Entrepôt de données
M1 Informatique

Melha MEHMEL
Alexis PLESSIER

UNIVERSITÉ Clermont Auvergne

Compte Rendu

TP 2-3-4

TP2

Question 1 :

- **Caractéristiques des partitions ACTVARS1:**

SQL> desc actvars1;

Name	Null?	Type
CUSTOMER_LEVEL	NOT NULL	CHAR(12)
PRODUCT_LEVEL	NOT NULL	CHAR(12)
CHANNEL_LEVEL	NOT NULL	CHAR(12)
TIME_LEVEL	NOT NULL	VARCHAR2(12)
UNITSSOLD	NOT NULL	FLOAT(126)
DOLLARSALES	NOT NULL	FLOAT(126)
DOLLARCOST		FLOAT(126)

SQL> SELECT segment_name, bytes, bytes/1024/1024 MB,segment_type FROM user_segments WHERE segment_name = 'ACTVARS1' ;

SEGMENT_NAME	BYTES	MB	SEGMENT_TYPE
ACTVARS1	192937984	184	TABLE PARTITION
ACTVARS1	218103808	208	TABLE PARTITION

- **Caractéristique des partitions TIMELEVEL1:**

SQL> desc TimeLevel1;

Name	Null?	Type
TID	NOT NULL	VARCHAR2(12)
YEAR_LEVEL	NOT NULL	NUMBER(4)
QUARTER_LEVEL	NOT NULL	VARCHAR2(6)
MONTH_LEVEL		NUMBER(2)

SQL> SELECT segment_name, bytes, bytes/1024/1024 MB,segment_type FROM user_segments WHERE segment_name = 'TIMELEVEL1';

SEGMENT_NAME BYTES MB SEGMENT_TYPE
---------- ---------- ---------- ----------
TIMELEVEL1 8388608 8 TABLE PARTITION
TIMELEVEL1 8388608 8 TABLE PARTITION

Question 2:

Q1:
```
SELECT Time_level, SUM(DollarSales)
FROM ACTVARS
WHERE Time_level = '199506'
GROUP BY Time_level;
```

Q2:
```
SELECT Tid, SUM(DollarSales)
FROM ACTVARS, TIMELEVEL
WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06
GROUP BY Tid;
```

SQL> **Set timing on**

☐ **Etape 1 :**

- **Q1 sur ACTVARS**

SQL> select time_level, sum(DollarSales) from actvars where time_level = '199506' group by time_level;

TIME_L SUM(DOLLARSALES)
------ ----------
199506 62017446

Elapsed: 00:00:04.43

→ Requête classique qui va passer en revue toute la table ACTVARS où time_level = 1995 donc temps long.

- **Q2 sur ACTVARS et TIMELEVEL**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.58

☐ **Etape 2**:

- **Q1 sur ACTVARS1**

SQL> select time_level, sum(DollarSales) from actvars1 where time_level = '199506' group by time_level;

TIME_L SUM(DOLLARSALES)
------ ----------
199506 62017446

Elapsed: 00:00:03.14

→ **On remarque une réduction du temps d'exécution de la requête Q1 par rapport à l'étape 1 (le temps d'exécution de la requête Q1 est plus efficace sur ACTVARS1 que sur ACTVARS , diminution du nombre d'E/S grâce au partitionnement)**

- **Q2 sur ACTVARS et TIMELEVEL1**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.63

→ **Temps légèrement supérieur, car la 1ère partition de TIMELEVEL 1 qui est utilisée est passée en intégralité alors qu'elle possède des valeurs allant jusqu'en 09/1995 sur le Tid. Pour construire la table, on utilise tout ACTVARS et la 1ère partition deTIMELEVEL1.**

- **Q2 sur ACTVARS1 et TIMELEVEL**

SQL> **SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;**

```
TID         SUM(DOLLARSALES)
----------- ----------------
199506                62017446
```

Elapsed: 00:00:00.38

→ Temps inférieur, car la table est construite avec la 1ère partition de ACTVARS 'jointe à TIMELEVEL donc moins d'Entrées à passer en revue.

- **Q2 sur ACTVARS1 et TIMELEVEL1**

SQL> **SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;**

```
TID         SUM(DOLLARSALES)
----------- ----------------
199506                62017446
```

Elapsed: 00:00:00.22

→ Temps quasi optimal concernant les partitions, la jointure utilisée est construite à partir de deux partitions presque identiques optimales à la requête et au résultat attendu.

