Premiers pas

Dans ce chapitre, nous ferons un tour rapide des fonctionnalités offertes par CouchDB et nous familiariserons avec *Futon*, l'interface d'administration intégrée dans le moteur. Nous créerons notre premier document et testerons le concept de vues. Mais avant de commencer, référez-vous à **l'annexe D, Installation depuis les sources (source.html)** et cherchez-y votre système d'exploitation. Vous devrez suivre les instructions qui s'y trouvent pour installer CouchDB avant de poursuivre la lecture de ce chapitre.

L'ensemble des systèmes est opérationnel !

Examinons l'*Application Programming Interface (API)* à l'aide de l'utilitaire en ligne de commande *curl*. Notez qu'il s'agit d'une manière parmi d'autres de s'adresser à CouchDB et que nous vous en indiquerons de nouvelles dans la suite de l'ouvrage. Ce qui est intéressant avec *curl*, c'est qu'il vous permet de forger votre requête HTTP et de voir ce qui se trouve « sous le capot » de votre base de données.

Assurez-vous que CouchDB est démarré et exécutez :

```
curl http://127.0.0.1:5984/
```

Cela envoie une requête de type `GET` à l'instance de CouchDB que vous venez d'installer.

La réponse devrait ressembler à :

```
{"couchdb":"Welcome","version":"0.10.1"}
```

Pas très spectaculaire. CouchDB vous souhaite le bonjour et indique sa version.

Ensuite, nous pouvons lister les bases :

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

La seule chose que nous avons ajoutée à la requête précédente est la chaîne de caractères `_all_dbs`.

La réponse devrait ressembler à :

```
[]
```

Ah ! oui ! nous n'avons pas encore créé de base de données. Nous récupérons donc une liste vide.

> La commande `curl` envoie une requête `GET` par défaut. Vous pouvez produire des requêtes `POST` avec `curl -X POST`. Afin de nous y retrouver dans l'historique de la console, nous utilisons l'option `-X` même pour les requêtes de type `GET`. Par la suite, si nous voulons envoyer la requête en `POST`, il suffit de changer la méthode.
>
> HTTP effectue davantage d'opérations sous le capot que celles que vous voyez ici. Si vous cherchez à voir tout ce qui passe sur le câble, ajouter l'option `-v` (c.-à-d. `curl -vX GET`) et vous verrez la tentative de connexion au serveur, les en-têtes de la requête et ceux de la réponse. Très utile pour déboguer !

Créons une base de données :

```
curl -X PUT http://127.0.0.1:5984/baseball
```

CouchDB va répondre :

```
{"ok":true}
```

La requête précédente de récupération de la liste des bases devient plus utile :

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

```
["baseball"]
```

> Le moment est propice pour évoquer *JavaScript Object Notation (JSON)*, le format de données compris par CouchDB. JSON est un format d'échange de données léger basé sur la syntaxe de JavaScript. Puisque JavaScript est intégré à votre navigateur web, cela en fait un client idéal.
>
> Les crochets (`[]`) dénotent une liste ordonnée et les accolades (`{}`) indiquent un tableau clé/valeur. Les clés doivent être des chaînes de caractères délimitées par des guillemets droits et doubles (`"`) tandis que les valeurs peuvent être des chaînes de caractères, des nombres, des booléens, des listes ou des tableaux. Pour plus de détails, référez-vous à l'**annexe E, Notions de JSON (json.html)**.

Créons une autre base de données :

```
curl -X PUT http://127.0.0.1:5984/baseball
```

CouchDB va répondre :

```
{"error":"file_exists","reason":"The database could not be created, the file already exists."}
```

Nous avons déjà une base qui porte ce nom, donc CouchDB nous renvoie une erreur. Recommençons en changeant le nom :

```
curl -X PUT http://127.0.0.1:5984/plankton
```

CouchDB va répondre :

```
{"ok":true}
```

Récupérons la liste des bases :

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

CouchDB va répondre :

```
["baseball", "plankton"]
```

Pour simplifier les choses, supprimons cette seconde base :

```
curl -X DELETE http://127.0.0.1:5984/plankton
```

CouchDB va répondre :

```
{"ok":true}
```

La liste des bases redevient celle d'avant :

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

CouchDB va répondre :

```
["baseball"]
```

Par souci de concision, nous passons sur l'exploitation des documents ; la prochaine section l'abordera avec un procédé plus simple. Dans les exemples qui suivent, gardez à l'esprit que ce qui est généré « sous le capot » correspond exactement à ce que vous venez de voir : tout est fait à l'aide de requêtes `GET`, `PUT`, `POST`, et `DELETE` sur une URI.

Bienvenue à bord de Futon

Après avoir vu l'API bas-niveau de CouchDB, essayons-nous à Futon, l'interface d'administration intégrée dans le moteur. Futon permet d'exploiter toutes les fonctionnalités de CouchDB et rend aisée l'utilisation des fonctionnalités avancées. À l'aide de Futon, nous pouvons créer et détruire des bases de données, consulter et éditer des documents, créer et parcourir les vues MapReduce et déclencher la réplication entre les bases de données.

Pour accéder à Futon depuis votre navigateur web, allez sur :

```
http://127.0.0.1:5984/_utils/
```

Si vous utilisez la version 0.9 ou supérieure, vous devriez voir quelque chose ressemblant à la **figure 1, La page d'accueil de Futon (#figure/1)**. Dans les chapitres suivants, nous nous intéresserons à l'exploitation de CouchDB par les langages côté serveur tels que Ruby ou Python. Pour lors, ce chapitre est l'occasion rêvée pour montrer un exemple d'application web servie directement par le serveur web intégré dans CouchDB ; chose qui peut vous intéresser pour vos propres applications.

La première chose à faire avec une nouvelle installation de CouchDB est d'exécuter la chaîne de tests pour vérifier que tout fonctionne bien. Cela vous garantit que les problèmes qui pourraient se poser ne sont pas dus à un problème d'installation. De plus, un test qui échoue signale qu'il y a des éléments à vérifier dans notre installation avant de tenter d'utiliser un serveur qui est peut-être vrillé. Cela nous évite de nous poser la question plus tard, lorsque les choses tournent mal.

