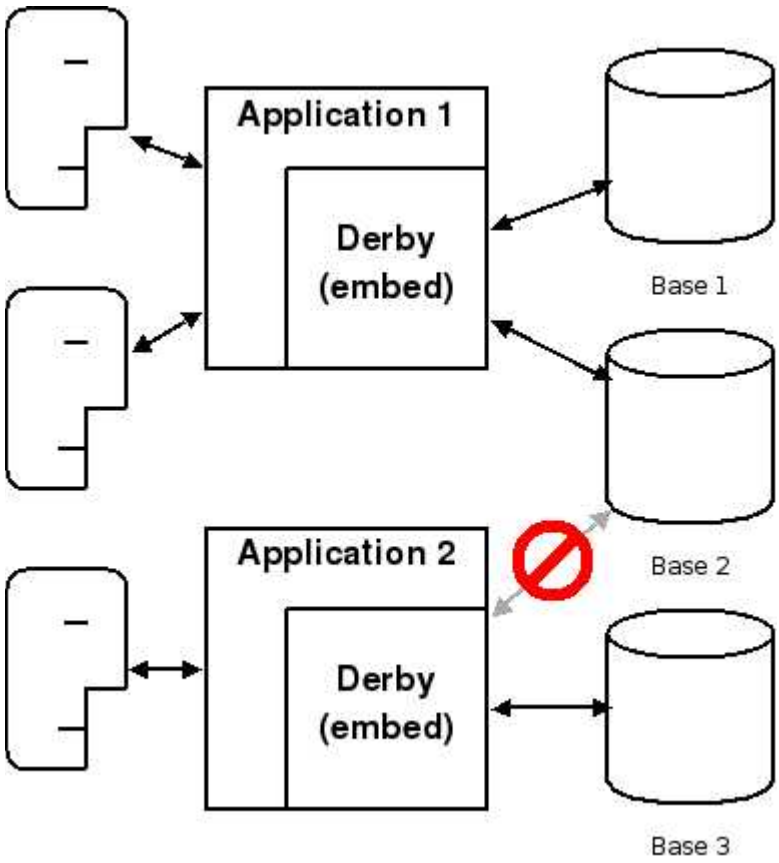
APACHE JAVA DERBY DB

Apache Derby est une base de donnée relationnelle écrite en pure Java et utilise les standard SQL et JDBC.

Derby a la particularité de pouvoir être utilisé comme gestionnaire de base de données **embarqué** dans une application Java. Ce qui rend inutile l'installation et la maintenance d'un serveur de base de données autonome. A l'inverse Derby supporte aussi un mode de fonctionnement **client-serveur**.

Dans ce document, nous allons utiliser Derby en mode embarqué pour créer notre première base de données – et nous servir de l'utilitaire `ij` fourni avec Derby pour effectuer quelques requêtes SQL histoire de nous familiariser avec cet outil.



Le **driver embarqué Derby** (*Derby Embedded Driver*) permet la connexion simultanée de plusieurs utilisateurs et l'accès concurrent à plusieurs bases. Néanmoins, dans cette configuration, un seul driver – et donc une seule application – peut accéder à une même base à un instant donné.

Les fonctionnalités de Derby sont :

- Moteur embarqué avec drivers JDBC
- Serveur Réseau
- Client JDBC Réseau

- Outils en ligne de commande : ij (SQL scripting), dblook (schema dump) et sysinfo (system info)

Java et versions JDBC versions supportées:

- Java SE 1.4 et plus avec JDBC 2.1, 3.0, 4.0 et 4.1
- Java ME CDC/Foundation Profile 1.1 avec JSR-169 JDBC (Optional Package for CDC/Foundation Profile).

1) Téléchargement de Derby et installation

Il est possible de télécharger Derby à l'adresse suivante :
http://db.apache.org/derby/derby_downloads.html

Attention à bien télécharger le package bin, car seul celui ci contient les fichiers compilés.

Actuellement la dernière version stable est la 10.13.1.1 au 25/10/2016, l'ensemble se présente sous la forme d'un package zip ou tar.gz.

There are four different distributions:

- bin distribution - contains the documentation, javadoc, and jar files for Derby.
- lib distribution - contains only the jar files for Derby.
- lib-debug distribution - contains jar files for Derby with source line numbers.
- src distribution - contains the Derby source tree at the point which the binaries were built.

db-derby-10.13.1.1-bin.zip [PGP] [MD5]
db-derby-10.13.1.1-bin.tar.gz [PGP] [MD5]

db-derby-10.13.1.1-lib.zip [PGP] [MD5]
db-derby-10.13.1.1-lib.tar.gz [PGP] [MD5]

db-derby-10.13.1.1-lib-debug.zip [PGP] [MD5]
db-derby-10.13.1.1-lib-debug.tar.gz [PGP] [MD5]

db-derby-10.13.1.1-src.zip [PGP] [MD5]
db-derby-10.13.1.1-src.tar.gz [PGP] [MD5] (Note that, due to long filenames, you will need gnu tar to unravel this tarball.)

Pour installer Java Derby il faut dans un premier temps décompresser le fichier zip ou tar.gz vers un répertoire de votre choix. Exemple « d:\derby » ou « $HOME\derby » sous Linux

On obtient les sous répertoires suivant :

- Le sous répertoire ***demo*** *contient les programmes de démonstration.*
- Le sous répertoire ***bin*** *contient les scripts pour exécuter les outils et ajuster l'environnement*.
- Le sous répertoire ***javadoc*** *contient la documentation.*
- Le sous répertoire ***docs*** *contient la documentation de* Derby.
- Le sous répertoire ***lib*** *contient les fichiers .jar de Derby.*
- Le sous répertoire **test** *contient les classes de tests de régression pour Derby.*

2) Positionner les variables d'environnement

Il faut positionner la variable d'environnement DERBY_HOME avec le chemin ou vous avez extrait la distribution du répertoire bin de DERBY.

En effet seule cette variable est nécessaire, les autres variables sont directement positionnées par les scripts shell ou batch qui se trouvent dans le répertoire « bin » .

```
UNIX (Korn Shell) => export DERBY_HOME=/opt/Derby
Windows => set DERBY_HOME=c:\Derby
```

Il est recommandé également d'initialiser la variable system PATH, chemin de recherche de vos programmes.

```
UNIX (Korn Shell) => export PATH="$DERBY_HOME/bin:$PATH"
Windows set PATH=%DERBY_HOME%\bin;%PATH%
```

Nb : Quand la variable DERBY_HOME est ajustée avec le sous-répertoire /bin, il vous est possible d'appeler les scripts de raccourcis pour lancer les outils de DERBY.

3) Démarrer DerbyDB en mode standalone (mono-utilisateur)

exemple d'appel sous ij :
Il faut lancer l'application ij, puis émettre la commande de connexion suivante :

CONNECT 'jdbc:derby:nombase;create=true/false' ;

si la base existe, il faut mettre à false la variable create, sinon la mettre à true.
Le fait de mettre à true cette variable create, force derby à créer la base de donnée dans le répertoire par défaut.
Attention si vous lancer l'accès à la base dans deux JVM différentes en utilisant create=true, cela risque d'écraser la base de données de départ.

Quelques commandes intéressantes sous ij :
Par défaut la commande **CONNECT**, donne un nom à la connexion que l'on vient de créer (CONNECTIONxx), il est possible de forcer le nom de cette connexion en utilisant la syntaxe de connexion suivante :

CONNECT 'JDBC:DERBY:NOMBASE;create=true/false' as NOMCONNECTION ;

Attention à bien utiliser les simples quotes pour délimiter la ligne de connexion.

Changer la connexion en cours :

Il est possible de changer la connexion en cours quand on en a plusieurs d'ouvertes, pour cela il faut utiliser la commande SET.

SET nomCONNECTION ;

SET CONNECTION1 ;

Deconnecter une base de donnée :

DISCONNECT NomCONNECTION ;

déconnecter la base courante
DICONNECT CURRENT ;

déconnecter toutes les bases
DISCONNECT ALL ;

Visualiser les éléments de la base :

SHOW connections ;
Permet de visualiser les noms de toutes les connexions.

SHOW Schemas ;
Permet de lister l'ensemble des schémas accessible par la connexion en cour.

SHOW [TABLES | VIEWS | PROCEDURES | FUNCTIONS | SYNONYMS] { IN schéma };
Permet de lister l'ensemble des tables, ou vues ou procédures ou fonctions ou synonymes accessible par la connexion en cours.

SHOW INDEXES { IN schéma | FROM table };
Affiche la liste des indexes d'un schéma ou d'une table particuliere.

SHOW ROLES ;
Répertorie tous les rôles définis dans la base de données.

SHOW ENABLED_ROLES ;
Liste tous les rôles activés pour la connexion en cours.

Transactions :

AUTOCOMMIT ON|OFF ;
Permet de forcer ou non le commit après chaque insertion ou suppressions de données. Si le forçage n'est pas notifié, il faudra valider chaque ajout/suppression de données par la commande COMMIT .

COMMIT & ROLLBACK ;
Les commandes qui permettent de valider ou annuler une transaction.

Autres commandes :

RUN 'nom de fichier';
Exécute des commandes depuis le fichier indiqué.
Très pratique pour créer des éléments comme des tables, insérer des données, il suffit d'écrire l'ensemble des commandes SQL dans un fichier au format ASCII et de le jouer dans ij avec cette

commande.

ELAPSEDTIME [ON | OFF];
Définit le mode temps écoulé pour ij

MAXIMUMDISPLAYWIDTH NombreEntier;
Définit que la largeur d'affichage maximum pour chaque colonne, est indiquée sous forme d'entier.

4) Démarrer DerbyDB en mode Client-Server (Multi-utilisateur)

Pour démarrer Apache Derby en mode muti-utilisateur, il est nécessaire de faire appel a celle ci en mode serveur.

L'appel de la connexion se fera de la manière suivante :

CONNECT 'jdbd:derby://AdresseServeur:Port/NomBase;create=false';

AdresseServeur peut prendre la valeur du nom de machine ou de l'adresse IP de celle ci, le port à pour valeur par défaut 1527 (sauf s'il à été modifié...).

5) Restreindre l'accès à Derby par mot de passe

Pour cela il faudra créer un fichier derby.properties que l'on déposera dans le répertoire DERBY_HOME.

Ce fichier properties doit comporter au minimum les 3 valeurs de sécurités suivantes :

derby.connection.requireAuthentication=true
derby.authentication.provider=BUILTIN
derby.user.NomUser=Password

NomUser doit prendre la valeur du login de l'utilisateur et password défini le mot de passe...

