UNIVERSITÉ CÔTE D'AZUR

MIAGE - MASTER 1

Projet de recherche sur les SGBD du marché.

VALENTIN BORDY, TIMMY DAUMAS, MATHIEU BIRGER, JEAN-BAPTISTE LOGNON, ALAEDINE KAROUIA

FSGBD - **Monsieur Mopolo, Monsieur Gally.**

Juin-2021.

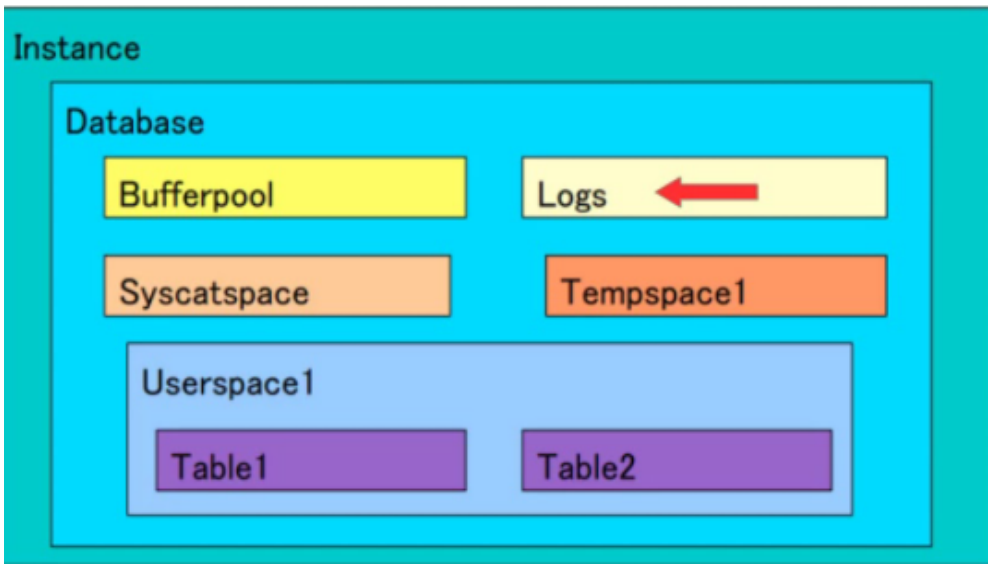
Identification du SGBD

Les informations qui permettent d'identifier le SGBD que nous avons choisis sont les suivantes:

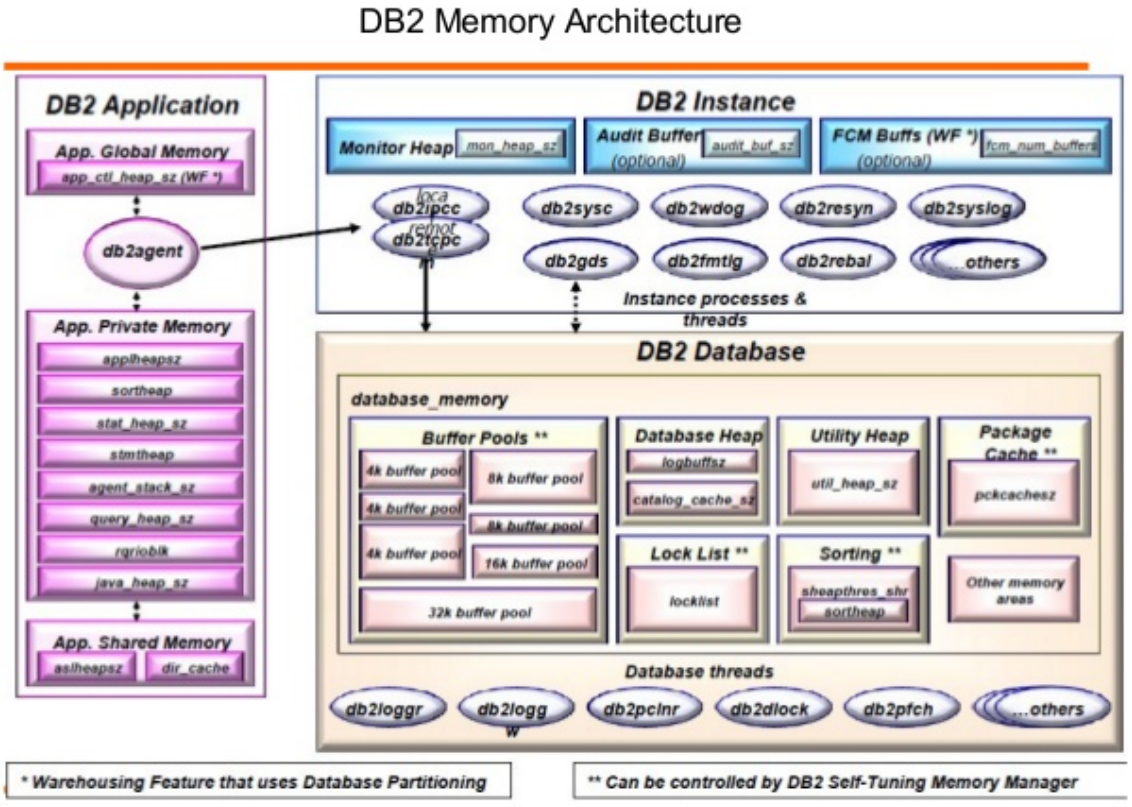
- Nom: Db2
- Date de parution:
    - La première version de Db2 (Db2 z/Os) est sortie en 1983 mais n'était utilisée que dans les mainframe d'IBM (System z).
    - La première version "grand public" (Db2 LUW) prévus pour Linux/Unix/Windows est quant à elle sortie en 1993.
- Créateur: Le créateur de Db2 est la société IBM.
- Propriétaire: IBM est toujours l'actuel propriétaire du SGBD.
- Modèle de données supportés: Db2 est en mesure de supporter plusieurs modèles de données:
    - Modèle relationnel (initialement)
    - Objet-relationnel (par extension)
    - Non relationnel comme JSON et XML (par extension)

Architecture fonctionnelles du SGBD

Dessins d'architecture



Les caches mémoires et leurs rôles



Les processus gravitant autour du cache mémoire et leur rôle

Gestion des connexions

Regroupement de connexions

Établir et fermer une connexion est une procédure très consommatrice en énergie et en ressources, qui peut impacter et avoir des répercussions sur les performances du serveur . Afin de réduire l'utilisation de traitement, le serveur DB2 Connect utilise le regroupement de connexions afin de gérer les connexions ouvertes à la base de données dans un pool facile d'accès.

