Retail passion. Innovative solutions.

Retek® Merchandising System™ 11.0

Guide d'exploitation - Volume 3

Présentation des programmes batch

Siège social :

Retek Inc.

Retek on the Mall

950 Nicollet Mall

Minneapolis, MN 55403

USA

888.61.RETEK (appel gratuit aux États-Unis:

+1 612 587 5000

Fax:

+1 612 587 5100

Siège européen :

Retek

110 Wigmore Street

Londres

W1U 3RW

Royaume-Uni

Standard :
+44 (0)20 7563 4600

Département commericale :
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Imprimé aux États-Unis d'Amérique.

Support clientèle

Horaires du support clientèle

Le support clientèle est disponible 7 jours sur 7, 24 heures sur 24 et 365 jours par an par e-mail, téléphone et Internet.

Selon l'option d'assistance choisie par un client donné (Standard, Plus ou Premium), les heures d'accès à certains services peuvent être limitées. Les problèmes de gravité 1 (graves) sont traités 7 jours sur 7 et 24 heures sur 24 et font l'objet d'une attention continue jusqu'à leur résolution, pour tous les clients qui disposent d'une maintenance active. Les clients de Retek avec des contrats de maintenance actifs peuvent entrer en contact avec support clientèle global selon des conditions de contrat dans un des manières suivantes :

Méthode de contact	**Coordonnées**
Internet (ROCS)	rocs.retek.com Site Web client sécurisé de Retek pour la mise à jour et la consultation des problèmes
E-mail	support@retek.com
Téléphone	+1 612 587 5800

Les solutions gratuites sont également disponibles dans diverses régions du monde :

Australie	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
La Corée	00 308 13 1342
Royaume Uni	0800 917 2863
Etats Unis	+1 800 61 RETEK or 800 617 3835
Courrier	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

Lorsque vous contactez l’assistance clientèle, veuillez fournir:

- La version du produit et le nom du programme/module.
- Une description fonctionnelle et technique du problème (y compris l’impact commercial).
- Les instructions de reconstitution, détaillées, étape par étape.
- Le message d’erreur exact reçu.
- Les copies d’écran de chaque étape que vous suivez.

Contenu

Chapitre 1 – Introduction

Ce document comprend deux parties.

La première partie récapitule les caractéristiques du traitement batch Pro*C dans RMS et décrit les éléments suivants :

- Reprise et récupération
- Multi-traitements
- Seuils d'exécution
- Traitement vectoriel
- Formats d'entrée et de sortie vers des applications et entités externes

La seconde partie récapitule les caractéristiques du traitement batch RETL et décrit les éléments suivants :

- Architecture
- Installation
- Configuration
- Code de retour au programme
- Fichiers de contrôle du statut du programme
- Consignation des messages
- Fichiers de rejet
- Fichiers de schéma
- Paramètres de ligne de commande
- Situations courantes d'exécution et de débogage

Chapitre 2 – Reprise et récupération Pro*C

RMS a mis en œuvre un processus de reprise et de récupération sur une grande partie de son architecture de batch. L'objectif principal de la reprise/récupération est de :

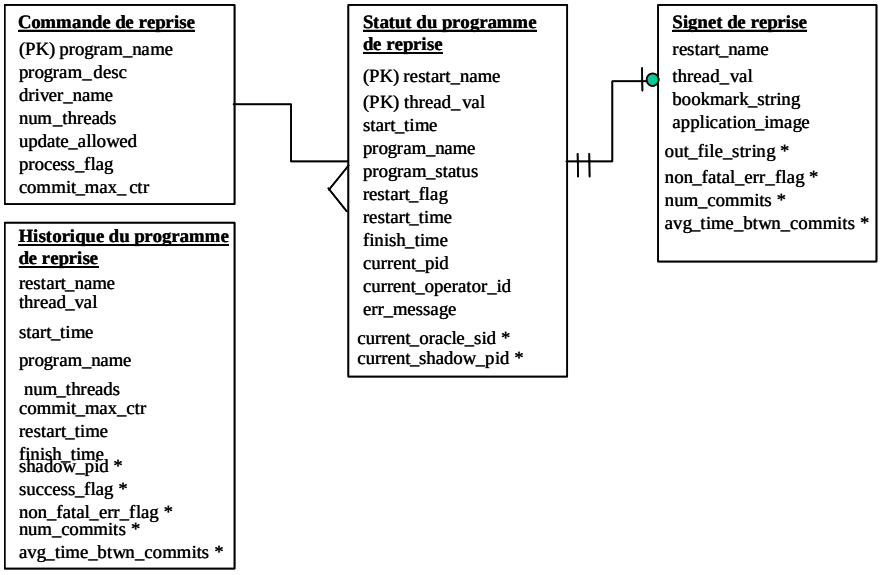
- Reprendre un processus interrompu à partir du point d'échec
- Empêcher les interruptions de système dues à un grand nombre de transactions
- Permettre l'activité simultanée de plusieurs instances d'un processus donné

Par ailleurs, la fonction de reprise/récupération de RMS enregistre les statistiques d'exécution des batch et ne requiert pas l'exécution d'une autorité DBA.

Les capacités de reprise sont centrées autour de l'unité de travail logique d'un programme (LUW). Un programme batch traite des transactions et des points d'exécution sont activés en fonction de la LUW. Les LUW consistent en une clé de transaction relativement unique (telle que référence/magasin) et en un nombre maximal d'exécutions. Les événements d'exécution interviennent lorsque le nombre de clés de transaction traitées atteint ou dépasse le nombre maximal d'exécutions. Par exemple, toutes les 10 000 combinaisons référence/magasin, une exécution intervient. Au moment de l'exécution, les informations de la clé nécessaires à la reprise sont stockées dans les tables de reprise. Lors d'une exception résolue ou non, les transactions sont renvoyées au dernier point d'exécution et au moment de la reprise, les informations de la clé sont extraites des tables pour permettre au traitement de continuer à partir du dernier point d'exécution.

Descriptions et définitions des tables

Le processus de reprise/récupération de RMS est géré par un ensemble de quatre tables. Le schéma 1 présente les relations entre les entités. Les descriptions des tables suivent.



Remarque : les champs suivis d'un astérisque (*) sont utilisés uniquement par les nouveaux programmes batch de la version 9.0 ou supérieure.

restart_control

La table restart_control est la table principale du groupe de tables de reprise/récupération. Elle contient un enregistrement pour chaque programme batch exécuté avec la logique de reprise/récupération activée. Le processus de reprise/récupération utilise cette table pour déterminer :

- si la reprise/récupération utilise des tables ou des fichiers,
- le nombre total de traitements utilisés pour chaque programme batch,
- le nombre maximal d'enregistrements traités avant l'intervention d'un événement d'exécution,
- le pilote de la logique de traitement (multi-traitements).

RESTART_CONTROL			
(PK) program_name	varchar2	25	Nom du programme batch.
program_desc	varchar2	50	Brève description de la fonction du programme
driver_name	varchar2	25	Pilote de requêtes, par exemple, rayon (non modifiable)
num_threads	num	10	Nombre de traitements utilisés pour le processus en cours
update_allowed	varchar2	2	Indique si l'utilisateur peut mettre à jour le nombre de traitements ou si cette mise à jour est faite par programme
process_flag	varchar2	1	Indique si le processus utilise des tables (T) ou des fichiers (F).
commit_max_ctr	num	6	Valeur numérique maximale du compteur avant l'exécution

restart_program_status

La table restart_program_status contient les informations enregistrées sur les traitements des programmes en cours. Le nombre de lignes pour un programme dans la table d'état est identique à la valeur num_threads de la table restart_control. La table d'état est modifiée lors de l'initialisation du processus de reprise/récupération et de la logique de clôture. Pour le processus qui utilise les tables, la logique d'initialisation de reprise/récupération affecte le prochain traitement disponible à un programme basé sur l'état du programme et la balise de reprise. Pour le processus qui utilise les fichiers, la valeur du traitement est définie à partir du nom du fichier d'entrée. Après l'affectation d'un traitement, le program_status est mis à jour pour empêcher l'affectation de ce traitement à un autre processus. Les informations sont reportées sur l'état actuel d'un traitement donné, ainsi que les informations enregistrées, telles que l'opérateur et la durée du processus.

Remarque relative à la configuration : autorisez le verrouillage au niveau des lignes et les "lectures impropres" (n'attendez pas le déverrouillage des lignes pour la lecture des tables).

RESTART_PROGRAM_STATUS			
(PK) restart_name	varchar2	50	Nom du programme.
(PK) thread_val	num	10	Nombre de traitements.
start_time	date		jj-mm-aa hh:mi:ss
program_name	varchar2	25	Nom du programme.
program_status	varchar2	25	Démarré, interrompu, interrompu lors de l'initialisation, interrompu lors du traitement, interrompu à la fin, terminé, prêt pour démarrage.
restart_flag	varchar2	1	Automatiquement défini à "N" après fin inhabituelle, doit être défini manuellement à "Y" pour redémarrer le programme.
restart_time	date		jj-mm-aa hh:mi:ss
finish_time	date		jj-mm-aa hh:mi:ss
current_pid	num	15	ID du programme de démarrage.
current_operator_id	varchar2	20	Opérateur qui a démarré le programme.
err_message	varchar2	255	Enregistrement à l'origine de l'interruption du programme et message d'erreur associé.
current_oracle_sid	num	15	Oracle SID pour la session associée au processus en cours.
current_shadow_pid	num	15	ID du processus O/S pour le processus en double associé au processus en cours. Utilisé pour localiser le fichier de traçage de la session lorsqu'un processus ne s'est pas terminé correctement.

restart_program_history

La table restart_program_history contient un enregistrement pour chaque traitement d'un programme terminé avec succès avec la logique de reprise/récupération. Lorsque le traitement d'un programme se termine avec succès, son enregistrement dans la table restart_program_status est inséré dans la table d'historique. Les utilisateurs peuvent supprimer des tables s'ils le désirent.

