



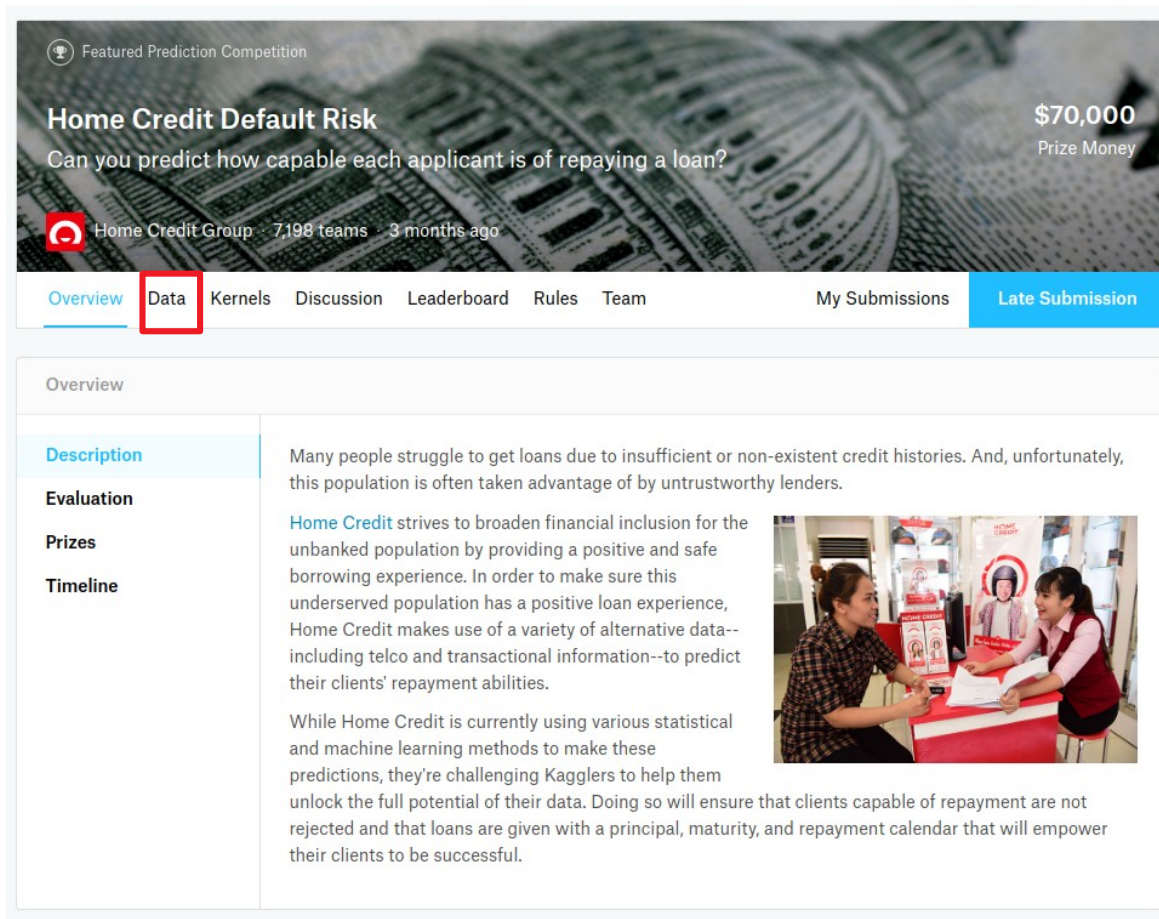
Projet intégration de données

Formation Data Scientist

ECAM 2018-2019

Source des données

<https://www.kaggle.com/c/home-credit-default-risk>



Featured Prediction Competition

Home Credit Default Risk

Can you predict how capable each applicant is of repaying a loan?


\$70,000
Prize Money

Home Credit Group · 7,198 teams · 3 months ago

Overview **Data** Kernels Discussion Leaderboard Rules Team My Submissions Late Submission

Overview

Description	
Evaluation	Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.
Prizes	Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.
Timeline	While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.