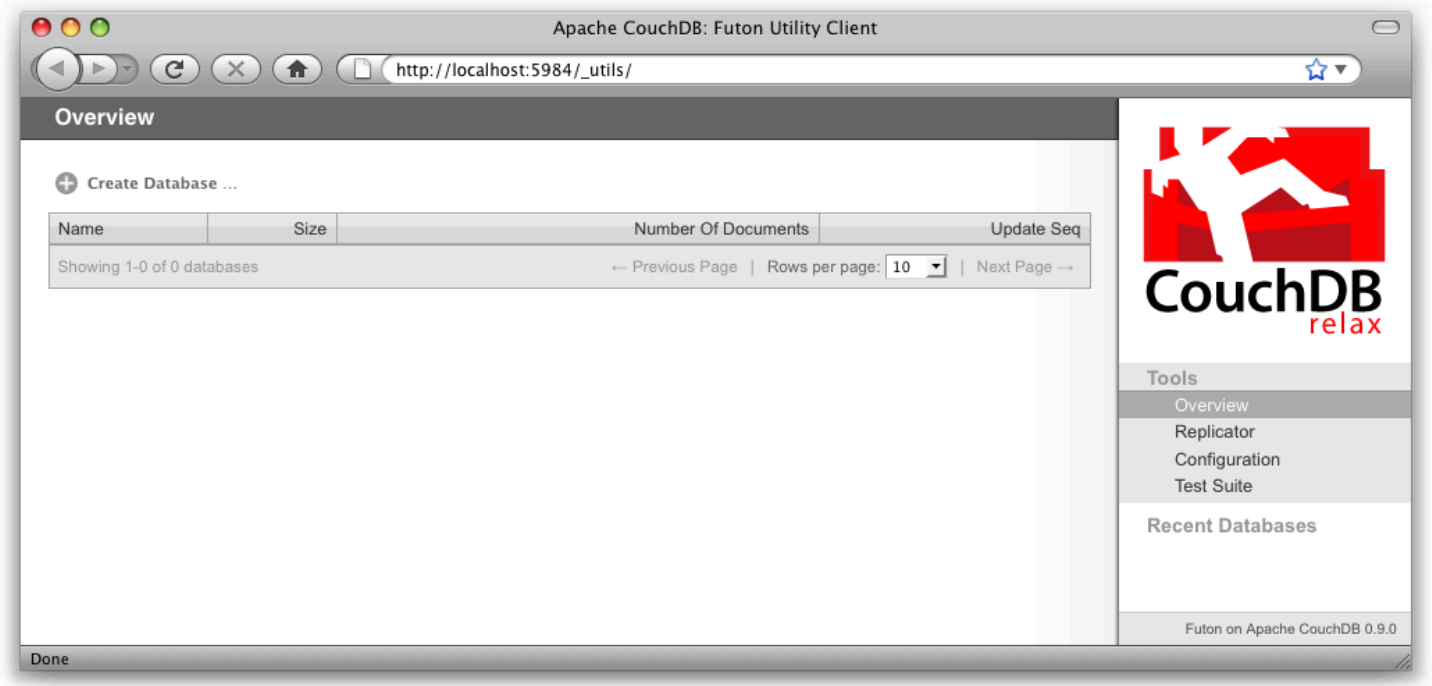

Figure 1. La page d'accueil de Futon

> Certains paramétrages du réseau (souvent rencontrés) causent l'échec du test de réplication quand il est lancé à partir de `localhost`. Vous pouvez contourner ce problème en vous adressant à *http://127.0.0.1:5984/_utils/*.

Rendez-vous sur la chaîne de test en cliquant sur « Test Suite » dans le menu latéral, puis cliquez sur « run all » en haut pour démarrer les tests. La **Figure 2, La chaîne de tests en cours d'exécution dans Futon (#figure/2)** illustre Futon en action.

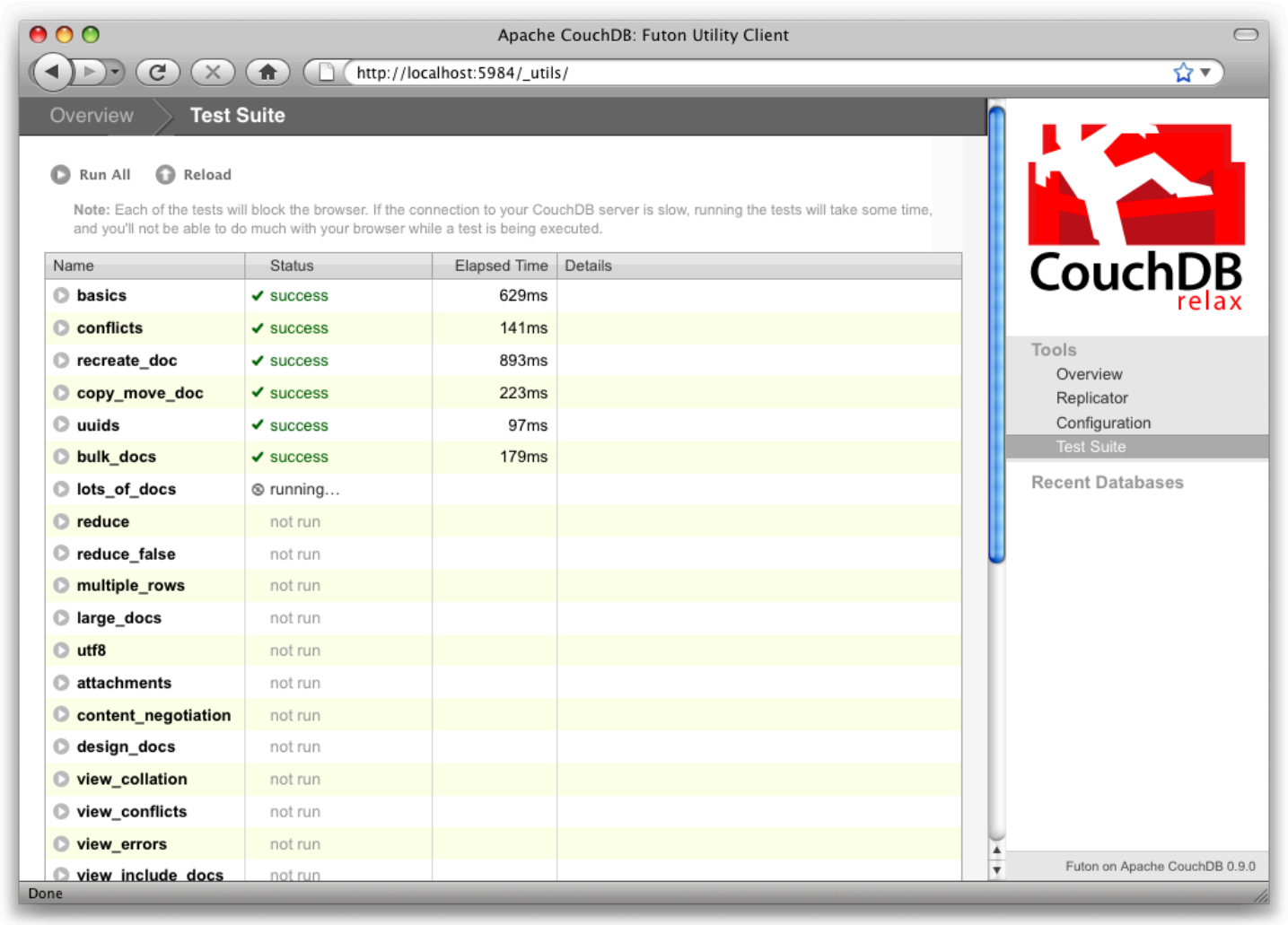

Figure 2. La chaîne de tests en cours d'exécution dans Futon

Puisque la chaîne de tests est exécutée depuis le navigateur, elle garantit à la fois que CouchDB fonctionne correctement et que la connexion de votre navigateur à la base de données est bien configurée, ce qui peut s'avérer utile pour diagnostiquer les problèmes de serveur mandataire ou d'autres intermédiaires HTTP.

