Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

mit dieser Ausgabe feiert der IBM INFORMIX Newsletter sein 8-jähriges Bestehen. In den 96 regulären Ausgaben, die inzwischen mehr als eintausend Leser erreichen, haben wir über 600 technische Artikel erstellt, sowie viele Hinweise auf Versionsänderungen und Veranstaltungen verbreitet. Wir bedanken uns bei Ihnen, unseren Lesern, für das rege Interesse an unserem Newsletter und freuen uns auf viele weitere Jahre mit Ihnen und INFORMIX. Damit Sie auch sicher durch die nächsten Jahre kommen, haben wir uns im Hinblick auf die nun kommende, dunkle Jahreszeit diesmal dem Thema Sicherheit gewidmet.

Wie immer haben wir für Sie eine Reihe an Tipps und Tricks zusammengestellt.
Viel Spaß mit den Tipps der aktuellen Ausgabe.

Ihr TechTeam

TechTipp: Informix und NoSQL (3): NoSQL-Collections und Dokumente manipulieren

Vorbemerkung:
Alle Beispiele wurden mit Informix Version 12.10.xC4 (auf Linux x86 64-bit) erprobt.
Da der Bereich NoSQL ständig neue Funktionalitäten erhält, könnte es sein, dass die Beispiele in älteren Versionen nicht oder nur teilweise lauffähig sind.

Wir verwenden wieder die Datenbank "stores_demo" und die Mongo Shell (siehe auch die September-Ausgabe des Informix Newsletters). In "stores_demo" sind schon einige SQL-Tabellen enthalten, unter anderem eine Tabelle "catalog", mit Artikelbeschreibungen eines Sportartikelhändlers. Für unsere ersten NoSQL-Schritte nehmen wir an, dass mit einer neuen web-Applikation aus den Daten in "stores_demo" eine web-Seite mit einem Artikelkatalog erstellt wurde. Nun soll die Möglichkeit geboten werden, einzelne Artikel zu bewerten. Die web-Applikation soll die Bewertungen mittels NoSQL in der Datenbank "stores_demo" speichern und auch wieder auf der web-Seite anzeigen. Mit unseren ersten NoSQL-Befehlen erkunden wir datenbankseitig, wie die web-Applikation NoSQL nutzen könnte.

Mit NoSQL werden Daten nicht als Zeilen in einer Tabelle gespeichert, sondern als *Dokument* in einer *Collection* (siehe auch die August-Ausgabe des Informix Newsletters). Da NoSQL-Collections kein vordefiniertes Schema haben, ist es nicht nötig, vor dem Einfügen von Dokumenten eine Collection explizit anzulegen. Eine Collection wird impliziert angelegt beim Einfügen des ersten Dokuments. Hierzu definieren wir in der Mongo Shell zuerst das Dokument mit seinen *key-value*-Paaren im JSON-Format. Beim Einfügen des Dokuments genügt es dann, den Namen der neuen Collection anzugeben. Für den Artikel mit der Katalognummer 10017 wollen wir in einer neuen Collection namens "rating" eine Bewertung von 7 Punkten speichern. Wir verbinden uns mit der Mongo Shell zur Datenbank "stores_demo", definieren das zu speichernde Dokument und fügen es mit der Funktion insert() in die Collection ein:

```
$ mongo localhost:26351/stores_demo
MongoDB shell version: 2.4.9
connecting to: localhost:26351/stores_demo
> document=({"catalog_num":10017,"rating_value":7})
{ "catalog_num" : 10017, "rating_value" : 7 }
> db.rating.insert(document)
>
```

Nach dem Einfügen des Dokuments kontrollieren wir mit dem Befehl show collections, ob die Collection tatsächlich erstellt wurde und benutzen mit Angabe der Collection die Funktion find() um sicher zu sein, dass das Dokument gespeichert ist:

```
> show collections
rating
> db.rating.find()
{ "_id" : ObjectId("53bfec56d70e6e3abc72c40a"), "catalog_num" :
10017, "rating_value" : 7 }
>
```

Wir sehen nicht nur, dass alles wie erwartet funktioniert hat, sondern auch, dass das Dokument ein zusätzliches Feld erhalten hat. Dies ist eine Objekt-ID, die immer automatisch erzeugt wird und jedes Dokument eindeutig identifiziert.

Die Funktion find() ist das Gegenstück zum SQL-Befehl SELECT. Der SQL-Befehl SELECT * from *table*; entspricht somit dem einfachen Aufruf der Funktion find() nach dem Muster db.*collection*.find(). Ungewohnt ist hierbei, dass die automatisch erzeugte Objekt-ID immer mit ausgegeben wird. Wenn uns diese Objekt-ID zu sehr stört, dann müssen wir beim Aufruf der Funktion find() das Feld der Objekt-ID explizit ausschliessen. Dazu geben wir der Funktion find() als zweiten Parameter eine Projektionsliste, in der wir nur {_id:0} angeben. Die 0 steht hier für 'false' und bedeutet, dass wir das Feld '_id' nicht erhalten wollen. Da die Projektionsliste für die Funktion find() der zweite Parameter ist, müssen wir auch den ersten Parameter angeben. Für den Moment genügt es, hier als 'leeren Parameter' einfach {} zu verwenden:

```
> db.rating.find({},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 7 }
>
```

Diese Methode zum Spezifizieren der Projektionsliste funktioniert nicht nur für das Feld der Objekt-ID, sondern auch für alle anderen Felder. Mit 0 'schalten' wir jeweils einzelne Felder aus:

```
> db.rating.find({},{_id:0,rating_value:0})
{ "catalog_num" : 10017 }
>
```

