Il n'est pas aisé de se lancer dans un chantier d'optimisation, car le contexte n'est généralement pas simple. Pour ceux qui ont eu l'occasion d'aborder ce genre d'audit ils vous diront que c'est différent d'un client à l'autre. *Comme on dit, « il vaut mieux prévenir que guérir ».*

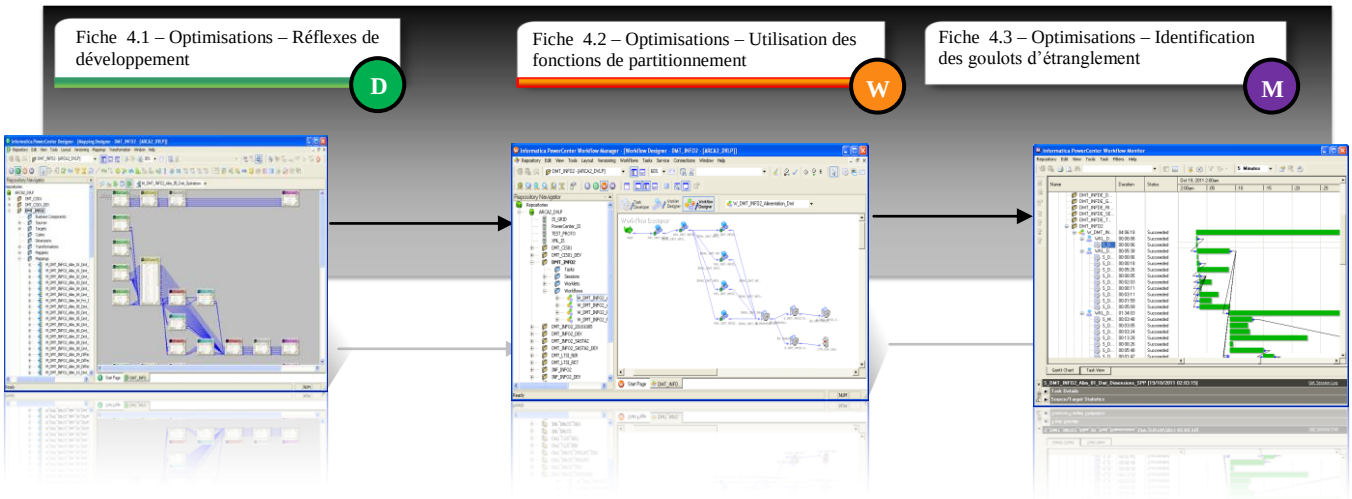
Malheureusement, on en arrive toujours à mettre des chantiers d'optimisations en place lorsqu'apparaissent des traitements d'alimentation ou des « fenêtres de chargement » de plus en plus longs.

Après audit, on s'aperçoit souvent qu'une majorité des problèmes auraient pu être évitée, si l'on avait suivi quelques conseils, qui s'apparentent plus à de la logique qu'à de l'expertise.

Dans ce Best practice, nous n'avons pas vocation à traiter ici des problématiques d'architecture, de machine ou encore de réseau. Bien que ceux-ci fassent partie intégrante d'un chantier optimisation, nous traiterons de ces sujets ultérieurement, avec notamment une fiche spécifiquement orientée hardware mettant en avant l'impact que peut avoir un serveur sur les performances des traitements de batchs ou encore de requêtes : en effet, les ressources d'une machine ne se limitent pas aux seuls CPU et RAM.

L'objectif de ce document, c'est de se concentrer sur les principaux modules de développements d'Informatica portant sur les phases:

- *de développement des mappings [Designer],*
- *d'ordonnancement des traitements avec l'option de partitionnement [Workflow manager] et enfin*
- *d'analyse de l'exécution des traitements [Workflow monitor].*



Nous nous efforcerons de mettre en avant une rapide check liste, que vous pourrez compléter selon vos propres expériences, et qui vous prémunira de la majorité de vos problèmes de chargement de plus en plus longs.

Ces documents sont le fruit d'experts provenant de chez Informatica, de personnes provenant du milieu des SSII ou encore d'indépendants. Sans eux, ces fiches n'existeraient pas.

Merci à chacun d'entre eux pour leur contribution.

Stéphane THIA

Relecteurs: Christophe FOURNEL, Funji MATEMU

SOMMAIRE

OPTIMISATION GLOBALE

Voilà un chantier aussi vaste que passionnant. Passionnant, car il n'y a pas de chantiers identiques d'un client à un autre. On ne peut pas créer une check liste des points à améliorer, et les mettre en œuvre les uns après les autres.

La check liste vous permettra de ne pas oublier des éléments importants mais ne vous aidera en rien à résoudre ou anticiper vos problèmes de performance. C'est comme si l'on vous donnait une grosse boite à outils dans le but de vous préparer aux éventuels problèmes de performances sans vous donner de notices. Et lorsqu'arrive la « cata », on ne sait pas par quel bout commencer !

Il faut à mon avis avoir vécu des problèmes de performances, pour savoir que dans un cas ou dans un autre, c'est tel ensemble de points de la check liste, ou tel architecture qu'il faudrait privilégier dans le contexte bien défini.

En clair, on ne pourrait utiliser la boite à outils au mieux qu'au travers de ses expériences passées. C'est ce qui fait toute la richesse des profils Informatica expérimentés.

Il existe deux types de projets. Il y a les nouveaux projets, et les projets existants.

Pour les nouveaux projets, l'adage *« **il vaut mieux prévenir que guérir** » prend toute son importance.* En effet, il est important d'anticiper la mise en place de l'architecture Informatica, l'arrivée de nouveaux projets, les moyens utilisés ou à utiliser (budgétaire, technique, etc.), ainsi que la mise en place d'une norme de développement qui assurera une fiabilité et une certaine facilité de maintenance des projets présents et futurs.

