Module 9 Entrepôts de données et OLAPs

IGE 487 Tudor Antohi

Objectifs

- Compléments ED et OLAPs
- SQLs typiques ED et CUBEs
- Implantation physique et performance

REVUE ED et Analytiques

Deux activités dans les bases de données

OLTP

– Traitement des transactions en ligne

– Transactions courtes

– Requêtes simples

– Touchez de petites portions de données

– Mises à jour fréquentes

- Normalisation

OLAP

- Traitement analytique en ligne

- Transactions longues

– Requêtes complexes

– Touchez de grandes parties des données

– Mises à jour peu fréquentes

- Dénormalisation en cubes et schémas étoiles

Deux activités dans les bases de données

Entrepôt de données

- Rassemblez les données des sources opérationnelles (OLTP) en un seul "entrepôt" pour l'analyse (OLAP)

Système d'aide à la décision (DSS)

- Infrastructure pour l'analyse des données
- Par exemple, un entrepôt de données adapté OLAP avec cubes et autres structures analytiques

Types de tables dans un entrepôt – revue

- **Faits**
- **Dimensions**
- **Hiérarchies**
- **Rallongées (outriggers)**
- **Mesures**
- **Slowly Changing Dimensions SCDs**

Types de tables de faits

Actualisation fréquentes, en mode insert :

- **Transactionnelles** – une ligne par transaction , évènementielle
- **Périodiques** – une rangée par période du temps
- **Cumulatives** – une rangée pour la vie d'un évènement

Ex.: ventes, inscriptions aux cours, vues de pages web

Dimensions – revue

Actualisations plus rares, petites tables, forunissent le contexte (qui, quoi, quand, ou, pourquoi et comment)

- Informations descriptives sur les valeurs numériques dans les tables de faits : ex.: Temps, Région, Type Produit
- En général pas plus de 15 (faire un merge des dimensions au cas ou plus on a plus de 15 modèles)
- Leurs attributs sont trouvés dans les clauses GROUP BY et WHERE

Ex.: magasins, étudiants, cours, pages web, utilisateurs

Dimensions – strategie d’historisation

Stratégies d'historisation:

– **SCD Type 1:** Écraser l'ancienne valeur avec la nouvelle

– **SCD Type 2:** Ajouter une ligne dans la table de dimension pour la nouvelle valeur

– **SCD Type 3:** Avoir deux colonnes dans la table de dimension correspondant à l'ancienne et la nouvelle valeur

Dimensions – SCD 1 et 2

idProduit	description	code	catégorie
1001	'BébéLala'	'ABC999-Z'	'Éducation'

idProduit	description	code	catégorie
1001	'BébéLala'	'ABC999-Z'	'Stratégie'

idProduit	description	code	catégorie	dateEffective	dateExpirée
1001	'BébéLala'	'ABC999-Z'	'Éducation'	'2007-10-08'	'9999-12-31'

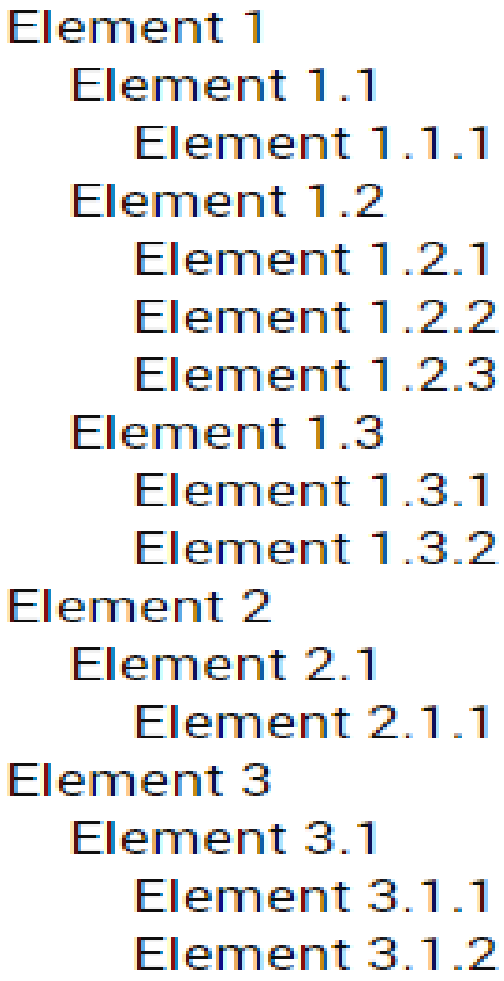
idProduit	description	code	catégorie	dateEffective	dateExpirée
1001	'BébéLala'	'ABC999-Z'	'Éducation'	'2007-10-08'	'2008-10-31'
1002	'BébéLala'	'ABC999-Z'	'Stratégie'	'2008-11-01'	'9999-12-31'

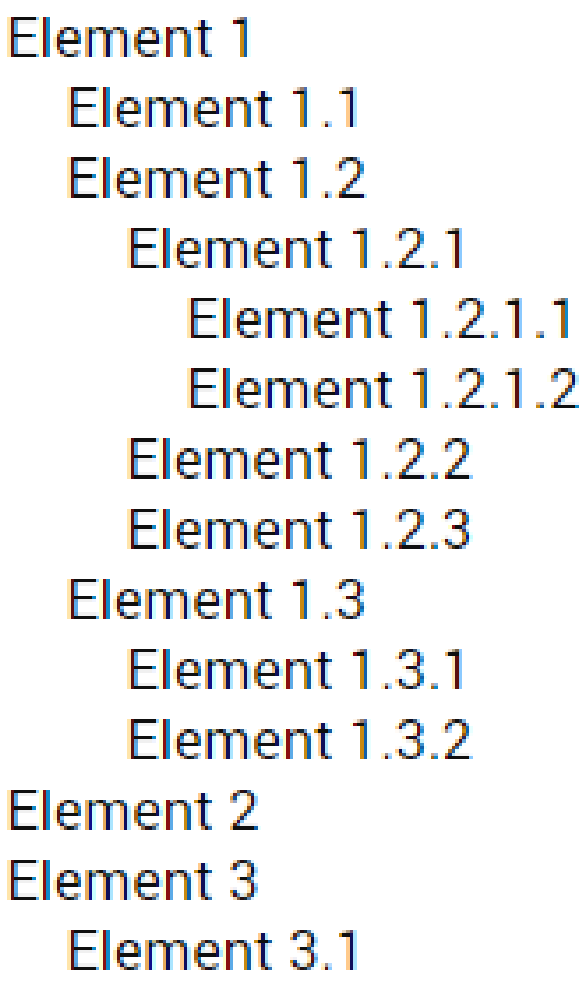
Hiérarchies (1)

- Une relation sur plusieurs nouveaux 1..N :