Exemple :
derby.connection.requireAuthentication=true
derby.authentication.provider=BUILTIN
derby.user.herve=monpass

il sera dans ce cas nécessaire de se connecter à notre système Derby avec les chaînes de connections suivante :

Dans le cas d'une connexion mono-utilisateur :
CONNECT 'jdbc:derby:mabase;create=false;user=herve;password=monpass' ;

Dans le cas d'une connexion multi-utilisateur :
CONNECT 'jdbc:derby://localhost:1527/mabase;create=false;user=herve;password=monpass ' ;

6) SQL Derby

Créer une table

Contrairement à ce que nous venons de voir pour la base elle-même, la création des **tables** se fait de façon plus conventionnelle. A l'aide d'une requête SQL. Commençons par la table représentant les *classes* d'animaux:

```
ij> CREATE TABLE Classe (
>     Id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
>     Designation VARCHAR(30) NOT NULL UNIQUE);
0 rows inserted/updated/deleted
```

Cette première table contient deux colonnes:

1. `Id` qui contient un identifiant unique pour l'espèce. Cet identifiant sera un numéro séquentiel généré automatiquement par Derby (`GENERATED ALWAYS AS IDENTITY`). Il nous servira de clé primaire (`PRIMARY KEY`).
2. `Designation` qui contiendra la désignation de la classe d'animaux (reptile, oiseau, etc.). Il ne peut pas y avoir deux classes ayant la même désignation (`UNIQUE`). Toute classe doit avoir une désignation (`NOT NULL`).

Dès maintenant il est possible de peupler la table des classes d'animaux:

```
ij> INSERT INTO Classe(Designation) VALUES
>     ('oiseaux'),
>     ('mammifères'),
>     ('reptiles');
3 rows inserted/updated/deleted
```

Piège:

Avec Derby, le délimiteur pour les **valeurs** chaînes de caractères est l'apostrophe. Vous *ne* pouvez *pas* utiliser de guillemets à la place. Ceux-ci sont réservés pour mettre autour des noms de **champs**:

```
ij> INSERT INTO Classe(Designation) VALUES ("bivalves");
ERROR 42X04: Column 'bivalves' is either not in any table in the FROM list or
appears within a join specification and is outside the scope of the join
specification or appears in a HAVING clause and is not in the GROUP BY list. If
this is a CREATE or ALTER TABLE  statement then 'bivalves' is not a column in
the target table.
```

Par ailleurs, Derby convertit automatiquement en MAJUSCULES les identifiants sauf s'ils sont délimités par les guillemets:

```
ij> INSERT INTO Classe("Designation") VALUES ('bivalves');
ERROR 42X14: 'Designation' is not a column in table or VTI 'APP.CLASSE'.
```

L'exemple ci-dessus provoque une erreur, car lors de la création de la table, nous n'avons pas utilisé de guillemets autour du champ `Designation`. Cet identifiant est donc connu par Derby sous sa forme majuscule. Ce qui explique pourquoi l'exemple ci-dessous, lui, est accepté:

```
ij> INSERT INTO Classe("DESIGNATION") VALUES ('bivalves');
1 row inserted/updated/deleted
```

Bien sûr, vous pouvez examiner le contenu de la table à l'aide d'une requête SELECT:

```
ij> SELECT * FROM Classe;
ID         |DESIGNATION
----------------------------------------
1          |oiseaux
2          |mammifères
3          |reptiles
4          |bivalves

4 rows selected
```

Ma première relation

Le sous-ensemble de SQL-92 supporté par Derby contient la définition des clés étrangères. Nous allons donc pouvoir créer la table `Espèce` en précisant de manière *explicite* sa relation avec la table `Classe`:

```
ij> CREATE TABLE Espèce (
>     Id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
>     Nom VARCHAR(50) NOT NULL UNIQUE,
>     Classe INT REFERENCES Classe(Id));
0 rows inserted/updated/deleted
```

Comme vous le voyez, le champ `Espèce.Classe` référence le champ `Classe.Id`. Autrement dit, cette contrainte m'empêche d'insérer une espèce donc la classe n'est pas présente dans la table correspondante:

```
ij> INSERT INTO Espèce(Nom, Classe) VALUES ('Mygale de Leblond', 6);
ERROR 23503: INSERT on table 'ESPÈCE' caused a violation of foreign key
constraint 'SQL090509011909552' for key (6).  The statement has been rolled
back.
```

Il n'y a pas d'enregistrement dans la table `Classe` dont `Id` vaut 6, donc l'insertion ci-dessus est refusée par Derby.

Par contre, les insertions valides fonctionnent (heureusement!):

```
ij> INSERT INTO Espèce (Nom, Classe) VALUES
>     ('Pie bavarde', 1),
>     ('Merle noir', 1),
>     ('Coucou gris', 1),
>     ('Mulot sylvestre', 2),
>     ('Lapin européen', 2),
>     ('Tortue des bois', 3),
>     ('Lézard des murailles', 3),
>     ('Hirondelle rustique', 1),
>     ('Crocodile du Nil', 1);
9 rows inserted/updated/deleted
```

Jointures

Derby supporte aussi les jointures. Par exemple, pour extraire l'ensemble des oiseaux connus dans la base de données, je peux utiliser la requête suivante:

```
ij> SELECT Classe.Designation, Espèce.Nom
>     FROM Classe
>     INNER JOIN Espèce ON Classe.Id = Espèce.Classe
>     WHERE Classe.Designation = 'Oiseaux';
DESIGNATION                    |NOM
--------------------------------------------------------------------------------
0 rows selected
```

Ah tiens? Ca ne marche pas? Le problème ici est que j'ai précisé dans ma clause `WHERE` *Oiseaux* avec un *O* majuscule. Or j'avais mis une minuscule lors de la saisie des données. Comme Derby fait une comparaison sensible à la casse, aucune donnée ne concorde. Ré-écrivons la requête avec *oiseaux* en minuscules:

```
ij> SELECT Classe.Designation, Espèce.Nom
>     FROM Classe
>     INNER JOIN Espèce ON Classe.Id = Espèce.Classe
>     WHERE Classe.Designation = 'oiseaux';
DESIGNATION                    |NOM

--------------------------------------------------------------------------------
oiseaux                        |Pie bavarde
oiseaux                        |Merle noir
oiseaux                        |Coucou gris
oiseaux                        |Hirondelle rustique
oiseaux                        |Crocodile du Nil

5 rows selected
```

Mieux. Mais que fait ce crocodile ici? J'ai du faire une petite erreur lors de la saisie. Nous allons corriger ceci dès la prochaine section.

Sous-requêtes

```
ij> SELECT * FROM Espèce WHERE Nom LIKE 'Croco%';
ID         |NOM                                                    |CLASSE
------------------------------------------------------------------------------
10         |Crocodile du Nil                                       |1
```

Et oui: les reptiles, ça n'est pas la classe 1, c'est la classe... euh... combien déjà? Inutile de faire appel à votre mémoire, puisque Derby supporte les requête imbriquées:

```
ij>  UPDATE Espèce
>      SET Classe = (SELECT Id FROM Classe WHERE Designation = 'reptiles')
>      WHERE Nom LIKE 'Croco%';
1 row inserted/updated/deleted

ij> SELECT Espèce.* FROM Espèce INNER JOIN Classe ON Espèce.Classe = Classe.Id
>     WHERE Classe.Designation = 'reptiles';
ID         |NOM                                                    |CLASSE
------------------------------------------------------------------------------
7          |Tortue des bois                                        |3
8          |Lézard des murailles                                   |3
10         |Crocodile du Nil                                       |3

3 rows selected
```

Bon nous avons vu les requêtes SELECT, INSERT et UPDATE. Juste par acquis de conscience essayons de faire un DELETE. Mais vous vous doutez bien que c'est une opération supportée par Derby.

```
ij> DELETE FROM Classe WHERE Designation = 'mammifères';
ERROR 23503: DELETE on table 'CLASSE' caused a violation of foreign key
constraint 'SQL090509011909552' for key (2).  The statement has been rolled
back.
```

Forcément, les contraintes d'intégrité référentielle nous empêchent de supprimer la classe de mammifères tant qu'il y a des espèces qui y font référence. Je dois donc supprimer d'abord les espèces de mammifères avant de supprimer cette classe. Ici aussi, comme la requête DELETE supporte les sous-requêtes, cela nous évite d'avoir à nous souvenir de l'Id des mammifères:

```
ij>  DELETE FROM Espèce
>      WHERE Classe = (SELECT Id FROM Classe WHERE Designation = 'mammifères');
2 rows inserted/updated/deleted
ij>  DELETE FROM Classe WHERE Designation = 'mammifères';
1 row inserted/updated/deleted
```

Accès concurrents

Voilà notre rapide tour d'horizon de Derby en mode "client texte" est fini. Rien de bien notable. Comme tous les SGBD-R, Derby supporte son propre sous-ensemble de SQL. Ceci dit, les requêtes de bases sont supportées sans problème.

Avant de terminer, un point est à bien comprendre. A un instant donné, une base Derby ne peut être utilisée que par une seule instance de Derby. Autrement dit, un seul programme peut être connecté simultanément à une base Derby.

Faites l'expérience de lancer `ij` dans deux shells différents en vous connectant à la même base. La seconde connexion devrait vous donner le message d'erreur suivant:

```
ij> CONNECT 'jdbc:derby:taxondb;user=herve;password=my-password';
ERROR XJ040: Failed to start database 'taxondb', see the next exception for details.
ERROR XSDB6: Another instance of Derby may have already booted the database
/home/herve/Developpement/derby/taxondb.
```

En mode embarqué, une base Derby ne peut donc pas être partagée par plusieurs applications. Si vous avez besoin d'avoir plusieurs applications qui accèdent de manière concurrente à une base Derby, il vous faudra utiliser le *serveur* livré avec la distribution de Derby.

7) Se connecter avec JDBC

Dans un premier temps, il est nécessaire de démarrer le serveur Derby, pour cela on va utiliser la classe NetworkServerControl, du pacakge org.apache.derby.drda.NetworkServerControl qui se trouve dans le fichier JAR derbynet.jar.

Le bout de code ci dessous montre comment lancer le serveur apache Derby à partir de notre application...

```
NetworkServerControl serverControl=null;
      try {
         serverControl = new NetworkServerControl();
         serverControl.start(null);
      } catch (Exception ex) {
         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
      }
```

Exemple de code Java pour une connexion à une base de donnée Apache Derby.

```
String connectionLine= "jdbc:derby:mabase;create=true;user=herve;password=monpass";
public Statement s = null;
private Connection conn = null;

Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
try {
         conn = DriverManager.getConnection(connectionLine);
         conn.setAutoCommit(true);
         s = conn.createStatement();
      } catch (SQLException ex) {
         Logger.getLogger(connectDB.class.getName()).log(Level.SEVERE, null, ex);
      }
```

Si l'on prend ce bout de code très simple, il permet de réaliser une connexion mono-utilisateur sur une base de données Derby nommé « mabase ».

pour permettre la connexion en multi-utilisateur, il suffit de remplacer la ligne de connexion par la ligne suivante :

```
String
connectionLine= "jdbc:derby://localhost:1527/mabase;create=true;user=herve;password=monpass"
;
```

Il est possible en fin d'utilisation de l'application d'arrêter le serveur Derby, pour cela on va utiliser l'api NetworkserverControl .

```
NetworkServerControl serverControl=null;
      try {
         serverControl = new NetworkServerControl();
         serverControl.shutdown();
      } catch (Exception ex) {
         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
      }
```

8) Utiliser DB Derby sous NetBeans

Configurer le SGBD Java DB

Java DB Derby est installé quand vous installez le JDK 7 ou JDK 8 (excepté sur Mac OS X). Si vous utilisez Mac OS X vous pouvez télécharger et installé Java DB manuellement ou utiliser Java DB qui est installé avec Jave EE avec l'installateur de l'IDE NetBeans.

