BigQuery et BigQuery ML avec TensorFlow

m2iformation.fr

Programme de la Formation

Module 1 : Introduction à BigQuery
Module 2 : Architecture et Fonctionnement
Module 3 : SQL Avancé dans BigQuery
Module 4 : Optimisation et Performance
Module 5 : Sécurité et Gouvernance
Module 6 : BigQuery ML - Fondamentaux
Module 7 : Modèles ML Avancés
Module 8 : Intégration TensorFlow
Module 9 : MLOps et Production
Module 10 : Cas d'Usage et Best Practices

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 1

Introduction à BigQuery

Objectifs du Module 1

- Comprendre ce qu'est BigQuery
- Découvrir l'écosystème Google Cloud
- Apprendre les concepts de base
- Comprendre le modèle de tarification
- Configurer son environnement
- Exécuter ses premières requêtes

Qu'est-ce que BigQuery ?

Définition

Service d'entrepôt de données cloud entièrement géré, sans serveur, hautement scalable et rentable pour l'analyse de données.

Caractéristiques Principales

- Serverless : Pas d'infrastructure à gérer
- Scalable : Pétaoctets de données
- Rapide : Requêtes en secondes sur TB de données
- Standard SQL : Compatible SQL ANSI
- Intégré : Écosystème Google Cloud et tiers
- ML intégré : BigQuery ML natif

Use Cases

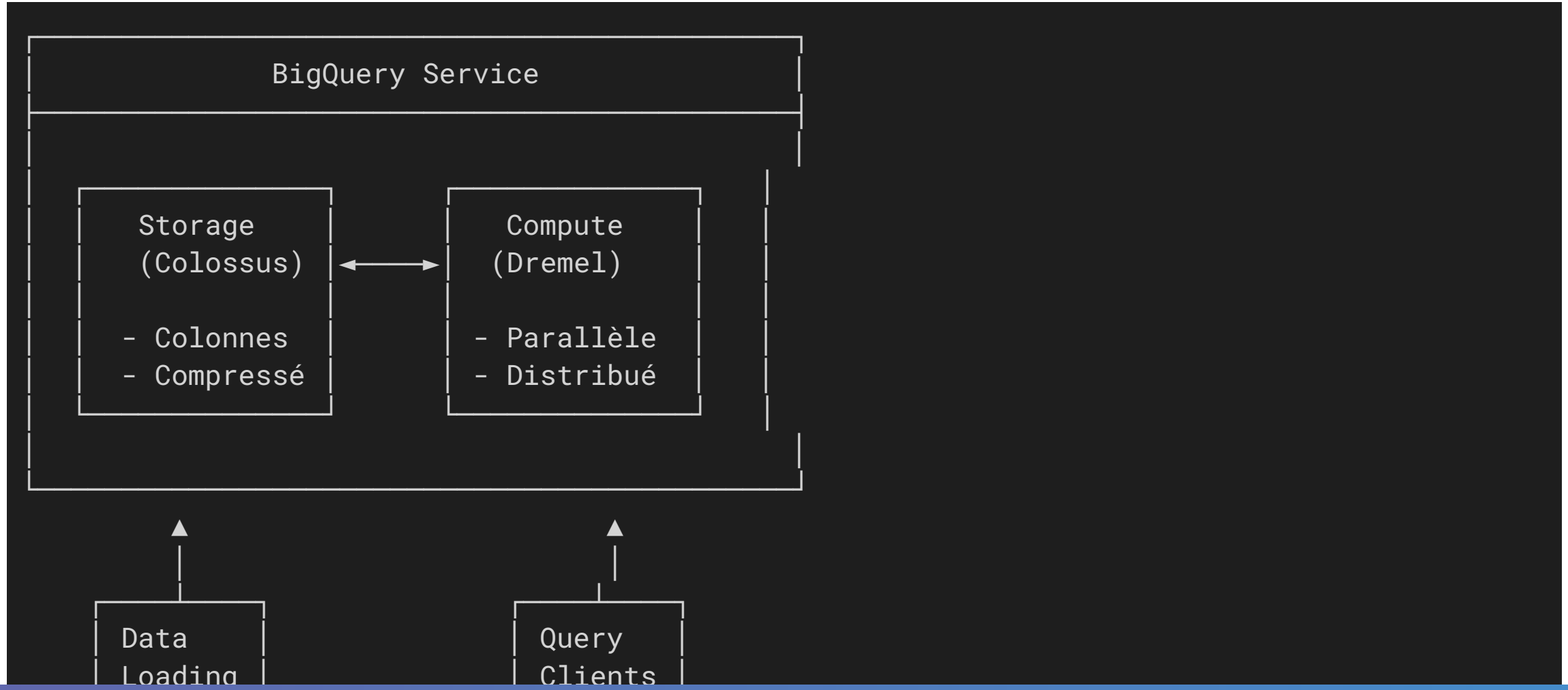
- Data warehousing
- Analytics en temps réel
- Machine Learning
- Business Intelligence

BigQuery vs Solutions Traditionnelles

Aspect	BigQuery	Entrepôts Traditionnels
Infrastructure	Serverless, géré	Provisionnement, maintenance
Scalabilité	Automatique, instantanée	Manuelle, planifiée
Performance	Optimisée colonne, parallèle	Dépend configuration
Coût	Pay-per-query ou flat-rate	Coûts fixes infrastructure
Temps Setup	Minutes	Semaines/Mois
ML Intégré	BigQuery ML natif	Outils externes
Stockage	Découplé du calcul	Couplé
Maintenance	Automatique	Manuelle

Avantages BigQuery : Rapidité de mise en place, pas de gestion infrastructure, scalabilité automatique, ML intégré

Architecture Serverless



Concepts de Base

Projet

Conteneur de niveau supérieur dans Google Cloud

- Facturation au niveau projet
- Quotas et limites

Dataset

Conteneur logique pour tables, vues, routines

- Niveau de sécurité et partage
- Localisation géographique
- Expiration optionnelle

Concepts de Base

Table

Stockage de données structurées

- Schéma défini (colonnes typées)
- Partitionnement optionnel
- Clustering optionnel

Vue

Requête SQL sauvegardée

- Logique réutilisable
- Pas de stockage physique

Types de Tables

Table Native

Stockage dans BigQuery, format propriétaire optimisé

Table Externe

Données stockées ailleurs (Cloud Storage, Drive, Bigtable)

- Pas de coût de stockage BigQuery
- Performance réduite
- Use case : Data Lakes

Table Partitionnée

Divisée en segments basés sur colonne

- Améliore performance
- Réduit coûts
- Types : Temps, Entier, Ingestion

Table Clusterisée

Données triées par colonnes spécifiques

- Combine avec partitionnement
- Améliore filtrage et agrégation

Hiérarchie des Ressources

```
Organization
    │
    ├── Folder (optionnel)
    │    └── Folder
    │
    └── Project
        ├── Dataset 1 (region: EU)
        │    ├── Table A
        │    ├── Table B (partitioned)
        │    ├── View C
        │    └── Routine D
        │
        └── Dataset 2 (region: US)
             ├── Table X
             └── ML Model Y
```

Modèle de Tarification

Stockage

- Actif : 0,020 USD/GB/mois (premiers 90 jours)
- Long terme : 0,010 USD/GB/mois (après 90 jours inactif)
- 10 GB gratuits/mois

Calcul (Requêtes)

On-Demand (par défaut)

- 6,25 USD par TB de données traitées
- 1 TB gratuit/mois
- Pas de coût pour cached results

Flat-Rate (réservation)

- À partir de 100 slots (2 000 USD/mois)
- Coût prévisible
- Pour workloads constants

Optimiser les Coûts

Réduire Données Traitées

- SELECT colonnes spécifiques (pas SELECT *)
- Filtrer avec WHERE
- Utiliser tables partitionnées
- Utiliser tables clusterisées
- Preview gratuit (1 000 lignes)

Utiliser Cache

- Cache 24h gratuit
- Actif par défaut

Optimiser les Coûts

Monitoring

- Tableau de bord de coûts
- Alertes de budget
- Quotas personnalisés

Exemple

```
-- Coûteux (1 TB traité)
SELECT * FROM `dataset.large_table`;

-- Optimisé (100 MB traité)
SELECT user_id, amount FROM `dataset.large_table`
WHERE date = '2024-01-01';
```

Configuration de l'Environnement

Prérequis

- Compte Google Cloud
- Projet GCP créé
- Facturation activée

Accès BigQuery

Console Web

- console.cloud.google.com/bigquery
- Interface graphique complète

bq CLI

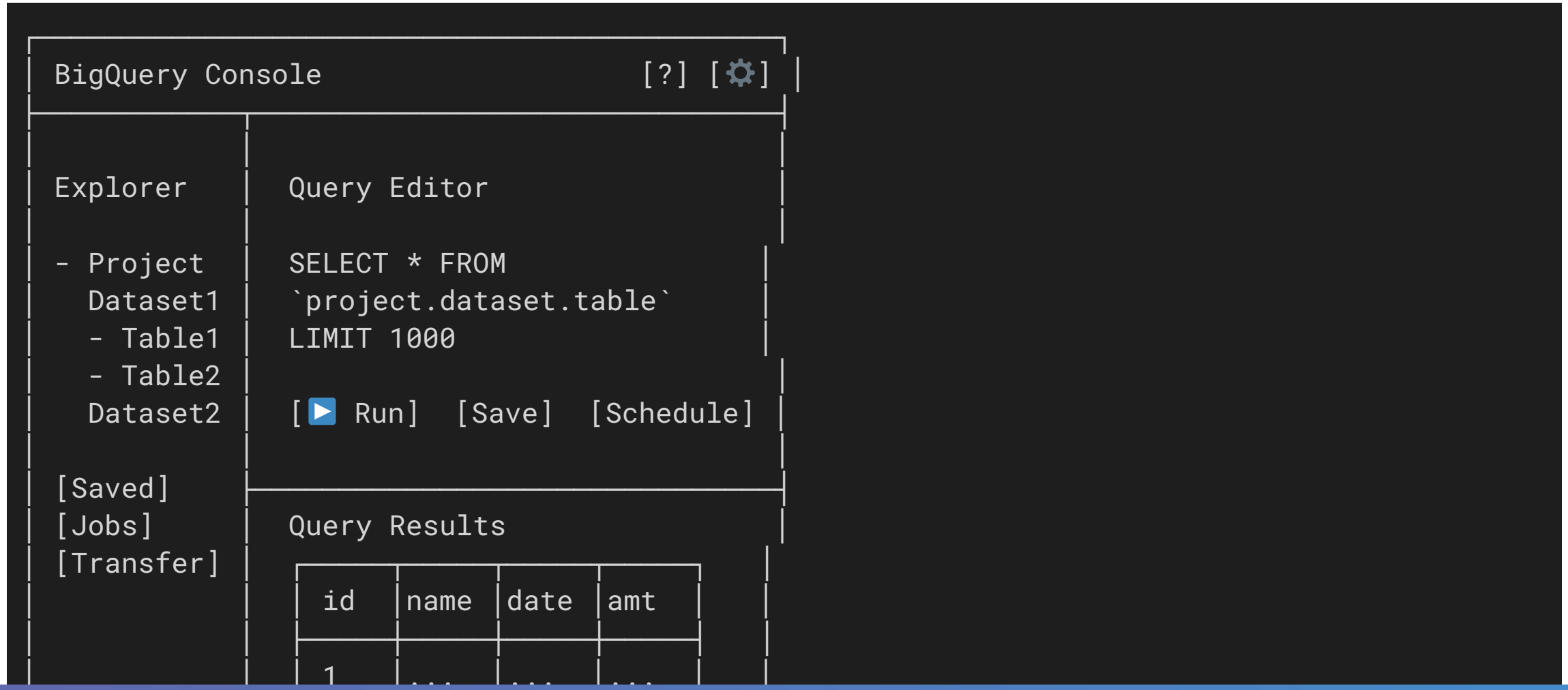
```
# Installation via Cloud SDK
gcloud components install bq

# Authentification
gcloud auth login
gcloud config set project PROJECT_ID
```

Client Libraries

Python, Java, Node.js, Go, etc.

Interface Web BigQuery



Première Requête SQL

Datasets Publics

BigQuery offre nombreux datasets publics gratuits

```sql
-- Requête sur dataset public
SELECT
  name,
  gender,
  SUM(number) as total
FROM `bigquery-public-data.usa_names.usa_1910_2013`
WHERE year = 2013
GROUP BY name, gender
ORDER BY total DESC
LIMIT 10;
```

Résultat

Top 10 prénoms aux USA en 2013

Coût

Gratuit (dans quota 1 TB/mois)

Créer un Dataset

Console

```
1. Cliquer sur les 3 points à côté du projet
2. "Create dataset"
3. Dataset ID : my_dataset
4. Location : EU (ou US, multi-region)
5. Expiration : None
6. Create dataset
```

Créer un Dataset

bq CLI

```
bq mk --location=EU --dataset PROJECT_ID:my_dataset

# Avec expiration
bq mk --location=EU \
  --default_table_expiration=3600 \
  PROJECT_ID:my_dataset
```

Créer un Dataset

Python

```
from google.cloud import bigquery

client = bigquery.Client()
dataset = bigquery.Dataset("PROJECT_ID.my_dataset")
dataset.location = "EU"
dataset = client.create_dataset(dataset)
```

Créer une Table

Depuis Fichier CSV

```
bq load --source_format=CSV \
  --skip_leading_rows=1 \
  --autodetect \
  my_dataset.my_table \
  gs://my-bucket/data.csv
```

Créer une Table

Depuis Query

```
CREATE OR REPLACE TABLE `my_dataset.my_table` AS
SELECT
  user_id,
  name,
  created_at
FROM `other_dataset.source_table`
WHERE created_at >= '2024-01-01';
```

Créer une Table

Schéma Explicite

```
CREATE TABLE `my_dataset.my_table` (
  user_id INT64,
  name STRING,
  email STRING,
  created_at TIMESTAMP
);
```

Charger des Données

Sources de Données

Fichiers Locaux/Cloud Storage

- CSV, JSON, Avro, Parquet, ORC
- Via console, bq CLI, API

Streaming

- API insertAll
- Dataflow
- Temps réel, pas de quota gratuit

Federated Queries

- Cloud SQL
- Cloud Spanner
- Google Sheets

Data Transfer Service

- SaaS (Google Ads, YouTube, etc.)
- Schedulé automatiquement

Exemple Chargement CSV

Fichier users.csv

```
user_id,name,email,age
1,Alice,alice@example.com,30
2,Bob,bob@example.com,25
3,Charlie,charlie@example.com,35
```

