Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

das Team der INFORMIX Newsletters wünscht Ihnen eine frohe und besinnliche Weihnachtszeit.

Wir bedanken uns bei allen Lesern für das rege Interesse an den vorgestellten Tipps und Tricks. Ein ganz besonderer Dank geht an denjenigen, die an das leibliche Wohl der Redaktion gedacht haben, und uns eine nette Weihnachtsüberraschung zukommen liessen. Wir werden die biologisch angebauten Nüsse, die Lebkuchen, das Marzipan und den Wein bei der Vorbereitung der kommenden Ausgabe als Stärkung sicher gebrauchen können. Kommen Sie gesund und glücklich ins Neue Jahr !

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 (5): NoSQL-Zugriff auf SQL-Daten

[Alle Beispiele wurden mit Informix Version 12.10.xC4 (auf Linux x86 64-bit) erprobt.]
Nachdem wir in der letzten Ausgabe des Newsletters mit SQL-Abfragen die Daten in der NoSQL-*Collection* verarbeitet haben, interessieren wir uns in dieser Ausgabe für den umgekehrten Weg: mit NoSQL-Befehlen der Mongo Shell wollen wir auf die Daten in normalen SQL-Tabellen zugreifen.

Abgesehen davon, dass wir hierzu die vielleicht noch etwas ungewohnten NoSQL-Befehle verwenden, ist der direkte Zugriff auf Daten in SQL-Tabellen denkbar einfach. Selbst unter der Annahme, dass wir das Schema der SQL-Tabellen nicht kennen, können wir uns erste Informationen aus dem Systemkatalog unserer Datenbank "stores_demo" holen. Um diese Informationen vorerst auf das Allernötigste zu beschränken, genügt es zu wissen, dass Daten zu den Tabellen selbst in der Tabelle "systables" gespeichert sind. Von dieser Tabelle interessieren uns vor allem die Namen der Tabellen, eventuell die Besitzer und die ID-Nummern, gespeichert in den entsprechenden Spalten "tabname", "owner" und "tabid". Da wir nicht an Tabellen des Systemkatalogs selbst interessiert sind, filtern wir nur die benutzererzeugten Tabellen. Dies sind Tabellen mit einer Tabellen-ID größer 99, so dass wir den Filter auf das Feld "tabid" mit dem Operator "$gt" (greater than) angeben könenn. Damit formulieren wir nach bereits bekanntem Muster folgenden "find()"-Befehl:

```
> db.systables.find({tabid:{$gt: 99}}, {tabid:1, owner:1, tabname:1})
  { "tabid" : 100, "owner" : "informix", "tabname" : "customer" }
  { "tabid" : 101, "owner" : "informix", "tabname" : "ext_customer" }
  { "tabid" : 102, "owner" : "informix", "tabname" : "orders" }
  { "tabid" : 103, "owner" : "informix", "tabname" : "manufact" }
  { "tabid" : 104, "owner" : "informix", "tabname" : "stock" }
  { "tabid" : 105, "owner" : "informix", "tabname" : "items" }
  { "tabid" : 106, "owner" : "informix", "tabname" : "state" }
  { "tabid" : 107, "owner" : "informix", "tabname" : "call_type" }
  { "tabid" : 108, "owner" : "informix", "tabname" : "cust_calls" }
  { "tabid" : 109, "owner" : "informix", "tabname" : "catalog" }
  { "tabid" : 110, "owner" : "informix", "tabname" : "tab" }
  { "tabid" : 111, "owner" : "informix", "tabname" : "warehouses" }
  { "tabid" : 112, "owner" : "informix", "tabname" : "classes" }
  { "tabid" : 113, "owner" : "informix", "tabname" : "employee" }
  { "tabid" : 114, "owner" : "informix", "tabname" : "rating" }
  { "tabid" : 115, "owner" : "informix", "tabname" : "rating_v" }
  >
```

Wir sehen in der Ausgabe auch jeweils ein NoSQL-*Dokument* für die NoSQL-*Collection* "rating" und den View "rating_v", den wir mittels SQL-Befehl auf die Collection "rating" angelegt haben. Beide erscheinen genauso wie die SQL-Tabellen.

Die Ausgabe erfolgt nach dem Feld "tabid" geordnet, welches für die Tabelle "systables" des Systemkatalogs die natürliche Sortierreihenfolge ist, wie auch für eine entsprechende SQL-Abfrage mit einem SELECT-Befehl. Diese Sortierung entspricht der Reihenfolge, in der die Tabellen erzeugt wurden, was das jeweilige automatische Einfügen der zugehörigen Datensätze in die Tabelle "systables" verursacht hat.

Die Ausgabe der NoSQL-*Dokumente* können wir auch beliebig anders sortieren, z.B. alphabetisch nach dem Tabellennamen:

```
  > db.systables.find({tabid:{$gt: 99}}, {tabid:1, owner:1,
tabname:1}).sort({tabname:1})
  { "tabid" : 107, "owner" : "informix", "tabname" : "call_type" }
  { "tabid" : 109, "owner" : "informix", "tabname" : "catalog" }
  { "tabid" : 112, "owner" : "informix", "tabname" : "classes" }
  { "tabid" : 108, "owner" : "informix", "tabname" : "cust_calls" }
  { "tabid" : 100, "owner" : "informix", "tabname" : "customer" }
  { "tabid" : 113, "owner" : "informix", "tabname" : "employee" }
  { "tabid" : 101, "owner" : "informix", "tabname" : "ext_customer" }
  { "tabid" : 105, "owner" : "informix", "tabname" : "items" }
  { "tabid" : 103, "owner" : "informix", "tabname" : "manufact" }
  { "tabid" : 102, "owner" : "informix", "tabname" : "orders" }
  { "tabid" : 114, "owner" : "informix", "tabname" : "rating" }
  { "tabid" : 115, "owner" : "informix", "tabname" : "rating_v" }
  { "tabid" : 106, "owner" : "informix", "tabname" : "state" }
  { "tabid" : 104, "owner" : "informix", "tabname" : "stock" }
  { "tabid" : 110, "owner" : "informix", "tabname" : "tab" }
  { "tabid" : 111, "owner" : "informix", "tabname" : "warehouses" }
  >
```

Mit der Kenntnis der Tabellennamen können wir nun mit "find()"-Befehlen auf die Daten der Tabellen in der Datenbank "stores_demo" zugreifen. Die Cursormethode "count()" erlaubt uns, zuerst festzustellen, wieviele Datensätze in einer Tabelle enthalten sind. NoSQL *Cursormethoden* sind Funktionen, die auf ein Ergebnis angewendet werden. Im letzten NoSQL-Befehl haben wir die Cursormethode "sort()" verwendet, um das Ergebnis der "find()"-Funktion nach unseren Wünschen zu sortieren. Die "count()"-Funktion wird in gleicher Weise angewandt, obwohl sie nach SQL-Verständnis eine Aggregatfunktion ist und damit in SQL anders verwendet werden muss als ein ORDER BY zum Sortieren.

Der einfachste Anwendungsfall der Cursormethode "count()" ist das Anhängen an einen "find()"-Befehl. Damit werden alle Dokumente im Ergebnis von "find()" gezählt:

```
  > db.catalog.find().count()
  74
  >
```

Wir wissen also, dass ein einfacher "find()"-Befehl auf die Tabelle "catalog" ein Ergebnis mit 74 Dokumenten erzeugen wird. Damit wir nicht gleich beim ersten Ausführen von "find()" auf die Tabelle "catalog" von zu vielen Dokumenten überrollt werden, wollen wir zuerst nur ein einzelnes Dokument ansehen. Hierzu gibt es in der Mongo Shell die spezielle Funktion "findOne()", die einfach nur das erste gefundene Dokument anzeigt, ähnlich dem SQL-Befehl "SELECT FIRST 1 ...". Die Funktion "findOne()" bietet eine einfache Möglichkeit, sicher zu sein, dass nur ein Dokument im Ergebnis enthalten ist. Dies vereinfacht die Logik von Programmen, die das Ergebnis weiterverarbeiten - das Programm muss nicht für den Fall gerüstet sein, dass möglicherweise mehrere Dokumente zu behandeln sind, auch wenn dies nur erfordet, sie zu ignorieren oder einen Fehler auszugeben.

```
> db.catalog.findOne()
{
        "catalog_num" : 10001,
        "stock_num" : 1,
        "manu_code" : "HRO",
        "cat_descr" : "Brown leather. Specify first baseman's or
                       infield/outfield style.  Specify right- or
                       left-handed.",
        "cat_picture" : null,
        "cat_advert" : "Your First Season's Baseball Glove"
}
>
```

Die Funktion "findOne()" eignet sich auch bequem dazu, ersteinmal die Spalten und deren Namen einer SQL-Tabelle zu sehen. In diesem Fall wissen wir, dass es sich bei "catalog" um eine SQL-Tabelle handelt, erzeugt mit dem Aufbau der Demodatenbank "stores_demo". Somit sind wir auch sicher, dass alle anderen Dokumente, die wir in der Tabelle "catalog" finden, genau dieselbe Struktur haben, denn die SQL-Tabelle hat ein festes Schema. Im Gegensatz dazu können wir durch Anzeige eines einzelnen Dokuments einer NoSQL-Collection nicht auf eine gleiche Struktur aller anderen Dokumente in derselben Collection schliessen. Jedes Dokument einer Collection kann andere Felder beinhalten.

In unsere Collection "rating" haben wir Bewertungen für zwei Katalogartikel eingefügt: für die Katalognummern 10017 und 10019. Schauen wir uns also aus der Tabelle "catalog" die Datensätze mit diesen zwei Katalognummern an.

Für die "find()"-Funktion geben wir einen Filter mit dem "$or"-Operator an und spezifizieren die gewünschten Katalognummern als Array, umschlossen mit eckigen Klammern:

```
  > db.catalog.find({$or:[{catalog_num:10017},{catalog_num:10019}]})

  { "catalog_num" : 10017,
    "stock_num" : 101,
    "manu_code" : "PRC",
    "cat_descr" : "Reinforced, hand-finished tubular. Polyurethane
belted.
                  Effective against punctures. Mixed tread for super
wear
                  and road grip.",
    "cat_picture" : null,
    "cat_advert" : "Ultimate in Puncture Protection, Tires Designed for
                   In-City Riding"
  }

  { "catalog_num" : 10019,
    "stock_num" : 102,
    "manu_code" : "SHM",
    "cat_descr" : "Thrust bearing and coated pivot washer/spring sleeve
for
                  smooth action. Slotted levers with soft gum hoods.
                  Two-tone paint treatment. Set includes calipers,
levers,
                  and cables.",
    "cat_picture" : null,
    "cat_advert" : "Thrust-Bearing and Spring-Sleeve Brake Set
Guarantees
                   Smooth Action"
  }

  >
```

Aufmerksame Leser haben mittlerweile wohl schon bemerkt, dass beim NoSQL-Befehl "find()" auf SQL-Tabellen in den Ergebnisdokumenten kein Feld "_id" mit der automatisch erzeugten Objekt-ID enthalten ist. Es ist also daher nicht nötig, dieses Feld auszublenden durch explizite Angabe des Parameters {_id:0} für "find()", wie dies bei einer NoSQL-Collection erforderlich wäre:

```
  > db.rating.find()
  { "_id" : ObjectId("54085bc08ffc6ab2c496dddf"), "catalog_num" : 10017,
"rating_value" : 7 }
  { "_id" : ObjectId("54085bc18ffc6ab2c496dde0"), "catalog_num" : 10017,
"rating_value" : 9 }
  ...
  >
```

Wollen wir das Feld "_id" nicht in den Ergebnisdokumenten, aber auch nicht den Parameter {_id:0} bei "find()" angeben, so haben wir die Möglichkeit, den View "rating_v" zu benutzen, den wir mit einem SQL-Befehl auf die Collection "rating" angelegt haben. Wie für SQL-Tabellen erscheint auch für Views das Feld "_id" nicht automatisch in den Ergebnisdokumenten. Folgender Befehl auf den View "rating_v" benutzt "find()" ohne Parameter und anschliessend die Cursormethode "sort()", um das Ergebnis zuerst nach der Katalognummer und nachrangig nach der jeweiligen Bewertung zu sortieren:

```
  > db.rating_v.find().sort({catalog_num:1,rating_value:1})
  { "catalog_num" : 10017, "rating_value" : 4, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 5, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 7, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 7, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "rating_value" : 9, "comment" : null }
  { "catalog_num" : 10019, "rating_value" : 3, "comment" : null }
  { "catalog_num" : 10019, "rating_value" : 3, "comment" : null }
  { "catalog_num" : 10019, "rating_value" : 7, "comment" : null }
  { "catalog_num" : 10019, "rating_value" : 8, "comment" : null }
  { "catalog_num" : 10019, "rating_value" : 9, "comment" : "A very
simple to use product." }
  >
```

Da die Bewertungen und eventuell zugehörige Kommentare in einer separaten Collection gespeichert sind, ergibt sich auch für den Benutzer von NoSQL-Befehlen der Mongo Shell recht bald die Frage, wie nun die Dokumente in der Collection mit den Katalogdatensätzen, zu denen sie gehören, kombiniert werden können.

Für einen SQL-Anwender ist diese Frage wohl selbstverständlich, denn das relationale Datenmodell mit möglichst normalisierten Tabellen bedingt automatisch, dass Joins mit mehreren Tabellen erforderlich sind, um die Daten sinnvoll zu verwenden. Für einen Anwender von NoSQL-Datenbanken sind Joins jedoch viel weniger wichtig, da das Datenmodell normalerweise ganz anders aussieht. In einer reinen NoSQL-Datenbank würden wir sehr wahrscheinlich die Bewertungen direkt in der NoSQL-Collection der Katalogartikel abspeichern, z.B. als geschachteltes Dokument mit einem Array von mehreren Bewertungen. Die Notwendigkeit, Dokumente aus verschiedenen NoSQL-Collections mittels Join zu einem aussagekräftigen Ganzen zusammenzuführen, besteht bei einer NoSQL-Datenbank daher oft gar nicht. Alle Informationen zu einem Datenobjekt (in unserem Beispiel wäre dies ein Katalogartikel) befinden sich im selben NoSQL-Dokument und Joins sind somit nicht notwendig. In der Menge von NoSQL-Befehlen der Mongo Shell sind Joins von mehreren Collections nicht wirklich vorgesehen.

Im Falle einer Mischung von beiden Welten, SQL und NoSQL, wie in unserem Beispiel mit der erweiterten "stores_demo" Datenbank, benötigen wir auch bei Benutzung der NoSQL-Befehle sehr bald eine Möglichkeit, so etwas wie einen Join auszuführen. Es ist zwar theoretisch möglich, Funktionen zu definieren (z.B. in JavaScript), und diese dann mit der Funktion "mapReduce()" auf eine Collection anzuwenden und so eine Art Join zu implementieren. In der Informix Version 12.10.xC4 unterstützt der NoSQL Wire listener "mapReduce()" aber noch nicht, so dass dies für uns vorerst ausscheidet.

Dieses Dilemma lösen wir am einfachsten mit SQL, indem wir einen entsprechenden View definieren, der die gewünschten Joins enthält. Wie in der letzten Ausgabe wollen wir die Datensätze aus den Tabellen "catalog" und "stock" mit der Collection "rating" kombinieren. Für die Definition des neuen Views verwenden wir anstatt der Collection "rating" den schon bestehenden View "rating_v". Den neuen View "catalog_ratings" erstellen wir mit folgendem SQL-Befehl:

```
$ dbaccess stores_demo -
Database selected.

  > CREATE VIEW catalog_ratings (catalog_num, description, rating_value,
comment)
      AS SELECT c.catalog_num, s.description, r.rating_value, r.comment
        FROM catalog c, stock s, rating_v r
        WHERE r.catalog_num = c.catalog_num
          AND c.stock_num = s.stock_num
          AND c.manu_code = s.manu_code;

  View created.
  >
```

In der Mongo Shell benutzen wir nun den neuen View "catalog_ratings" wie eine normale Collection bzw. Tabelle. Zuerst prüfen wir mit der Cursormethode "count()", wieviele Dokumente ein einfacher "find()"-Befehl liefert:

```
  $ mongo localhost:26351/stores_demo
  MongoDB shell version: 2.4.9
  connecting to: localhost:26351/stores_demo
  > db.catalog_ratings.find().count()
  13
  >
```

Die 13 Ergebnisdokumente sollten alle einem Dokument in der Collection "rating" entsprechen, d.h. wir erwarten mit dem inneren Join des Views "catalog_ratings" keine Dokumente, zu denen es keine Bewertung gibt.

Kontrollieren wir mit dem entsprechenden "find()"-Befehl und sortieren das Ergebnis gleich wie zuvor nach Katalognummer und Bewertung:

```
  > db.catalog_ratings.find().sort({catalog_num:1, rating_value:1})
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 4, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 5, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 7, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 7, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 8, "comment" : null }
  { "catalog_num" : 10017, "description" : "bicycle tires",
"rating_value" : 9, "comment" : null }
  { "catalog_num" : 10019, "description" : "bicycle brakes",
"rating_value" : 3, "comment" : null }
  { "catalog_num" : 10019, "description" : "bicycle brakes",
"rating_value" : 3, "comment" : null }
  { "catalog_num" : 10019, "description" : "bicycle brakes",
"rating_value" : 7, "comment" : null }
  { "catalog_num" : 10019, "description" : "bicycle brakes",
"rating_value" : 8, "comment" : null }
  { "catalog_num" : 10019, "description" : "bicycle brakes",
"rating_value" : 9, "comment" : "A very simple to use product." }
  >
```

Über den Umweg, mit SQL-Befehlen einen entsprechenden View zu definieren, können wir auch mit den NoSQL-Befehlen der Mongo Shell Abfragen ausführen, die Daten von verschiedenen Tabellen und Collections miteinander kombinieren. Der im Beispiel erzeugte View "catalog_ratings" veranschaulicht dies in handlichem Format, da wir von den drei beteiligten Quellen nur die Spalten bzw. Felder ausgewählt haben, die uns gerade interessieren. In einem produktiven System würden wir bestimmt wesentlich mehr Spalten der SQL-Tabellen in den View einbauen, vielleicht sogar alle, so dass der View universeller verwendbar ist.

Was wir auf diese Weise allerdings nicht bekommen, das sind geschachtelte Dokumente, in denen die Bewertungen z.B. als Array dargestellt sind - was vielleicht eher den Vorstellungen eines eingefleischten NoSQL-Benutzers entsprechen würde.

In der nächsten Ausgabe des Newsletters ist das Thema eine erste Betrachtung des REST-API.

TechTipp: Reorganisieren von Indexen und Primary Keys

In früheren Ausgaben des Informix Newsletters haben wir beschrieben, wie sich Tabellen reorganisieren lassen, bei denen durch stetiges Wachstum immer mehr Extents hinzugekommen sind. Ein Blick auf die Liste der Extents zeigt jedoch, dass nicht nur Tabellen, sondern auch Indexe sehr viele Extents besitzen können. Nun erreichte uns die Frage, wie man diese Indexe reorganisieren kann, ohne sie zu löschen und neu aufzubauen. Gerade beim Primärschlüssel (Primary Key Constraint), zu dem ein impliziter Index erstellt wird, hätte ein Löschen die Folge, dass auch alle Fremdschlüssel (Foreign Keys) implizit mit gelöscht würden, und daher anschliessend neu erstellt werden müssten.

Aus diesem Grund zeigen wir an einem Beispiel die Reorganisation eines impliziten Index, der bei der Anlage des Primärschlüssels vom System automatisch erstellt wurde.

Schritt 1: Ermitteln der „partnum“ des zu reorganisierenden Index:

oncheck -pT <datenbank>:<tabelle>

(Implizite Indexe haben immer ein Leerzeichen am Beginn des Namens)

Schritt 2: In der Ausgabe des oncheck -pT die „Partnum“ des Index suchen, der reorganisiert werden soll.

Schritt 3: Den Task „defragment partnum“ ausführen:

```
database sysadmin;
execute function task ("defragment partnum",<partnum>)
```

Der Befehl kommt mit der Meldung **OK** zurück.

Schritt 4: Kontrolle der Reorganisation mittels erneutem Aufruf des Befehls "oncheck -pT"

Beispiel:

```
oncheck -pT <database>:<table>
TBLspace Report for <datenbankname>:<tabelle>
Physical Address                3:5
Creation date                   12/05/2014 12:22:20
...
Index  112_1 fragment partition datadbs in DBspace datadbs
     Physical Address                3:6
     Creation date                   12/05/2014 12:22:20
     TBLspace Flags                  802        Row Locking
     TBLspace use 4 bit bit-maps
     Maximum row size                2004
     Number of special columns       0
     Number of keys                  1
     Number of extents               29
     ...
```

```
        First extent size               4
        Next extent size                1024
        Number of pages allocated       6144
        Number of pages used            6144
        Number of data pages            0
        Number of rows                  0
        Partition partnum               3145731
        Partition lockid                3145730
```

Reorganisation mittels Task:

```
database sysadmin;
execute function task ("defragment partnum",3145731)
```

Erneuter Aufruf des „oncheck -pT“ zur Kontrolle:

```
    Index  112_1 fragment partition datadbs in DBspace datadbs
        Physical Address                3:6
        Creation date                   12/05/2014 12:22:20
        TBLspace Flags                  802         Row Locking
        TBLspace use 4 bit bit-maps
        Maximum row size                2004
        Number of special columns       0
        Number of keys                  1
        Number of extents               1
        ...
        First extent size               4
        Next extent size                4
        Number of pages allocated       6144
        Number of pages used            6144
        Number of data pages            0
        Number of rows                  0
        Partition partnum               3145731
        Partition lockid                3145730
```

TechTipp: dbimport [-nv]

Beim Importieren einer Datenbank mittels „dbimport“ werden nach dem Füllen der Daten die Fremdschlüssel angelegt. Während dieses Vorgangs werden die zugehörigen Constraints überprüft. Bei grossen Tabellen kann diese Überprüfung einen erheblichen Anteil an der Gesamtlaufzeit des „dbimport“ ausmachen.
Ist sichergestellt, dass die Quelle des „dbimport“ konsistent ist, so kann die Überprüfung deaktiviert werden, indem der Schalter „-nv“ (no validate) dem dbimport mitgegeben wird. Die Fremdschlüssel (foreign keys) der Datenbank werden beim Importieren in der Datenbank als „enabled“ erstellt, obwohl die Konsistenz nicht explizit geprüft wird.
Um die Funktionsweise dieser Option zu testen, haben wir eine Datenbank mit zwei Tabellen erstellt, die über einen Fremdschlüssel verbunden sind:

```
create table "informix".tab1
  (
    f1 integer,
    f2 char(3),
    primary key (f1)
  ) extent size 16 next size 16 lock mode row;

create table "informix".tab2
  (
    f3 integer,
    f4 integer
  ) extent size 16 next size 16 lock mode row;

alter table "informix".tab2 add constraint (foreign key (f3)
    references "informix".tab1  constraint "informix".fk1);
```

Nach einem „dbexport -ss“ der Datenbank, haben wir in der Tabelle „tab2“ das erste Feld eines Datensatzes in der Unload-Datei so verändert, dass dazu kein Fremdschlüssel in der Tabelle tab1 mehr existsiert.
Wird nun der „dbimport“ aufgerufen, so bricht dieser ab mit der Fehlermeldung:

```
alter table "informix".tab2 add constraint (foreign key (f3)
    references "informix".tab1  constraint "informix".fk1);
*** execute sqlobj
525 - Failure to satisfy referential constraint fk1.
111 - ISAM error:  no record found.
```

Verwendet man den Schalter „-nv“, so unterbleibt die Prüfung und die Datenbank wird vollständig geladen:

```
alter table "informix".tab2 add constraint (foreign key (f3)
    references "informix".tab1  constraint "informix".fk1);
dbimport completed
```

Hinweis:
Dieses Feature sollte mit der entsprechenden Vorsicht genutzt werden, da hierdurch Daten in eine Tabelle geschrieben werden können, die bei aktiven Constraints nicht in der Tabelle vorkommen dürften. Es wird daher dringend empfohlen diese Option ausschliesslich dann zu verwenden, wenn die Konsistenz der Daten anhand der Quelle (z.B. dbexport einer konsistenten Datenbank) gewährleistet ist.

TechTipp: dbimport [-D]

Seit Version 11.x wird beim dbimport anhand der Rowsize und der Anzahl der Datensätze eine Berechnung der Grössen für „Extent Size“ und „Next Size“ vorgenommen, falls diese nicht in der Schemadatei des „dbexport“ explizit angegeben wurden. Dies ist z.B. dann der Fall, wenn der dbexport nicht mit der Option „-ss“ erfolgt ist.
Werden Datentypen wie z.B. VARCHAR bzw. LVARCHAR verwendet, so kann die Berechnung anhand der Rowsize und Anzahl der Datensätze stark von der notwendigen Grösse der Tabelle abweichen. In diesem Fall kann mit der Option „-D“ die Grössenberechnung für „Extent Size“ und „Next Size“ deaktiviert werden, so dass die Defaultgrösse genutzt wird, wenn „Extent Size“ und „Next Size“ nicht explizit angegeben in der Schemadatei stehen.

TechTipp: Environment „NOVALIDATE“

Das im vorigen Artikel beschriebene Verhalten, dass Constraints beim Aufruf von „dbimport“ nicht überprüft werden, kann für das Erstellen von Fremdschlüsseln oder das Aktivieren von Constraints, die auf Disabled oder Filtering standen, durch das Setzen der Environment „NOVALIDATE“ erzwungen werden.
Vor dem entsprechenden Statement wird dann

```
set environment NOVALIDATE 'on'
```

aufgerufen. Der Vorteil liegt darin, dass die Zeit der Überprüfung eingespart wird.

TechTipp: dbaccess mit Shell-Commands

Die Ausgabe von Daten in eine Datei, die im Namen z.B. einen Zeitstempel haben soll, ist in Informix mit üblichem SQL nicht einfach zu realisieren.
Der Name einer Unload-Datei kann keine Variablen enthalten, so dass hier weder „today“ noch „current“ noch eine andere der mit SQL verfügbaren Zeitangaben verwendet werden kann.
Die Lösung für das Problem ist die Verwendung von Shell-Anweisungen innerhalb des SQL, die mit dem Escape-Zeichen „!“ beginnen.
Beispiel (fma_unload.sql):

```
unload to "/tmp/fma.unl"
select * from firma;
!mv  /tmp/fma.unl /tmp/firma_unload.$(date +%Y%m%d-%H%M%S)
```

Der Aufruf von „dbaccess <database> fma_unload.sql“ erstellt zuerst eine Datei mit dem Namen „fma.unl“, in die die Daten entladen werden. Anschliessend wird die Datei in den gewünschten Namen umbenannt:

```
ls /tmp/firma*:
firma_unload.20141213-234213
```

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

Markus Holzbauer IBM Informix Advanced Support
IBM Software Group, Information Management Support
holzbauer@de.ibm.com

Sowie unterstützende Teams im Hintergrund.

Fotonachweis: Gerd Kaluzinski (Sternschnuppenmarkt Wiesbaden)