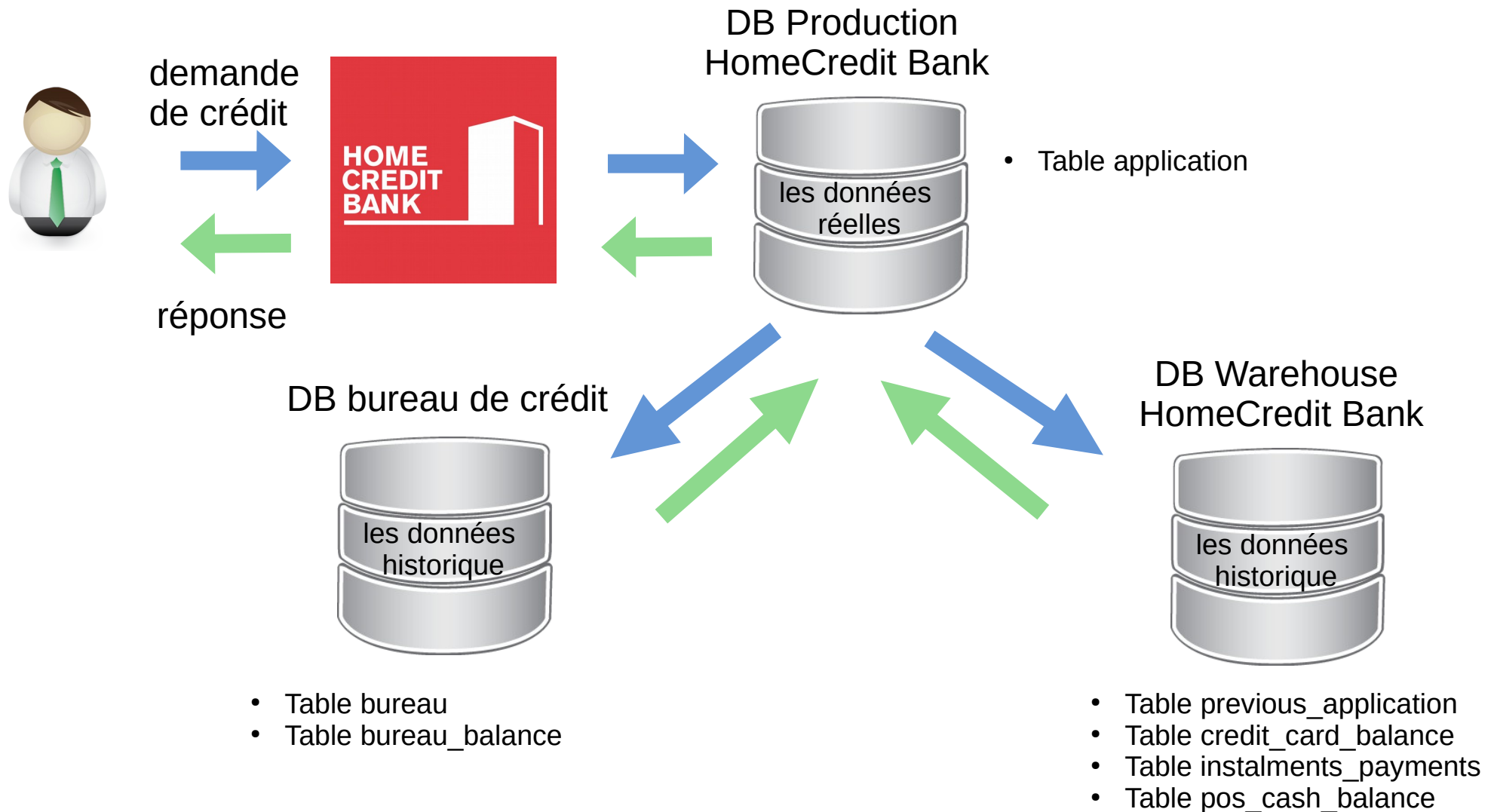
kaggle™

Data (688 MB)