Commande

```
bq load \
  --source_format=CSV \
  --skip_leading_rows=1 \
  my_dataset.users \
  users.csv \
  user_id:INTEGER,name:STRING,email:STRING,age:INTEGER
```

Exporter des Données

Vers Cloud Storage

```
bq extract \
  --destination_format=CSV \
  --compression=GZIP \
  my_dataset.my_table \
  gs://my-bucket/export/data-*.csv.gz
```

Formats Supportés

- CSV
- JSON (newline delimited)
- Avro
- Parquet

Limitations

- Export max 1 GB par fichier (utiliser wildcard *)
- Pas d'export direct vers local (passer par GCS)

Alternative : Téléchargement

Pour petits volumes, download direct depuis console

Quotas et Limites

Quotas Requêtes

- On-demand : 2 000 requêtes concurrentes/projet
- 100 TB données traitées/jour (modifiable)

Quotas Chargement

- 1 500 jobs de chargement/table/jour
- 15 TB données chargées/table/jour
- 10 GB taille max fichier CSV

Quotas Streaming

- 1 GB/sec par projet
- 100 000 lignes/sec par table

Limites Tables

- 20 000 tables par dataset
- 10 000 partitions par table partitionnée
- 4 colonnes clustering max

Atelier Pratique : Premiers Pas

Exercice 1 : Exploration Dataset Public

1. Ouvrir console BigQuery
2. Explorer bigquery-public-data
3. Examiner le schéma de usa_names.usa_1910_2013
4. Exécuter requête pour trouver les prénoms populaires

Exercice 2 : Créer Dataset et Table

1. Créer dataset personnel
2. Créer table avec données de test
3. Charger données depuis CSV
4. Exécuter requêtes SELECT

Atelier Pratique : Premiers Pas

Exercice 3 : Export

1. Exporter résultats vers Cloud Storage
2. Télécharger et vérifier données

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 2

Architecture et Fonctionnement

Objectifs du Module 2

- Comprendre l'architecture interne de BigQuery
- Découvrir Dremel et Colossus
- Apprendre le stockage en colonnes
- Comprendre l'exécution distribuée
- Explorer le shuffle service
- Maîtriser le caching

Architecture BigQuery

Composants Principaux

Dremel

Moteur d'exécution de requêtes

- Analyse SQL
- Optimisation du plan d'exécution
- Coordination de l'exécution distribuée
- Agrégation des résultats

Colossus

Système de stockage distribué Google

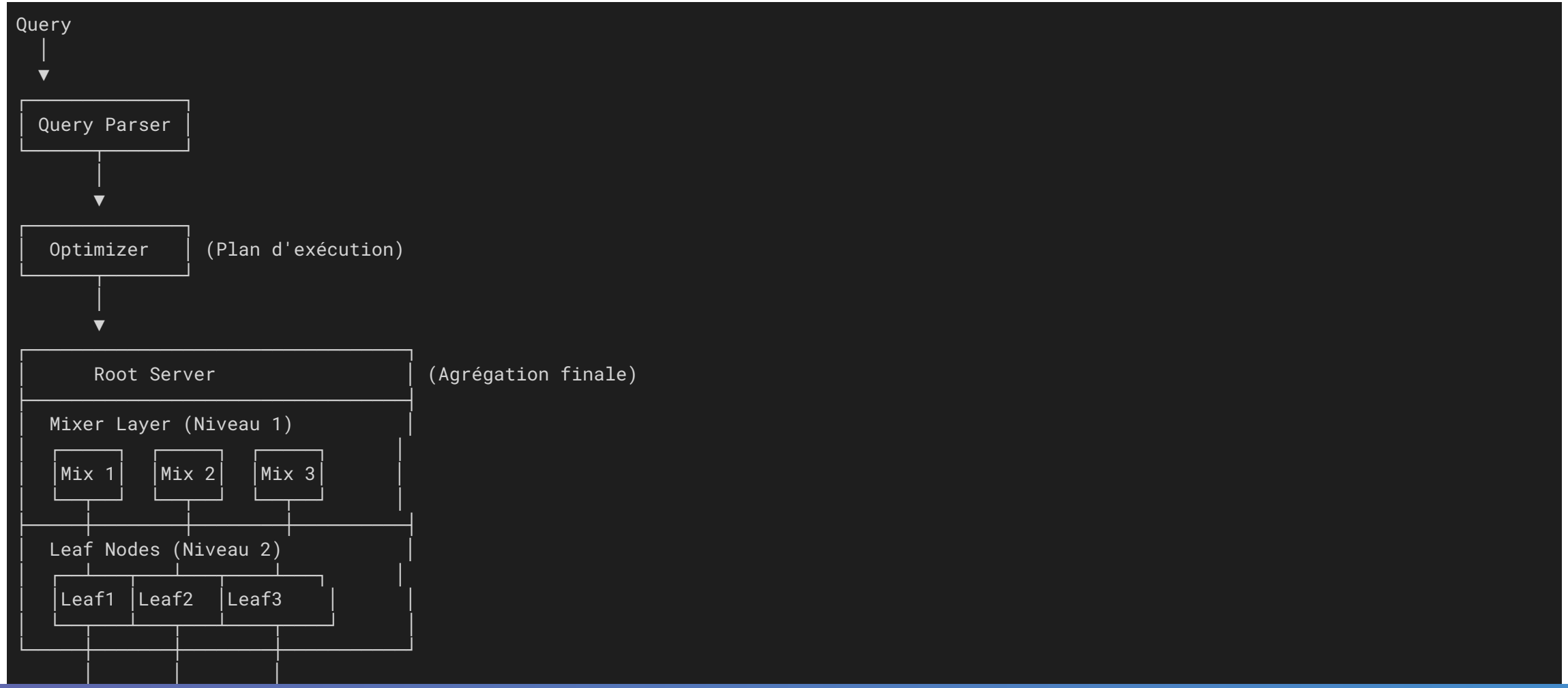
- Réplication automatique
- Haute disponibilité
- Compression
- Encryption

Borg / Kubernetes

Orchestration des ressources

- Allocation dynamique de slots
- Isolation des workloads

Dremel - Moteur d'Exécution



Stockage en Colonnes

Row-Oriented vs Column-Oriented

Row Storage (traditionnel)

```
[1, Alice, alice@example.com, 30]
[2, Bob, bob@example.com, 25]
[3, Charlie, charlie@example.com, 35]
```

Lecture séquentielle, bon pour OLTP

Column Storage (BigQuery)

```
user_id:   [1, 2, 3]
name:      [Alice, Bob, Charlie]
email:     [alice@example.com, bob@example.com, charlie@example.com]
age:       [30, 25, 35]
```

Avantages

- Lecture seulement colonnes nécessaires
- Compression efficace (valeurs similaires)
- Agrégations rapides
- I/O réduit

Compression et Encodage

Techniques de Compression

Run-Length Encoding (RLE)

Valeurs répétitives

```
[A, A, A, B, B, C, C, C, C]
→ [A:3, B:2, C:4]
```

Dictionary Encoding

Valeurs avec cardinalité faible

```
[Paris, London, Paris, Tokyo, London]
→ Dictionary: {0:Paris, 1:London, 2:Tokyo}
→ [0, 1, 0, 2, 1]
```

Bit Packing

Entiers petits stockés sur moins de bits

Résultat

Ratio compression typique 10:1, jusqu'à 100:1 pour certaines données

Capacité et Slots

Qu'est-ce qu'un Slot ?

Unité de capacité de calcul dans BigQuery

- CPU virtuel
- Mémoire
- Réseau

Allocation Dynamique

On-Demand

- 2 000 slots par défaut (peut varier)
- Partagés entre requêtes
- Auto-scaling

Flat-Rate

- Slots dédiés
- Garantis
- Planification de capacité

Exemple

Requête simple : 10 slots

Requête complexe : 500+ slots

Exécution de Requête en Détail

Phases

1. **Parsing et Validation**
   - Syntaxe SQL vérifiée
   - Résolution des noms
   - Vérification permissions
2. **Optimization**
   - Réécriture de requête
   - Choix des index
   - Pruning de partitions
   - Statistiques

Exécution de Requête en Détail

3. **Scheduling**
   - Allocation de slots
   - Placement de workers
4. **Execution**
   - Lecture parallèle depuis Colossus
   - Transformation des données
   - Shuffles si nécessaire
5. **Aggregation**
   - Combinaison des résultats
   - Retour au client

Shuffle Service

Qu'est-ce qu'un Shuffle ?

Redistribution des données entre workers pour opérations comme JOIN, GROUP BY, ORDER BY

Problème Traditionnel

- Goulot d'étranglement
- Spill to disk

BigQuery Shuffle Service

- Service géré séparé
- Stockage temporaire distribué
- Pas de spill to disk local
- Performance améliorée pour gros shuffles

Shuffle Service

Quand se produit-il ?

```
-- Nécessite shuffle
SELECT user_id, COUNT(*)
FROM large_table
GROUP BY user_id;  -- Redistribution par user_id

-- Nécessite shuffle
SELECT a.*, b.*
FROM table_a a
JOIN table_b b ON a.id = b.id;  -- Redistribution pour JOIN
```

Partitionnement

Définition

Division d'une table en segments basés sur une colonne

Types de Partitionnement

Time-unit column

DATE, TIMESTAMP, DATETIME

```
CREATE TABLE `dataset.events` (
  event_id INT64,
  event_date DATE,
  data STRING
)
PARTITION BY event_date;
```

Ingestion time

Automatique basé sur temps d'insertion

```
CREATE TABLE `dataset.logs`
PARTITION BY _PARTITIONTIME;
```

Partitionnement

Integer range

Pour colonnes entières

```
CREATE TABLE `dataset.users`
PARTITION BY RANGE_BUCKET(user_id, GENERATE_ARRAY(0, 1000000, 1000));
```

Avantages du Partitionnement

Performance

Pruning de partitions : lecture seulement partitions nécessaires

Exemple

```
-- Sans partitionnement : scan 1 TB
SELECT * FROM `dataset.events`
WHERE event_date = '2024-01-15';

-- Avec partitionnement : scan 1 GB (1 partition)
-- BigQuery lit seulement la partition du 15 janvier
```

Coûts Réduits

Moins de données traitées = moins cher

Gestion

- Expiration automatique de partitions anciennes
- Pas de coût de maintenance
- Transparent pour utilisateur

Limite

4 000 partitions par table

Clustering

Définition

Tri automatique des données dans BigQuery basé sur colonnes spécifiées

Syntaxe

```
CREATE TABLE `dataset.events` (
  user_id INT64,
  country STRING,
  event_date DATE,
  amount FLOAT64
)
PARTITION BY event_date
CLUSTER BY user_id, country;
```

Clustering

Avantages

- Améliore performance des filtres et agrégations
- Block-level pruning
- Combine avec partitionnement
- Pas de limite de cardinalité

Quand Utiliser

- Colonnes fréquemment filtrées
- Cardinalité haute (> 10K valeurs)
- JOINs sur colonnes

Partitionnement vs Clustering

Aspect	Partitionnement	Clustering
Nombre colonnes	1 seule	Jusqu'à 4
Cardinalité	Basse à moyenne	Haute recommandée
Limite segments	4 000 partitions	Pas de limite
Maintenance	Aucune	Auto-reclustering
Coût amélioration	Très prévisible	Approximatif
Expiration	Oui (par partition)	Non
Use case	Temps, ID ranges	Filtres multiples

Partitionnement vs Clustering

Best Practice

Combiner les deux : PARTITION BY date CLUSTER BY user_id, country

Exemple

```
SELECT * FROM `dataset.events`
WHERE event_date = '2024-01-15'   -- Partition pruning
  AND user_id = 12345              -- Block pruning (clustering)
  AND country = 'FR';              -- Block pruning (clustering)
```

Cache de Résultats

Fonctionnement

BigQuery cache automatiquement les résultats de requêtes pendant 24h

Conditions pour Cache Hit

- Requête identique (SQL exact)
- Tables sources inchangées
- Non-déterminisme absent (CURRENT_TIMESTAMP(), RAND())
- Pas de streaming récent sur table

Avantages

- Gratuit (pas de données traitées)
- Instantané
- Réduit charge

Désactiver Cache

```
-- Query settings
-- Cache des résultats : Désactivé
```

Use Case Cache

Dashboards, requêtes répétées

BI Engine

Service d'Accélération en Mémoire
Caractéristiques

- Cache intelligent
- Optimisé pour BI et dashboards
- Sub-seconde latency
- Transparent pour applications

BI Engine

Activation

```
bq mk --reservation --location=US \
  --bi_capacity_bytes=10737418240 \
  bi_reservation
```

Tarification

- Payé par GB de capacité
- Environ 0,48 USD/GB/mois (US)

Use Cases

- Looker, Data Studio dashboards
- Requêtes interactives répétées
- Exploration de données

Matérialized Views

Vues Matérialisées

Résultats précalculés et stockés d'une requête

Syntaxe

```
CREATE MATERIALIZED VIEW `dataset.daily_summary`
AS
SELECT
  event_date,
  user_id,
  COUNT(*) as event_count,
  SUM(amount) as total_amount
FROM `dataset.events`
GROUP BY event_date, user_id;
```

Matérialized Views

Avantages

- Performance : Pas de recompute
- Coût réduit : Pas de scan table de base
- Refresh automatique
- Transparence : Query rewriter peut les utiliser

Refresh

Automatique et intelligent (incrémental quand possible)

Query Rewriting

Optimisation Automatique

BigQuery réécrit automatiquement les requêtes pour utiliser materialized views quand bénéfique

Exemple

```
-- Requête originale
SELECT event_date, user_id, COUNT(*)
FROM `dataset.events`
GROUP BY event_date, user_id;

-- BigQuery réécrit automatiquement pour utiliser MV
SELECT event_date, user_id, event_count
FROM `dataset.daily_summary`;
```

Conditions

- MV couvre la requête
- Tables de base inchangées ou MV à jour
- Bénéfice performance/coût

Transparence

Pas de modification du code applicatif

Monitoring des Requêtes

INFORMATION_SCHEMA

Métadonnées sur jobs, tables, vues

```sql
-- Jobs récents
SELECT
  job_id,
  user_email,
  query,
  total_bytes_processed,
  creation_time
FROM `region-us`.INFORMATION_SCHEMA.JOBS_BY_PROJECT
WHERE creation_time > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
ORDER BY creation_time DESC
LIMIT 100;

-- Tables les plus volumineuses
SELECT
  table_name,
  size_bytes / POW(10, 9) as size_gb
FROM `dataset`.INFORMATION_SCHEMA.TABLES
ORDER BY size_bytes DESC
LIMIT 10;
```