A ce sujet, je vous invite à récupérer les fiches existantes dont le socle commun, se base sur la norme VELOCITY d'Informatica :

- Cf. la fiche Best Practice - Fiche 3 - Méthodologie VELOCITY
- Cf. la fiche Best practice - Fiche 2 - Normalisation et codification
- Cf. la fiche Best practice - Fiche 1 - Organisation et droits

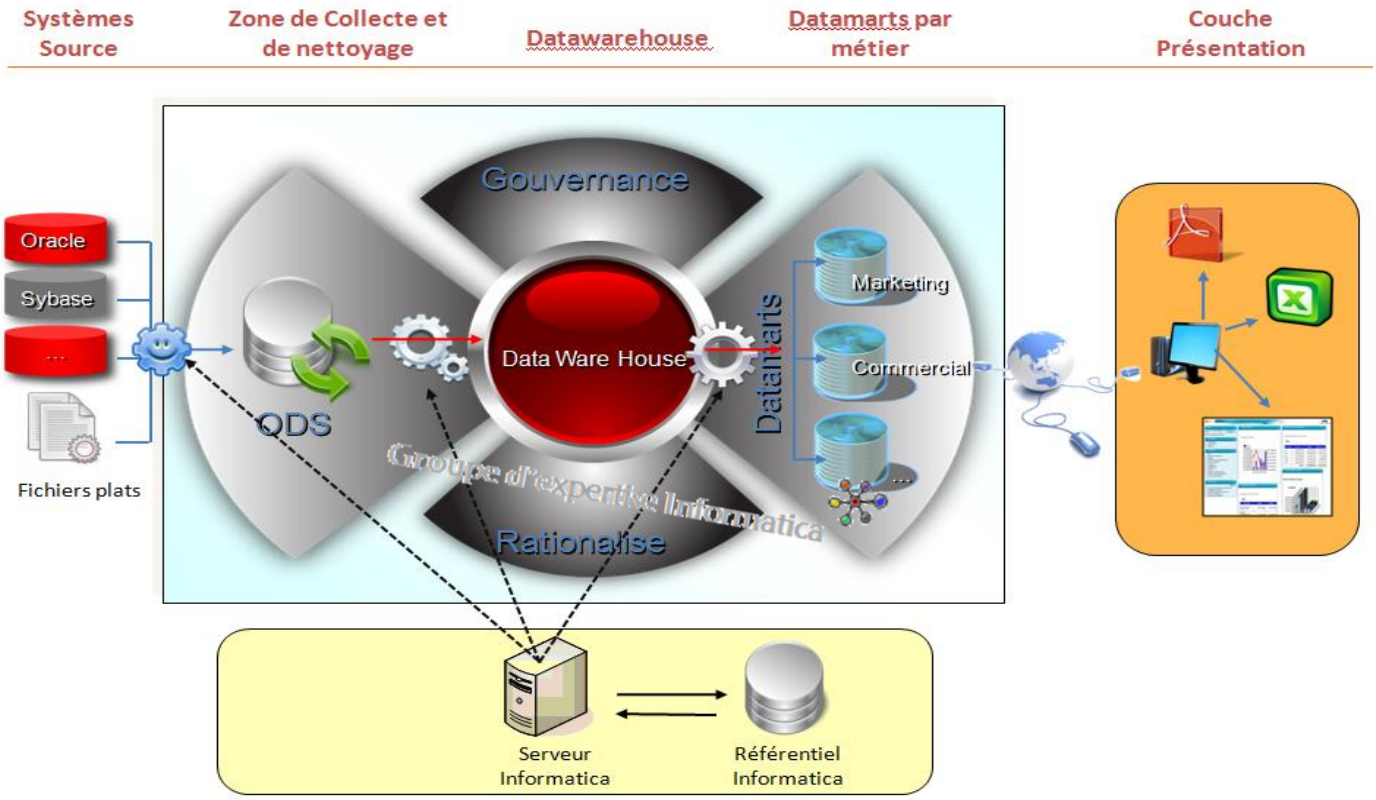
Pour les projets existants, si vous avez des problématiques de temps de chargement qui sont de plus en plus longs, il vous faudra de la patience, et passer en revue les différents éléments de cette fiche, pour identifier quelles pourraient être la ou les causes des ralentissements constatés.

Cette fiche, bien que relativement complète, n'est sûrement pas exhaustive, aussi, je vous invite à la compléter au fil de vos expériences.

Quelque soit le type de projet, vérifiez les points qui suivront, lorsque cela est possible :

A. EXEMPLE D'UNE ARCHITECTURE STANDARD

Il y a beaucoup de typologies d'architecture ; pour cette fiche, on prendra l'exemple d'une « architecture fonctionnelle » simple, intégrant des systèmes sources, un ODS, un Datawarehouse, et des datamarts.



B. CONNAISSANCE DE SON ENVIRONNEMENT

Au niveau des données applicatives, il est intéressant pour chaque application source, d'identifier:

- Son emplacement géographique (architecture centralisée, ou pas, etc.),
- Sa typologie des données (bases de données, fichiers plats, etc.),
- Si le serveur qui l'héberge est dédié ou mutualisé avec d'autres applications,
- Un planning de sollicitation des applications sources (les serveurs des applications sources sont ils très sollicité uniquement en journée, ou en permanence du fait d'une répartition des serveurs dans le monde),
- Le débit réseau entre chaque serveur

Au niveau des environnements de développements, ces derniers *(environnement de développement, de recette, d'intégration et de production)* sont souvent différents en termes de ressource et peuvent être plus ou moins sollicités *(un environnement d'intégration est moins sollicité qu'un environnement de production, par exemple)*: le temps de chargement d'un même traitement Informatica peut varier du simple ou double selon l'environnement dans lequel il est exécuté.

- Vérifier les versions utilisées:
  - Version de la base de données
  - Version des clients des bases de données présents sur le serveur Informatica
  - Version d'Informatica
  - Version des clients Informatica présents sur les postes de développement,
  - Version des systèmes d'exploitation
- Vérifier l'utilisation des bons drivers Informatica *(Par exemple, pour Oracle, privilégier le driver Informatica dédié à Oracle : DataDirect x.x Oracle Wire Protocol).*
-

Au niveau de la base de données,

- Contrôler la volumétrie des données:
  - À charger : s'il y a beaucoup de données, peut-on effectuer des chargements de données en mode incrémentale, utilise t-on les fonctionnalités de partitions offertes par les bases de données quand cela est possible ?
  - À analyser : doit-on recalculer systématiquement des indicateurs agrégés en se basant sur un historique complet (plusieurs années), ou peut-on uniquement prendre en compte les deux dernières années ? Si certaines données anciennes ne « bougent » plus, ne peut on pas les mettre en read-only (fonction permise sur certaines bases de données), ou tout simplement les supprimer (après les avoir sauvegardées).
- Faire attention aux normes de sauvegardes (Par exemple, la norme BALE 2 peut imposer une fréquence de sauvegarde plus ou moins rapprochée), et l'impact qu'elles peuvent avoir sur la monopolisation des ressources,
- Auditer la modélisation à l'intérieur des bases de données :
  - Identifier les requêtes les plus consommatrices, et :
    - Par l'intermédiaire d'un dba qui pourra analyser la charge du serveur,
    - Par un audit des reporting existants, et utilisés de façon récurrente,
    - Identifier si les indexes nécessaires existent,
  - Identifier s'il est possible d'améliorer la modélisation. Si les requêtes font l'objet d'une exécution récurrente, ne peut on pas alimenter une table de fait modélisée en étoile ?