Si vous avez le serveur GlassFish enregistré dans votre IDE NetBeans, Java DB devrai déjà être installé.

Si vous venez de télécharger java DB.

1. Décompressez le package. Un répertoire nommé 'javadb' sera créé dans le répertoire local.
2. Sur votre système, creez un nouveau répertoire qui sera utilisé comme une répertoire home pour les instances du serveur base de données.

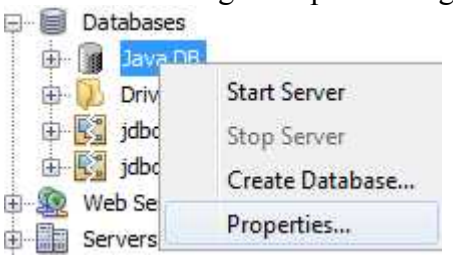
Avant de continuer plus loing, il est important de connaître les composants que l'on trouve dans le répertoire racine de la base de données Java DB:

- Le sous répertoire `demo` contient les programmes de démonstrations.
- Le sous répertoire `bin` contient les scripts permettant d'éxécuter les utilitaires et la configuration de l'environnement.
- Le sous répertoire `javadoc` contient la documentation des API.
- Le sous répertoire `docs` contient la documentation de Java DB.
- Le sous répertoire `lib` contient les fichiers JAR de la base de données Java DB.

Enregistrer le moteur SGBD dans l'IDE NetBeans

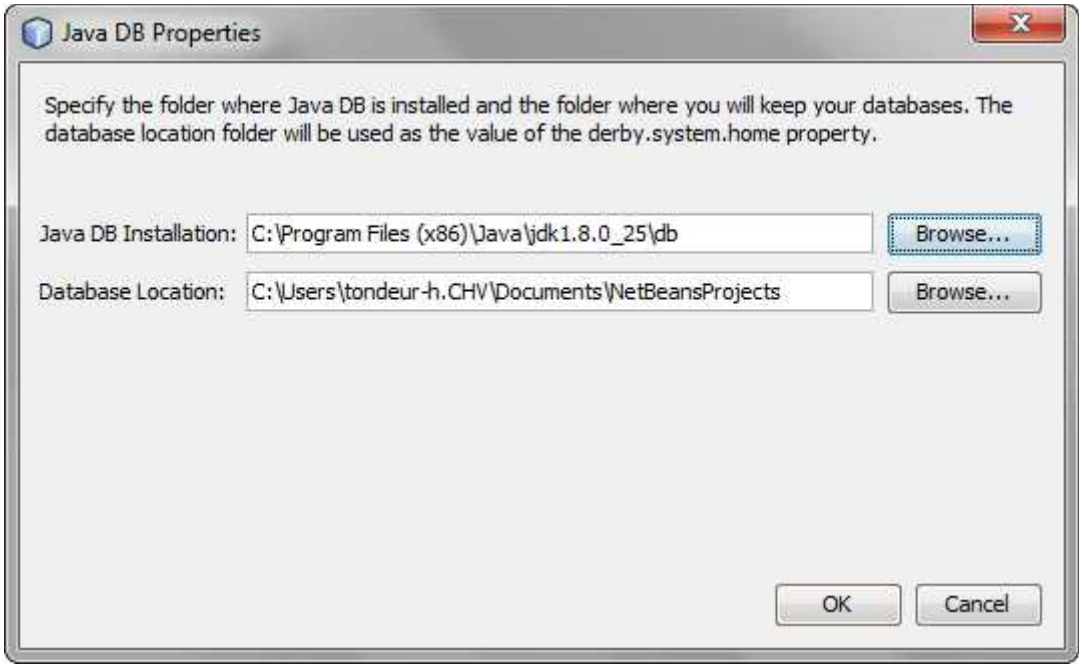
IL est parfois nécessaire d'enregistrer Java DB dans Netbeans, notamment si vous voulez utilisez une installation autres que celle par défaut fourni avec Netbeans, pour cela il faut suivre les étapes suivantes :

1. Dans la fenêtre Services, cliquez droit sur le nœud Java DB DATABASE et choisissez « Properties » pour ouvrir la boite de dialogue de paramétrage Java DB.



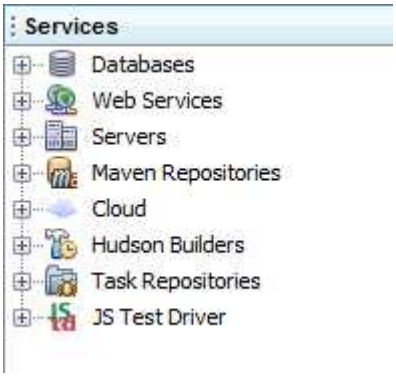
2. Dans le TextFiled « Java DB Installation », entrez le chemin racine de Java DB.
3. Dans le TextFiled « Database Location », Utilisez un chemin par defaut si celui n'est pas déjà renseigné. Cliquez sur OK

   Par exemple, le chemin par defaut pourait ressembler à `C:\Documents and Settings\username\.netbeans-derby` sur une machine Windows.



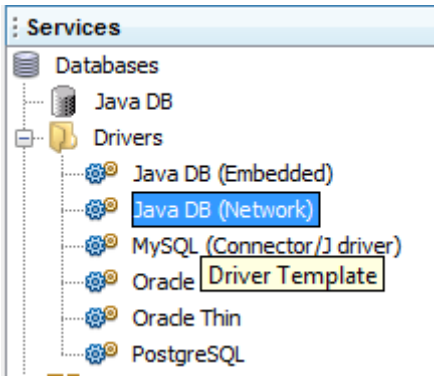
A) **<u>Creer une connection JavaDB dans NetBeans</u>**

Ouvrir le panneau des services, avec le menu « Window|Services » ctrl+5.



Ce panneau propose les différents services disponibles avec Netbeans, et notamment « Databases » qui permet de réaliser les connections et la gestion des bases de données du marchés, comme mysql, postgres, Oracle et bien entendu JvaDB.

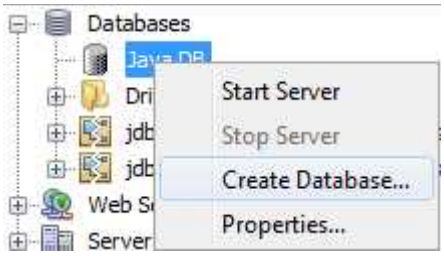
Ouvrir le feuille de l'arbre des services comme ci dessous



Vous retrouver dans cet arbre, les éléments « Drivers » qui propose les drivers installés sur votre machine.

C'est le cas pour Java DB embedded et serveur.

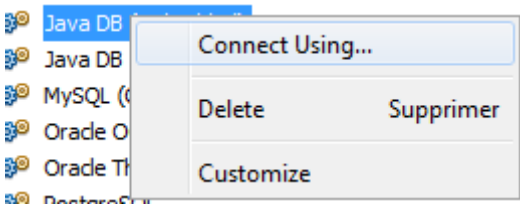
Pour créer une base de données en mode client-serveur, il est également possible de créer la base de données plus simplement en cliquant droit sur le menu « JavaDB » et »Create database ».



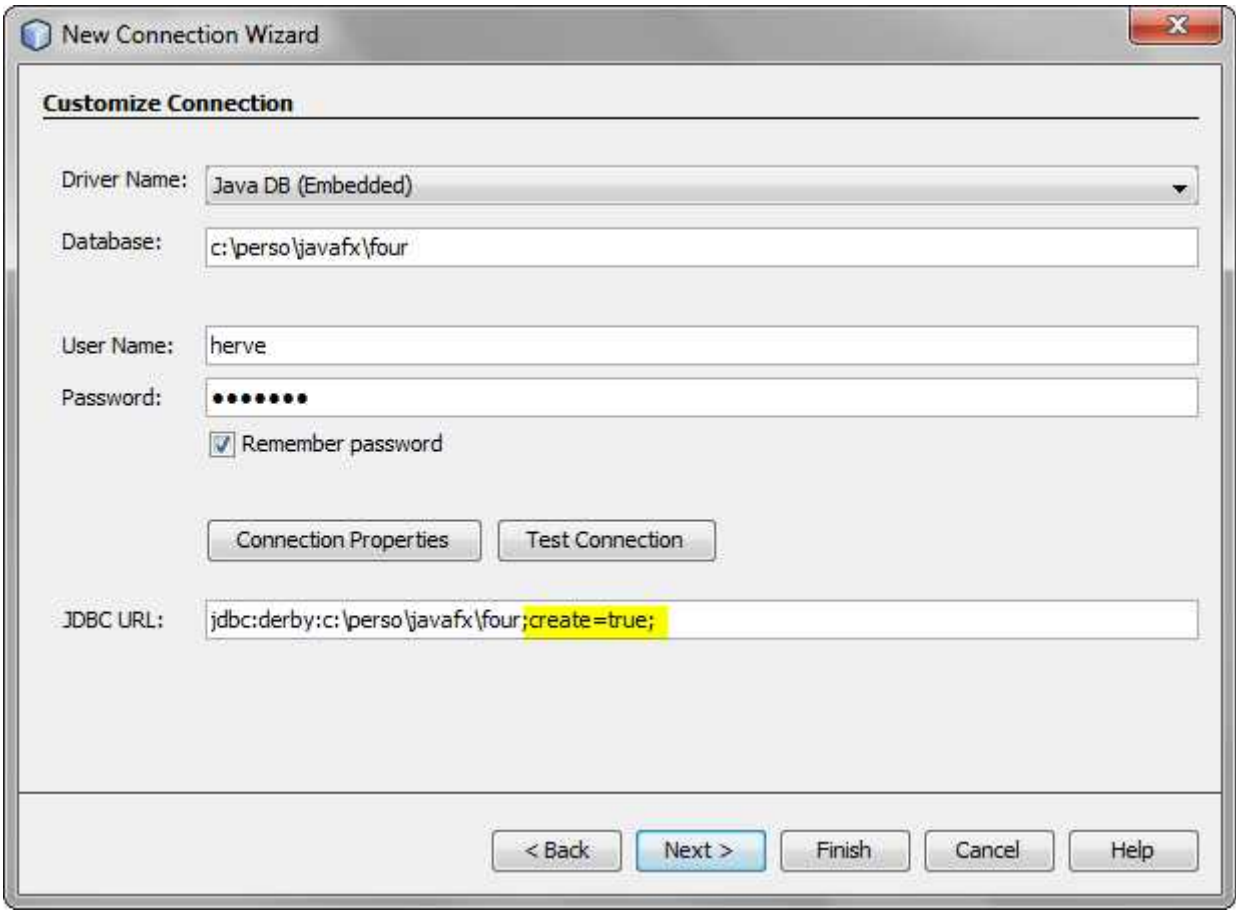
Voir création d'une base de données en mode client-serveur...