RESTART_PROGRAM_HISTORY			
(PK) restart_name	varchar2	50	
(PK) thread_val	Num	10	
(PK) start_time	Date		
program_name	varchar2	25	
num_threads	Num	10	
commit_max_ctr	Num	6	
restart_time	date		
finish_time	date		
shadow_pid	Num	15	ID du processus O/S pour le processus en double associé au processus. Utilisé pour localiser le fichier de traçage de la session.
success_flag	varchar2	1	Indique si le processus s'est terminé avec succès (pour utilisation future).
non_fatal_err_flag	varchar2	1	Indique si des erreurs non fatales sont intervenues durant le processus.
num_commits	Num	12	Nombre total d'exécutions pour le processus. La dernière exécution possible lors de la clôture du processus de reprise/récupération n'est pas comptabilisée.
avg_time_btwn_commits	Num	12	Durée moyenne cumulée entre les exécutions pour le processus. La dernière exécution possible lors de la clôture du processus de reprise/récupération n'est pas comptabilisée.

restart_bookmark

Lorsqu'un traitement du programme de reprise/récupération est en cours, qu'il est démarré ou interrompu et qu'un enregistrement correspondant existe dans la table restart_bookmark, la logique d'initialisation de reprise/récupération insère l'enregistrement dans la table pour le traitement d'un programme. Le processus d'exécution de reprise/récupération met à jour l'enregistrement avec les informations de reprise suivantes :

- une chaîne concaténée des valeurs clés pour le traitement des tables,
- la valeur du pointeur de fichier pour le traitement des fichiers,
- des informations sur le contexte de l'application, tels que compteurs et accumulateurs.

Le processus de clôture de la reprise/récupération supprime l'enregistrement de traitement du programme lorsque le programme se termine avec succès. Lors d'une reprise, les informations de cette table concernant le traitement du programme permettent au processus de démarrer à partir du dernier point d'exécution.

RESTART_BOOKMARK			
restart_name	varchar2	50	
thread_val	Num	10	
bookmark_string	varchar2	255	Chaîne de caractères de la clé du dernier enregistrement exécuté.
application_image	varchar2	1000	Paramètres de l'application à partir du dernier point de sauvegarde.
out_file_string	varchar2	255	Pointeurs de fichiers concaténés (Unix utilise parfois le terme de positions continues pour ces pointeurs) de tous les fichiers de sortie à partir du dernier point d'exécution du processus en cours. Utilisés pour retourner au point de reprise approprié pour tous les fichiers de sortie lors du processus de reprise.
non_fatal_err_flag	varchar2	1	Indique si des erreurs non fatales sont intervenues durant le processus en cours.
num_commits	Num	12	Nombre d'exécutions pour le processus en cours. La dernière exécution possible lors de la clôture du processus de reprise/récupération n'est pas comptabilisée.
avg_time_btwn_commits	Num	12	Durée moyenne entre les exécutions pour le processus en cours. La dernière exécution possible lors de la clôture du processus de reprise/récupération n'est pas comptabilisée.

v_restart_x

Des vues de reprise sont utilisées pour les programmes avec requêtes nécessitant plusieurs traitements. Des vues distinctes sont créées pour chaque pilote de traitement, rayon ou magasin par exemple. Une jointure sera appliquée sur une vue basée sur un pilote de traitement pour permettre la séparation des données discrètes en traitements spécifiques. Reportez-vous à la section consacrée aux traitements pour plus de détails.

V_RESTART_X		
driver_name	varchar2	Par exemple, rayon, magasin, région, etc.
num_threads	number	Nombre total de traitements dans l'ensemble (défini dans la table restart_control).
driver_value	number	Valeur numérique de driver_name.
thread_val	number	Valeur du traitement défini pour la combinaison driver_value et num_threads.

Présentation des modèles de données

Pourquoi les tables restart_program_status et restart_bookmark sont t-elles séparées

Le processus d'initialisation doit extraire toutes les lignes associées au schéma restart_name, mais ne met à jour q'une seule ligne. Le processus d'exécution verrouille de façon continue une ligne avec une valeur restart_name et une valeur thread_val spécifiques. Les données impliquées dans ces deux processus sont divisées entre les deux tables pour réduire le nombre d'interruptions pouvant intervenir du fait du verrouillage des lignes. Même si vous autorisez les "lectures impropres" sur les lignes verrouillées, un processus peut encore s'interrompre s'il tente de mettre à jour une ligne verrouillée. Le processus d'exécution ne s'intéresse qu'à une seule ligne. Par conséquent, si nous déplaçons les données du processus d'exécution dans une table séparée avec verrouillage des lignes (et non des pages), aucun problème de conflit n'interviendra au cours de l'exécution. Avec des tables séparées, le processus d'initialisation détecte un nombre moins important de conflits car les lignes ne sont verrouillées que deux fois, au début et à la fin du processus.

Configuration physique

Le processus de reprise/récupération doit être aussi robuste que possible dans l'éventualité d'une défaillance de la base de données. Les coûts sont compensés par les avantages apportés par le placement des tables de reprise/récupération dans une base de données distincte. Cependant, ces tables doivent être définies dans un espace de table distinct doublé avec un segment de repositionnement séparé.

Reprise/récupération avec tables et fichiers

Le processus de reprise/récupération stocke toutes les données nécessaires à la reprise du processus à partir du dernier point d'exécution. Les informations nécessaires sont donc mises à jour dans la table restart_bookmark avant que les données traitées ne soient exécutées. Les modules avec requêtes et fichiers stockent différentes informations dans les tables de reprise et appellent donc différentes fonctions au sein de l'API de reprise/récupération pour effectuer leurs tâches.

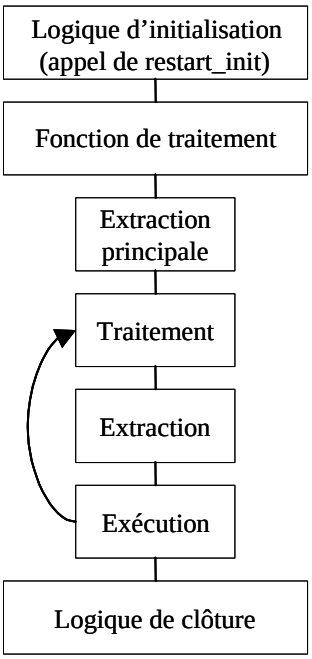
Lorsque le processus d'un programme est basé sur des requêtes, c'est-à-dire qu'un module est géré par une requête pilote qui traite les lignes extraites, les informations stockées dans la table restart_bookmark sont associées aux données extraites de la requête pilote. Si le programme échoue lors du traitement, les informations stockées dans les tables de reprise peuvent être utilisées dans la clause WHERE conditionnelle de la requête pilote pour extraire uniquement les données qui doivent être traitées depuis le dernier événement d'exécution.

Cependant, les traitements avec fichiers doivent simplement stocker l'emplacement du fichier au moment du dernier point d'exécution. L'emplacement par octets du fichier est stocké dans la table restart_bookmark et est extrait lors d'une reprise. Ces informations sur l'emplacement sont utilisées pour rechercher dans le fichier rouvert le dernier point d'exécution des données.

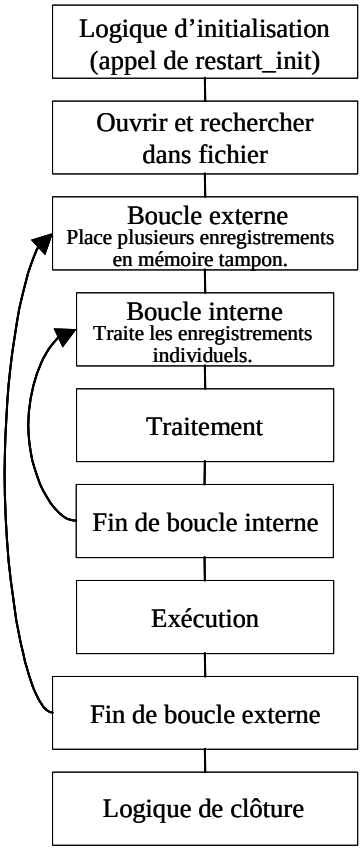
Dans la mesure où différentes informations sont enregistrées et extraites de la table restart_bookmark pour chaque type différent de traitement, différentes fonctions doivent être appelées pour exécuter la logique de reprise/récupération. Le traitement avec requêtes appelle les fonctions restart_init ou retek_init et restart_commit ou retek_commit, tandis que le traitement avec fichiers appelle les fonctions restart_file_init et restart_file_commit.

Outre les différences en ce qui concerne les appels de fonctions API, le flux de traitement batch de la reprise/récupération diffère selon les fichiers. Le processus de reprise/récupération avec tables utilise un flux logique d'extraction principal alors que le traitement avec fichiers lit généralement les lignes dans un batch. Le traitement avec tables requiert que sa structure assure la modification de la clé LUW avant qu'un événement d'exécution soit autorisé, alors que le traitement avec fichiers n'a pas besoin d'évaluer la LUW qui peut être généralement considérée comme le type de transaction traité par le fichier d'entrée.

Le schéma ci-dessous décrit le flux du programme de reprise/récupération avec *tables :*



Le schéma ci-dessous décrit le flux du programme de reprise/récupération avec *fichiers*



Logique d'initialisation :

- Déclarations de variables
- Initialisation de fichiers
- Appel de la fonction restart_init() - détermine la logique de démarrage ou de reprise
- Première extraction sur la requête pilote

Logique de démarrage : initialise les compteurs/accumulateurs sur les valeurs de démarrage

Logique de reprise :

- Analyse du champ application_image de la table de signets dans les compteurs/accumulateurs
- Initialisation des compteurs/accumulateurs pour les valeurs des champs analysés

Processus/boucle d'exécution :

- Mises à jour et manipulations du processus
- Extraction du nouvel enregistrement
- Création de varchar depuis les compteurs/accumulateurs à transmettre au champ application_image de la table restart_bookmark
- Appel de restart_commit()

Logique de clôture :

- Remise à zéro des pointeurs
- Clôture des curseurs/fichiers
- Appel de restart_close()

Description des fonctions API

restart_init :

Fonction d'initialisation pour le traitement batch avec tables.