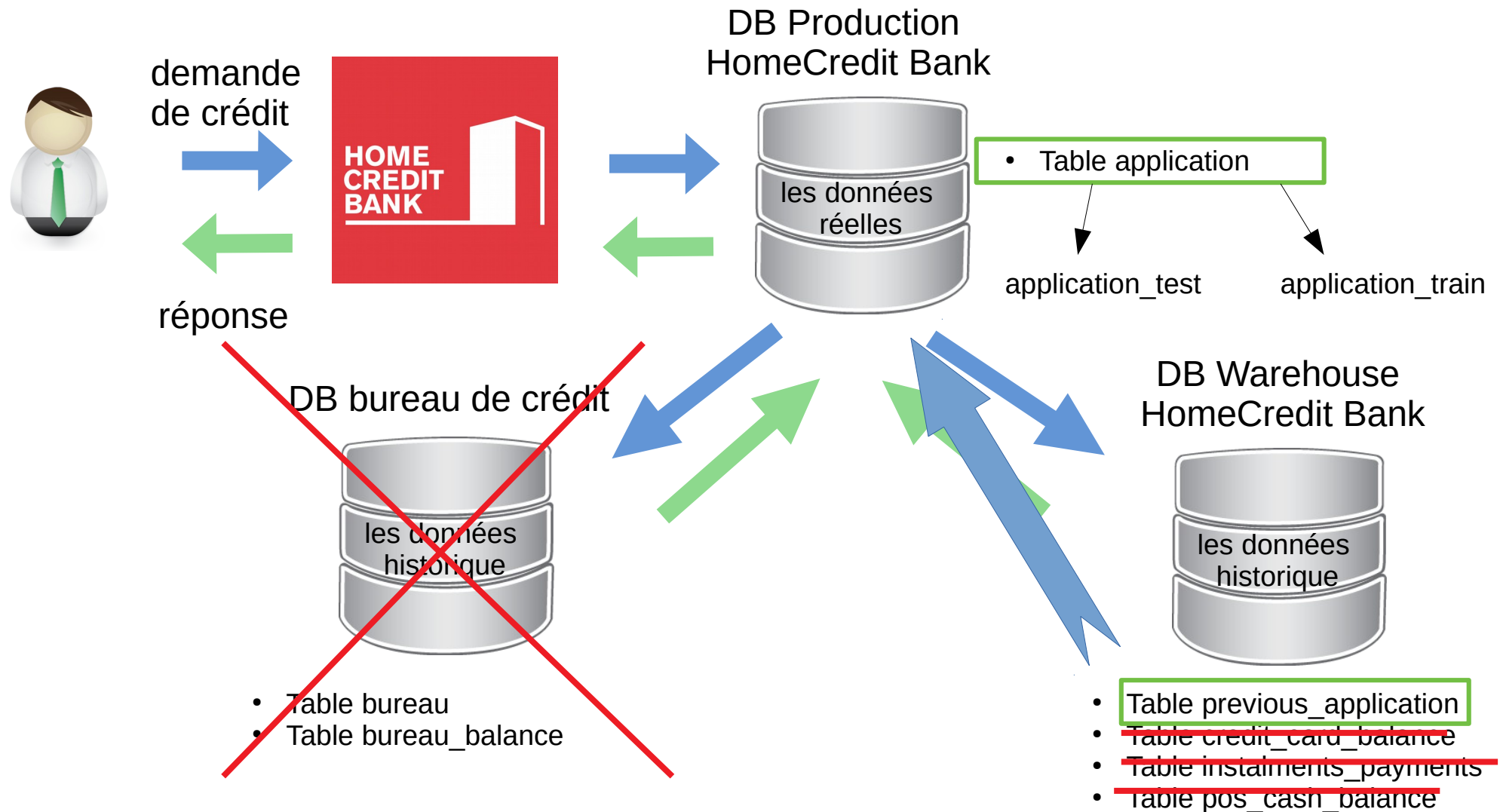
Data Sources

application_test.csv	48.7k x 121
application_train.csv	308k x 122
bureau.csv	1.72m x 17
bureau_balance.csv	27.3m x 3
credit_card_balanc...	3.84m x 23
HomeCredit_columns_...	219 x 5
installments_payme...	13.6m x 8
POS_CASH_balance...	10.0m x 8
previous_applicatio...	1.67m x 37
sample_submission....	48.7k x 2

L'essence de la compétition



L'essence du projet





Motivation

- Project intégration de données
- Leaderboard
- PostgreSQL
- Créer une utilité ETL

Liste des Tâches

Tâches	Charges	Ressources	Dépendances	Coût
Télécharger les données	1 jour	1 développeur	site http://kaggle.com/	500 euro
Importer tous les données dans la base de données	3 jour	1 développeur	RDBMS Environnement, Télécharger les données	1500 euro
Extrait les données pour le projet	1 jour	1 développeur	RDBMS Environnement, Importer tous les données dans la base de données	500 euro
Supprimer des données supplémentaires	0.5 jour	1 développeur	RDBMS Environnement, Extrait les données pour le projet	250 euro
Ajouter des contraintes et des indices	0.5 jour	1 développeur	RDBMS Environnement, Supprimer des données supplémentaires	250 euro
Création de requêtes SQL pour l'analyse	1 jour	1 développeur	RDBMS Environnement, Ajouter des contraintes et des indices	500 euro
Présentation des résultats	1 jour	1 développeur	Création de requêtes SQL pour l'analyse	500 euro
Prix Total				4 000 euro



Tableau de bord

Charge total du projet: 8 j/h

Durée totale du projet: 8 jours

Coût de ressources:

1 j/h développement = 250 Euro

1 jour RDBMS environment = 10 Euro

Coût total du projet: 2080 Euro

Schéma d'Architecture Logique


Dossier avec des fichiers *.csv 

Table name	lines	columns
application_test.csv	48 741	121
application_train.csv	307 511	122
previous_application.csv	1 670 214	37



ETL

- lire le dossier avec les fichiers et insérer le nom des fichiers dans la table temporaire
- lire la première ligne des fichiers, divisée par des virgules et créer des tables avec colonnes du type texte
- lire les données des fichiers csv et remplir les tableaux



DB Prod



Extrait les données pour le projet

- Scripts SQL pour la création de tables finales basées sur les données brutes .
- Transformation de données
- Nettoyage les tables supplémentaires



Transformation finale de la base de données BI

- Convertir les colonnes en un type de données approprié
- Ajouter des index et des contraintes



DB BI



Table name	lines	columns
calendar	4255	6
credit_types	4	2
demande_de_credit	1 670 214	10
achat_types	28	2
client	356 255	4



Requêtes SQL



Résultats





ETL “fait maison”

Problème

Nous avons trois fichiers avec beaucoup des colonnes, comment importer tout dans notre base de données?

Solutions

- faire à la main
- utiliser un ETL
- créer notre propre ETL

ETL étapes

1. Télécharger les données et stocker dans un dossier
2. Lire le dossier avec les fichiers et insérer le nom des fichiers dans la table temporaire:

```
EXECUTE format(E'COPY yk_data_struct FROM PROGRAM \'ls -l %I |  
                sed \'"s/.csv//\'" \' DELIMITER \',\' ;',  
                in_folder);
```

ETL étapes

3. Lire la première ligne des fichiers, divisée par des virgules et créer des tables avec colonnes du type texte, lire les données des fichiers csv et remplir les tableaux

```
req := 'select distinct table_name from yk_data_struct';
OPEN curs_tables FOR EXECUTE(req);
LOOP FETCH curs_tables INTO cur_table;
    EXIT WHEN NOT FOUND;
    -- Save column names to tmp table
    CREATE TEMP TABLE IF NOT EXISTS yk_tmp_cols(cols text) ON COMMIT DROP;
    full_path := format(E'%s%s.csv', in_folder, cur_table);
    EXECUTE format(E'COPY yk_tmp_cols FROM PROGRAM \'head -n1 %I\';', full_path);
    -- Tables creation
    SELECT      format('CREATE TABLE %I(', LOWER(cur_table)) ||
                string_agg(quote_ident(REPLACE (LOWER(col), ' ', ''))
                || ' text', ',') || ')' INTO req
    FROM (SELECT cols FROM yk_tmp_cols LIMIT 1) t ,
        UNNEST(string_to_array(t.cols, ',')) col;
    EXECUTE req;
    -- Import data
    EXECUTE format(E'copy %s from %L DELIMITER \',\' CSV HEADER;',
                lower(cur_table), in_csv_path);
    --Cleanup
    DELETE FROM yk_tmp_cols;
END LOOP;
```

DB Prod is Ready


Dossier avec des fichiers *.csv 

Table name	lines	columns
application_test.csv	48 741	121
application_train.csv	307 511	122
previous_application.csv	1 670 214	37



ETL

- lire le dossier avec les fichiers et insérer le nom des fichiers dans la table temporaire
- lire la première ligne des fichiers, divisée par des virgules et créer des tables avec colonnes du type texte
- lire les données des fichiers csv et remplir les tableaux

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

Transformation finale de la base de données BI

- Convertir les colonnes en un type de données approprié
- Ajouter des index et des contraintes



PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

Extrait les données pour le projet

- Scripts SQL pour la création de tables finales basées sur les données brutes .
- Transformation de données
- Nettoyage les tables supplémentaires



PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

DB Prod



PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL

PostgreSQL



DB BI



Table name	lines	columns
calendar	4255	6
credit_types	4	2
demande_de_credit	1 670 214	10
achat_types	28	2
client	356 255	4



