Willkommen zum Informix Newsletter

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

das Jahr ist nun fast zu Ende. Die Weihnachtsmärkte locken mit Glühwein, Flammkuchen und Apfelküchle.
Wir hoffen, dass es für Sie ein glückliches und erfolgreiches Jahr war, und Sie nun die Tage bis Weihnachten gemütlich ausklingen lassen können.
Auch die Redaktion zieht sich für einige Tage zurück, bis wir im Neuen Jahr wieder tatkräftig für Sie da sind (falls es uns nicht einschneit).

In diesem Sinne wünschen wir eine erholsame und entspannte Weihnachtszeit !

Ihr TechTeam

TechTipp: ONCONFIG - AUTOLOCATE

Bei der Anlage von Datenbanken, Tabellen und Indexen besteht die Option, diese Objekte explizit in einem DBSpace zu erstellen, oder diese mittels Fragmentierung auf mehrere DBSpaces zu verteilen.
Wird beim Anlegen einer Datenbank nicht explizit ein DBSpace angegeben, so wird die Datenbank im ROOTDBS angelegt. Bei der Anlage von Tabellen und Indexen wird ohne die explizite Angabe des DBSpace der DBSpace verwendet, in dem die Datenbank erstellt wurde.

Mit Hilfe des Parameters AUTOLOCATE lässt sich die Auswahl des Speicherortes von Objekten automatisieren.
Der Parameter kann die Werte

0	Keine Verteilung der Objekte auf DBSpaces (default)
1-32	Verteilung der Objekte auf die vorgegebene Anzahl an DBSpaces (falls vorhanden)

annehmen.
Wird ein Wert ungleich 0 angegeben, so werden Datenbanken bei der Erstellung nicht im ROOTDBS erstellt, sondern in einem anderen der vorhandenen DBSpaces.

Der Wert lässt sich dynamisch mittels „onmode -wf“ ändern.
Der Parameter wird zudem vom Aufruf „SET ENVIRONMENT“ unterstützt.

Da es sich beim AUTOLOCATE um eine Fragmentierung mit „RoundRobin“ handelt, sind in der Workgroup Edition nur die Werte 0 und 1 erlaubt.

Das nachfolgende Beispiel verdeutlicht die Funktionsweise:

Fall 1 (AUTOLOCATE 0):

```
create table if not exists test (
f1   int,
f2   char(20)
);
```

Die Tabelle wird in dem DBSpace erstellt, der für die Erstellung der Datenbank angegeben wurde:

```
dbschema -d test -t test -ss

DBSCHEMA Schema Utility       INFORMIX-SQL Version 14.10.FC10
{ TABLE "informix".test row size = 24 number of columns = 2 index size = 13 }

create table "informix".test
  (
    f1 integer,
    f2 char(20)
  ) extent size 8 next size 16 lock mode row;
```

Fall 2 (AUTOLOCATE 3):

Die Tabelle wird über drei DBSpaces hinweg mittels „RoundRobin“ fragmentiert.

```
dbschema -d test -t test -ss

DBSCHEMA Schema Utility       INFORMIX-SQL Version 14.10.FC10
{ TABLE "informix".test row size = 24 number of columns = 2 index size = 13 }

create table "informix".test
  (
    f1 integer,
    f2 char(20)
  )
  fragment by round robin partition datadbs2_1 in datadbs2, partition datadbs3_2
    in datadbs3, partition datadbs_3 in datadbs
  extent size 8 next size 16 lock mode row;
```

Sollen nicht alle verfügbaren DBSpaces für die Fragmentierung der Tabellen genutzt werden, so kann dies mittels des Tasks „autolocate database“ geändert werden. Siehe hierzu den folgenden Artikel.

TechTipp: Task (“autolocate database“)

Der Task „autolocate database“ bietet die Möglichkeit je Datenbank anzugeben, in welchen DBSpaces die Objekte der Datenbank optimiert abgelegt werden sollen.

Der Aufruf lautet:

```
database sysadmin;
execute function task ('autolocate database','<database>','<dbspaces>');
```

Der Eintrag ist nur für die angegebene Datenbank gültig.
Die Fragmentierung erfolgt über die angegebenen DBSpaces, die in einer Liste mit Komma getrennt angegeben wurden.
Sind weniger DBSpaces angegeben als der Wert von AUTOLOCATE in der Konfiguration, so werden mehrere Fragmente in einem DBSpace erstellt.

