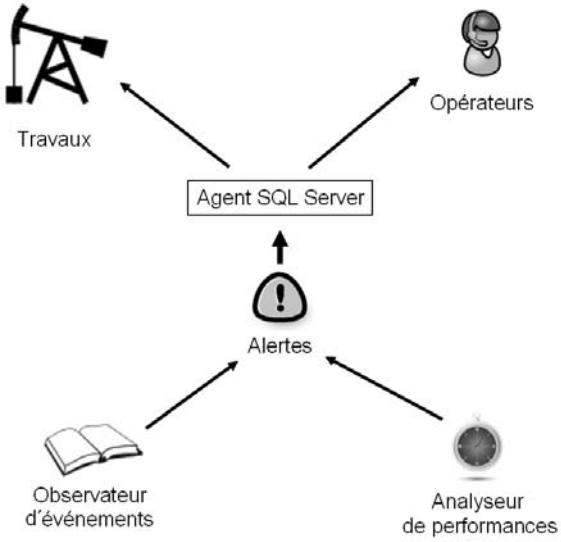
Chapitre 5
Tâches planifiées

1. Introduction

SQL Server donne la possibilité d'automatiser les tâches administratives. Il n'est bien sûr pas possible d'automatiser toutes les tâches mais les tâches planifiées représentent un bon complément à l'optimisation faite par défaut par SQL Server. De plus, avec ces tâches prédéfinies, l'administrateur possède un rôle d'anticipateur, ce qui lui donne plus de possibilités pour en tirer le meilleur tant au niveau des performances que de la fiabilité.

La gestion des tâches planifiées, des alertes et des opérateurs sont des services rendus par l'agent SQL Server. Ce service doit être démarré afin que ces éléments soient gérés. L'agent SQL Server travaille avec l'Observateur d'événements pour la gestion des erreurs SQL Server, l'Analyseur de performances pour la gestion des alertes sur des conditions de performances, et la base MSDB afin de connaître la réponse à appliquer face à une alerte, ou bien les tâches planifiées à exécuter.



Principe de fonctionnement

Face à une alerte, l'agent peut réagir en exécutant un travail et/ou en prévenant un opérateur afin que ce dernier soit au courant du problème qui vient de surgir. Bien entendu, l'exécution d'une tâche peut conduire au déclenchement de nouvelles alertes et ainsi de suite.

D'autres tâches planifiées vont être exécutées par le service SQL Server Agent, non pas en réponse à une erreur mais sur une base de temps. Par exemple, une reconstruction des index peut être planifiée une fois par semaine dans la nuit du samedi au dimanche.

L'agent SQL Server permet de réaliser une administration préventive des problèmes qui peuvent se poser lors de l'exploitation courante d'un serveur de bases de données.

2. Configuration des services

Comme l'exécution automatique de travaux administratifs repose sur le service SQL Server Agent, il est important que ce dernier soit correctement configuré.

Remarque

La configuration du service MSSQL Server a été abordée lors de l'installation.

2.1 La sécurité de SQL Server Agent

Le service SQL Server Agent permet la gestion de nombreux éléments. Si le service doit posséder des droits élevés sur le serveur pour être capable de réaliser correctement toutes les tâches qui lui sont assignées, l'utilisation de ce service doit être contrôlée au plus juste. Ce contrôle est assuré par les trois rôles de base de données définis sur la base msdb :

- **SQLAgentUserRole** : peuvent créer leurs propres travaux
- **SQLAgentReaderRole** : peuvent en plus lister tous les travaux du serveur
- **SQLAgentOperatorRole** : ont tous les droits sur la gestion des travaux, alertes et opérateurs

L'appartenance à ces rôles n'est nécessaire que pour les utilisateurs non-membres du rôle de serveur sysadmin.

Par exemple, si un utilisateur se connecte à la console graphique SQL Server Management Studio sans être membre de l'un de ces trois rôles, alors l'outil ne présentera tout simplement pas le nœud relatif à SQL Server Agent. Ainsi, l'utilisateur n'est pas capable de modifier, ni même de connaître le travail réalisé au niveau de l'automatisation de tâches. Le même niveau de sécurité est défini au niveau Transact SQL.

