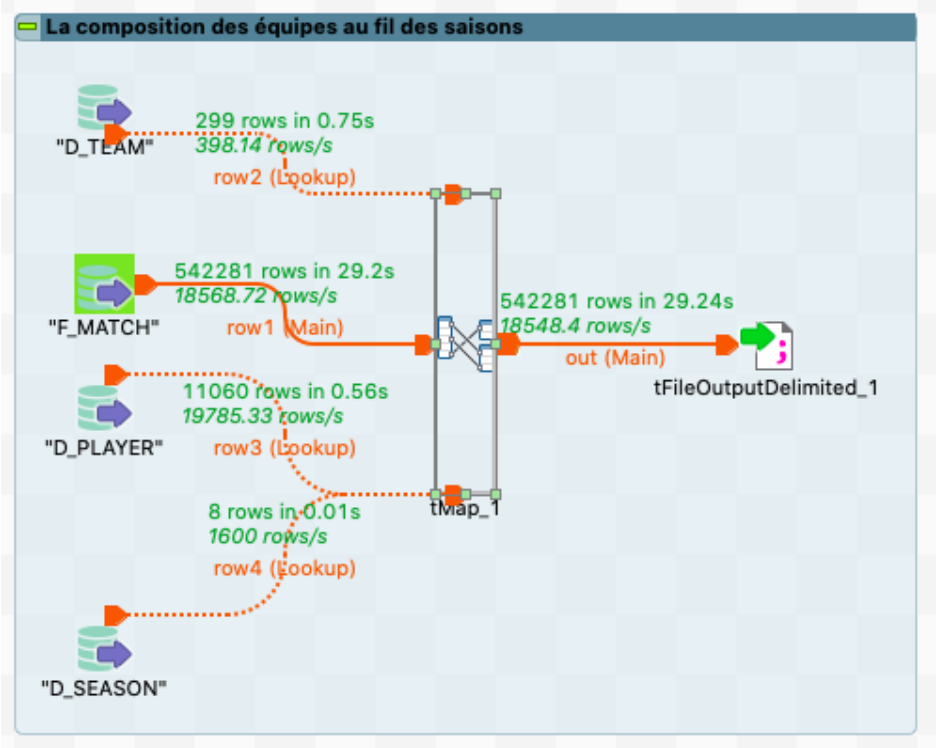

De plus, nous avons la dimension saison. Donc, on doit changer la colonne saison par id_season comme la clé étrangère de la dimension. Un tMap est utilisé pour récupérer les season_ids correspondants dans la table D_season.




### 3) Alimentation tables KPI

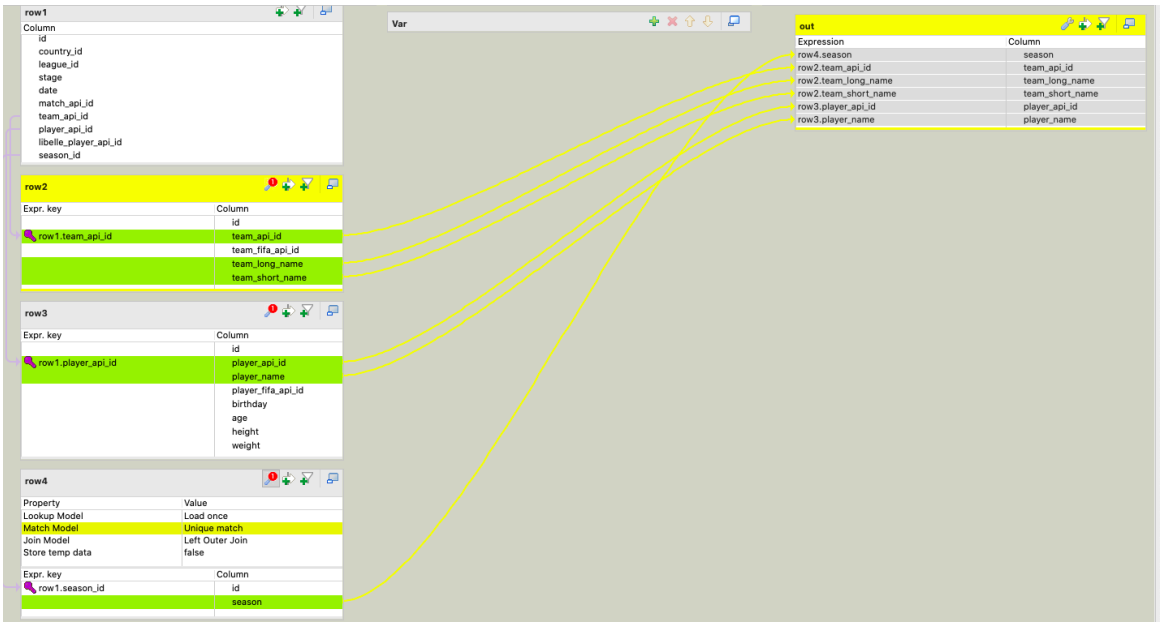
#### a) La composition des équipes au fil des saisons

On avait compris la composition des équipes au fil des saisons comme la composition de joueurs des équipes pour chaque saison. Donc, on a nos axes d'analyse, les dimensions : saison, équipe et player, et notre table de fait. Pour joindre les 4 tables, le composant tMap a été utilisé.

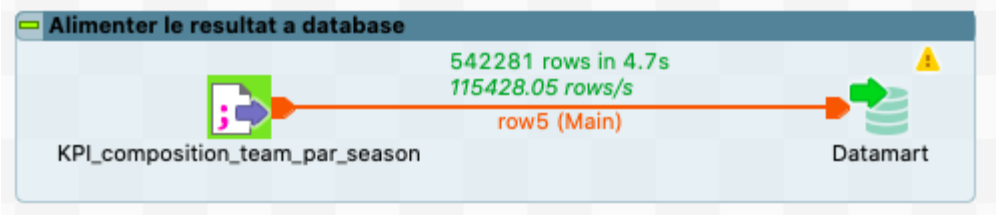



Les jointure entre la table de fait et les dimensions sont left join et unique match. Puisque les tables de fait contiennent des records uniques.