Le processus rassemble des informations à partir des tables de contrôle de reprise

- Nombre total de traitements pour un programme et valeur de traitement affectée au processus en cours.
- Nombre d'enregistrements à itérer dans le curseur pilote avant exécution (LUW).
- Chaîne de démarrage - signet de la dernière exécution à utiliser pour la reprise ou chaîne nulle si le processus en cours est démarré pour la première fois et initialise l'enregistrement de la reprise (restart_program_status).
- Le statut du programme est modifié en "démarré" pour le premier traitement disponible.
- Les informations d'activité sont mises à jour : opérateur, processus, heure de début, etc. et tables de signets (restart_bookmark).
- Lors d'un premier démarrage, un enregistrement est inséré.
- Lors d'une reprise, les informations relatives à la chaîne de démarrage et au contexte d'application de la dernière exécution sont extraites.

restart_file_init :

Fonction d'initialisation pour le traitement batch avec fichiers. Elle est appelée depuis les modules de programme.

1 Le processus rassemble des informations à partir des tables de contrôle de reprise :
   - nombre d'enregistrements à lire depuis le fichier pour le traitement vectoriel et pour le cycle d'exécution
   - point de démarrage du fichier - signet de la dernière exécution à utiliser pour la reprise ou 0 pour un premier démarrage

2 Le processus initialise l'enregistrement de reprise (restart_program_status) :
   - le statut du programme est modifié en "démarré" pour le traitement en cours
   - les informations d'activité sont mises à jour : opérateur, processus, heure de début, etc.

3 Le processus initialise les tables de signets de reprise (restart_bookmark) :
   - lors d'un premier démarrage, un enregistrement est inséré.
   - lors d'une reprise, les informations relatives au point de démarrage du fichier et au contexte d'application de la dernière exécution sont extraites

restart_commit :

Fonction d'exécution des transactions traitées pour un nombre donné d'extractions de requêtes pilotes. Elle est appelée depuis les modules de programme.

Le processus met à jour les informations de la chaîne de démarrage restart_bookmark et de l'image d'application si un événement d'exécution est intervenu :

- le nombre actuel d'extractions de requêtes pilotes est supérieur ou égal au nombre maximal défini dans la table restart_program_status (et extrait dans la fonction restart_init)
- la chaîne de signet du dernier enregistrement traité est supérieure ou égale au maximum défini dans la table restart_program_status (et extraite dans la fonction restart_init)
- la chaîne de signet augmente le compteur
- la chaîne de signet définit la chaîne actuelle comme la dernière chaîne clé extraite

restart_file_commit :

Fonction d'exécution des transactions traitées après lecture d'un nombre de lignes depuis un fichier simple. Elle est appelée depuis les modules de programme.

Le processus met à jour la table restart_bookmark :

- start_string est défini à l'emplacement du pointeur de fichier pour la lecture actuelle du fichier simple
- l'image d'application est mise à jour avec les informations de contexte

restart_close :

Fonction de mise à jour des tables de reprise après la fin d'un programme.

Le processus détermine si le programme s'est terminé avec succès. Si le programme se termine avec succès :

- la table restart_program_status est mise à jour avec les informations de fin et le statut est remis à zéro
- l'enregistrement correspondant de la table restart_bookmark est supprimé
- la table restart_program_history contient une copie de l'enregistrement de la table restart_program_status inséré dans celle-ci
- le restart_program_status est réinitialisé

Si le programme se termine avec des erreurs

- les transactions sont exécutées une nouvelle fois
- la colonne program_status de la table restart_program_status est définie à "interrompu dans *" où * correspond à l'une des trois fonctions principales du batch : initialisation, traitement ou finalisation
- les modifications sont envoyées

parse_array_args :

Cette fonction décompose une chaîne en composants et place les résultats dans un tableau multidimensionnel. Elle est appelée uniquement dans le cadre de fonctions API et jamais dans les modules de programme.

Le processus utilise une chaîne pour l'analyse et un pointeur vers un tableau de caractères.

Le premier caractère de la chaîne utilisée est le séparateur.

restart_file_write :

Cette fonction insère les résultats des fichiers temporaires dans des fichiers de sortie définitifs lorsqu'un point d'exécution est atteint. Elle est appelée depuis les modules de programme.

restart_cat :

Cette fonction contient la logique d'insertion d'un fichier dans un autre. Elle est appelée uniquement dans le cadre des fonctions API de reprise/récupération et jamais directement dans les modules de programme.

En-têtes et bibliothèque de reprise

Les fichiers d'en-tête restart.h et std_err.h sont inclus dans retek.h pour pouvoir utiliser la fonctionnalité de reprise/récupération.

restart.h

Ce fichier d'en-tête de bibliothèque contient des constantes, des substitutions de macros et des définitions de variables globales externes ainsi que des prototypes de fonctions de reprise/récupération.

Les variables globales définies incluent :

- le nombre de traitements affectés au processus en cours
- la valeur du nombre maximal de traitements du processus en cours
  - pour les traitements avec tables, ce nombre est identique au nombre d'itérations de la requête pilote avant exécution
  - pour les traitements avec fichiers, ce nombre est identique au nombre de lignes lues depuis un fichier simple et traitées à l'aide d'un tableau structuré avant que l'exécution ne puisse intervenir
- le nombre actuel d'itérations de requêtes pilotes utilisées pour le traitement avec tables ou l'index du tableau actuel utilisé dans le traitement avec fichiers
- le nom affecté à l'unité de travail logique ou au programme par le programmeur. Ce nom est identique à la colonne restart_name des tables restart_program_status, restart_program_history et restart_bookmark

std_rest.h

Ce fichier d'en-tête de bibliothèque contient les déclarations de variables de reprise standard qui sont visibles dans les modules de programme.

Les définitions des variables incluses sont les suivantes :

- valeur de la chaîne concaténée de la clé de requête pilote extraite en cours de traitement
- valeur de la chaîne concaténée de la clé de requête pilote suivante dans le traitement
- message d'erreur transmis à la fonction restart_close et mis à jour dans restart_program_status
- chaîne concaténée des informations de contexte d'application, par exemple, les compteurs et accumulateurs
- nom du pilote de traitement, par exemple, rayon, magasin, entrepôt, etc.
- nombre total de traitements utilisés par ce programme
- pointeur à transmettre à la fonction d'initialisation pour détailler le nombre de valeurs de traitement

En-têtes et bibliothèques de reprise mis à jour

La bibliothèque de reprise/récupération RMS actuelle a été mise à jour avec la version 9, 10 et 11 afin d'optimiser la gestion, de simplifier le codage et d'améliorer les performances. Tout en préservant la fonctionnalité et le mécanisme actuels de la reprise/récupération batch, les améliorations et perfectionnements suivants ont été apportés :

- Organisation des variables globales associées à la reprise/récupération
- Possibilité pour le développeur des batch de contrôler entièrement les paramètres des variables de reprise/récupération transmis au cours de l'initialisation
- Retrait des fichiers d'écriture temporaires pour accélérer le processus d'exécution
- Déplacement d'un nombre plus important d'informations et de processus du code batch vers le code de bibliothèque
- Ajout d'un nombre plus important d'informations dans les tables de reprise/récupération à des fins d'optimisation

retek_2.h

Ce fichier d'en-tête de bibliothèque est inclus par tous les codes C de Retek et permet de centraliser toutes les insertions du système, les définitions de macros, les variables globales, les prototypes de fonctions et notamment, les structures à utiliser dans la nouvelle bibliothèque de reprise/récupération.

Les variables globales utilisées par l'ancienne bibliothèque de reprise/récupération sont toutes supprimées. À la place, chaque programme batch déclare les variables requises et appelle la fonction retek_init() pour les renseigner à partir des tables de reprise/récupération. Par conséquent, seules les variables suivantes sont déclarées :

- gi_no_commit : balise de l'option de ligne de commande NO_COMMIT (utilisée à des fins d'optimisation)
- gi_error_flag : balise d'erreur fatale
- gi_non_fatal_err_flag : balise d'erreur non fatale

En outre, la structure de rtk_file est définie pour la gestion de toutes les interfaces de fichiers associées à la fonction de reprise/récupération. Les fonctions d'activité sur la structure de fichiers sont également définies.

```
#define NOT_PAD            1000  /* Flag not to pad thread_val */
#define PAD                1001  /* Flag to pad thread_val at the
end */
#define TEMPLATE           1002  /* Flag to pad thread_val using
filename template */
#define MAX_FILENAME_LEN   50
typedef struct
{
   FILE* fp;                              /* File pointer */
   char  filename[MAX_FILENAME_LEN + 1];  /* Filename */
   int   pad_flag;   /* Flag whether to pad thread_val to filename
*/
} rtk_file;


int   set_filename(rtk_file* file_struct, char* file_name, int
pad_flag);
FILE* get_FILE(rtk_file* file_struct);
int   rtk_print(rtk_file* file_struct, char* format, ...);
int   rtk_seek(rtk_file* file_struct, long offset, int whence);
```

Les paramètres que retek_init() doit renseigner doivent être transmis dans un format connu de retek_init(). Une structure est définie ici à cet effet. Un tableau contenant les paramètres de ce type de structure est obligatoire dans chaque programme batch. Les autres conditions sont les suivantes :

Initialisation obligatoire à chaque programme batch.