TechTipp: Task (“autolocate database add“)

Der Task „autolocate database add“ bietet die Möglichkeit weitere DBSpaces zum Autolocate einer Datenbank hinzuzufügen.

Der Aufruf lautet:

```
database sysadmin;
execute function task ('autolocate database add','<database>','<dbspaces>');
```

Der Eintrag ist für die angegebene Datenbank gültig. Die Fragmentierung erfolgt über die DBSpaces, die bisher bereits beim Autolocate der Datenbank angegeben wurden, sowie über die neu hinzugefügten DBSpaces, die in einer Liste mit Komma getrennt angegeben werden.
Die Datenbank muss beim Aufruf existieren, so dass dieser Befehl nicht dazu genutzt werden kann, dass eine zukünftige Datenbank bei der Anlage automatisch in dem gewünschten DBSpace erstellt wird.

TechTipp: Task (“autolocate database remove“)

Der Task „autolocate database remove“ bietet die Möglichkeit DBSpaces aus der Liste des Autolocate einer Datenbank zu entfernen.

Der Aufruf lautet:

```
database sysadmin;
execute function task ('autolocate database remove','<database>','<dbspaces>');
```

Der Eintrag ist für die angegebene Datenbank gültig. Die Fragmentierung erfolgt über die DBSpaces, die bisher bereits beim Autolocate der Datenbank angegeben wurden, jedoch nicht mehr über die DBSpaces, die in einer Liste mit Komma getrennt angegeben werden.

ADD und REMOVE wirken sich nur auf Objekte aus, die neu erstellt werden. Bestehende Tabellen und Indexe werden dadurch nicht in der Fragmentierung geändert.

TechTipp: Task (“autolocate database anywhere“)

Der Task „autolocate database anywhere“ nutzt alle in Frage kommenden DBSpaces für die Verteilung der Objekte der Datenbank ausser dem ROOTDBS.

```
database sysadmin;
execute function task ('autolocate database anywhere','<database>');
```

Hierbei werden Alle DBSpaces genutzt, ausser DBSpaces, die für besondere Aufgaben erstellt wurden, wie z.B. der Physdbs, der mittels „onspaces -c -P“ erstellt wurde.

TechTipp: Task (“autolocate database off“)

Der Task „autolocate database off“ deaktiviert die automatische Verteilung der Objekte einer Datenbank auf die vorhandenen DBSpaces.

```
database sysadmin;
execute function task (‘autolocate database off’,‘<database>‘);
```

TechTipp: Autolocate und Json Wire Listener

Werden über den Json Wire Listener Collections erstellt, so wird implizit der Wert von AUTOLOCATE auf 1 gesetzt, so dass die Location der neuen Collection als ein Fragment in einem der DBSpaces abgelegt wird. Die Syntax der Fragmentierung mit einem Fragment ist hierbei als Ausnahme auch in der Workgroup Edition erlaubt.

TechTipp: Datentyp JSON / BSON

In den Ausgaben August 2014 bis Februar 2015 des Informix Newsletters haben wir uns mit der Anbindung einer Mongo Datenbank über den Json Wire Listener beschäftigt. Damals galt der Hauptaugenmerk der Zulieferung von Daten über die Mongo DB, sowie der Möglichkeiten der SQL Abfragen, der über Mongo erstellten Collections.

In dieser Ausgabe befassen wir uns mit der Möglichkeit JSON und BSON Daten direkt in der Informix Datenbank zu nutzen, ohne dass eine externe Anbindung erfolgt.

Die Datentypen JSON und BSON können bis zu 32k gross werden. Bis zu einer Grösse von 4k werden diese mit den anderen Daten der Tabelle in herkömmlichen DBSpaces gespeichert. Werte über 4k werden in SBSpaces ausgelagert.