B) **Création d'une base de donnée embedded**

Cliquer droit sur la feuille nommée « Java DB (Embedded), ceci permet d'afficher un menu popup, avec comme menu « Connect Using... »

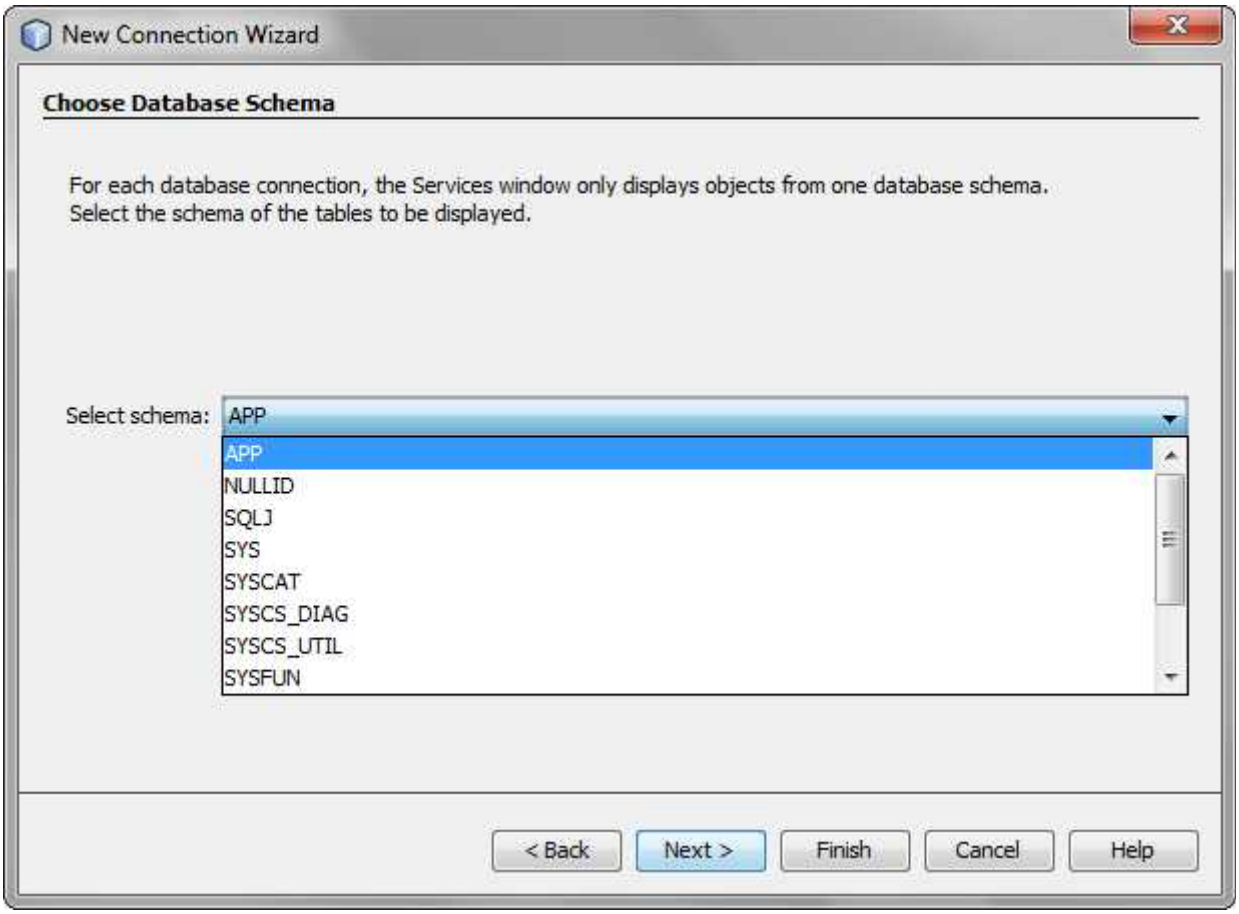


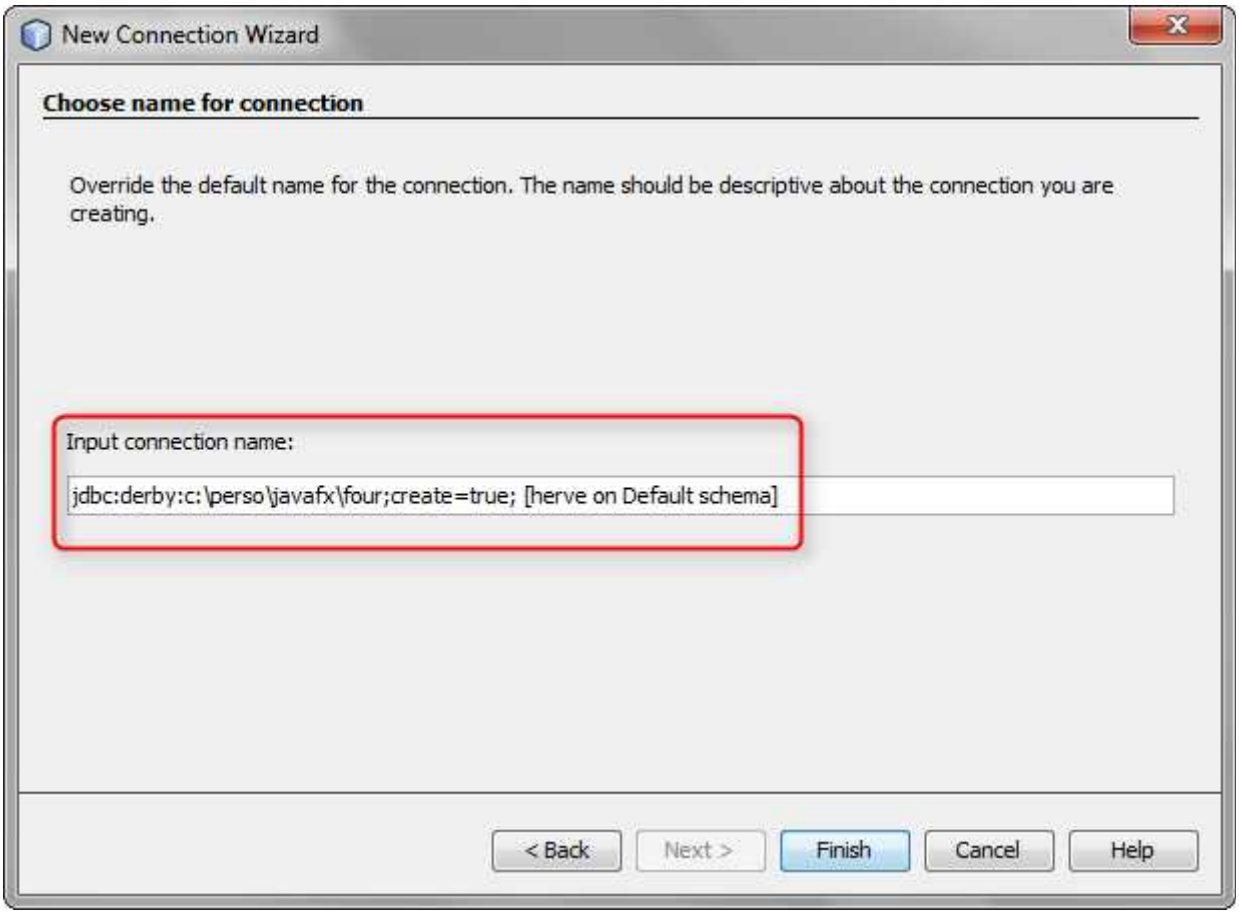
On sélectionne ce menu, ce qui permet d'afficher le dialogue ci dessous que l'on remplira avec les informations utiles comme ci dessous...



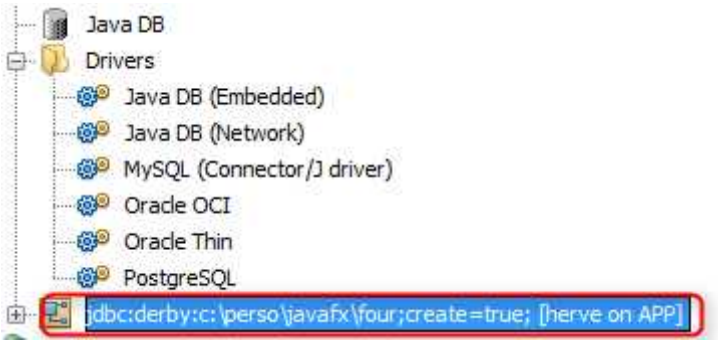
Saisir le chemin de la base de donnée (ou vont être écrit les fichiers datas)
Le login et mot de passe, il est impératif d'ajouter «;create=true ; » a l'url JDBC comme sur la capture d'écran ci dessus lors de la première création de la base de données.

Cliquez sur « Next », vous propose le dialogue suivant qui présente les schémas disponibles dans la base de données que vous allez créer, notre application doit utiliser le schéma APP par défaut, les autres schémas restant réservés au système de la SGBD.


New Connection Wizard
Choose Database Schema
For each database connection, the Services window only displays objects from one database schema.
Select the schema of the tables to be displayed.
Select schema:
APP
APP
NULLID
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
< Back
Next >
Finish
Cancel
Help

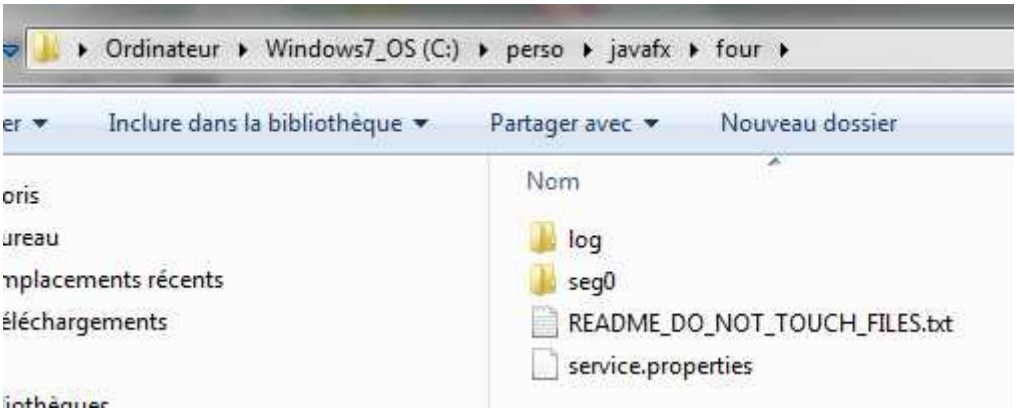


Dés validation de cette création de connexion, on retrouve notre ligne de connexion dans l'arbre des connections.