Les outputs sont retirés par les tables de dimension, parce qu'on a besoin des information parlantes, players name, équipe name ect, au lieu des ids.



De plus, nous ne pouvons pas lire les tables d'un database SQLite et écrire une autre table en même temps. Donc, on crée d'abord un fichier output. Après, on utilise ce fichier pour alimenter nos résultats dans database.



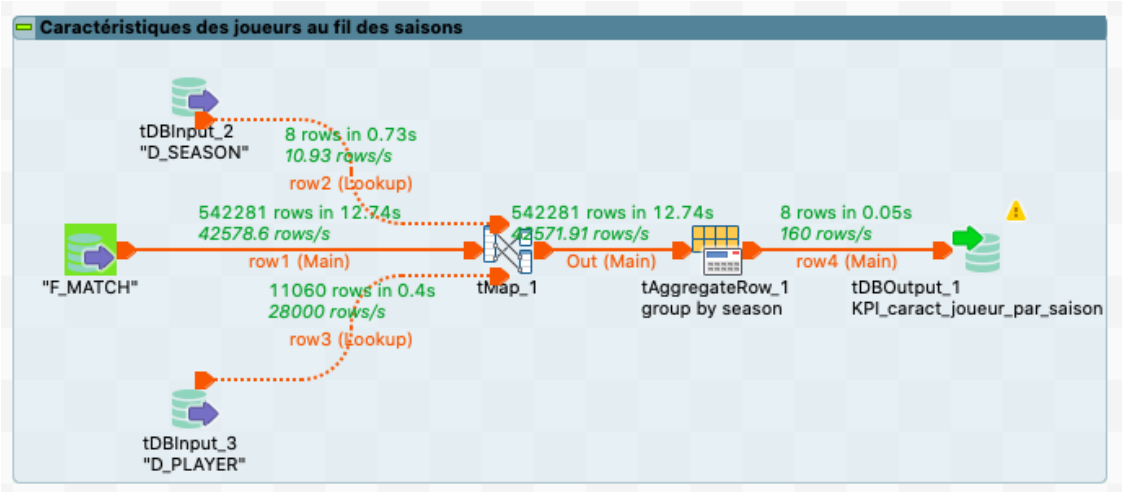

### b) Caractéristiques des joueurs au fil des saisons

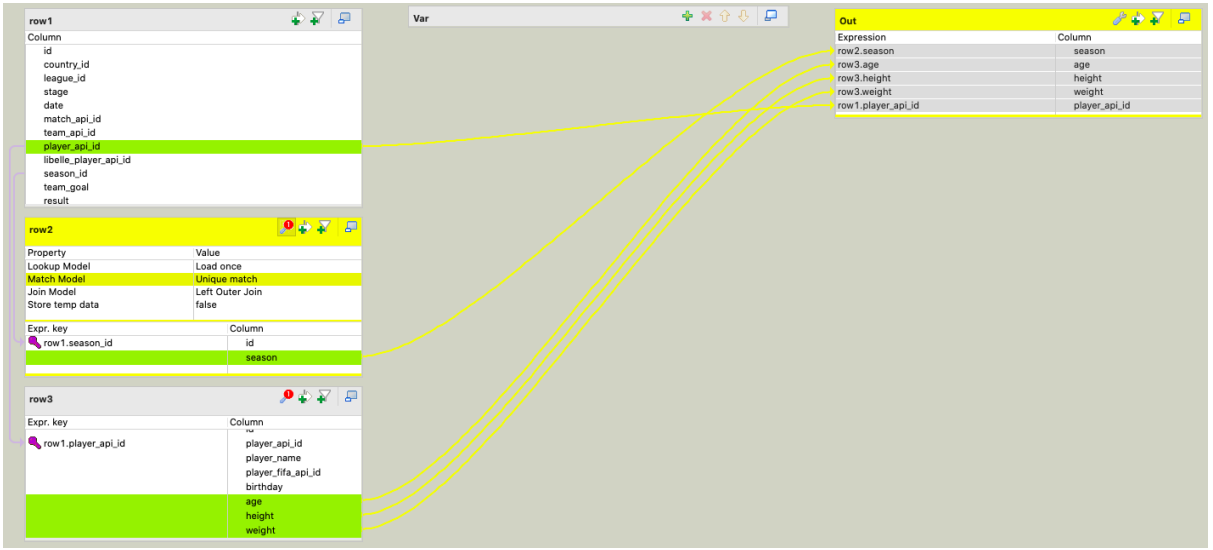
Pour récupérer les caractéristiques des joueurs au fil des saisons, on a deux axes d'analyse joueur et saison, de plus notre fait.

Un tMap a été utilisé pour joindre les trois tables. Les paramètres sont toujours left join et match unique. On récupère les caractéristiques âge, poids et taille de la dimension player, les saisons viennent de la dimension saison.