Au niveau des traitements Informatica, vous pourrez passer en revue un certain nombre de questions telles que :

- Quels sont les environnements disponibles, et pour chacun d'eux …
- Quelles sont les ressources disponibles (CPU, RAM, DD) sur chacun des serveurs (pour certaines manipulations, faut-il privilégier la base de données ou l'utilisation des transformations Informatica, ou encore les deux ?),
- Quelle est la fréquence des chargements et la mise en place d'un planning multi-projets des traitements Informatica (éviter que tous les projets démarrent leurs traitements en même temps, répartir les chargements mensuels sur les deux ou trois premiers jours),
- Que les serveurs de base de données ne se situent pas sur le même serveur que le serveur Informatica,
- Que le débit réseau entre chacun des serveurs est suffisant,
- Quel est le nombre de sessions pouvant être exécutées en parallèle sur Informatica, et si ce nombre est suffisant,
- Quel sont les paramétrages du DeadlockSleep et du NumOfDeadlockRetries. Ces actions permettant des accès concurrents en écriture sur une table.
- Etc.

Ce ne sont là que quelques éléments d'indication; je vous invite à établir vous-même votre liste que vous compléterez au fil de vos « épreuves du feu».

Globalement, on pourrait résumer les éléments précédents, par le fait qu'il faut avoir une connaissance générale de l'environnement de ses clients et de leur architecture :

- Mettre en place une architecture adapté au contexte géographique *(Bases de données déportées, locales, centralisées, hébergées, etc.)*,
- Bien connaître son environnement, pour répartir au mieux la charge d'exécution des traitements, sur les serveurs Informatica/Base de données selon les ressources disponibles,
- Faire attention à la différence de ressources entre les différents environnements,
- Faire attention au débit des réseaux,
- Utiliser les fonctions de bases de données disponibles *(Pour Oracle, on peut parler des partitions, des index de jointure, des fonctions analytiques, etc.)*,
- Identifier le nombre de projets Informatica,
- Identifier la fréquence des traitements, et élaborer un planning des chargements Informatica,
- Identifier le nombre de sessions Informatica pouvant tourner en parallèle et vérifier si cela est suffisant en fonction du planning précédemment élaboré.

OPTIMISATION AU NIVEAU DES DEVELOPPEMENTS

A. DE FAÇON GENERALE

Le socle,

Il est important pour optimiser les développements, et leur maintenance, de mettre en place des normes et des directives de développements. Nous vous invitons à récupérer sur notre site www.unovia.fr l'ensemble des fiches disponibles gratuitement :

- Cf. la fiche Best Practice - Fiche 3 - Méthodologie VELOCITY
- Cf. la fiche Best practice - Fiche 2 - Normalisation et codification
- Cf. la fiche Best practice - Fiche 1 - Organisation et droits

Dans ces fiches, vous trouverez également les documents de référence en provenance d'Informatica.

Les développements,

Dans ce *Best practice – Optimisation*, nous traiterons :

- Des reflexes de développements (cette présente fiche),
- De l'option de partitionnement (Prochaine fiche).
- Des goulots d'étranglements (analyse des chargements de données – Prochaine fiche),

Il est important, lors des développements, de connaître le B.A. BA des outils utilisés pour éviter d'être confronté à des problématiques de performance ultérieure.

Ce B.A. BA que nous allons exposer dans cette fiche peut paraître simpliste et logique, mais il est important d'en lister les points pour s'assurer de sa mise en œuvre lors des développements, ou pour ne pas en oublier lors de vos chantiers d'optimisation.

Créons ensemble cette check liste.

B. DE FAÇON PLUS PRECISE, QUELQUES REFLEXES

1. AU NIVEAU DES MAPPINGS

De façon générale sur le projet

- Mettre en place et suivre des règles de nommage.
- Mettre des commentaires (Transformations, Mappings, Sessions, Workflows, etc.).
- Prévoir les solutions en cas de plantage des traitements :
  - ✔ Renforcement des Business Rules,
  - ✔ Redémarrage du chargement sans corruption des données ?
  - ✔ Gestion des rejets fonctionnels ? Retraitement ?
- Capitaliser le plus possible sur des objets réutilisables :
  - ✔ On peut ainsi capitaliser et centraliser un bout de développement, une règle de gestion :
    - ✔ Des objets Partagés, des sources & cibles :
      - Systématiquement les mettre dans *Shared Folder :*
      - Analyse d'impact simplifiée,
      - Maintenance simplifiée,
      - Redondance évitée,
      - Attention : déclaration ODBC sur les PC !
    - ✔ Industrialisation:
      - Transformations réutilisables : par anticipation ou à posteriori, potentiellement, à mettre dans un *Shared Folder*,
      - Mapplets : Bénéficier des travaux des développeurs experts et minimiser les risques d'erreurs lors de l'implémentation de fonctions similaires.
  - ✔ On peut utiliser des fonctions permettant la réutilisabilité : cf. notre fiche sur la fonction d'exécution concurrente, disponible à partir de la version 8.5 de PWC (Fiche astuce - Concurrent Execution).
  - ✔ A ne mettre en œuvre qu'en cas d'usage multiples.

Au niveau des WORKFLOWS & des SESSIONS

- Configurer les caches des sessions pour éviter le SWAP (génération de fichiers de cache).
- Configurer les IS (Integration Service) en DataMovement Code = ASCII si possible (le mode Unicode peut dégrader les performances).
- Consulter les logs de session (Thème de la prochaine fiche : Identification des goulots d'étranglement) pour identifier s'il existe des goulots d'étranglement dans vos traitements, ou encore identifier pour identifier les caches à redimensionner.
- Utiliser les options de partitionnement (en option) lorsque cela est possible (Thème de la prochaine fiche : Utilisations des fonctions de partitionnement).