2.2 Configuration de la messagerie

La gestion des mails avec SQL Server passe par la messagerie de base de données. Cette fonctionnalité peut servir dans le cadre des bases de données utilisateurs afin d'envoyer des mails liés aux applications utilisateurs correspondantes ou bien dans le cadre de l'agent SQL Server pour des notifications de réussite et/ou échec aux administrateurs.

Le service de mail de base de données utilise le protocole standard SMTP pour envoyer les mails. Il ne repose pas sur MAPI, ce qui rend facultatif l'installation d'un client de messagerie comme Outlook. Au travers de ce protocole, le service de mail de base de données prend en charge l'envoi de mails au format HTML.

La messagerie de base de données n'est pas active par défaut, aussi il faut l'activer soit avec l'assistant de configuration au travers de SQL Server Management Studio, soit par l'intermédiaire de scripts Transact SQL en faisant appel à des instructions et des procédures stockées spécifiques. Ce paramétrage permettra de créer des profils de messagerie associés à un ou plusieurs comptes de messagerie.

Compte tenu du caractère ponctuel de cette action de configuration, seul le mode graphique est présenté dans cet ouvrage.

Il existe deux types de profils :

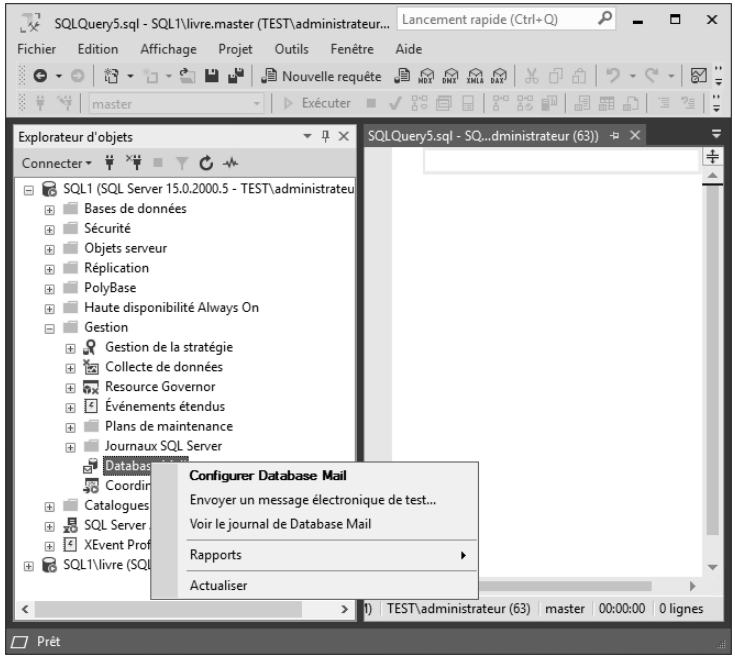
– Les profils publics, qui sont accessibles à tous les utilisateurs qui appartiennent au rôle **DatabaseMailUserRole** sur la base msdb. C'est parce qu'un utilisateur est membre de ce rôle qu'il peut envoyer des messages électroniques.
– Les profils privés, également définis sur la base msdb, mais cette fois-ci chaque utilisateur doit être précisément identifié afin de pouvoir utiliser le profil.

2.2.1 Configuration depuis SQL Server Management Studio

L'assistant de configuration de la messagerie de base de données va permettre de réaliser simplement et en étant guidé l'une des actions suivantes :