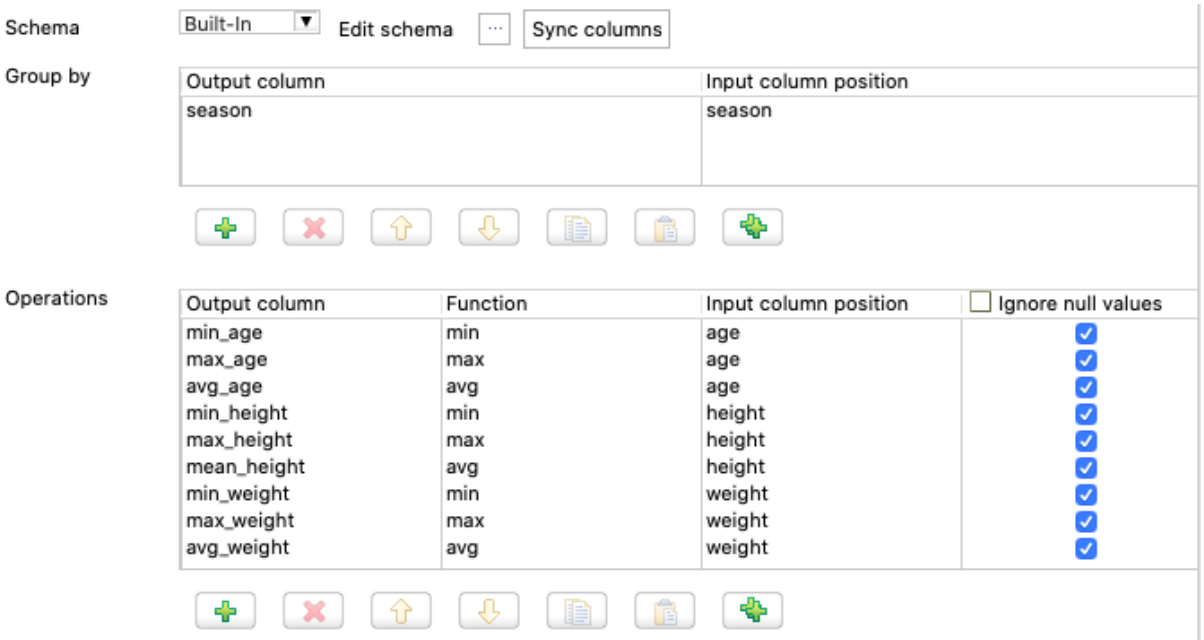
Ensuite, on utilise un tAggregateRow pour agréger les données output du tMap par saison et on calcule les min, max et moyenne des trois caractéristiques.

Finalement, les résultats agrégés sont enregistrés dans notre nouvelle table KPI.





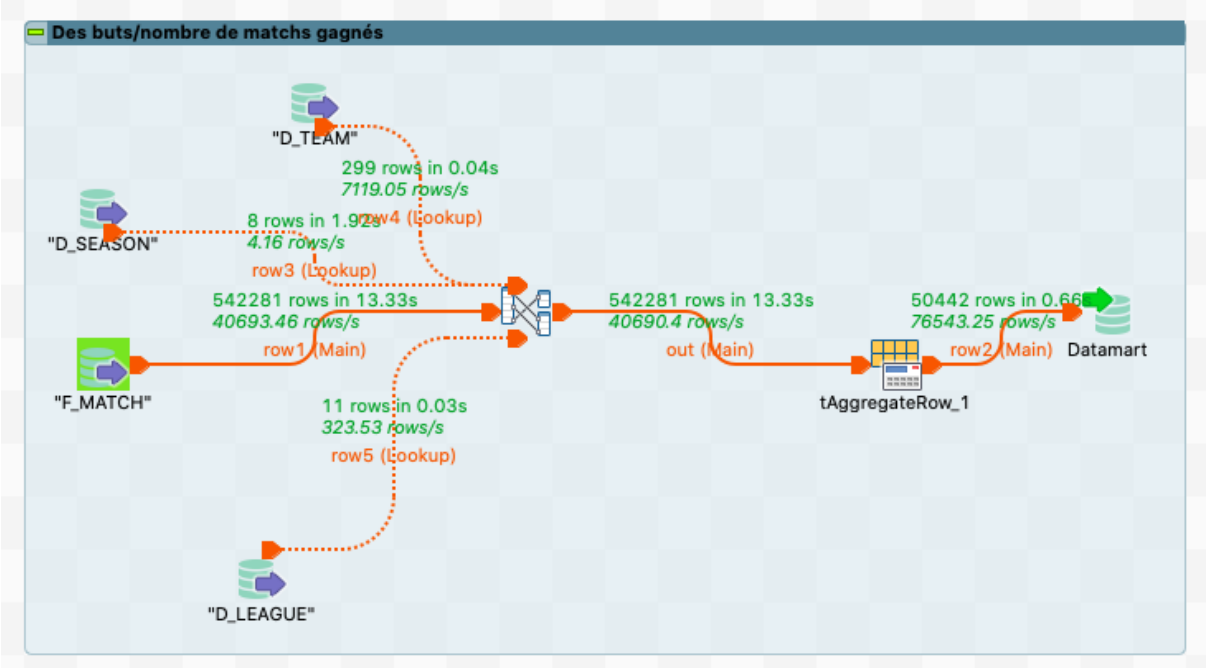

*tMap - jointure et select*

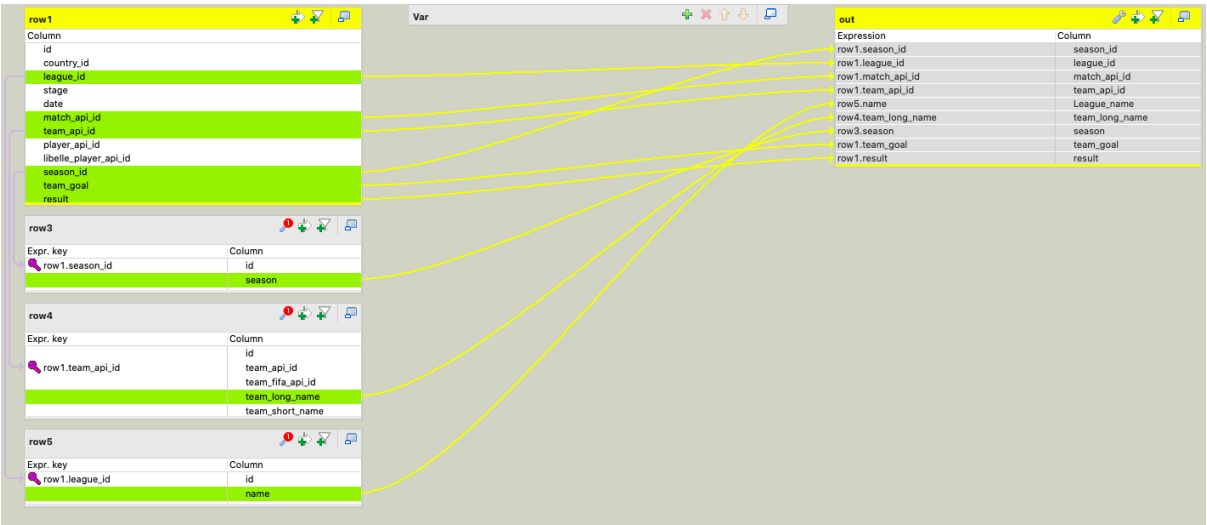



*tAggregateRow - group by saison*

### c) Des buts/nombre de matchs gagnés

Nous avons besoin d'analyser par axes saisons, leagues et teams. Donc, les dimensions saison, league, team et table de fait ont été inclus. Le tMap fait les jointure entre le fait et les dimensions. De plus le output contient que des colonnes qu'on va utiliser pour analyser : league name, season, team name, goal et résultat du match.