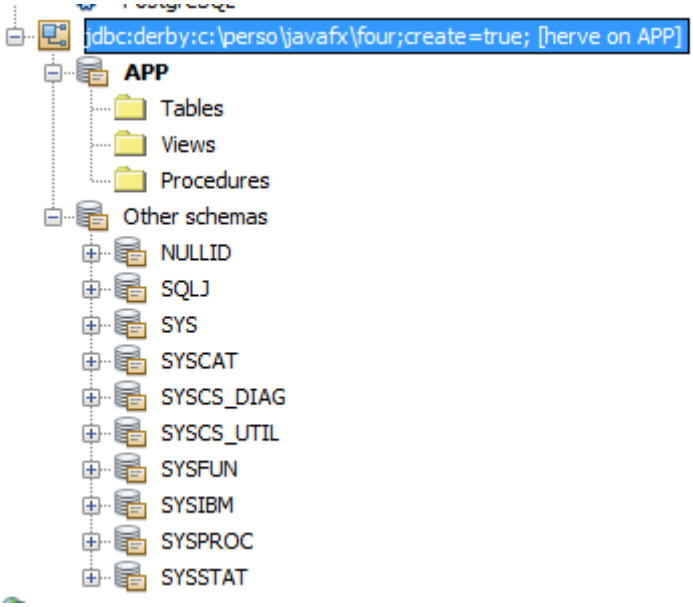


Ou se trouvent mes fichiers de données ?

Simplement dans le répertoire de destination que vous avez indiqué lors de la création de la base de données pour nous dans notre exemple : c:/perso/javfx/four

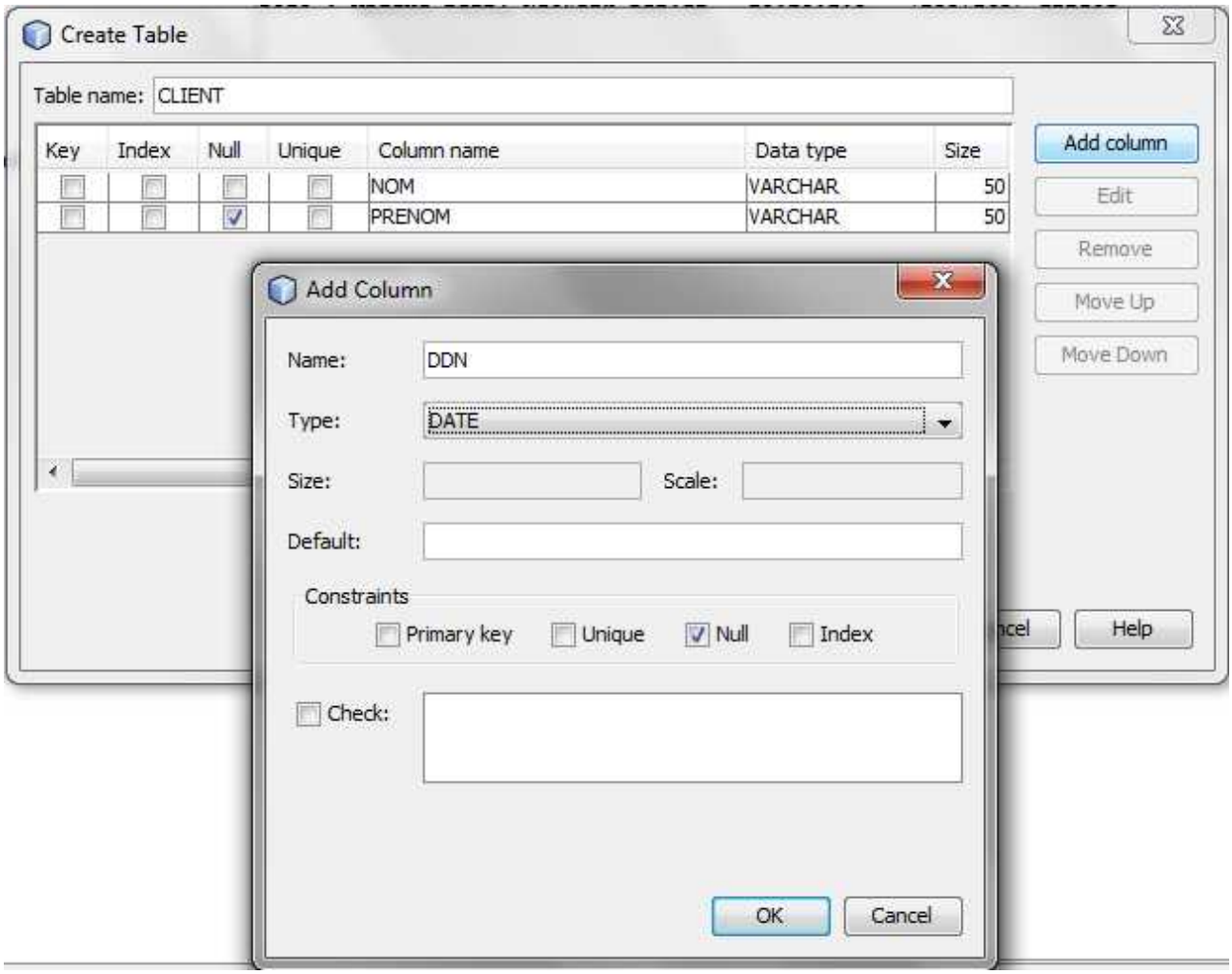


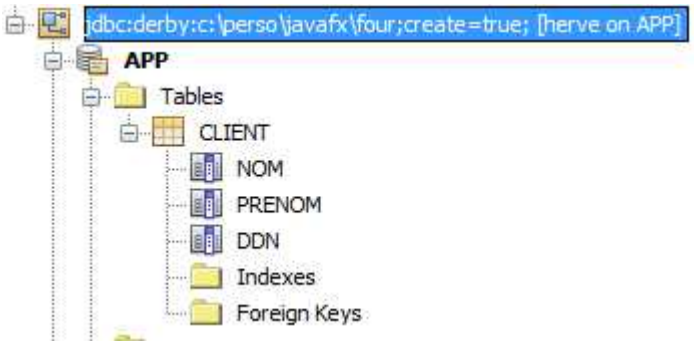
Il est possible maintenant d'accéder aux éléments de ma base de données embarquée.



Un clic droit sur l'élément « TABLE » vous permet de créer une nouvelle table

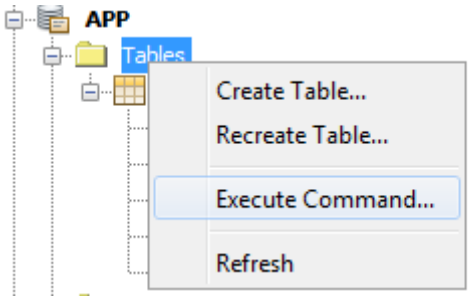
Ceci permet d'accéder a un dialogue de conception d'une table en mode graphique, le bouton « AddColum », permet d'ajouter des champs à la table...

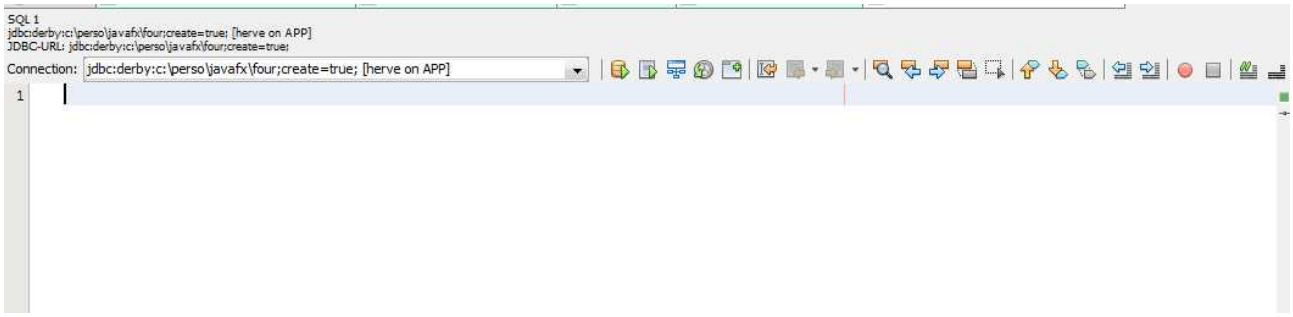


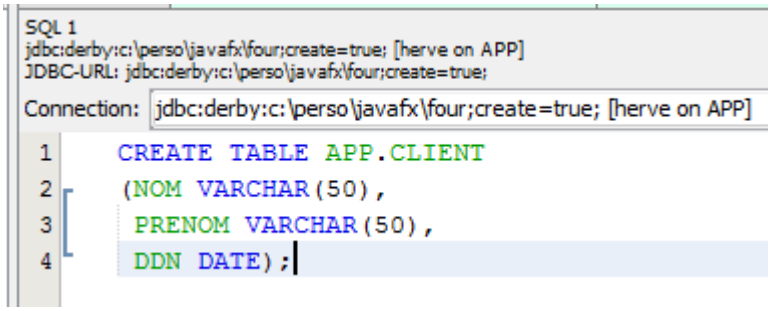


C) **Créer une table par commande SQL**

Sue l'élément TABLE, par un clic droit vous obtenez également le menu « Execute Command... », ce menu vous permet d'ouvrir un dialogue de saisie d'une requête SQL.

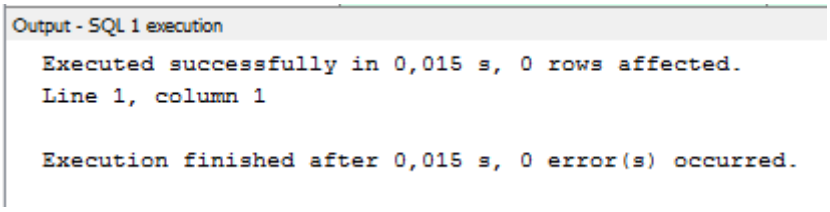




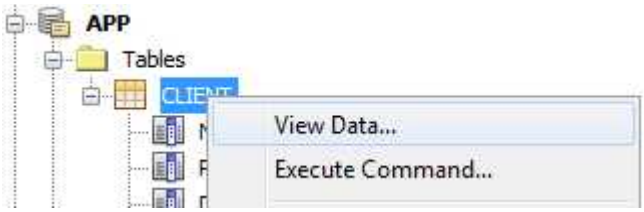


Le bouton Exécute SQL permet d'exécuter la requête…

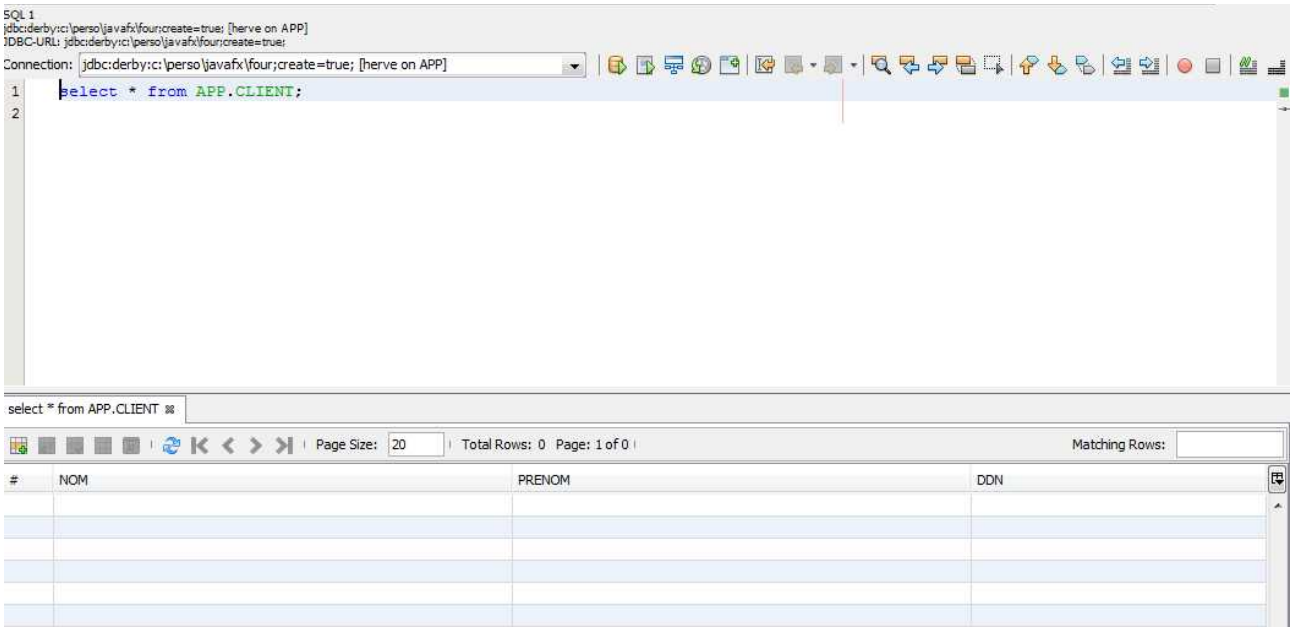
Avec si réussite un message sur la console Netbeans