Année – Trimestre – Mois – Jour – Heure

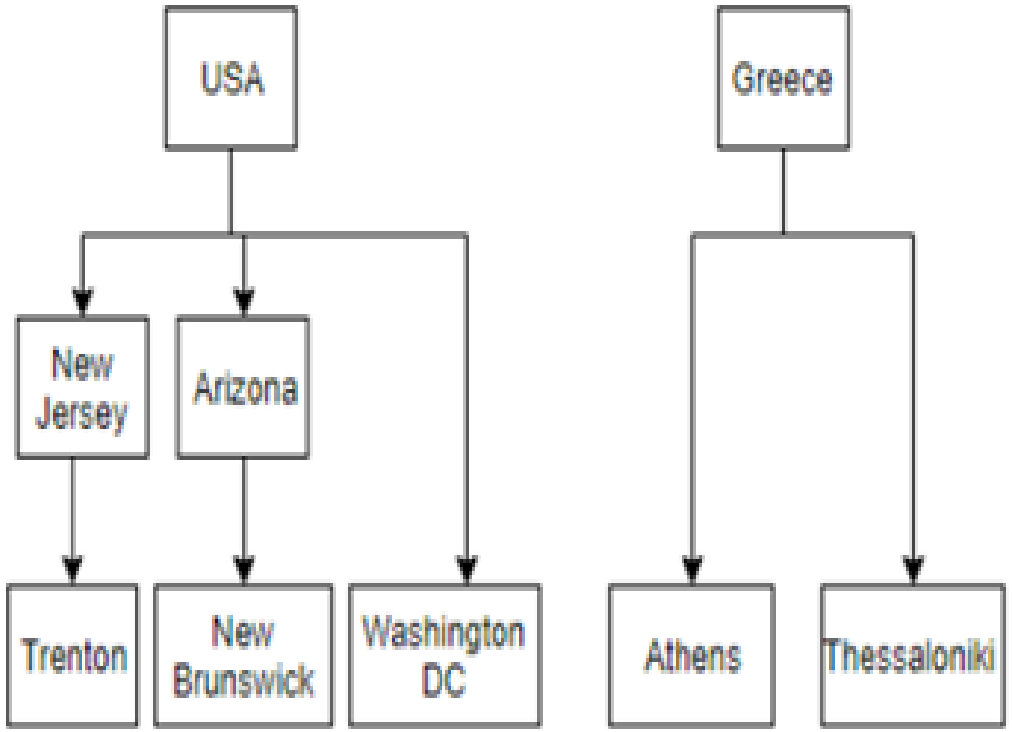
- Les hiérarchies peuvent être :
  - *Balancées*
  - *Non-balancées*





Hiérarchies (2)

Les hiérarchies **irrégulières** se produisent si les parents logiques de certains membres d'une hiérarchie de dimension sont séparés de deux niveaux ou plus. En d'autres termes, il existe des trous vides dans la hiérarchie des dimensions.



Rallongées - outriggers

- Les « Outriggers » sont utilisés pour améliorer les performances qui nous permettent de créer des modèles de données mieux optimisés.
- Généralement utilisée lorsqu'une table de dimensions devient volumineuse, en termes de nombre de colonnes.
- La grande table de dimensions est divisée en blocs gérables en fonction des relations de données, des besoins d'analyse et intégrée à l'entrepôt de données.
- Scénarios similaires peuvent survenir avec deux dimensions différentes se référençant l'une à l'autre avec des données factuelles, par ex. **Client et Géographie** sont deux dimensions où la dimension Client est liée à la Géographie basée sur le code postal.

Mesures – valeurs numériques dans une table de faits, la même granularité et synchronicité

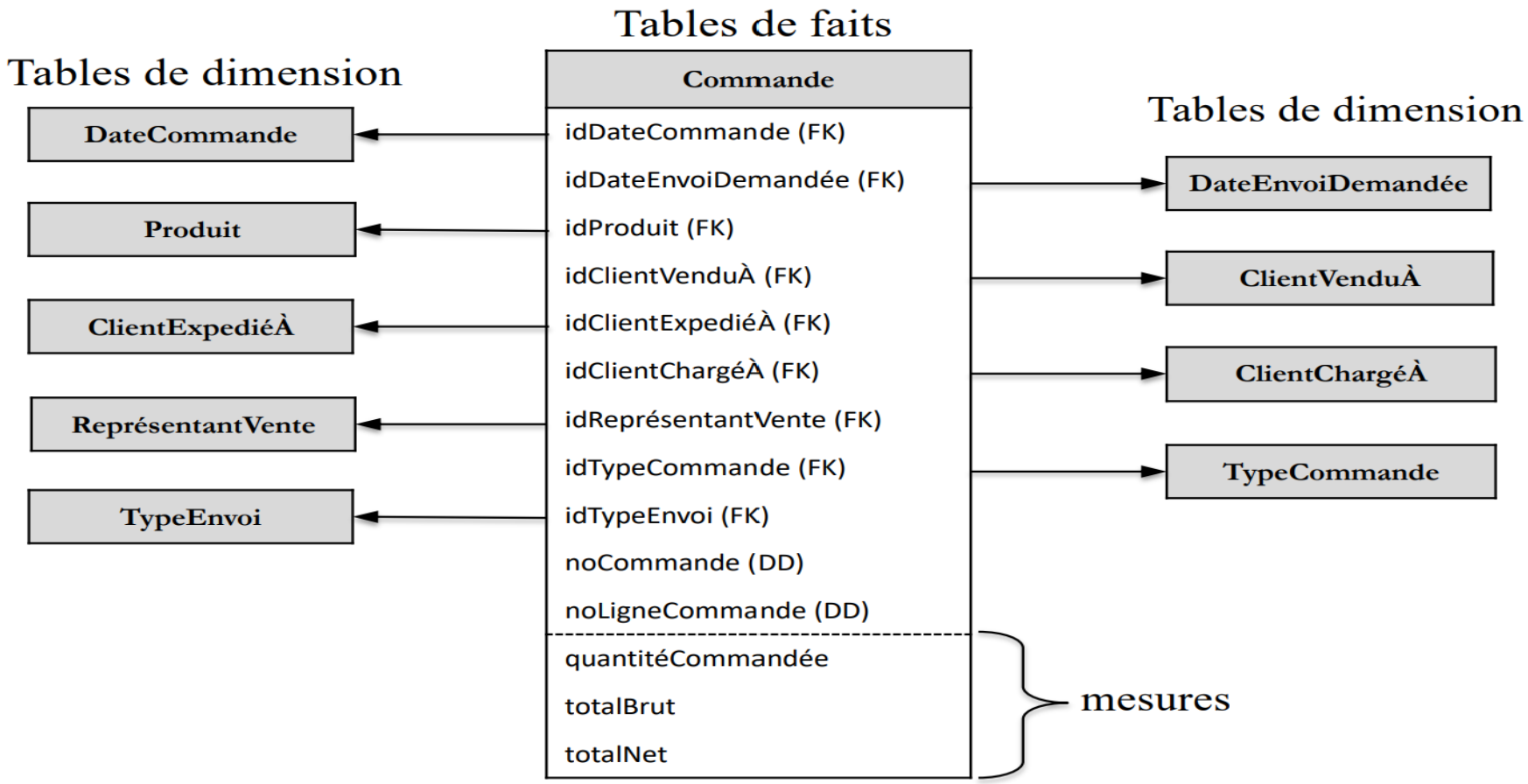


Schéma étoile

- La table de faits référence vers les tables des dimensions
- La table de faits contient des attributs de dimensions et des attributs dépendants.