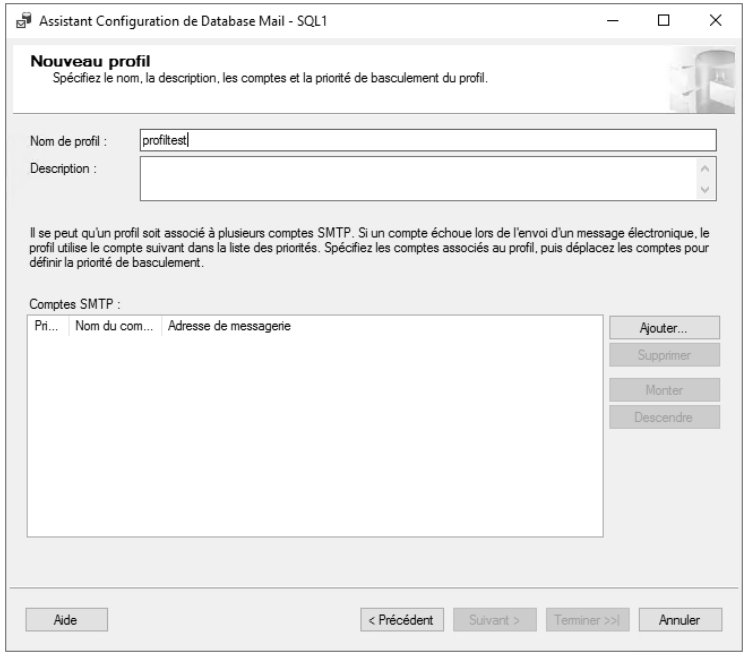
– Configurer la messagerie de base de données.
– Gérer les comptes de la messagerie de base de données.
– Gérer les profils de sécurité.
– Gérer les paramètres système.

L'assistant est lancé depuis SQL Server Management Studio en sélectionnant **Configurer Database Mail** dans le menu contextuel associé au nœud **Gestion - Database Mail** de l'instance SQL Server sur laquelle la configuration doit être faite.



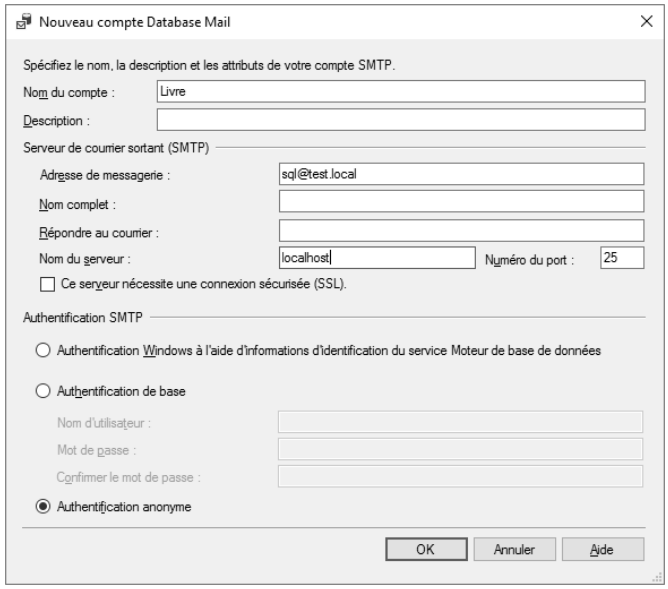
Le premier écran de l'assistant (après l'écran d'accueil) permet de sélectionner l'action que l'on souhaite réaliser avec l'assistant. La première étape consiste à configurer Database Mail.

Pour que la messagerie puisse fonctionner, il faut activer le composant en sélectionnant **Oui** à la question posée. La création de profils peut commencer.



Définition d'un premier profil

En utilisant le bouton **Ajouter**, il est possible de définir un ou plusieurs comptes de courrier.