Concentrateur de connexions

Le concentrateur de connexion réduit le nombre de ressources requises sur les serveurs de base de données DB2 for z/OS afin de prendre en charge un grand nombre de postes de travail et d'utilisateurs Web. Cette fonction peut accroître de manière très significative l'extensibilité de votre solution DB2 for z/OS et DB2 Connect tout en assurant une tolérance des pannes et un équilibrage de charge des transactions dans les environnements DB2 for z/OS avec partage de données.

Processus d’exécution des requêtes

Pour émettre une requête on utilise le SQL:

Select...
From...
Where…

Qui va donc se diriger dans la table qui nous intéresse pour y chercher ce que l’on souhaite tout en respectant un ensemble de contraintes données.

Le dictionnaire de données

- **SYSCATALOGS** contient une ligne pour chaque base de données relationnelle à laquelle un utilisateur peut se connecter.
- **SYSCHKCST** contient une ligne pour chaque contrainte de contrôle dans le schéma SQL.

- **SYSCOLAUTH** contient une ligne pour chaque privilège accordé sur une colonne.
    - Notez que cette vue de catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser une colonne, car le privilège d'utilisation d'une colonne peut être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ). De plus, le privilège d'utilisation d'une colonne s'acquiert également par le biais de privilèges accordés sur la table.
- **SYSCOLUMNS** contient une ligne pour chaque colonne de chaque table et vue du schéma SQL (y compris les colonnes du catalogue SQL).
- **SYSCOLUMNS2** contient une ligne pour chaque colonne de chaque table et vue du schéma SQL (y compris les colonnes du catalogue SQL).
- **SYSCOLUMNSTAT** contient une ligne pour chaque colonne dans une partition de table ou un membre de table qui a une collection de statistiques de colonnes. Si la table est une table distribuée, les partitions qui résident sur d'autres nœuds de la base de données ne sont pas contenues dans cette vue du catalogue.
- **SYSCONTROLS** contient une ligne pour chaque autorisation de ligne ou masque de colonne défini par les instructions CREATE PERMISSION ou CREATE MASK.
- **SYSCONTROLSDEP** enregistre les dépendances des autorisations de lignes et des masques de colonnes.
- **SYSCST** contient une ligne pour chaque contrainte du schéma SQL.
- **SYSCSTCOL** enregistre les colonnes sur lesquelles des contraintes sont définies. Il y a une ligne pour chaque colonne dans une contrainte unique, une clé primaire et une contrainte de contrôle, ainsi que pour les colonnes de référence d'une contrainte référentielle.
- **SYSCSTDEP** enregistre les tables sur lesquelles des contraintes sont définies.
- **SYSFIELDS** contient une ligne pour chaque colonne qui a une procédure de champ.
- **SYSFUNCS** contient une ligne pour chaque fonction créée par l'instruction CREATE FUNCTION.

- **SYSHISTORYTABLES** contient une ligne pour chaque table d'historique, que cette table fasse partie ou non d'une relation de versionnage établie.
- **SYSINDEXES** contient une ligne pour chaque index du schéma SQL créé à l'aide de l'instruction SQL CREATE INDEX, y compris les index du catalogue SQL.
- **SYSINDEXSTAT** contient une ligne pour chaque partition d'index SQL.
- **SYSJARCONTENTS** contient une ligne pour chaque classe définie par un jarid dans le schéma SQL.
- **SYSJAROBJECTS** contient une ligne pour chaque jarid du schéma SQL.
- **SYSKEYCST** contient une ou plusieurs lignes pour chaque UNIQUE KEY, PRIMARY KEY ou FOREIGN KEY du schéma SQL. Il y a une ligne pour chaque colonne dans chaque contrainte de clé unique ou primaire et les colonnes de référence d'une contrainte référentielle.
- **SYSKEYS** contient une ligne pour chaque colonne d'un index dans le schéma SQL, y compris les clés des index du catalogue SQL.
- **SYSMQTSTAT** contient une ligne pour chaque partition de table matérialisée.
- **SYSPACKAGE** contient une ligne pour chaque package SQL dans le schéma SQL.
- **SYSPACKAGEAUTH** contient une ligne pour chaque privilège accordé sur un package.
    - Notez que cette vue du catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser un paquet, car le privilège d'utilisation d'un paquet peut être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSPACKAGESTAT** contient une ligne pour chaque paquetage SQL dans le schéma SQL.
- **SYSPACKAGESTMTSTAT** contient une ligne pour chaque instruction SQL dans chaque paquetage SQL.
- **SYSPARMS** contient une ligne pour chaque paramètre d'une procédure créée par l'instruction CREATE PROCEDURE ou d'une fonction créée par l'instruction CREATE FUNCTION. Le résultat d'une fonction scalaire et les colonnes de résultat d'une fonction de table sont également retournés.
- **SYSPARTITIONDISK** contient une ligne pour chaque unité de disque utilisée pour stocker les données de chaque partition de table ou membre de table. Si

la table est une table distribuée, les partitions qui résident sur d'autres nœuds de base de données ne sont pas contenues dans cette vue de catalogue. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.