- La longueur des noms, types et sous-types ne doit pas dépasser les définitions ici.
- Le type ne peut être que : "int", "uint", "long", "string" ou "rtk_file".
- Pour les types "int", "uint" ou "long", utilisez "" comme sous-type.
- Pour le type "string", le sous-type peut être uniquement "S" (chaîne de démarrage) sauf si la chaîne représente la valeur du traitement ou le nombre de traitements, dans ce cas utilisez “” comme sous-type ou "I" (chaîne d'image).
- Pour le type "rtk_file", le sous-type peut uniquement être "I" (saisie) ou "O" (sortie).

```
#define NULL_PARA_NAME        51
#define NULL_PARA_TYPE        21
#define NULL_PARA_SUB_TYPE    2
typedef struct
{
   char name[NULL_PARA_NAME];
   char type[NULL_PARA_TYPE];
   char sub_type[NULL_PARA_SUB_TYPE];
} init_parameter;
```

Nouvelles fonctions de reprise/récupération

Depuis la version 9.0, tous les nouveaux programmes batch sont codés à l'aide de nouvelles fonctions de reprise/récupération. Les programmes batch utilisant les anciennes fonctions API de reprise/récupération sont encore utilisés. Par conséquent, Retek met actuellement à jour deux ensembles de bibliothèques de reprise/récupération.

int retek_init(int num_args, init_parameter *parameter, ...)

retek_init initialise la reprise/récupération (pour traitements avec tables et fichiers) :

1 Transmet num_args comme nombre d'éléments dans le tableau init_parameter, puis dans le tableau init_parameter, puis les variables qu'un programme batch doit initialiser dans l'ordre et les types définis dans le tableau init_parameter. Toutes les variables int, uint et long doivent être transmises pour référence.

2 Extrait toutes les valeurs au niveau des variables globales et des modules des bases de données.

3 Initialise les enregistrements pour RESTART_PROGRAM_STATUS et RESTART_BOOKMARK.

4 Analyse les variables d'initialisation définies par l'utilisateur (variable arg list).

5 Renvoie NO_THREAD_AVAILABLE s'il n'existe aucun enregistrement qualifié dans RESTART_CONTROL ou RESTART_PROGRAM_STATUS.

6 Exécution.

int retek_commit(int num_args, ...)

retek_commit effectue une vérification et une exécution, le cas échéant (pour les traitements avec tables et fichiers) :

1 Transmet num_args, puis les variables pour start_string en premier et celles de la chaîne d'image (le cas échant) ensuite. num_args représente le nombre total de variables pour ces deux groupes. Ce sont toutes des variables de type chaîne transmises dans le même ordre que dans retek_init() ;

2 Concatène start_string, soit par transmission de variables (traitement avec tables), soit depuis la fonction ftell des pointeurs de fichiers d'entrée (traitement avec fichiers) ;

3 Vérifie si le point d'exécution est atteint (vérification du compteur et si traitement avec tables, comparaison des chaînes de démarrage) ;

4 Si le point d'exécution est atteint, image_string est concaténé à partir des variables transmises (le cas échéant) et la fonction internal_commit() est appelée pour extraire out_file_string et mettre à jour la table RESTART_BOOKMARK ;

5 Lors du traitement avec tables, pl_current_count est incrémenté et ps_cur_string est mis à jour.

int commit_point_reached(int num_args, ...)

commit_point_reached vérifie si le point d'exécution a été atteint (pour les traitements avec tables et fichiers). La différence entre cette fonction et la vérification dans retek_commit() réside dans le fait que pl_current_count et ps_cur_string ne sont pas mis à jour ici. Cette fonction de vérification est conçue pour être utilisée avec retek_force_commit(), et la logique permettant d'assurer l'intégrité de la LUW existe dans le programme batch de l'utilisateur. Elle peut également être utilisée avec retek_commit() pour d'autres traitements au moment de l'exécution.

1 Transmet num_args, puis toutes les variables de type chaîne pour start_string dans le même ordre que dans retek_init(). num_args représente le nombre de variables pour start_string. S'il n'existe aucun start_string (comme dans traitement avec fichiers) la transmission est NULLE.

2 Pour les traitements avec tables, si pl_curren_count atteint pl_max_counter et si la chaîne de signet nouvellement concaténée est différente de ps_cur_string, 1 est renvoyé ; dans le cas contraire, 0 est renvoyé.

3 Pour les traitements avec fichiers, si pl_curren_count atteint pl_max_counter, 1 est renvoyé ; dans le cas contraire, 0 est renvoyé.

int retek_force_commit(int num_args, ...)

retek_force_commit s'exécute toujours (pour traitements avec tables et fichiers) ;

1 Transmet num_args, puis les variables pour start_string en premier et celles de la chaîne d'image (le cas échant) ensuite. num_args représente le nombre total de variables pour ces deux groupes. Ce sont toutes des variables de type chaîne transmises dans le même ordre que dans retek_init() ;

2 Concatène start_string, soit par transmission de variables (traitements avec tables), soit depuis la fonction ftell des pointeurs de fichiers d'entrée (traitements avec fichiers) ;

3 image_string est concaténé à partir des variables transmises (le cas échéant) et la fonction internal_commit() est appelée pour extraire out_file_string et mettre à jour la table RESTART_BOOKMARK ;

4 Lors du traitement avec tables, pl_current_count est incrémenté et ps_cur_string est mis à jour.

int retek_close(void)

retek_close clôt la reprise/récupération (pour les traitements avec tables et fichiers) :

1 Si l'option de ligne de commande gi_error_flag ou NO_COMMIT est VRAIE, tous les changements des bases de données sont exécutés à nouveau.

2 Mise à jour de la table RESTART_PROGRAM_STATUS en fonction de gi_error_flag.

3 S'il n'existe aucun gi_error_flag, un enregistrement est inséré dans la table RESTART_PROGRAM_HISTORY avec des informations extraites des tables RESTART_CONTROL, RESTART_PROGRAM_BOOKMARK et RESTART_PROGRAM_STATUS.

4 S'il n'existe aucun gi_error_flag, l'enregistrement RESTART_BOOKMARK est supprimé.

5 Exécution.

6 Clôt toutes les suites de données des fichiers ouverts.

Int retek_refresh_thread(void)

Actualise le traitement d'un programme pour pouvoir l'exécuter une nouvelle fois.

1 Met à jour l'enregistrement RESTART_PROGRAM_STATUS pour que le statut PROGRAM_STATUS du programme en cours soit "prêt pour démarrage".

2 Supprime tous les enregistrements RESTART_BOOKMARK pour le programme en cours.

3 Exécution.

void increment_current_count(void)

increment_current_count augmente pl_current_count de 1.

 **Remarque :** appelé depuis get_record() de intrface.pc pour E/S avec fichiers.

int parse_name_for_thread_val(char* name)

parse_name_for_thread_val analyse la valeur du traitement à partir de l'extension du nom de fichier défini.

int is_new_start(void)

is_new_start vérifie si l'exécution en cours est un nouveau démarrage ; si tel est le cas, 1 est renvoyé, dans le cas contraire, 0 est renvoyé.

Seuil d'exécution avec requêtes

Les capacités de reprise sont centrées autour de l'unité de travail logique d'un programme (LUW). Un programme batch exécute des transactions et active des points d'exécution en fonction de la LUW. Une LUW est composée d'une clé de transaction (telle que article-magasin) et d'un nombre maximal d'exécutions. Les événements d'exécution interviennent après le traitement d'un nombre donné de clés de transaction. Au moment de l'exécution, les informations de la clé nécessaires à la reprise sont stockées dans la table de reprise. Lors d'une exception résolue ou non, les transactions sont renvoyées vers le dernier point d'exécution. Au moment de la reprise, les informations de la clé de reprise sont extraites des tables pour permettre au processus de continuer le traitement des données non traitées.

Chapitre 3 – Multi-traitements Pro*C

L'exécution de plusieurs instances d'un programme donné peut se faire à l'aide de "traitements". Des curseurs pilotes doivent être divisés en segments discrets de données exécutés par différents traitements. Ce traitement est réalisé à l'aide de procédures stockées qui séparent les mécanismes de traitements (par exemple, rayons ou magasins) en traitements particuliers pour une valeur donnée (par exemple, rayon 1001) et du nombre total de traitements pour un processus donné.

L'exécution avec fichiers n'utilise pas réellement de “traitement“. Le même fichier de données ne sera jamais traité par plusieurs processus. Le multi-traitements est réalisé en divisant les données en fichiers distincts qui seront traités par un processus distinct. La valeur du traitement est reportée dans le fichier d'entrée. Cela est nécessaire pour s'assurer que les informations appropriées peuvent être associées au fichier correspondant dans l'éventualité d'une reprise.

RMS a une longueur de stockage de 10 chiffres. Par conséquent, les valeurs de traitement pouvant être fonction du numéro de stockage, doivent pouvoir comprendre 10 chiffres également. Dans la mesure où les valeurs de traitement sont déclarées comme des variables "C" de type int (long), elles sont limitées à 9 chiffres par le système.

Cela ne signifie pas que vous ne pouvez pas utiliser de numéros de stockage à 10 chiffres. Cela signifie que si vous utilisez des numéros de stockage à 10 chiffres, vous ne pouvez pas les utiliser comme valeurs de traitement.

Description de l'exécution de traitements

L'utilisation de plusieurs traitements ou processus dans le traitement batch de Retek améliore l'efficacité et réduit la durée de traitement. Le processus de traitement a fourni un maximum de flexibilité à l'utilisateur final en ce qui concerne la définition du nombre de traitements selon lequel un programme doit être divisé.

À l'origine, la fonction de traitement devait être utilisée directement dans les requêtes pilotes. Cependant, cette méthode s'est avérée très lente et donc inutilisable. Au lieu d'utiliser l'appel de fonctions directement dans les requêtes pilotes, l'application effectue la jointure des tables de requêtes pilotes en une seule vue (par exemple, v_restart_store) qui inclut la fonction.

Fonction de traitement avec requêtes

Une procédure stockée a été créée pour déterminer les valeurs de traitement. restart_thread_return renvoie une valeur de traitement dérivée de la valeur numérique d'un pilote, telle que le nombre de rayons, et le nombre total de traitements dans un processus donné. Les clients doivent être en mesure de déterminer le meilleur algorithme pour leur application et si une méthode différente de segmentation de données est nécessaire, la fonction restart_thread_return peut être modifiée ou une autre fonction peut être utilisée dans les vues contenant la fonction.