JSON Daten bieten die Möglichkeit, dass ohne Schemaänderung einer Tabelle, unterschiedlichste zusätzliche Felder gespeichert und wieder abgefragt werden können. Beispiel für eine JSON Struktur sind:

```
{ "Vorname" : "Gerd",
  "Nachname" : "Kaluzinski",
  "Alter" : 42,
  "Beruf" : „Künstler“
}
```

JSON Daten erlauben auch einen hierarchischen Aufbau:

```
{ "Person" : {
     "Vorname" : "Gerd",
     "Nachname" : "Kaluzinski",
     "Alter" : 42,
     "Beruf" : [ "Künstler", "Autor" ]
     }
}
```

Die binäre Form von JSON ist BSON, die maschinell besser verarbeitet werden kann.

Bei den Möglichkeiten der Nutzung bestehen teilweise grosse Unterschiede. Diese werden in der folgenden Tabelle aufgelistet:

	Supported for BSON	/	**JSON data type**
Create with one or more columns of the data type	Yes	/	Yes
Alter table to add column of the data type	Yes	/	Yes
Drop a column of the data type	Yes	/	Yes
Create a B-tree index on column of the data type	No	/	No
Create an index or a functional index on a field	Yes	/	**No**
Truncate tables with columns of the data type	Yes	/	Yes
Fragment a table based on field values	Yes	/	Yes
Create a view based on columns of the data type	Yes	/	**No**
Compress data in a column of the data type	Yes	/	Yes
Create a cast on the data type	Yes	/	Yes
Create triggers on fields	Yes	/	**No**
Include a column of the data type in TimeSeries	Yes	/	**No**

	Supported for BSON /	JSON data type
Cast data	Yes /	**No**
Use a cursor to read data	Yes /	Yes
Select contents of column of the data type	Yes /	Yes
Select field values with dot notation	Yes /	**No**
Select field values with BSON processing functions	Yes /	**No**
Insert data	Yes /	Yes
Find the size of a field value	Yes /	**No**
Update statistics with columns of the data type	Yes /	Yes
Merge tables based on field values	Yes /	**No**
Join tables based on field values	Yes /	**No**
Load and unload data with the LOAD and UNLOAD	Yes /	Yes
dbschema, dbexport, and dbimport utilities	Yes /	Yes
Replicate data to RSS	Yes /	**No**
Shard data among Enterprise Replication servers	Yes /	**No**

Sehen wir uns an einem Beispiel die Nutzung der Datentypen an:

```
create table if not exists json_table
  (
    id int not null ,
    data1 "informix".bson,
    data2 "informix".json
  ) put data1 in (sbdbs);

insert into json_table values (
        42,
        '{
        "nr":"001",
        "Name":"Kaluzinski",
        "Vorname":"Gerd",
        "Location":"München",
        "Abteilung":"Expert Labs"
        }'::JSON,
        '{
        "nr":"001",
        "Name":"Kaluzinski",
        "Vorname":"Gerd",
        "Location":"München",
        "Abteilung":"Expert Labs"
        }'::JSON
)
;
```

Zuerst fällt auf, dass die Syntax für die Eingabe der Werte identisch ist. Es wird jeweils eine Zeichenfolge eingegeben, die auf den Datentyp JSON gecastet wird.

Bei der Abfrage unterscheiden sich die beiden Datentypen gewaltig. Währen JSON direkt lesbar ist, wird bei BSON der Teil der binären Speicherung angezeigt, der mit dem ASCII Zeichensatz darstellbar ist.

```
select * from json_table;

### Ergebnis:
id     42
data1  j
       nchen
data2  {
        "nr":"001",
        "Name":"Kaluzinski",
        "Vorname":"Ge
       rd",
        "Location":"München",
        "Abteilung":"Expert Labs"

             }
```

Die Daten in data1, welches im BSON Format gespeichert ist, lassen sich mittels eines Casts auf JSON analog der Daten in data2 darstellen.

Die in BSON gespeicherten Daten erlauben es, dass mit der „Punkt Notation“ die einzelnen Komponenten abgefragt werden:

```
select  data1.nr::int as Nummer,
        data1.Name::char(128) as Name,
        data1.Vorname::char(128) as Vorname,
        data1.Location::char(128) as Location,
        data1.Abteilung::char(128) as Abteilung
from json_table
;

### Ergebnis:
nummer     1
name        Kaluzinski
vorname    Gerd
location   München
abteilung  Expert Labs
```

Für die Verarbeitung des Datentyps BSON gibt es eine Reihe an Funktionen, die im nachfolgenden Artikel behandelt werden.