- **SYSPARTITIONINDEXDISK** contient une ligne pour chaque unité de disque utilisée pour stocker les données d'index de chaque partition de table ou membre de table. Si l'index est un index distribué, les partitions qui résident sur d'autres nœuds de base de données ne sont pas contenues dans cette vue de catalogue. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.
- **SYSPARTITIONINDEXES** contient une ligne pour chaque index construit sur une partition de table ou un membre de table. Si la table est une table distribuée, les index sur les partitions qui résident sur d'autres noeuds de base de données ne sont pas contenus dans cette vue de catalogue. Ils sont contenus dans les vues de catalogue des autres nœuds de base de données.
- **SYSPARTITIONINDEXSTAT** contient une ligne pour chaque index construit sur une partition de table ou un membre de table. Les index qui partagent l'arbre binaire d'un autre index ne sont pas inclus. Si la table est une table distribuée, les index sur les partitions qui résident sur d'autres nœuds de la base de données ne sont pas contenus dans cette vue de catalogue. Ils sont contenus dans les vues de catalogue des autres nœuds de base de données.
- **SYSPARTITIONMQTS** contient une ligne pour chaque table matérialisée construite sur une partition de table ou un membre de table. Si la table est une table distribuée, les tables matérialisées sur les partitions qui résident sur d'autres nœuds de base de données ne sont pas contenues dans cette vue de catalogue. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.
- **SYSPARTITIONSTAT** contient une ligne pour chaque partition de table ou membre de table. Si la table est une table distribuée, les partitions qui résident sur d'autres nœuds de base de données ne sont pas contenues dans cette vue de catalogue. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.
- **SYSPERIODS** contient une ligne pour chaque période définie pour une table temporelle dans un schéma SQL.

- **SYSPROCS** contient une ligne pour chaque procédure créée par l'instruction CREATE PROCEDURE.
- **SYSPROGRAMSTAT** contient une ligne pour chaque programme, programme de service et module qui contient des instructions SQL.
- **SYSPROGRAMSTMTSTAT** contient une ligne pour chaque instruction SQL intégrée dans un programme, un module ou un programme de service.
- **SYSREFCST** contient une ligne pour chaque clé étrangère dans le schéma SQL.
- **SYSROUTINEAUTH** contient une ligne pour chaque privilège accordé sur une routine.
    - Notez que cette vue de catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser une routine, car le privilège d'utilisation d'une routine pourrait être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSROUTINEDEP** enregistre les dépendances des routines.
- **SYSROUTINES** contient une ligne pour chaque procédure créée par l'instruction CREATE PROCEDURE et chaque fonction créée par l'instruction CREATE FUNCTION.
- **SYSSCHEMAAUTH** contient une ligne pour chaque privilège accordé sur un schéma.
    - Notez que cette vue du catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser un schéma, car le privilège d'utilisation d'un schéma peut être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSSCHEMAS** contient une ligne pour chaque schéma de la base de données relationnelle.
- **SYSSEQUENCEAUTH** contient une ligne pour chaque privilège accordé sur une séquence.
    - Notez que cette vue de catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser une séquence, car le privilège d'utilisation d'une séquence pourrait être acquis par le biais

d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).

- **SYSSEQUENCES** contient une ligne pour chaque objet séquence du schéma SQL.
- **SYSTABAUTH** contient une ligne pour chaque privilège accordé sur une table ou une vue.
    - Notez que cette vue de catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser une table ou une vue, car le privilège d'utilisation d'une table ou d'une vue pourrait être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSTABLEDEP** enregistre les dépendances des tables de requêtes matérialisées.
- **SYSTABLEINDEXSTAT** contient une ligne pour chaque index qui a au moins une partition ou un membre construit sur une table. Si l'index est sur plus d'une partition ou membre, les statistiques incluent toutes ces partitions et membres. Si la table est une table distribuée, les partitions qui résident sur d'autres nœuds de la base de données ne sont pas incluses. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.
- **SYSTABLES** contient une ligne pour chaque table, vue ou alias du schéma SQL, y compris les tables et les vues du catalogue SQL.
- **SYSTABLESTAT** contient une ligne pour chaque table qui a au moins une partition ou un membre. Si la table a plus d'une partition ou d'un membre, les statistiques incluent toutes les partitions et tous les membres. Si la table est une table distribuée, les partitions qui résident sur d'autres nœuds de base de données ne sont pas incluses. Elles sont contenues dans les vues de catalogue des autres nœuds de base de données.
- **SYSTRIGCOL** contient une ligne pour chaque colonne référencée implicitement ou explicitement dans la clause WHEN ou les instructions SQL déclenchées d'un déclencheur.
- **SYSTRIGDEP** contient une ligne pour chaque objet référencé dans la clause WHEN ou les instructions SQL déclenchées d'un déclencheur.