> Si les résultats de la chaîne de test indiquent un grand nombre d'erreurs, reportez-vous à l'**annexe D, Installation depuis les sources (source.html)** pour trouver comment réparer votre installation.

Une fois les tests achevés, vous avez vérifié que CouchDB est opérationnel et vous êtes prêt à voir ce que Futon a à offrir.

Votre première base de données et votre premier document

Créer une base de données avec Futon est simple : depuis la page de synthèse, cliquez sur « Create database ». Saisissez ensuite le nom, ici `hello-world`, et cliquez sur le bouton « Create ».

Une fois votre base créée, Futon affiche la liste de tous les documents qu'elle contient. Au début, elle est vide (**Figure 3, Une base de données vide dans Futon (#figure/3)**), donc créons notre premier document. Cliquez sur « Create document » et sur le bouton « Create » de la *fenêtre modale* [NdT : *pop up*]. Prenez garde à laisser l'identifiant du document vide, car CouchDB va générer un UUID pour vous.

> Pour les besoins de la démonstration, l'UUID déterminé par CouchDB suffit. Toutefois, quand vous créez votre premier programme, nous vous encourageons à assigner vos propres UUIDs. En effet, si vous laissez le soin au serveur de générer vos UUIDs, que votre première requête est annulée et que vous la renvoyez, il est possible que vous génériez deux UUIDs et que vous ne receviez que le second. En assignant vos propres UUIDs, vous serez certain d'éviter la duplication d'un document.

Futon va afficher le document qui vient d'être créé, avec les seuls champs `_id` et `_rev`. Pour ajouter un champ, cliquez sur le bouton « Add field ». Nommons le `hello`. Cliquez sur l'icône vert (ou pressez « entrée ») pour finaliser l'opération. Double-cliquez sur le la colonne présentant la valeur de `hello` (positionnée par défaut à `null`) pour l'éditer.

Si vous tentez de positionner la nouvelle valeur à `world`, vous obtiendrez une erreur après avoir cliqué sur l'icône verte. C'est normal : dans CouchDB, les valeurs doivent être saisies au format JSON. Saisissez plutôt `"world"` (avec les guillemets droits et doubles), ce qui est une chaîne de caractères valide en JSON. Vous pouvez tester les autres valeurs, par exemple `[1, 2, "c"]` ou `{"foo":"bar"}`. Une fois les valeurs saisies, relevez la valeur de l'attribut `_rev` et cliquez sur « Save Document ». Le résultat devrait avoisiner celui de la **Figure 4, Un « hello world » avec Futon (#figure/4)**.