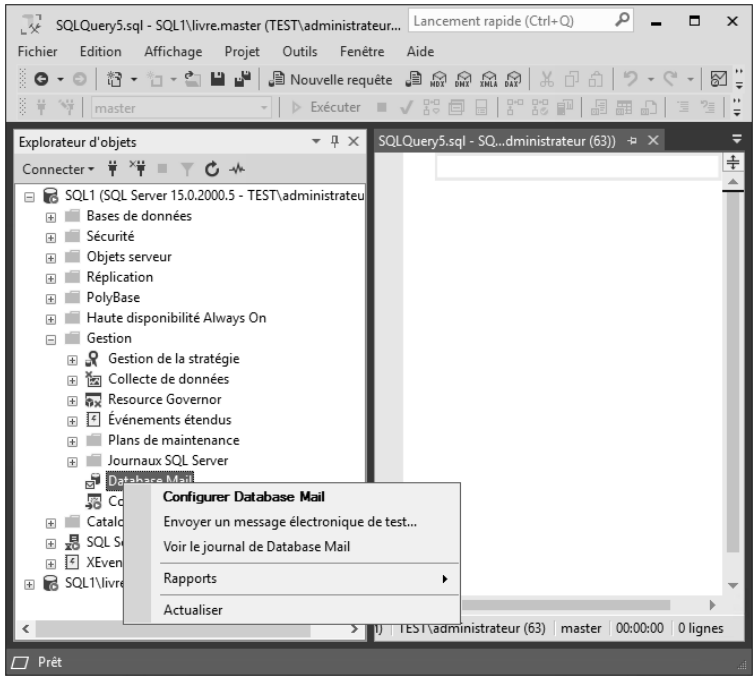
Définition d'un compte mail

Par la suite, l'assistant propose simplement de rendre public le profil, c'est-à-dire accessible à l'ensemble des utilisateurs du serveur. Enfin, l'assistant se termine par un écran de synthèse qui résume les différentes opérations demandées. La validation de cette synthèse entraîne la création du profil.

2.2.2 Tester le service

Seuls les utilisateurs membres du rôle de serveur **sysadmin** ou bien du rôle de base de données **databaseMailUserRole** défini sur la base **msdb**, peuvent envoyer des mails.

Il est facile de tester le profil, en sélectionnant **Envoyer un message électronique de test** depuis le menu contextuel associé au nœud **Database Mail** depuis l'explorateur d'objets de SQL Server Management Studio.



Un écran permet alors de préciser le profil à utiliser ainsi que le destinataire du message de test.

La messagerie est maintenant opérationnelle et il est possible d'envoyer des mails à l'aide de la procédure stockée sp_send_dbmail.

Chapitre 4
Transact-SQL : le langage procédural

1. Le SQL procédural

SQL Server est un serveur de base de données relationnelle et à ce titre, il fournit tous les éléments pour stocker de façon structurée les données mais aussi les outils nécessaires pour travailler avec les données au travers de SQL. Avec le Transact-SQL, il est également possible de définir des traitements procéduraux directement dans la base de données. Ces traitements vont pouvoir être utilisables par tous les utilisateurs de la base sous réserve qu'ils possèdent les privilèges nécessaires. Il est possible de conserver la définition de ces traitements et de les rendre paramétrables par l'intermédiaire de la création de procédures et de fonctions.

Des traitements procéduraux pourront également être mis en place pour définir des contraintes d'intégrité complexes, il s'agira alors de déclencheurs de base de données (TRIGGER).

Le Transact-SQL est un langage procédural qui intègre complètement et nativement le langage SQL. Il est ainsi possible de tirer parti des deux langages. Par exemple, il va être très facile de définir une variable en Transact-SQL puis d'inclure cette variable dans une requête SQL. Lors de l'exécution de la requête, c'est la valeur contenue par la variable qui est prise en compte.

Le Transact-SQL n'a pas pour objectif de se substituer aux requêtes SQL mais plutôt de permettre un complément par rapport aux tâches réalisables en SQL.

À l'aide du Transact-SQL, il va être possible de définir des procédures et des fonctions dans la base de données. Les déclencheurs de base de données vont permettre la mise en place au niveau de la base de règles métiers complexes.

1.1 Les variables

1.1.1 Les variables utilisateur

Une variable est une zone mémoire, caractérisée par un nom et un type, permettant de stocker une valeur. Les variables Transact-SQL doivent obligatoirement être déclarées avant leur utilisation. Elles peuvent ensuite remplacer n'importe quelle expression dans les instructions SQL.

Déclaration de variables

```
DECLARE @nom_variable type [= expr][,...][;]
```

`nom_variable`

Nom précédé du caractère @.

`type`

Type système ou défini par l'utilisateur.

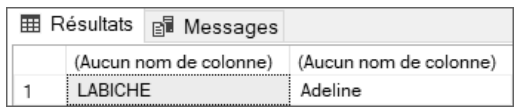
Valorisation des variables

```
SELECT @nom_variable = expr [,...][FROM...]
```

Exemple

Dans les scripts suivants, les variables sont tout d'abord définies. La variable @choix est ensuite valorisée avec la valeur 20. Puis une requête de type SELECT permet de renseigner le contenu des variables @nom et @prenom. Cette requête fonctionne correctement car la restriction sur une valeur unique de la clé primaire permet de garantir qu'une seule ligne de données sera remontée depuis la base. Enfin, la dernière instruction SELECT permet de visualiser le contenu des variables @nom et @prenom.

```
DECLARE @choix int;
DECLARE @nom CHAR(30), @prenom CHAR(30);
SELECT @choix=20;
SELECT @nom=nom, @prenom=prenom
FROM clients
WHERE idClient = @choix;
SELECT @nom, @prenom;
```


Résultats Messages

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	LABICHE	Adeline

1.1.2 Les variables système

Ces variables sont définies par le système, et peuvent être utilisées seulement en lecture. Elles se distinguent des variables utilisateur par le double @.

@@CONNECTIONS

Nombre de connexions ou de tentatives de connexion depuis le dernier démarrage de SQL Server.

@@CPU_BUSY

Temps consacré par l'unité centrale à SQL Server depuis le dernier démarrage de celui-ci. Le résultat est exprimé en unité CPU. Il faut multiplier par @@TIMETICKS pour obtenir un résultat en microsecondes.

@@CURSOR_ROWS

Nombre de lignes affectées dans le dernier curseur ouvert. Les curseurs sont présentés dans la section Les curseurs de ce chapitre.

`@@DATEFIRST`

Renvoie la valeur courante du paramètre SET DATEFIRST.

`@@DBTS`

Valeur du type de données timestamp courant pour la base de données.

`@@ERROR`

Dernier numéro d'erreur généré par le système.

`@@FETCH_STATUS`

Contient le statut d'une commande de curseur FETCH.

`@@IDENTITY`

Enregistre la dernière valeur IDENTITY insérée.

`@@IDLE`

Temps, en millisecondes, pendant lequel SQL Server est resté inactif depuis son dernier démarrage.

`@@IO_BUSY`

Temps, en millisecondes, consacré par SQL Server à effectuer des opérations d'entrée/sortie depuis son dernier démarrage.

`@@LANGID`

Identificateur de la langue actuellement utilisée.

`@@LANGUAGE`

Langue actuellement utilisée.

`@@LOCK_TIMEOUT`

Timeout, en millisecondes, de la session en cours.

`@@MAX_CONNECTIONS`

Nombre maximal de connexions simultanées qu'il est possible d'établir avec SQL Server.

`@@MAX_PRECISION`

Renvoie le niveau de précision utilisé par les types de données **decimal** et **numeric**.

`@@NESTLEVEL`

Niveau d'imbrication de l'instruction en cours d'exécution.

`@@OPTIONS`

Informations sur les valeurs courantes des options SET.

`@@PACK_RECEIVED`

Nombre de paquets entrants lus par SQL Server depuis son dernier démarrage.

`@@PACK_SENT`

Nombre de paquets sortants écrits par SQL Server depuis son dernier démarrage.

`@@PACKET_ERRORS`

Nombre d'erreurs qui se sont produites alors que SQL Server envoyait ou recevait des paquets depuis son dernier démarrage.

`@@PROCID`

Identificateur de la procédure stockée Transact-SQL du traitement en cours d'exécution.

`@@REMSERVER`

Renvoie le nom du serveur contenu dans l'enregistrement des noms d'accès d'un serveur distant.

`@@ROWCOUNT`

Nombre de lignes affectées par la dernière instruction.

`@@SERVERNAME`

Nom du serveur SQL local.

`@@SERVICENAME`

Nom du service en cours d'exécution.

`@@SPID`

Numéro d'identification du processus courant sur le serveur.