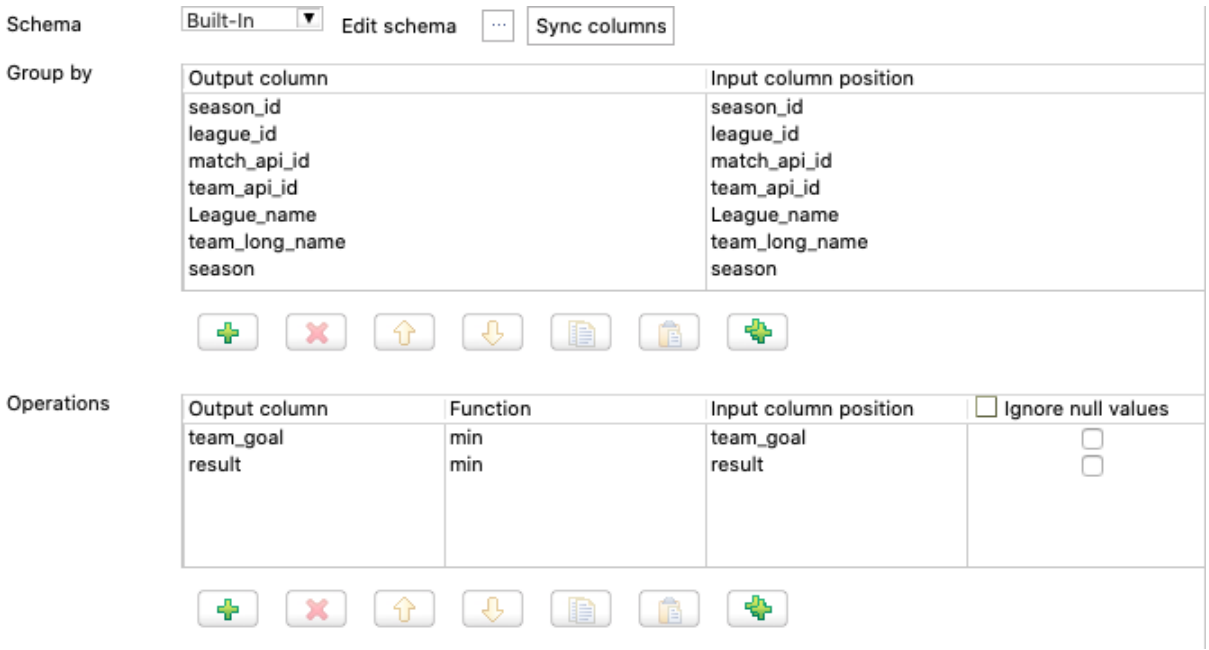



*tMap - jointure et select*

Après, on fait l'agrégation par des axes dans le sujet, saison, league et match. D'ailleurs, dans la table de fait, on a pivoté la table par player, donc pour chaque match on a 11 lignes correspondant pour chaque team, les valeurs du résultat et du goal pour de ces 11 lignes sont les mêmes. Alors, si on veut savoir le résultat et goal pour chaque team on peut choisir l'opération min (ou max, avg…).

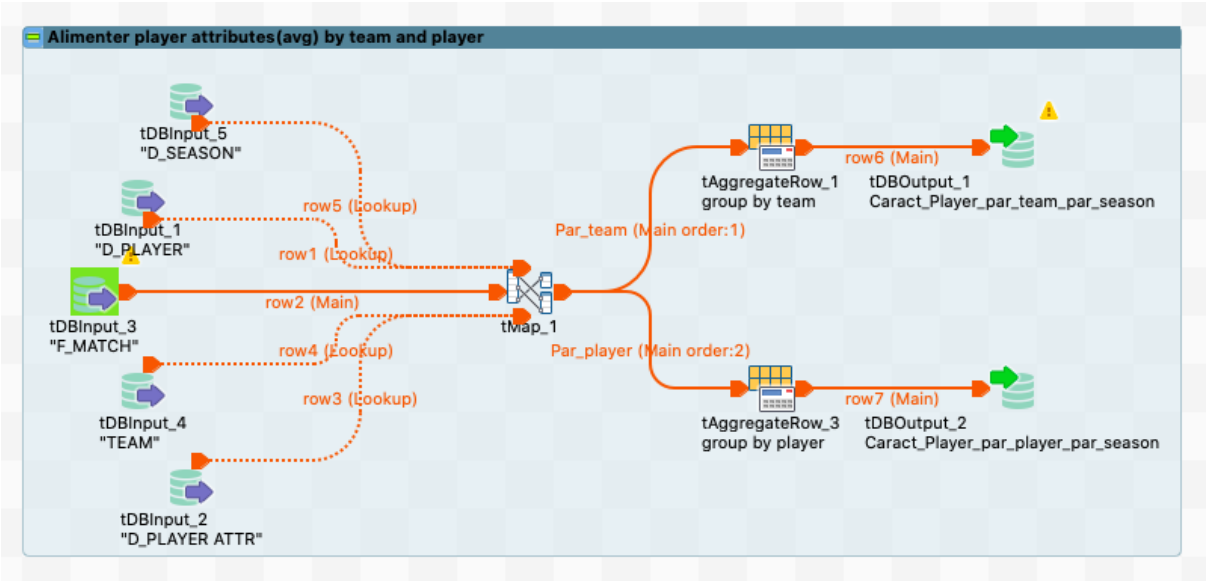
A la fin, les résultats agrégés sont enregistrés dans notre nouvelle table KPI.