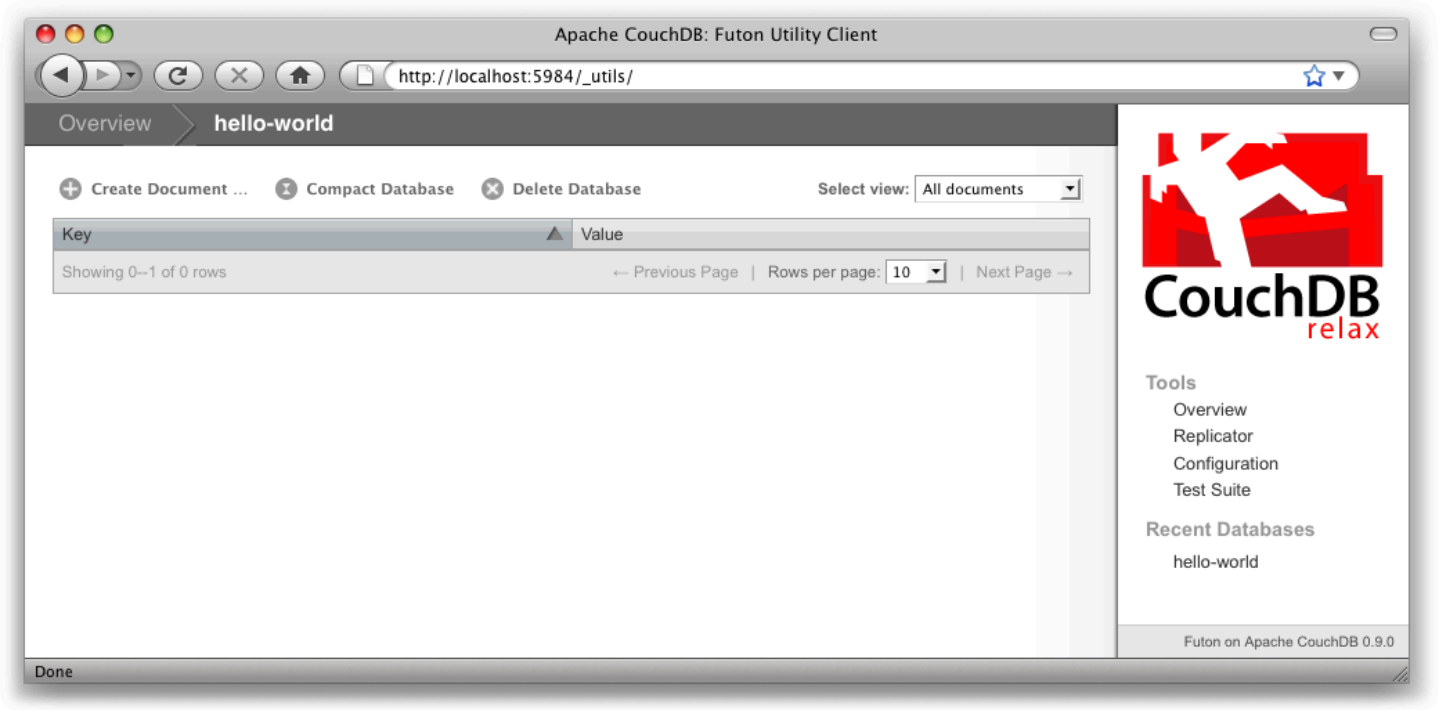

Figure 3. Une base de données vide dans Futon

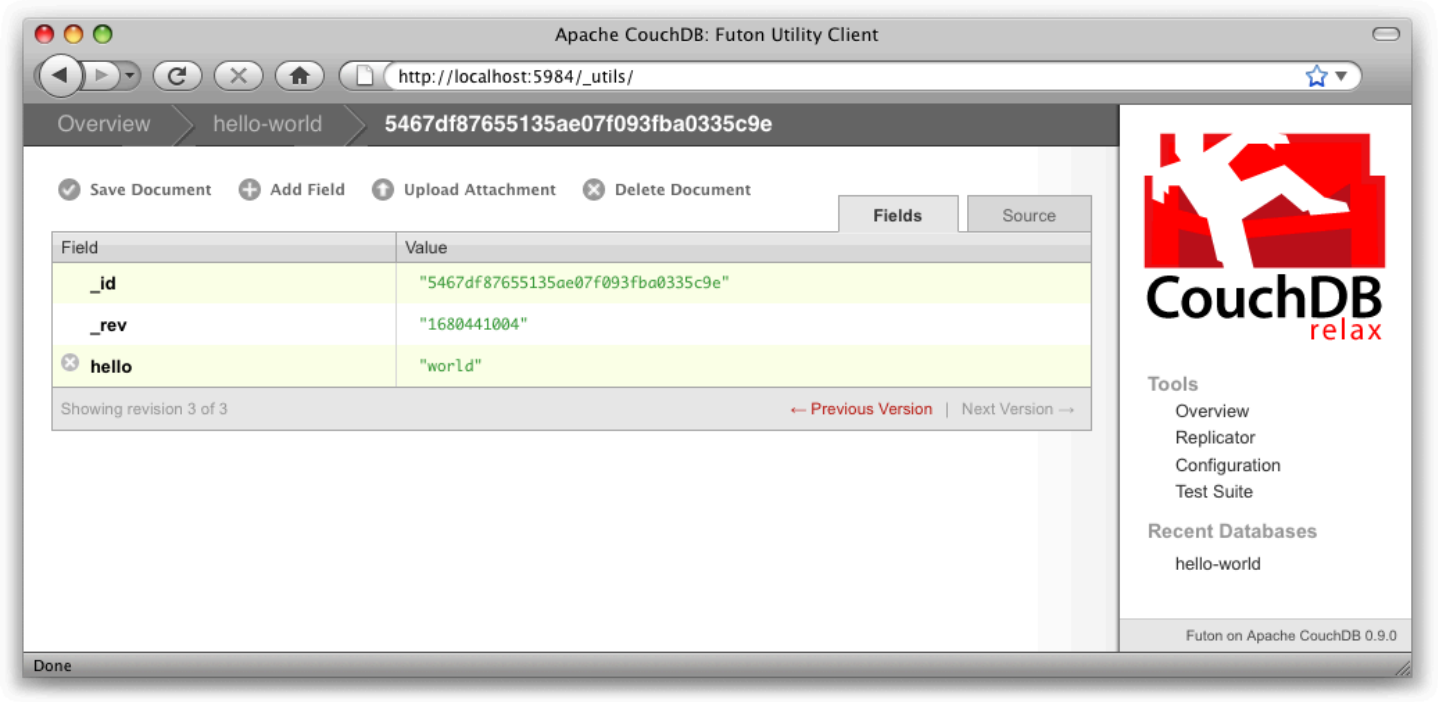

Figure 4. Un « hello world » avec Futon

Vous noterez que la révision du document a changé (`_rev`). Nous détaillerons cela dans un autre chapitre. Pour le moment, il est suffisant de se rappeler que `_rev` agit comme un garde-fou durant la sauvegarde. Tant que CouchDB et vous-même êtes d'accord sur le dernier `_rev` d'un document, vous pouvez sauvegarder vos changements.

Futon permet aussi d'afficher les données JSON directement, ce qui peut s'avérer être plus compact et plus facile à lire selon les données que vous traitez. Pour consulter la version JSON de notre « hello world », cliquez sur l'onglet « Source ». Vous devriez avoir quelque chose ressemblant à la **Figure 5, La forme JSON du document « hello world » avec Futon (#figure/5)**.

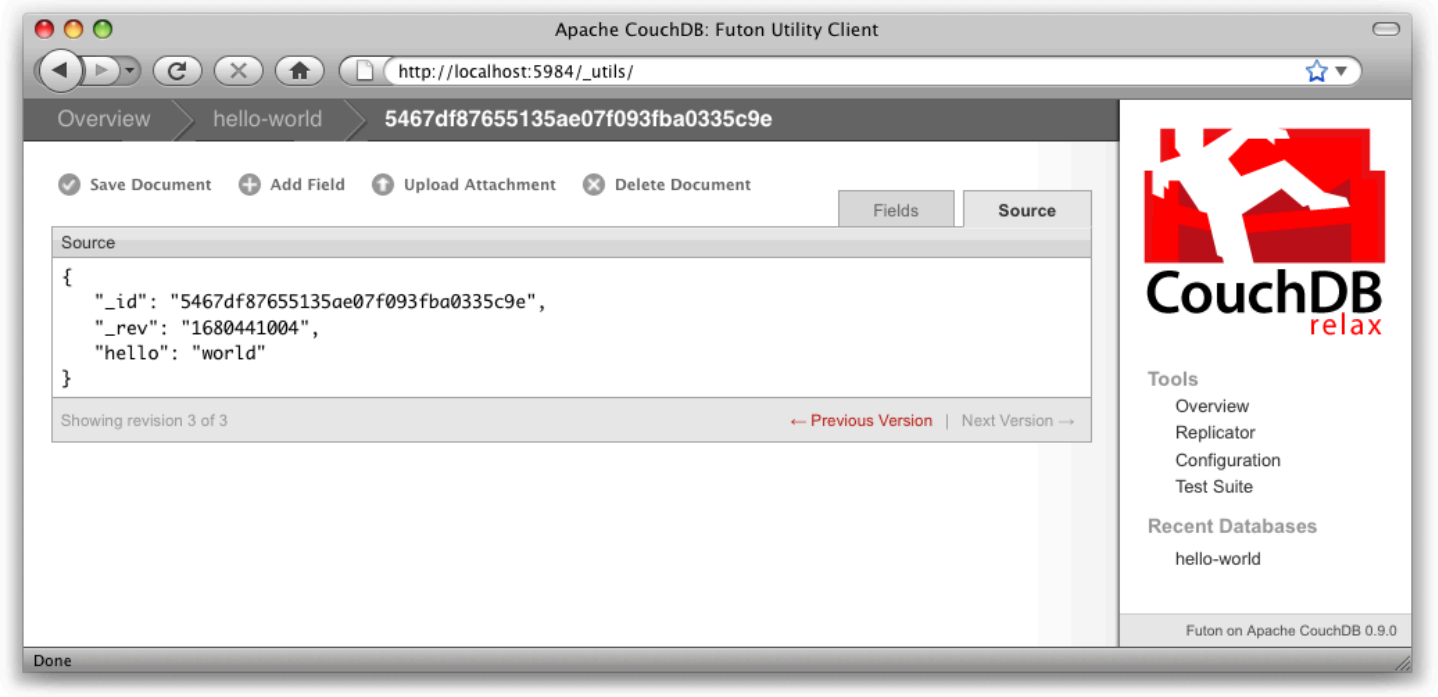

Figure 5. La forme JSON du document « hello world » avec Futon

Exécuter une requête avec MapReduce

Les traditionnelles bases de données relationnelles vous permettent d'exécuter n'importe quelle requête tant que vos données sont structurées convenablement. De son côté, CouchDB exploite des fonctions *map* (subdiviser) et *reduce* (agréger) dans un style connu sous le nom de MapReduce. La combinaison de ces fonctions offre une grande souplesse, car elles peuvent s'adapter aux variations de la structure d'un document. De plus, les index de chaque document peuvent être calculés de manière indépendante et en parallèle. Cette combinaison forme ce que CouchDB appelle une *vue*.