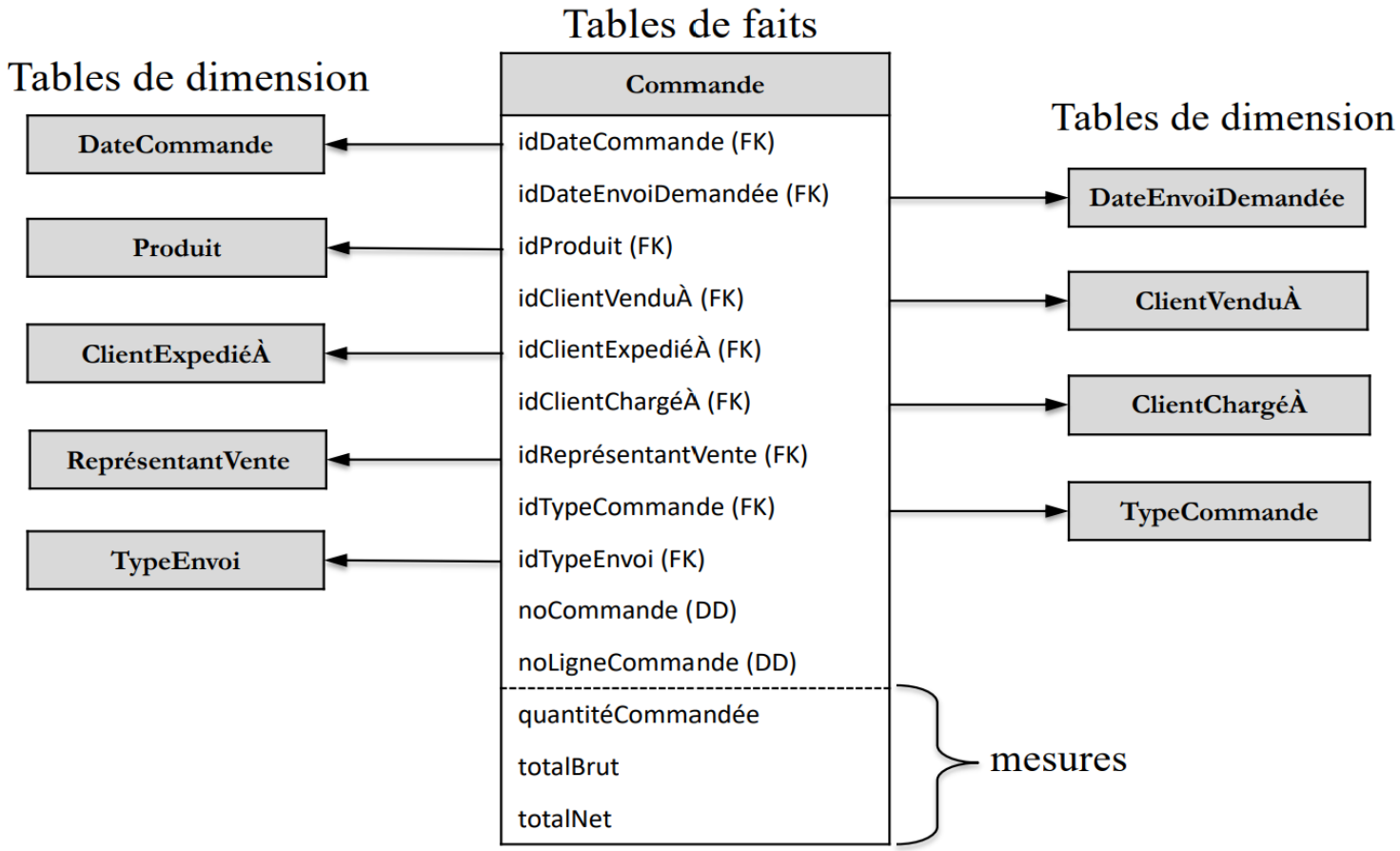
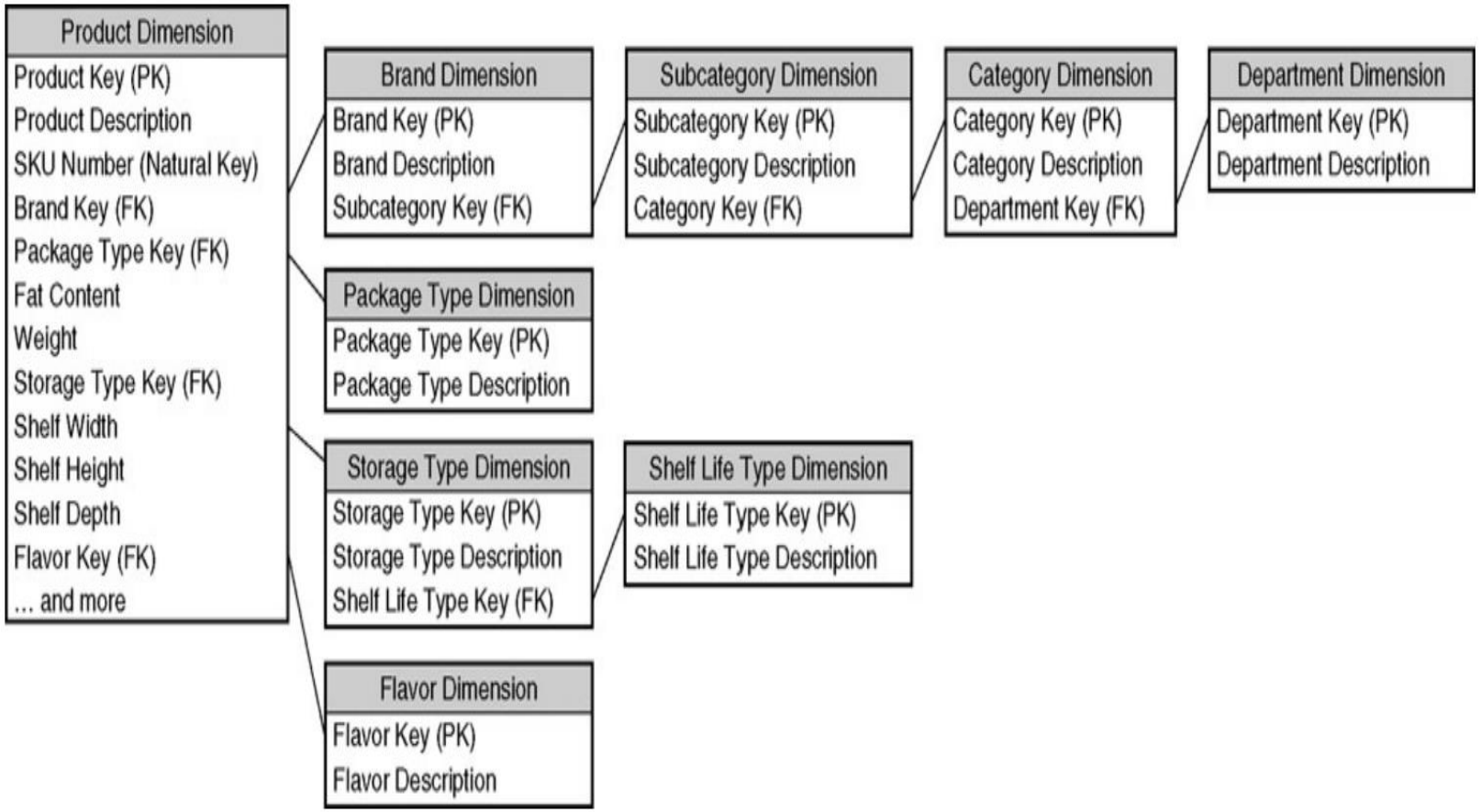


Schéma en flocons – les dimensions normalisées

- La table de faits référence vers les tables des dimensions
- La table de faits contient des attributs de dimensions et des attributs dépendants.