☐ **Etape 3:**

- **<u>Division de chaque partition de TIMELEVEL1 en deux partitions</u>**

- Partition part9508 (Première partition)

SQL> ALTER TABLE TIMELEVEL1 SPLIT PARTITION part9508 AT (199504) into (PARTITION part9509, PARTITION part95010);

Table altered.

- Partition part9605 (Deuxième partition)

SQL> ALTER TABLE TIMELEVEL1 SPLIT PARTITION part9605 AT (199601) into (PARTITION part9606, PARTITION part9607);

- **Q1 sur ACTVARS1**

SQL> select time_level, sum(DollarSales) from actvars1 where time_level = '199506' group by time_level;

TIME_L SUM(DOLLARSALES)
------ ----------
199506 62017446

Elapsed: 00:00:00.82

→ On remarque une réduction du temps d'exécution de la requête Q1 par rapport à l'étape 2 (Car on l'exécute pour la deuxième fois)

- **Q2 sur ACTVARS et TIMELEVEL1**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:04.21

→ On remarque une augmentation du temps d'exécution de cette requête par rapport à l'étape 2 (La jointure est très coûteuse car elle consiste à joindre la table ACTVARS avec la table timelevel1 après l'avoir partitionné)

- **Q2 sur ACTVARS1 et TIMELEVEL**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.29

→ On remarque une diminution du temps d'exécution de cette requête par rapport à l'étape 2 (La jointure est moins coûteuse, car elle consiste à joindre la table ACTVARS1 avec la table TIMELEVEL après partitionnement)

- **Q2 sur ACTVARS1 et TIMELEVEL1**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.21

→ On remarque une légère diminution du temps d'exécution de cette requête par rapport à l'étape 2 (La jointure est un petit peu moins coûteuse qu'à l'étape 2 grâce à la répartition de la table TIMELEVEL1)

☐ **Etape 4:**

Division de chaque partition de ACTVARS1 en deux partitions

SQL> ALTER TABLE ACTVARS1 SPLIT PARTITION part9508 AT (199504) into (PARTITION part9509,PARTITION part95010);

SQL> ALTER TABLE ACTVARS1 SPLIT PARTITION part9605 AT (199601) into (PARTITION part9606,PARTITION part9607);

- **Q1 sur ACTVARS1**

SQL> select time_level, sum(DollarSales) from actvars1 where time_level = '199506' group by time_level;

TIME_L SUM(DOLLARSALES)
------ ----------
199506 62017446

Elapsed: 00:00:02.24

→ On remarque une augmentation du temps d'exécution de cette requête par rapport à l'étape 3 et cela est dû au partitionnement des partitions de Actvars1.

- **Q2 sur ACTVARS et TIMELEVEL1**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)

---------- ----------
199506 62017446

Elapsed: 00:00:01.46

→ On remarque une diminution du temps d'exécution de cette requête par rapport à l'étape 3, car la jointure est moins coûteuse après la partition des partitions de actvars1.

- **Q2 sur ACTVARS1 et TIMELEVEL**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.33

→ La table ACTVARS1 n'est pas assez volumineuse pour bénéficier d'une amélioration de temps d'exécution.

- **Q2 sur ACTVARS1 et TIMELEVEL1**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.14

→ On remarque une diminution du temps d'exécution de cette requête par rapport à l'étape 2, la jointure est beaucoup moins coûteuse après le partitionnement des deux tables ACTVARS1 et TIMELEVEL1.

Question 3:

Partition avec TIMELEVEL2

SQL> CREATE TABLE TIMELEVEL2(Tid varchar(12) NOT NULL, year_level Number(4) NOT NULL,Quarter_level varchar(6) NOT NULL, month_level Number(2)) PARTITION BY HASH (Tid) PARTITIONS 4;

Partition avec ACTVARS2

SQL> CREATE TABLE ACTVARS2(Customer_level char(12) NOT NULL, Product_level char(12) NOT NULL, Channel_level char(12) NOT NULL, Time_level varchar(12) NOT NULL, UnitsSold FLOAT NOT NULL, DollarSales FLOAT NOT NULL, DollarCost FLOAT, FOREIGN KEY (Customer_level) REFERENCES custlevel(store_level), FOREIGN KEY (product_level) REFERENCES prodlevel(code_level), FOREIGN KEY (Channel_level) REFERENCES chanlevel(base_level), FOREIGN KEY (Time_level) REFERENCES timelevel(tid)) PARTITION BY HASH (Time_level) PARTITIONS 4;

- **Q1 sur ACTVARS2**

SQL> select time_level, sum(DollarSales) from actvars1 where time_level = '199506' group by time_level;