> Pour les développeurs accoutumés aux bases de données relationnelles, MapReduce est une approche qui peut nécessiter un temps d'adaptation. Plutôt que déclarer quels enregistrements de quelles tables doivent apparaître dans le résultat de la requête et de laisser au moteur de la base le choix du meilleur moyen de les obtenir, les requêtes d'agrégation (*reduce*) se basent sur des intervalles des clés générées par la fonction de subdivision (*map*).

La fonction de subdivision (*map*) est appelée une fois par document et reçoit celui-ci en argument. La fonction peut alors choisir d'ignorer le document ou d'*émettre* un ou plusieurs enregistrements sous la forme de couples clé/valeur. Les fonctions de subdivision ne doivent pas dépendre d'éléments externes au document. Cette indépendance est ce qui permet à CouchDB de générer les vues de manière incrémentale et en parallèle.

Dans CouchDB, les vues sont stockées comme des enregistrements qui sont triés par leur clé. Il est ainsi possible de retourner rapidement les enregistrements correspondants à un intervalle de clés, même avec des millions d'enregistrements stockés. Lorsque vous écrivez une fonction d'agrégation, votre but premier est de construire un index qui affecte une clé qui a du sens pour trouver la donnée qui se trouve derrière.

Avant de pouvoir tester une vue MapReduce, nous avons besoin de quelques données sur lesquelles opérer. Nous allons donc créer des documents stockant le prix d'articles de supermarché comme on en trouve dans divers magasins. Faisons-le pour les pommes, les oranges et les bananes (laissez CouchDB générer les champs `_id` et `_rev`). Utilisez Futon jusqu'à obtenir un résultat similaire à :

```
{
    "_id" : "bc2a41170621c326ec68382f846d5764",
    "_rev" : "2612672603",
    "item" : "apple",
    "prices" : {
        "Fresh Mart" : 1.59,
        "Price Max" : 5.99,
        "Apples Express" : 0.79
    }
}
```

Ce document devrait ressembler à la **Figure 6, Un document contenant le prix des pommes dans Futon (#figure/6)**.

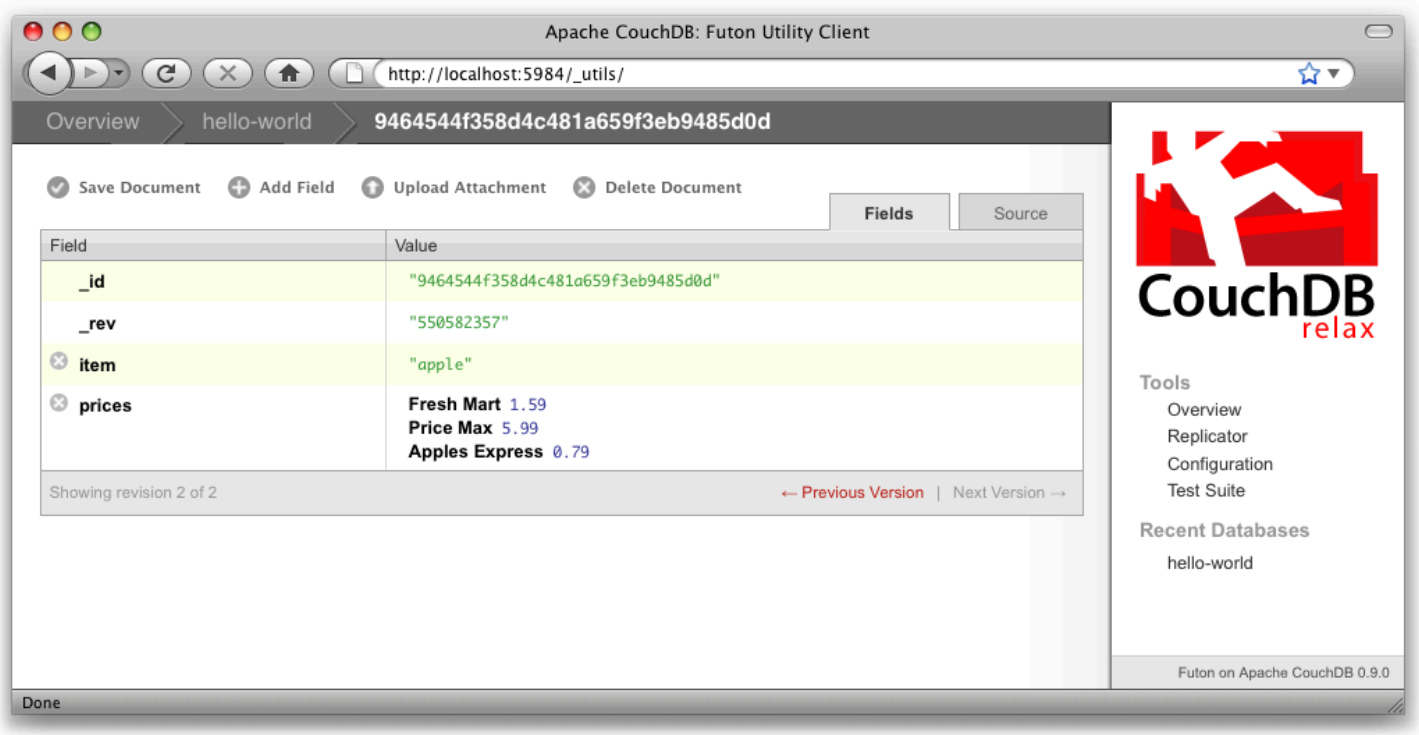

Figure 6. Un document contenant le prix des pommes dans Futon

Parfait ! Maintenant que c'est fait, créons les oranges :

```
{
    "_id" : "bc2a41170621c326ec68382f846d5764",
    "_rev" : "2612672603",
    "item" : "orange",
    "prices" : {
        "Fresh Mart" : 1.99,
        "Price Max" : 3.19,
        "Citrus Circus" : 1.09
    }
}
```

Et enfin les banaes :

```
{
    "_id" : "bc2a41170621c326ec68382f846d5764",
    "_rev" : "2612672603",
    "item" : "banana",
    "prices" : {
        "Fresh Mart" : 1.99,
        "Price Max" : 0.79,
        "Banana Montana" : 4.22
    }
}
```

Imaginez que nous préparions un banquet, mais que le client soit très sensible au prix. Pour trouver les prix les plus bas, nous allons créer notre première vue qui montrera chaque fruit trié par prix. Cliquez sur « hello-world » pour revenir à l'écran de synthèse, puis dans le menu « select view », choisissez « Temporary view ». Vous devriez obtenir quelque chose de similaire à la **Figure 7, Une vue temporaire dans Futon (#figure/7)**.