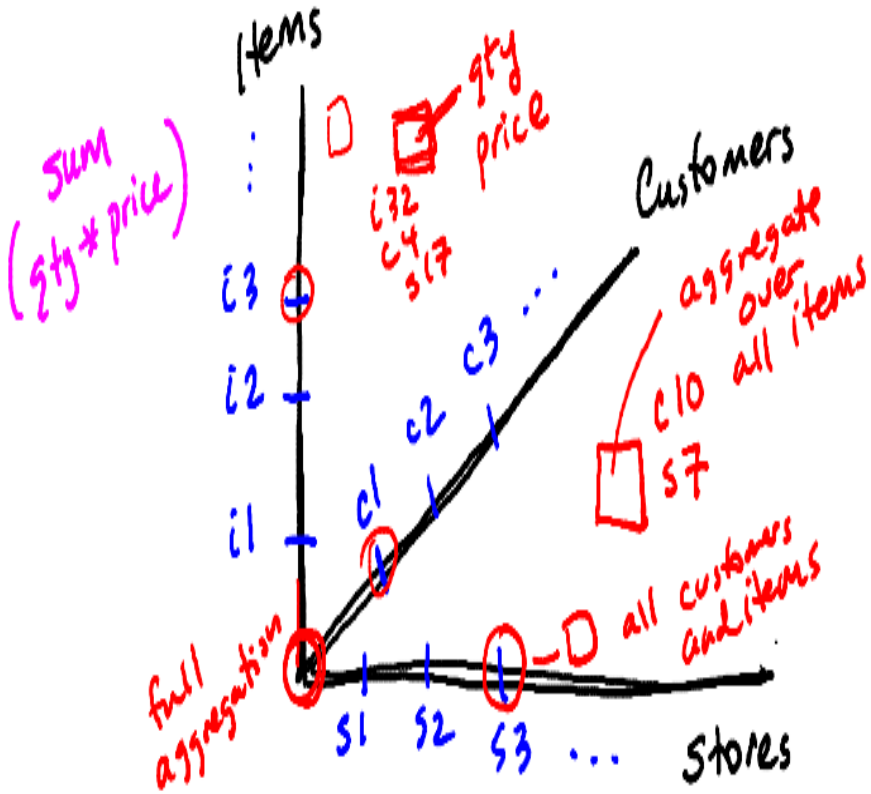


Cubes de données – OLAP multi-dimensions

Les **données de dimension** forment les axes du "cube". **Données factuelles** (dépendantes) dans les cellules. **Données agrégées** sur les côtés, les bords, les coins.

Ex.:

- **Coin** – agrégation complète
- **Côtés ex.: i1,2,3** – agrégation group by sur Clients et Magasins filtrée pour Item 1
- **Faces intersection s2,c2** – agrégation group by par i , filtrage par s2,c2.



SQLs typiques ED et CUBEs

Constructions SQL typique Cube (1)

Select dimension-attrs, aggregates
From tables (faits ou dimensions ou cubes)
Where conditions
Group By
--Dimension-attrs
--Cube(dimension-attrs)
--Rollup(dimension-attrs***)***

Ajouter au résultat : faces, arêtes et coins de cube en utilisant des valeurs NULL

Constructions SQL typique ED et Cube (2)

Ajouter au résultat : face D1,D2,D3 sur ED

==========

```
Select D1,D2,
aggregates sur F
From F join D1 join
D2
Where D3.attragg =
valeur
Group By D1,D2
```

Ajouter au résultat : face D1,D2 sur Cube déjà calculé

==========

```
Select Valeur
From Cube
Where d1 is null and
d2 is null and d3 =
Valeur
```

Constructions SQL typique ED et Cube (2)

Ajouter au résultat : faces, arêtes et coin de cube en utilisant des valeurs NULL

```
Select dimension-attrs, aggregates
From F, D1,D2,D3
Where conditions AND dimension1 is NULL
Group By
--Dimension-attrs
--Cube(dimension-attrs)
--Rollup(dimension-attrs)
```

Constructions SQL drill-down et roll-up

Drill-down – on ajoute la catégorie

Select state, **(category),** brand, Sum(qty*price)

From Sales F, Store S, Item I

Where F.storeID = S.storeID And F.itemID = I.itemID

Group By state, **(category),** brand

Roll-up – on élimine state

Select ~~state,~~ brand, Sum(qty*price)

From Sales F, Store S, Item I

Where F.storeID = S.storeID And F.itemID = I.itemID

Group By ~~state~~, brand

Question 1

Si nous avons 2 magasins, 5 articles et 10 clients, combien d'entrées potentielles y a-t-il dans le cube de données et pourquoi ?

1. 17
2. 100
3. 117
4. **198**

Question 2 – ex. de drill down et rollup

Q1: Select itemID, color, size, Sum(qty*unitPrice)

From Sales

Group By itemID, color, size

Q2: Select itemID, size, Sum(qty*unitPrice)

From Sales

Group By itemID, size

Q3: Select itemID, size, Sum(qty*unitPrice)

From Sales

Where size < 10

Group By itemID, size

Q1 vers Q2 ?

Q3 vers Q1 ?

Q2 vers Q1 ?

Q2 vers Q3 ?

Q1 vers Q3 ?

Analytiques directes sur les tables normalisées

Si on n’a pas la patience ou $ pour construire des entrepôts de données , on a des solutions SQL (qui peuvent tuer les systèmes en passant)

- virtualiser les entrepôts et les cubes avec de vues
- calcules sur les hiérarchies de dimensions d’une façon récursive
- calcules des fenêtres avec fonctions d’agrégations
- utiliser des SQL APIs complexes et les exposer comme des tables (ex.: science de données dans les bases de données temporelles)

SQL de base n’est pas une machine Turing

Simple, pratique, déclaratif , assez expressif pour la plupart des requêtes de base de données. **Mais le SQL de base ne peut pas exprimer des calculs illimités qui peuvent être produites par les hiérarchies.**

Exemples :

Parents : ParentDe(parent,enfant) – trouves les ancêtres de Marie

Compagnie : Employé (Id, salaire), Gestionnaire (mID, eID), Projet(nom, mId) – trouve le cout de projet

Vols : Vol(origine, destination, ligne aérienne, cout) – trouver le meilleur cout de A à B

Récursivité linéaire pour les dimensions hiérarchiques

WITH R1(A1,A2,…,Am)

AS (query-1),

R2 **AS** (query-2), ...

Rn **AS** (query-n)

<query involving R (and other tables)>

WITH RECURSIVE R

AS

(base query)

UNION

recursive query --- *qui fait référence a R*)

<query involving R (and other tables)>

Ex.: ParentDe – les ancêtres de Marie(1)

```
WITH RECURSIVE Ancetre(a,d) AS
(SELECT parent as a , enfant AS d from ParentDe
UNION
SELECT
Ancetre.a ,
ParentDe.enfant AS d
FROM Ancetre, ParentDe
WHERE Ancetre.d = ParentDe.parent
)
SELECT a FROM Ancetre WHERE Ancetre.d = ‘Marie’
```

La façon typique de définir une requête récursive est d'avoir une requête de base et ce serait sur non, sur des tables autres que R donc pas R dans cette requête de base.

En quelque sorte pour démarrer la récursivité, puis R sera le résultat de cette requête de base avec la **requête récursive.** Donc, ici, nous ferons référence à R.

Le résultat de R, le R qui est vu lorsque nous exécutons la requête ici, est ce que l'on appelle le **point fixe de l'exécution** de cette union encore et encore et la fin est **jusqu'à ce que nous n'ajoutions plus de tuples supplémentaires à R** .