Au niveau des MAPPINGS

- Éviter les développements externes (C++/Java/PlSql) et privilégier les composants PowerCenter.
- Éviter les pipelines indépendants autant que possible, et privilégier les lectures de type « Single-pass » : 1 SQ et N Cibles :
  - Power Center lit les données pour chaque SQ, autant que la requête soit exécutée une seule fois.
- « Calculer une fois, et utiliser plusieurs fois »:
  - Éviter de calculer la même valeur à plusieurs reprises,
  - Utiliser les ports « Variables » dans les Expressions pour calculer une valeur pouvant être utilisée à n reprises dans les transformations (Expression, Aggregator et Sorter).
- Limiter si possible, le nombre de transformations (il est préconisé de ne pas dépasser la 50aine).
- Filtrer les données le plus en amont possible :
  - Le faire par exemple dans la requête du SQ pour que le filtre soit traité en base de données,
  - Positionner les Transformations Actives (Filtres, Aggrégators) le plus en amont possible pour limiter le plus possible le nombre d'enregistrements en transit dans le mapping
- Ne connecter que ce qui est utilisé :
  - Supprimer les liens inutiles entre les transformations :
    - Diminution de la volumétrie en transit et maintenance facilitée,
    - Lookups : Attention aux ports déconnectés inutiles.
- Éviter de mettre des ports en out si une colonne n'a pas de flux sortant.
- Éviter les conversions implicites de types de données et privilégier les conversions explicites : en effet, si dans un mapping, on a par exemple dans SQ une colonne de type numérique, et que cette colonne est de type caractère dans une transformation, Informatica effectuera une transformation implicite. Il serait préférable dans un objet de type transformation, d'utiliser la fonction Informatica adéquate, rendant la conversion explicite :
  - Meilleures performances,
  - Diminution du risque de corruption de données.
- Éviter de mettre des champs de taille différentes (champs source plus long/court que le champ destination):
  - Warning dans les logs qui peuvent potentiellement représenter de gros volume (en Go),
  - Performances détériorées,
  - Si on doit effectivement réduire la taille d'une donnée, il faut privilégier la fonction Substring.
- Éviter les lookups sans cache excepté pour les lookups déconnectés peu sollicités.
- LOOKUP :
  - Lookups déconnectés vs lookup connectés: si un lookup connecté ne retourne qu'une seule colonne et qu'il est appelé plusieurs fois dans le mappings, son remplacement par un lookup déconnecté permettra de n'accéder qu'une seule fois au SGBDR et prendra moins d'espace,
  - Attention à la volumétrie !
  - Par défaut, Cache Enabled.
- Procédure Stockée :
  - Éviter les procédures stockées appelées pour chaque enregistrement, sauf si la volumétrie est très faible.Cela produit aussi un manque d'informations au niveau de la log Informatica.
- SEQUENCE GENERATOR : Il faut faire attention à la livraison de cette transformation d'un environnement à l'autre. En effet, dans l'environnement de développement, la dernière séquence écrite en développement pourrait être le n° 500 par exemple, et le dernier n° de séquence dans l'environnement de production pourrait être le n° 1280. Lors de sa livraison, l'objet SEQUENCE GENERATOR va conserver le numéro 500, ce qui entraînera une contrainte d'intégrité, car le n° 500 existe déjà dans l'environnement de production. Yannick préconise donc de coupler le SEQUENCE GENERATOR à un LOOKUP dont le seul but sera de récupérer la plus grande séquence en base, avant de commencer à charger.

2. Au niveau des donnees

a. Les bases de donnees en general

- Vérifier l'existence d'indexs au niveau des bases de données :
  - Select sur 20% des données (ou moins), il est pertinent de positionner un index sur les colonnes présentes dans le « Where »,
  - Oracle 11 : apparition d'index de jointure,
  - A l'inverse, des indexs peuvent parfois être à l'origine de performances dégradées,
  - Ajout d'index : travaux en collaboration avec DBA.
- En cas de forte volumétrie, peut-on partitionner les tables, et pour les partitions les plus anciennes,
  - les données sont-elles toujours analysées ? Peut-on les sauvegarder et les supprimer ?
  - Si les données sont toujours utilisées, peut-on compresser les partitions ? Les mettre en read-only ?
- Vérifier que les statistiques des bases de données sont mises à jour régulièrement.
- Vérifier la fragmentation des disques durs, des tablespaces Oracle.
- Analyser l'existence d'éventuels goulots d'étranglement.

b. Extraire les donnees

SGBDR | Système de Gestion de Base de Données Relationnel

- Il faut éviter les SQL Override et privilégier le composant Informatica.
  - A la place, on peut utiliser dans le SQ les options suivantes:
    - Source Filter
    - Join Condition
    - Number of Sorted Inputs
  - On peut utiliser les sql override lorsque le SQL de la base de données est plus performant ou par complexité du SQL (exemple sous Oracle des fonctions analytiques, des fonctions decode, nvl, etc.),
- Ne sélectionner que les données utiles :
  - Ne prendre que les colonnes utiles (diminue le volume qui transite sur le réseau, diminue les risques de transformations implicites, les ports en out sans flux sortant, les risques d'erreur de mapping, etc.).
  - Positionner des conditions dans le SQL (filtrer les données ramenées par la requête).

Fichiers plats

- Généralement, les performances d'extraction d'un fichier plat sont supérieures aux performances d'un SGBDR.
- Lorsqu'on traite des fichiers plats, privilégier les fichiers de type **Fixed Width** dont les performances de chargement seront supérieures aux fichiers de type **Delimited FF**, nécessitant plus de parsing.
- En cas de doute sur le typage des données d'un fichier plat:
  - Définir tous les champs en String
  - Vérifier les types via Is_Date(), Is_Number()
  - Convertir les formats via To_Char, To_Date, To_Decimal etc.

Fichiers XML

Dans le cas où l'on doit traiter des fichiers xml, Informatica préconise d'utiliser son outil B2B DT (spécialement conçu pour les fichiers XML en input/output). Cependant, il est possible, sans B2B de travailler des fichiers XML.

- Le fichier XML étant un fichier hiérarchique, Power Center a besoin pour le parser de le stocker intégralement en mémoire, ce qui peut être très pénalisant,
- Pour améliorer les performances, il est préconiser de:
  - ✓ Réduire la longueur des champs aux longueurs requises (import du XML, option « override All Infinite Length »)
  - ✓ Essayer de splitter les gros fichiers en plusieurs plus petits
  - ✓ Accroitre le « DTM buffer size » pour éviter le Swap

c. TRIER LES DONNEES

Il est possible de trier des données soit à partir du code sql, soit à partir de la transformation sorter prévu à cet effet.
On ne peut ici mettre en avant un best practice, car le choix de la solution dépendra :

- De la volumétrie des données à trier.
- Des ressources de la machine hébergeant la base de données (RAM, CPU, DD).
- Des ressources de la machine hébergeant Informatica (RAM, CPU, DD).
- Du réseau entre les machines.
- Des transformations actives utilisées dans un mapping.

Dans le cas de petites volumétries, la question ne se pose pas forcément, cependant, en cas de volumétrie plus importante, nous préconisons de tester les différentes solutions.