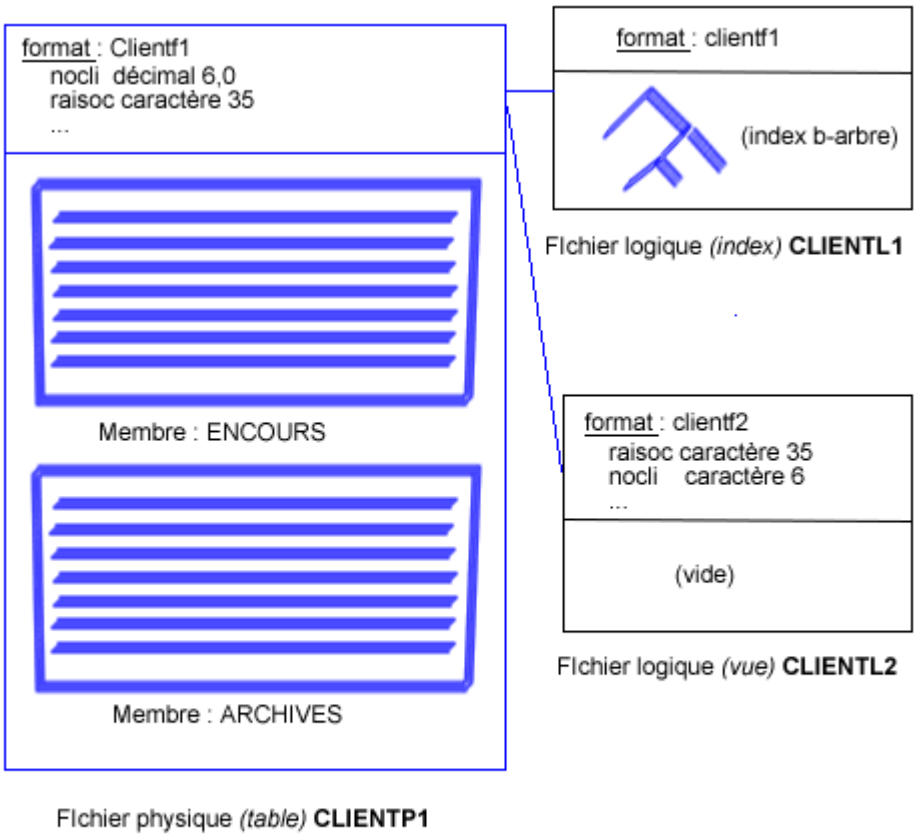
- **SYSTRIGGERS** contient une ligne pour chaque déclencheur dans un schéma SQL.
- **SYSTRIGUPD** contient une ligne pour chaque colonne identifiée dans la liste des colonnes UPDATE, le cas échéant.
- **SYSTYPES** contient une ligne pour chaque type de données intégré et chaque type distinct et type de tableau créé par l'instruction CREATE TYPE.
- **SYSUDTAUTH** contient une ligne pour chaque privilège accordé sur un type.
    - Notez que cette vue du catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser un type, car le privilège d'utilisation d'un type peut être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSVARIABLEAUTH** contient une ligne pour chaque privilège accordé sur une variable globale.
    - Notez que cette vue de catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à utiliser une variable globale, car le privilège d'utilisation d'une variable globale peut être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **SYSVARIABLEDEP** enregistre les dépendances des variables.
- **SYSVARIABLES** contient une ligne pour chaque variable globale.
- **SYSVIEWDEP** enregistre les dépendances des vues sur les tables, y compris les vues du catalogue SQL.
- **SYSVIEWS** contient une ligne pour chaque vue du schéma SQL, y compris les vues du catalogue SQL.
- **SYSXSROBJECTAUTH** contient une ligne pour chaque privilège accordé sur un schéma XML.
    - Notez que cette vue du catalogue ne peut pas être utilisée pour déterminer si un utilisateur est autorisé à un schéma XML car le privilège d'utiliser un schéma XML pourrait être acquis par le biais d'un profil utilisateur de groupe ou d'une autorité spéciale (telle que *ALLOBJ).
- **XSRANNOTATIONINFO** contient une ligne pour chaque annotation dans un schéma XML afin d'enregistrer les informations de la table et des colonnes concernant l'annotation.

- **XSROBJECTCOMPONENTS** contient une ligne pour chaque composant (document) d'un schéma XML.
- **XSROBJECTHIERARCHIES** contient une ligne pour chaque composant (document) d'un schéma XML afin d'enregistrer la relation hiérarchique entre les documents du schéma XML.
- **XSROBJECTS** contient une ligne pour chaque schéma XML enregistré.

Organisation physique du SGBD

Chaque bibliothèque contient des fichiers base de données autonomes :
Les fichiers physiques ou tables SQL contiennent les données (enregistrements).
Les fichiers logiques proposent un accès particulier aux données :
un critère de tri particulier (index).
un format d'enregistrement particulier (sélection de lignes, formatage des colonnes, jointure)

un fichier physique se présente sous la forme:



Fichiers physiques et logiques sur un AS/400

Organisation logique des données

Un Fichier logique est un objet de type *FILE. Il possède une description et 1 ou plusieurs membres.

CES MEMBRES NE CONTIENNENT PAS DE DONNÉES,

ils ne possèdent qu'un chemin d'accès qui pointe sur les données d'un ou plusieurs fichiers physiques.

- historiquement : index sous forme d'arbre binaire [bi-tree].

- il est possible de construire des index EVI

LES FICHIERS LOGIQUES VONT NOUS PERMETTRE :

- d'avoir une clé différente du fichier physique, donc d'accéder dans un nouvel ordre aux enregistrements

- de regrouper les données de plusieurs fichiers physiques

- de sélectionner une partie des enregistrements d'1 ou plusieurs fichiers physiques.

Une Clé différente du fichier physique

Prenons l'exemple du fichier physique CLIENT1 ci-dessous.

```
A**********************
A*   FICHIER CLIENTP1
A**********************
A                                   REF(REPERTP1)
A          R CLIENTF1               TEXT('FORMAT DU FICHIER CLIENTP1')
A            NOCLI     R
A            NOM01     R
A            ADR01     R
A            ADR02     R
A            CODPS     R
A            CODEX     R
A            CRLIM     R
A            TOTAL     R
A          K NOCLI
```

Gestion de la concurrence d’accès

Pour assurer le contrôle de la concurrence et empêcher l'accès incontrôlé aux données, le gestionnaire de base de données place des verrous sur les pools de mémoire tampon, les tables, les partitions de données, les blocs de table ou les lignes de table.

Un verrou associe une ressource du gestionnaire de base de données à une application, appelée le propriétaire du verrou, afin de contrôler la façon dont les autres applications accèdent à la même ressource.

Le gestionnaire de base de données utilise le verrouillage au niveau de la ligne ou de la table, selon le cas, en fonction de ce qui suit :
Le niveau d'isolation spécifié au moment de la précompilation ou lorsqu'une application est liée à la base de données. Le niveau d'isolement peut être l'un des suivants :
Lecture non engagée (UR)
Stabilité du curseur (CS)
Stabilité de lecture (RS)
Lecture répétable (RR)
Les différents niveaux d'isolement sont utilisés pour contrôler l'accès aux données non engagées, empêcher les mises à jour perdues, autoriser les lectures non répétables de données et empêcher les lectures fantômes. Pour minimiser l'impact sur les performances, utilisez le niveau d'isolation minimum qui répond aux besoins de votre application.
Le plan d'accès sélectionné par l'optimiseur. Les balayages de table, les balayages d'index et les autres méthodes d'accès aux données nécessitent chacun différents types d'accès aux données.
L'attribut LOCKSIZE de la table. La clause LOCKSIZE de l'instruction ALTER TABLE indique la granularité des verrous utilisés lors de l'accès à la table. Les choix sont les suivants : ROW pour les verrous de ligne, TABLE pour les verrous de table, ou BLOCKINSERT pour