Ex.: ParentDe – les ancêtres de Marie (2)

```
WITH RECURSIVE Ancetre(a,d) AS
(SELECT parent as a , enfant AS d
from ParentDe
UNION
SELECT Ancetre.a , ParentDe.enfant
AS d FROM Ancetre, ParentDe
WHERE Ancetre.d =
ParentDe.parent
)
SELECT a FROM Ancetre WHERE
Ancetre.d = 'Marie'
```

1. La table Parent.De:

Enfant Parent

Marie Louise
Marie Basile
Basile Marc
Basile Helene
Louise Philippe
Louise Diane

2. La relation de base en ordre parent enfant

Louise Marie
Basile Marie
Marc Basile
Helene Basile
Philippe Louise
Diane Louise

3. Union avec la jointure entre les deux relations avec tuples similaires dans la première phase ajoute les ancêtres des parents en gardant le reste :

Louise Marie
Basile Marie
Marc Basile
Helene Basile
Philippe Louise
Diane Louise

Marc Marie
Helene Marie
Philippe Marie
Diane Marie

4. Prochaine exécution entre le resultat de 3 et la base de 2 ne change rien, donc la sélection finale est le numéro 3

Récursivité non-linéaire

```
WITH RECURSIVE Ancetre(a,d) AS
(SELECT parent as a , enfant AS d FROM ParentDe
UNION
SELECT A1.a a, A2.a AS d
FROM Ancêtre A1, Ancêtre A2
WHERE A1.d = A2.d
)
SELECT a FROM Ancêtre WHERE Ancêtre.d = ‘Marie’
```

Récursivité mutuelle

```
WITH RECURSIVE
R1 AS (Q1),
R2 AS (Q2), -- Reference vers R1
R2 AS (Q3), -- Reference vers R2
------
Rn AS (Qn), -- reference vers Rn-1
 )
SELECT * FROM R1,R2, …, Rn
```

Windows (1) – fenêtres

Le terme fenêtre décrit l'ensemble des lignes sur lesquelles opère la fonction.

Une fonction de fenêtre utilise les valeurs des lignes d'une fenêtre pour calculer les fonctions.

```
select emp_name, dealer_id, sales, avg(sales) over() as avgsales from q1_sales;
|-----------------|------------|--------|-----------|
|    emp_name     | dealer_id  | sales  | avgsales  |
|-----------------|------------|--------|-----------|
| Beverly Lang    | 2          | 16233  | 13631     |
| Kameko French   | 2          | 16233  | 13631     |
| Ursa George     | 3          | 15427  | 13631     |
| Ferris Brown    | 1          | 19745  | 13631     |
| Noel Meyer      | 1          | 19745  | 13631     |
| Abel Kim        | 3          | 12369  | 13631     |
| Raphael Hull    | 1          | 8227   | 13631     |
| Jack Salazar    | 1          | 9710   | 13631     |
| May Stout       | 3          | 9308   | 13631     |
| Haviva Montoya  | 2          | 9308   | 13631     |
|-----------------|------------|--------|-----------|
```

Windows (2) – partition by

La clause PARTITION BY est une sous-clause de la clause OVER. La clause PARTITION BY divise les résultats d'une requête en partitions. La fonction de fenêtre est appliquée pour chaque partition.

```
SELECT
    first_name,
    last_name,
    department_id,
    ROUND(AVG(salary) OVER (
        PARTITION BY department_id
    )) avg_department_salary
FROM
    employees;
```

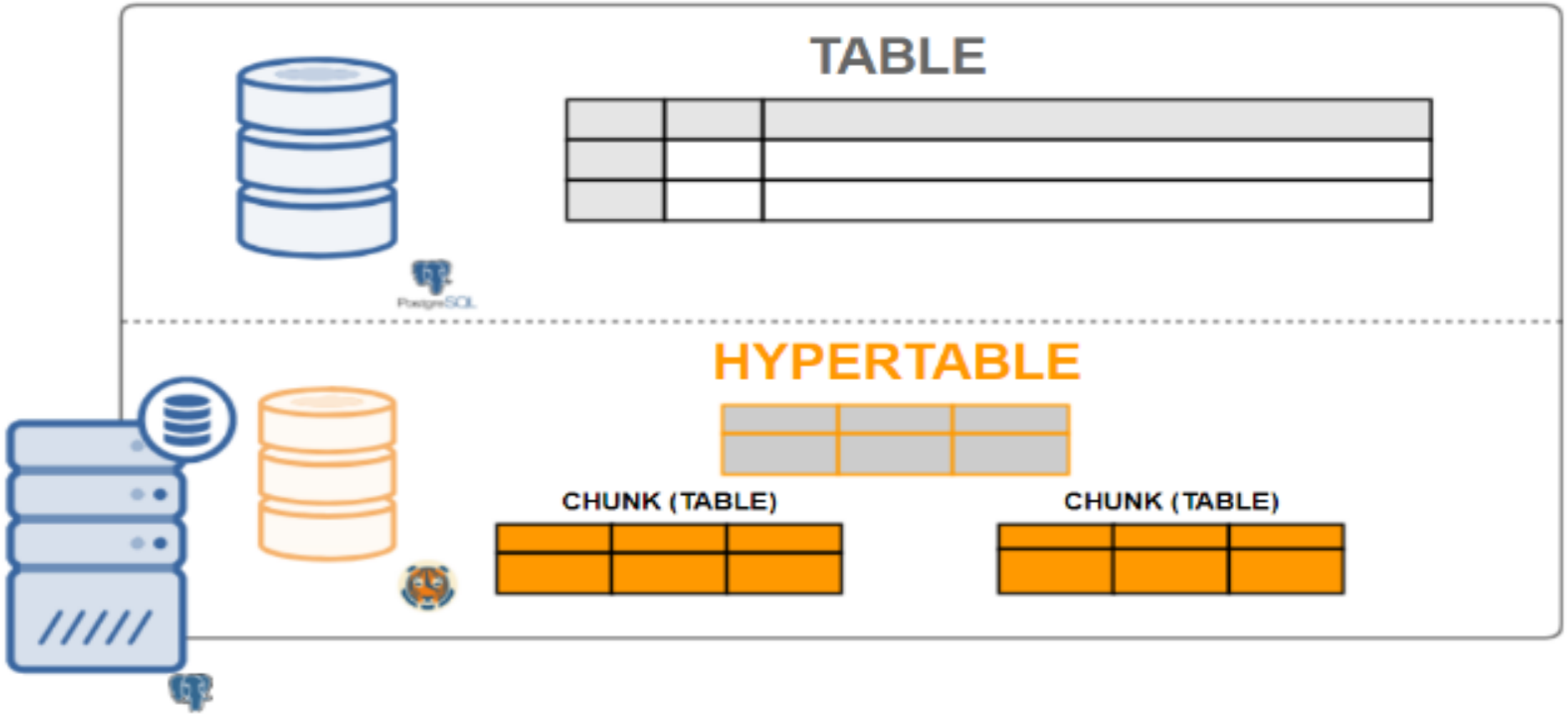
first_name	last_name	salary	department_id	avg_department_salary
Jennifer	Whalen	4400.00	1	4400
Michael	Hartstein	13000.00	2	9500
Pat	Fay	6000.00	2	9500
Den	Raphaely	11000.00	3	4150
Alexander	Khoo	3100.00	3	4150
Shelli	Baida	2900.00	3	4150
Sigal	Tobias	2800.00	3	4150
Guy	Himuro	2600.00	3	4150
Karen	Colmenares	2500.00	3	4150
Susan	Mavris	6500.00	4	6500
Matthew	Weiss	8000.00	5	5886
Adam	Fripp	8200.00	5	5886
Payam	Kaufling	7900.00	5	5886

Windows (3)

Il existe d'autres fonctions a parte de MAX, MIN, SUM etc.