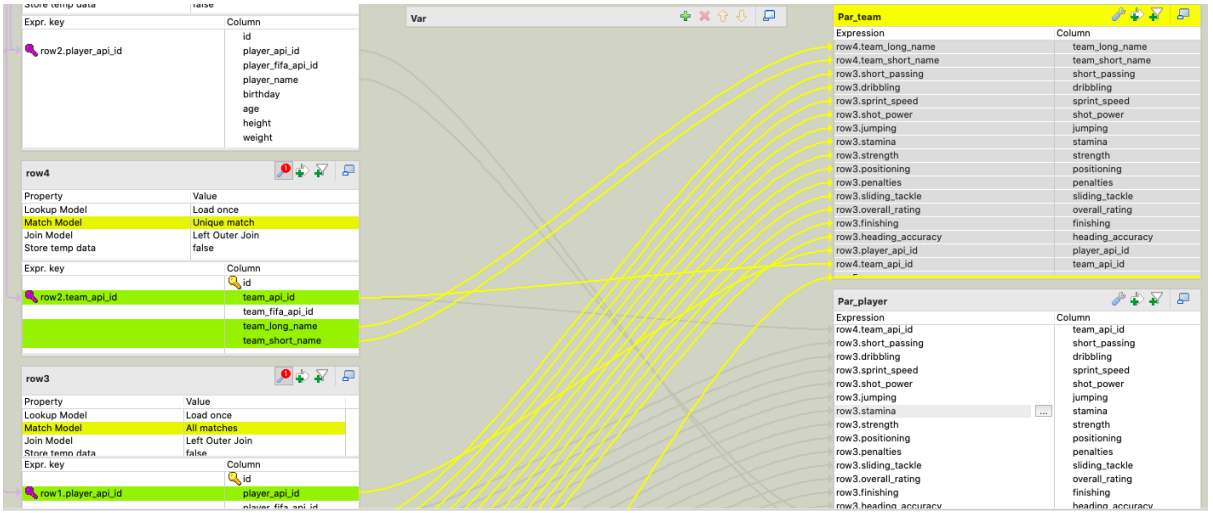

*tAggregateRow - group by saison, league et team*

### d) Radar des compétences des joueurs

Pour comparer les compétences des joueurs par rapport à leur équipe au fil des saisons, les dimensions saison, équipe, player et player attributs sont nécessaire, de plus le table de fait. Ensuite, il faut agréger par joueur et par équipe.




Les jointure entre la dimension et la table de fait dans le tMap sont left join et match unique. D'ailleurs, on a deux flux output, un pour team un autre pour player. Il faut contenir player_api_id dans l'output player et team_api_id dans l'output team.




Après, les tAggregateRow sont utilisés pour agréger par team et player. L'output player agrégé est group by season et player_api_id, l'output team agrégé est group by season et team_api_id. Les opérations des compétences sont moyennes. Ensuite, les deux tables compétences intervalles sont enregistrées.

Group by

| Output column | Input column position |
|---|---|
| team_api_id | team_api_id |
| team_long_name | team_long_name |
| team_short_name | team_short_name |
| season | season |

Operations

| Output column | Function | Input column position | Ignore null values |
|---|---|---|---|
| short_passing | avg | short_passing | ☑ |
| dribbling | avg | dribbling | ☑ |
| sprint_speed | avg | sprint_speed | ☑ |
| shot_power | avg | shot_power | ☑ |
| jumping | avg | jumping | ☑ |
| stamina | avg | stamina | ☑ |
| strength | avg | strength | ☑ |
| positioning | avg | positioning | ☑ |
| penalties | avg | penalties | ☑ |
| sliding_tackle | avg | sliding_tackle | ☑ |
| overall_rating | avg | overall_rating | ☑ |
| finishing | avg | finishing | ☑ |
| heading_accuracy | avg | heading_accuracy | ☑ |

*tAggregateRow - group by saison et team*

Schema: Built-In | Edit schema | Sync columns

Group by

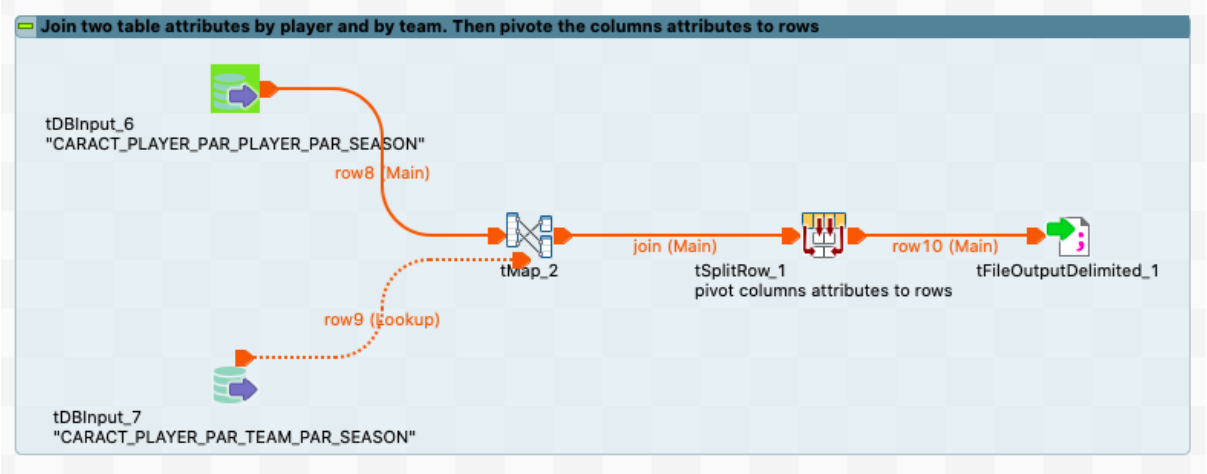
| Output column | Input column position |
|---|---|
| player_api_id | player_api_id |
| player_name | player_name |
| team_api_id | team_api_id |
| season | season |