`@@TEXTSIZE`

Longueur maximale, en octets, des données texte ou image renvoyées par une instruction SELECT.

`@@TIMETICKS`

Nombre de millisecondes par pulsation.

`@@TOTAL_ERRORS`

Nombre d'erreurs rencontrées par SQL Server en lisant ou en écrivant des données depuis son dernier démarrage.

`@@TOTAL_READ`

Nombre de lectures de données sur le disque effectuées par SQL Server depuis son dernier démarrage.

`@@TOTAL_WRITE`

Nombre d'écritures de données sur le disque effectuées par SQL Server depuis son dernier démarrage.

`@@TRANCOUNT`

Nombre de transactions actuellement actives pour l'utilisateur courant.

`@@VERSION`

Date, numéro de version et type de processeur de la version courante de SQL Server.

Ces variables système contiennent de nombreuses informations et en les interrogeant il est déjà possible d'obtenir beaucoup d'informations sur le serveur.

Exemple

Dans l'exemple ci-dessous, la variable @@VERSION est interrogée et la version exacte de SQL Server est retournée. Avec ce numéro de version, il est par exemple possible de savoir quels Service Packs sont appliqués sur le serveur en s'appuyant sur les tableaux de correspondance disponibles sur le site web Technet de Microsoft.

```
SELECT @@VERSION;
Microsoft SQL Server 2019 (RTM-GDR) (KB4517790) - 15.0.2070.41 (X64)
Oct 28 2019 19:56:59   Copyright (C) 2019 Microsoft Corporation
Developer Edition (64-bit) on Windows 10 Pro 10.0 <X64> (Build 18363: )
```

1.1.3 L'affichage

L'instruction PRINT affiche un message.

Syntaxe

```
PRINT {'texte'|@variable|@@variablesystème}
```

Exemple

```
DECLARE @nb INT;
SELECT @nb=COUNT(*) FROM Clients;
PRINT 'Nombre de clients :'
PRINT @nb;
Affichage
Nombre de clients :
5
```

1.2 Les transactions

1.2.1 Le principe

Il est possible de définir une transaction comme un ensemble indivisible d'instructions, c'est-à-dire dans lequel toutes les instructions réussissent ou bien aucune. Cette notion de transaction est bien présente dans le monde réel, par exemple lors d'un retrait d'argent à un distributeur automatique bancaire les deux éléments indivisibles sont le débit du compte et la distribution des billets. Si l'un de ces éléments ne peut être exécuté correctement (et quelle qu'en soit la raison) alors c'est l'ensemble qui est mis en échec.

En effet, il n'est pas envisageable de distribuer des billets sans que le compte ne soit débité du montant correspondant.

1.2.2 La gestion des transactions

Le premier point à prendre en considération est le verrouillage des informations. En effet, lorsque SQL Server lit des données ou les modifie, il verrouille les lignes d'informations manipulées. Ce verrouillage dure le temps de l'exécution de l'instruction ou le temps de la transaction.

Suivant le type de verrous posés, il est possible ou non à d'autres transactions d'accéder simultanément à la même information.

Une transaction est un ensemble d'instructions de manipulations de données s'exécutant dans une même unité de travail. La validation d'une transaction assure que toutes les instructions en faisant partie se sont correctement terminées, l'annulation de la transaction assurera l'annulation de l'ensemble des instructions.

Seules les instructions du DML (SELECT, INSERT, UPDATE, DELETE) sont prises en compte dans une transaction.

Les transactions sont utiles pour assurer l'intégrité et la cohérence des données lors de modifications multiples, pour améliorer les performances, pour tester les effets de modification, pour gérer les verrouillages.

Une transaction est caractérisée par le mot-clé ACID (*Atomicity Consistency Isolation Durability*) qui peut se traduire en français par Atomique Consistance Indépendance Durée.

– Atomique car une transaction représente une unité atomique (non divisible) de travail pour le serveur.
– Consistance car à la fin de la transaction les informations présentes dans la base doivent être consistantes, c'est-à-dire cohérentes par rapport aux règles de structuration des données mises en place.