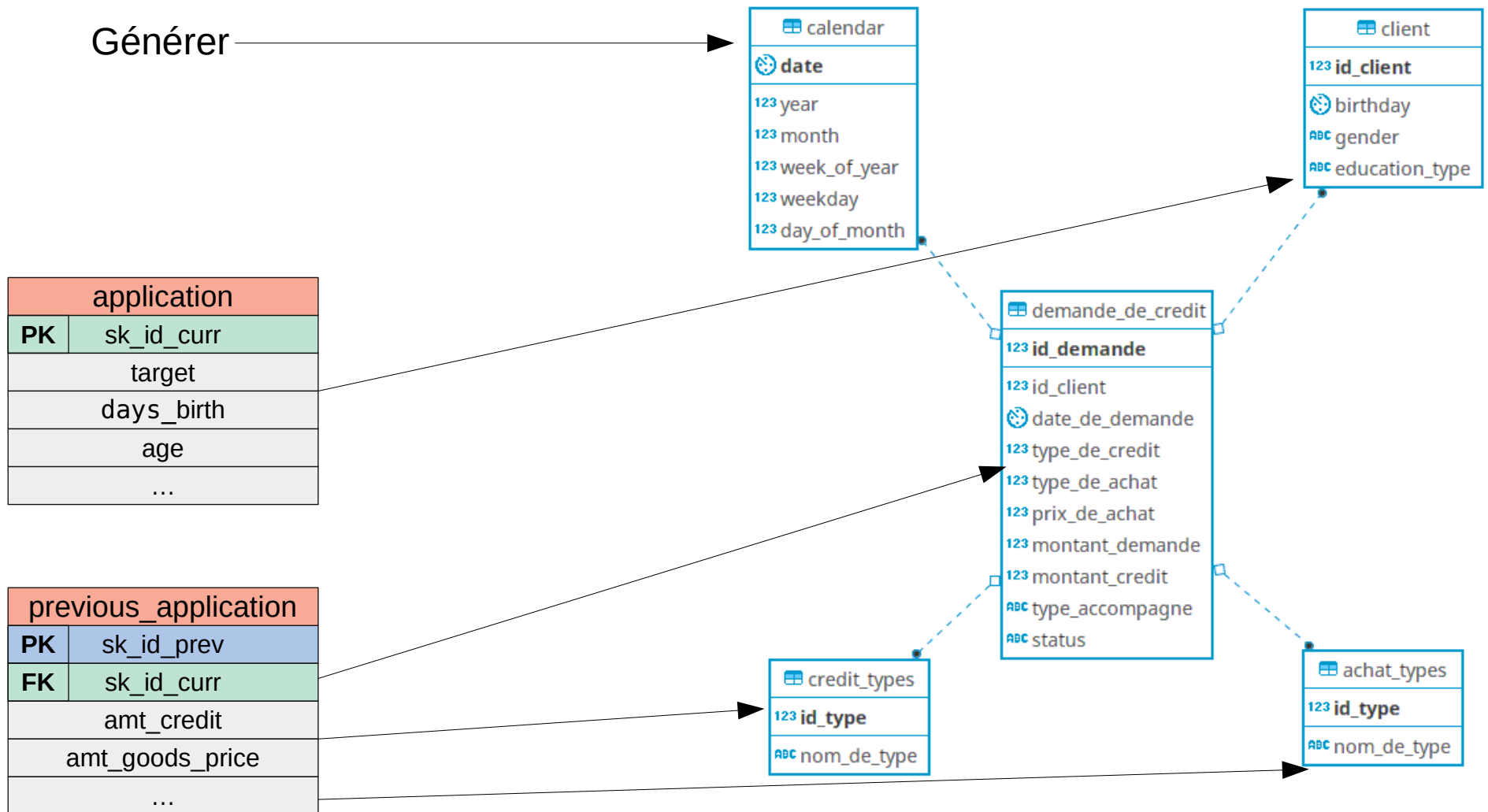
Requêtes SQL



Résultats



Modèle logique des données



Extrait les données pour le projet

```
CREATE TABLE client AS (  
    SELECT  
        sk_id_curr AS id_client,  
        (ZERO_POINT + (INTERVAL '1 day' * days_birth::SMALLINT))::DATE AS birthday,  
        code_gender AS gender,  
        name_education_type AS education_type  
    FROM  
        application  
);  
  
CREATE TABLE credit_types AS (  
    SELECT  
        DISTINCT DENSE_RANK() OVER(ORDER BY NAME_CONTRACT_TYPE) AS id_type ,  
        NAME_CONTRACT_TYPE AS nom_de_type  
    FROM  
        previous_application  
);  
  
CREATE TABLE achat_types AS (  
    SELECT  
        DISTINCT DENSE_RANK() OVER(ORDER BY name_goods_category) AS id_type ,  
        name_goods_category AS nom_de_type  
    FROM  
        previous_application  
);
```