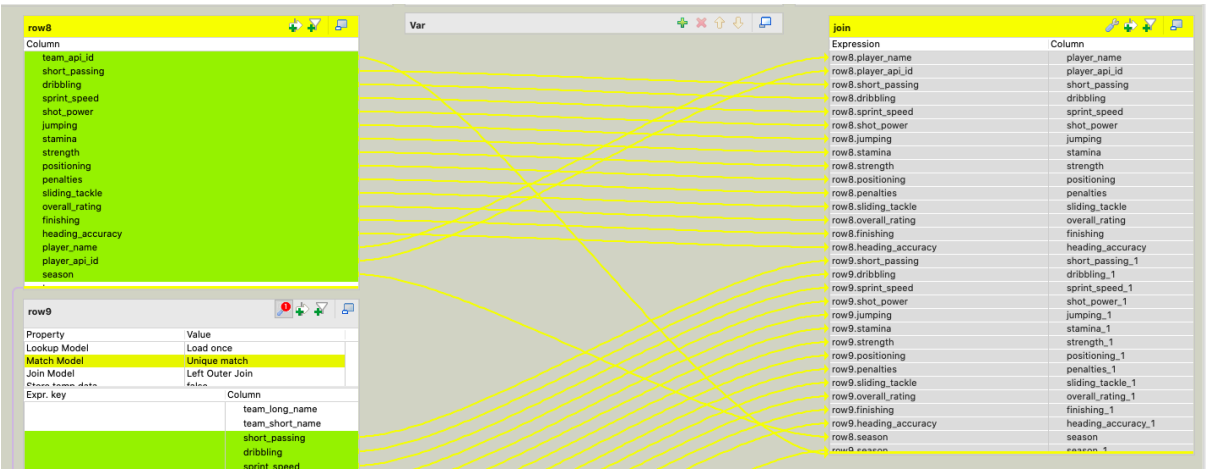
Operations

| Output column | Function | Input column position | Ignore null values |
|---|---|---|---|
| short_passing | avg | short_passing | ☑ |
| dribbling | avg | dribbling | ☑ |
| sprint_speed | avg | sprint_speed | ☑ |
| shot_power | avg | shot_power | ☑ |
| jumping | avg | jumping | ☑ |
| stamina | avg | stamina | ☑ |
| strength | avg | strength | ☑ |
| positioning | avg | positioning | ☑ |
| penalties | avg | penalties | ☑ |
| sliding_tackle | avg | sliding_tackle | ☑ |
| overall_rating | avg | overall_rating | ☑ |
| finishing | avg | finishing | ☑ |
| heading_accuracy | avg | heading_accuracy | ☑ |

*tAggregateRow - group by saison et player*

Après, on utilise tMap pour joindre les deux tables agrégé par team_id et saison, garder toutes les informations de la table agrégé par player et ajouter les colonnes des compétences de leur équipe qui viennent de la table agrégé par team.



La graphe radar a besoin d'une colonne catégorie et leur valeur correspondante. Donc, on a besoin de pivoter les colonne compétence en lignes. De plus, on doit comparer les compétences de joueur avec leur équipe, donc c'est nécessaire de séparer les valeurs des compétences de joueurs et celles de leur équipe en deux colonnes.

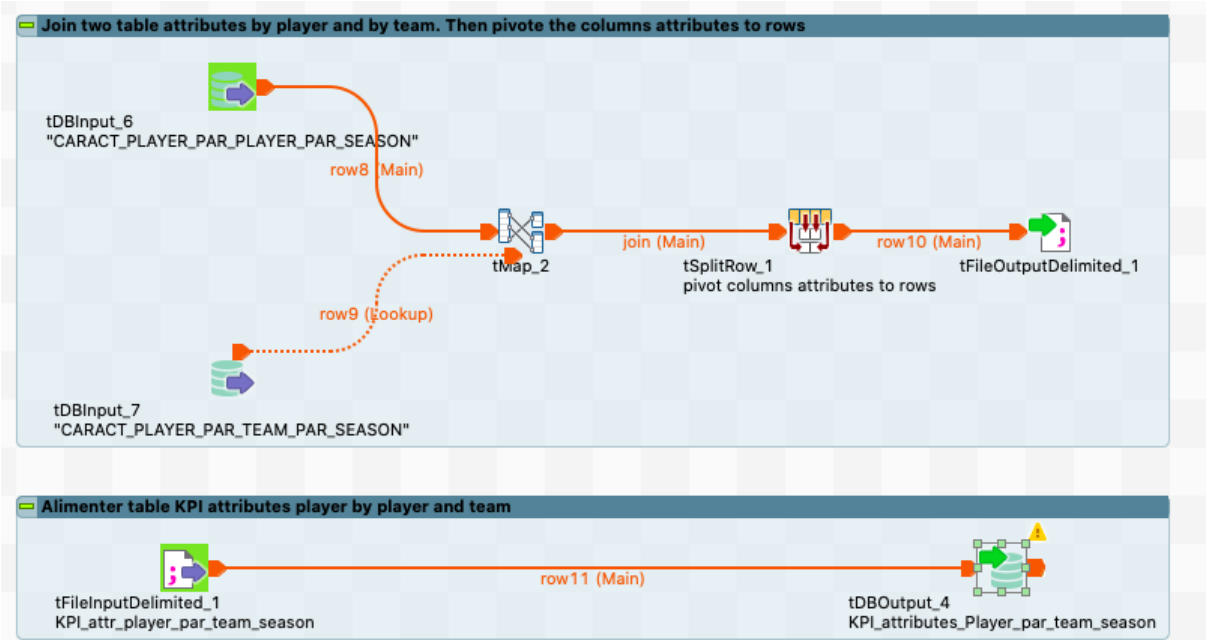
Le componente tSplitRow a été utilisé pour faire ce traitement. On ajoute une colonne attributs pour mettre touts les nom de compétence et les deux colonnes, player compétences et team compétences, qui sont les colonnes correspondantes du output.