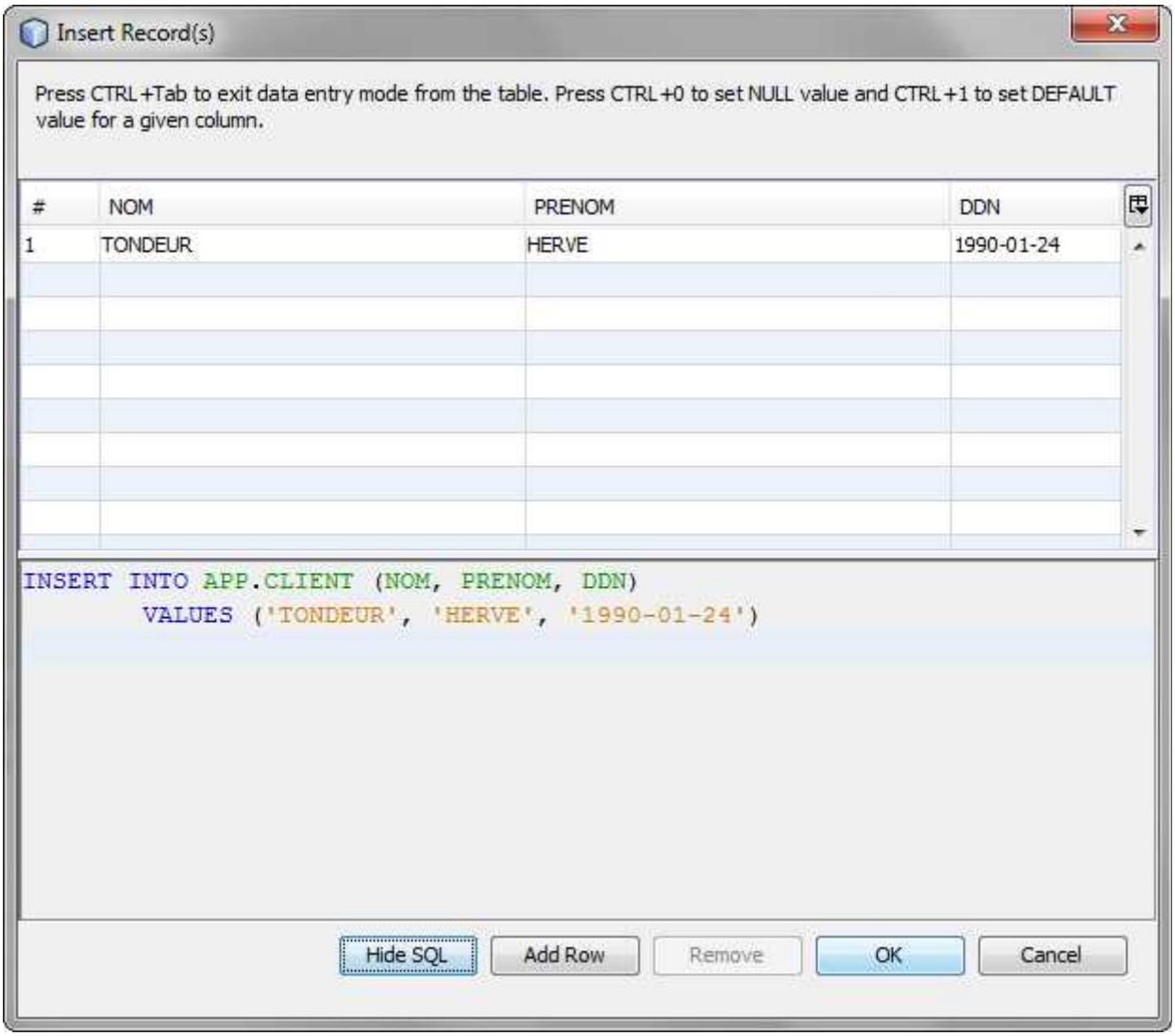
D) **<u>Ajouter des données dans une table</u>**



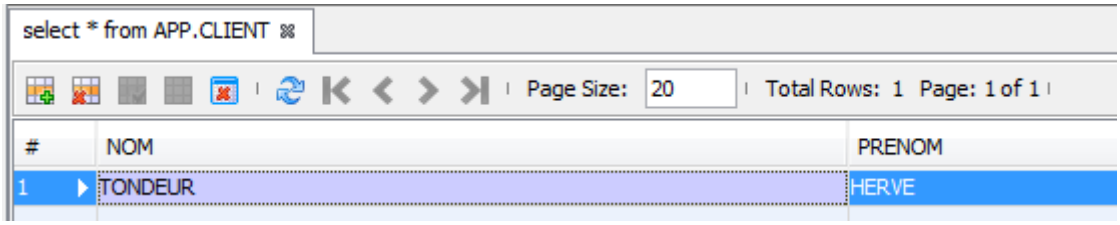
Le menu « View Data... » dans la table sur laquelle on a réalisé le clic droit permet d'afficher un dialogue présentant une zone requête et une grille avec les données présentes dans la table…



Il est donc possible d'insérer des données par requête ou par insertion directe via l'outil grille de données…

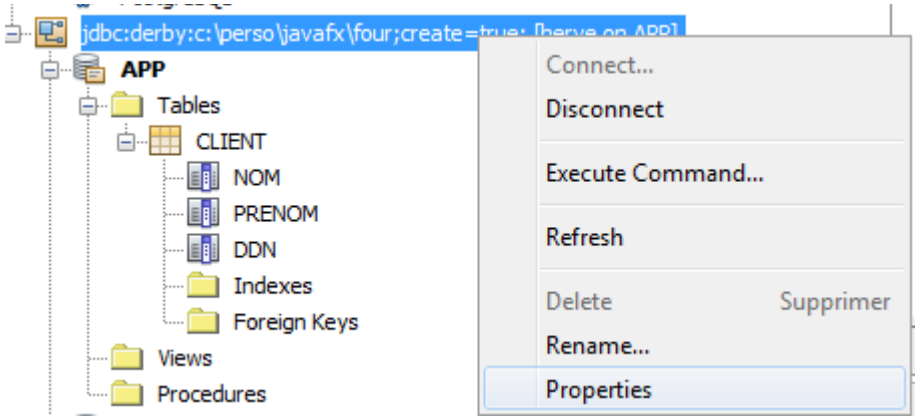


Il en est de même pour la suppression, modification, le commit, et la navigation dans les données, la barre d'outil présente l'ensemble des boutons nécessaire pour réaliser ces actions.

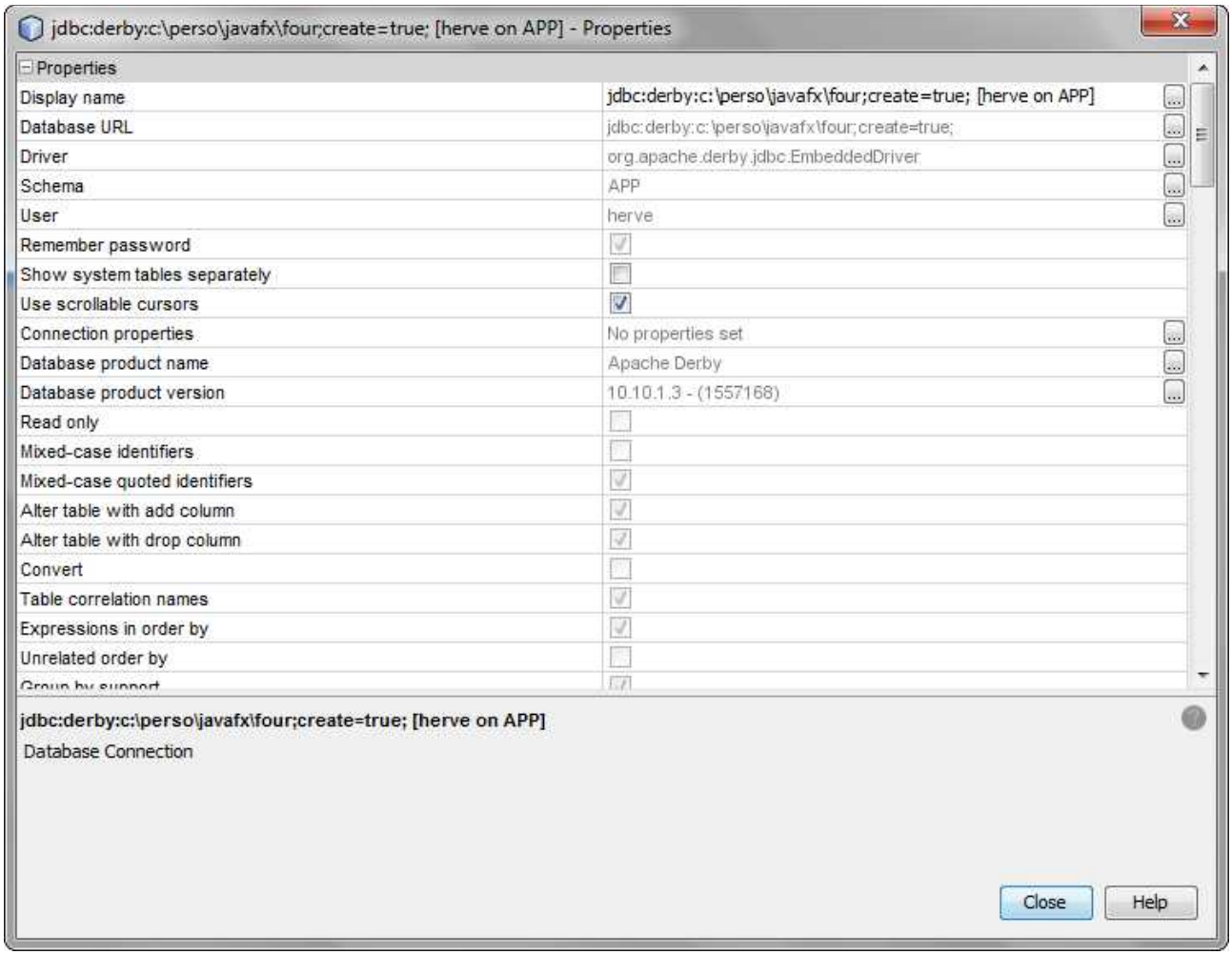


Bien entendu, toutes ces actions peuvent également être réalisées par requêtes SQL.