Cependant, les règles de base sont :

- Éviter de filtrer les données quand ce n'est pas nécessaire.
- Filtrer les données avant de les trier.
- Filtrer et trier une fois, et utiliser plusieurs fois.
- En cas de grosse volumétrie, tester :
  - ✓ SQL avec un Order By (on fait travailler la base de données)
  - ✓ SQL + la transformation Sorter (on fait travailler Informatica)

MÉTHODE	A UTILISER QUAND	A ÉVITER QUAND
SOURCE QUALIFIER	- La Source est relationnelle : le tri est exécuté en base. PowerCenter génère les ordres SQL, les envoie au Sgbdr et obtient les données triées en mode stream. - La Table est indexée sur les champs de tri - Le moteur du Sgbdr est performant (64 bit, CPU/RAM/ I/O / Network…)	- La Source n'est pas relationnelle - Les perfs du SGBDR sont mauvaises - On souhaite minimiser les impacts sur les ressources du Sgdbr source.
SORTER	- La Source est un FF non trié - La Source est un SQ pointant vers une application où les données ne peuvent pas être triées - Le tri doit avoir lieu après la phase de transformation des données - Le serveur PWC dispose de ressources importante (CPU/ RAM / I/O) et l'option « partitioning » est disponible: l'augmentation des ressources affectées aux jointures devraient réduire les durées de jointure	- Très grand volumes de données avec peu de ressources disponibles sur le serveur PowerCenter

d. JOINDRE LES DONNEES

Les règles de base sont :

- Privilégier les jointures sur des champs numériques
- Lorsque l'on utilise la transformation JOIN,
  - Cas de données non triées: Mettre en « Master » = la table contenant le moins d'enregistrements
  - Cas de données triées: Mettre en « Master » la table avec le moins de doublons sur la condition de jointure
- Dans les lookups:
  - Penser à désélectionner les ports non utilisés,
  - Concernant l'option « Lookup Policy on multiple Match » présente dans l'onglet properties du lookup, on a la valeur « Use Any Value » par défaut. Un index est alors créé seulement sur les ports en jointure (sinon index sur tous les ports en sortie),
  - Penser à utiliser le Lookup Source Filter présente dans l'onglet properties du lookup, lorsque l'on peut diminuer les données,
  - Placer toutes les conditions d'équi-jointure en 1er, puis les autres conditions par la suite.

Ci-dessous un tableau (en anglais … sigh) provenant de la documentation Informatica :

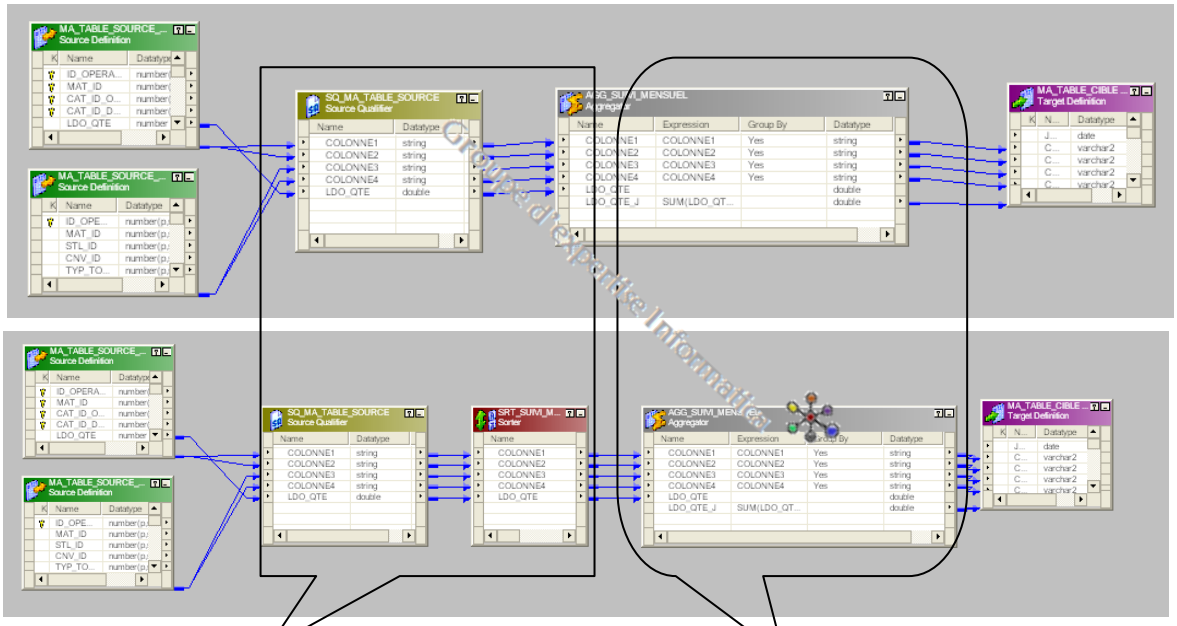
MÉTHODE	A UTILISER QUAND	A ÉVITER QUAND
SOURCE QUALIFIER	- Source objects are relational tables from same database: join statement is executed on database. PowerCenter generate SQL statement, send it to database, and get data joined in stream mode - Tables have index on join condition - Join reduces data volume: less data transported - Database is very efficient: 64 bit, high CPU/RAM/ I/O / Network - More than 2 tables from same database to join: Only Source Qualifier could joins more than 2 tables in 1 objects - Need to call complex database function that could take time to replicate it in PowerCenter - Join types are: inner, left outer, right outer, full outer, and/or inequality - Join condition contains functions	- La Source n'est pas relationnelle - Les perfs du SGBDR sont mauvaises - On souhaite minimiser les impacts sur les ressources du Sgdbr source.
JOINER Transformation	- Heterogeneous sources: join 1 flat file with 1 table, 2 tables from distinct database (1 from SAP, 1 from SQL Server for instance) - Need to perform join operation after data transformation - Data are sorted on join condition: always use joiner in this case and enable joiner option "Sorted Input" to indicate that data are sorted. Joiner will not sort data again - PowerCenter 64 bit version installed and enough resource to avoid cache - PowerCenter server has important resource available (CPU/ RAM / I/O) and partitioning option is enabled: join become perfectly scalable, so increasing resource affected to join will probably reduce join time - Join types are: inner, left outer, right outer or full outer.	- Inequality join condition (not supported) - More than 2 tables to join, otherwise need (n - 1) joiner to perform n joins - Join condition contains functions. To use joiner in this case, perform functions in an expression transformation before joining data - High data volume with low resources available on PowerCenter server and/or PowerCenter is 32 bit - You just need to enrich data when join condition is true, and ignore it otherwise. Use a lookup or Source qualifier in this case