Explain Plan

Analyser Plans d'Exécution

Console BigQuery

- Après exécution : onglet "Execution Details"
- Visualisation graphique du plan
- Temps par étape
- Bytes shuffled, rows processed

CLI

```
bq query --dry_run --use_legacy_sql=false \
'SELECT * FROM dataset.table WHERE date = "2024-01-01"'
```

Métriques Importantes

- Bytes processed (coût)
- Execution time par stage
- Shuffle bytes
- Partition pruning effectué

Atelier Pratique : Architecture

Exercice 1 : Partitionnement

1. Créer table partitionnée par date
2. Insérer données sur plusieurs dates
3. Comparer performance requêtes avec/sans filtre date
4. Observer bytes processed

Exercice 2 : Clustering

1. Créer table clusterisée
2. Exécuter requêtes avec filtres sur colonnes cluster
3. Analyser execution plan
4. Comparer avec table non-clusterisée

Atelier Pratique : Architecture

Exercice 3 : Materialized Views

1. Créer materialized view avec agrégations
2. Exécuter requête couverte par MV
3. Vérifier que MV est utilisée (execution plan)
4. Mesurer amélioration performance

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 3

SQL Avancé dans BigQuery

Objectifs du Module 3

- Maîtriser les fonctions SQL BigQuery
- Utiliser les fonctions d'agrégation avancées
- Manipuler les types de données complexes
- Travailler avec ARRAY et STRUCT
- Utiliser les fonctions de fenêtre
- Comprendre les WITH clauses et CTEs

Types de Données BigQuery

Types Scalaires

- INT64, FLOAT64, NUMERIC, BIGNUMERIC
- STRING, BYTES
- BOOL
- DATE, DATETIME, TIME, TIMESTAMP
- GEOGRAPHY, JSON (preview)

Types Complexes

- ARRAY : Liste ordonnée de valeurs du même type
- STRUCT : Collection de champs nommés et typés

Exemples

```
-- ARRAY
ARRAY<INT64>  → [1, 2, 3, 4, 5]
ARRAY<STRING> → ['a', 'b', 'c']

-- STRUCT
STRUCT<name STRING, age INT64> → {name: 'Alice', age: 30}

-- Combinaison
ARRAY<STRUCT<name STRING, age INT64>>
→ [{name: 'Alice', age: 30}, {name: 'Bob', age: 25}]
```

Travailler avec ARRAY

Créer un ARRAY

```
-- Littéral
SELECT [1, 2, 3, 4, 5] as numbers;

-- ARRAY_AGG
SELECT ARRAY_AGG(product_id) as products
FROM orders
WHERE user_id = 123;

-- GENERATE_ARRAY
SELECT GENERATE_ARRAY(1, 10, 2) as odd_numbers;  -- [1,3,5,7,9]
```

Accéder aux Éléments

```
SELECT
  products,
  products[OFFSET(0)] as first_product,    -- Premier élément (0-indexed)
  products[ORDINAL(1)] as first_product,   -- Premier élément (1-indexed)
  ARRAY_LENGTH(products) as count
FROM user_orders;
```

UNNEST - Aplatir les ARRAY

Transformer ARRAY en Lignes

```
-- Table avec ARRAY
CREATE TEMP TABLE orders AS
SELECT 1 as order_id, ['apple', 'banana', 'cherry'] as items
UNION ALL
SELECT 2, ['banana', 'date'];

-- UNNEST
SELECT
  order_id,
  item
FROM orders
CROSS JOIN UNNEST(items) as item;

-- Résultat:
-- order_id | item
-- 1        | apple
-- 1        | banana
-- 1        | cherry
-- 2        | banana
-- 2        | date
```

Use Case

Données nested JSON importées dans BigQuery

Travailler avec STRUCT

Créer un STRUCT

```
-- Littéral
SELECT STRUCT('Alice' as name, 30 as age, 'FR' as country) as user;

-- Depuis colonnes
SELECT
  STRUCT(name, age, country) as user_info
FROM users;
```

Accéder aux Champs

```
SELECT
  user_info.name,
  user_info.age,
  user_info.country
FROM user_table;
```

ARRAY de STRUCT

```
SELECT
  order_id,
  ARRAY_AGG(STRUCT(product_id, quantity, price)) as items
FROM order_items
GROUP BY order_id;
```

Fonctions d'Agrégation

Standards

```
SELECT
  COUNT(*) as total_orders,
  COUNT(DISTINCT user_id) as unique_users,
  SUM(amount) as total_revenue,
  AVG(amount) as avg_order_value,
  MIN(amount) as min_order,
  MAX(amount) as max_order
FROM orders;
```

Avancées

```
-- COUNTIF
SELECT COUNTIF(status = 'completed') as completed_orders
FROM orders;

-- STRING_AGG
SELECT user_id, STRING_AGG(product_name, ', ') as products
FROM purchases
GROUP BY user_id;

-- ARRAY_AGG avec ORDER BY et LIMIT
SELECT
  category,
  ARRAY_AGG(product ORDER BY sales DESC LIMIT 5) as top_products
FROM products
```

Fonctions de Fenêtre (Window Functions)

Syntaxe Générale

```
fonction() OVER (
  [PARTITION BY col1, col2]
  [ORDER BY col3]
  [ROWS/RANGE frame_specification]
)
```

Fonctions de Classement

```
SELECT
  product_name,
  sales,
  ROW_NUMBER() OVER (ORDER BY sales DESC) as row_num,
  RANK() OVER (ORDER BY sales DESC) as rank,
  DENSE_RANK() OVER (ORDER BY sales DESC) as dense_rank,
  NTILE(4) OVER (ORDER BY sales DESC) as quartile
FROM products;
```

Différence : RANK gaps vs DENSE_RANK continu

Fonctions de Fenêtre Analytiques

Agrégations Fenêtre

```
SELECT
  date,
  revenue,
  SUM(revenue) OVER (ORDER BY date) as cumulative_revenue,
  AVG(revenue) OVER (
    ORDER BY date
    ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
  ) as moving_avg_7days
FROM daily_sales;
```

LEAD et LAG

```
SELECT
  date,
  revenue,
  LAG(revenue, 1) OVER (ORDER BY date) as prev_day_revenue,
  LEAD(revenue, 1) OVER (ORDER BY date) as next_day_revenue,
  revenue - LAG(revenue, 1) OVER (ORDER BY date) as day_over_day_change
FROM daily_sales;
```

PARTITION BY dans Window Functions

Par Groupe

```sql
-- Top 3 produits par catégorie
SELECT *
FROM (
  SELECT
    category,
    product_name,
    sales,
    ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) as rn
  FROM products
)
WHERE rn <= 3;

-- Part de marché
SELECT
  category,
  product_name,
  sales,
  SUM(sales) OVER (PARTITION BY category) as category_total,
  sales / SUM(sales) OVER (PARTITION BY category) * 100 as market_share_pct
FROM products;
```

Common Table Expressions (CTE)

WITH Clause

```
WITH
  daily_stats AS (
    SELECT
      DATE(created_at) as date,
      COUNT(*) as orders,
      SUM(amount) as revenue
    FROM orders
    GROUP BY date
  ),
  moving_avg AS (
    SELECT
      date,
      revenue,
      AVG(revenue) OVER (
        ORDER BY date
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
      ) as ma_7d
    FROM daily_stats
  )
SELECT * FROM moving_avg
WHERE date >= '2024-01-01';
```

Avantages

- Lisibilité

CTE Récursives

Hiérarchies et Graphes

```sql
WITH RECURSIVE employee_hierarchy AS (
  -- Base case : Top managers
  SELECT employee_id, name, manager_id, 1 as level
  FROM employees
  WHERE manager_id IS NULL

  UNION ALL

  -- Recursive case
  SELECT e.employee_id, e.name, e.manager_id, eh.level + 1
  FROM employees e
  JOIN employee_hierarchy eh ON e.manager_id = eh.employee_id
)
SELECT * FROM employee_hierarchy
ORDER BY level, employee_id;
```

Use Cases

- Organigrammes
- Bill of materials
- Chemins dans graphes

Fonctions de Date et Temps

Extraction

```
SELECT
  EXTRACT(YEAR FROM created_at) as year,
  EXTRACT(MONTH FROM created_at) as month,
  EXTRACT(WEEK FROM created_at) as week,
  DATE(created_at) as date_only,
  TIME(created_at) as time_only
FROM orders;
```

Manipulation

```
SELECT
  DATE_ADD(CURRENT_DATE(), INTERVAL 7 DAY) as next_week,
  DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH) as last_month,
  DATE_DIFF(CURRENT_DATE(), '2024-01-01', DAY) as days_since_ny,
  TIMESTAMP_DIFF(
    CURRENT_TIMESTAMP(),
    created_at,
    HOUR
  ) as hours_ago
FROM orders;
```

Fonctions de Date - Truncate

DATE_TRUNC

```
SELECT
  DATE_TRUNC(created_at, MONTH) as month,
  COUNT(*) as orders,
  SUM(amount) as revenue
FROM orders
GROUP BY month
ORDER BY month;

-- Autres intervalles
DATE_TRUNC(date, YEAR)    -- Premier jour de l'année
DATE_TRUNC(date, QUARTER) -- Premier jour du trimestre
DATE_TRUNC(date, WEEK)    -- Lundi de la semaine
DATE_TRUNC(date, DAY)     -- Minuit du jour
```

FORMAT_DATE

```
SELECT
  FORMAT_DATE('%Y-%m-%d', created_at) as iso_date,
  FORMAT_DATE('%B %d, %Y', created_at) as readable_date,
  FORMAT_DATE('%A', created_at) as day_of_week
FROM orders;
```

Fonctions de String

Manipulation

```
SELECT
  CONCAT(first_name, ' ', last_name) as full_name,
  UPPER(email) as email_upper,
  LOWER(country) as country_lower,
  LENGTH(description) as desc_length,
  SUBSTR(phone, 1, 3) as area_code,
  TRIM(name) as name_trimmed,
  REPLACE(text, 'old', 'new') as replaced
FROM users;
```

Pattern Matching

```
-- LIKE
WHERE email LIKE '%@gmail.com'

-- REGEXP_CONTAINS
WHERE REGEXP_CONTAINS(email, r'@(gmail|yahoo|hotmail)\.com$')

-- REGEXP_EXTRACT
SELECT REGEXP_EXTRACT(email, r'@(.+)') as domain
```

Fonctions JSON

Parsing JSON

```sql
-- JSON_EXTRACT_SCALAR : Extraire valeur scalaire
SELECT
  JSON_EXTRACT_SCALAR(json_data, '$.name') as name,
  JSON_EXTRACT_SCALAR(json_data, '$.age') as age,
  JSON_EXTRACT_SCALAR(json_data, '$.address.city') as city
FROM events;

-- JSON_EXTRACT : Extraire objet/array
SELECT
  JSON_EXTRACT(json_data, '$.items') as items_array,
  JSON_EXTRACT_ARRAY(json_data, '$.items') as items
FROM events;

-- TO_JSON_STRING : Convertir en JSON
SELECT TO_JSON_STRING(STRUCT(name, age, country)) as json
FROM users;
```

PIVOT et UNPIVOT

PIVOT (Standard SQL)

```sql
-- Transformer lignes en colonnes
WITH sales AS (
  SELECT 'Q1' as quarter, 'Product A' as product, 100 as amount
  UNION ALL SELECT 'Q2', 'Product A', 150
  UNION ALL SELECT 'Q1', 'Product B', 200
  UNION ALL SELECT 'Q2', 'Product B', 250
)
SELECT *
FROM sales
PIVOT(SUM(amount) FOR quarter IN ('Q1', 'Q2'));

-- Résultat:
-- product    | Q1  | Q2
-- Product A  | 100 | 150
-- Product B  | 200 | 250
```

Alternative : Agrégation avec CASE

```sql
SELECT
  product,
  SUM(CASE WHEN quarter = 'Q1' THEN amount ELSE 0 END) as Q1,
  SUM(CASE WHEN quarter = 'Q2' THEN amount ELSE 0 END) as Q2
FROM sales
GROUP BY product;
```

User-Defined Functions (UDF)

SQL UDF

```
CREATE TEMP FUNCTION celsius_to_fahrenheit(celsius FLOAT64)
RETURNS FLOAT64
AS (
  (celsius * 9/5) + 32
);

SELECT
  city,
  temp_celsius,
  celsius_to_fahrenheit(temp_celsius) as temp_fahrenheit
FROM weather;
```

JavaScript UDF

```
CREATE TEMP FUNCTION parse_user_agent(ua STRING)
RETURNS STRING
LANGUAGE js AS """
  var parser = require('ua-parser-js');
  var result = parser(ua);
  return result.browser.name;
""";

SELECT parse_user_agent(user_agent) as browser
FROM web_logs;
```

Procédures Stockées

Création

```
CREATE OR REPLACE PROCEDURE `dataset.update_user_status`()
BEGIN
  -- Variables
  DECLARE rows_updated INT64;

  -- Logic
  UPDATE `dataset.users`
  SET status = 'inactive'
  WHERE last_login < DATE_SUB(CURRENT_DATE(), INTERVAL 90 DAY);

  SET rows_updated = @@row_count;

  -- Log
  INSERT INTO `dataset.audit_log` (action, rows_affected, timestamp)
  VALUES ('deactivate_users', rows_updated, CURRENT_TIMESTAMP());
END;
```

Exécution

```
CALL `dataset.update_user_status`();
```

Scripting dans BigQuery

Variables et Control Flow

```
DECLARE row_count INT64;
DECLARE status STRING;

-- Requête dans variable
SET row_count = (
  SELECT COUNT(*)
  FROM `dataset.orders`
  WHERE status = 'pending'
);

-- Conditions
IF row_count > 1000 THEN
  SET status = 'High';
ELSEIF row_count > 100 THEN
  SET status = 'Medium';
ELSE
  SET status = 'Low';
END IF;

-- Loops
WHILE row_count > 0 DO
  -- Process batch
  SET row_count = row_count - 100;
END WHILE;
```

Atelier Pratique : SQL Avancé

Exercice 1 : ARRAY et STRUCT

1. Créer table avec colonnes ARRAY
2. Utiliser ARRAY_AGG pour agréger données
3. UNNEST pour aplatir
4. Créer STRUCT avec informations imbriquées