E) **Obtenir les propriétés de la base de données.**



Un clic droit sur le nom de la connexion, permet d'accéder au menu propriétés, celui ci propose le dialogue ci dessous.



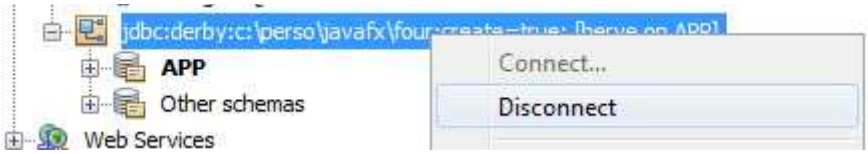
Les données a retenir sur ce dialogue sont :
- Database URL
- Driver
- Schema

- User

qui représentent les informations dont on aura besoin pour notre développement JDBC.

F) **Déconner une base de donnée locale.**

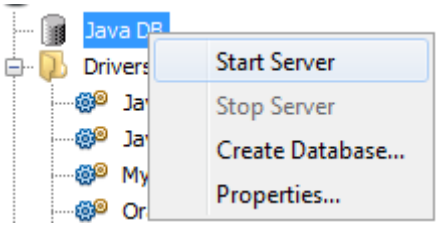
Le menu « deconnect » obtenu par un clic droit sur la connexion de notre base de données Derby, permet de déconnecter la base de données et de libérer les ressources, ainsi que le verrou sur les données.



G) Connexion **à une base de données Derby en client-Serveur**

Première méthode

Il est nécessaire de démarrer le serveur Java DB avant de réaliser cette action, pour cela il suffit de faire un click droit sur l'élément JavaDB et de faire « Start Server »

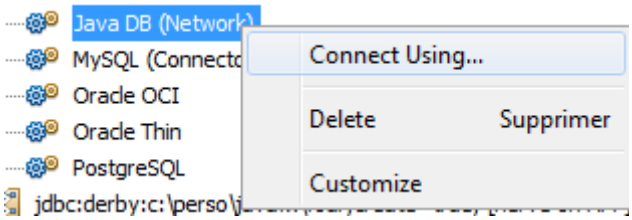



```
Sat Jan 24 11:14:37 CET 2015 : Le gestionnaire de sécurité a été installé à l'aide de la stratégie de sécurité de serveur de base.
Sat Jan 24 11:14:37 CET 2015 : Apache Derby Network Server - 10.10.1.3 - (1557168) démarré et prêt à accepter les connexions sur le port 1527
```

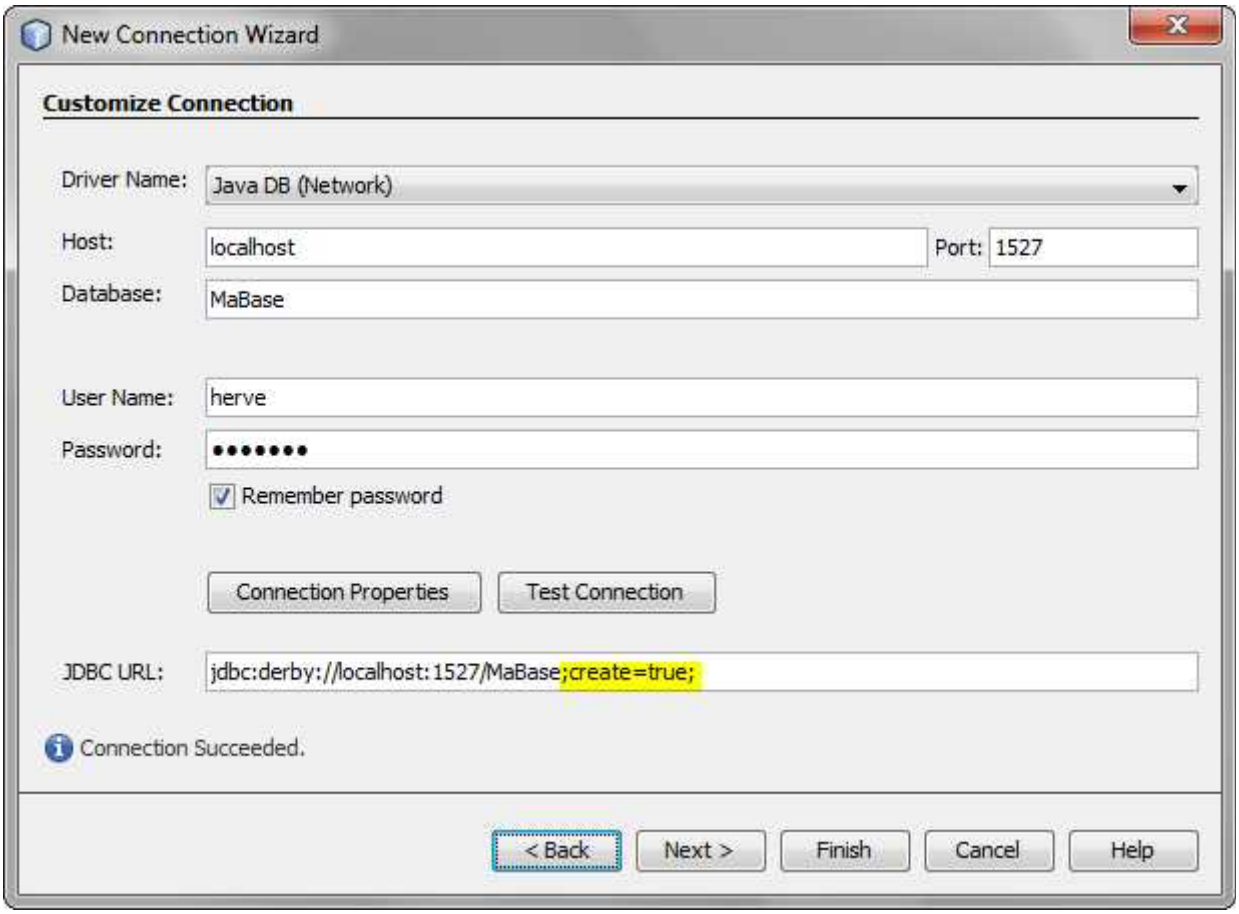
L'affichage ci dessus nous indique bien que le serveur est correctement démarré sur l'IP locale de la machine et sur le port 1527

ce qui nous permet maintenant de créer notre base de données en mode client-serveur.

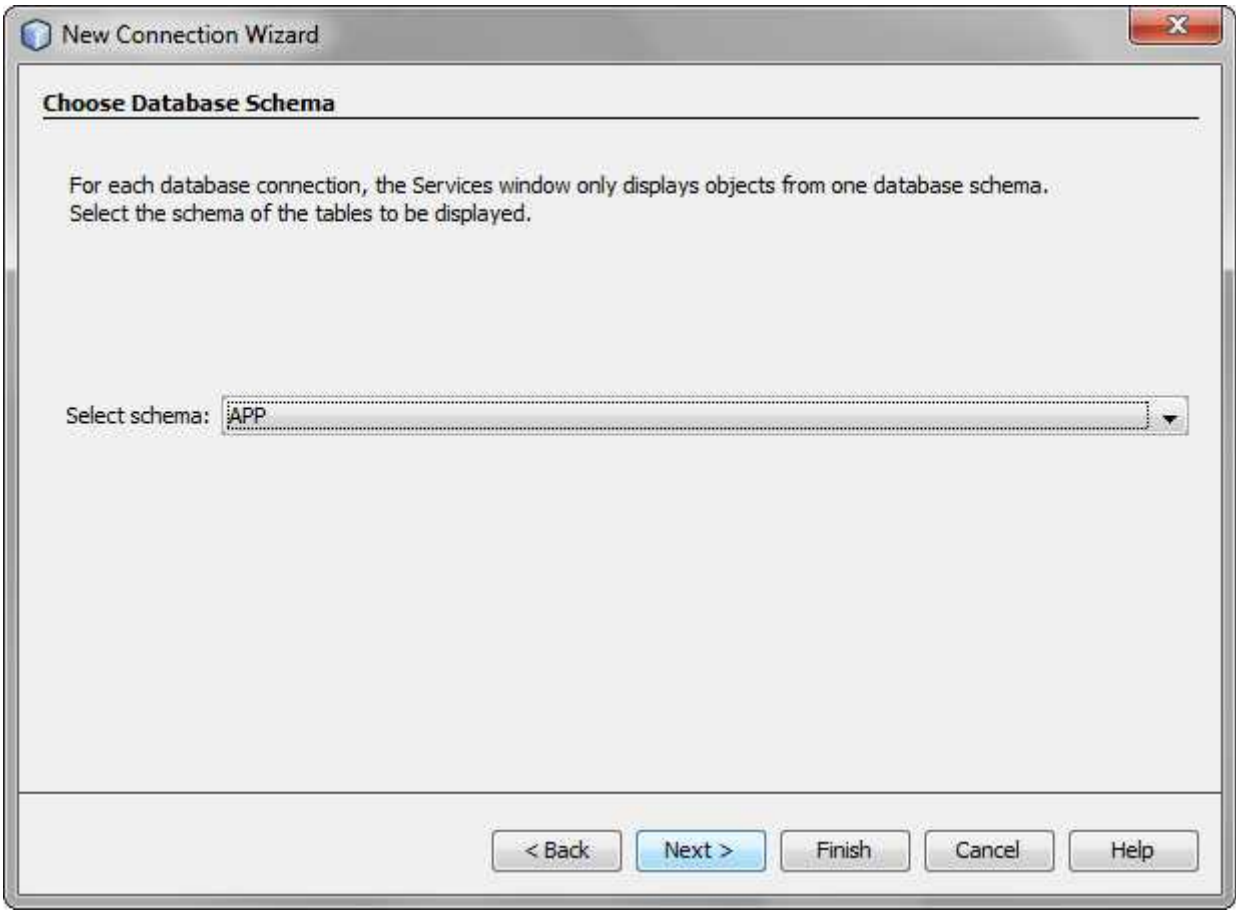
Un clic droit sur le Driver « Java DB (Network) permet d'obtenir le menu ci dessous, la sélection du menu « Connect Using... » permet de créer notre base de données.