Bislang haben wir nur ein Dokument in unserer Collection "rating" gespeichert. Um die Funktionsweise von find() weiter zu erkunden, fügen wir der Collection nun weitere Dokumente hinzu. Anstatt zuerst ein Dokument mit seinen key-value-Paaren zu defineren, um dieses dann mit der Funktion insert() einzufügen, ist es auch möglich, das Dokument direkt als Parameter für insert() zu definieren. (Der '>'-Prompt der Mongo Shell fehlt bei den folgenden Zeilen, damit Cut&Paste in einem Zug einfacher ist für diejenigen, die folgende Beispiele selbst nachvollziehen wollen.)

```
db.rating.insert({"catalog_num": 10017, "rating_value": 9})
db.rating.insert({"catalog_num": 10018, "rating_value": 7})
db.rating.insert({"catalog_num": 10017, "rating_value": 5})
db.rating.insert({"catalog_num": 10017, "rating_value": 8})
db.rating.insert({"catalog_num": 10018, "rating_value": 3})
db.rating.insert({"catalog_num": 10017, "rating_value": 7})
db.rating.insert({"catalog_num": 10017, "rating_value": 8})
db.rating.insert({"catalog_num": 10017, "rating_value": 4})
db.rating.insert({"catalog_num": 10018, "rating_value": 8})
db.rating.insert({"catalog_num": 10018, "rating_value": 3})
db.rating.insert({"catalog_num": 10017, "rating_value": 8})
```

Rufen wir nun wieder die Funktion find() auf (mit Ausschluss des Objekt-ID-Feldes), so bekommen wir als Ergebnis alle Dokumente, die mittlerweile in der Collection enthalten sind:

```
> db.rating.find({},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 9 }
{ "catalog_num" : 10018, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 5 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 4 }
{ "catalog_num" : 10018, "rating_value" : 8 }
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10017, "rating_value" : 8 }
>
```

Einfach alle Dokumente einer Collection anzuzeigen, ist nicht immer nützlich. Oft sind nur bestimmte Dokumente von Interesse, die mit einem Filter ausgewählt werden. Um z.B. nur die Bewertungen für den Artikel mit der Katalognummer 10017 zu sehen, spezifizieren wir als ersten Parameter der Funktion find() den Filter {catalog_num:10017}, wie im folgenden Beispiel:

```
> db.rating.find({catalog_num:10017},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 9 }
{ "catalog_num" : 10017, "rating_value" : 5 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 4 }
{ "catalog_num" : 10017, "rating_value" : 8 }
>
```

Der Filter wirkt also wie ein lokales Prädikat in der WHERE-Klausel einer SQL-SELECT Abfrage. Und genauso könen wir auch mehrere Filterbedingungen logisch miteinander verknüpfen, nur die Syntax ist etwas anders. Mehrere Bedingungen verknüpfen wir mit einem logischen UND einfach, indem wir sie mit Komma getrennt auflisten:

```
 > db.rating.find({catalog_num:10017,rating_value:8},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
>
```

Etwas komplizierter ist die Kombination von mehreren Filterbedingungen mittels einem logischen ODER. Im folgenden Beispiel behalten wir den Filter für Artikel mit der Katalognummer 10017 bei, möchten jedoch als Ergebnis sowohl Bewertungen mit 8 als auch mit 9 erhalten. Dazu benutzen wir $or:[], wobei wir die einzelnen Bedingungen innerhalb [] jeweils mit {} umschlossen und durch Komma getrennt auflisten:

```
 > db.rating.find({catalog_num:10017,$or:[{rating_value:8},
{rating_value:9}]},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 9 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
>
```

Wollen wir die Liste aller Bewertungen nach Punktzahl sortiert haben, so verwenden wir zusätzlich die Funktion sort() und umschliessen mit {} den Parameter mit der gewünschten Sortierbedingung:

```
 > db.rating.find({},{_id:0}).sort({rating_value:1})
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10017, "rating_value" : 4 }
{ "catalog_num" : 10017, "rating_value" : 5 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10018, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10018, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 8 }
{ "catalog_num" : 10017, "rating_value" : 9 }
>
```

Hierbei bedeutet die Ziffer 1 in der Sortierbedingung {rating_value:1}, dass nach dem Feld "rating_value" aufsteigend sortiert werden soll. Für eine absteigende Sortierung würden wir -1 angeben. Sind wir mehr daran interessiert, die Bewertungen jedes einzelnen Artikels für sich sortiert zu haben, so geben wir in der Sortierbedingung zuerst das Feld der Katalognummer und dann das Feld der Bewertung an, jeweils aufsteigend:

```
  > db.rating.find({},
{_id:0}).sort({catalog_num:1,rating_value:1})
  { "catalog_num" : 10017, "rating_value" : 4 }
  { "catalog_num" : 10017, "rating_value" : 5 }
  { "catalog_num" : 10017, "rating_value" : 7 }
  { "catalog_num" : 10017, "rating_value" : 7 }
  { "catalog_num" : 10017, "rating_value" : 8 }
  { "catalog_num" : 10017, "rating_value" : 8 }
  { "catalog_num" : 10017, "rating_value" : 8 }
  { "catalog_num" : 10017, "rating_value" : 9 }
  { "catalog_num" : 10018, "rating_value" : 3 }
  { "catalog_num" : 10018, "rating_value" : 3 }
  { "catalog_num" : 10018, "rating_value" : 7 }
  { "catalog_num" : 10018, "rating_value" : 8 }
  >
```

Um festzustellen, wieviele Dokumente unsere Collection enthält, benuzten wir statt der Funktion find() die Funktion count(), im einfachsten Fall ohne Parameter, was uns die Anzahl aller Dokumente in der Collection liefert:

```
  > db.rating.count()
  12
  >
```

Wie bei der Funktion find() können wir auch bei count() eine Filterbedingung angeben, z.B. um festzustellen, wieviele Dokumente es gibt für den Artikel mit der Katalognummer 10018, oder wieviele Bewertungen mit 7 Punkten es gibt:

```
  > db.rating.count({catalog_num:10018})
  4
  > db.rating.count({rating_value:7})
  3
  >
```

Eine weitere Funktion ist distinct(), der man den Namen des gewünschten Feldes übergibt. Die dafür gefundenen unterschiedlichen Werte werden dann als Array, mit [ ] umschlossen, zurückgegeben:

```
> db.rating.distinct("rating_value")
[ 3, 4, 5, 7, 8, 9 ]
> db.rating.distinct("catalog_num")
[ 10017, 10018 ]
>
```