TechTipp: BSON Funktionen

Um mit Daten zu arbeiten, die im Format BSON gespeichert wurden, sind besondere Funktionen notwendig.

Folgende Funktionen stehen zur Verfügung:

- BSON_GET - Wert eines Elements extrahieren
- BSON_UPDATE - Wert eines Elements ändern
- BSON_SIZE - Länge in Byte eines Elements ausgeben
- BSON_VALUE_BIGINT - Wert eines Elements als BigInt ausgeben
- BSON_VALUE_BOOLEAN - Wert eines Elements als Boolean ausgeben
- BSON_VALUE_DATE - Wert eines Elements als Date ausgeben
- BSON_VALUE_INT - Wert eines Elements als Integer ausgeben
- BSON_VALUE_DOUBLE - Wert eines Elements als Double ausgeben
- BSON_VALUE_VARCHAR - Wert eines Elements als Varchar ausgeben
- BSON_VALUE_LVARCHAR - Wert eines Elements als LVarchar ausgeben
- BSON_VALUE_TIMESTAMP - Wert eines Elements als TimeStamp ausgeben
- BSON_VALUE_OBJECTID - Objektid eines mit Mongo erstelten Elements

Mit BSON_GET kann gezielt nach Inhalten der Komponenten einer BSON Struktur gesucht werden. BSON_UPDATE ermöglicht es, den Inhalt einer der Komponenten einer BSON Struktur zu ändern.

Beispiel:

```
update json_table
         set data1 = bson_update(
              data1,'{$set: {Location:"BAYERN"}}'
         )
      where bson_get(data1,"nr") = '{"nr":"001"}'::JSON::BSON


select      bson_value_varchar(data1'Name') as Name,
            bson_size(data1,'Name') as size
from json_table
```

Ergebnis:

```
Name  Kaluzinski
Size  15
```

Die Size 15 kommt daher, dass der Bezeichner „Name“ 4 Byte einnimmt, „Kaluzinski“ 10 Byte und der Trenner ‚:‘ ein Byte.

TechTipp: GENBSON Funktion

Mit der Funktion genbson() lassen sich Inhalte von herkömmlichen Tabellen in BSON/JSON Format umwandeln.

```
select genbson(customer)::JSON from customer
```

```
(expression)  {"_id":ObjectId("655cc2ecc9fb65230000000d"),"customer_num":101,"f
              name":"Ludwig","lname":"Pauli","company":"All Sports Supplies","a
              ddress1":"213 Erstwild Court","city":"Sunnyvale","state":"CA","zi
              pcode":"94086","phone":"408-789-8075"}

(expression)  {"_id":ObjectId("655cc2ecc9fb65230000000e"),"customer_num":102,"f
              name":"Carole","lname":"Sadler","company":"Sports Spot","address1
              ":"785 Geary St","city":"San Francisco","state":"CA","zipcode":"9
              4117","phone":"415-822-1289"}

(expression)  {"_id":ObjectId("655cc2ecc9fb65230000000f"),"customer_num":103,"f
              name":"Philip","lname":"Currie","company":"Phil's Sports","addres
              s1":"654 Poplar","address2":"P. O. Box 3498","city":"Palo Alto","
              state":"CA","zipcode":"94303","phone":"415-328-4543"}

(expression)  {"_id":ObjectId("655cc2ecc9fb652300000010"),"customer_num":104,"f
              name":"Anthony","lname":"Higgins","company":"Play Ball!","address
              1":"East Shopping Cntr.","address2":"422 Bay Road","city":"Redwoo
              d City","state":"CA","zipcode":"94026","phone":"415-368-1100"}
```

Mit genbson() kann auch eine Auswahl an Spalten für bestimmte Datensätze in das JSON-Format umgewandelt werden:

```
select tabid, tabname, genBSON(ROW(tabid, tabname, nrows, created, ustlowts), 0,
1)::JSON as json_data
FROM systables
where tabid in (13,23,42);
```

```
tabid         13
tabname       syschecks
json_data     {"tabid":13,"tabname":"syschecks","nrows":0,"created":ISODate("20
              22-10-11T00:00:00.000Z"),"ustlowts":ISODate("2023-11-24T09:16:11.
              755Z")}
tabid         23
tabname       systrigbody
json_data     {"tabid":23,"tabname":"systrigbody","nrows":0,"created":ISODate("
              2022-10-11T00:00:00.000Z"),"ustlowts":ISODate("2023-11-24T09:16:1
              1.938Z")}

tabid         42
tabname       sysopclasses
json_data     {"tabid":42,"tabname":"sysopclasses","nrows":2,"created":ISODate(
              "2022-10-11T00:00:00.000Z"),"ustlowts":ISODate("2023-11-24T09:16:
              12.375Z")}
```