AVG *	BIT_AND_AGG *	BIT_OR_AGG *	BIT_XOR_AGG *	CHECKSUM *
CLUSTER_DETAILS	CLUSTER_DISTANCE	CLUSTER_ID	CLUSTER_SET	CORR *
COUNT *	COVAR_POP *	COVAR_SAMP *	CUME_DIST	DENSE_RANK
FEATURE_DETAILS	FEATURE_ID	FEATURE_SET	FEATURE_VALUE	FIRST
FIRST_VALUE *	KURTOSIS_POP *	KURTOSIS_SAMP *	LAG	LAST
LAST_VALUE *	LEAD	LISTAGG	MATCH_RECOGNIZE	MAX *
MEDIAN	MIN *	NTH_VALUE *	NTILE	PERCENT_RANK
PERCENTILE_CONT	PERCENTILE_DISC	PREDICTION	PREDICTION_COST	PREDICTION
PREDICTION_COST	PREDICTION_DETAILS	PREDICTION_PROBABILITY	PREDICTION_SET	RANK
RATIO_TO_REPORT	REGR_ (Linear Regression) Functions *	ROW_NUMBER	SKEWNESS_POP *	SKEWNESS_SAMP *
STDDEV *	STDDEV_POP *	STDDEV_SAMP *	SUM *	VAR_POP *
VAR_SAMP *	VARIANCE *	String Aggregation	Top-N Queries	

TimescaleDB (1) – analytiques temporelles



TimescaleDB architecture

TimescaleDB (2) – APIs comme table

```
CREATE EXTENSION timescaledb;
CREATE TABLE conditions (
  time        TIMESTAMPTZ       NOT NULL,
  location    TEXT              NOT NULL,
  temperature DOUBLE PRECISION  NULL,
  humidity    DOUBLE PRECISION  NULL
);
SELECT create_hypertable('conditions',
'time');
INSERT INTO conditions(time, location,
temperature, humidity) VALUES (NOW(),
'office', 70.0, 50.0);
```

```
SELECT
time_bucket('15 minutes', time)
AS fifteen_min, location,
COUNT(*), MAX(temperature) AS
max_temp, MAX(humidity) AS
max_hum
FROM conditions
WHERE
time > NOW() - interval '3 hours'
GROUP BY fifteen_min, location
ORDER BY fifteen_min DESC,
max_temp DESC;
```

Implantation physique et optimisation de la performance de systèmes OLAP

Implantation physique et optimisations

Tables normales avec partitionnement et indexes

- Rafraichissement Complete
- Rafraichissement Incrémental – **MERGE**

Vues matérialisées pour les EDs et les Cubes et autres agrégations :

- Rafraichissement Complete
- Rafraichissement Incrémental – utilisant de journaux sur les tables de base on construit une requête delta qui prend en charge les changements

Optimisation par partitionnement (1)

Le partitionnement fait référence au fractionnement de ce qui est logiquement une grande table en plus petits morceaux physiques.

Range = la table est partitionnée en « plages » définies par une colonne clé ou un ensemble de colonnes, sans chevauchement entre les plages de valeurs attribuées aux différentes partitions.

List = la table est partitionnée en répertoriant explicitement les valeurs de clé qui apparaissent dans chaque partition.

Hash = la table est partitionnée en spécifiant par exemple une fonction basée sur un module et un reste pour chaque partition. Chaque partition contiendra les lignes pour lesquelles la valeur de hachage de la clé de partition divisée par le module spécifié produira le reste spécifié.

Composé = diverses alternatives , partitionnement et sous-partitionnement

Optimisation par partitionnement (2)

```
CREATE TABLE measurement
( city_id int not null,
logdate date not null,
peaktemp int,
unitsales int
);
```

```
CREATE TABLE measurement
( city_id int not null,
logdate date not null,
peaktemp int,
unitsales int
PARTITION BY RANGE
(logdate)
);
```

```
CREATE TABLE measurement_y2006m03 PARTITION OF measurement
FOR VALUES FROM ('2006-03-01') TO ('2006-04-01');
```

Optimisation par indexes en partitions (3)

Index globaux

Un index un-à-plusieurs, dans lequel un index correspond à toutes les tables partitionnées. Pas en PostgreSQL mais en Oracle oui

Index partitionné (partitionnement d'index) sur toutes les partitions

Lorsque les index globaux deviennent trop volumineux, ils sont partitionnés pour que les performances et la maintenance restent gérables. On peut partitionner par rapport au partitionnement de données ou partitionnement privé personnalisation pour index.

Index local sur une partition

Un index local est un index local à une partition spécifiée d’une table spécifique ; c'est-à-dire qu'il ne s'étend pas sur plusieurs partitions. PostgreSQL utilise la terminologie « index partitionné » lorsqu'il fait référence à des index locaux.

Indexes en jointures en étoiles (1)

- Précalculer les lignes des **tables** de dimension pouvant être jointes avec la table de faits;
- Évite de **joindre les tables de dimension les unes après les autres** en ordre de normalisation.
- **Bitmap join index** (Oracle)
  - Index bitmap sur des colonnes situées dans des tables de dimension différentes;
  - Les colonnes à pré-joindre doivent avoir un domaine restreint (comme pour les index bitmap standards);
  - Peut accélérer jusqu'à 10 fois la jointure (benchmarks Oracle)

Indexes en jointures en étoiles (2)

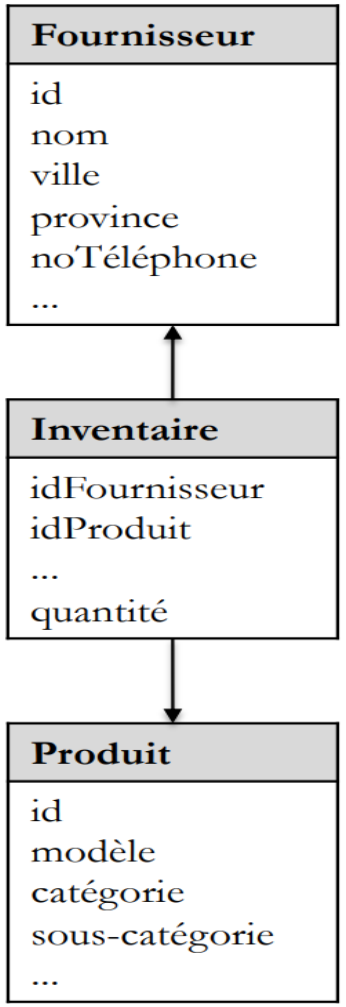
Bitmap join index (Oracle)

- Requête à optimiser:

```
SELECT SUM(Inventaire.quantité)
FROM Inventaire, Produit, Fournisseur
WHERE Inventaire.idProduit = Produit.id AND
      Inventare.idFournisseur = Fournisseur.id AND
      Produit.catégorie='moteur_mazda' AND
      Fournisseur.province='QC'
```

- Index à créer:

```
CREATE BITMAP INDEX indexJointure ON
Inventaire(Fournisseur.province, Produit.catégorie)
FROM Inventaire, Fournisseur, Produit
WHERE Inventaire.idProduit = Produit.id AND
      Inventare.idFournisseur = Fournisseur.id
```