Nachdem wir einige Basisfunktionen zum Selektieren von Daten in einer Collection kennengelernt haben, schauen wir uns nun die Funktion update() an, um zu verstehen, wie man existierende Dokumente verändern kann. In der einfachsten Form nimmt update zwei Parameter entgegen, einen Filter, um das zu verändernde Dokument zu bestimmen, und die Angabe, was update() mit dem Dokument dann tun soll. Auch wenn dies bezüglich der besagten web-Applikation nicht sinnvoll ist, versuchen wir nun der Übung halber, die Bewertungen mit der Punktzahl 8 um jeweils 2 Punkte auf 10 zu erhöhen, wofür wir den operator $set{} verwenden. Anschliessend kontrollieren wir sofort, ob die Operation erfolgreich war:

```
> db.rating.update({rating_value:8},{$set:{rating_value:10}})
> db.$cmd.findOne({getlasterror:1})
{
        "serverUsed" : "localhost/127.0.0.1:24731",
        "updatedExisting" : true,
        "n" : 1,
        "err" : null,
        "ok" : 1,
        "connectionId" : 8,
        "wtime" : 0,
        "waited" : 0
}
>
```

Das Ergebnis der 'Kontrollfunktion' db.$cmd.findOne({getlasterror:1}) mit den Werten "ok":1 und "err":null teilt uns mit, dass der Update erfolgreich war. Schauen wir uns also die Dokumente an:

```
> db.rating.find({},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 9 }
{ "catalog_num" : 10018, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 5 }
{ "catalog_num" : 10017, "rating_value" : 10 }
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 8 }
```

```
  { "catalog_num" : 10017, "rating_value" : 4 }
  { "catalog_num" : 10018, "rating_value" : 8 }
  { "catalog_num" : 10018, "rating_value" : 3 }
  { "catalog_num" : 10017, "rating_value" : 8 }
  >
```

Das Ergebnis entspricht vielleicht nicht ganz unseren Erwartungen, ist jedoch völlig korrekt. Die Funktion update() verändert normalerweise nur das erste vom Filter ausgewählte Dokument. Im Gegensatz zum SQL-Befehl UPDATE werden weitere Dokumente, die auch den Filterkriterien entsprechen, im Normalfall nicht verändert. Wollen wir dies jedoch erreichen, so müssen wir es der Funktion update() mit einem weiteren Parameter {multi:true} explizit mitteilen. Wenden wir das an, dann erhalten wir folgendes Ergebnis:

```
  > db.rating.update({rating_value:8},{$set:{rating_value:10}},
{multi:true})
  > db.$cmd.findOne({getlasterror:1})
  {
          "serverUsed" : "localhost/127.0.0.1:24731",
          "updatedExisting" : true,
          "n" : 3,
          "err" : null,
          "ok" : 1,
          "connectionId" : 8,
          "wtime" : 0,
          "waited" : 0
  }
  > db.rating.find({},{_id:0})
  { "catalog_num" : 10017, "rating_value" : 7 }
  { "catalog_num" : 10017, "rating_value" : 9 }
  { "catalog_num" : 10018, "rating_value" : 7 }
  { "catalog_num" : 10017, "rating_value" : 5 }
  { "catalog_num" : 10017, "rating_value" : 10 }
  { "catalog_num" : 10018, "rating_value" : 3 }
  { "catalog_num" : 10017, "rating_value" : 7 }
  { "catalog_num" : 10017, "rating_value" : 10 }
  { "catalog_num" : 10017, "rating_value" : 4 }
  { "catalog_num" : 10018, "rating_value" : 10 }
  { "catalog_num" : 10018, "rating_value" : 3 }
  { "catalog_num" : 10017, "rating_value" : 10 }
  >
```

Mit Angabe des Parameters {multi:true} verhält sich die Funktion update() eher so, wie wir es vom SQL-Befehl UPDATE gewohnt sind. Bei genauer Betrachtung der Ausgabe zu 'getlasterror' sehen wir auch, dass wir ohne {multi:true} für "n" den Wert 1 bekommen haben. Mit {multi:true} hingegen berichtet die Fehlerkontrolle für "n" den Wert 3, denn dies ist die Anzahl der jeweils veränderten Dokumente.

Neben dem Operator $set gibt es für die Funktion update() noch weitere Operatoren wie z.B. $unset und $inc, und zudem weitere Funktionen zum Verändern von Dokumenten, auf die wir zu einem späteren Zeitpunkt genauer eingehen werden.

Schliesslich wollen wir die nun verfälschten Bewertungen besser ganz entfernen, indem wir die Funktion remove() anwenden, um ganze Dokumente aus der Collection zu löschen. Im Gegensatz zur Funktion update() wirkt die Funktion remove() auf alle Dokumente, die durch den angegebenen Filter ausgewählt sind. Versuchen wir zuerst, nur die 10-Punkte-Bewertungen des Artikels mit der Katalognummer 10017 zu löschen:

```
> db.rating.remove({catalog_num:10017,rating_value:10})
> db.rating.find({},{_id:0})
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 9 }
{ "catalog_num" : 10018, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 5 }
{ "catalog_num" : 10018, "rating_value" : 3 }
{ "catalog_num" : 10017, "rating_value" : 7 }
{ "catalog_num" : 10017, "rating_value" : 4 }
{ "catalog_num" : 10018, "rating_value" : 10 }
{ "catalog_num" : 10018, "rating_value" : 3 }
>
```

Ein zusätzlicher Parameter 'justOne', mit dem man bei der MongoDB das Löschen von Dokumenten mittels der Funktion remove() auf das erste Dokument beschränken kann, ist mit der NoSQL-Funktionalität von Informix nicht implementiert (und auch so dokumentiert).