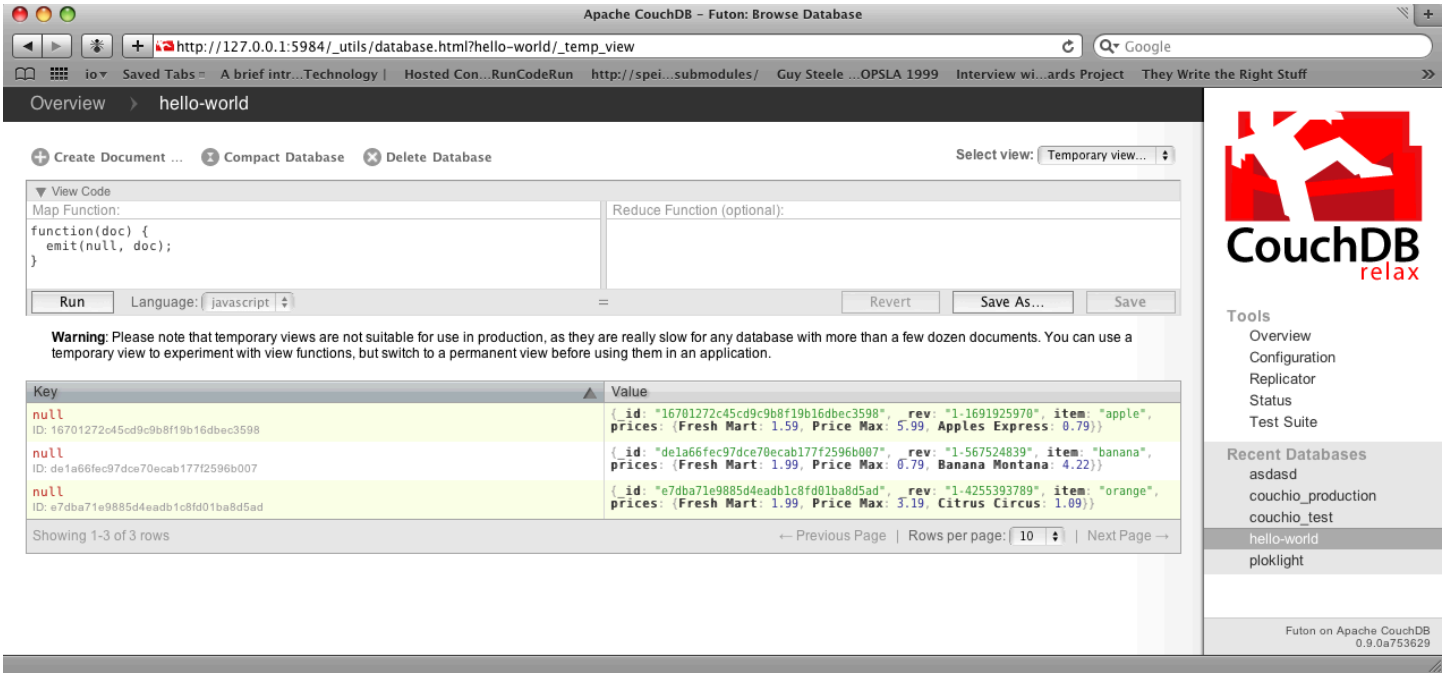

Figure 7. Une vue temporaire dans Futon

Éditez la fonction de subdivision (*map*) pour qu'elle contienne :

```
function(doc) {
    var store, price, value;
    if (doc.item && doc.prices) {
        for (store in doc.prices) {
            price = doc.prices[store];
            value = [doc.item, store];
            emit(price, value);
        }
    }
}
```

Il s'agit d'une fonction *JavaScript* que CouchDb exécuter sur chacun des documents et qui génère la vue. Nous laissons la fonction d'agrégat (*reduce*) vide pour le moment.

Cliquez sur « Run » et vous devriez voir les enregistrements comme sur la **Figure 8, Les résultats d'une vue avec Futon (#figure/8)**, c'est-à-dire avec les objets triés par prix. Cette fonction d'agrégation pourrait être plus utile si elle regroupait les objets par type pour que les prix des bananes soient à côté les uns des autres. L'algorithme de tri des clés de CouchDB permet d'avoir n'importe quel tableau JSON dans la clé. Dans notre cas, nous allons créer un tableau de la forme `[item, price]` (objet, prix) pour que CouchDB regroupe les résultats par type et prix.