les verrous de bloc sur les tables de clustering multidimensionnel (MDC) uniquement. Lorsque la clause BLOCKINSERT est utilisée sur une table MDC, le verrouillage au niveau des lignes est effectué, sauf pendant une opération d'insertion, où le verrouillage au niveau des blocs est effectué à la place. Utilisez l'instruction ALTER TABLE...LOCKSIZE BLOCKINSERT pour les tables MDVerrous et contrôle de la concurrence
Pour assurer le contrôle de la concurrence et empêcher l'accès incontrôlé aux données, le gestionnaire de base de données place des verrous sur les pools de mémoire tampon, les tables, les partitions de données, les blocs de table ou les lignes de table.

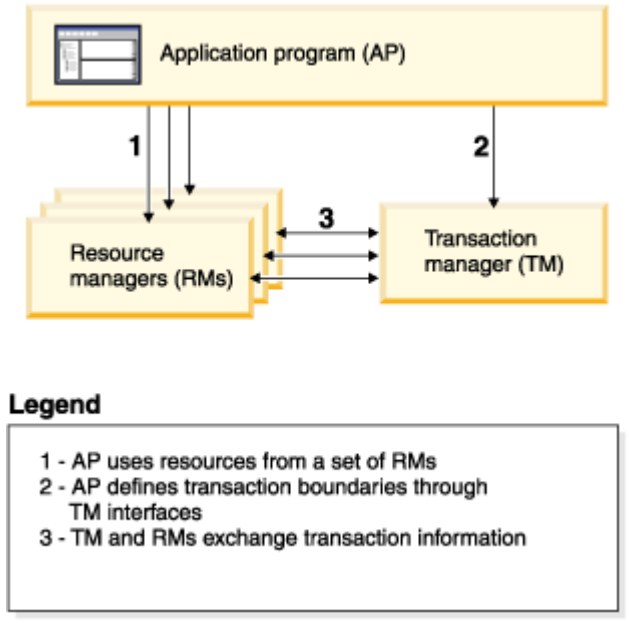
C lorsque des transactions doivent effectuer des insertions importantes dans des cellules disjointes. Utilisez l'instruction ALTER TABLE...LOCKSIZE TABLE pour les tables en lecture seule. Cela réduit le nombre de verrous nécessaires à l'activité de la base de données. Pour les tables partitionnées, les verrous de table sont d'abord acquis, puis les verrous de partition de données sont acquis, en fonction des données auxquelles on accède.
La quantité de mémoire consacrée au verrouillage, qui est contrôlée par le paramètre de configuration de la base de données locklist. Si la liste des verrous se remplit, les performances peuvent se dégrader en raison des escalades de verrous et de la réduction de la concurrence entre les objets partagés de la base de données. Si les escalades de verrous sont fréquentes, augmentez la valeur de locklist, maxlocks ou les deux. Pour réduire le nombre de verrous détenus à un moment donné, assurez-vous que les transactions sont validées fréquemment.
Un verrou de pool de mémoire tampon (exclusif) est défini chaque fois qu'un pool de mémoire tampon est créé, modifié ou abandonné. Vous pouvez rencontrer ce type de verrou lors de la collecte de données de surveillance du système. Le nom du verrou est l'identifiant (ID) du pool de mémoire tampon lui-même.

En général, le verrouillage au niveau de la ligne est utilisé sauf si l'une des conditions suivantes est vraie :
Le niveau d'isolement est la lecture non engagée
Le niveau d'isolement est la lecture répétée et le plan d'accès nécessite une analyse sans prédicats de plage d'index.
L'attribut LOCKSIZE de la table est TABLE.
La liste des verrous se remplit, ce qui entraîne une escalade des verrous.
Un verrou de table explicite a été acquis par l'intermédiaire de l'instruction LOCK TABLE, ce qui empêche les processus d'application concurrents de modifier ou d'utiliser une table.
Dans le cas d'une table MDC, le verrouillage au niveau du bloc est utilisé au lieu du verrouillage au niveau de la ligne lorsque :
l'attribut LOCKSIZE de la table est BLOCKINSERT
Le niveau d'isolation est repeatable read et le plan d'accès implique des prédicats.
Une opération de mise à jour ou de suppression recherchée implique des prédicats sur les colonnes de dimension uniquement.
La durée du verrouillage des lignes varie en fonction du niveau d'isolement utilisé :
Scans UR : Aucun verrouillage de ligne n'est maintenu à moins que les données de la ligne ne soient modifiées.

Scans CS : Les verrous de rangée ne sont généralement maintenus que lorsque le curseur est positionné sur la rangée. Notez que dans certains cas, les verrous peuvent ne pas être maintenus du tout pendant un scan CS.
Balayages RS : Les verrous de ligne qualifiée ne sont maintenus que pendant la durée de la transaction.
Scans RR : Tous les verrous de ligne sont maintenus pendant toute la durée de la transaction.

Gestion des transactions distribuées

Fonctionnement général



Application Program (AP)

Il s'agit du programme souhaitant récupérer et pousser des données dans la base de données. Il utilise pour cela le Transaction Manager et des Resource Managers, souvent à travers de la précompilation par Db2 des requêtes SQL ou alors directement par le développeur de l'API. Par exemple, un programme CICS de gestion de comptes.

Transaction Manager (TM)

Le Db2 Transaction Manager (TM) assigne des identifiants à chaque transaction, surveille leurs progrès et prend la responsabilité de la complétion et de l'échec des transactions. Il enregistre toutes les informations des transactions dans sa base de données TM.

De plus, le Transaction Manager fournit des fonctions qui peuvent être utilisées pour coordonner la mise à jour de plusieurs bases de données dans une seule unité de travail. Le client coordonne automatiquement l'unité de travail, et utilise la base de données du Transaction Manager pour enregistrer chaque transaction et voir son statut de complétion.