Exercice 2 : Window Functions

1. Calculer running total
2. Ranking de produits par catégorie
3. Moving average sur 7 jours
4. Identifier top/bottom performers

Atelier Pratique : SQL Avancé

Exercice 3 : CTE et UDF

1. CTE multi-niveaux pour analyse complexe
2. Créer UDF pour calcul métier
3. Procédure stockée pour ETL

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 4

Optimisation et Performance

Objectifs du Module 4

- Identifier les requêtes inefficaces
- Optimiser les requêtes SQL
- Utiliser les bonnes pratiques de partitionnement
- Minimiser les coûts
- Comprendre les anti-patterns
- Monitorer les performances

Principes d'Optimisation

Objectifs

- Réduire données traitées
- Minimiser I/O
- Éviter shuffles coûteux
- Réutiliser résultats (cache, MV)

Approche

1. Mesurer performance actuelle
2. Identifier bottlenecks
3. Appliquer optimisations
4. Valider amélioration
5. Itérer

Outils

- Execution plan
- Query validator (dry-run)
- INFORMATION_SCHEMA

Éviter SELECT *

Anti-Pattern

```
-- BAD : Traite toutes les colonnes
SELECT *
FROM `project.dataset.large_table`
WHERE date = '2024-01-01';
-- Bytes processed : 10 GB
```

Best Practice

```
-- GOOD : Seulement colonnes nécessaires
SELECT user_id, event_name, timestamp
FROM `project.dataset.large_table`
WHERE date = '2024-01-01';
-- Bytes processed : 500 MB
```

Impact

Réduction 95% du coût et amélioration performance

Exceptions

- Exploration initiale (utiliser LIMIT)
- SELECT COUNT(*)

Filtrer Tôt et Fort

Pruning de Partitions

```
-- Utiliser colonne de partitionnement dans WHERE
SELECT user_id, event_name
FROM `dataset.events`
WHERE _PARTITIONDATE = '2024-01-01'  -- Partition pruning
  AND event_name = 'purchase';        -- Filtre supplémentaire
```

Colonnes de Clustering

```
-- Filtrer sur colonnes cluster
WHERE user_id = 12345      -- Block-level pruning
  AND country = 'FR';
```

Ordre des Filtres

BigQuery optimise automatiquement, mais logique claire aide

Anti-Pattern

```
-- Filtre après calculs coûteux
SELECT * FROM (
  SELECT *, COMPLEX_FUNCTION(data) as result
  FROM large_table
)
WHERE date = '2024-01-01';  -- Trop tard
```

Optimiser les JOINs

Ordre des Tables

BigQuery optimise automatiquement, mais aide :

- Grande table à gauche
- Petite table à droite (broadcast join)

JOIN sur Colonnes Clusterisées

```
-- Tables clusterisées sur user_id
SELECT a.*, b.*
FROM `dataset.events` a
JOIN `dataset.users` b ON a.user_id = b.user_id
WHERE a._PARTITIONDATE = '2024-01-01';
```

Éviter JOINs Croisés

```
-- TRÈS COÛTEUX : Cross join
SELECT * FROM table_a, table_b;  -- N * M lignes

-- Mieux : JOIN avec condition
SELECT * FROM table_a a
JOIN table_b b ON a.id = b.id;
```

Pré-Agrégation

Principe

Agréger avant JOIN pour réduire volume

Anti-Pattern

```
-- JOIN puis agrégation
SELECT
  u.country,
  COUNT(*) as orders
FROM orders o
JOIN users u ON o.user_id = u.user_id
GROUP BY u.country;
```

Optimisé

```
-- Agrégation avant JOIN
WITH order_counts AS (
  SELECT user_id, COUNT(*) as order_count
  FROM orders
  GROUP BY user_id
)
SELECT
  u.country,
  SUM(oc.order_count) as orders
FROM order_counts oc
JOIN users u ON oc.user_id = u.user_id
GROUP BY u.country;
```

Éviter DISTINCT Coûteux

Problème

DISTINCT nécessite shuffle pour dédupliquer

Alternative : GROUP BY

```
-- Potentiellement lent
SELECT DISTINCT user_id
FROM events;

-- Souvent plus rapide
SELECT user_id
FROM events
GROUP BY user_id;
```

Éviter DISTINCT Inutile

```
-- Si user_id déjà unique dans events par date
SELECT user_id
FROM events
WHERE date = '2024-01-01';
-- Pas besoin de DISTINCT
```

APPROX_COUNT_DISTINCT

Pour comptages approximatifs (erreur < 1%)

```
SELECT APPROX_COUNT_DISTINCT(user_id) as approx_users
```

Optimiser GROUP BY

Cardinalité Élevée

Grouper sur colonnes avec beaucoup de valeurs uniques coûte cher

Technique : Pré-filtrage

```sql
-- Réduire données avant GROUP BY
SELECT
  product_id,
  COUNT(*) as sales
FROM orders
WHERE date >= '2024-01-01'  -- Filtrer d'abord
GROUP BY product_id;
```

Approximate Aggregation

```sql
-- Pour estimations rapides
SELECT
  country,
  APPROX_COUNT_DISTINCT(user_id) as approx_users,
  APPROX_QUANTILES(amount, 100)[OFFSET(50)] as median_amount
FROM orders
GROUP BY country;
```

Éviter Sous-Requêtes Corrélées

Anti-Pattern

```
-- Sous-requête corrélée : Exécutée pour chaque ligne
SELECT
  user_id,
  (SELECT COUNT(*) FROM orders o WHERE o.user_id = u.user_id) as order_count
FROM users u;
```

Optimisé : JOIN

```
WITH order_counts AS (
  SELECT user_id, COUNT(*) as order_count
  FROM orders
  GROUP BY user_id
)
SELECT
  u.user_id,
  COALESCE(oc.order_count, 0) as order_count
FROM users u
LEFT JOIN order_counts oc ON u.user_id = oc.user_id;
```

Performance

Amélioration 10-100x possible

Gérer les Données Skewed

Problème : Data Skew

Distribution inégale des données cause déséquilibre workers

Exemple

Grouper par country : USA = 80% des données

Solution 1 : Salting

```
-- Ajouter randomness
SELECT
  country,
  MOD(ABS(FARM_FINGERPRINT(CAST(user_id AS STRING))), 10) as salt,
  COUNT(*) as cnt
FROM users
GROUP BY country, salt;

-- Puis agréger
SELECT country, SUM(cnt) as total
FROM previous_result
GROUP BY country;
```

Solution 2 : Traitement séparé

Traiter gros segments séparément

Utiliser Approximate Functions

Fonctions Approximatives

Résultats avec petite marge d'erreur, beaucoup plus rapides

```
-- Exact (peut être lent)
SELECT
  COUNT(DISTINCT user_id) as exact_users,
  PERCENTILE_CONT(amount, 0.5) OVER() as exact_median
FROM large_table;

-- Approximatif (rapide)
SELECT
  APPROX_COUNT_DISTINCT(user_id) as approx_users,
  APPROX_QUANTILES(amount, 100)[OFFSET(50)] as approx_median,
  APPROX_TOP_COUNT(product_id, 10) as top_products
FROM large_table;
```

Quand Utiliser

- Exploration de données
- Dashboards temps réel
- Métriques où précision 100% non critique

Dry Run pour Estimation Coût

Vérifier Avant Exécution

Console

Validator affiche bytes qui seront traités

CLI

```
bq query --dry_run --use_legacy_sql=false \
'SELECT user_id, COUNT(*) FROM `dataset.large_table` GROUP BY user_id'

# Output : This query will process X GB when run.
```

Python

```
from google.cloud import bigquery

client = bigquery.Client()
job_config = bigquery.QueryJobConfig(dry_run=True)

query_job = client.query(query, job_config=job_config)

print(f"This query will process {query_job.total_bytes_processed} bytes")
print(f"Estimated cost: ${query_job.total_bytes_processed / 1e12 * 6.25:.2f}")
```

Monitoring des Requêtes

Identifier Requêtes Coûteuses

```sql
SELECT
  user_email,
  job_id,
  query,
  total_bytes_processed / POW(10, 12) as tb_processed,
  total_bytes_billed / POW(10, 12) as tb_billed,
  total_slot_ms / 1000 / 60 as slot_minutes,
  TIMESTAMP_DIFF(end_time, start_time, SECOND) as duration_sec
FROM `region-us`.INFORMATION_SCHEMA.JOBS_BY_PROJECT
WHERE
  creation_time >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)
  AND job_type = 'QUERY'
  AND state = 'DONE'
ORDER BY tb_billed DESC
LIMIT 50;
```

Métriques Clés

- total_bytes_billed : Coût
- slot_ms : Ressources compute
- duration : Temps d'exécution

Quotas et Rate Limits

Quotas Principaux

API Requests

- 300 requêtes/sec par projet (can be increased)

Concurrent Queries

- On-demand : 100 concurrent (interactive), 50 (batch)
- Reservations : Selon slots réservés

Slot Usage

- On-demand : 2 000 slots (peut varier)

Actions si Limites Atteintes

- Optimiser requêtes (réduire complexité)
- Espacer requêtes (retry with backoff)
- Upgrading vers flat-rate si workload constant
- Demander augmentation quotas

Best Practices Résumé

Schéma

- Partitionner sur colonnes temps/ID fréquemment filtrées
- Clustériser sur colonnes utilisées dans WHERE/JOIN
- Types de données appropriés (INT64 vs STRING pour IDs)

Requêtes

- SELECT colonnes spécifiques, éviter SELECT *
- Filtrer sur partitions et clusters
- Pré-agréger avant JOINs
- Éviter sous-requêtes corrélées
- Utiliser approximate functions quand approprié

Coûts

- Dry-run avant exécution
- Utiliser cache (requêtes répétées)
- Materialized views pour agrégations récurrentes
- Scheduler requêtes batch hors heures de pointe

Anti-Patterns Courants

1. Partitions Trop Fines

```
-- BAD : Partition par heure (35K partitions/an)
PARTITION BY DATETIME_TRUNC(timestamp, HOUR)

-- GOOD : Par jour (365 partitions/an)
PARTITION BY DATE(timestamp)
```

2. Self-JOIN Inutile

```
-- BAD
SELECT a.*, b.other_col
FROM table a
JOIN table a b ON a.id = b.id;

-- GOOD
SELECT *, other_col FROM table;
```

3. Clustering sur Colonnes Haute Cardinalité Seulement

Si user_id très unique, ajoutez colonne cardinalité plus basse

Atelier Pratique : Optimisation

Exercice 1 : Audit Requêtes

1. Identifier top 10 requêtes coûteuses du projet
2. Analyser leurs execution plans
3. Lister optimisations possibles

Exercice 2 : Optimisation Pratique

1. Créer table non-optimisée
2. Exécuter requête inefficace, mesurer
3. Appliquer partitionnement et clustering
4. Ré-exécuter, comparer bytes processed et temps

Atelier Pratique : Optimisation

Exercice 3 : Refactoring

1. Prendre requête avec sous-requête corrélée
2. Refactorer avec CTE et JOIN
3. Mesurer amélioration performance

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 5

Sécurité et Gouvernance

Objectifs du Module 5

- Comprendre le modèle de sécurité BigQuery
- Implémenter IAM et contrôles d'accès
- Utiliser le chiffrement des données
- Gérer l'audit et la conformité
- Implémenter la gouvernance des données
- Masquer les données sensibles

Modèle de Sécurité BigQuery

Niveaux de Contrôle

```
Organization
    │
    ├── IAM Policies (Org level)
    │
    └── Project
        ├── IAM Policies (Project level)
        │
        └── Dataset
            ├── IAM Policies (Dataset level)
            ├── Authorized Views
            └── Authorized Routines
                │
                └── Tables
                    └── Column-level security
                        └── Row-level security
```

Principe du Moindre Privilège

Donner seulement permissions nécessaires

Identity and Access Management (IAM)

Rôles Prédéfinis BigQuery

Niveau Project

- BigQuery Admin : Contrôle total
- BigQuery Data Editor : Lire et modifier données
- BigQuery Data Viewer : Lecture seule
- BigQuery Job User : Exécuter requêtes
- BigQuery User : Créer datasets, exécuter requêtes

Niveau Dataset

- BigQuery Data Owner : Contrôle dataset
- BigQuery Data Editor : Modifier tables
- BigQuery Data Viewer : Lecture seule

Granularité

IAM au niveau table non supporté directement (utiliser authorized views)

Accorder des Permissions

Console

```
Dataset → Sharing → Permissions → Add Principal
```

gcloud CLI

```
# Project level
gcloud projects add-iam-policy-binding PROJECT_ID \
  --member="user:analyst@company.com" \
  --role="roles/bigquery.dataViewer"

# Dataset level
bq update --dataset \
  --add_access_entry="role=READER,userByEmail=analyst@company.com" \
  PROJECT_ID:DATASET_ID
```

Python

```
from google.cloud import bigquery

client = bigquery.Client()
dataset = client.get_dataset("project.dataset")
access_entries = list(dataset.access_entries)
access_entries.append(
    bigquery.AccessEntry("READER", "userByEmail", "analyst@company.com")
)
dataset.access_entries = access_entries
```

Authorized Views

Concept

Vue autorisée à accéder dataset source même si utilisateur ne peut pas accéder directement

Use Case

Donner accès à subset de données sans exposer table complète

Exemple

```
-- Dataset source (users n'ont pas accès)
CREATE TABLE `restricted_dataset.sensitive_data` AS
SELECT user_id, name, ssn, salary FROM employees;

-- Vue autorisée dans dataset partagé
CREATE VIEW `shared_dataset.employee_names` AS
SELECT user_id, name
FROM `restricted_dataset.sensitive_data`;

-- Autoriser la vue
bq update --dataset \
  --add_authorized_view="shared_dataset.employee_names" \
  restricted_dataset
```

Row-Level Security

Filtrer Lignes par Utilisateur

Création de Policy

```
CREATE ROW ACCESS POLICY regional_policy
ON `dataset.sales`
GRANT TO ("user:manager-eu@company.com")
FILTER USING (region = 'EU');

CREATE ROW ACCESS POLICY regional_policy_us
ON `dataset.sales`
GRANT TO ("user:manager-us@company.com")
FILTER USING (region = 'US');

-- Admin voit tout
CREATE ROW ACCESS POLICY admin_policy
ON `dataset.sales`
GRANT TO ("user:admin@company.com")
FILTER USING (TRUE);
```