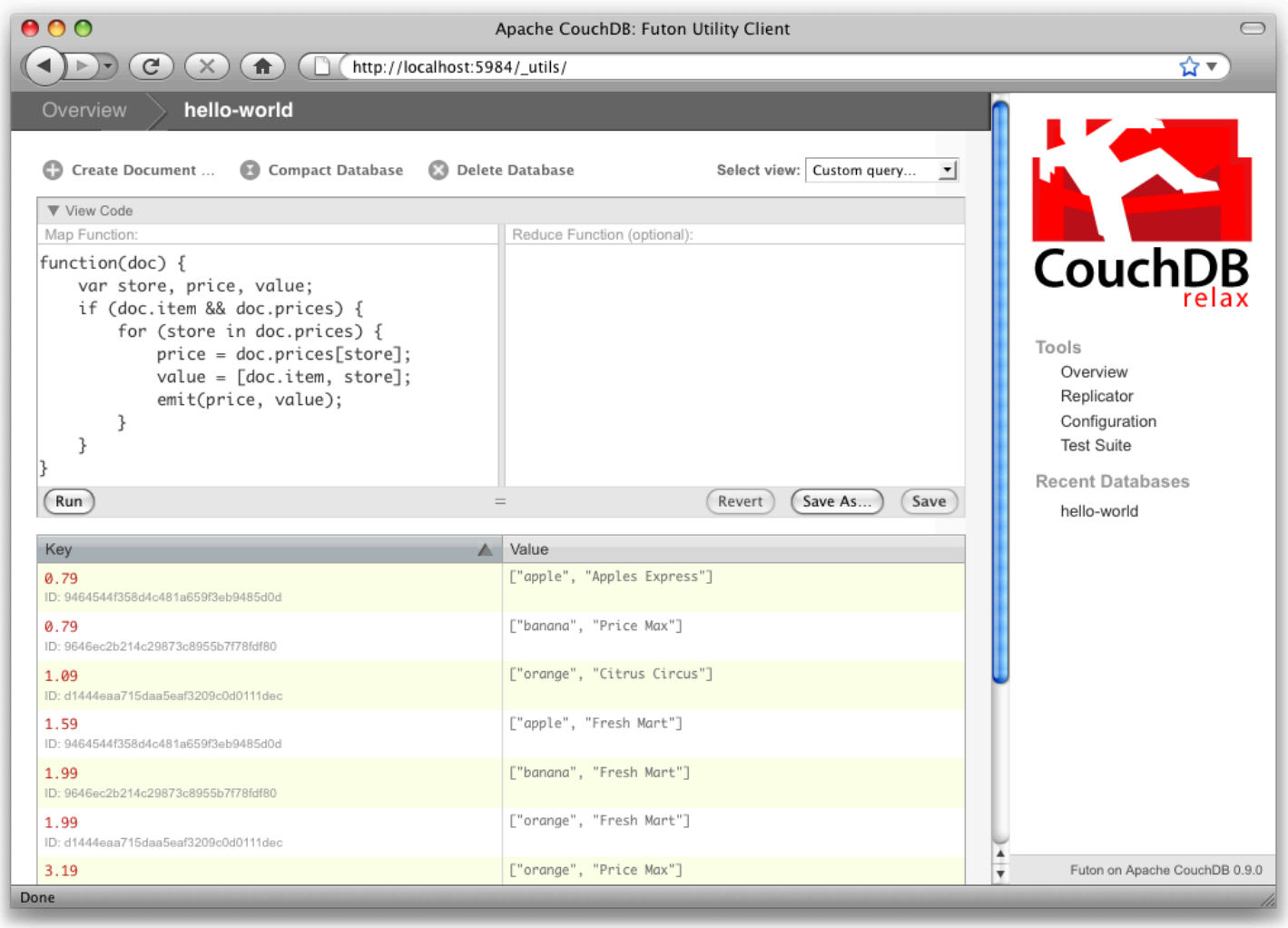

Figure 8. Les résultats d'une vue avec Futon

Modifions la vue :

```
function(doc) {
    var store, price, key;
    if (doc.item && doc.prices) {
        for (store in doc.prices) {
            price = doc.prices[store];
            key = [doc.item, price];
            emit(key, store);
        }
    }
}
```

Ici, nous vérifions tout d'abord que le document contient les champs que nous voulons utiliser. CouchDB ne s'inquiète pas de quelques erreurs dans l'exécution de la fonction d'agrégation, mais quand l'échec est récurrent (que ce soit à cause d'un champ manquant ou d'une exception JavaScript), CouchDB cesse d'indexer pour éviter de consommer des ressources. C'est pourquoi il est nécessaire de vérifier l'existence des champs avant de les utiliser. Dans notre cas, la fonction d'agréation ignorera le premier document que nous avons créé tout à l'heure, cela en n'émettant aucun enregistrement ni aucune erreur. Le résultat de la requête devrait être similaire à ce que présente la **Figure 9, Vue résultante après avoir regroupé par type et prix (#figure/9)**.

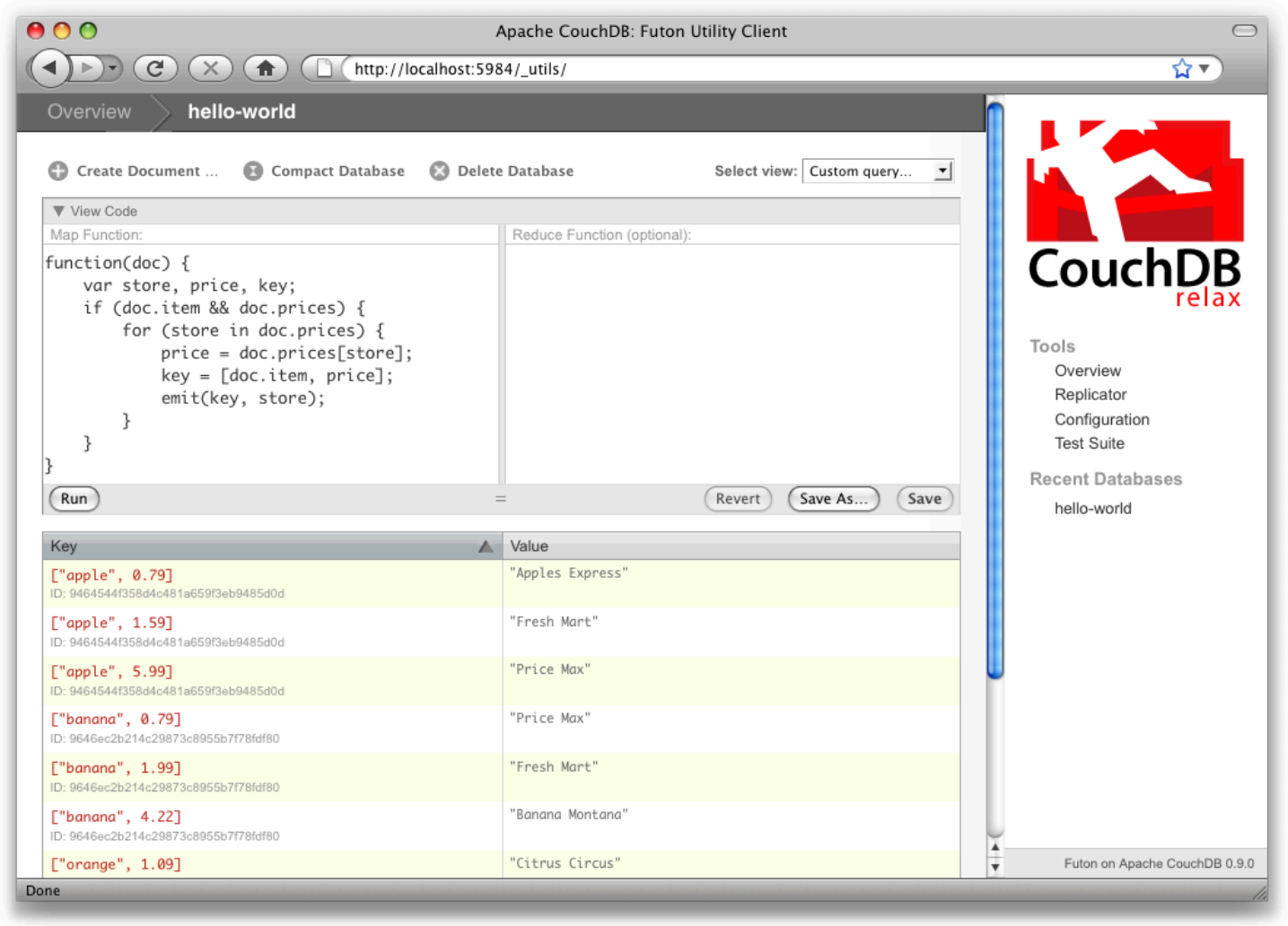

Figure 9. Vue résultante après avoir regroupé par type et prix

Une fois que nous avons confirmé que nous traitons un document qui contient un type d'objet et quelques prix, nous parcourons les prix pour émettre des couples clé/valeur. La clé est un tableau contenant l'objet et le prix, et définit l'index trié de CouchDB. La valeur sera ici le nom de l'enseigne où l'objet peut être trouvé à ce prix.