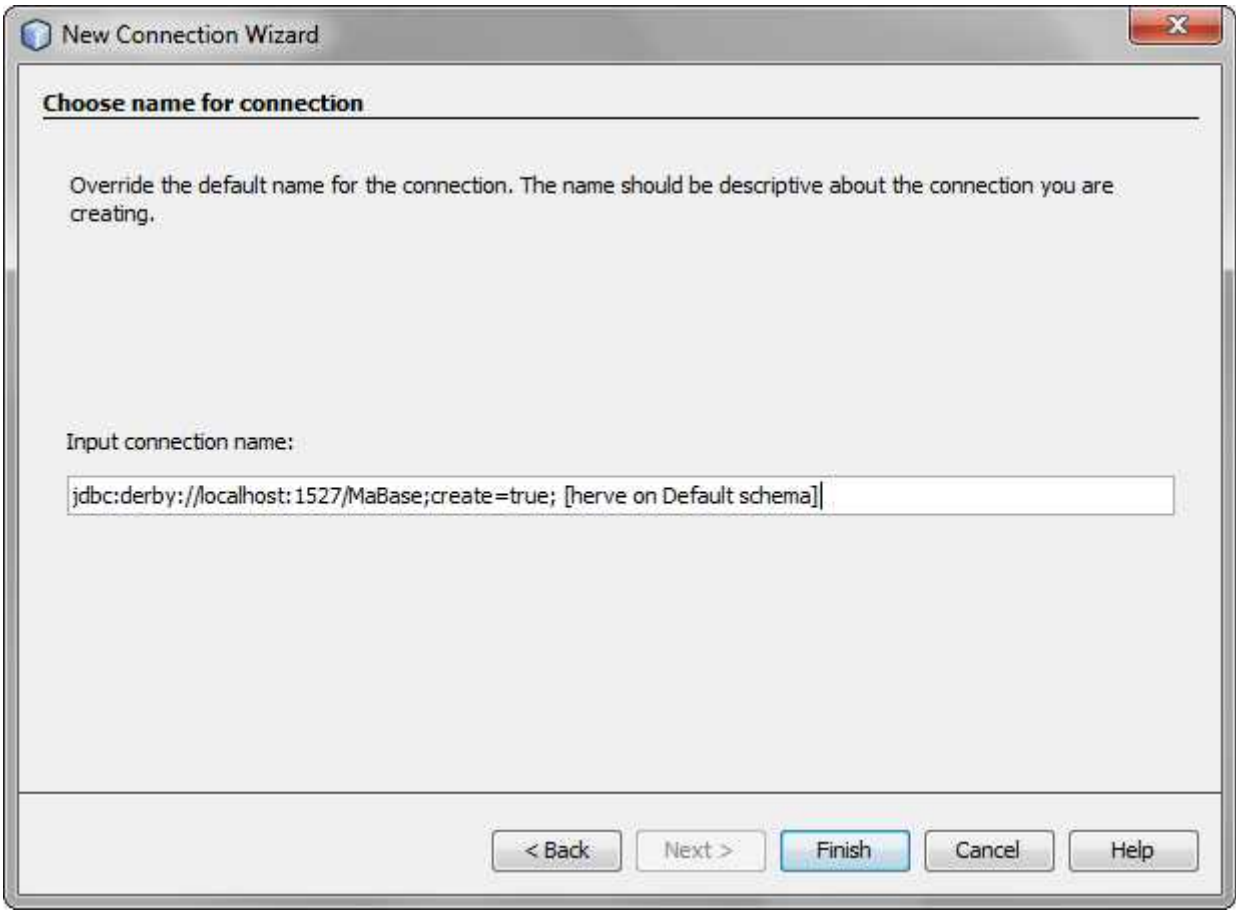


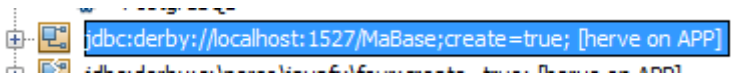
Il faut renseigner le dialogue comme ci dessous, sans oublier de mettre l'information «;create=true ; » sur la ligne de connexion.



Le bouton « Next » permet d'accéder au dialogue suivant qui présente le sélecteur de schéma, on choisira également ici APP.


New Connection Wizard
Choose Database Schema
For each database connection, the Services window only displays objects from one database schema.
Select the schema of the tables to be displayed.
Select schema:
APP
< Back
Next >
Finish
Cancel
Help



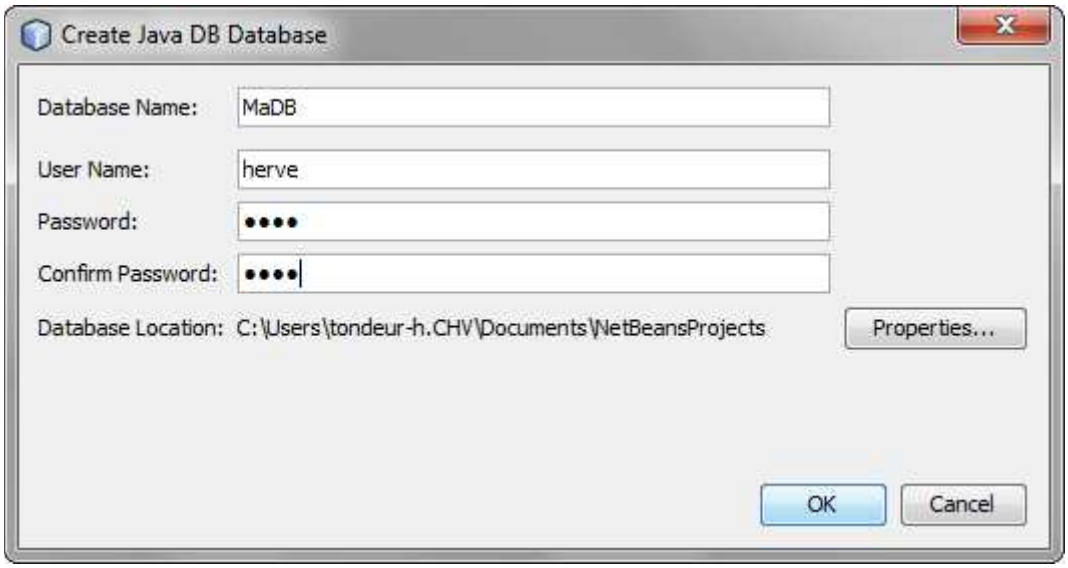


Ou se trouvent mes fichiers de données ?

Dans le répertoire de repository NetBeans…

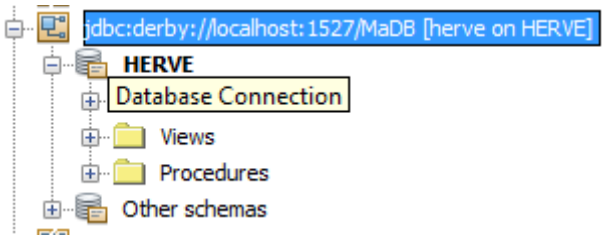
Seconde méthode

Un clique droit sur le menu « JavaDB » donne accès au menu « Create dataBase », ce menu ouvre le dialogue ci dessous.



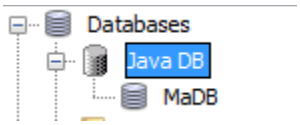
Il suffit de remplir les différents champs comme ci dessus et valider par « OK »

On récupère, immédiatement une ligne de connexion sur cette base de données.



Vous remarquerez que le schéma par défaut cette fois ci est HERVE, le nom du compte créer, bien plus intéressant.

Ainsi que le nom de la base de données qui apparaît en dessous de la feuille « Java DB »



Un clique droit sur le nom de la base de données, permet d'ouvrir un menu permettant soit la connexion, soit la suppression de cette base de données.

Nb : Pour ce qui est de la gestion de la base de données, c'est comme d'habitude, ainsi que pour la récupération des informations de connections sur la base de données.

9) Exemple de programme Java console d'accés JDBC a une base de données Derby Embedded

```
package dbtest;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DbTest {

   Connection connect;

   public DbTest() {
      try {
         //charger le drivers JDBC
         Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
      } catch (ClassNotFoundException ex) {
         Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
      }

      //se connecter a la base de données
      String DBurl = "jdbc:derby:C:\\Users\\tondeur-
h.CHV\\Documents\\NetBeansProjects\\DbTest\\dbtest;create=true;";
      try {
         connect = DriverManager.getConnection(DBurl);
      } catch (SQLException ex) {
         Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
      }

      //inserer des donnees

      String req="INSERT INTO APP.client(nom,prenom) VALUES('TONDEUR','HERVE')";

      Statement stmt;
      try {
         stmt = connect.createStatement();
         int nbMaj = stmt.executeUpdate(req);
         System.out.println("Nombre de lignes maj… "+nbMaj) ;
      } catch (SQLException ex) {
         Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
      }
```

```
        //afficher les données sur la console

        try{
            stmt=connect.createStatement();;
        req="SELECT nom,prenom FROM APP.client";
        ResultSet rs=stmt.executeQuery(req);

        while (rs.next()){
            System.out.println("NOM="+rs.getString("nom")+" PRENOM="+rs.getString("prenom"));
        }
        } catch (Exception e) {e.printStackTrace();}

    }


    public static void main(String[] args) {
        new DbTest();
    }

}
```

Résultat :

```
Nombre de lignes maj… 1
NOM=TESTNOM PRENOM=TESTPRENOM
NOM=TONDEUR PRENOM=HERVE
BUILD SUCCESSFUL (total time: 0 seconds)
```

10) Exemple de programme Java console d'accès JDBC a une base de données Derby en mode client-serveur

```
package dbtest;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.derby.drda.NetworkServerControl ;

public class DbTest {
```

```
    Connection connect;

    public DbTest() {

//demarrer le serveur Java DB
NetworkServerControl serverControl=null;
        try {
            serverControl = new NetworkServerControl();
            serverControl.start(null);
        } catch (Exception ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }

    try {
            //charger le drivers JDBC
            Class.forName("org.apache.derby.jdbc.ClientDriver");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
        }

        //se connecter a la base de données
        String DBurl = "jdbc:derby://127.0.0.1:1527/testDB;create=true;";
        try {
            connect = DriverManager.getConnection(DBurl);
        } catch (SQLException ex) {
            Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
        }

        //inserer des donnees

        String req="INSERT INTO APP.client(nom,prenom) VALUES('TONDEUR','HERVE')";

        Statement stmt;
        try {
            stmt = connect.createStatement();
            int nbMaj = stmt.executeUpdate(req);
            System.out.println("Nombre de lignes maj… "+nbMaj);
        } catch (SQLException ex) {
            Logger.getLogger(DbTest.class.getName()).log(Level.SEVERE, null, ex);
        }

        //afficher les données sur la console

        try{
            stmt=connect.createStatement();;
        req="SELECT nom,prenom FROM APP.client";
        ResultSet rs=stmt.executeQuery(req);
```

```
        while (rs.next()){
           System.out.println("NOM="+rs.getString("nom")+" PRENOM="+rs.getString("prenom"));
        }
        } catch (Exception e) {e.printStackTrace();}


//arreter le serveur Java DB
        try {
           serverControl.shutdown();
        } catch (Exception ex) {
           Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }


  }

    public static void main(String[] args) {new DbTest();}
}
```

Il est important pour que ces programmes fonctionnent, d'importer les librairies suivantes :