FOCUS SUR LES LOOK UP		
MÉTHODE	**A UTILISER QUAND**	**A ÉVITER QUAND**
Uncached connected lookup	- Huge lookup table BUT called very few times (low source data volume) otherwise NEVER use it - You only need to enrich data or check presence of data - Lookup table is a flat file or relational table	- You call it more than few times: 100, 200, 500, 1000. It will depend on your environment and lookup table size - Inner join and full outer join - Lookup on application data (not supported)
Uncached unconnected lookup	- You only need 1 field to be returned from lookup table - Called very few times - Called from several transformations	- You need to use SQL Override - Call lookup many times - You need more than 1 field to be returned - You access lookup as for each source row - Inner join and full outer join - Lookup on application data (not supported)
Cached connected lookup	- You access lookup for each source row - You only need to enrich data or check presence of data. Passive outer join (for each call, you have only 1 response, which is null if join condition is false) - You have enough memory to avoid PowerCenter SWAP (cache) - Override SQL (to join with multiple tables, add complex filters, sub queries, use database function…)	- High data volume with low resources available on PowerCenter server and/or PowerCenter is 32 bit - Called very few times - Inner join and full outer join - Lookup on application data (not supported)
Persistent lookup cache	- You need to access same lookup data across multiple sessions - SQL Statement takes long time to retrieve data - You only need to enrich data or check presence of data - You have enough memory to avoid PowerCenter SWAP	- Lookup data updated between persistent file creation and persistent file usage - Very low data volume - Called very few times - Inner join and full outer join
Cached unconnected lookup	- You don't access lookup for each source row - Called from several transformations - You only need to enrich data or check presence of data	- Called very few times - Inner join and full outer join - Lookup on application data (not supported) - High data volume with low resources available on PowerCenter server and/or PowerCenter is 32 bit
Dynamic cached connected lookup	- You need to update lookup cache in order to keep synchronize cache with target table - You access lookup as for each source row - You only need to enrich data or check presence of data - You have enough memory to avoid PowerCenter SWAP	- You don't need to refresh cache data - Inner join and full outer join - High data volume with low resources available on PowerCenter server and/or PowerCenter is 32 bit
Pipeline lookup	- Lookup from application source qualifier (SAP, Siebel, Oracle EBiz, zOs, AS400, SAS, …) and relational Source Qualifier - Could perform all Source Qualifier operations: join on multiple sources, SQL Override, - You only need to enrich data or check presence of data - Could be instanced multiples times in a mapping - Called for each source row - SQL statement could be executed during session initialization (not waiting for first lookup call). SQL Statement is executed in parallel - Identify where a table is used	- Inner join and full outer join - Huge data volume you cannot put it entirely in memory - Called very few times - Connected source qualifier

e. AGREGER LES DONNEES

Règles de base:

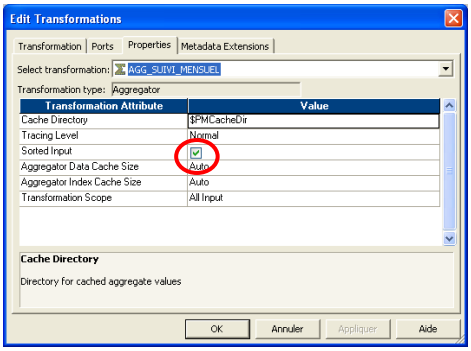
- Filtrer les données en amont : en premier lieu dans le SQ lorsqu'il s'agit d'une base de données en source (si les ressources machines et le réseau le permettent), puis le plus tôt possible dans le pipeline du mapping si on utilise une transformation de type Filter.
- La transformation Aggregator :
  - Lorsque l'on utilise cette transformation, il faut faire attention :
    - ✓ aux ressources disponibles sur le serveur PWC, et au partage de ces ressources avec les autres projets,
    - ✓ à la mémoire requise pour la transformation aggregator, qui est fonction du nombre d'enregistrements en sortie de l'aggregator.
- Dans le cas d'une forte volumétrie, et que le taux d'aggrégation est élevé, il est préconisé :
  - de faire en sorte que les données soient déjà triées lorsqu'elles entrent dans la transformation Aggregator. (les données seront soit triées via un order by au niveau de la base de données, soit par une transformation sorter. Des tests doivent être effectués, car cela dépendra des ressources disponibles. Attention, ne pas oublier que les ressources peuvent différentes selon les environnements. Les tests doivent donc s'effectuer dans un environnement le plus proche possible des conditions de production),
  - de cocher dans les propriétés de la transformation aggregator, la fonction **sorted input**.



1 - Tri des données sources :
- schéma1 : par un order by (dans le SQ)
- schéma2 : par une transformation SORTER

2 - Cocher Sorted input:

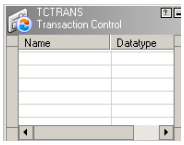
Remarque : quand les volumes de données sont peu élevés, le Sorter n'est pas requis.



- L'objectif, c'est de ne pas mettre toutes les données dans le cache de l'aggregator : à chaque nouvelle clef d'agrégation, la donnée sortira de la transformation. On évite ainsi des caches trop volumineux, et on parallélise la lecture et l'écriture.

f. GERER LES TRANSACTIONS – TRANSACTION CONTROL

Cette transformation, peu connue permet d'optimiser la gestion des transactions.
Cette transformation est particulièrement intéressante dans le cas de grosses volumétries, notamment grâce à son impact sur le flush mémoire des transformations actives.



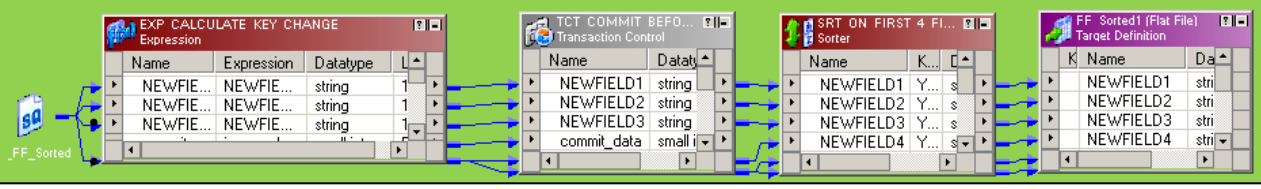
Etant donné qu'il s'agit d'une transformation peu courante dans les développements, nous l'illustrerons ci-après à partir d'un mapping standard et ce même mapping intégrant le TCT.