Columns mapping

| season | team_api_id | player_name | player_api_id | Attribute | Player_attr | Team_attr |
|---|---|---|---|---|---|---|
| join.season | join.team_api_id | join.player_name | join.player_api_id | "short_passing" | join.short_passing | join.short_passing |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "dribbling" | join.dribbling | join.dribbling_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "sprint_speed" | join.sprint_speed | join.sprint_speed_ |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "shot_power" | join.shot_power | join.shot_power_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "jumping" | join.jumping | join.jumping_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "stamina" | join.stamina | join.stamina_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "strength" | join.strength | join.strength_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "positioning" | join.positioning | join.positioning_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "penalties" | join.penalties | join.penalties_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "sliding_tackle" | join.sliding_tackle | join.sliding_tackle_ |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "overall_rating" | join.overall_rating | join.overall_rating_ |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "finishing" | join.finishing | join.finishing_1 |
| join.season | join.team_api_id | join.player_name | join.player_api_id | "heading_accuracy | join.heading_accur | join.heading_accur |

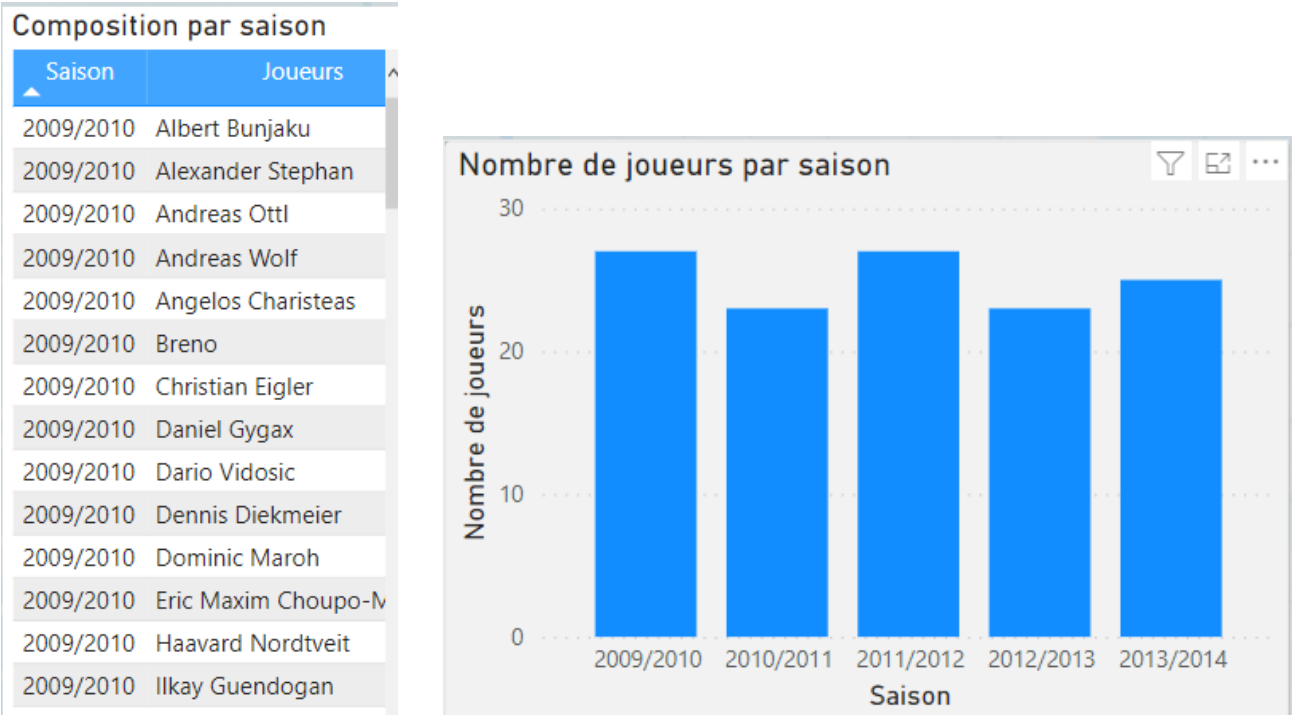
*tSplitRow - pivot les colonnes compétences en lignes*

Finalement, nous ne pouvons pas lire les tables d'un database SQLite et écrire une autre table en même temps. Donc, on crée d'abord un fichier output. Après, on utilise ce fichier pour alimenter nos résultats dans database.



## 6. Les rapports BI expliqués

### a) La composition des équipes au fil des saisons



Composition par saison

| Saison | Joueurs |
|---|---|
| 2009/2010 | Albert Bunjaku |
| 2009/2010 | Alexander Stephan |
| 2009/2010 | Andreas Ottl |
| 2009/2010 | Andreas Wolf |
| 2009/2010 | Angelos Charisteas |
| 2009/2010 | Breno |
| 2009/2010 | Christian Eigler |
| 2009/2010 | Daniel Gygax |
| 2009/2010 | Dario Vidosic |
| 2009/2010 | Dennis Diekmeier |
| 2009/2010 | Dominic Maroh |
| 2009/2010 | Eric Maxim Choupo-M |
| 2009/2010 | Haavard Nordtveit |
| 2009/2010 | Ilkay Guendogan |



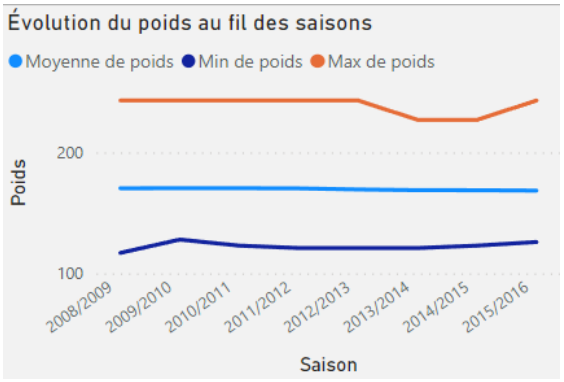
La composition des équipes est représentée à l'aide de 2 éléments :

- Une **table** à deux colonnes pour afficher les noms des joueurs de chaque équipe à chaque saison. La colonne "Saison" est remplie à partir de la colonne "season" de la table `D_season`, et la colonne "Joueurs" à partir de "player_name" de la table `D_Player`.
- Un **histogramme** représentant le nombre de joueurs par saison, avec en axe des X la colonne "season" de la table `D_season` et en Y le nombre de valeurs distinctes de la colonne `player_api_id` de la table `F_Match`.