Dans un environnement avec des unités de travail distribuées (Distributed Unit Of Work; DUOW), les transactions qui ont été préparées, mais pas encore commit sont gardées dans une liste liées, la resync list. Cette liste est utilisée pour se remettre après un échec d'un commit ou d'un rollback due à la connexion réseau, au hardware, ou au software.

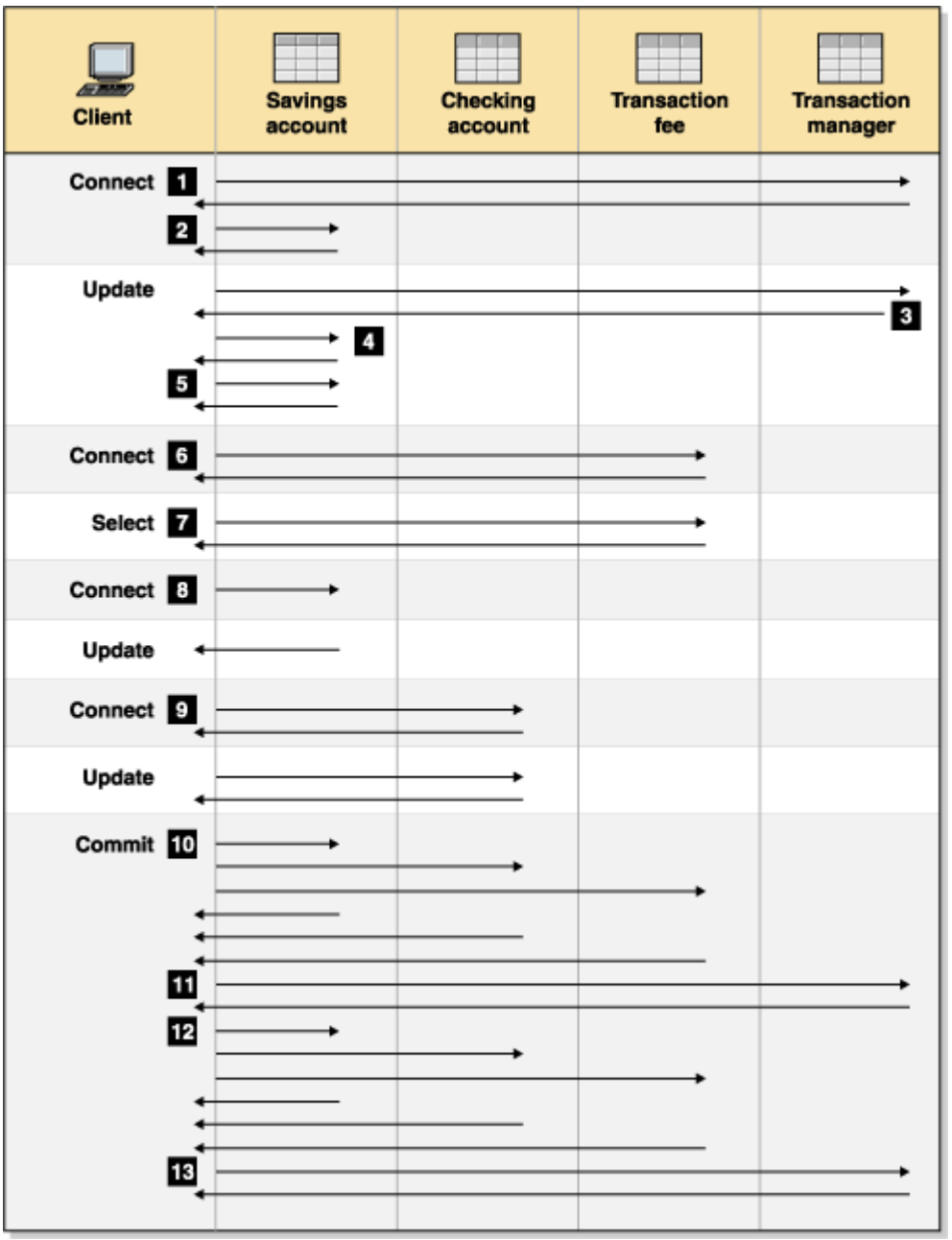
La resync list est stockée dans le resync heap, qui est dans la mémoire partagée du serveur.

Chaque entrée dans la resync list est appelée une resync entry. Une entrée peut être une entrée TM (Transaction Manager) ou une entrée RM (Resource Manager). Chaque entrée TM est identifiée de manière unique par le XID de la transaction. Chaque entrée RM est identifiée de manière unique par le XID de la transaction et l'alias du RM.

Resource Managers (RMs)

Un Resource Manager (RM) est une base de données distincte identifiée par le Transaction Manager grâce à une chaîne de caractères xa_open.

Fonctionnement en détail avec un exemple



0

L’application est préparée pour le commit en deux phases. Ceci peut être accompli par des options de précompilation ou alors grâce à des configurations faites avec Db2 CLI.

1

Quand le client souhaite se connecter à la base de données SAVINGS_DB (pour accéder à la table SAVINGS_ACCOUNT), il doit d’abord se connecter à la base de données du Transaction Manager (TM). La base de données du TM retourne alors une réponse positive au client.

Le client peut également choisir de mettre le paramètre tm_database à 1ST_CONN. A ce moment là, c'est SAVINGS_DB qui fera office de TM pour la durée de la session.

2

La connexion à la base de données SAVINGS_BD a lieu et est acceptée.

3

Le client commence la mise à jour de la table SAVINGS_ACCOUNT . Ceci commence l'unité de travail. La base de données TB répond alors au client en lui fournissant une transaction ID pour l'unité de travail.

Il est important de noter que l'enregistrement d'une unité de travail est fait au moment où la première instruction SQL de l'unité de travail est exécutée, pas pendant l'établissement de la connexion.

4

Après avoir reçu une transaction ID, le client enregistre l'unité de travail avec la base de données contenant la table SAVINGS_ACCOUNT (ici SAVINGS_DB). Une réponse est envoyée au client pour lui indiquer que l'unité de travail a bien été enregistrée.