Résultat

Chaque utilisateur voit seulement lignes correspondant à sa policy

Column-Level Security

Taxonomy et Policy Tags

1. **Créer Taxonomy** (Data Catalog)

```
gcloud data-catalog taxonomies create "SensitiveData" \
  --location=us \
  --display-name="Sensitive Data Classification"
```

2. **Créer Policy Tags**

```
gcloud data-catalog taxonomies policy-tags create "PII" \
  --taxonomy="SensitiveData" \
  --display-name="Personally Identifiable Information"
```

3. **Appliquer Tag à Colonne**

```
ALTER TABLE `dataset.users`
ALTER COLUMN email SET OPTIONS (
  policy_tags=('projects/PROJECT/locations/us/taxonomies/TAXONOMY_ID/policyTags/TAG_ID')
);
```

4. **Accorder Accès**

Dynamic Data Masking

Masquage à la Volée

Policy Tag avec Masking Rule

```
gcloud data-catalog taxonomies policy-tags create "PII_Masked" \
  --taxonomy="SensitiveData" \
  --display-name="PII with Default Masking"
```

Configuration Masking

- Default masking : Valeur nulle/hash
- Custom masking : UDF

Application

```
-- Appliquer policy tag avec masking
ALTER TABLE `dataset.customers`
ALTER COLUMN credit_card SET OPTIONS (
  policy_tags=('projects/.../policyTags/PII_MASKED')
);
```

Résultat

Utilisateurs sans permission voient valeurs masquées

```
-- Avec permission : 1234-5678-9012-3456
-- Sans permission : XXXX-XXXX-XXXX-3456
```

Chiffrement des Données

At Rest

Par Défaut

- Google-managed encryption keys
- Transparent, automatique
- Rotation automatique

Customer-Managed Encryption Keys (CMEK)

```
# Créer dataset avec CMEK
bq mk --location=US \
  --default_kms_key=projects/KMS_PROJECT/locations/us/keyRings/RING/cryptoKeys/KEY \
  DATASET_ID
```

In Transit

- TLS automatique pour toutes communications
- Pas de configuration nécessaire

Application Level

Chiffrement additionnel possible dans application avant insertion

Audit Logging

Types de Logs

Admin Activity Logs

- Opérations administratives (create/delete dataset)
- Activé par défaut
- Gratuit

Data Access Logs

- Requêtes exécutées
- Tables accédées
- Doit être activé explicitement
- Payant

System Event Logs

- Événements internes Google
- Automatique

Access Transparency Logs

- Accès Google personnel
- Disponible uniquement certains contrats

Activer Data Access Logs

Console

```
IAM & Admin → Audit Logs → BigQuery API →
Enable "Admin Read", "Data Read", "Data Write"
```

gcloud

```
# Récupérer policy actuelle
gcloud projects get-iam-policy PROJECT_ID > policy.yaml

# Éditer policy.yaml pour ajouter :
auditConfigs:
- auditLogConfigs:
  - logType: DATA_READ
  - logType: DATA_WRITE
  - logType: ADMIN_READ
  service: bigquery.googleapis.com

# Appliquer
gcloud projects set-iam-policy PROJECT_ID policy.yaml
```

Coût

Environ 0,50 USD/GB de logs générés

Analyser les Logs d'Audit

Cloud Logging

```
-- Logs exportés vers BigQuery
SELECT
  timestamp,
  protopayload_auditlog.authenticationInfo.principalEmail as user,
  protopayload_auditlog.resourceName as resource,
  protopayload_auditlog.methodName as method,
  protopayload_auditlog.servicedata_v1_bigquery.jobCompletedEvent.job.jobConfiguration.query.query as query
FROM `project.dataset.cloudaudit_googleapis_com_data_access_*`
WHERE
  DATE(_TABLE_SUFFIX) = CURRENT_DATE()
  AND resource.type = 'bigquery_resource'
ORDER BY timestamp DESC
LIMIT 100;
```

Insights

- Qui accède à quelles données
- Requêtes exécutées
- Tables modifiées

VPC Service Controls

Périmètre de Sécurité

Objectif

Isoler ressources GCP dans périmètre virtuel

Use Cases

- Prévenir exfiltration de données
- Contrôler accès depuis Internet
- Conformité réglementaire

Configuration

```
# Créer périmètre
gcloud access-context-manager perimeters create PERIMETER_NAME \
  --title="BigQuery Data Perimeter" \
  --resources=projects/PROJECT_NUMBER \
  --restricted-services=bigquery.googleapis.com

# Règles ingress/egress
gcloud access-context-manager perimeters update PERIMETER_NAME \
  --set-ingress-policies=INGRESS_POLICY.yaml
```

Résultat

Accès BigQuery limité aux sources autorisées

Data Loss Prevention (DLP) API

Détection Automatique de Données Sensibles

Scan de Table

```
from google.cloud import dlp_v2

dlp = dlp_v2.DlpServiceClient()
project = f'projects/{project_id}'

# Configuration
inspect_config = {
    "info_types": [
        {"name": "EMAIL_ADDRESS"},
        {"name": "PHONE_NUMBER"},
        {"name": "CREDIT_CARD_NUMBER"},
    ],
    "min_likelihood": dlp_v2.Likelihood.LIKELY,
}

# Source BigQuery
storage_config = {
    "big_query_options": {
        "table_reference": {
            "project_id": project_id,
            "dataset_id": dataset_id,
            "table_id": table_id,
        }
```

Data Catalog

Service de Métadonnées

Fonctionnalités

- Découverte de données
- Métadonnées techniques et business
- Recherche unifiée
- Taxonomies et tags

Créer Entry

```
# Automatique pour BigQuery datasets/tables

# Ajouter tags personnalisés
gcloud data-catalog tags create \
  --entry=ENTRY \
  --tag-template=TAG_TEMPLATE \
  --tag-file=tag.json
```

tag.json

```
{
  "owner": "data-team@company.com",
  "pii_contains": true,
  "classification": "confidential"
}
```

Organization Policies

Contraintes au Niveau Organisation

Exemples Policies

Restreindre Locations

```
# Autoriser seulement EU regions
gcloud resource-manager org-policies set-policy \
  --organization=ORG_ID \
  policy.yaml
```

policy.yaml

```
constraint: constraints/gcp.resourceLocations
listPolicy:
  allowedValues:
  - in:eu-locations
```

Autres Contraintes

- Require CMEK
- Disable public datasets
- Restrict dataset sharing

Conformité

Certifications BigQuery

- ISO 27001, 27017, 27018
- SOC 1, 2, 3
- PCI DSS
- HIPAA
- GDPR compliant

Responsabilité Partagée

Google

- Infrastructure sécurisée
- Certifications
- Chiffrement par défaut

Client

- IAM approprié
- Classification données
- Audit logging
- Compliance avec régulations métier

Best Practices Sécurité

IAM

- Principe du moindre privilège
- Utiliser groupes, pas utilisateurs individuels
- Révision régulière des permissions
- Service accounts pour applications

Données Sensibles

- Identifier et classifier (PII, PHI, etc.)
- Column-level security ou masking
- Row-level security si nécessaire
- Authorized views pour partage limité

Audit

- Activer data access logs
- Exporter logs vers BigQuery pour analyse
- Alertes sur activités suspectes
- Révision régulière

Conformité

- Utiliser Organization Policies

Atelier Pratique : Sécurité

Exercice 1 : IAM et Permissions

1. Créer dataset avec différents niveaux d'accès
2. Accorder permissions à différents utilisateurs/groupes
3. Tester accès avec différents comptes
4. Créer authorized view

Exercice 2 : Row-Level Security

1. Créer table avec données multi-régions
2. Implémenter row access policies
3. Tester avec différents utilisateurs
4. Vérifier filtrage automatique

Exercice 3 : Audit et Monitoring

1. Activer data access logs
2. Exporter logs vers BigQuery

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 6

BigQuery ML - Fondamentaux

Objectifs du Module 6

- Comprendre BigQuery ML (BQML)
- Découvrir les types de modèles disponibles
- Créer son premier modèle ML
- Entraîner et évaluer des modèles
- Faire des prédictions
- Comprendre le workflow ML

Introduction à BigQuery ML

Qu'est-ce que BigQuery ML ?

Capacité de créer et exécuter des modèles de Machine Learning directement dans BigQuery en utilisant SQL.

Avantages

- Pas besoin d'exporter données
- Utilise SQL standard
- Pas de gestion d'infrastructure ML
- Intégration native avec données BigQuery
- Démocratisation du ML pour analystes SQL

Workflow Simplifié

```
-- Tout en SQL !
CREATE MODEL `dataset.my_model`
OPTIONS(model_type='linear_reg') AS
SELECT feature1, feature2, label
FROM training_data;

-- Prédire
SELECT * FROM ML.PREDICT(MODEL `dataset.my_model`,
  (SELECT feature1, feature2 FROM new_data));
```

Types de Modèles Supportés

Supervised Learning

Régression

- Linear Regression : Prédire valeur continue
- Boosted Tree Regressor : XGBoost pour régression

Classification

- Logistic Regression : Classification binaire/multi-classe
- Boosted Tree Classifier : XGBoost pour classification
- DNN Classifier : Deep Neural Network

Unsupervised Learning

- K-Means : Clustering
- PCA : Réduction dimensionnalité

Autres

- Time Series (ARIMA+) : Prévisions temporelles
- Matrix Factorization : Systèmes de recommandation
- AutoML Tables : Automatique

Importation

- TensorFlow models

Workflow BigQuery ML

```
1. Préparer Données
   ↓
2. CREATE MODEL (Entraînement)
   ↓
3. ML.EVALUATE (Évaluation)
   ↓
4. ML.PREDICT (Prédiction)
   ↓
5. Itération et Amélioration
   ↓
6. Déploiement (Scheduled Queries, etc.)
```

Itératif

Améliorer progressivement en ajustant features et hyperparamètres

Premier Modèle : Régression Linéaire

Problème

Prédire prix d'une maison basé sur caractéristiques

Données

```
CREATE OR REPLACE TABLE `dataset.housing_data` AS
SELECT
  1000 as sqft, 3 as bedrooms, 2 as bathrooms, 10 as age_years, 250000 as price
UNION ALL SELECT 1500, 4, 3, 5, 350000
UNION ALL SELECT 800, 2, 1, 20, 180000
-- ... plus de données
```

Créer Modèle

```
CREATE OR REPLACE MODEL `dataset.housing_price_model`
OPTIONS(
  model_type='linear_reg',
  input_label_cols=['price']
) AS
SELECT
  sqft,
  bedrooms,
  bathrooms,
  age_years,
  price
FROM `dataset.housing_data`;
```

Évaluer le Modèle

ML.EVALUATE

```
SELECT *
FROM ML.EVALUATE(MODEL `dataset.housing_price_model`);
```

Métriques Régression

- mean_absolute_error : Erreur moyenne absolue
- mean_squared_error : Erreur quadratique moyenne
- mean_squared_log_error : Pour valeurs positives
- median_absolute_error : Robuste aux outliers
- r2_score : Coefficient de détermination (0-1, 1 = parfait)

Interprétation

```
r2_score: 0.85 → Le modèle explique 85% de la variance
mean_absolute_error: 15000 → Erreur moyenne de 15K USD
```

Faire des Prédictions

ML.PREDICT

```
SELECT
  *,
  predicted_price
FROM ML.PREDICT(MODEL `dataset.housing_price_model`, (
  SELECT
    1200 as sqft,
    3 as bedrooms,
    2 as bathrooms,
    8 as age_years
));
```

Prédiction sur Table

```
CREATE OR REPLACE TABLE `dataset.predictions` AS
SELECT
  house_id,
  sqft,
  bedrooms,
  bathrooms,
  predicted_price
FROM ML.PREDICT(MODEL `dataset.housing_price_model`, (
  SELECT * FROM `dataset.new_houses`
));
```

Classification Binaire

Problème

Prédire si un client va acheter (churn prediction, etc.)

Modèle Logistique

```
CREATE OR REPLACE MODEL `dataset.customer_purchase_model`
OPTIONS(
  model_type='logistic_reg',
  input_label_cols=['will_purchase']
) AS
SELECT
  age,
  total_visits,
  avg_session_duration,
  pages_viewed,
  will_purchase
FROM `dataset.customer_behavior`;
```

will_purchase

Colonne booléenne (TRUE/FALSE ou 0/1)

Évaluer Classification Binaire

Métriques

```
SELECT * FROM ML.EVALUATE(MODEL `dataset.customer_purchase_model`);
```

Métriques Retournées

- accuracy : Taux de prédictions correctes
- precision : Vrais positifs / (Vrais positifs + Faux positifs)
- recall : Vrais positifs / (Vrais positifs + Faux négatifs)
- f1_score : Moyenne harmonique precision/recall
- log_loss : Fonction de perte
- roc_auc : Area Under ROC Curve (0.5-1, 1 = parfait)

Matrice de Confusion

```
SELECT *
FROM ML.CONFUSION_MATRIX(MODEL `dataset.customer_purchase_model`);
```

Seuil de Classification

Probabilités de Prédiction

```
SELECT
  customer_id,
  predicted_will_purchase,
  predicted_will_purchase_probs
FROM ML.PREDICT(MODEL `dataset.customer_purchase_model`, (
  SELECT * FROM `dataset.new_customers`
));
```

Ajuster Seuil

```
-- Seuil personnalisé (défaut 0.5)
SELECT
  customer_id,
  CASE
    WHEN predicted_will_purchase_probs[OFFSET(1)].prob > 0.7
    THEN TRUE
    ELSE FALSE
  END as high_confidence_purchase
FROM ML.PREDICT(...);
```

Use Case

Precision élevée pour campagnes coûteuses, recall élevé pour ne pas manquer prospects

Classification Multi-Classe

Problème

Catégoriser produits en plusieurs catégories

Modèle

```
CREATE OR REPLACE MODEL `dataset.product_category_model`
OPTIONS(
  model_type='logistic_reg',
  input_label_cols=['category']
) AS
SELECT
  title,
  description,
  price,
  category  -- 'Electronics', 'Clothing', 'Books', etc.
FROM `dataset.products`;
```

Prédiction

```
SELECT
  product_id,
  title,
  predicted_category,
  predicted_category_probs
FROM ML.PREDICT(MODEL `dataset.product_category_model`, (
  SELECT * FROM `dataset.new_products`
));
```