Actuellement la fonction restart_thread_return est une routine de module très simple :

```
CREATE OR REPLACE FUNCTION RESTART_THREAD_RETURN (in_unit_value
NUMBER,
                          in_total_threads NUMBER)
  RETURN NUMBER IS
  ret_val NUMBER;
BEGIN
  ret_val := MOD(ABS(in_unit_value),in_total_threads) + 1;
  RETURN ret_val;
END;
```

Vue de reprise avec requêtes

Chaque vue de reprise contient quatre éléments :

- le nom du mécanisme de traitement, driver_name
- le nombre total de traitements dans un groupement, num_threads
- la valeur du mécanisme pilote, driver_value
- la valeur du traitement pour cette combinaison donnée de driver_name, num_threads et de valeur de pilote, thread_val

La vue est basée sur la table restart_control et une table d'informations, telle que DEPS ou STORES. Une ligne existe dans la vue pour chaque valeur de pilote et chaque total de valeur de traitements. Par conséquent, si un détaillant utilise toujours le même nombre de traitements pour un pilote donné (rayon, magasin, etc.), la vue est relativement petite. Par exemple, si tous les programmes d'un détaillant, traités par rayon, contiennent un nombre total de 5 traitements, la vue ne contient qu'une seule valeur pour chaque rayon. Par exemple, si le nombre total de rayons est 10, v_restart_dept contiendra 10 lignes. Cependant, si le détaillant souhaite que l'un des programmes contienne 10 traitements, il y aura 2 lignes pour chaque rayon : une pour 5 traitements et une autre pour 10 traitements (par exemple, si le nombre total de rayons est de 10, v_restart_dept contiendra 20 lignes). Bien évidemment, il est recommandé aux détaillants de réduire au maximum le nombre total de traitements pour un pilote de traitement afin de réduire la portée de la jointure des tables du curseur pilote de la vue.

Voici un exemple dans lequel la même valeur de pilote peut résulter en différentes valeurs de traitements. Cet exemple utilise la fonction restart_thread_return actuellement écrite pour dériver les valeurs de traitement.

DRIVER_NAME	NUM_THREADS	DRIVER_VAL	THREAD_VAL
DEPT	1	101	1
DEPT	2	101	2
DEPT	3	101	3
DEPT	4	101	2
DEPT	5	101	2
DEPT	6	101	6
DEPT	7	101	4

Voici un exemple de l'aspect d'une répartition de magasins avec 10 magasins et 5 traitements :

DRIVER_NAME	NUM_THREADS	DRIVER_VAL	THREAD_VAL
STORE	5	1	2
STORE	5	2	3
STORE	5	3	4
STORE	5	4	5
STORE	5	5	1
STORE	5	6	2
STORE	5	7	3
STORE	5	8	4
STORE	5	9	5
STORE	5	10	1

Syntaxe :

Voici un exemple de la syntaxe nécessaire à la création de la vue pour la jointure multi-traitements, créée avec script (reportez-vous à la section sur les traitements pour plus d'informations sur la fonction restart_thread_return) :

```
create or replace view v_restart_store as
 select rc.driver_name driver_name,
     rc.num_threads num_threads,
     s.store driver_value,
     restart_thread_return(s.store, rc.num_threads) thread_val
  from restart_control rc, store s
  where rc.driver_name = 'STORE'
```

Retek Sales Audit ou ReSA (audit des ventes Retek) utilise un schéma de traitement différent. Dans la mesure où ReSA doit être exécuté 24 heures sur 24 et 7 jours sur 7, il n'existe aucun écran batch. Cela signifie que des programmes batch peuvent être exécutés lorsque des utilisateurs sont en ligne. ReSA a résolu ce problème de conflit en créant un mécanisme de verrouillage pour les données organisées par jour magasin. Ces verrouillages fournissent un schéma de traitement naturel. Les programmes qui utilisent toutes les données des jours magasin essaient de verrouiller d'abord le jour magasin. Si le verrouillage échoue, le programme passe tout simplement au jour magasin suivant. Cela a pour effet d'équilibrer automatiquement la charge de travail entre tous les programmes en cours d'exécution.

Gestion du schéma de traitement

Tous les noms de programmes sont stockés dans la table restart_control avec leur description fonctionnelle, le pilote de requête (rayon, magasin, famille, etc) et le nombre associé de traitements défini par l'utilisateur. L'utilisateur doit pouvoir naviguer dans tous les programmes pour consulter le nom, la description et le pilote de requête et si la balise update_allowed est définie sur vrai, modifier le nombre de traitements (la mise à jour est définie sur vrai).

Avec fichiers

Les exécutions avec fichiers n'utilisent pas vraiment plusieurs traitements. Par conséquent, le nombre de traitements défini dans la table restart_control doit toujours être égal à un. Cependant, un enregistrement restart_program_status doit être créé pour chaque fichier d'entrée traité pour le module du programme. Par ailleurs, la valeur de traitement affectée doit être contenue dans le nom du fichier d'entrée. La fonction restart_parse_name incluse dans le module de programme analyse la valeur de traitement à partir du nom du programme et l'utilise pour déterminer la disponibilité et les conditions de reprise de la table restart_program_status.

Reportez-vous au début de la section consacrée au multi-traitements pour une présentation des limites de l'utilisation de valeurs de traitement importantes (supérieures à 9 chiffres).

Avec requêtes

Lorsque le nombre de traitements est modifié dans la table restart_control, le formulaire doit d'abord vérifier qu'aucun enregistrement pour ce programme n'est en cours de traitement dans la table restart_program_status (c'est-à-dire que tous les enregistrements = "Terminé"). Le programme doit insérer ou supprimer des lignes selon que le nouveau nombre de traitements est supérieur ou inférieur à l'ancien nombre. Si le nouveau nombre est inférieur à l'ancien, tous les enregistrements pour le program_name dont le nombre de traitements est supérieur au nouveau seront supprimés. Si le nouveau nombre est supérieur à l'ancien, de nouvelles lignes seront insérées. Un nouvel enregistrement est inséré pour chaque combinaison restart_name/thread_val.

Par exemple, si le programme batch SALDLY voit son nombre de traitements passer de 2 à 3, une ligne supplémentaire (3) est ajoutée à la table restart_program_status. De même, si le nombre de traitements est réduit à 1 dans cet exemple, les lignes 2 et 3 sont supprimées.

Table restart_program_status originale :

ligne n°	restart_name	thread_val	program_name	etc…
1	WinSal -main	1	WinSal	…
2	WinSal -main	2	WinSal	…

Table restart_program_status après insertion :

ligne n°	restart_name	thread_val	program_name	etc…
1	WinSal -main	1	WinSal	…
2	WinSal -main	2	WinSal	…
3	WinSal -main	3	WinSal	…

Table restart_program_status après suppression :

ligne n°	restart_name	thread_val	program_name	etc…
1	WinSal -main	1	WinSal	…

Les utilisateurs doivent également être en mesure de modifier la colonne commit_max_ctr de la table restart_program_status. Ils peuvent ainsi contrôler le nombre d'itérations dans la requête pilote ou le nombre de lignes lu à partir d'un fichier simple qui détermine l'unité de travail logique (LUW).

Gestion des batchs

Les utilisateurs doivent être en mesure de consulter le statut de tous les enregistrements de la table restart_program_status. Il s'agit de l'emplacement où l'utilisateur peut consulter les messages d'erreur des programmes interrompus ainsi que les statistiques et l'historique des exécutions de batch. Les seuls champs modifiables sont program_status et restart_flag. L'utilisateur doit pouvoir redéfinir le champ restart_flag de "N" à "Y" dans les enregistrements dont le statut est interrompu, redéfinir les enregistrements démarrées à interrompu en cas d'interruption (fin inhabituelle) et tous les enregistrements en cas de restauration à partir d'une inscription/nouvelle exécution de tous les batch.

Planification et initialisation du batch de reprise

Avant d'exécuter tout batch avec la logique de reprise/récupération, un programme d'initialisation doit être exécuté pour mettre à jour le statut dans la table restart_program_status. Ce programme doit mettre à jour le program_status sur "prêt pour démarrage" lorsque le program_status d'un enregistrement est défini à "terminé". Tous les programmes qui ont échoué lors de la dernière exécution des batch restent donc inchangés.

Pré- et post-traitements

En raison de la nature de l'algorithme de traitement, les programmes individuels doivent exécuter un pré- ou post-programme pour initialiser les variables ou les fichiers avant l'exécution de tout traitement ou la mise à jour des données finales après l'exécution de tous les traitements. La décision a été prise de créer des pré- et post-programmes dans ce cas, plutôt que de laisser la logique de reprise/récupération décider si le traitement en cours d'exécution est le premier ou le dernier d'un programme donné.

Chapitre 4 – Traitement vectoriel Pro*C

L'architecture des batch de Retek utilise le traitement vectoriel pour améliorer les performances lorsque cela est possible. Au lieu de traiter des instructions SQL à l'aide de données scalaires, les données sont regroupées en tableaux et utilisées comme variables de liaison dans les instructions SQL. Cette méthode permet d'améliorer les performances en réduisant le trafic sur le serveur/client et le réseau.

Le traitement vectoriel est utilisé pour sélectionner, insérer et mettre à jour les instructions. En règle générale, Retek ne définit pas les tailles des tableaux de façon statistique, mais utilise la variable d'exécution de reprise maximale comme multiple de dimensionnement. Les utilisateurs doivent se souvenir de cela lorsqu'ils définissent le nombre maximal d'exécutions dans le système.

Un facteur important à prendre en compte lors de l'utilisation du traitement vectoriel réside dans le fait qu'Oracle ne permet pas d'activité vectorielle sur plus de 32 000 enregistrements à la fois. Les bibliothèques de reprise/récupération de Retek ont été mises à jour pour définir des macros pour la valeur suivante : MAX_ORACLE_ARRAY_SIZE.

Tous les programmes batch qui utilisent le traitement vectoriel doivent limiter la taille de leurs tableaux à la valeur MAX_ORACLE_ARRAY_SIZE.