Extrait les données pour le projet

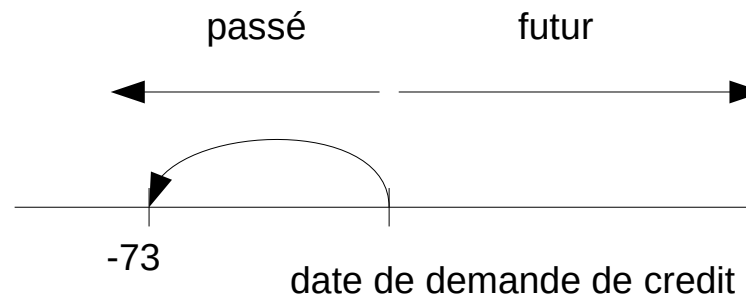
```
CREATE TABLE calendar AS (
  SELECT
    d.date AS DATE,
    EXTRACT(YEAR FROM d.date)::SMALLINT AS YEAR,
    EXTRACT(MONTH FROM d.date)::SMALLINT AS MONTH,
    EXTRACT(WEEK FROM d.date)::SMALLINT AS week_of_year,
    EXTRACT(isodow FROM d.date)::SMALLINT AS weekday,
    EXTRACT(DAY FROM d.date)::SMALLINT AS day_of_month
  FROM
    (
      SELECT
        date_trunc('day',dd):: DATE AS DATE
      FROM
        generate_series (
          ZERO_POINT - CALENDAR_DELTA ,
          ZERO_POINT + CALENDAR_DELTA ,
          '1 day'::INTERVAL
        ) dd
    )d
);

CREATE TABLE demande_de_credit AS (
  SELECT
    p.sk_id_prev AS id_demande,
    p.sk_id_curr AS id_client,
    (ZERO_POINT + (INTERVAL '1 day' * days_decision)::SMALLINT)::DATE AS date_de_demande,
    c.id_type AS type_de_credit,
    a.id_type AS type_de_achat,
    p.amt_goods_price AS prix_de_achat,
    p.amt_application AS montant_demande,
    p.amt_credit AS montant_credit,
    p.name_type_suite AS type_accompagne,
    p.name_contract_status AS status
  FROM
    previous_application p
  LEFT JOIN credit_types c ON
    c.nom_de_type = p.name_contract_type
  LEFT JOIN achat_types a ON
    a.nom_de_type = p.name_goods_category
);
```

Transformation de données

```
select days_decision from previous_application;
```

	123 days_decision
1	-73
2	-164
3	-301
4	-512
5	-781
6	-684



```
ZERO_POINT CONSTANT DATE := '2018-05-18'::DATE - (INTERVAL '100 days');  
CALENDAR_DELTA CONSTANT INTERVAL := INTERVAL '10 years';
```


Conversion des colonnes en un type de données approprié

Tout le colonnes sont **TEXT**

Nous essayons faire conversion:

```
EXECUTE format('ALTER TABLE %s ALTER COLUMN %s TYPE NUMERIC USING %s::NUMERIC;',  
               table_name,  
               column_name,  
               column_name);
```

Et attrape exception:

```
EXCEPTION  
WHEN invalid_text_representation then -- can not convert text to number  
    NULL;  
WHEN cannot_coerce THEN -- can not convert date to number  
    NULL;
```

Ajouter des indexes et des contraintes

--primary keys

```
ALTER TABLE calendar ADD PRIMARY KEY (DATE);  
ALTER TABLE client ADD PRIMARY KEY (id_client);  
ALTER TABLE credit_types ADD PRIMARY KEY (id_type);  
ALTER TABLE achat_types ADD PRIMARY KEY (id_type);  
ALTER TABLE demande_de_credit ADD PRIMARY KEY (id_demande);
```

--foreign keys

```
ALTER TABLE demande_de_credit ADD CONSTRAINT fk_date_de_demande FOREIGN KEY (date_de_demande) REFERENCES calendar (DATE);  
ALTER TABLE demande_de_credit ADD CONSTRAINT fk_type_de_credit FOREIGN KEY (type_de_credit) REFERENCES credit_types (id_type);  
ALTER TABLE demande_de_credit ADD CONSTRAINT fk_type_de_achat FOREIGN KEY (type_de_achat) REFERENCES achat_types (id_type);  
ALTER TABLE demande_de_credit ADD CONSTRAINT fk_id_client FOREIGN KEY (id_client) REFERENCES client (id_client);
```

--indexes

```
CREATE UNIQUE INDEX idx_id_demande ON demande_de_credit (id_demande);  
CREATE UNIQUE INDEX idx_client_id_client ON client (id_client);  
CREATE INDEX idx_id_client ON demande_de_credit (id_client);
```