Feature Engineering dans BQML

Transformations SQL

```
CREATE OR REPLACE MODEL `dataset.advanced_model`
OPTIONS(model_type='linear_reg', input_label_cols=['price']) AS
SELECT
  sqft,
  bedrooms,
  bathrooms,
  -- Feature engineering
  sqft * bedrooms as sqft_bedrooms_interaction,
  CASE
    WHEN age_years < 5 THEN 'new'
    WHEN age_years < 20 THEN 'moderate'
    ELSE 'old'
  END as age_category,
  LOG(sqft) as log_sqft,  -- Transformation log
  POW(sqft, 2) as sqft_squared,
  price
FROM `dataset.housing_data`;
```

Categorical Features

BQML gère automatiquement one-hot encoding

TRANSFORM Clause

Appliquer Transformations Automatiquement

```
CREATE OR REPLACE MODEL `dataset.model_with_transform`
TRANSFORM(
  * EXCEPT(raw_text),
  ML.BUCKETIZE(sqft, [500, 1000, 1500, 2000]) as sqft_bucket,
  ML.QUANTILE_BUCKETIZE(price, 10) as price_decile,
  ML.FEATURE_CROSS(STRUCT(bedrooms, bathrooms)) as bed_bath_cross
)
OPTIONS(
  model_type='linear_reg',
  input_label_cols=['label']
) AS
SELECT * FROM training_data;
```

Avantage

Transformations appliquées automatiquement lors de ML.PREDICT

Fonctions TRANSFORM

- ML.BUCKETIZE : Discrétisation
- ML.QUANTILE_BUCKETIZE : Buckets par quantiles
- ML.FEATURE_CROSS : Interaction de features
- ML.POLYNOMIAL_EXPAND : Features polynomiales

Hyperparamètres

Ajuster l'Entraînement

```
CREATE OR REPLACE MODEL `dataset.tuned_model`
OPTIONS(
  model_type='logistic_reg',
  input_label_cols=['label'],
  l1_reg=0.1,               -- Régularisation L1 (Lasso)
  l2_reg=0.1,               -- Régularisation L2 (Ridge)
  max_iterations=50,        -- Nombre max d'itérations
  learn_rate=0.1,           -- Taux d'apprentissage
  early_stop=TRUE,          -- Arrêt anticipé
  min_rel_progress=0.01     -- Seuil amélioration
) AS
SELECT * FROM training_data;
```

Hyperparamètres Clés

- l1_reg, l2_reg : Éviter overfitting
- max_iterations : Durée entraînement
- learn_rate : Vitesse convergence
- early_stop : Économiser ressources

Train/Test Split

DATA_SPLIT_METHOD

```
CREATE OR REPLACE MODEL `dataset.model_with_split`
OPTIONS(
  model_type='logistic_reg',
  input_label_cols=['label'],
  data_split_method='random',  -- 'random', 'seq', 'no_split', 'custom'
  data_split_eval_fraction=0.2 -- 20% pour évaluation
) AS
SELECT * FROM training_data;
```

Méthodes Split

- random : Split aléatoire (défaut)
- seq : Séquentiel (pour time series)
- no_split : Tout pour entraînement
- custom : Colonne custom_split_column

Custom Split

```
OPTIONS(
  data_split_method='custom',
  data_split_col='split_col'
)
-- split_col : 'TRAIN', 'EVAL', 'TEST'
```

Clustering avec K-Means

Use Case

Segmentation de clients

Modèle

```
CREATE OR REPLACE MODEL `dataset.customer_segments`
OPTIONS(
  model_type='kmeans',
  num_clusters=5,           -- Nombre de clusters
  standardize_features=TRUE -- Normalisation
) AS
SELECT
  total_purchases,
  avg_order_value,
  days_since_last_purchase,
  total_sessions
FROM `dataset.customers`;
```

Prédiction (Assignment)

```
SELECT
  customer_id,
  CENTROID_ID as cluster
FROM ML.PREDICT(MODEL `dataset.customer_segments`, (
  SELECT * FROM `dataset.customers`
));
```

Évaluer Clustering

Davies-Bouldin Index

```
SELECT * FROM ML.EVALUATE(MODEL `dataset.customer_segments`);
```

Retourne davies_bouldin_index : Plus bas = meilleur clustering

Centroids

```
SELECT * FROM ML.CENTROIDS(MODEL `dataset.customer_segments`);
```

Caractéristiques moyennes de chaque cluster

Choisir K

Méthode Elbow : Tester différentes valeurs de k, tracer davies_bouldin_index

```
-- Créer modèles avec k=3,4,5,6,7
-- Comparer scores
-- Choisir k optimal
```

Time Series Forecasting

ARIMA_PLUS

```
CREATE OR REPLACE MODEL `dataset.sales_forecast`
OPTIONS(
  model_type='ARIMA_PLUS',
  time_series_timestamp_col='date',
  time_series_data_col='sales',
  auto_arima=TRUE,
  data_frequency='DAILY'
) AS
SELECT
  date,
  sales
FROM `dataset.daily_sales`;
```

Prédire le Futur

```
SELECT *
FROM ML.FORECAST(MODEL `dataset.sales_forecast`,
  STRUCT(30 AS horizon, 0.95 AS confidence_level));
```

Retourne : forecast_timestamp, forecast_value, prediction_interval_lower/upper

Recommandation avec Matrix Factorization

Use Case

Recommander produits aux utilisateurs

Modèle

```
CREATE OR REPLACE MODEL `dataset.product_recommendations`
OPTIONS(
  model_type='matrix_factorization',
  user_col='user_id',
  item_col='product_id',
  rating_col='rating',
  feedback_type='implicit'  -- ou 'explicit'
) AS
SELECT
  user_id,
  product_id,
  rating
FROM `dataset.user_product_interactions`;
```

Recommandations

```
SELECT *
FROM ML.RECOMMEND(MODEL `dataset.product_recommendations`,
  (SELECT 12345 as user_id));
```

Atelier Pratique : Premier Modèle ML

Exercice 1 : Régression

1. Utiliser dataset public (ex: housing)
2. Créer modèle de régression linéaire
3. Évaluer performance
4. Faire prédictions sur nouvelles données
5. Améliorer avec feature engineering

Exercice 2 : Classification

1. Dataset de classification binaire
2. Créer modèle logistique
3. Analyser métriques et confusion matrix
4. Ajuster seuil de prédiction
5. Optimiser avec hyperparamètres

Exercice 3 : Clustering

1. Dataset clients

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 7

Modèles ML Avancés

Objectifs du Module 7

- Utiliser Boosted Trees (XGBoost)
- Implémenter Deep Neural Networks
- Utiliser AutoML Tables
- Optimiser les hyperparamètres
- Gérer le déséquilibre des classes
- Interpréter les modèles

Boosted Tree Regressor

XGBoost dans BigQuery

Algorithme d'ensemble utilisant gradient boosting

Avantages

- Performance supérieure sur données tabulaires
- Gère relations non-linéaires
- Feature importance intégré
- Robuste aux outliers

Création

```
CREATE OR REPLACE MODEL `dataset.house_price_boosted`
OPTIONS(
  model_type='boosted_tree_regressor',
  input_label_cols=['price'],
  num_parallel_tree=100,
  max_iterations=50,
  tree_method='hist',
  min_tree_child_weight=2,
  colsample_bytree=0.8,
  subsample=0.8
) AS
SELECT * FROM `dataset.housing_data`;
```

Hyperparamètres Boosted Trees

Contrôle de Complexité

- num_parallel_tree : Nombre d'arbres (défaut 100)
- max_tree_depth : Profondeur max arbre (défaut 6)
- min_tree_child_weight : Min poids noeud enfant (régularisation)
- subsample : Fraction données pour chaque arbre (0.8 = 80%)
- colsample_bytree : Fraction features par arbre

Apprentissage

- max_iterations : Rounds de boosting
- early_stop : Arrêt si pas amélioration
- learn_rate : Shrinkage (0.1-0.3 recommandé)

Exemple Optimisé

```
OPTIONS(
  model_type='boosted_tree_regressor',
  num_parallel_tree=200,
  max_tree_depth=6,
  learn_rate=0.1,
  subsample=0.8,
  early_stop=TRUE
)
```

Feature Importance

ML.FEATURE_IMPORTANCE

```
SELECT
  feature,
  importance,
  RANK() OVER (ORDER BY importance DESC) as rank
FROM ML.FEATURE_IMPORTANCE(MODEL `dataset.house_price_boosted`)
ORDER BY importance DESC
LIMIT 10;
```

Résultat

```
feature          | importance | rank
sqft             | 0.45       | 1
location_score   | 0.23       | 2
age_years        | 0.15       | 3
bedrooms         | 0.10       | 4
...
```

Utilisation

- Feature selection
- Insights métier
- Réduire dimensions

Boosted Tree Classifier

Classification avec XGBoost

```
CREATE OR REPLACE MODEL `dataset.churn_prediction_boosted`
OPTIONS(
  model_type='boosted_tree_classifier',
  input_label_cols=['churned'],
  num_parallel_tree=150,
  max_tree_depth=8,
  learn_rate=0.1,
  subsample=0.9,
  colsample_bytree=0.9,
  tree_method='hist'
) AS
SELECT
  customer_tenure_months,
  monthly_charges,
  total_charges,
  contract_type,
  payment_method,
  num_support_tickets,
  churned
FROM `dataset.customer_data`;
```

Évaluation

Même métriques que logistic_reg (accuracy, precision, recall, roc_auc)

Deep Neural Networks (DNN)

DNN Classifier

```
CREATE OR REPLACE MODEL `dataset.image_classifier`
OPTIONS(
  model_type='dnn_classifier',
  input_label_cols=['category'],
  hidden_units=[256, 128, 64],  -- Architecture du réseau
  activation_fn='relu',
  dropout=0.2,
  batch_size=64,
  max_iterations=100,
  learn_rate=0.001,
  optimizer='adam'
) AS
SELECT
  feature_1,
  feature_2,
  -- ... features numériques
  category
FROM `dataset.training_data`;
```

hidden_units

Liste définissant l'architecture : [256, 128, 64] = 3 couches cachées

Hyperparamètres DNN

Architecture

- hidden_units : [256, 128, 64, 32] - Couches et neurones
- activation_fn : 'relu', 'tanh', 'sigmoid'

Régularisation

- dropout : Probabilité de dropout (0.1-0.5)
- l1_reg, l2_reg : Régularisation

Optimisation

- optimizer : 'adam', 'adagrad', 'sgd', 'ftrl'
- learn_rate : Taux d'apprentissage (0.001 typique)
- batch_size : Taille mini-batch (16, 32, 64, 128)

Best Practice

Commencer avec réseau simple, ajouter complexité si nécessaire

DNN Regressor

Régression avec DNN

```
CREATE OR REPLACE MODEL `dataset.price_prediction_dnn`
OPTIONS(
  model_type='dnn_regressor',
  input_label_cols=['price'],
  hidden_units=[128, 64, 32],
  activation_fn='relu',
  dropout=0.1,
  batch_size=32,
  max_iterations=50,
  learn_rate=0.001,
  optimizer='adam',
  early_stop=TRUE
) AS
SELECT
  * EXCEPT(id, timestamp)
FROM `dataset.training_data`;
```

Quand Utiliser DNN vs Boosted Trees

- DNN : Données haute dimension, images, texte
- Boosted Trees : Données tabulaires classiques (souvent meilleur)

AutoML Tables

Machine Learning Automatique

BigQuery ML peut utiliser AutoML Tables pour exploration automatique d'architectures

Création

```
CREATE OR REPLACE MODEL `dataset.automl_model`
OPTIONS(
  model_type='automl_classifier',
  input_label_cols=['label'],
  budget_hours=1.0  -- Budget temps entraînement
) AS
SELECT * FROM `dataset.training_data`;
```

Processus AutoML

- Feature engineering automatique
- Sélection d'architecture
- Hyperparameter tuning
- Ensemble de modèles

Trade-off

Plus long et coûteux, mais souvent meilleure performance

Gestion du Déséquilibre de Classes

Problème

Dataset avec 95% classe A, 5% classe B

Solution 1 : Class Weights

```
CREATE OR REPLACE MODEL `dataset.balanced_model`
OPTIONS(
  model_type='logistic_reg',
  input_label_cols=['label'],
  auto_class_weights=TRUE  -- Weights automatiques
) AS
SELECT * FROM imbalanced_data;
```

Solution 2 : Upsampling/Downsampling

```
-- Équilibrer manuellement
WITH balanced AS (
  SELECT * FROM training_data WHERE label = 'minority'
  UNION ALL (
    SELECT * FROM training_data WHERE label = 'majority'
    ORDER BY RAND() LIMIT (SELECT COUNT(*) FROM training_data WHERE label = 'minority')
  )
)
SELECT * FROM balanced;
```

Optimisation d'Hyperparamètres

Grid Search Manuel

```sql
-- Créer plusieurs modèles avec différents hyperparamètres
CREATE OR REPLACE MODEL `dataset.model_lr_01`
OPTIONS(model_type='logistic_reg', learn_rate=0.1) AS ...;

CREATE OR REPLACE MODEL `dataset.model_lr_001`
OPTIONS(model_type='logistic_reg', learn_rate=0.01) AS ...;

CREATE OR REPLACE MODEL `dataset.model_lr_0001`
OPTIONS(model_type='logistic_reg', learn_rate=0.001) AS ...;

-- Comparer performances
SELECT
  'model_lr_01' as model,
  * FROM ML.EVALUATE(MODEL `dataset.model_lr_01`)
UNION ALL
SELECT
  'model_lr_001',
  * FROM ML.EVALUATE(MODEL `dataset.model_lr_001`)
UNION ALL
SELECT
  'model_lr_0001',
  * FROM ML.EVALUATE(MODEL `dataset.model_lr_0001`);
```

Hparam Tuning avec AutoML

Automatique avec AutoML

```
CREATE OR REPLACE MODEL `dataset.tuned_model`
OPTIONS(
  model_type='automl_classifier',
  budget_hours=2.0,
  optimization_objective='MAXIMIZE_AU_ROC'
) AS
SELECT * FROM training_data;
```

AutoML fait

- Test de différents algorithmes
- Hyperparameter search
- Architecture search (pour DNN)
- Feature engineering

Accès Détails

```
SELECT * FROM ML.TRAINING_INFO(MODEL `dataset.tuned_model`);
```