Der Aufruf der Funktion remove() ohne jeglichen Parameter entfernt alle Dokumente aus der Collection:

```
> db.rating.remove()
> db.rating.find()
>
```

Um die nun leere Collection zu entfernen, verwenden wir die Funktion drop(). Wenn wir versuchen, diese Funktion sofort im Anschluss an die obigen Operationen aufzurufen, dann erhalten wir allerdings folgenden Fehler:

```
>  db.rating.drop()
Mon Oct 06 17:13:48.473 drop failed: {
        "code" : null,
        "ok" : 0,
        "errmsg" : "Unable to drop namespace stores_demo.rating |
                    Unable to drop namespace stores_demo.rating |
                    Could not open database table
(informix.rating).
                    (SQL Code: -242, SQL State: IX000) |
                    ISAM error: non-exclusive access.
                    (SQL Code: -106, SQL State: IX000)"
} at src/mongo/shell/collection.js:383
>
```

Wir verlassen also die Mongo Shell (durch Eingabe von CTRL-D am >-Prompt), rufen die Mongo-Shell erneut auf, und können dann die Collection erfolgreich entfernen:

```
> CTRL-D
bye
$ mongo localhost:26351/stores_demo
MongoDB shell version: 2.4.9
connecting to: localhost:26351/stores_demo
> db.rating.drop()
true
> show collections
>
```

In der nächsten Ausgabe des Informix Newsletters werden wir sehen, wie wir mit SQL-Befehlen auf eine Collection und die Daten in ihren Dokumenten zugreifen können.

TechTipp: Berechtigungen auf Tabellenebene

In den letzten Wochen erreichen uns immer mehr Anfragen zum Thema „Berechtigungsverwaltung auf Tabellen“. Aus diesem Grund haben wir hier die wichtigsten Tipps zum Thema nochmals aufbereitet und kleine Scripts erstellt, die Ihnen helfen können, die Benutzerberechtigungen sicher und effizient zu verwalten.

Wer sich bisher nicht mit dem Thema „Berechtigungen“ beschäftigt hat, der wird im erstem Moment erstaunt sein, dass als Defaulteinstellung alle Anwender über die Gruppe „public“ auf allen Tabellen die Rechte “Select“, “Insert“, “Update“, “Delete“ und „Index“ erhalten, ohne dass diese explizit vergeben werden. Versuche, diese Rechte einzelnen Benutzern mittels „revoke all on <tabelle> from <user>“ zu entziehen, werden zwar von der Datenbank mit „permission revoked“ bestätigt, zeigen aber keine Wirkung, da der Benutzer ohnehin keine Rechte besass, sondern diese über die Rolle „public“ erhalten hatte.
Will man bereits beim Anlegen einer Tabelle oder beim Import einer Datenbank verhindern, dass die Gruppe „public“ Rechte erhält, so kann vor dem Start der Datenbankverbindung die Umgebungsvariable „**NODEFDAC=yes**“ gesetzt werden.

Sind die Rechte bereits vergeben, so können diese mittels „revoke“ entzogen werden. Um die Rechteverwaltung einfach und flexibel zu gestalten, empfehlen wir eine eigene Verwaltungstabelle anzulegen, in der alle berechtigten Benutzer aufgeführt sind, ggf. mit Berechtigungsgruppe und weiteren Informationen (wie z.B. Abteilung). Sind die Benutzer in solch einer Tabelle gepflegt, kann im ersten Schritt die Berechtigung für alle „unbekannten“ Benutzer entzogen werden, die nicht in dieser Tabelle aufgeführt sind.

Beispiel:

```
unload to "exec_revoke.sql"
delimiter "     "
select "revoke all on "||trim(t.tabname)||" from "||
trim(a.grantee)||";"
from systabauth a, systables t
where a.tabid = t.tabid
and t.tabid > 99
and t.tabtype = "T"
and a.grantee not in ( select b.login from userlist b)
order by 1
```

Hinweis: Der „delimiter“ ist ein <tab> und wird beim Kopieren meist als eine Reihe von Leerzeichen kopiert.
Da Sie sicher den Benutzernamen „public“ nicht in dieser Tabelle aufgenommen haben, werden damit die Rechte für „public“ gleich mit entzogen, wenn das erstellte SQL-Script „exec_revoke.sql“ dann auf der Datenbank ausgeführt wird.

Nachdem nun die ungewollten Rechte entzogen wurden, ist es Zeit, die Rechte wieder zu vergeben. Bei vielen Benutzern mit identischen Rechten auf den Tabellen wäre der beste Weg, Rollen für die unterschiedlichen Berechtigungen zu erstellen, und den Benutzern mittels „grant default role <rolename> to <user>“ diese Rollen als Standard zu hinterlegen. Diese Technik wurde in einer früheren Ausgabe des Informix Newsletters bereits vorgestellt. Im aktuellen Artikel beschränken wir uns auf die Vergabe der Rechte direkt an die Benutzer. Ein SQL-Script, das die Vergabe der Rechte vorbereitet, könnte folgendermassen aussehen:

```
unload to "exec_grant.sql"
delimiter "     "
select "grant "||
    case
       when b.permission = "ALL" then "all"
       when b.permission = "WRT" then "select,insert,update"
       when b.permission = "REA" then "select"
    end
||" on "||trim(t.tabname)||" to "||trim(b.login)||" as "||
trim(t.owner)||";"
from systables t, userlist b
where t.tabid > 99
and t.tabtype = "T"
and trim(t.tabname)||'_'||trim(b.login) not in (
        select trim(t.tabname)||'_'||trim(a.grantee)
        from systabauth a
        where a.tabid = t.tabid

)
order by 1;
```

Die Ausgabe exec_grant.sql enthält Zeilen der Form:

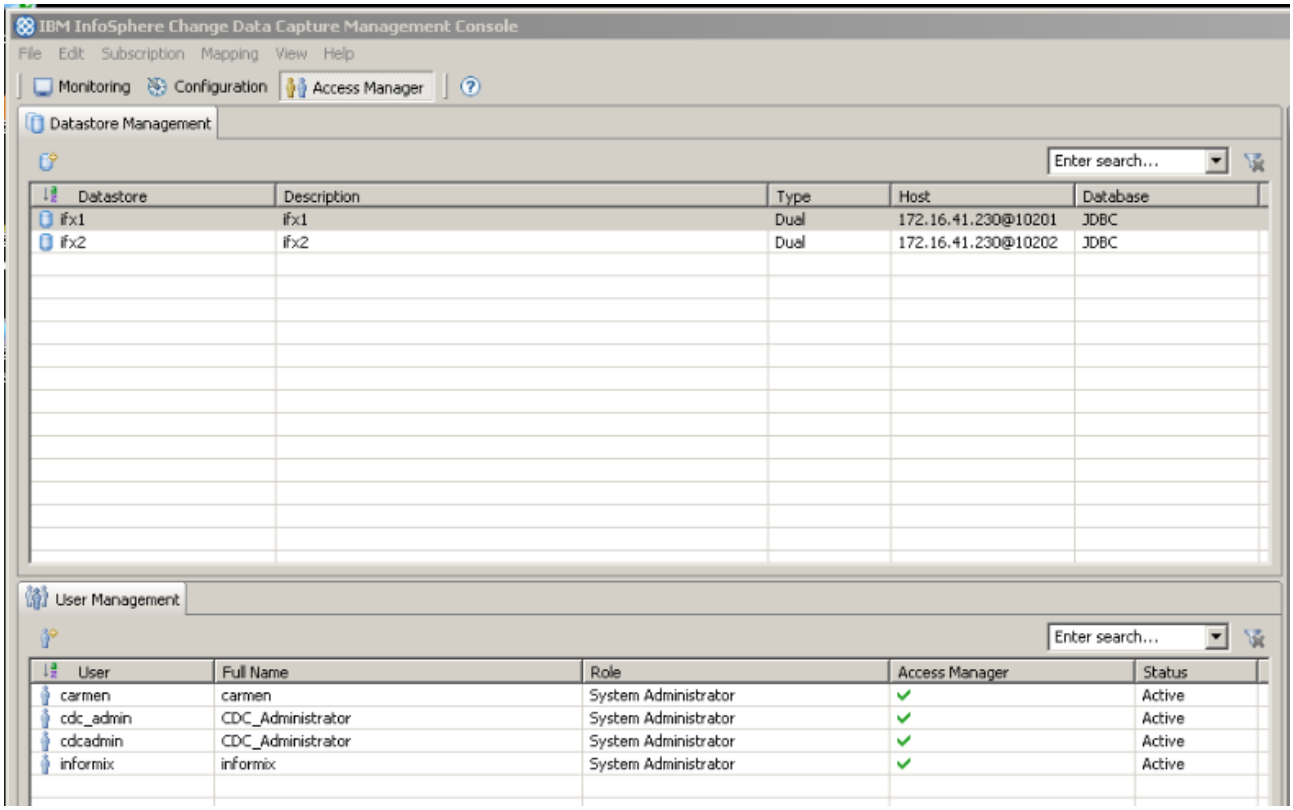
```
grant all on firma to kalu as informix;
grant all on person to kalu as informix;
...
grant select on firma to carina as informix;
grant select on person to carina as informix;
…
grant select,insert,update on firma to carmen as informix;
grant select,insert,update on person to carmen as informix;
```

Für die Praxis wird es sicher notwendig sein, das SQL so zu verändern, dass zu den Benutzergruppen noch eine Auswahl an Tabellen für die Vergabe der Berechtigung angegeben wird. Üblicherweise erfolgt auch diese Zuordnung ebenfalls über eine Verwaltungstabelle.

TechTipp: Auditing mit Hilfe von Infosphere Change Data Capture (CDC)

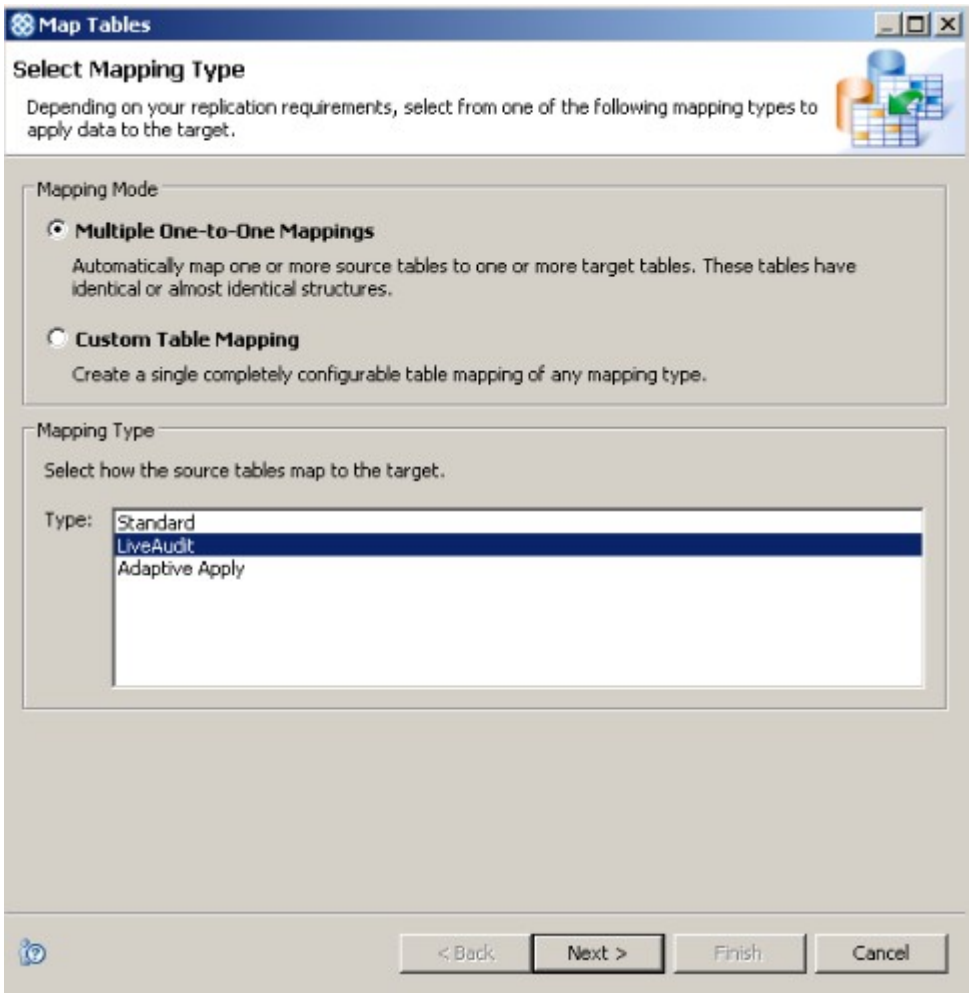
Informix beinhaltet eine Option für Auditing, die mittels „onaudit“ eingerichtet und über den Befehl „onshowaudit“ ausgelesen werden kann. Dieses Auditing kann die Auditprotokolle in einer Art Unloadformat erstellen, das für Auswertungen genutzt werden kann.

Eine weit komfortablere Variante, ein Auditing einzurichten lässt sich mittels „IBM InfoSphere Data Replication IIDR“, besser bekannt unter „IBM Infosphere CDC - Change Data Capture“ implementieren. Hierbei wird die Replikation genutzt, um Änderungen mit Zeitstempel und ID des Benutzers direkt in Audit-Tabellen auf einem entfernten Datenbankserver zu protokollieren. In der Zieldatenbank werden hierbei die Daten in Tabellen mit dem selben Aufbau wie auf der Quelle erstellt, wobei zusätzliche Spalten für einen Zeitstempel, den Verursacher der Änderung, und den Typ der Änderung als Spalten an die Tabelle angefügt werden. Lesende Aktion können hierbei nicht dokumentiert werden, da CDC auf die Informationen in den Transaktionslogs zugreift, in die rein lesende Vorgänge nicht eingetragen werden. Die Einrichtung erfolgt, sobald die Datenbankkomponenten und der Access-Server laufen, über die Management Console. Zuerst werden die „Datastores“ (Datenbankverbindungen) erstellt und mit den Verbindungsdaten, dem Benutzer und dem für die Verbindung notwendigen Passwort versehen.

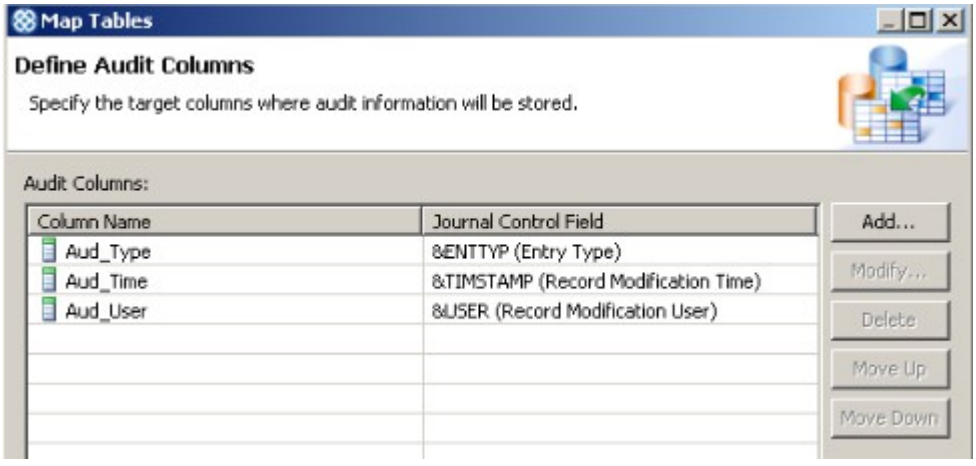


Anschliessend wird eine Subscription erstellt, die dann das eigentliche Replikationset darstellt. In dieses können beliebig viele Tabellen unterschiedlicher Eigentümer eingestellt sein.

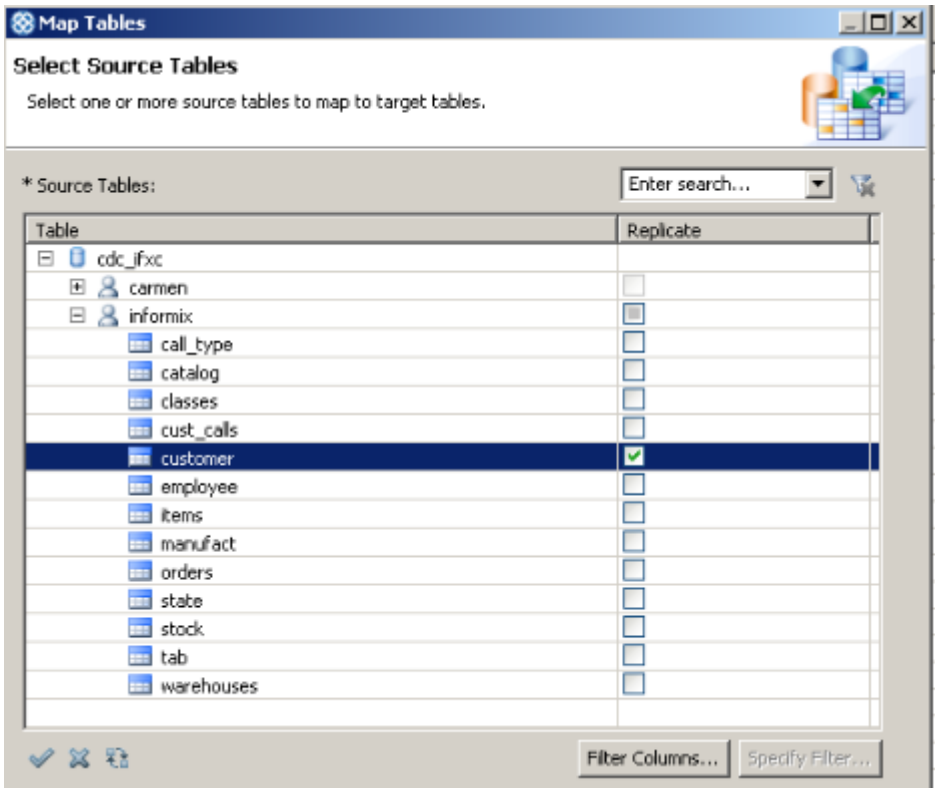
Danach wird das Tablemapping eingerichtet. Für das Audit reicht uns die einfachste Variante, in der jeder Source-Tabelle eine Audit-Tabelle zugeordnet ist. Wichtig ist der Anwahlpunkt „LiveAudit“. „Standard“ steht für die übliche Replikation der Daten.



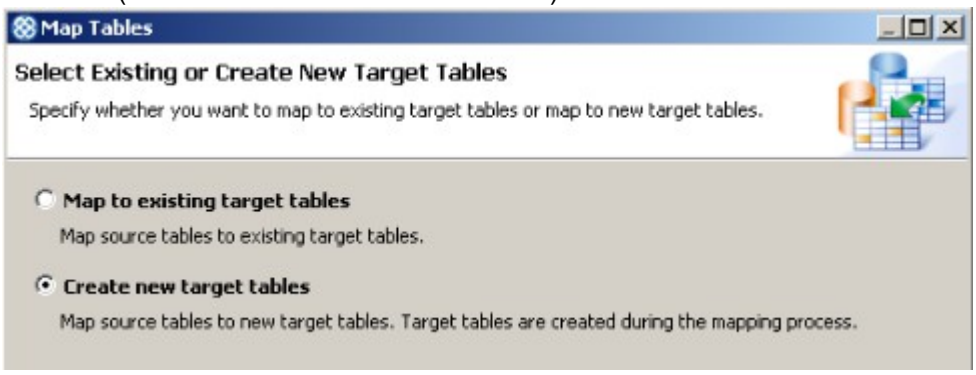
InfoSphere CDC fügt die Auditspalten an (die verändert oder abgewählt werden können):



Nun werden die im Audit zu replizierenden Tabellen ausgewählt (mit dem Haken links unten lassen sich alle Tabellen bzw. alle Tabellen eines Eigentümers gleichzeitig auswählen):

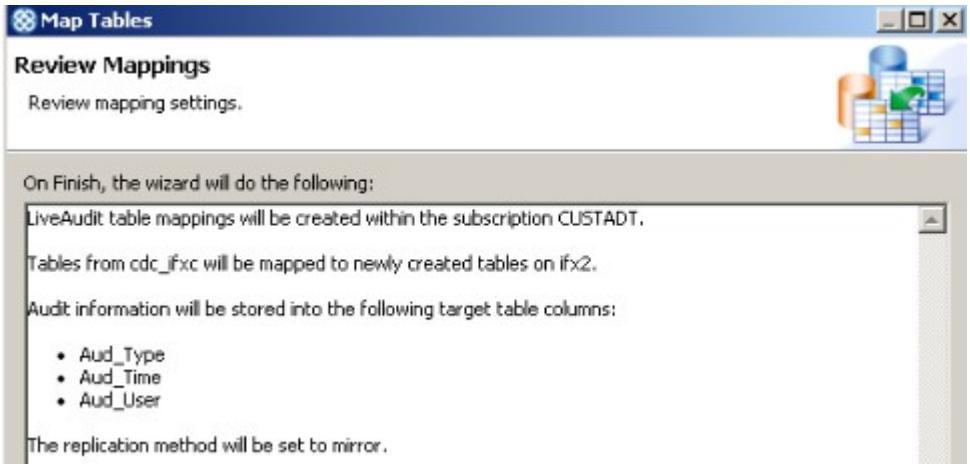


Es bestehen die Optionen in eine bestehende Tabelle zu schreiben, oder eine neue Tabelle (nach dem Schema der Quelle) durch CDC auf dem Ziel aufzubauen:



Die neuen Tabellen können mit einem Prefix oder einem Postfix am Tabellennamen markiert werden (z.B. _adt).

Sind diese einfachen Schritte abgeschlossen, so werden in einer Übersicht die Einstellungen angezeigt:



In der Übersicht der Subscriptions bleibt nun nur der letzte Schritt, die Subscrition zu starten. Dabei werden (wenn dies nicht abgewählt wurde) zu Beginn alle Daten in das Auditsysstem übertragen (refresh). Danach werden nur noch die Änderungen auf die Zieltabellen übermittelt.

Beispiel:

```
update customer set lname = "Kaluzinski" where customer_num = 104;
```

fügt ein BEFORE-IMAGE:

```
customer_num   104
fname          Anthony
lname          Higgins
…
phone          415-368-1100
aud_type       UB                              <-Update Before
aud_time       2014-10-16 12:03:19.00000       <-TimeStamp
aud_user       1042                            <-User-Id auf Source
```

und ein AFTER-IMAGE:

```
customer_num   104
fname          Anthony
lname          Kaluzinski
…
phone          415-368-1100
aud_type       UP                              <-Update Post
aud_time       2014-10-16 12:03:19.00000
aud_user       1042
```

Beim Löschen wird der gelöschte Datensatz geschrieben, beim Einfügen der neue Datensatz:

```
delete from customer where customer_num = 103;
customer_num  103
fname         Gerd
lname         Pauler
…
phone         415-328-4543
aud_type      DL                              <-Delete
aud_time      2014-10-16 12:04:25.00000
aud_user      1042

insert into customer (lname, fname) values ("Kaluzinski","Carina")
customer_num  129
fname         Carina
lname         Kaluzinski
…
phone
aud_type      PT                              ← Insert
aud_time      2014-10-16 12:04:25.00000
aud_user      1042
```

Als Audit Type können folgende Kürzel vorkommen:

```
PT = insert (add)
UB = update (before image)
UP = update (after image)
DL = delete
RS = a dummy row inserted on the target to flag to any process
     querying or processing the data that a refresh was performed
RR = a row refreshed (i,e, inserted)
```

„Refresh“ entspricht der kompletten Neubefüllung der Tabelle auf der Zielseite mit den Daten der Quelle.

TechTipp: Infosphere CDC Replikation als „normalen Benutzer“ einrichten

Die Replikation zwischen INFORMIX und anderen relationalen Datenbanken kann mittels InfosSphere Change Data Capture (CDC) eingerichtet werden. Auch zwischen unterschiedlichen Informix Instanzen lässt sich diese Option nutzen, z.B. für das Auditing.

Wer vermeiden will, dass ein Benutzer mit Sonderrechten, wie „informix“ oder „root“, für die CDC Replikation Zugriff von Aussen auf die Informix Instanz hat, der kann dies über die folgenden Schritte realisieren:

1. Die Installation der CDC-Informix-Komponente erfolgt als der Benutzer, der dann auch für die Übertragung der Daten genutzt werden soll. Dieser Benutzer MUSS bereits bei der Installation der Gruppe "informix" angehören. Nachträgliches Hinzufügen hilft nicht mehr.

2. Da die CDC Replikation drei Tabellen in der zu replizierenden Datenbank aufbaut, benötigt der Benutzer das Recht, Tabellen zu erstellen. Hierzu reicht das Recht „resource“ aus. So kann z.B. mit dem Befehl "grant resource to <user>" auf der Quell-Datenbank die Basis geschaffen werden, damit der Benutzer die für die Replikation notwendigen Tabellen „ts_auth“, „ts_bookmark“ und „ts_confaud“ erstellen darf.

3. Die Datenbank „syscdcv1“ muss als Benutzer „informix“ mit Hilfe des Befehls
„dbaccess sysmaster $INFORMIXDIR/etc/syscdcv1.sql“
erstellt werden. Anschliessend müssen dem Benutzer, der für die Replikation eingesetzt wird, die DBA-Rechte auf der Datenbank erteilt werden, indem der Befehl
"grant dba on syscdcv1 to <user>"
im dbaccess auf der Datenbank syscdcv1 abgesetzt wird.

4. Um die Daten, die übertragen werden sollen, lesen zu können, benötigt der Benutzer noch die entsprechenden Rechte. Ist INFORMIX ausschliesslich als Quell-Datenbank eingebunden, so reicht ein "grant select on ... to <user>" auf allen Tabellen, die repliziert werden sollen. Soll auch die Gegenrichtung genutzt werden, so sind die entsprechenden Rechte notwendig. Da bei der Konfliktbehandlung „always“ aus einem „update“ auch ein „insert“ werden kann, falls der Datensatz am Ziel nicht vorhanden ist, sollte dies bei der Rechtevergabe beachtet werden.

5. Die CDC-Informix-Komponente wird anschliessend als der gewünschte Benutzer mit dem Befehl „dmts64 -I <cdc_instanz>“ gestartet.

6. Schliesslich werden am Access-Server noch die Datastores für die Replikation erstellt. Als Benutzer für die Anbindung an die Datenbank wird hierfür ebenfalls nur der ausgewählte Benutzer mit seinem Passwort benötigt.

Ist das Zielsystem ebenfalls Informix und soll der Benutzer die zu replizierenden Tabellen neu anlegen, so wird auf dem Zielsystem zumindest das Recht „resource“ benötigt.

Termin: Informix Bootcamp mit TimeSeries + Zertifizierung

Auch in diesem Herbst findet wieder das beliebte Informix Bootcamp statt. Neben den Neuerungen des Informix Servers werden auch neue Features der TimeSeries vorgestellt. Veranstaltungsort ist Düsseldorf. Das Bootcamp findet von Dienstag, den 11.November bis Freitag, den 14. November statt. Aktuell, Mitte Oktober, stehen noch Restplätze zu dieser Veranstaltung verfügbar. Alle Teilnehmer an diesem Bootcamp können kostenlos an einer Informix Zertifizierung teilnehmen.
Der Link zur Anmeldung steht unter:
http://www.ibm.com/events/idr/idrevents/detail.action?meid=5614&ieid=12020

Termin: Informix Genero Bootcamp + Zertifizierung

Zum Thema Informix Genero findet ein Bootcamp vom 4. bis 6. November statt. Aktuell gibt es noch Restplätze. Der Veranstaltungsort ist Ehningen. Alle Teilnehmer an diesem Bootcamp können kostenlos an einer Informix Zertifizierung teilnehmen. Der Link zur Anmeldung ist:
http://www.ibm.com/events/idr/idrevents/detail.action?meid=8493&ieid=11786

Termin: 65. IUG-Workshop in Hamburg

Am Mittwoch, den 29. Oktober 2014 findet in Hamburg der 65. IUG Workshop statt. Das Thema ist diesmal:

Business Intelligence, Entwicklungs-Methode und Internet-Technologie

Wie gewohnt findet am Vorabend der beliebte IUG Stammtisch statt, an dem alte Verbindungen gepflegt, und neue Kontakte geknüpft werden können. Die Agenda wird in den nächsten Tagen zur Verfügung stehen.

Weitere Informationen und den Link zur Anmeldung finden Sie unter:
http://www.iug.de/index.php?option=com_content&task=view&id=296&Itemid=383

Anmeldung / Abmeldung / Anmerkung

Der Newsletter wird ausschließlich an angemeldete Adressen verschickt. Die Anmeldung erfolgt, indem Sie eine Email mit dem Betreff „**ANMELDUNG**“ an **ifmxnews@de.ibm.com** senden.
Im Falle einer Abmeldung senden Sie „ABMELDUNG“ an diese Adresse.

Das Archiv der bisherigen Ausgaben finden Sie zum Beispiel unter:

http://www.iiug.org/intl/deu
http://www.iug.de/index.php?option=com_content&task=view&id=95&Itemid=149
http://www.informix-zone.com/informix-german-newsletter
http://www.drap.de/link/informix
http://www.nsi.de/informix/newsletter
http://www.cursor-distribution.de/index.php/aktuelles/informix-newsletter
http://www.listec.de/Newsletter/IBM-Informix-Newsletter/View-category.html
http://www.bereos.eu/software/informix/newsletter/

Die hier veröffentlichten Tipps&Tricks erheben keinen Anspruch auf Vollständigkeit. Da uns weder Tippfehler noch Irrtümer fremd sind, bitten wir hier um Nachsicht falls sich bei der Recherche einmal etwas eingeschlichen hat, was nicht wie beschrieben funktioniert.

Die Autoren dieser Ausgabe

Gerd Kaluzinski IT-Specialist Informix Dynamic Server und DB2 UDB
IBM Software Group, Information Management
gerd.kaluzinski@de.ibm.com +49-175-228-1983

Martin Fuerderer IBM Informix Entwicklung, München
IBM Software Group, Information Management
martinfu@de.ibm.com

Sowie unterstützende Teams im Hintergrund.

Fotonachweis: Gerd Kaluzinski
(Torte, nicht selbst angefertigt, aber selbst in Teilen nach dem Foto gegessen)