DB BI is Ready


Dossier avec des fichiers *.csv 

Table name	lines	columns
application_test.csv	48 741	121
application_train.csv	307 511	122
previous_application.csv	1 670 214	37



ETL

- lire le dossier avec les fichiers et insérer le nom des fichiers dans la table temporaire
- lire la première ligne des fichiers, divisée par des virgules et créer des tables avec colonnes du type texte
- lire les données des fichiers csv et remplir les tableaux



DB Prod



Extrait les données pour le projet

- Scripts SQL pour la création de tables finales basées sur les données brutes .
- Transformation de données
- Nettoyage les tables supplémentaires



Transformation finale de la base de données BI

- Convertir les colonnes en un type de données approprié
- Ajouter des index et des contraintes



DB BI



Table name	lines	columns
calendar	4255	6
credit_types	4	2
demande_de_credit	1 670 214	10
achat_types	28	2
client	356 255	4



Requêtes SQL



Résultats



Requêtes SQL

1.

SELECT

```
c.gender ,  
COUNT(d.id_demande) "count" ,  
to_char(PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY prix_de_achat),  
        '999 999 999.99') med_of_prix ,  
to_char(MIN(d.montant_demande), '999 999 999.99') "min demande" ,  
to_char(MAX(d.montant_demande), '999 999 999.99') "max demande" ,  
to_char(AVG(d.montant_demande-d.montant_credit),  
        '999 999 999.99') "avg (demande - credit)" ,  
to_char(SUM(CASE WHEN d.status = 'Approved' THEN 1 ELSE 0 END)::FLOAT  
        / COUNT(d.id_demande)* 100, '99.99 %') "% of approved" ,  
to_char(SUM(CASE WHEN d.status = 'Refused' THEN 1 ELSE 0 END)::FLOAT  
        / COUNT(d.id_demande)* 100, '99.99 %') "% of refused"
```

FROM

```
demande_de_credit d  
LEFT JOIN client c ON d.id_client = c.id_client
```

GROUP BY c.gender;

Résultats:

gender	count	med_of_prix	min demande	max demande	avg (demande - credit)	% of approved	% of refused
F	1131886	113 211.00	.00	6 905 160	-21 389.81	62.00 %	17.18 %
M	538273	101 434.50	.00	4 455 000	-19 808.61	62.23 %	17.87 %
XNA	55	180 000.00	.00	1 269 000	-16 301.47	41.82 %	45.45 %

Requêtes SQL

2.

```
WITH age_range AS (
  SELECT      a.id_demande,
              CASE
                WHEN a.age_when_demande > 0   AND a.age_when_demande < 25 THEN '0-25'
                WHEN a.age_when_demande >= 25 AND a.age_when_demande < 40 THEN '25-40'
                WHEN a.age_when_demande >= 40 AND a.age_when_demande < 65 THEN '40-65'
                WHEN a.age_when_demande >= 65 THEN '65+'
              END AS RANGE
  FROM
    (
      SELECT
        d.id_demande,
        EXTRACT(YEAR FROM age(d.date_de_demande,
                                c.birthday))::SMALLINT AS age_when_demande
      FROM
        demande_de_credit d
      LEFT JOIN client c ON d.id_client = c.id_client )a
    )
  SELECT      a.range AS "age when demande range",
    to_char(SUM(CASE WHEN d.type_accompagne = 'Family' THEN 1 ELSE 0 END)::FLOAT
              / COUNT(a.id_demande)* 100, '99.99 %') "% of family",
    to_char(SUM(CASE WHEN d.type_accompagne = 'Group of people' THEN 1 ELSE 0 END)::FLOAT
              / COUNT(a.id_demande)* 100, '99.99 %') "% of group",
    to_char(SUM(CASE WHEN d.type_accompagne = 'Unaccompanied' THEN 1 ELSE 0 END)::FLOAT
              / COUNT(a.id_demande)* 100, '99.99 %') "% of unaccompanied",
    to_char(SUM(CASE WHEN d.type_accompagne = 'Children' THEN 1 ELSE 0 END)::FLOAT
              / COUNT(a.id_demande)* 100, '99.99 %') "%of children",
    to_char(SUM(CASE WHEN d.type_accompagne = 'Spouse, partner' THEN 1 ELSE 0 END)::FLOAT
              / COUNT(a.id_demande)* 100, '99.99 %') "% of spouse"
  FROM age_range a
  INNER JOIN demande_de_credit d ON d.id_demande = a.id_demande
  GROUP BY a.range
  ORDER BY a.range ;
```