Model Explainability

ML.EXPLAIN_PREDICT

Explications au niveau prédiction individuelle

```
SELECT *
FROM ML.EXPLAIN_PREDICT(
  MODEL `dataset.house_price_model`,
  (SELECT * FROM `dataset.new_houses` WHERE house_id = 123),
  STRUCT(3 as top_k_features)
);
```

Résultat

- Prédiction
- Top K features contribuant
- Attributions (SHAP values-like)

Use Cases

- Debugging prédictions
- Conformité réglementaire (expliquer décisions)
- Trust et adoption

Global Explainability

ML.GLOBAL_EXPLAIN

Explications au niveau modèle global

```
SELECT *
FROM ML.GLOBAL_EXPLAIN(
  MODEL `dataset.house_price_model`
);
```

Information Retournée

- Features les plus importantes globalement
- Direction de l'effet (positif/négatif)
- Magnitude

Comparaison

- ML.FEATURE_IMPORTANCE : Importance pure
- ML.GLOBAL_EXPLAIN : Importance + direction effet

Model Registry et Versioning

Versioning Modèles

BigQuery ML maintient historique des modèles

```
-- Liste versions
SELECT * FROM `dataset`.INFORMATION_SCHEMA.MODELS
WHERE model_name = 'my_model';

-- Recréer modèle = nouvelle version automatique
CREATE OR REPLACE MODEL `dataset.my_model`
OPTIONS(...) AS ...;

-- Référencer version spécifique
SELECT * FROM ML.PREDICT(
  MODEL `dataset.my_model`@1234567890,  -- version_id
  (...)
);
```

Best Practice

- Tagger versions importantes
- Documenter changements
- Tests A/B sur versions

Export de Modèles

Exporter vers GCS

```
-- TensorFlow SavedModel format
EXPORT MODEL `dataset.my_model`
OPTIONS(URI='gs://my-bucket/models/my_model');
```

Use Cases

- Déploiement hors BigQuery
- Vertex AI Endpoints
- TensorFlow Serving
- Mobile/Edge avec TensorFlow Lite

Import de Modèles

```
-- Importer TensorFlow model
CREATE OR REPLACE MODEL `dataset.imported_model`
OPTIONS(MODEL_TYPE='TENSORFLOW', MODEL_PATH='gs://bucket/model/*');
```

Scheduled Retraining

Automatiser Ré-entraînement
Scheduled Queries

1. Créer requête de ré-entraînement

```
CREATE OR REPLACE MODEL `dataset.model`
OPTIONS(...) AS
SELECT * FROM `dataset.training_data`
WHERE date >= DATE_SUB(CURRENT_DATE(), INTERVAL 90 DAY);
```

2. Scheduler dans console BigQuery

- Frequency : Daily, Weekly, Monthly
- Notification si échec

Use Case
Modèles sur données changeantes (drift detection)

Drift Detection

Détecter Dégradation Performance

Monitoring Métriques

```
-- Évaluer sur données récentes
WITH recent_eval AS (
  SELECT *
  FROM ML.EVALUATE(MODEL `dataset.production_model`, (
    SELECT * FROM `dataset.labeled_data`
    WHERE date >= DATE_SUB(CURRENT_DATE(), INTERVAL 7 DAY)
  ))
)
SELECT
  CURRENT_DATE() as eval_date,
  *
FROM recent_eval;
```

Alertes

Si accuracy < seuil, trigger ré-entraînement

Data Drift

Comparer distributions features training vs production

Atelier Pratique : Modèles Avancés

Exercice 1 : Boosted Trees

1. Créer modèle XGBoost
2. Tuner hyperparamètres
3. Analyser feature importance
4. Comparer avec linear/logistic

Exercice 2 : Deep Neural Network

1. Créer DNN classifier/regressor
2. Expérimenter architectures (hidden_units)
3. Optimiser hyperparamètres
4. Évaluer sur test set

Exercice 3 : AutoML

1. Utiliser AutoML Tables
2. Comparer avec modèles manuels

Conclusion Module 7

Récapitulatif

Nous avons exploré les modèles ML avancés dans BigQuery :

- XGBoost (Boosted Trees) pour performance
- Deep Neural Networks pour données complexes
- AutoML pour automatisation
- Techniques d'optimisation
- Explainability et interprétabilité
- Gestion du cycle de vie des modèles

Prochaine Étape

Intégration avec TensorFlow et MLOps

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 8

Intégration TensorFlow

Objectifs du Module 8

- Importer des modèles TensorFlow dans BigQuery
- Utiliser TensorFlow pour preprocessing avancé
- Créer des modèles personnalisés
- Déployer des modèles TensorFlow
- Intégrer avec Vertex AI
- Utiliser pre-trained models

Importer Modèle TensorFlow

Prérequis

Modèle TensorFlow au format SavedModel dans GCS

Import dans BigQuery

```
CREATE OR REPLACE MODEL `dataset.tf_imported_model`
OPTIONS(
  model_type='TENSORFLOW',
  model_path='gs://my-bucket/tf_model/*'
);
```

Utilisation

```
SELECT *
FROM ML.PREDICT(MODEL `dataset.tf_imported_model`, (
  SELECT feature1, feature2, feature3
  FROM input_data
));
```

Limitations

- Input/Output doivent être compatibles BigQuery types
- Certaines ops TensorFlow non supportées

Créer Modèle TensorFlow Custom

Python + TensorFlow

```
import tensorflow as tf
from google.cloud import bigquery

# Définir modèle
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(10,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Entraîner (depuis BigQuery ou ailleurs)
# ...

# Sauvegarder
model.save('gs://my-bucket/custom_model')
```

Importer dans BigQuery

Pre-trained Models

TensorFlow Hub

Utiliser modèles pré-entraînés pour transfer learning

Exemple : Text Embeddings

```
import tensorflow_hub as hub

# Charger embedding model
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")

# Créer modèle avec embeddings
inputs = tf.keras.layers.Input(shape=(), dtype=tf.string)
embeddings = hub.KerasLayer(embed)(inputs)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(embeddings)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(...)
```

Use Cases

- NLP : Embeddings, classification texte
- Vision : Transfer learning sur images
- Recommendation : Embeddings produits

Remote Models avec Vertex AI

CREATE MODEL avec Endpoint Vertex AI

```
CREATE OR REPLACE MODEL `dataset.vertex_remote_model`
REMOTE WITH CONNECTION `project.region.connection_id`
OPTIONS(
  ENDPOINT='projects/PROJECT_ID/locations/REGION/endpoints/ENDPOINT_ID'
);
```

Connexion Cloud Resource

```
bq mk --connection \
  --location=US \
  --project_id=PROJECT_ID \
  --connection_type=CLOUD_RESOURCE \
  connection_id
```

Prédiction

```
SELECT *
FROM ML.PREDICT(MODEL `dataset.vertex_remote_model`, (
  SELECT * FROM input_table
));
```

Vertex AI Custom Training

Entraîner Modèle Complexe sur Vertex AI
Pour modèles trop complexes pour BQML
Workflow

1. Exporter données BigQuery vers GCS
2. Entraîner sur Vertex AI Custom Training
3. Déployer endpoint Vertex AI
4. Référencer dans BigQuery comme remote model

Avantages

- GPU/TPU pour entraînement
- Frameworks libres (PyTorch, etc.)
- Contrôle total
- Scale compute indépendamment

Inconvénient
Complexité accrue

Feature Store

Vertex AI Feature Store
Gestion centralisée des features
Intégration BigQuery

1. Créer Feature Store
2. Ingest features depuis BigQuery

```
gcloud ai feature-stores entities batch-serve \
  --feature-store=STORE \
  --entity-type=TYPE \
  --read-instances-uri=bq://project.dataset.table
```

3. Utiliser features pour ML

```
from google.cloud.aiplatform import FeatureStore

fs = FeatureStore('store_id')
features = fs.read_features(['feature1', 'feature2'], entity_ids)
```

Avantages

- Features réutilisables

Pipelines ML avec Kubeflow

Orchestration ML

Kubeflow Pipelines sur Vertex AI

```
from kfp.v2 import dsl
from kfp.v2.dsl import component

@component
def bigquery_train():
    from google.cloud import bigquery
    client = bigquery.Client()
    query = """
    CREATE OR REPLACE MODEL `dataset.model`
    OPTIONS(...) AS SELECT * FROM training_data
    """
    client.query(query).result()

@dsl.pipeline(name='bq-ml-pipeline')
def pipeline():
    train_task = bigquery_train()
    # ... autres étapes

# Compiler et exécuter
```

Use Case

MLOps : entraînement, évaluation, déploiement automatisés

BigQuery + Vertex AI Workbench

Notebooks Managés
Intégration

```
# Dans Vertex AI Workbench notebook
from google.cloud import bigquery
import tensorflow as tf

# Query BigQuery
client = bigquery.Client()
query = """
SELECT * FROM `project.dataset.training_data`
LIMIT 10000
"""
df = client.query(query).to_dataframe()

# Entraîner TensorFlow
model = tf.keras.Sequential([...])
model.fit(df[features], df[label])

# Sauvegarder
model.save('gs://bucket/model')
```

Avantages

- Environnement intégré
- BigQuery magics

Streaming Predictions

Real-time ML Inference Architecture

```
Pub/Sub → Dataflow → ML.PREDICT → Destination
```

Dataflow Pipeline

```
import apache_beam as beam
from apache_beam.ml.inference.base import RunInference
from apache_beam.ml.inference.tensorflow_inference import TFModelHandlerTensor

pipeline = beam.Pipeline()

model_handler = TFModelHandlerTensor(model_uri='gs://bucket/model')

predictions = (
    pipeline
    | 'ReadPubSub' >> beam.io.ReadFromPubSub(subscription=SUB)
    | 'Preprocess' >> beam.Map(preprocess)
    | 'Predict' >> RunInference(model_handler)
    | 'Postprocess' >> beam.Map(postprocess)
    | 'WriteBQ' >> beam.io.WriteToBigQuery(table)
)
```

Batch Predictions avec Dataflow

Pour Volumes Importants

Lire depuis BigQuery, Prédire, Écrire

```
with beam.Pipeline() as p:
    predictions = (
        p
        | 'ReadBQ' >> beam.io.ReadFromBigQuery(query=QUERY, use_standard_sql=True)
        | 'ExtractFeatures' >> beam.Map(extract_features)
        | 'Predict' >> RunInference(model_handler)
        | 'Format' >> beam.Map(format_output)
        | 'WriteBQ' >> beam.io.WriteToBigQuery(
            'project:dataset.predictions',
            schema=SCHEMA,
            write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND
        )
    )
```

Avantage

Scale automatique, plus efficace que ML.PREDICT pour très gros volumes

Model Monitoring

Vertex AI Model Monitoring

Pour modèles déployés sur Vertex AI endpoints

Configuration

```
from google.cloud import aiplatform

aiplatform.init(project=PROJECT_ID)

# Créer monitoring job
monitoring_job = aiplatform.ModelDeploymentMonitoringJob.create(
    display_name='model-monitoring',
    endpoint=endpoint,
    logging_sampling_strategy=sampling_strategy,
    objective_configs=[skew_config, drift_config],
    schedule_config=schedule_config
)
```

Métriques

- Training-serving skew
- Prediction drift
- Feature attribution drift

CI/CD pour ML

MLOps avec Cloud Build

.cloudbuild.yaml

```yaml
steps:
  # Test données
  - name: 'gcr.io/cloud-builders/gcloud'
    args: ['bq', 'query', '--use_legacy_sql=false', '...']

  # Entraîner modèle
  - name: 'gcr.io/cloud-builders/gcloud'
    args: ['bq', 'query', '--use_legacy_sql=false',
           'CREATE OR REPLACE MODEL ...']

  # Évaluer
  - name: 'python'
    entrypoint: 'python'
    args: ['evaluate_model.py']

  # Déployer si métriques OK
  - name: 'gcr.io/cloud-builders/gcloud'
    args: ['bq', 'cp', 'dataset.model', 'dataset.model_prod']
```

Trigger

Sur commit Git, exécution automatique

Atelier Pratique : TensorFlow Integration

Exercice 1 : Import TensorFlow Model

1. Créer simple modèle TensorFlow (Keras)
2. Sauvegarder en SavedModel format
3. Uploader vers GCS
4. Importer dans BigQuery
5. Faire prédictions

Exercice 2 : Custom Model

1. Entraîner modèle TF avec embeddings
2. Exporter et importer
3. Utiliser pour classification texte
4. Évaluer performance

Exercice 3 : Remote Model

1. Déployer modèle sur Vertex AI

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

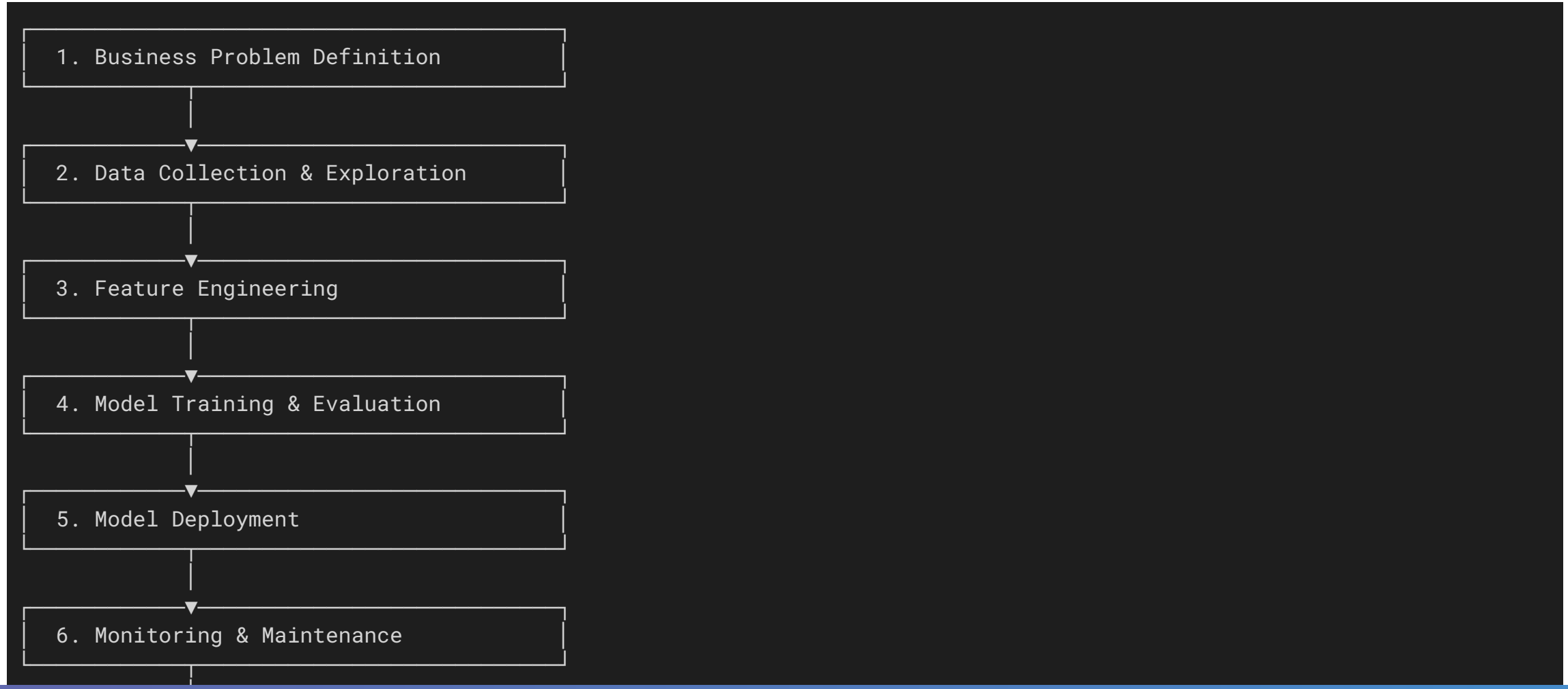
Module 9

MLOps et Production

Objectifs du Module 9

- Mettre en production des modèles ML
- Automatiser le workflow ML
- Monitoring et maintenance
- A/B testing de modèles
- Gestion des versions
- Best practices MLOps