A titre d'exemple, nous imaginerons un pipeline standard, comprenant un SQ, puis un Sorter (on aurait pu également prendre une transformation *Rank, Aggregator à la place du Sorter*) :

- En source, on a un fichier plat, contenant 10 champs, 100 millions de lignes, trié sur les 3 premiers champs,
- Une contrainte fonctionnelle qui est d'effectuer:
  - un tri ascendant sur les 3 premiers champs,
  - un tri descendant sur le 4ème champ.

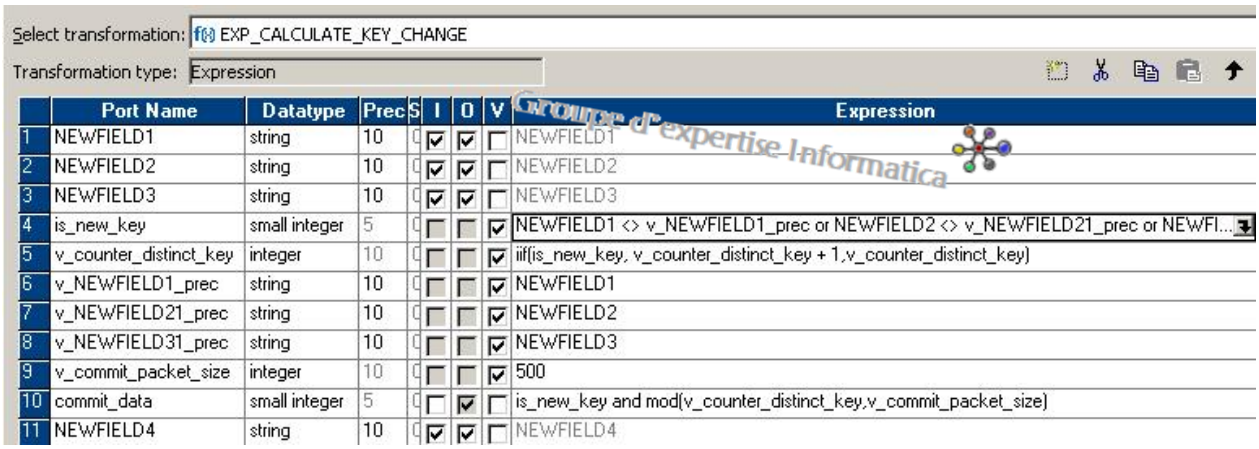
✔ Scénario 1: Sans TCT
- On utiliserait un Sorter avec une quantité très importante de RAM, et des temps de chargement très longs, car l sorter devra tout mettre en cache pour effectuer son tri,
- Par ailleurs, on ne tire pas avantage du tri existant dans le fichier sur les 3 1ers champs

✔ Scénario 2: Avec TCT
- On intègre dans le mapping, le TCT avant le sorter,
- Le TCT permet de flusher la mémoire du Sorter à chaque changement de clef (3 champs).



Focus sur la transformation TCT

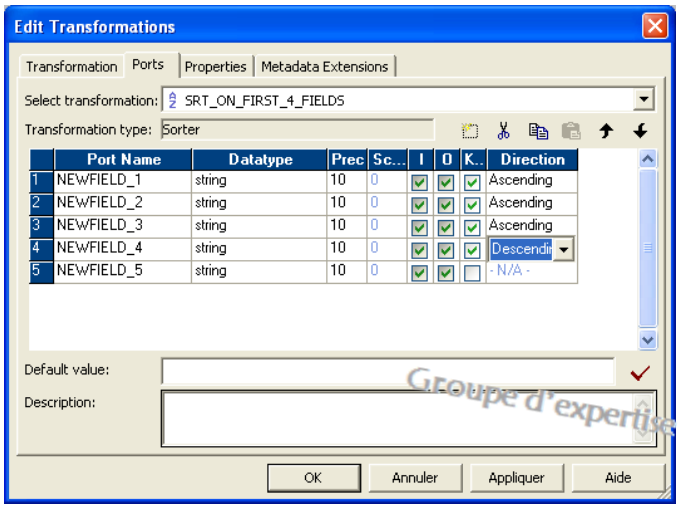
- L'Expression calcule le changement de clef, et on décide du « pas » de commit. Par exemple, le champ *commit_data*, ci-dessous valorisé à 500, effectuera un commit tous les 500 changements de clefs.

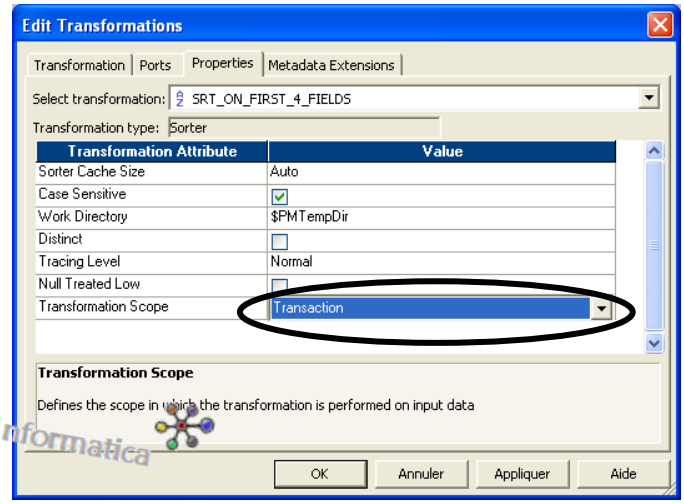


- La TCT effectue les commits :
  IIF(commit_data, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)
  TC_COMMIT_BEFORE: l'enregistrement en cours ne fait pas partie de la transaction en cours, mais de la suivante

Focus sur la transformation SORTER

- Préciser le sens du tri (ascendant et descendant)
- Préciser les propriétés du SORTER





g. CHARGER LES DONNEES

Les fichiers plats,

- Privilégier les « Delimited » FF pour un gain d'espace disque,
- Privilégier les « Fixed Length » FF pour des raisons de performance au cas où le FF soit lu par la suite,
- Format Date à éviter, à traduire en String (contrôle facilité),
- Le FF peut être écrit avec les données triées.