Hinweis: IBM TechXchange in Barcelona

International Informix specific track and sessions

There will be **8 expert led sessions,** carefully curated by the International Informix User Group. Plus, hands-on experiences, product demonstrations, instructor-led labs, and certifications tailored for professionals

Sessions

01 **Informix Product Roadmap**
Scott Pickett and Carton Doe | IBM

02 **Things Fail! - Getting Serious About Redundancy!**
Tom Girsch | AutoEurope

03 **Capturing Data Changes from IBM Informix into your applications without polling using IBM Informix**
Art Kagel | AskDB

04 **Informix, the Database that does so much more**
Mike Walker | xDB Systems

05 **Informix and JDBC – ETL and DBA tools, tricks and traps .**
Adam Reynolds | Proficio

06 **Understanding how to use Informix asynchronous post commit triggers .**
Carlton Doe | IBM

07 **The data integration between Watsonx.data, IBM Informix and Siemens journey into the MetaVerse**
Henri Cujass| Leolo and Mahran Meissner | Siemens

08 **Database Operations (DBOps)**
Davorin Kremenjas and Marin Rasic

IBM TechXchange EMEA Summit

22-25 Jan 2024

International Barcelona Convention Center (CCIB)

3 full days of activities

70+ products covered

8 technical tracks

AI · Data · Automation · Security · Sustainability · Hybrid Cloud · Infrastructure · Quantum

1,500 technical peers to engage and networks

300+ sessions, demos, instructor-led labs, roadmap discussions

Don’t miss this chance to enhance your skill, stay ahead in your field, and connect with peers and experts across the industry

IBM

Nutzung des INFORMIX Newsletters

Die hier veröffentlichten Tipps&Tricks erheben keinen Anspruch auf Vollständigkeit.
Die IUG hat sich dankenswerterweise dazu bereit erklärt, den INFORMIX Newsletter auf ihren Web Seiten zu veröffentlichen.
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.
Rückmeldungen hierzu sind herzlich Willkommen !
Die gefundenen Tippfehler dürfen zudem behalten und nach Belieben weiterverwendet werden.
Eine Weiterverbreitung in eigenem Namen (mit Nennung der Quelle) oder eine Bereitstellung auf der eigenen HomePage ist ausdrücklich erlaubt. Alle hier veröffentlichen Scripts stehen uneingeschränkt zur weiteren Verwendung zur Verfügung.

Die Autoren dieser Ausgabe

Andreas Legner … ein INFORMIX Profi, der immer ein offenes Ohr für meine Fragen hat. Danke !
INFORMIX Advanced Support
HCL Software

Martin Fuerderer **…** ein Genie (*), das Technik und Rechtschreibung beherrscht.
Database Development
HCL Software

Gerd Kaluzinski … der im Job Alles notiert, was für den Newsletter interessant sein könnte und es dann hier zusammenfasst.
IBM Expert Lab DACH Consultant
Data & AI Software
gerd.kaluzinski@de.ibm.com +49-175-228-1983

(*) Martin wollte den Ausdruck „Genie“ streichen, aber Ehre wem Ehre Gebührt.
Ohne Martin hätte jede Ausgabe dutzende Tippfehler mehr. Zudem wurden immer wieder missverständliche Beispiele oder Formulierungen von Martin in eine eindeutige und verständliche Form gebracht.

Dank auch an die vielen Helfer im Hintergrund.

Nicht zu vergessen der Dank an die Informix User Group, ohne die es den INFORMIX Newsletters heute nicht mehr geben würde, und die dankenswerter Weise die Verteilung übernimmt.

Foto Nachweis:
Lindauer Weihnachtsmarkt 2023 (Gerd Kaluzinski)
(Einverständnis der abgebildeten Person liegt vor)