TIME_L SUM(DOLLARSALES)
------ ----------------
199506 62017446

Elapsed: 00:00:00.21

→ **Amélioration significative du temps d'éxécution car les partitions sont de tailles égales donc mieux distribuées.**

- **Q2 sur ACTVARS et TIMELEVEL2**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
------------ ----------------
199506 62017446

Elapsed: 00:00:02.29

→ **Dégradation légère du temps d'exécution car distribution aléatoire donc similaire à une table de base.**

- **Q2 sur ACTVARS2 et TIMELEVEL**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
------------ ----------------

199506 62017446

Elapsed: 00:00:00.21

→ **Grosse diminution du temps d'exécution car la distribution des partitions de ACTVARS1 est faite pour la jointure avec une autre table avec une clause d'égalités**

- **Q2 sur ACTVARS2 et TIMELEVEL2**

SQL> SELECT Tid, SUM(DollarSales) FROM ACTVARS1, TIMELEVEL1 WHERE Time_level = Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

TID SUM(DOLLARSALES)
---------- ----------
199506 62017446

Elapsed: 00:00:00.14

→ **Les deux tables sont partitionnées en HASH, on a donc une jointure sur prédicat d'égalité, cette méthode est faite pour cette situation.**

Question 4:

Conclusion :

On remarquera que les temps d'exécution ne sont pas toujours plus efficace après partitionnement, car le partitionnement n'est bénéfique que si les tables à partitionner sont volumineuses (la volumétrie des tables est calculé en nombre d'octets ou bien en page de données et non en nombre de lignes) et si le mode adapté à la requête est bien utilisé.

TP 3

Question 1:

- Exécution des 3 requêtes en l'absence de toutes vues matérialisées

Q1 : **Elapsed: 00:00:04.88**
Q2 : **Elapsed: 00:00:01.09**
Q3 : **Elapsed: 00:00:01.05**

Question 2:
A- Création des vues matérialisées :

- Création de la vue **Vq1** pour la requête 1

SQL> CREATE MATERIALIZED VIEW Vq1 BUILD IMMEDIATE REFRESH COMPLETE ON COMMIT AS SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL WHERE Time_level=Tid AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

Elapsed: 00:00:00.92

- Création de la vue **Vq2** pour la requête 2

SQL> CREATE MATERIALIZED VIEW Vq2 BUILD IMMEDIATE REFRESH COMPLETE ON COMMIT AS SELECT Code_level, SUM(DollarSales) FROM ACTVARS, TIMELEVEL, PRODLEVEL WHERE Time_level=Tid AND Product_level=Code_level AND Year_level=1995 AND Month_level=06 AND Family_level='T2KQVF62K7B6' GROUP BY Code_level;

Elapsed: 00:00:00.84

- Création de la vue **Vq3** pour la requête 3

CREATE MATERIALIZED VIEW Vq3 BUILD IMMEDIATE REFRESH COMPLETE ON COMMIT AS SELECT Tid, SUM(DollarSales) FROM ACTVARS, TIMELEVEL, CHANLEVEL WHERE Time_level=Tid AND Channel_level=Base_level AND Base_level='DU2BS2ODXAU9' AND Year_level=1995 AND Month_level=06 GROUP BY Tid;

Elapsed: 00:00:00.91

B - Verification si les requêtes prennent en compte des vues

Q1 : **Elapsed: 00:00:00.62**
Q2 : **Elapsed: 00:00:00.64**
Q3 : **Elapsed: 00:00:00.66**

→ **On remarque que les requêtes prennent bien en compte les nouvelles vues créées (le temps de réduction a considérablement diminué de 37% pour Q3 à 87% pour Q1).**

C- L'ajout d'un tuple dans la table de faits

```
SQL> INSERT INTO ACTVARS VALUES
('TERDKUD','KDIZMZEI','KDIPPE','199506','1','10','10');
```

Elapsed: 00:00:00.02

D- Relever au COMMIT les temps de mise à jour des vues.

```
SQL> commit;
```

Commit complete.

Elapsed: 00:00:02.05

E- Vérification de la mise à jour

```
SQL> select * from actvars where time_level = '199506' and customer_level = 'TERDKUD';

CUSTOMER_LEV PRODUCT_LEVE CHANNEL_LEVE TIME_LEVEL    UNITSSOLD
DOLLARSALES
------------ ------------ ------------ ---------- -----------
DOLLARCOST
----------
TERDKUD      KDIZMZEI         KDIPPE       199506               1               10
          10
```

Elapsed: 00:00:00.40

→ **Les mises à jour ont bien été prise en compte**

Question 3:

A- Création des vues et de leurs logs

SQL> CREATE MATERIALIZED VIEW LOG ON TIMELEVEL WITH ROWID;

Materialized view log created.