Toute la page est filtrée avec la colonne `team_long_name` de la table `D_Team`.

### b) L'évolution des caractéristiques des joueurs au fil des saisons

L'évolution des caractéristiques des joueurs au fil des saisons est illustrée à l'aide de

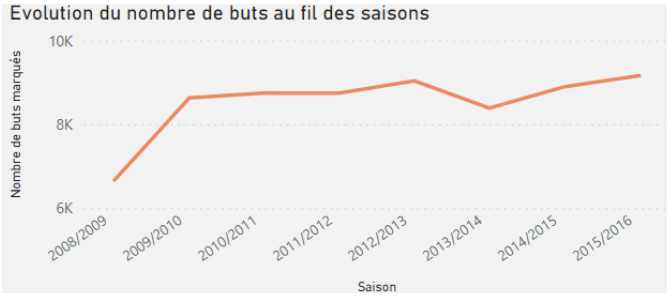


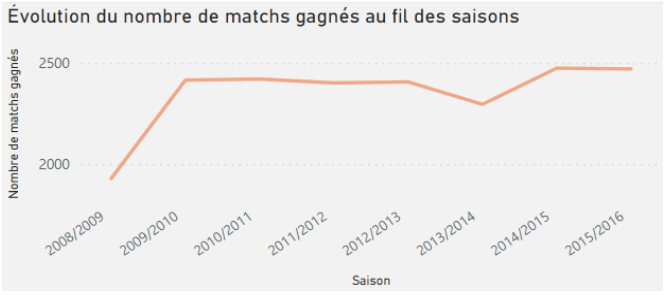

graphiques en courbes, avec en X les saisons et en Y les valeurs des différentes caractéristiques des joueurs (âge, taille, poids). Dans un même graphique sont mis à la fois la moyenne, le maximum et le minimum des valeurs des caractéristiques.

Nous avons rempli les graphes de deux façons différentes :

- La première en utilisant les tables de dimension et de fait du modèle initial (tables `D_Player` et `D_Season`)
- La deuxième en se servant de la table `KPI_caract_joueur_par_saison` construite avec Talend en agrégeant les caractéristiques

### c) Évolution du nombre de buts et de matchs gagnés



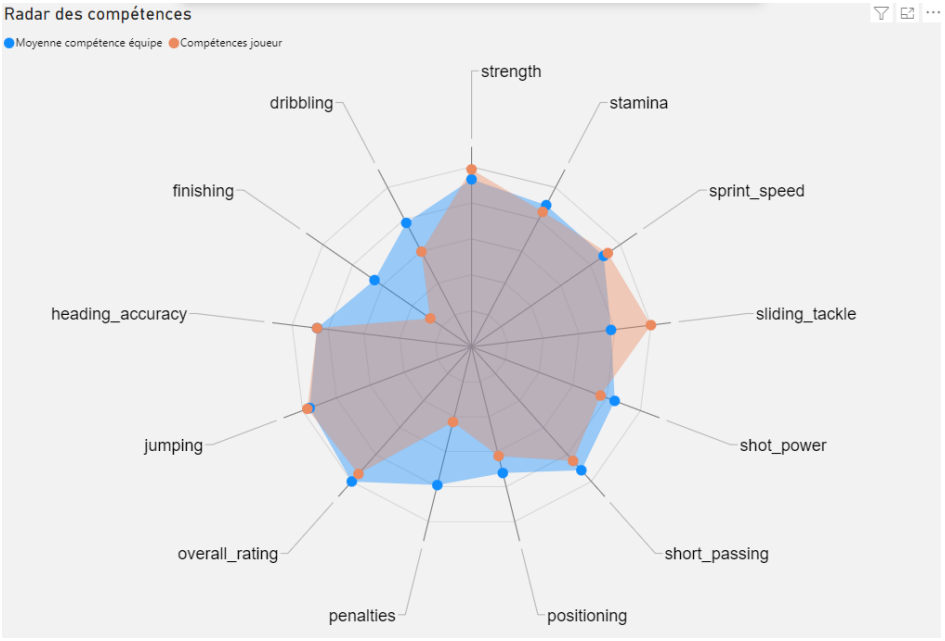




On s'est servi de la table `KPI_Nb_but_gagne_par_team_league_season` que l'on a créée sur Talend par agrégation des résultats des matchs

- ☐ L'évolution du nombre de buts au fil des saisons est représentée par un graphique en courbe avec en X les saisons (`season`) et en Y le nombre de buts marqués (somme de `team_goal`)
- ☐ Même chose pour l'évolution du nombre de matchs gagnés mais avec la somme de la colonne `result` en Y.

Les deux graphiques de la page sont filtrés par ligue (`League_name`), saison (`season`) et équipe (`team_long_name`)

### d) Radar des compétences des joueurs




Il est construit à partir de la table `KPI_attributes_Player_par_team_season` d'agrégation. Nous y avons représenté les compétences des joueurs (`Player_attr`) et la moyenne des compétences de l'équipe (`Team_attr`) simultanément pour permettre la comparaison. Les catégories sont récupérées de la colonne `Attribute`.

Le graphique est filtré par saison (`season`) et par joueur (`player_name`).