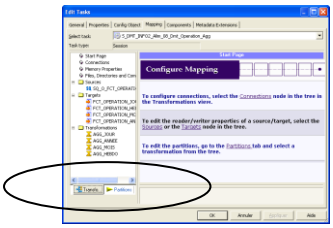
Les tables de bases de données,

- Transaction relationnelles de type *Delete et Update*:
  - Index sur les tables ⇔ Clef logique dans la cible PWC.
  - Mise à jour régulière des statistiques de bases de données.
- Stratégie d'alimentation, focus sur le *Mode bulk*:
  - Ce mode d'alimentation est préconisé en cas de forte volumétrie.
  - Etape 1 : Suppression Index/Contraintes.
  - Etape 2 : Chargement des données.
  - Etape 3 : Reconstruction des index/Contraintes.
- Possibilité de partitionner les chargements (fiche Fiche 4.3 – Optimisations – Utilisation des fonctions de partitionnement),
- Auditez régulièrement les requêtes et les traitements les plus consommateurs.
- Analysez-les avec l'aide d'un DBA !

LA GESTION DES PARTITIONS

Fiche 4.2 – Optimisations – Utilisation des fonctions de partitionnement :

Lors de vos développements, vous avez dû remarquer dans la configuration de votre session, qu'il y avait un onglet Transformations, et un onglet Partitions :



Cette fonction de partitions n'est pas toujours disponible, car c'est (en tout cas j'en suis resté là) une option payante d'Informatica. Nous vous exposerons dans cette fiche les différentes façons d'utiliser cette fonctionnalité, et vous exposerons les préconisations de développement.

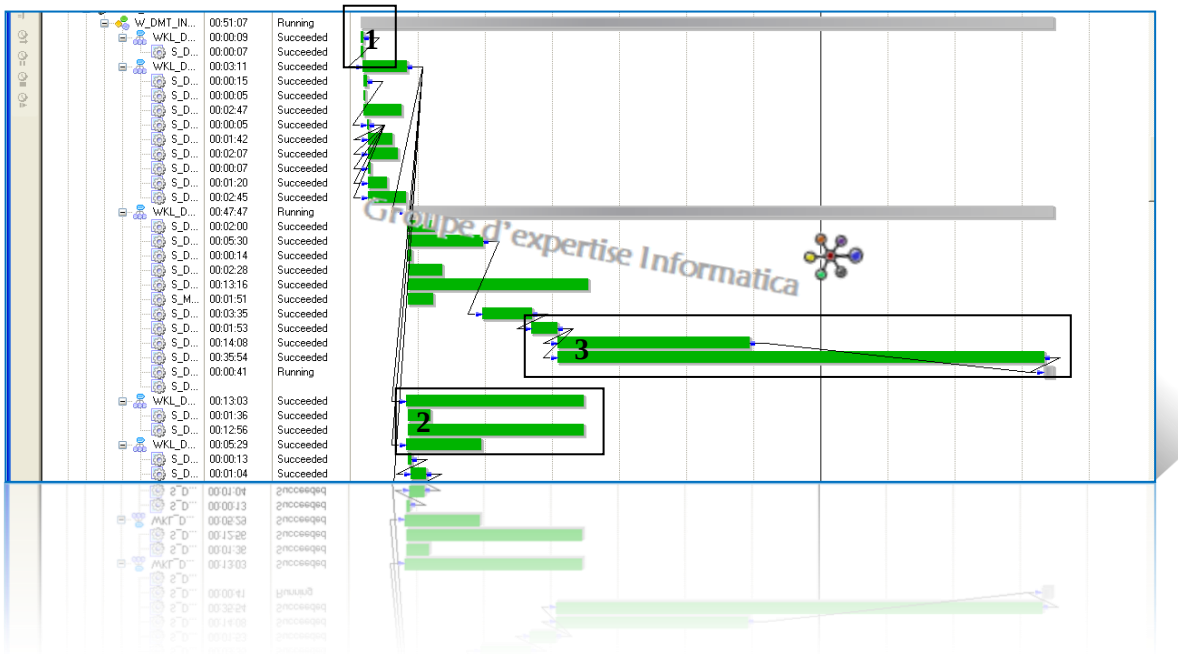
LES « GOULOTS D'ETRANGLEMENT »

Fiche 4.3 – Optimisations – Identification des goulots d'étranglement :

Dans le cadre de cette fiche, nous resterons focalisés sur les traitements Informatica, et plus précisément sur l'analyse des sessions. Dans l'ordre, nous tenterons d'identifier en premier les goulots dans le Workflow monitor, puis dans les sessions elle-même (on saucissonnera la session, pour voir s'il existe des temps de latences entre les phases d'extraction, de transformation et d'écriture).

Par exemple,

- On pourrait imaginer dans la session 1, qu'il y a un goulot d'étranglement. On voit cependant, qu'il n'y a pas lieu de l'analyser ou de l'optimiser, car le traitement dure très peu de temps (et ne conditionne pas le démarrage de traitements plus « lourds »).
- On voit que le traitement 2, même s'il peut paraître un peu long, n'est pas suivi d'autres traitements. De même, son temps de traitement est bien inférieur au traitement 3. On ne le considèrera pas comme étant une session à optimiser, bien qu'il faille le surveiller.
- Par contre, on voit bien que la session 3 dure très longtemps, et conditionne la suite des traitements (en running). Typiquement, dans notre fiche, nous vous expliquerons :
  - comment saucissonner les différentes étapes de la session,
  - comment analyser les différentes étapes de la session.



CONCLUSION

- On peut avoir mis en place une super architecture, des serveurs de folies, suivre les normes de développements et avoir optimisé la base de données, et être pénalisé par le débit du réseau,
- On peut avoir une bonne architecture, des serveurs bien taillés, des bases optimisées, mais ne pas avoir planifié le démarrage des nombreux projets qui démarrent tous en même temps,
- On peut avoir une bonne architecture, les bonnes ressources, tout bien planifier, et avoir des requêtes mal écrites dans les SQ Override,
- etc.

Ce qui est passionnant dans ce genre d'audit, c'est qu'on doit s'intéresser à une pléthore de paramètres illustrés par la diversité de la population concernée. On peut aller discuter avec un développeur sql ou avec un dba, en passant par l'architecte, le responsable réseau ou encore le DSI pour des problématiques.

Les problématiques variant d'un client à l'autre, le plus important, au début d'un audit, est de ne pas se disperser. Il faut avant tout cartographier l'environnement et saucissonner l'architecture, puis identifier les goulots d'étranglements pour affiner au fur et à mesure.

J'espère que cette fiche vous servira et qu'il sera un bon support pour démarrer vos premiers audits !

DOCUMENTS DE REFERENCE

Fiche de la communauté

- Best practice - Fiche 1 - Organisation et droits
- Best practice - Fiche 2 - Normalisation et codification
- Best Practice - Fiche 3 - Méthodologie VELOCITY

PowerCenter Designer Guide 8.5.1

- Workflow administration guide
  → Support for ASCII and Unicode Data Movement modes