5

Les instructions SQL sont effectuées dans la base de données SAVINGS_DB d'une façon normale. La réponse à chacune de ses instructions est retourné dans le SQLCA lorsqu'on travaille avec des instructions SQL utilisées dans un programme.

6

La transaction ID est enregistrée dans la base de données FEE_DB (qui contient la table TRANSACTION_FEE), pendant le premier accès à la base de données dans l'unité de travail.

7

Toutes les instructions SQL effectuées sur la base de données FEE_DB sont exécutées normalement.

8

Des instructions SQL supplémentaires peuvent être exécutées sur la base de données SAVINGS_DB en effectuant une connexion appropriée. Comme l'unité de travail a déjà été enregistrée dans la base de données pendant l'étape 4, le client ne doit pas s'enregistrer à nouveau.

9

Se connecter et utiliser la base de données CHECKING_DB (et la table CHECKING_ACCOUNT) se fait de la même manière et avec les mêmes règles que celles décrites pendant les étapes 6 et 7.

10

Quand le client demande à ce que l'unité de travail soit commit, un message *prepare* est envoyé à toutes les bases de données participant à l'unité de travail. Chaque base de données écrit "PREPARED" dans ces log files et répond au client.

11

Après que le client ait reçu une réponse positive de la part de toutes les bases de données, il envoie un message à la base de données TM, l'informant que l'unité de travail est prête à être commit (PREPARED).

La base de données TM écrit alors "PREPARED" dans son fichier de log et renvoie une réponse au client, l'informant que la seconde phase de commit peut commencer.

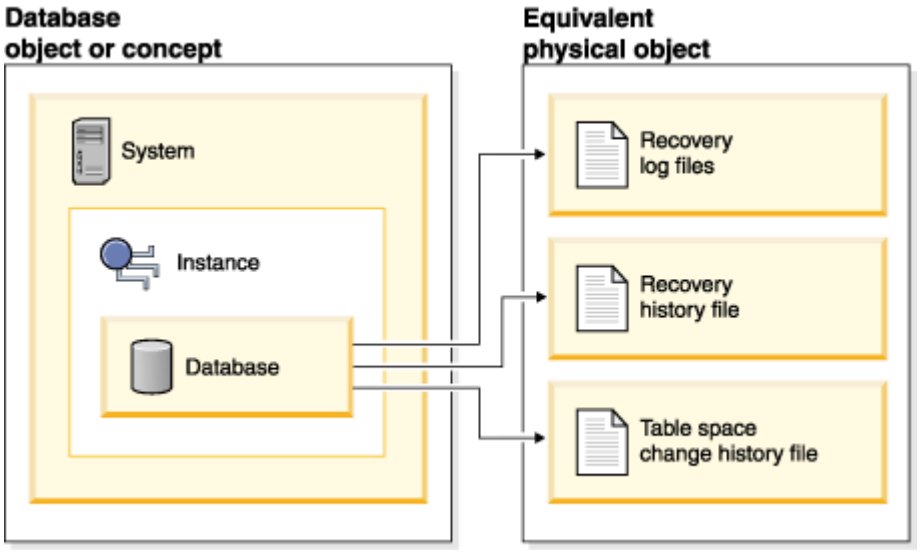
12

Pendant la seconde phase du processus de commit, chaque base de donnée écrit "COMMITTED" dans son fichier de log et retire les verrous mis en place pour cette unité de travail. Quand une base de données à finit de commit les changements, elle envoie une réponse au client.

13

Après que le client ait reçu une réponse positive de toutes les bases de données participantes à l'unité de travail, il envoie un message à la base de données du Transaction Manager, l'informant que l'unité de travail a été complétée. Le Transaction Manager écrit alors "COMMITTED" dans son fichier de log, indiquant que l'unité de travail est complète, puis répond au client en indiquant qu'il a terminé.

Gestion de la reprise sur panne



Crash Recovery

Db2 utilise des logs files afin de s'assurer de la reprise sur panne. Il existe des Recovery log files, des recovery history file et des table space change history file. Chacun de ces fichiers permets de stocker les informations nécessaires à l'annulation des modifications des données faites par des transactions non complétées au moment du crash.

Grâce au logs files créés lors de chaque transaction pour la base de données TM et les base de données RMs, il est possible de faire des crash recovery lors d'une panne. Il est ainsi possible de rollback / recommencer les transactions non complétées au moment du crash, mais cela ne permet pas de récupérer les données qui auraient été endommagées par un problème de disque dur par exemple.

Version Recovery

Db2 encourage également le backup des données. Ainsi, il devient possible d'effectuer un versionning des données, et au moment d'une panne, il est possible de faire un retour en arrière vers une version backup des données. Il est ainsi possible de récupérer des données perdues.

Bien sûr, il est de la tâche du développeur de vérifier que des backups soient bien réalisés. Il est important de définir la durée entre deux backups et le nombre de backups à conserver car cela va dépendre de la criticité des données ainsi que de l'espace disque disponible pour effectuer ces backups.

Rollforward Recovery

Après avoir effectué une Version Recovery, il est peut être nécessaire de rajouter des commit ayant eu lieu depuis la dernière backup. Db2 propose alors des mécanismes de Rollforward Recovery permettant d'effectuer les commits manquant de la version afin de pouvoir retourner à un état plus récent de la base de données.