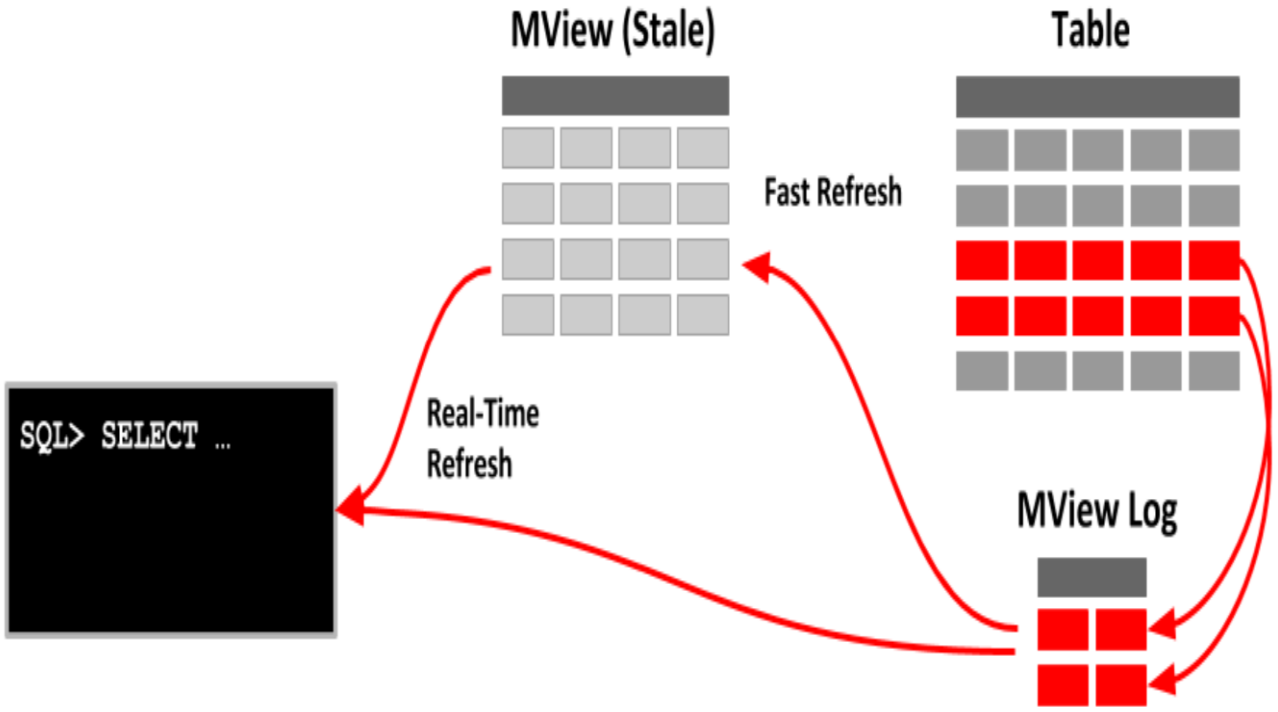


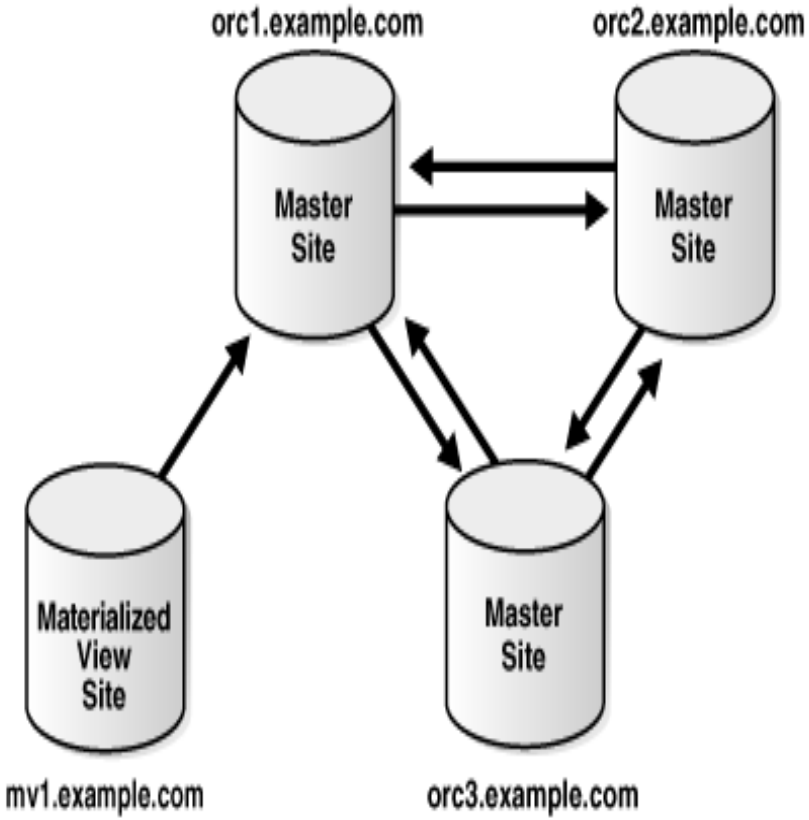
Vue matérialisée – physique

- Table physique synchronisée avec les résultats d'une requête
- Synchronisation temps-réel, en lot ou sur demande
- Permet les indexes, le partitionnement, contrôles d'accès, etc.
- Hiérarchie d'agrégations possible en créant une nouvelle vue à partir d'autres vues

```
CREATE MATERIALIZED VIEW cust_mth_sales_mv
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE AS
SELECT s.time_id, s.prod_id,
SUM(s.quantity_sold), SUM(s.amount_sold),
       p.prod_name, t.calendar_month_name,
COUNT(*),COUNT(s.quantity_sold),
COUNT(s.amount_sold)
FROM sales s, products p, times t
WHERE  s.time_id = t.time_id AND s.prod_id =
p.prod_id
GROUP BY t.calendar_month_name, s.prod_id,
p.prod_name, s.time_id;
```

MV – réplication rapide





MV – exemple recent (1)

Oracle 12.2 a introduit le concept de vues matérialisées en temps réel, qui permettent un retour en arrière au niveau de l'instruction d'une vue matérialisée obsolète, faisant apparaître les données comme fraîches dans l'instruction.

```
CREATE TABLE order_lines (
  id           NUMBER(10),
  order_id     NUMBER(10),
  line_qty     NUMBER(5),
  total_value  NUMBER(10,2),
  created_date DATE,
  CONSTRAINT orders_pk PRIMARY KEY (id)
);
```

MV – exemple recent (2)

```
DROP MATERIALIZED VIEW
LOG ON order_lines;

CREATE MATERIALIZED
VIEW LOG ON order_lines
WITH ROWID,
SEQUENCE(order_id,
line_qty, total_value)
INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW
order_summary_rtmv
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
ENABLE ON QUERY COMPUTATION
AS
SELECT order_id,
    SUM(line_qty) AS sum_line_qty,
    SUM(total_value) AS sum_total_value,
    COUNT(*) AS row_count
FROM   order_lines
GROUP BY order_id;
```

MV – exemple recent (3)

EXEC DBMS_MVIEW.refresh ('EMP_MV’,’F’);

```
SELECT order_id,       SUM(line_qty) AS sum_line_qty,
SUM(total_value) AS sum_total_value,       COUNT(*) AS row_count FROM
 order_lines WHERE  order_id = 1 GROUP BY order_id

Plan hash value: 1165901663

-----------------------------------------------------------------------------------------
| Id  | Operation                  | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                   |       |       |     4 (100)|          |
|* 1 |  MAT_VIEW REWRITE ACCESS FULL| ORDER_SUMMARY_RTMV |     1 |    17 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

MV – refresh force et enable rewrite

Paramètre	Description
REFRESH FORCE	Synchronisation incrémentale lorsque possible, sinon complète
ENABLE QUERY REWRITE	Permet de réécrire la requête si cela améliore la performance

Rafraichissement par changement de :

- **Table**
- **Partition**

Il existe des logiciels capable de regarder le code de la vue matérialisée et évaluer la possibilité incrémentale et donner des recommandations