Si le nombre maximal d'exécutions est utilisé pour la taille du traitement vectoriel, vérifiez-le après l'appel à la fonction restart_init() et, le cas échéant, redéfinissez-le à la valeur maximale si elle est supérieure. Si la fonction retek_init() est utilisée pour l'initialisation, vérifiez le nombre maximal d'exécutions renvoyé et redéfinissez-le à la taille maximale si elle est supérieure. Dans le cas de rsetek_init(), redéfinissez le nombre maximal d'exécutions internes de la bibliothèque en appelant la variable externe de type "int" limit_commit_max_ctr (int sans signe new_max_ctr).

Si d'autres variables sont utilisées pour le dimensionnement du traitement vectoriel, l'étape en cours du traitement vectoriel doit être encapsulée dans une boucle d'appel qui effectue les activités vectorielles dans des sous-segments des tableaux où chaque sous-segment correspond au maximum à la valeur MAX_ORACLE_ARRAY_SIZE. Actuellement, tous les programmes batch de Retek sont mis en œuvre de cette façon.

Chapitre 5 – Formats d'entrée et de sortie Pro*C

Les programmes batch de Retek utilisent les saisies à la fois des tables et des fichiers simples. Par ailleurs, les traitements peuvent entraîner la modification des structures des données et l'écriture des données de sortie. Le traitement E/S avec fichiers joue le rôle d'interface entre Retek et les systèmes externes.

Présentation générale de l'interface

Pour simplifier les conditions d'interface, Retek exige que toutes les transactions entrantes et sortantes sur fichiers utilisent des présentations de fichiers standard. Il existe deux types de présentations de fichiers, présentation de détails uniquement et présentation de détails principaux, qui sont décrits ci-après.

Une interface API existe au sein de Retek pour simplifier le codage et la gestion des fichiers d'entrée. L'interface API fournit des fonctionnalités de lecture des fichiers d'entrée, assure l'intégrité de la présentation des fichiers et écrit et gère les fichiers des transactions rejetées.

Présentations des fichiers standard

La bibliothèque d'interface RMS prend en charge deux présentations de fichiers standard ; une pour le traitement des détails principaux et une pour le traitement des détails uniquement. Les sous-détails ne sont pas pris en charge par les fonctions de la bibliothèque d'interface de base RMS.

Un code d'identification à 5 caractères ou un type d'enregistrement identifie tous les enregistrements d'un fichier E/S, quel que soit le type de fichier. Les types d'enregistrements valides incluent les valeurs suivantes :

- FHEAD—En-tête de fichier
- FDETL—Détail de fichier
- FTAIL—En-queue de fichier
- THEAD—En-tête de transaction
- TDETL—Détail de transaction
- TTAIL—En-queue de transaction

Chaque ligne du fichier doit commencer par le code du type d'enregistrement suivi d'un ID de l'enregistrement à 10 caractères.

Fichiers de détails uniquement

Les présentations de fichiers possèdent un enregistrement d'en-tête de fichier standard, un enregistrement détaillé pour chaque transaction à traiter ainsi qu'un enregistrement d'en-queue de fichier. Les types d'enregistrement valides sont FHEAD, FDETL et FTAIL.

Exemple :

```
FHEAD0000000000STKU1996010100000019960929
FDETL0000000001SKU100000040000011011
FDETL0000000001SKU100000050003002001
FDETL0000000001SKU100000050003002001
FTAIL00000000020000000003
```

Fichiers de détails principaux

Les présentations de fichiers possèdent un enregistrement d'en-tête de fichier standard, un ensemble d'enregistrements pour chaque transaction à traiter et un enregistrement d'en-queue de fichier. Cet ensemble de transactions contient un enregistrement d'en-tête d'ensemble de transactions, les détails de l'ensemble de transactions pour les détails au sein de la transaction et un enregistrement d'en-queue de transaction. Les types d'enregistrements valides sont FHEAD, THEAD, TDETL, TTAIL et FTAIL.

Exemple :

```
FHEAD0000000001RTV 19960908172000
THEAD00000000020000000000123419960909120200000000000003R
TDETL00000000030000000000123400000 1SKU10000012
TTAIL0000000004000001
THEAD00000000050000000000123419960909120200121572013 1R
TDETL00000000060000000000123400000 1UPC400100002667
TDETL00000000070000000000123400000 1UPC400100002643 0
TTAIL0000000008000002
FTAIL00000000090000000007
```

Nom de l'enregistrement	Nom du champ	Type de champ	Valeur par défaut	Description
En-tête de fichier	Descripteur d'enregistrement de type de fichier	Char.(5)	FHEAD	Identifie le type d'enregistrement de fichier
	Identifiant de ligne de fichier	Number (10)	Défini par le système externe	Numéro de la ligne du fichier actuel

Nom de l'enregistrement	Nom du champ	Type de champ	Valeur par défaut	Description
	Définition du type de fichier	Char.(4)	s/o	Identifie le type de transaction
	Date de création du fichier	Date	Date de création	Date d'écriture du fichier par le système externe.
En-tête de transaction	Descripteur d'enregistrement de type de fichier	Char.(5)	THEAD	Identifie le type d'enregistrement de fichier
	Identifiant de ligne de fichier	Number (10)	Défini par le système externe	Numéro de la ligne du fichier actuel
	Numéro contrôle ensemble transactions	Char.(14)	Défini par le système externe	Utilisé pour assurer une vérification unique des transactions.
	Date de la transaction	Char.(14)	Défini par le système externe	Date de transaction créée dans le système externe.
Détail de transaction	Descripteur d'enregistrement de type de fichier	Char.(5)	TDETL	Identifie le type d'enregistrement de fichier
	Identifiant de ligne de fichier	Number (10)	Défini par le système externe	Numéro de la ligne du fichier actuel
	Numéro contrôle ensemble transactions	Char.(14)	Défini par le système externe	Utilisé pour assurer une vérification unique des transactions.
	Numéro de la séquence du détail	Char.(6)	Défini par le système externe	Numéro séquentiel affecté pour détailler les enregistrements d'une transaction.
En-queue de transaction	Descripteur d'enregistrement de type de fichier	Char.(5)	TTAIL	Identifie le type d'enregistrement de fichier

Nom de l'enregistrement	Nom du champ	Type de champ	Valeur par défaut	Description
	Identifiant de ligne de fichier	Number (10)	Défini par le système externe	Numéro de la ligne du fichier actuel
	Comptage des lignes de détail de la transaction	Number (6)	Total des lignes de détails	Nombre de lignes de détails dans une transaction.
Fin de fichier	Descripteur d'enregistrement de type de fichier	Char.(5)	FTAIL	Identifie le type d'enregistrement de fichier
	Identifiant de ligne de fichier	Number (10)	Défini par le système externe	Numéro de la ligne du fichier actuel
	Nombre total de lignes transaction	Number (10)	Total de toutes les lignes de transaction	Toutes les lignes du fichier moins les enregistrements d'en-tête et d'en-queue de fichier

Echange de données informatisé (EDI)

Apparus dans la version 7.0, les fichiers EDI utilisés ou créés par RMS ont un format générique : RMS ne prend plus en charge les normes EDI particulières. En traitant les entrées et sorties EDI au format générique, RMS ne se limite plus à une seule norme, ce qui permet aux clients de Retek d'utiliser au mieux toutes les normes qu'ils ont choisies. La conversion du format des fichiers d'entrée et de sortie EDI de n'importe quel format et vers n'importe quel format à l'aide d'un logiciel tiers est une "méthode courante".

Dans le passé, les transactions EDI dans RMS étaient conformes aux normes ASC X12/VICS (version 3040) et ANA/TRADACOMS. Elles adoptent maintenant un format qui applique les normes d'interface de fichier RMS. Les fichiers entrants et sortants sont écrits dans une présentation à champs fixes avec des enregistrements standard d'en-tête et d'en-queue de fichier. Les informations de transaction sont incluses dans des enregistrements de présentation principaux/de détails ou de détails uniquement. Les présentations correspondent aux fichiers d'interface utilisés ailleurs dans RMS.

Les processus batch EDI de RMS écrivent les fichiers de transactions sortantes dans un format de présentation générique, ces fichiers sont ensuite convertis par le logiciel tiers pour appliquer la norme requise par chaque partenaire commercial. Les versions après conversion sont transmises au partenaire commercial. Les transactions entrantes doivent être mises en forme par le partenaire commercial dans une norme prédéfinie, transmises puis converties par le logiciel de conversion du client Retek en présentation de fichier générique. Le fichier générique est utilisé comme fichier d'entrée pour le traitement batch EDI de RMS.

Retek ne peut plus continuer à gérer des codes qui prennent en charge des normes EDI particulières. Il existe plusieurs normes valides utilisées par les fournisseurs et les détaillants. Par ailleurs, ces normes existent en plusieurs versions. La majorité des détaillants utilisent déjà des logiciels pour associer et convertir les transactions EDI dans la norme ou la version requise. Il existe d'excellents logiciels tiers, tels que le convertisseur Gentran™ de Sterling Software, qui convertit de façon efficace les transactions entrantes et sortantes aux formats requis. L'utilisation de logiciels tiers n'est pas seulement une pratique commune, mais également la meilleure méthode utilisée à ce jour par les détaillants.

Chapitre 6 – Architecture RETL pour système RMS-RDF

Le système RMS travaille avec la structure RETL (Retek Extract Transform and Load). Cette architecture optimise un outil haute performance de traitement des données qui permet aux processus batch de bases de données de tirer parti des capacités de traitement parallèles.

La structure RETL exécute et analyse les opérateurs valides qui composent les scripts XML.

Ce chapitre offre une présentation du traitement RMS RETL. Vous trouverez des informations supplémentaires sur l'outil RETL dans le tout dernier Guide du programmateur RETL.