Les enregistrements d'une vue sont triés par leur clé – dans cet exemple, d'abord par objet, puis par prix. Cette méthode de tri complexe est essentielle pour créer des index utiles avec CouchDB.

> Utiliser MapReduce peut-être difficile, tout particulièrement si vous avez l'habitude des bases de données relationnelles. Ce qui importe, c'est de se rappeler que les fonctions de subdivision (*map*) vous permettent de trier les données en utilisant la clé qui vous convient, et que CouchDB s'attèle à fournir un accès rapide et efficace aux données dans un intervalle de clés.

Déclencher la réplication

Futon peut déclencher la réplication entre deux bases de données locales, entre une base locale et une distante, ou entre deux bases distantes. Nous allons voir comment répliquer les données d'une base locale vers une autre, ce qui est un moyen simple de faire une sauvegarde.

Tout d'abord, nous devons créer une base de données vide pour pouvoir y répliquer les données. Revenez à l'écran de synthèse et créez une base `hello-replication`. Maintenant, cliquez sur « Replicator » dans le menu latéral et sélectionnez `hello-world` comme source et `hello-replication` comme destinataire. Cliquez sur « Replicate » pour déclencher la réplication. Vous devriez obtenir quelque chose ressemblent à **Figure 10, Déclencher une réplication avec Futon (#figure/10)**.

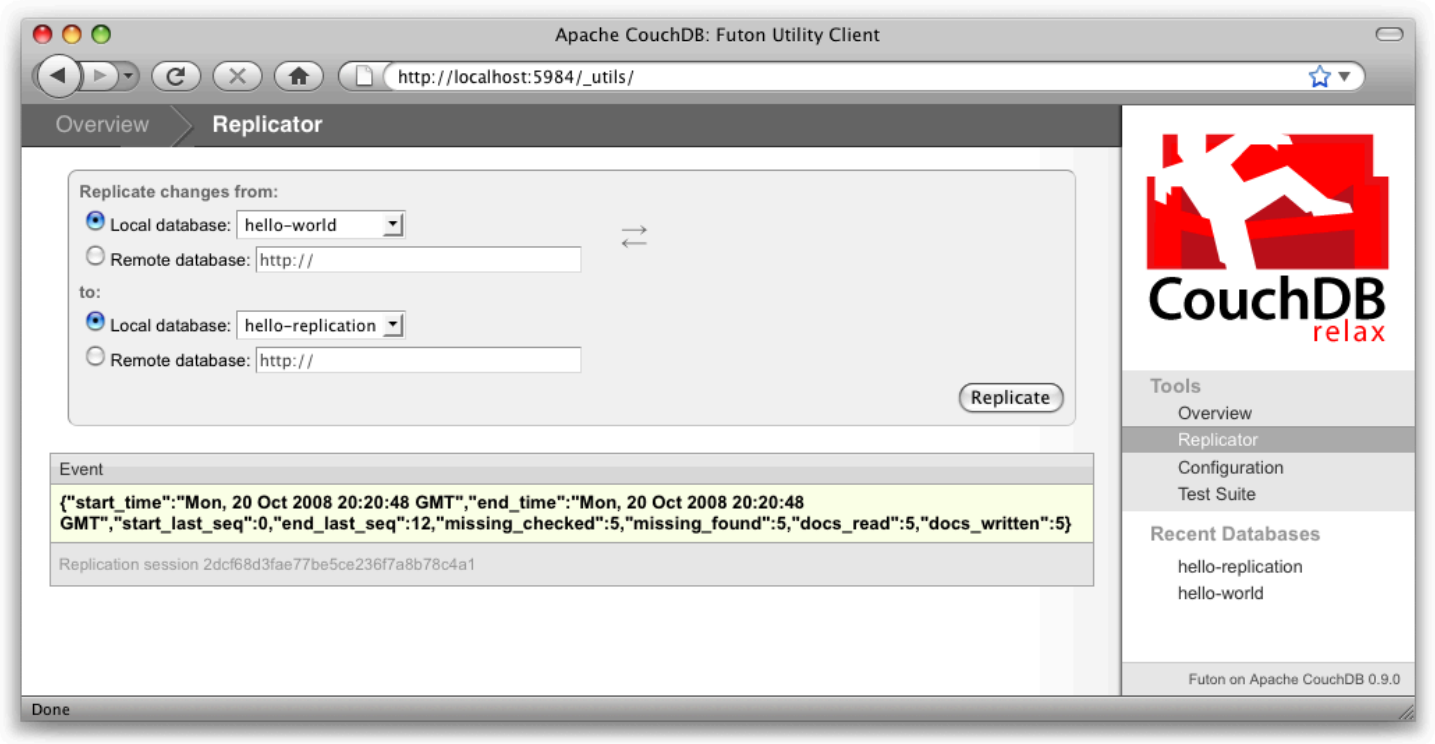

Figure 10. Déclencher une réplication avec Futon

Pour d'imposantes bases de données, la réplication peut prendre beaucoup plus de temps. Il est essentiel de garder la fenêtre du navigateur ouverte durant la réplication. Alternativement, vous pouvez la déclencher à l'aide de `curl` ou d'un autre client HTTP capable de gérer les connexions de longue durée. Si vous fermez la connexion avant la fin de la réplication, vous devrez la relancer. Heureusement, CouchDB reprendra là où il s'était arrêté plutôt que de reprendre au début.

Résumé

Maintenant que vous avez un aperçu des principales fonctionnalités de Futon, vous êtes parés à explorer vos données pendant qu'au fil des chapitres suivants nous concevons notre exemple d'application. L'approche tout-JavaScript de Futon pour la gestion de CouchDB montre qu'il est possible de bâtir une application web complète en utilisant uniquement l'API HTTP de CouchDB et son serveur web intégré.

Mais avant cela, nous allons détailler plus précisément l'API HTTP de CouchDB. *Curl*-ons-nous sur le sofa et détendons-nous.