```
MVNAME               CAPABILITY_NAME   POSSIBLE  RELATED_TEXT  MSGTXT
-------------------  ---------------   --------  ------------  ----------------
CUST_MTH_SALES_MV    PCT               Y         SALES
CUST_MTH_SALES_MV    PCT_TABLE         Y         SALES
CUST_MTH_SALES_MV    PCT_TABLE         N         PRODUCTS      no partition key
                                                                or PMARKER
                                                                in SELECT list
CUST_MTH_SALES_MV    PCT_TABLE         N         TIMES         relation is not
                                                                partitionedtable
```

Évolutions 2022

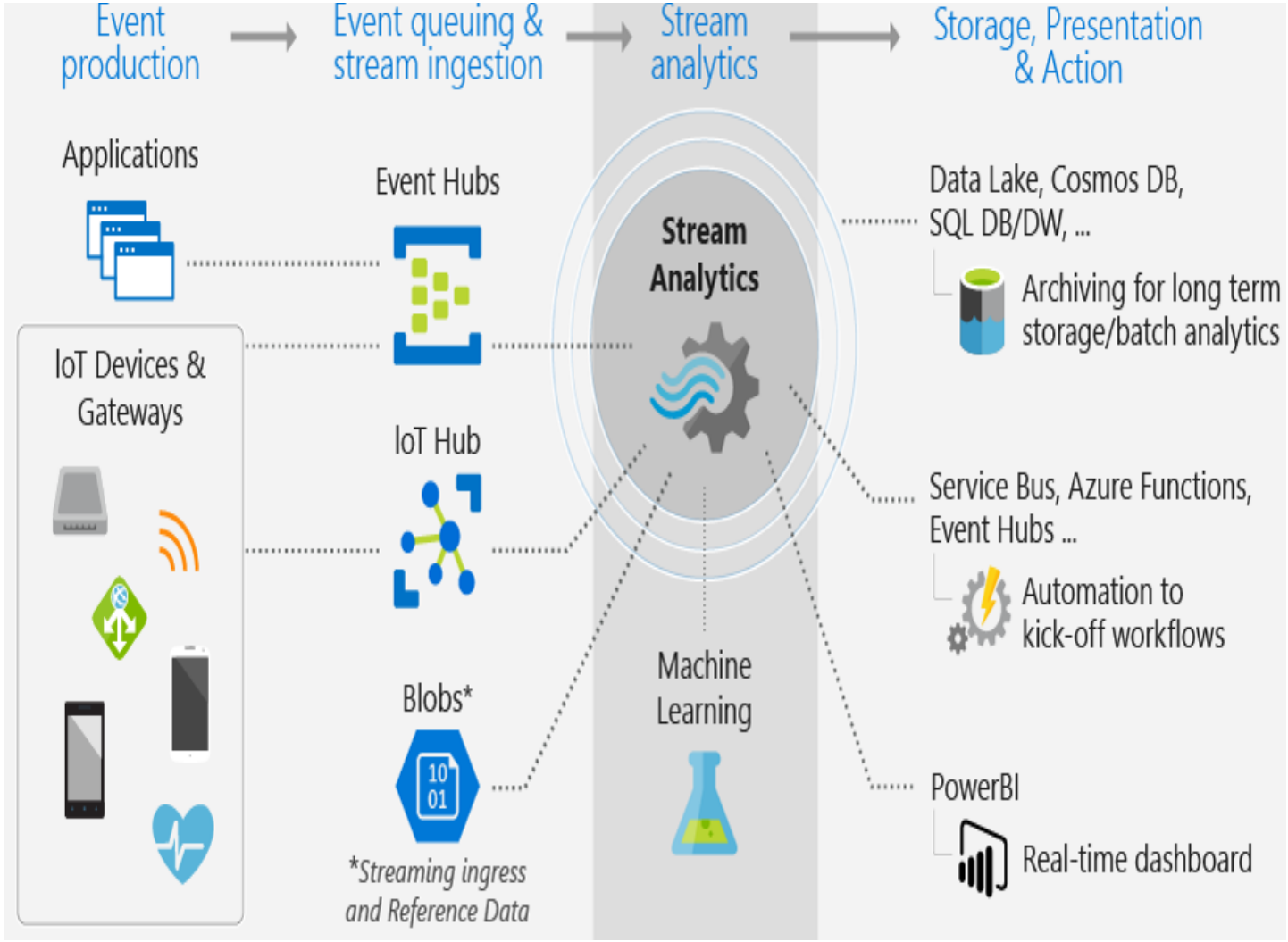
Power BI prend la place de Azure Analytics et SSAS.

- **Modelé multidimensionnel,** mature, mise a échelle de stockage, utilise des agrégations prédéfinies, langage MDX difficile
- **Tabulaire ,** solution in-memory, demande beaucoup de RAM, utilise DAX similaire à Excel , plus rapide, lecture des informations en mode colonne en mémoire (pensez 1NF ;-), utilise le relationnel

Évolutions 2022 analytiques en streaming

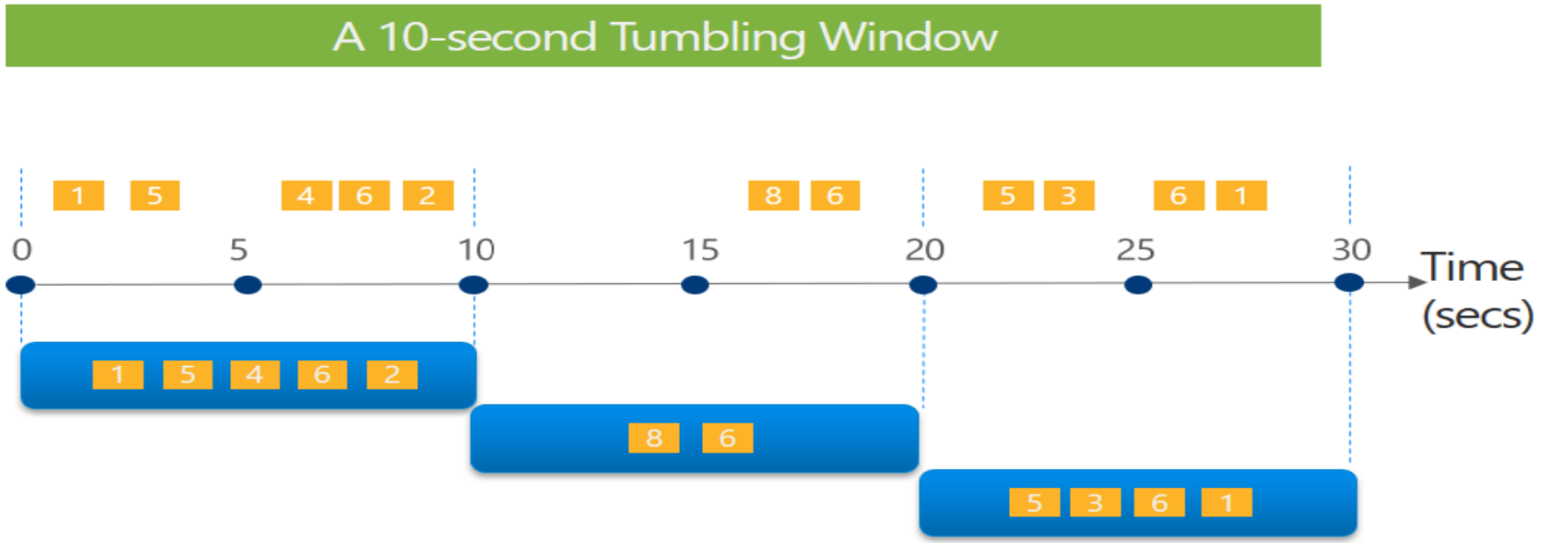
Streaming Analytics

1. Consommation des messages des **IdO** (a partir de queues de messages)
2. Analytiques avancées
3. Publication dans des outils de visualisation



Exemple

Tell me the count of tweets per time zone every 10 seconds




```
SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
```

Évolutions 2022

1. Microstrategy
2. Azure Databricks
3. Azure Analytics
4. SSAS
5. Azure Synapse Analytics
6. Azure Streaming Analytics