Elapsed: 00:00:00.38

SQL> CREATE MATERIALIZED VIEW LOG ON ACTVARS WITH ROWID;

Elapsed: 00:00:00.03

→ Création d'un journal de log recensant les modifications sur les tables TIMELEVEL et ACTVARS.

SQL> CREATE MATERIALIZED VIEW fast_vue BUILD IMMEDIATE REFRESH FAST ON COMMIT ENABLE QUERY REWRITE AS SELECT a.rowid as arowid, t.rowid as trowid, a.*, t.* FROM ACTVARS a inner join TIMELEVEL t on a.Time_level=t.Tid AND t.Year_level = 1995 AND t.Month_level=06;

Elapsed: 00:00:04.32

B-Temps d'execution des trois requêtes

Q1- **Elapsed: 00:00:00.63**
Q2- **Elapsed: 00:00:00.61**
Q3- **Elapsed: 00:00:00.48**

C- L'ajout d'un tuple dans la table de faits

SQL> INSERT INTO ACTVARS VALUES ('TERDKUD ','KDIZMZEI ',' KDIPPE ',' 199506 ',' 2' ,' 20','10');

Elapsed: 00:00:00.12

D- Relever au COMMIT les temps de mise à jour des vues.

SQL> commit;

Commit complete.

Elapsed: 00:00:02.50

E- Vérification de la mise à jour

SQL> select * from actvars where time_level = '199506' and customer_level = 'TERDKUD';

CUSTOMER_LEV PRODUCT_LEVE CHANNEL_LEVE TIME_LEVEL UNITSSOLD
DOLLARSALES
---------- ---------- ---------- ---------- ---------- ----------
DOLLARCOST

TERDKUD KDIZMZEI KDIPPE 199506 1 10
10

TERDKUD KDIZMZEI KDIPPE 199506 2 20
10

Elapsed: 00:00:00.47

→ **Les mises à jour ont bien été prise en compte**

Question 4:

Sur oracle pour la création d'une vue sur une jointure, il faut vérifier certaines restrictions pour ce type de rafraichissement. Donc, il faut créer des logs avec le rowid pour chaque table intervenant dans la jointure. Les temps d'exécution diminuent aussi surtout pour la question 2.

Question 5:

→ Nous pouvons construire des vues matérialisées de valeur sur des vues matérialisées de jointures. Par exemple, la somme des salaires de la vue matérialisée de la jointure d'ACTVARS et TIMELEVEL peut être une vue :

CREATE MATERIALIZED Vq5 BUILD IMMEDIATE REFRESH FAST ON COMMIT ENABLE QUERY REWRITE AS SELECT * FROM fast_vue;

TP4

Question 1:

- **Le temps d'exécution des deux requêtes Q1 et Q2**

Q1 **: Elapsed: 00:00:04.82**
Q2 **: Elapsed: 00:00:00.60**

- **Le temps d'exécution de l'ajout d'un tuple a la table de faits**

SQL> INSERT INTO ACTVARS VALUES
('TERDKUD','KDIZMZEI','KDIPPE','199606','3','20','10');

1 row created.

Elapsed: **00:00:00.11**

Question 2:

- **Installation d'un index B-tree sur Product_level de ACTVARS.**

SQL> CREATE INDEX index_btree ON ACTVARS (Product_level);

Index created.

Elapsed: **00:02:53.18**

- **Le temps d'exécution des deux requêtes après la création de l'index B-tree**

Q1: **Elapsed: 00:00:00.15**
Q2: **Elapsed: 00:00:00.10**

→ **Grâce à la création de l'index B-tree sur Product_level de ACTVARS le temps d'exécution de Q1 à considérablement diminué, mais aussi le temps d'exécution de la requête Q2 a diminué.**

- **Le temps d'exécution de l'ajout d'un tuple dans la table ACTVARS**

SQL> INSERT INTO ACTVARS VALUES ('TERDKUD ',' KDIZMZEI ','KDIPPE ','199606 ','4 ',' 20','10');

1 row created.

Elapsed: **00:00:00.58**

Question 3:

- **Destruction de l'index B-tree**

SQL> drop index index_btree;

Index dropped.

Elapsed: 00:00:09.12

→ L'index index_btree est supprimé avec succès

- **Création de l'index BITMAP**

SQL> CREATE BITMAP INDEX index_bitmap ON ACTVARS (Product_level);

Index created.