ML Lifecycle



Déploiement de Modèles

Stratégies

Shadow Mode

Modèle s'exécute en parallèle, logs prédictions mais pas utilisé

Canary Deployment

5% trafic → nouveau modèle, 95% → ancien modèle

Blue-Green

Switch instantané entre modèles

A/B Testing

Split utilisateurs, mesurer métriques métier

Implementation BigQuery

```
-- Modèle production actuel
`dataset.model_prod_v1`

-- Nouveau modèle
`dataset.model_prod_v2`

-- Routing logic dans application
IF random() < 0.05 THEN
  ML.PREDICT(MODEL `dataset.model_prod_v2`, ...)
ELSE
  ML.PREDICT(MODEL `dataset.model_prod_v1`, ...)
```

Monitoring en Production

Métriques à Suivre

Performance Modèle

```
-- Évaluation continue
CREATE OR REPLACE TABLE `dataset.model_metrics_daily` AS
SELECT
  CURRENT_DATE() as date,
  'model_v2' as model_version,
  *
FROM ML.EVALUATE(MODEL `dataset.model_prod_v2`, (
  SELECT * FROM `dataset.labeled_data_recent`
));
```

Prédiction Distribution

```
-- Distribution prédictions
SELECT
  DATE(prediction_timestamp) as date,
  APPROX_QUANTILES(predicted_value, 100) as percentiles,
  AVG(predicted_value) as avg_prediction
FROM `dataset.predictions`
GROUP BY date;
```

Alertes

Cloud Monitoring + BigQuery pour alertes sur drift

Retraining Strategy

Quand Ré-entraîner ?

Triggers

- Scheduled : Hebdomadaire, mensuel
- Performance degradation : Métriques < seuil
- Data drift détecté
- Nouvelles données disponibles

Implementation

```
-- Scheduled Query (daily)
CREATE OR REPLACE MODEL `dataset.model_prod_latest`
OPTIONS(model_type='boosted_tree_classifier', ...) AS
SELECT * FROM (
  SELECT * FROM `dataset.training_data`
  WHERE date >= DATE_SUB(CURRENT_DATE(), INTERVAL 90 DAY)
)
WHERE RAND() < 0.8;  -- 80% training

-- Évaluation
INSERT INTO `dataset.model_metrics`
SELECT CURRENT_TIMESTAMP(), *
FROM ML.EVALUATE(MODEL `dataset.model_prod_latest`, (
  SELECT * FROM `dataset.training_data`
  WHERE date >= DATE_SUB(CURRENT_DATE(), INTERVAL 90 DAY)
```

A/B Testing Framework

Comparer Modèles en Production

Setup

```sql
-- Table assignment
CREATE OR REPLACE TABLE `dataset.ab_test_assignments` AS
SELECT
  user_id,
  IF(MOD(FARM_FINGERPRINT(CAST(user_id AS STRING)), 100) < 50,
     'model_a', 'model_b') as variant
FROM `dataset.users`;

-- Logging prédictions avec variant
INSERT INTO `dataset.prediction_log`
SELECT
  CURRENT_TIMESTAMP() as timestamp,
  user_id,
  a.variant,
  predicted_value,
  actual_value
FROM ML.PREDICT(...) p
JOIN `dataset.ab_test_assignments` a USING(user_id);
```

A/B Testing Framework

Analyse

```sql
SELECT
  variant,
  COUNT(*) as predictions,
  AVG(ABS(predicted_value - actual_value)) as mae
FROM `dataset.prediction_log`
WHERE DATE(timestamp) >= DATE_SUB(CURRENT_DATE(), INTERVAL 7 DAY)
GROUP BY variant;
```

Model Documentation

Model Cards

Documenter modèles pour gouvernance et reproductibilité

Informations Clés

- Use case et objectif
- Features utilisées
- Algorithme et hyperparamètres
- Métriques performance
- Limitations connues
- Considérations éthiques
- Contact et ownership

Stockage

```
-- Table metadata
CREATE OR REPLACE TABLE `dataset.model_registry` (
  model_name STRING,
  version STRING,
  created_date TIMESTAMP,
  training_data_query STRING,
  features ARRAY<STRING>,
  metrics STRUCT<accuracy FLOAT64, auc FLOAT64>,
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Module 10

Cas d'Usage et Best Practices

Objectifs du Module 10

- Explorer des cas d'usage réels
- Patterns courants
- Best practices architecture
- Optimisation coûts
- Troubleshooting
- Resources et formations continues

Cas d'Usage : Churn Prediction

Problème Business

Prédire quels clients vont partir

Solution BigQuery ML

```sql
-- 1. Feature engineering
CREATE OR REPLACE TABLE `dataset.customer_features` AS
SELECT
  customer_id,
  DATE_DIFF(CURRENT_DATE(), first_purchase_date, DAY) as tenure_days,
  COUNT(DISTINCT order_id) as total_orders,
  SUM(order_amount) as total_revenue,
  AVG(order_amount) as avg_order_value,
  DATE_DIFF(CURRENT_DATE(), MAX(order_date), DAY) as days_since_last_order,
  COUNT(DISTINCT CASE WHEN order_date >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
    THEN order_id END) as orders_last_30d,
  churned  -- Label
FROM `dataset.orders`
JOIN `dataset.customers` USING(customer_id)
GROUP BY customer_id, first_purchase_date, churned;

-- 2. Modèle
CREATE OR REPLACE MODEL `dataset.churn_model`
OPTIONS(model_type='boosted_tree_classifier',
        input_label_cols=['churned']) AS
SELECT * EXCEPT(customer_id) FROM `dataset.customer_features`;

-- 3. Prédictions actionnables
```

Cas d'Usage : Recommandation Produits

Matrix Factorization

```
CREATE OR REPLACE MODEL `dataset.product_recs`
OPTIONS(
  model_type='matrix_factorization',
  user_col='user_id',
  item_col='product_id',
  rating_col='implicit_rating'
) AS
SELECT
  user_id,
  product_id,
  COUNT(*) as implicit_rating  -- Nombre achats
FROM `dataset.purchases`
GROUP BY user_id, product_id;

-- Recommandations
SELECT
  user_id,
  ARRAY_AGG(STRUCT(product_id, predicted_rating)
    ORDER BY predicted_rating DESC LIMIT 10) as recommendations
FROM ML.RECOMMEND(MODEL `dataset.product_recs`)
GROUP BY user_id;
```

Cas d'Usage : Forecasting Demande

Time Series ARIMA

```
CREATE OR REPLACE MODEL `dataset.demand_forecast`
OPTIONS(
  model_type='ARIMA_PLUS',
  time_series_timestamp_col='date',
  time_series_data_col='daily_sales',
  holiday_region='US'
) AS
SELECT date, SUM(quantity) as daily_sales
FROM `dataset.sales`
GROUP BY date;

-- Prévision 30 jours
SELECT *
FROM ML.FORECAST(MODEL `dataset.demand_forecast`,
  STRUCT(30 AS horizon, 0.95 AS confidence_level));
```

Business Impact

Optimisation inventory, réduction ruptures stock

Best Practices : Architecture

Séparation Environnements

```
- dev-project
  - ml_experiments (dataset)
- staging-project
  - ml_models_staging (dataset)
- prod-project
  - ml_models_prod (dataset)
  - predictions (dataset)
```

Data Pipelines

- ETL séparé du ML
- Feature store centralisé
- Validation data quality

Governance

- IAM approprié par environnement
- Audit logging activé
- Documentation modèles

Best Practices : Performance

Données

- Partitionnement sur date d'entraînement
- Clustering sur features communes
- Matérialized views pour agrégations

Modèles

- Commencer simple (linear, logistic)
- Complexifier si nécessaire (boosted trees, DNN)
- Feature selection basée sur importance
- Hyperparameter tuning limité

Prédictions

- Batch predictions schedulées
- Caching résultats si possible
- ML.PREDICT avec filtres WHERE efficaces

Best Practices : Coûts

Optimisation

Stockage

- Supprimer anciennes versions modèles
- Expiration automatique partitions

Compute

- Dry-run avant entraînement
- Utilisez échantillons pour expérimentation
- Scheduled queries hors heures de pointe

Slots

- On-demand pour workloads variables
- Flat-rate si usage prévisible élevé

Monitoring

```
SELECT
  user_email,
  SUM(total_bytes_billed) / POW(10, 12) as tb_billed,
  SUM(total_bytes_billed) / POW(10, 12) * 6.25 as estimated_cost_usd
FROM `region-us`.INFORMATION_SCHEMA.JOBS_BY_PROJECT
WHERE creation_time >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
GROUP BY user_email
```

Troubleshooting Commun

Modèle ne converge pas

- Vérifier distributions features (normalisation)
- Outliers dans données
- Learning rate trop élevé
- Ajouter max_iterations

Performance faible

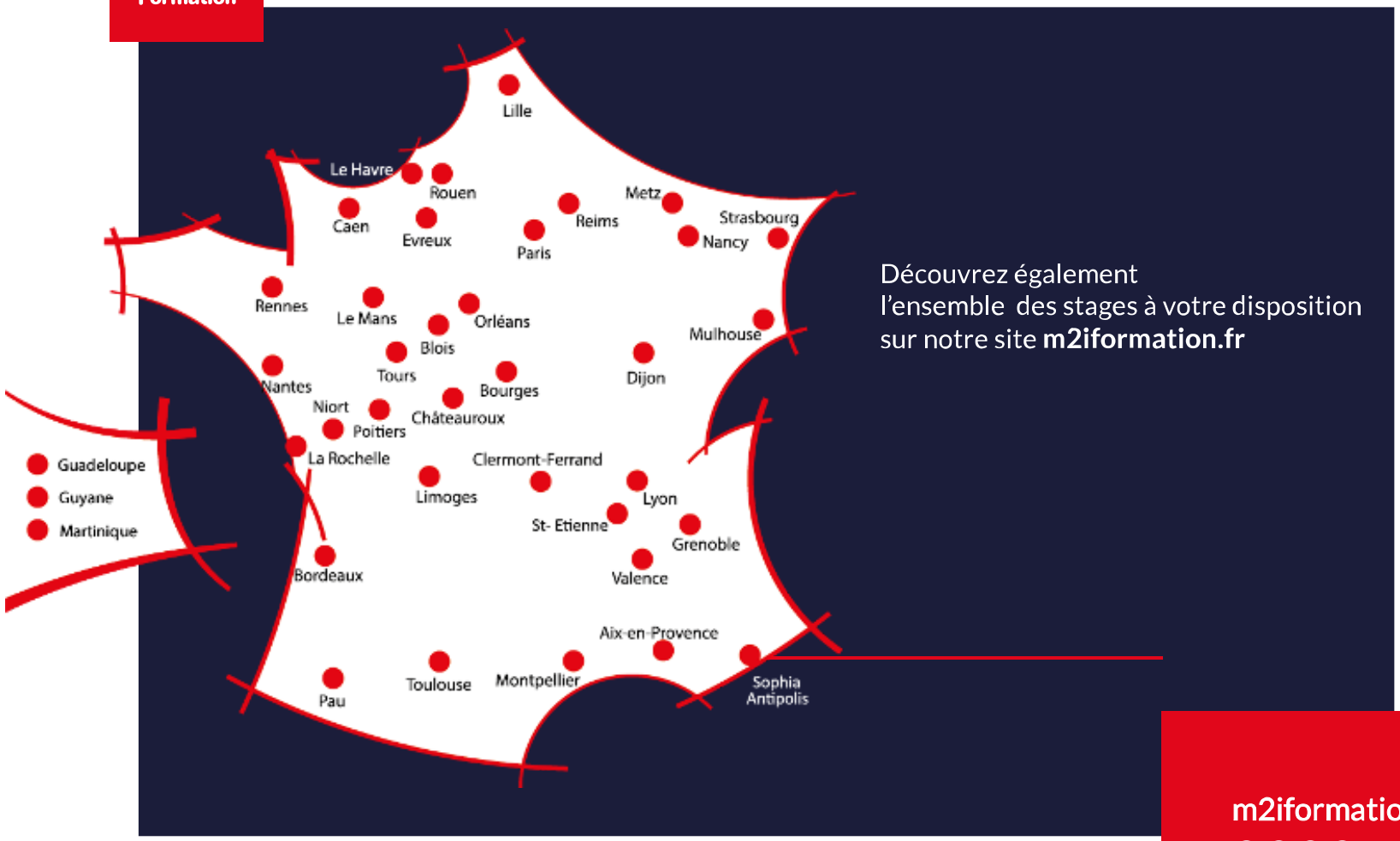
- Features non informatives
- Déséquilibre classes
- Overfitting : Augmenter régularisation
- Underfitting : Modèle plus complexe

Prédictions lentes

- Trop de données scannées : Ajouter filtres
- Modèle complexe : Simplifier si possible
- Utiliser cached results

Erreurs Quota

- Requêtes trop fréquentes : Batch



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr