

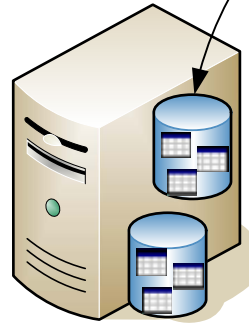
# REQUÊTE ACTION

Modification de données : Insert, Update, Delete  
Modification du schéma : Create / Drop / Alter Table ...

6°/ Ouverture de la connexion

7°/ Exécuter la commande en appliquant la méthode *ExecuteNonQuery()* sur la commande. Cette méthode retourne le nombre d'enregistrements impactés dans la base de données

8°/ Fermer la connexion



# SELECT SCALAIRE

(select count(\*)...)

8°/ Caster le résultat suivant le type attendu (entier, chaîne de caractères, booléen...)  
9°/ Fermer la connexion

6°/ Ouverture de la connexion

7°/ Exécuter la commande en appliquant la méthode *ExecuteScalar()* sur la commande

**Principe :** on crée un curseur qui lit les données directement dans la base de données, au travers d'un DataReader. Peu gourmand en mémoire.

### Contraintes :

- Accès séquentiel aux données en avant seulement
- On ne peut modifier les données
- La connexion reste ouverte pendant l'exploitation des données

**MODE CONNECTE**

**MODE DECONNECTE :** on rapatrie dans la mémoire de l'application, les données contenues dans une base de données

# UPDATE

6°/ Créer un DataAdapter (commande)

7°/ Paramétrer le DataAdapter (CommandBuilder)

- InsertCommand
- UpdateCommand
- DeleteCommand

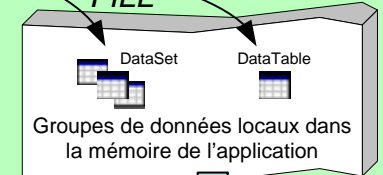
8°/ Créer le groupe de données local

9°/ Ouvrir la connexion (optionnel)

10°/ Alimenter le groupe de données local

11°/ Fermer la connexion (optionnel)

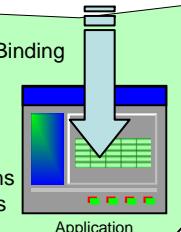
# FILL



12°/ DataBinding

13°/ Modifications de données

- 14°/ Deux choix :
- A : annulation des modifications
  - B : validation des modifications



# SELECT TABULAIRE

(select \*...)

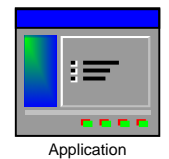
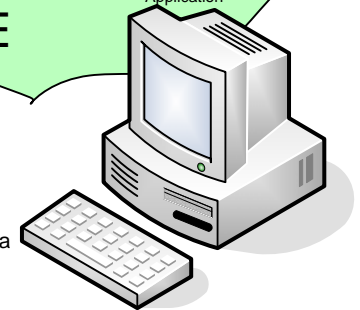
6°/ Ouverture de la connexion

7°/ Créer un DataReader en appliquant la méthode *ExecuterReader(<options>)* sur la commande

8°/ Parcourir les données While (*DataReader.Read()*)

```
{  
    DataReader.Getxxx(i);  
}
```

9°/ Fermer la connexion



James RAVAILLE  
<http://blogs.dotnet-france.com/jamesr>