Requêtes SQL

Résultats:

age range when demande	% of family	% of group	% of unaccompanied	%of children	% of spouse
0-25	15.14 %	0.33 %	30.44 %	0.24 %	6.30 %
25-40	13.43 %	0.15 %	28.87 %	1.58 %	5.15 %
40-65	12.12 %	0.10 %	31.74 %	2.36 %	2.95 %
65+	6.59 %	0.05 %	27.51 %	0.61 %	0.90 %

Requêtes SQL

3.

```
WITH age_range AS (
  SELECT a.id_demande,
    CASE
      WHEN a.age_when_demande > 0 AND a.age_when_demande < 25 THEN '0-25'
      WHEN a.age_when_demande >= 25 AND a.age_when_demande < 40 THEN '25-40'
      WHEN a.age_when_demande >= 40 AND a.age_when_demande < 65 THEN '40-65'
      WHEN a.age_when_demande >= 65 THEN '65+'
    END AS RANGE
  FROM (
    SELECT
      d.id_demande,
      EXTRACT(YEAR FROM age(d.date_de_demande, c.birthday))::SMALLINT AS
                                                                    age_when_demande
    FROM
      demande_de_credit d
    LEFT JOIN client c ON d.id_client = c.id_client )a
  )
SELECT
  a_t.nom_de_type as "Type de achat",
  SUM(CASE WHEN a_r.range = '0-25' THEN 1 ELSE 0 END) "0-25",
  SUM(CASE WHEN a_r.range = '25-40' THEN 1 ELSE 0 END) "25-40",
  SUM(CASE WHEN a_r.range = '40-65' THEN 1 ELSE 0 END) "40-65",
  SUM(CASE WHEN a_r.range = '65+' THEN 1 ELSE 0 END) "65+",
  COUNT(d.id_demande) "Total"
FROM
  age_range a_r
  INNER JOIN demande_de_credit d ON d.id_demande = a_r.id_demande
  INNER JOIN achat_types a_t ON a_t.id_type = d.type_de_achat
GROUP BY a_t.nom_de_type
ORDER BY "Total" DESC;
```

Requêtes SQL

Résultats:

Type de achat	0-25	25-40	40-65	65+	Total
XNA	31014	333139	570422	16234	950809
Mobile	33757	110730	79746	475	224708
Consumer Electronics	9102	47547	64007	920	121576
Computers	15091	50036	40179	463	105769
Audio/Video	11128	44771	43017	525	99441
Furniture	3083	20340	29718	515	53656
Photo / Cinema Equipment	3933	12898	8146	44	25021
Construction Materials	847	8179	15611	358	24995
Clothing and Accessories	1702	9739	11872	241	23554

Requêtes SQL

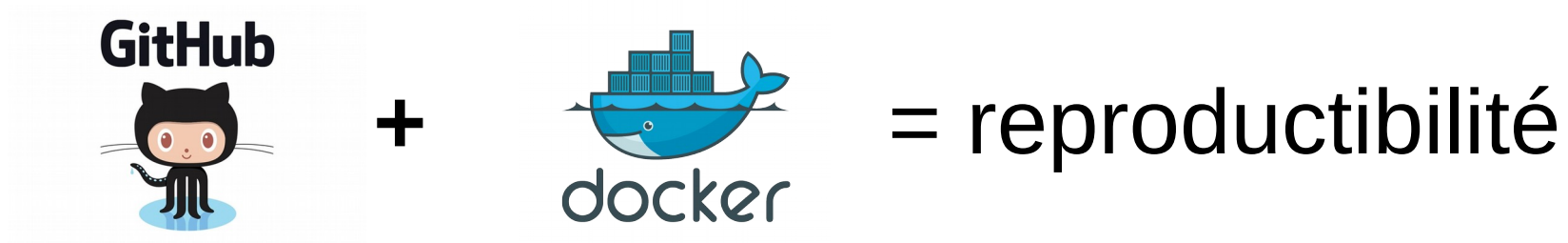
4.

```
SELECT
    c."year",
    COUNT(d.id_demande)
FROM calendar c
    LEFT JOIN demande_de_credit d ON c."date" = d.date_de_demande
GROUP BY c."year"
ORDER BY c."year";
```

Résultats:

year	count
2008	0
2009	0
2010	73 110
2011	99 028
2012	81 955
2013	98 374
2014	126 900
2015	203 421
2016	357 152
2017	585 369
2018	44 905
2019	0
2020	0

Conclusion



https://github.com/konyshev/ecam_ds/tree/master/IntProj



Merci

Q&R