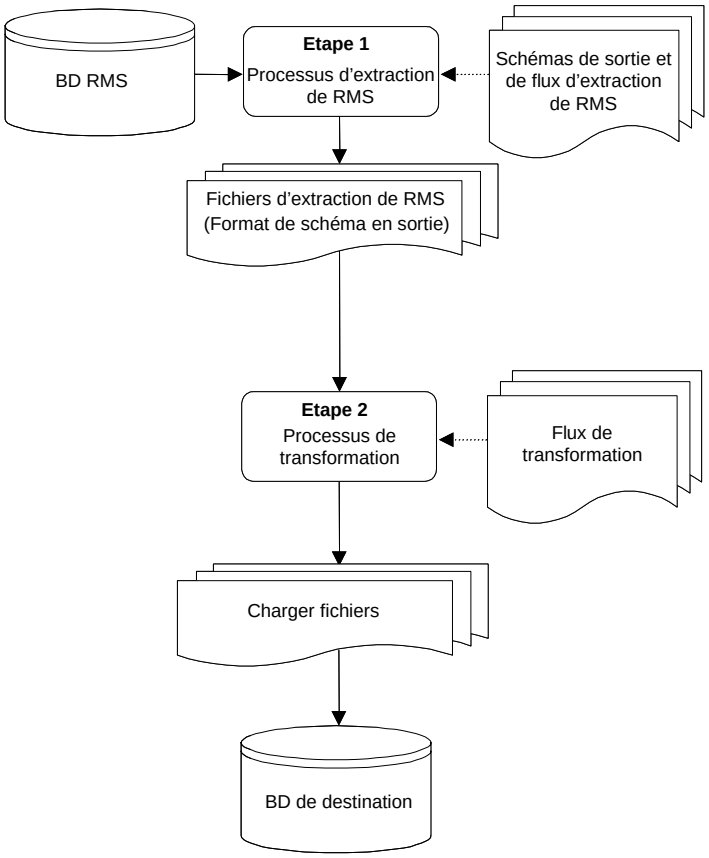
Concept architectural

Le schéma ci-dessous illustre l'architecture de traitement de l'extraction. Plutôt que de gérer les captures de modification à mesure qu'elles se produisent dans le système source au cours de la journée, le processus extrait les données actuelles du système source. Les données extraites sont sorties sur des fichiers plats. Ces fichiers plats sont alors disponibles pour des produits tels que RDW (Retek Data Warehouse) et RDF (Retek Demand Forecasting, Prévision de la demande Retek).

Le système cible (RDW ou RDF par exemple) possède sa propre méthode de finalisation des transformations et de chargement des données nécessaires, qui peuvent passer par un traitement supplémentaire dans le nouvel environnement.

L'architecture est basée sur deux étapes distinctes, illustrées par le schéma ci-dessous. Etape 1 : extraction de la base de données RMS à l'aide de flux spécifiquement définis. On obtient en sortie des fichiers de données au format schéma spécifiquement défini. Cette étape ne comprend aucun code spécifique à la destination.

Etape 2 : introduit un flux spécifique à la destination. Dans ce cas, les flux destinés au produit RDF/RPAS transforment les données de manière à ce que le système RDF puisse importer correctement les données.



Les deux étapes du traitement RETL

Chapitre 7 – Présentation du programme RETL pour l'interface RMS-RDF

Ce chapitre récapitule les caractéristiques du programme RETL utilisées pour les extractions RMS (RMSE). Vous trouverez des informations supplémentaires sur l'outil RETL dans le tout dernier Guide du programmateur RETL.

Remarque : Dans cette section, certains exemples font référence à des programmes RETL n'ayant aucun lien avec le système RMS. Les références à ces programmes ne sont données qu'à titre d'indication.

Installation

Sélectionnez le répertoire où vous souhaitez installer RMS ETL. Ce répertoire (aussi intitulé MMHOME) est l'emplacement d'où les fichiers RMS ETL sont extraits.

L'arborescence de codes ci-dessous est utilisée pour la structure RETL au cours des extractions, transformations et chargements. Il y est fait référence dans cette documentation.

```
<base directory (MMHOME)>
        /data
        /error
        /log
        /rfx
              /bookmark
              /etc
              /lib
              /schema
              /src
```

Configuration

RETL

Avant de configurer et d'exécuter RMS ETL, installez la version 11.2 de RETL ou une version supérieure (requise pour exécuter RMS ETL). Exécutez le script “verify_retl” (inclus dans l'installation de RETL) pour vous assurer du bon fonctionnement de RETL avant de poursuivre.

Utilisateur et autorisations RETL

RMS ETL est installé et exécuté en tant qu'utilisateur RETL. Par ailleurs, les autorisations sont configurées suivant le Guide du programmateur RETL. RMS ETL lit les données, crée, supprime et met à jour les tables. Si ces autorisations ne sont pas configurées correctement, les extractions échoueront.

Variables d'environnement

Consultez le Guide du programmateur RETL pour connaître les variables d'environnement RETL à configurer pour votre version de RETL. Vous devrez définir MMHOME comme répertoire de base pour RMS RETL. Il s'agit du répertoire de plus haut niveau que vous ayez sélectionné au cours de l'installation (reportez-vous à la section "Installation" ci-dessus). Dans votre fichier .kshrc, vous devez ajouter une ligne analogue à celle ci-dessous :

```
export MMHOME=<base directory for RMS ETL>
```

paramètres rmse_config.env

Certaines variables doivent être modifiées en fonction de vos paramètres locaux :

Par exemple :

```
export DBNAME=int9i
export RMS_OWNER=steffej_rms1011
export BA_OWNER=rmsint1011
```

Vous devez configurer la variable d'environnement PASSWORD soit dans le fichier rmse_config.env, soit dans .kshrc, soit à un autre emplacement pouvant être consulté. Dans l'exemple ci-dessous, si la ligne est ajoutée à rmse_config.env, le mot de passe ‘mypasswd’ servira à ouvrir une session sur la base de données :

```
export PASSWORD=mypasswd
```

Pour le système RMSE, veillez à revoir les paramètres d'environnement du fichier rmse_config.env avant d'exécuter des modules batch.

Etapes à suivre pour configurer RETL

1 Ouvrez une session sur le serveur Unix avec un compte Unix pouvant exécuter les scripts RETL.

2 Changez les répertoires en : `<base_directory>/rfx/etc`.

3 Modifiez le script rmse_config.env :

   a Donnez à la variable DBNAME le nom de la base de données RMS.

   b Donnez à la variable RMS_OWNER le nom d'utilisateur du propriétaire du schéma RMS.

   c Donnez à la variable BA_OWNER le nom d'utilisateur de l'utilisateur du batch RMSE.

Code de retour au programme

Les programmes RETL utilisent un code de retour pour indiquer une exécution réussie. Si l'exécution du programme a réussi, le code renvoyé est zéro (0). Si l'exécution du programme a échoué, il s'agira d'une valeur non nulle.

Fichiers de contrôle du statut du programme

Pour éviter qu'un programme ne s'exécute alors que le même programme est déjà en cours d'exécution avec le même groupe de données, le code RMSE utilise un fichier de contrôle du statut du programme. Au début de chaque module, rmse_config.env est exécuté. Il vérifie l'existence du fichier de contrôle du statut du programme. Si le fichier existe, le message "${PROGRAM_NAME} has already started" est reporté et le module se ferme. Si le fichier n'existe pas, un fichier de contrôle du statut du programme est créé et le module s'exécute.

Si le module échoue à un moment donné, le fichier de contrôle du statut du programme n'est pas supprimé. C'est à l'utilisateur de le supprimer avant d'exécuter de nouveau le module.

Conventions de dénomination des fichiers

La convention de dénomination du fichier de contrôle de statut du programme permet d'exécuter un programme, dont l'entrée est un fichier texte, plusieurs fois et en même temps avec différents fichiers.

Le nom et le répertoire du fichier de contrôle du statut du programme est défini dans le fichier de configuration (rmse_config.env). Le répertoire par défaut est $MMHOME/error. La convention de dénomination du fichier de contrôle du statut du programme attribue par défaut le nom de fichier suivant, séparé par des points :

- Nom du programme
- "statut"
- Date virtuelle d'activité à laquelle le module a été exécuté

Par exemple, le fichier de contrôle de statut du programme invildex est intitulé comme suit pour l'exécution de batch du 5 janvier 2001 :

```
$MMHOME/error/rmse_daily_sales.status.20010105
```

Reprise et récupération

RETL traitant tous les enregistrements comme un tout (et non pas individuellement), la méthode de reprise et de récupération doit être différente de celle utilisée pour Pro*C. Le processus de reprise et de récupération a deux objectifs :

1 Eviter la perte de données due à la défaillance du programme ou de la base de données.

2 Améliorer les performances, lors d'une reprise suite à une défaillance du programme ou de la base de données, en limitant le volume de données devant être retraitées.

Les modules RMS Extract (RMSE) extraient les données d'une base de données de transaction ou d'un fichier texte sources et inscrivent les données dans un fichier texte. Les modules RMS Load (RMSL) importent les données de fichiers plats, effectuent des transformations si nécessaire, puis chargent les données dans les tables RMS appropriées.

La plupart des modules utilisent un seul flux RETL et ne requièrent ni reprise ni récupération. Si pour quelque raison que ce soit, le processus d'extraction échoue, il est possible de régler le problème et d'exécuter de nouveau le processus depuis le début sans perte de données. Pour un module utilisant un fichier texte comme entrée, il existe deux solutions permettant d'exécuter de nouveau le module depuis le début :

1 Exécuter de nouveau le module avec l'intégralité du fichier d'entrée.

2 Exécuter de nouveau le module uniquement avec les enregistrements n'ayant pas été traités correctement la première fois et enchaîner le fichier obtenu avec le fichier de sortie de la première exécution.

Pour limiter le volume de données devant être retraitées, il existe des modules plus complexes qui requièrent l'usage de plusieurs flux RETL et qui utilisent une méthode de signets pour la reprise et la récupération. Cette méthode permet de relancer le module à partir du dernier point d'exécution réussi et de terminer l'exécution. La méthode de reprise/récupération par signets introduit un signet qui indique l'étape suivante du processus devant être exécutée. Pour chaque étape, le signet est inscrit dans un fichier de signets et lu à partir de celui-ci.