Geographically dispersed Db2 pureScale Cluster (GDPC)

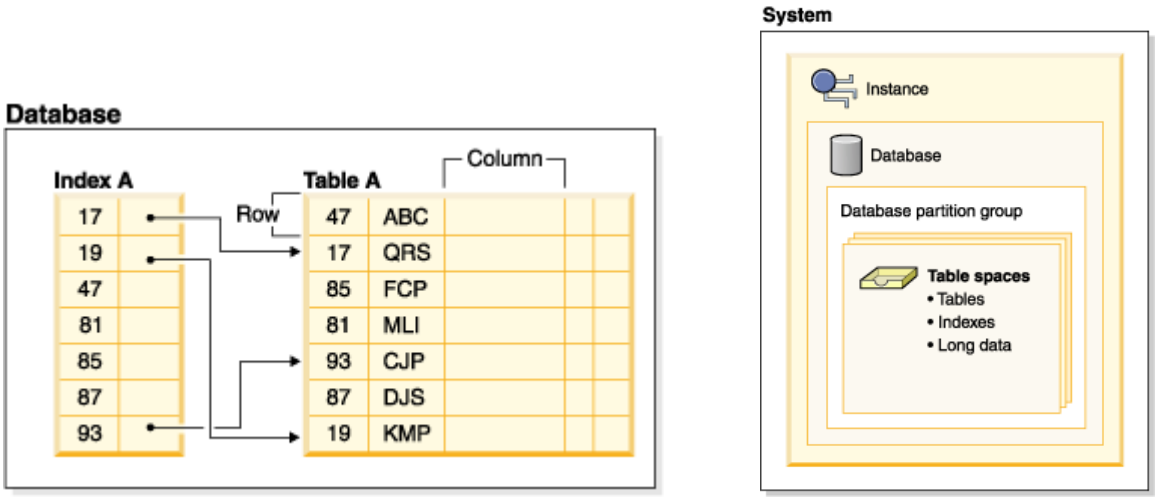
Dans le cas d'un GDPC, Db2 fournit des moyens de reprise sur panne rapide et pratique. Ainsi, lors d'une panne d'un site, les systèmes de Workload Balancing et d'Automatic Client Reroute vont permettre d'envoyer le trafic aux autres sites.

Lorsque le site va revenir en ligne, il pourra alors communiquer avec les autres sites et récupérer les données perdues à cause du crash.

Bien sûr, tout ceci peut être surveillé grâce à des commandes tels que *db2cluster -verify* et *db2instance -list* ou bien *db2cluster -list -alert* afin d'afficher les alertes des différents clusters.

Ainsi, lorsque le GDPC est répliqué, la panne peut être résolue automatiquement, avec comme coup le retard des transactions de mise à jour le temps de la résolution de la panne.

Techniques d'indexation



Un index est un ensemble de pointeurs qui sont ordonnés logiquement par la valeur d'une ou plusieurs clés. Ils permettent d'optimiser la vitesse du SGBD car il est généralement plus rapide d'accéder aux données grâce à un index. De plus, ils peuvent aussi permettre l'unicité des lignes de la table, car une table avec un index assurera que chaque ligne a une valeur unique pour la colonne de l'index.

Dans le cas de Db2, il est possible de créer 4 types d'index:

- Des index pour assurer l'unicité des lignes de la table,
- Des index bidirectionnel pour parcourir la table d'un sens ou de l'autre,
- Des clustered index qui permettent d'accélérer les requêtes,
- Des Expression-based index qui permettent d'évaluer efficacement les queries basées sur ces index.

Il est possible de créer des index assez simplement:

```
CREATE UNIQUE INDEX EMP_IX
ON EMPLOYEE(EMPNO)
INCLUDE(FIRSTNAME, JOB)
```

Ici, l'exemple de la création d'un index unique portant le nom EMP_IX sur la table EMPLOYEE avec comme clés EMPNO, FIRSTNAME et JOB. L'utilisation du Include permet de spécifier que l'index se base sur les clés FIRSTNAME et JOB, mais ne force pas l'unicité sur ces valeurs, n'utilisant que EMPNO pour cela.

Ainsi, les requêtes SELECT EMPNO, FIRSTNAME, JOB FROM EMPLOYEE seront plus rapides grâce à l'utilisation de l'index.

Optimisation de requêtes

Optimizer

Db2 utilise un Optimizer pour optimiser automatiquement toutes les requêtes. Il va par exemple choisir l'ordre des requêtes à effectuer en interne afin de fournir un résultat plus rapide. Il choisira d'utiliser ou non les index, mais il est également capable de réécrire des morceaux de la requête pour transformer certaines parties en requêtes plus optimisées. Par exemple, une sous requête transformée en jointure.

Indexation des données

Afin d'optimiser les requêtes, il est utile d'utiliser des index. Mais ces index ne doivent pas être mal utilisés, car chaque index va augmenter le temps de mise à jour des données.

De plus, il vaut mieux utiliser des petites colonnes telles que INTEGER et TIMESTAMP plutôt que VARCHAR pour ces index afin de maximiser les performances.

Il faut également essayer d'utiliser un disque dur différent pour le stockage des index que celui du stockage des fichiers de la base de données, ainsi les requêtes seront encore plus rapides.

Il est possible de grandement améliorer les performances en utilisant la technique du Index-Only Access. Il s'agit de ne jamais utiliser la table afin de récupérer des données et de n'utiliser que l'index. Ainsi, si l'on veut récupérer une table de pilote avec leur nom/prénom/id, on pourra utiliser un index contenant le nom, le prénom et l'id. Lors de la requête, le SGBD n'aura pas à utiliser la table car toutes les valeurs sont déjà présente dans l'index, ce qui augmentera grandement la rapidité.

L'écriture des requêtes

L'écriture d'une requête est un paramètre non négligeable. Il est possible de mal écrire des requêtes de manière à ce que l'Optimizer ne puisse pas améliorer la requête.

Par exemple, il vaut mieux utiliser LIKE plutôt que substr dans une requête, car substr rend l'utilisation d'un Index impossible alors que LIKE le permet.

Il faut également mieux éviter les opérations arithmétiques dans la requête car cela ralentit considérablement le temps de traitement.

Il faut également éviter de convertir les données dans un autre type car cela va rajouter du temps de traitement non nécessaire.

Il vaut mieux utiliser l'opérateur UNION ALL comparé à UNION si l'on est sûr que les résultats seront identiques.

Il faut aussi faire attention au niveau de locking utilisé pour la requête, tout en étant sûr de spécifier le bon locking au bon moment pour bloquer au minimum l'utilisation des données.

Parfois, il est important de bien savoir choisir entre sous-requête et jointure. Il est parfois préférable d'effectuer une sous-requête qu'une jointure et inversement.