Base de Données Relationnelle - II

Erick STATTNER

Maître de Conférences en Informatique

Université des Antilles

erick.stattner@univ-antilles.fr

www.erickstattner.com

Description de l'enseignement

Objectifs pédagogiques:

- Découvrir un nouveau SGBD: SQL Server
- Se familiariser avec le T-SQL
- Maitriser la notion de transactions
- Accéder à une BD depuis un programme JAVA

Organisation:

- 10h CM, 10h TD,10h TP
- 2 notes CC + 1 projet
- 1 CT

Sommaire

Chapitre I.
Introduction à SQL Server

I. Introduction à SQL Server

Dans un contexte professionnel

- Besoin de sécurité
- Besoin de cohérence
- Besoin d'intégrité
- Besoin de scalabilité
- Besoin de performance

Problème

- Limite l'utilisation des SGBD simples: Access, MySQL, Postgre, etc.

I. Introduction à SQL Server

Microsoft SQL Server

- Système de Gestion de base de données (SGBD) relationnelle
- Commercialisé par Microsoft
- Sortie en 1989
- Comme tous les SGBD:
  - Garantie les opérations de stockage et CRUD
- Mais SQL Serveur
  - Gère les relations entre les tables et assure l'intégrité
  - Garantit la cohérence même en cas de panne
  - Traitements parallèles
  - Offre une sécurité forte
- Porté sur plusieurs plateformes, dont Windows, Linux et Mac,

I. Introduction à SQL Server

Microsoft SQL Server

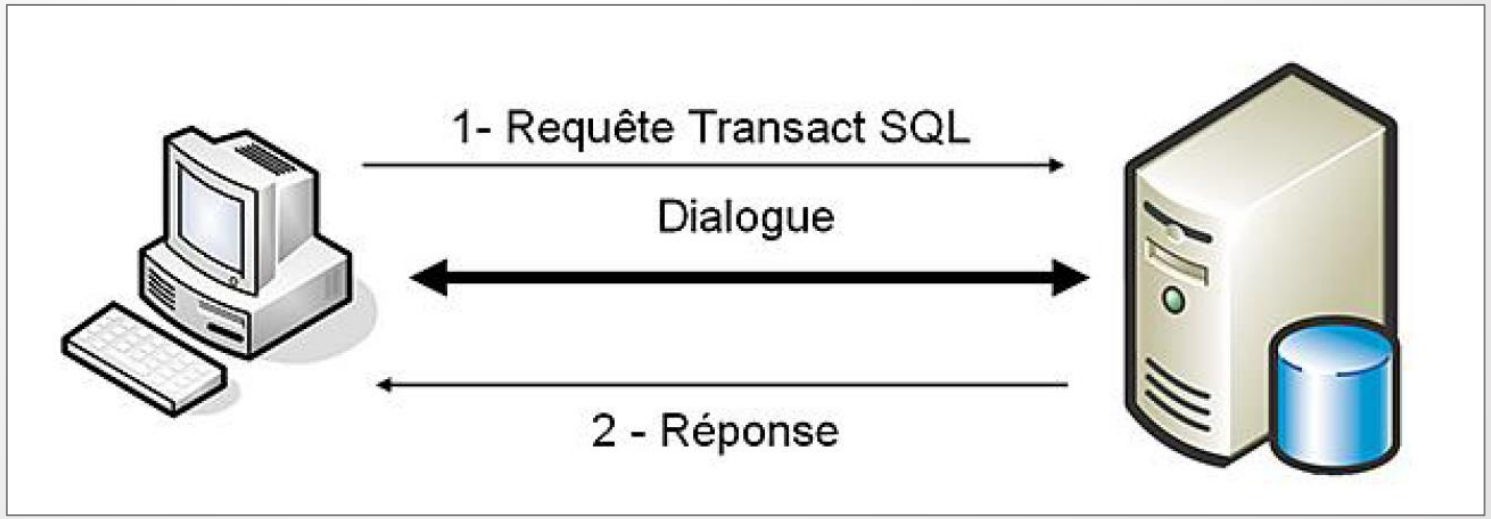
- Utilise le T-SQL (Transact-SQL)
  - Implémentation évoluée de SQL
  - Eléments de programmation
  - Prend en charge les procédure stockées, triggers
  - Transfert des données

On peut pratiquement tout faire sur SQL Server avec le T-SQL !

I. Introduction à SQL Server

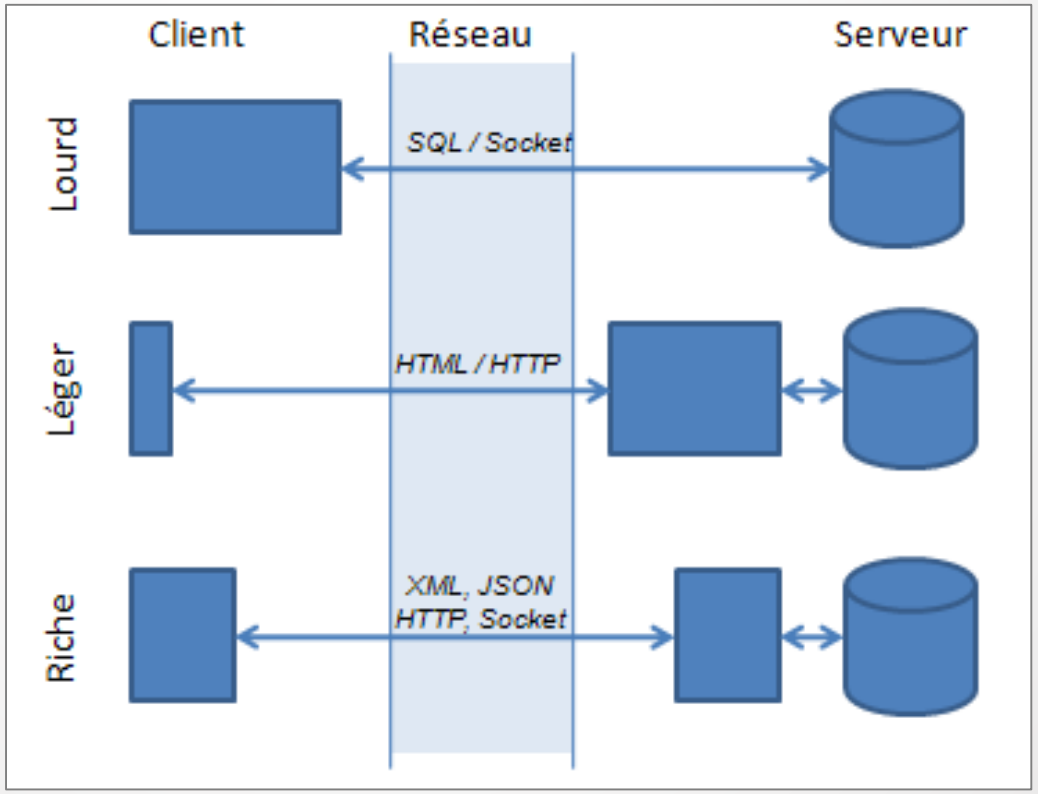
Microsoft SQL Server

- Une application qui utilise SQL Server s'appuie sur une architecture client/serveur
- Client: chargé de l'interface (executé sur plusieurs postes clients simultanément)
- Serveur: chargé de la gestion des données



I. Introduction à SQL Server

3 types de clients:



I. Introduction à SQL Server

Pour utiliser SQL Server

- Installer le SGBD (3 versions disponibles)
  - SQL Server: version entreprise (payante)
  - SQL Express: version gratuite pour application de bureau, appli web ou appli serveur
  - SQL Server dev.: fonctionnalités cédées sous licence
- Un outil de configuration/administration
  - SQL Server Management Studio (SSMS): offre une interface graphique pour administrer la base de données
- Un outil pour gérer tous les services
  - SQL Server Configuration Manager: permet de gérer tous les services autour de SQL Server

I. Introduction à SQL Server

Plusieurs briques logicielles autour de SQL Server

- **SQL Server Integration**
  Outil d'importation de d'exportation des données, facile à mettre en place et à paramétrer
- **Reporting services**
  Permet la création de rapport pour présenter au mieux les informations contenues dans SQL serveur
- **Replication des données**
  permet de stoker les données sur plusieurs sites
- **Intégration de code**
  Intégration du CLR (Common Language Runtime) qui permet d'étendre les fonctionnalités du T-SQL avec .Net ou C#

I. Introduction à SQL Server

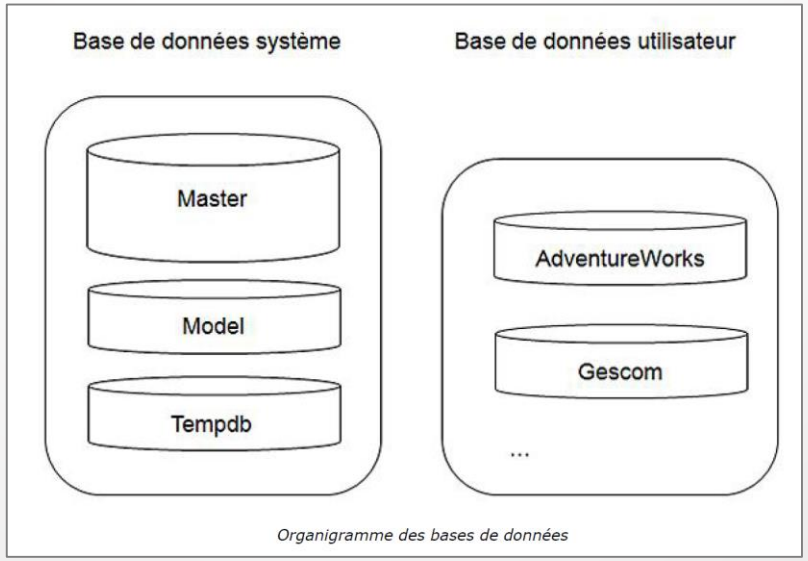
Objets de la base avec SQL Server

- Chaque base créée avec SQL Server contient un certain nombre d'objets logiques
- On peut les regrouper en 3 grandes catégories:
  - **Gestion et stockages des données**
    tables, colonnes, types de données, valeur par défaut, etc.
  - **Accès aux données**
    vues, procédures stockées, etc.
  - **Gestion de l'intégrité**
    triggers, contraintes d'intégrité, etc.

I. Introduction à SQL Server

Structuration au sein de SQL Server

- SQL Serveur s'utilise lui-même pour stocker l'ensemble des données.
- Deux types de base de données
  - Base de données système (ne pas modifier, ni renommer)
  - Base de données utilisateur



Organigramme des bases de données

I. Introduction à SQL Server

Structuration au sein de SQL Server

- **Base de données système**
  - **master**
    base principale, ensemble des données stratégiques (compte utilisateurs, option de configuration, base utilisateurs, fichiers, etc.)
  - **model**
    ensemble des éléments inscrits dans les bases utilisateurs.
  - **msdb**
    utilisée par l'Agent SQL Server pour planifier des alertes et des travaux, ainsi que par d'autres fonctionnalités telles que SQL
  - **tempdb**
    espace temporaire de stockage partagé. Elle est récréée à chaque démarrage.
- **Base de données utilisateur**
  - Héberge les données des applications métiers

I. Introduction à SQL Server

Base de données système

- Ne pas renommer
- Eviter de modifier (bien qu'un administrateur puisse le faire !)
- Pour interroger, il est déconseillé de le faire directement par une requête SELECT
  - Passer par des procédures stockées, des fonctions systèmes et des vues

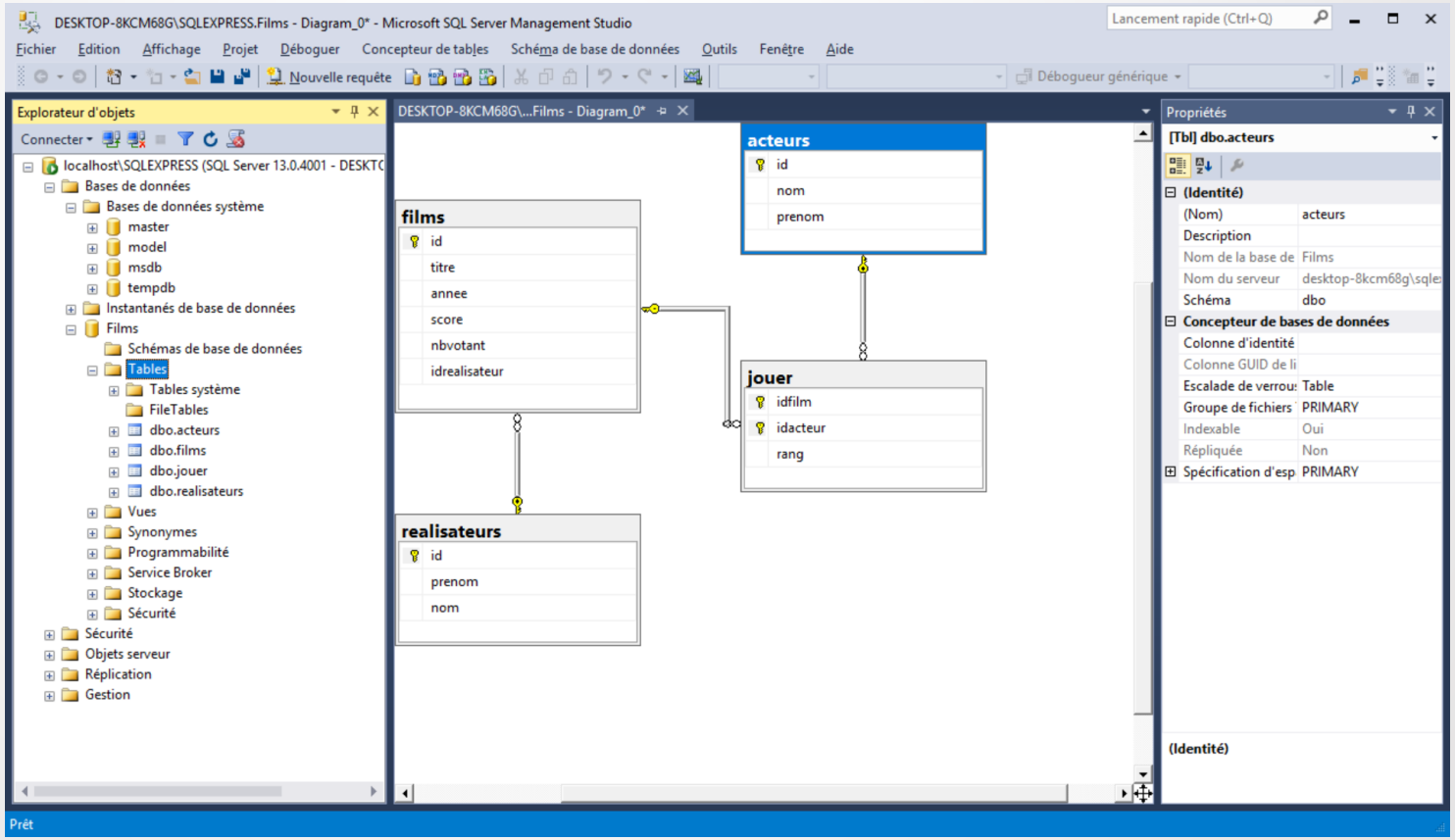
I. Introduction à SQL Server

Exemple de quelques table système de la base Master

Table système	Fonction
syslogins	Une ligne pour chaque utilisateur ou groupe Windows autorisé à se connecter au serveur SQL.
sysmessages	Une ligne pour chaque message d'erreur défini et pour chaque langue.
sysdatabases	Une ligne pour chaque base de données utilisateur.
sysconfigures	Une ligne pour chaque option de configuration du serveur.
sysusers	Une ligne pour chaque utilisateur défini dans la base.
syscolumns	Une ligne pour chaque colonne des tables, vues et pour chaque paramètre des procédures stockées.
sysobjects	Une ligne pour chaque objet de la base de données.

I. Introduction à SQL Server

Objets de la base avec SQL Server



Chapitre II. Transact-SQL

II. Transact-SQL

Définition

- Transact SQL (T-SQL) est un langage procédural (≠SQL déclaratif)
- Permet de programmer des algorithmes au sein de SQL Server
  - En particulier: transaction, procédures stockées, triggeurs
- Créé dans les années 80
- Crée pour répondre au besoin de programmer des algo. dans les SGBD
- Ne respecte pas les normes introduites avec SQL2 et 3

SGBDR	Langage	Respect norme
Oracle	PL/SQL	3/5
MS SQL Server	Transact SQL	2/5
PostGreSQL	SQL3	4/5
InterBase	ISQL	3/5

II. Transact-SQL

SQL Serveur

- Supporte les fonctions et la syntaxe SQL de base
- Transact-SQL qui étend les fonctionnalités du SQL de base

Attention à l'utilisation du T-SQL

- A utiliser dans
  - Procédures stockées
  - Triggers
- Bien identifié ce qui doit
  - être calculé et
  - Etre fait par la couche métier

II. Transact-SQL

Commentaires

- Permet de commenter une partie du code T-SQL
- Le commentaire n'est pas interprété
  - Placer entre les instructions /* et */ pour un bloc d'instructions
  - Précédé de -- pour une ligne de commentaire

Exemple

```
/* Ceci est un
commentaire dans le code T-SQL */
--Ceci aussi
```

II. Transact-SQL

Les instructions

- Encadrés au sein d'un bloc d'instructions qui constitue un groupe au moment de l'exécution
- Délimité par **BEGIN** et **END**

Exemple

```
BEGIN
    /* Mes instructions sont ici ! */
END
```

II. Transact-SQL

Identifiant

- Permet de faire référence à des objets
  - Variable locale, variable globale, tables, etc.
- Règles dans la création des identifiants
  - Pas plus de 128 caractères
  - Pas de caractères spéciaux, ni d'espace
  - La casse n'a pas d'importance
- @ précède le nom d'une variable
- @@ précède le nom d'une variable globale
- # précède le nom d'une table temporaire
- ## précède le nom d'une table temporaire globale

II. Transact-SQL

Variable locale

- Permet de déclarer des variables dans un bloc
- Doit être déclarée avec un identifiant et un type
- Déclaré avec **DECLARE**
- A la suite de **DECLARE**, la variable est initialisée à **NULL**

Exemple

```
DECLARE @id int
DECLARE @nom varchar(100)
```

II. Transact-SQL

Variable locale

- Plusieurs types possibles
- Ceux utilisés dans SQL Server
  - Int, Float, Real
  - Datetime, Timestamp
  - Varchar, Text
  - Table
  - etc.
- Les valeurs de types chaînes doivent être délimitées par des guillemets
- Il n'existe pas de tableau en T-SQL **-> utiliser type table**

II. Transact-SQL

Affectation

- Affecte une valeur à une variable
- Utiliser le mot clé **SET**
- Une requête peut servir à affecter une ou plusieurs variables
  - La requête ne doit produire qu'une ligne
  - Autrement, seule la dernière est prise en compte

Exemple

- SET @id = 42
- SELECT @nom=nom
  FROM acteurs
  WHERE id=@id

II. Transact-SQL

Variables globales du système

- Variables définies par le SGBD

Variable	Description
@@connections	nombre de connexions actives
@@error	**code de la dernière erreur rencontrée (0 si aucune)**
@@fetch_status	état d'un curseur lors de la lecture (0 si lecture proprement exécutée)
@@identity	**dernière valeur insérée dans une colonne auto incrémentée pour l'utilisateur en cours**
@@max_connections	nombre maximums d'utilisateurs concurrents
@@procid	identifiant de la procédure stockée en cours
@@rowcount	**nombre de lignes concernées par le dernier ordre SQL**
@@servername	nom du serveur SGBDR courant
@@spid	identifiant du processus en cours
@@trancount	nombre de transaction en cours

II. Transact-SQL

Afficher le contenu d'une variable

- Pour afficher le contenu d'une variable
- Utiliser la fonction **PRINT**
- Utiliser clause **SELECT**

Exemple

```
DECLARE @nom varchar(100)
SET @nom = 'toto'
SELECT @nom
```

II. Transact-SQL

Structure conditionnelle

- Permet de créer des branchements
- Conditionner l'exécution de certaines instructions
- IF <condition>
  instruction si condition vraie
  ELSE
  instruction si condition fausse
- Pas de ~~THEN~~
- Une seule instruction permise, autrement utiliser BEGIN / END

Exemple

```
IF(SELECT sum(score) FROM films WHERE genre ="SF") < 5
        DELETE FROM film WHERE genre ="SF"
```

II. Transact-SQL

Structure conditionnelle

- Variantes:
  - IF [NOT] condition
    instruction
    [ELSE instruction]
  - IF [NOT] EXISTS <requête select>
    instruction
    [ELSE instruction]

II. Transact-SQL

Boucle

- Permet d'exécuter un bloc d'instructions tant qu'une condition est vérifiée
- WHILE <condition>
  Instruction ou bloc d'instructions SQL
- Une seule instruction permise, autrement utiliser BEGIN / END

Exemple

```
WHILE(select sum(score) from film where genre='SF') < 2
BEGIN
    UPDATE film SET score=score+0.1  WHERE genre='SF'
END
```

II. Transact-SQL

Créer table temporaire

- Permet de créer une table temporaire pour stocker de l'information
- Permet de manipuler un ensemble de données
- Toutes les fonctions SQL s'utilisent sur cette table temporaire
- La table est stockée dans la base système **TEMPDB** et existe jusqu'au redémarrage du serveur

Exemple

```
CREATE TABLE #TableTmp (id int, prenom varchar(100)

INSERT INTO # TableTmp VALUE (1, 'toto')
SELECT * from # TableTmp
```

II. Transact-SQL

Créer des fonctions

- T-SQL permet également de créer des UDF (User Define Functions)
- Dans T-SQL une fonction est considérée comme un objet
  - Apres sa création, elle fait partie de la BD
  - Peut être utilisée dans du code T-SQL (triggers, transactions, etc.)
- Deux principaux types de fonctions
  - Fonction qui renvoie une valeur
    Fonction scalaire
  - Fonction qui renvoie une table
    Fonction table

II. Transact-SQL

Fonction scalaire

- Fonction qui renvoie une valeur
- Peut être utilisée dans du code T-SQL

Syntaxe

```
CREATE FUNCTION <nom_fonction> (
        @param1  type,
        @param2  type
        …
)
RETURNS type_retour
AS
BEGIN
        /* code ici */
        RETURN type_retour
END
```

II. Transact-SQL

Exemple fonction scalaire

- Elever un nombre au carré

```
CREATE FUNCTION carre ( @p  int )
RETURNS int
AS
BEGIN
            RETURN @p * @p
END
```

II. Transact-SQL

Fonction table

- Fonction qui renvoie une table
- Deux possibilités:
  1. Utiliser tables existantes
  2. Construire intégralement la table

II. Transact-SQL

Fonction table: à partir de tables existantes

- **Syntaxe**

```
CREATE FUNCTION <nom_fonction> (@param1  type, …)
RETURNS TABLE
AS
RETURN(
        -- Une requête ici
)
```

- **Exemple: fonction qui renvoie les meilleurs et les pires films**

```
CREATE FUNCTION BestAndWorstFilms()
RETURNS TABLE
AS
RETURN(
        SELECT id, titre FROM film WHERE score >= 8
        UNION
        SELECT id, titre FROM film WHERE score <= 2
)
```

II. Transact-SQL

Fonction table: construite intégralement

- **Syntaxe**

```
CREATE FUNCTION <nom_fonction> (@param1  type, …)
RETURNS <nomTable> TABLE (<définition de la table>)
AS
BEGIN
        -- La table est créée ici
        RETURN
END
```

- **Exemple: fonction qui renvoie table contenant entiers de 1 à N**

```
CREATE FUNCTION getDixPremiersEntiers (@N int)
RETURNS @maTable TABLE (N int PRIMARY KEY NOT NULL)
AS
BEGIN
        DECLARE @i int
        SET @i = 1
        WHILE @i < @N
        BEGIN
                INSERT INTO @maTable VALUE( @i )
                SET @i = @i + 1
        END
        RETURN
END
```

Chapitre III.
Gestion des transactions

III. Gestion des transactions

Contexte: Sonic veut acheter un drone

Etapes pour effectuer l'achat

1. Débiter argent sur le compte de Sonic	1. UPDATE client SET solde=solde-1000 WHERE nom='Sonic'
2. Créditer argent sur le compte du vendeur	2. UPDATE vendeur SET solde=solde+1000 WHERE nom='Vendeur'
3. Diminuer le stocke de drone de 1	3. UPDATE article SET stock=stock-1 WHERE nom='Drone'

III. Gestion des transactions

Contexte

- Lorsqu'on débute, on fait les requêtes les unes après les autres
- On ne se soucie pas de savoir si
  - Les requêtes échouent
  - Si il y a une panne pendant l'opération
  - Si les données sont cohérentes
  - S'il y a des conflits avec des requêtes faites en parallèle

Quels sont les problèmes qui peuvent survenir avec l'achat de Sonic ?

III. Gestion des transactions

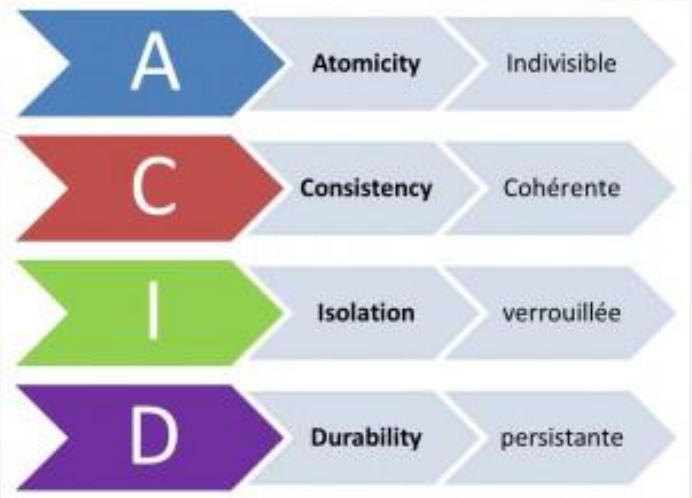
Exemple de Sonic

- Le compte de Sonic doit bien être débité.
- Le compte du vendeur n'est crédité qu'une fois le compte de Sonic débité
- Si un problème survient, l'ensemble des opérations doit être annulé
- Aucune valeur incohérente ne doit être générée
  - Compte de Sonic doit être positif
  - Stocke de drone ne peut pas être négatif

III. Gestion des transactions

Transaction

- Une séquence d'opérations qui font passer la base de données d'un état A (antérieur à la transaction) à un état B (postérieur)
- Des mécanismes garantissent que les opérations se déroulent bien
- La transaction est **ACID**



III. Gestion des transactions

Transaction

- **Atomique:** la suite d'opérations est indivisible (Tout ou rien !)
  En cas d'échec sur une seule des opérations, la suite d'opérations doit être complètement annulée quel que soit le nombre d'opérations déjà réussies.
- **Cohérente:** La transaction s'assure de garder la base cohérente.
  C'est-à-dire conforme aux règles métiers. Un contenu final incohérent doit entraîner l'échec et l'annulation de toutes opérations de la transaction.
- **Verrouillée:** lorsque deux transactions A et B sont exécutées en même temps, les modifications effectuées par A ne sont ni visibles par B, ni modifiables par B tant que la transaction A n'est pas terminée et validée.
- **Persistante:** Une fois validé, l'état de la base de données doit être permanent, et aucun incident technique (exemple: crash) ne doit pouvoir engendrer une annulation des opérations effectuées durant la transaction.

III. Gestion des transactions

Transaction en SQL Serveur

- Une transaction est un ensemble de requêtes que l'on regroupe en une seule unité logique
- Une transaction doit NECESSAIREMENT être soit **validée** ou **annulée**.
- Une transaction commence toujours par
  BEGIN TRANSACTION <nomTransaction>
  /* Le code à protéger est apres */
- Se termine par
  - COMMIT TRANSACTION <nomTransaction>
    Si la transaction est validée
  - ROLLBACK TRANSACTION <nomTransaction>
    Si la transaction est annulée -> Annule toutes les opérations effectuées

SQL Server fonctionne par défaut en mode AUTOCOMMIT, i.e. toute requête est automatiquement validée

III. Gestion des transactions

Exemple avec Sonic

```
BEGIN TRANSACTION achatArticle
        -- Le compte de Sonic est débité
        UPDATE client SET solde=solde-1000 WHERE nom='Sonic'
        -- Le compte du vendeur est crédité
        UPDATE vendeur SET solde=solde+1000 WHERE nom='Vendeur'
        -- Le stocke est diminué
        UPDATE article SET stock=stock-1 WHERE nom='Drone'
COMMIT TRANSACTION achatArticle
```

III. Gestion des transactions

Exemple avec Sonic

```
BEGIN TRANSACTION achatArticle
        -- Le compte de Sonic est débité
        UPDATE client SET solde=solde-1000 WHERE nom='Sonic'
        -- Le compte du vendeur est crédité
        UPDATE vendeur SET solde=solde+1000 WHERE nom='Vendeur'
        -- Le stocke est diminué
        UPDATE article SET stock=stock-1 WHERE nom='Drone'
COMMIT TRANSACTION achatArticle
```

En cas d'erreur les opérations sont tout de même validées !

III. Gestion des transactions

Exemple avec Sonic

```
BEGIN TRANSACTION achatArticle
        DECLARE @err int
        SET @err = 0

        -- Le compte de Sonic est débité
        UPDATE client SET solde=solde-1000 WHERE nom='Sonic'
        SET @err = @err + @@error

        IF(SELECT solde FROM client WHERE nom='sonic) < 0
              SET @err = @err + 1

        SET @err = @err + @@error

         /* Le reste des requêtes et des vérifications */


        IF @err = 0
              COMMIT TRANSACTION achatArticle
        ELSE
              ROLLBACK TRANSACTION achatArticle
```

III. Gestion des transactions

Principes des Transactions dans SQL Server

- Tant qu'aucun COMMIT/ROLLBACK, instructions internes à la transaction stockées en mémoire
  - Instructions tracées dans fichier journal
  - Pas d'écriture définitive en base
- A fréquence régulière, un programme vérifie le journal
- Peut reprendre le cas échéant

Permet à SQL Server de reprendre la transaction même en cas de crash du systeme