Remarque : Si pour régler le problème à l'origine de la défaillance, il faut modifier des données de la table ou du fichier source, le fichier de signets doit être supprimé et le processus doit être exécuté de nouveau depuis le début afin d'extraire les données modifiées.

Fichier de signets

Le nom et le répertoire du fichier de signets pour la reprise et la récupération sont définis dans le fichier de configuration (rmse_config.env). Le répertoire par défaut est $MMHOME/rfx/bookmark. La convention de dénomination du fichier de signets attribue par défaut le nom de fichier suivant, séparé par des points :

- Nom du programme
- Le premier nom de fichier, s'il en est spécifié un dans la ligne de commande
- "bkm"
- Date virtuelle d'activité à laquelle le module a été exécuté

Par exemple, le signet du programme `invildex` est inscrit dans le fichier suivant pour l'exécution de batch du 5 janvier 2001 :

```
$MMHOME/rfx/bookmark/invildex.invilddm.txt.bkm.20010105
```

Consignation des messages

Les journaux des messages sont écrits quotidiennement au format décrit dans cette section.

Fichier journal quotidien

Chaque programme RETL écrit un message dans le fichier journal quotidien lorsqu'il démarre et se ferme. Le nom et le répertoire du fichier journal quotidien est défini dans le fichier de configuration (rmse_config.env). Le répertoire par défaut est $MMHOME/log. Tous les fichiers journaux sont codés au format UTF-8.

La convention de dénomination du fichier journal quotidien attribue par défaut le nom de fichier suivant, séparé par des points :

- Date virtuelle d'activité à laquelle les modules sont exécutés
- ".log"

Par exemple, l'emplacement et le nom du fichier journal pour la date virtuelle d'activité du 5 janvier 2001 est :

```
$MMHOME/log/20010105.log
```

Format

Comme l'illustrent les exemples ci-dessous, chaque message écrit dans un fichier journal comporte le nom du programme, une indication temporelle et un message d'information ou d'erreur :

```
cusdemogdm 13:20:01: Program Starting...
cusdemogdm 13:20:05: Build update and insert data.
cusdemogdm 13:20:13: Analyze table rdw10dev.cust_demog_dm_upd
cusdemogdm 13:20:14: Insert/Update target table.
cusdemogdm 13:20:23: Analyze table rdw10dm.cust_demog_dm
cusdemogdm 13:20:27: Program Completed...
```

Si un programme s'interrompt avant exécution complète, un fichier d'erreur est normalement généré, indiquant où le problème s'est produit dans le processus. Certains messages d'erreur inscrits sur le fichier journal, tels que ‘No output file specified’ (aucun fichier de sortie spécifié), ne requièrent l'écriture d'aucune explication supplémentaire dans le fichier d'erreurs.

Fichier d'erreurs de programme

En plus du fichier journal quotidien, chaque programme écrit son propre flux détaillé et ses propres messages d'erreur. Afin d'éviter la saturation du fichier journal quotidien, chaque programme écrit ses erreurs dans un fichier d'erreurs distinct propre à chaque exécution.

Le nom et le répertoire du fichier d'erreurs du programme est défini dans le fichier de configuration (RMSE_config.env). Le répertoire par défaut est $MMHOME/error. Toutes les erreurs et *tous les messages de traitement courants* d'un programme donné, un jour donné, sont répertoriées dans ce fichier d'erreurs (par exemple, ce fichier contiendra les erreurs stderr et stdout de l'appel à RETL). Tous les fichiers d'erreurs sont codés au format UTF-8.

La convention de dénomination du fichier d'erreurs du programme attribue par défaut le nom de fichier suivant, séparé par des points :

- Nom du programme
- Date virtuelle d'activité à laquelle le module a été exécuté

Par exemple, toutes les erreurs et informations de consignation du programme `rms_item_master` sont écrites dans le fichier suivant pour l'exécution de batch du 5 janvier 2001 :

```
$MMHOME/error/rms_item_master.20010105
```

Fichiers de rejet RMSE

Les modules d'extraction RMSE produisent parfois un fichier de rejet en cas de problème lié aux données (données introuvables dans des tables de conversion requises par exemple). Le module tente de traiter toutes les données puis indique que certains enregistrements ont été rejetés. Ainsi, tous les problèmes relatifs aux données peuvent être identifiés en une seule fois et résolus. Le module peut alors être exécuté de nouveau. Si un module rejette des enregistrements, le fichier de rejet *n'est pas* supprimé. C'est à l'utilisateur de le supprimer avant de lancer une nouvelle exécution du module.

Les enregistrements du fichier de rejet contiennent un message d'erreur et des informations de clés provenant de l'enregistrement rejeté. L'exemple suivant illustre un enregistrement rejeté pour cause de problèmes rencontrés dans la bibliothèque de conversion des devises :

```
Currency Conversion Failed|101721472|20010309
```

L'exemple suivant illustre un enregistrement rejeté pour cause de problèmes rencontrés lors de la recherche d'informations dans une table source :

```
Unable to find item_master record for Item|101721472
```

Le nom et le répertoire du fichier de rejet est défini dans le fichier de configuration (rmse_config.env). Le répertoire par défaut est $MMHOME/data.

Remarque : Un répertoire destiné uniquement aux fichiers de rejet peut être créé. Le fichier rmse_config.env doit être modifié pour renvoyer à ce répertoire.

La convention de dénomination du fichier de rejet attribue par défaut le nom de fichier suivant, séparé par des points :

- Nom du programme
- Le premier nom de fichier, s'il en est spécifié un dans la ligne de commande
- "rej"
- Date virtuelle d'activité à laquelle le module a été exécuté

Par exemple, tous les enregistrements rejetés du programme `slsildmex` sont placés dans le fichier suivant pour l'exécution de batch du 5 janvier 2001 :

```
$MMHOME/data/slsildmex.slsildmdm.txt.rej.20010105
```

Fichiers de schéma

RETL utilise des fichiers de schéma pour spécifier le format des groupes de données entrants ou sortants. Le fichier de schéma définit le type de données et le format de chaque colonne, qui est ensuite utilisée dans RETL pour formater/manipuler les données. Vous trouverez des informations supplémentaires sur les fichiers de schéma dans le tout dernier Guide du programmateur RETL. Etant donné que les noms des fichiers de schéma ne changent pas régulièrement, ils sont figés dans le code de chaque module. Tous les noms de fichiers de ce type finissent par ".schema" et sont placés dans le répertoire "rfx/schema".

Paramètres de ligne de commande

Pour que chaque module RETL fonctionne, il est parfois nécessaire de faire passer les chemins et noms de fichiers de données entrants et sortants par la ligne de commande Unix.

RMSE

Les modules d'extraction RMSE ne requièrent la saisie d'aucun paramètre. Le chemin/nom du fichier sortant par défaut est $DATA_DIR/(nom du programme RMSE).dat. De même, le format de schéma des enregistrements de ces fichiers sont spécifiés dans le fichier - $SCHEMA_DIR/(nom du programme RMSE).schema.

Situations courantes d'exécution et de débogage

Les exemples suivants illustrent les situations courantes d'exécution et de débogage pour divers types de programmes. Tous les noms de fichiers mentionnés ci-dessous (fichier journal, d'erreur, etc...) font référence à l'exécution d'un module réalisée à la date virtuelle d'activité du 9 mars 2001. Pour connaître l'emplacement de chaque fichier, reportez-vous aux conventions de dénomination décrites antérieurement.

Par exemple :

Pour exécuter rmse_stores.ksh :

1 Modifiez les répertoires : $MMHOME/rfx/src.

2 A l'invite Unix, saisissez :

```
%rmse_stores.ksh
```

Si le module s'exécute correctement, les résultats suivants sont obtenus :

1 **Fichier journal :** Le fichier journal d'aujourd'hui, 20010309.log, contient les messages “Program started …” et “Program completed successfully” pour rmse_stores.

2 **Données :** Le fichier rmse_stores.dat se trouve dans le répertoire de données et contient les enregistrements extraits.

3 **Schéma :** Le fichier rmse_stores.schema se trouve dans le répertoire des schémas et contient la définition du fichier de données fournie ci-dessus, au numéro 2.

4 **Fichier d'erreurs :** Le fichier d'erreurs du programme, rmse_stores.20010309, contient le flux RETL standard (qui finit par "All threads complete" et "Flow ran successfully") et ne contient aucun message d'erreur supplémentaire.

5 **Contrôle du statu du programme :** Le fichier de contrôle du statut du programme rmse_stores.status.20010309 n'existe pas.

6 **Fichier de rejet :** Le fichier de rejet rmse_stores.rej.20010309 n'existe pas.

Si le module *ne* s'exécute *pas*, les résultats suivants sont obtenus :

1 **Fichier journal :** Le fichier journal d'aujourd'hui, 20010309.log, ne contient pas le message “Program completed successfully” pour rmse_stores.

2 **Données :** Le fichier rmse_stores.dat se trouve peut-être dans le répertoire de données mais ne contient pas obligatoirement tous les enregistrements extraits.

3 **Schéma :** Le fichier rmse_stores.schema se trouve dans le répertoire des schémas et contient la définition du fichier de données fournie ci-dessus, au numéro 2.

4 **Fichier d'erreurs :** Le fichier d'erreurs du programme, rmse_stores.20010309, contient peut-être un message d'erreur.

5 **Contrôle de l'état du programme :** Le fichier de contrôle du statut du programme, rmse_stores.status.20010309, existe.

6 **Fichier de rejet :** Le fichier de rejet rmse_stores.rej.20010309 n'existe pas car ce module ne rejette pas les enregistrements.

7 **Fichier de signets :** Le fichier de signets rmse_stores.bkm.20010309 n'existe pas car ce module n'utilise ni la reprise ni la récupération.

Pour exécuter de nouveau le module, procédez comme suit :

1 Trouvez et réglez le problème à l'origine de l'erreur.

2 Supprimez le fichier de contrôle du statut du programme.

3 Modifiez les répertoires : $MMHOME/rfx/src. A l'invite Unix, saisissez :

```
%rmse_stores.ksh
```