Elapsed: 00:00:27.23

- **Le temps d'exécution des deux requêtes après la création de l'index Bitmap**

Q1 : **Elapsed: 00:00:02.91**
Q2 : **Elapsed: 00:00:03.24**

→ **On remarque une augmentation du temps d'exécution en utilisant l'index Bitmap, car les valeurs de la colonne 'Product_Level' sont différentes.**

- **Le temps d'exécution de l'ajout d'un tuple dans la table ACTVARS**

SQL> INSERT INTO ACTVARS VALUES ('TERDKUD ',' KDIZMZEI ','KDIPPE ','199606 ','4 ',' 20','10');

1 row created.

Elapsed: 00:00:00.20

Question 4:

- **Destruction de l'index Bitmap**

SQL> drop index index_bitmap;

Index dropped.

Elapsed: 00:00:09.38

- **Création d'un index de jointure entre ACTVARS et PRODLEVEL.**

SQL> CREATE BITMAP INDEX XActvars ON ACTVARS (Product_level) FROM ACTVARS a, PRODLEVEL p where a.Product_level = p.code_level;

Index created.

Elapsed: 00:00:30.04

- **Le temps d'exécution des deux requêtes après la création de l'index de jointures.**

Q1:**Elapsed: 00:00:04.90**
Q2:**Elapsed: 00:00:00.11**

→ **On remarque qu'Il n'y a plus de temps à cause de la jointure.**

- **Le temps d'exécution de l'ajout d'un tuple dans la tacle ACTVARS**

SQL> INSERT INTO ACTVARS VALUES ('TERDKUD ',' KDIZMZEI ','KDIPPE ','199606 ','4 ',' 20','10');

1 row created.
Elapsed: **00:00:00.86**

Question 5:

- **Description de l'index de jointure de la question 4**

SQL> drop index XActvars;

Index dropped.

Elapsed: 00:00:10.45

- **Création d'une vue matérialisant les deux jointures de la requête Q2 avec la sélection sur Year_level et Month_level.**

Comme on avait déjà créé un log pour les tables ACTVARS et TIMELEVEL il va falloir faire la même chose pour la table PRODLEVEL aussi.

SQL> Create materialized view log on PRODLEVEL with rowid;

Materialized view log created.

Elapsed: 00:00:00.46

SQL> CREATE MATERIALIZED VIEW join_vue BUILD IMMEDIATE REFRESH FAST ON COMMIT ENABLE QUERY REWRITE AS SELECT a.rowid as arowid, t.rowid as trowid, p.rowid as prowid, a.*, t.*,p.* FROM ACTVARS a, TIMELEVEL t, PRODLEVEL p WHERE a.Time_level = t.Tid AND a.Product_level = p.Code_level AND t.Year_level = 1995 AND t.Month_level = 06;

Materialized view created.

Elapsed: 00:00:08.09

- **Création d'un index sur la vue qu'on vient de créer**

SQL> CREATE INDEX index_btree ON join_vue (Product_level);

Index created.

Elapsed: 00:00:06.07

- **Le temps d'exécution des deux requêtes après la création d'un index sur la vue**

Q1 : **Elapsed: 00:00:01.10**
Q2 : **Elapsed: 00:00:00.01**

- **Le temps de mise à jour d'un tuple dans ACTVARS**

SQL> UPDATE ACTVARS SET CUSTOMER_LEVEL = 'HDJKSHS' WHERE TIME_LEVEL = '199606 ';

2 rows updated.

Elapsed: **00:00:03.39**

Question 6:

- **Le bilan :**

Index de jointure : Il permet d'augmenter la performance des jointures des tables,

Index B-tree : Procure de bonnes performances pour une large gamme de requêtes avec des restrictions et les performances ne se dégradent pas trop lorsque la taille de la table augmente (Il est utile pour optimiser l'accès à une colonne comportent de nombreuses valeurs distinctes et en particulier lorsque chaque valeur concerne un petit nombre d'enregistrement), par contre les index B-tree occupent beaucoup d'espace et prend beaucoup de temps pour la mise à jour.

Index Bitmap : les index bitmap sont plus utiles dans un environnement de l’entrepôt de données (Les index Bitmap sont destinés à l'indexation de colonnes qui comportent peu de valeurs distinctes et beaucoup d'enregistrements pour chacune de ces valeurs). Un index bitmap est plus petit qu’un index b-tree donc son temps d’exécution est inférieur par rapport à celui d’index b-tree.