Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles....1
TechTipp: Informix und NoSQL (10): Indexe, Teil 2....2
TechTipp: Time-Moving-Objects (STS_GetFirstTimeByPoint) Teil 6....10
TechTipp: Time-Moving-Objects (STS_GetNearestObject) Teil 7....11
TechTipp: Optionen des ONMODE (onmode -F)....12
TechTipp: Optionen des ONMODE (onmode -wf / -wm)....13
TechTipp: Optionen des ONMODE (onmode -we)....14
TechTipp: Optionen des ONMODE (onmode -wi)....14
TechTipp: Änderung beim „Auto Update Statistics AUS“....15
Referenzen: INFORMIX erhält CISCO Excellence Award....15
Anmeldung / Abmeldung / Anmerkung....16
Die Autoren dieser Ausgabe....16

Aktuelles

Liebe Leserinnen und Leser,

der Herbst zeigt seine schönsten Farben. Nun heisst es die letzten warmen Tage in der Natur zu nutzen, bevor die ungemütliche Zeit beginnt. Falls es draussen doch schon zu ungemütlich ist, dann haben wir für Sie im Newsletter die Vorstellung weiterer Funktionen zu den Time-Moving-Objects und einige interessante Aufrufe des „onmode“ zusammengestellt. Zudem finden Sie hier die Fortsetzung der Reihe zu NoSQL und einen wichtigen Hinweis zum „auto update statistics“.

Viel Spaß mit den Tipps der aktuellen Ausgabe.

Ihr TechTeam

TechTipp: Informix und NoSQL (10): Indexe, Teil 2

[Alle Beispiele wurden mit Informix Version 12.10.xC5 (auf Linux x86 64-bit) erprobt.]

Im letzten Artikel haben wir die reguläre Collection-Funktion createIndex() der MongoDB API benutzt, um verschiedene Indexe auf unsere Collection "rating" anzulegen, und mit der Funktion explain() untersucht, wie solche Indexe bei Abfragen benutzt werden. Neben dieser regulären Syntax von createIndex() gibt es eine Informix-Erweiterung, mit der man den Typ der zu indizierenden Felder explizit angeben kann. In diesem Artikel betrachten wir, wie diese Erweiterung benutzt wird und welche Vorteile sich daraus für Abfragen ergeben.

Über unsere Beispiel-Collection "rating" wissen wir, dass alle Dokumente im Feld "rating_value" einen Integer-Wert enthalten. Damit ist das Feld ein geeigneter Kandidat für einen typ-spezifischen Index, um damit Abfragen mit Filter auf bestimmte Werte von "rating_value" oder mit Sortierung auf das Feld noch weiter zu verbessern. Die Angabe des expliziten Typs beim Aufruf der Funktion createIndex() ist neben der Sortierreihenfolge ein weiterer Parameter, der zum jeweiligen Feld gehört. Damit ergibt sich eine kleine Aufzählung, die wir in JSON-Manier mit eckigen Klammern umschliessen. Erstellen wir also einen Index auf das Feld "rating_value" mit aufsteigender Sortierung und vom Typ Integer:

```
> db.rating.createIndex({rating_value: [1, "$int"]})
>
```

Die Ausgabe der Funktion getIndexes() zeigt uns den Typ entsprechend an. Statt dem allgemeinen "$bson", das für Werte aller Typen gilt, sehen wir nun "$int":

```
> db.rating.getIndexes()
[
  {
  ...
  },
  {
    "ns" : "stores_demo.rating",
    "key" : {
      "rating_value" : [
        1,
        "$int"
      ]
    },
    "name" : "rating_value_1,$int",
    "v" : 1,
    "index" : "rating_rating_value_1$int"
  }
]
>
```

Versuchen wir nun, alle Dokumente abzufragen, deren "rating_value" sieben ist. Wie auch im letzten Artikel verwenden wir die Funktion explain(), um Informationen zum Query-Plan zu erhalten (anstatt der Ergebnisdokumente aus der Collection). Das Ergebnisdokument der Funktion explain() ist in den folgenden Beispielen jeweils mit den entsprechenden Zeilenumbrüchen und Leerzeichen 'aufbereitet', damit es hier im Text besser lesbar erscheint:

```
> db.rating.find({rating_value: 7},{_id:0}).explain()
{
  "version" : 1,
  "explain" :
    "QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 15:34:42)
    ------
    SELECT LIMIT 101 bson_new( data, '{ "_id" : 0.0 }')::bson as data
    FROM rating
    WHERE bson_get(data, 'rating_value')
          = '{ "rating_value" : 7.0 }'::json::bson

    Estimated Cost: 4
    Estimated # of Rows Returned: 1

      1) informix.rating: SEQUENTIAL SCAN  (Serial, fragments: ALL)

            Filters: informix.equal(
                     BSON_GET (informix.rating.data , 'rating_value' )
                     ,UDT )

    UDRs in query:
    --------------
        UDR id  :    -1654
        UDR name:    equal
        UDR id  :    506
        UDR name:    json_to_bson
        UDR id  :    473
        UDR name:    json_in"
}
>
```

Zu unserer Enttäuschung müssen wir feststellen, dass der Integer-Index gar nicht benutzt wird. Stattdessen erfolgt ein 'sequential scan'. Bei der geringen Datenmenge in unserem Beispiel wirkt sich das zwar nicht messbar auf die Schnelligkeit der Abfrage aus, jedoch sollte der extra erstellte Index trotzdem benutzt werden. Was ist nun schiefgelaufen? Bei genauerer Betrachtung des 'explain'-Textes sehen wir, dass im Verlauf der Abfrageverarbeitung aus dem Filterwert 7 ein 7.0 wurde. D.h. aus unserem Integer wurde irgendwie ein Float-Wert, und das verhindert die Benutzung des Integer-Indexes. Die Erklärung für diesen Umstand ist recht banal: Die Mongo Shell hat die grundsätzliche Eigenschaft, alle Zahlenwerte zu Float-Werten 'aufzubereiten', auch Integer-Werte.

Über den Sinn oder Unsinn dieses Verhaltens der Mongo Shell kann man sicher streiten, jedoch bleibt uns nichts anderes übrig, als diese Umwandlung zu verhindern. Eine Möglichkeit ist die Funktion NumberInt(), die wir wie folgt direkt in unsere Abfrage einbauen können:

```
> db.rating.find({rating_value: NumberInt(7)},{_id:0}).explain()
{
  "version" : 1,
  "explain" :
    "QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 15:55:53)
    ------
    SELECT LIMIT 101 bson_new( data, '{ "_id" : 0.0 }')::bson as data
    FROM rating
    WHERE bson_value_int(data, 'rating_value') = 7

    Estimated Cost: 1
    Estimated # of Rows Returned: 1

      1) informix.rating: INDEX PATH

        (1) Index Name: informix.rating_rating_value_1$int
            Index Keys: :informix.bson_value_int(data,'rating_value')
                        (Serial, fragments: ALL)
            Lower Index Filter: BSON_VALUE_INT
                                (informix.rating.data , 'rating_value' ) = 7

    UDRs in query:
    --------------
        UDR id  :    -1669
        UDR name:    bson_value_int"
}
>
```

Das Ergebnis im 'explain'-Text sieht nun schon viel besser aus, denn nun wird wie erwartet statt eines 'sequential scans' tatsächlich der Integer-Index benutzt. Aber was haben wir nun mit dem typ-spezifischen Index gegenüber dem allgemeingültigen Index vom letzten Artikel gewonnen? Vergleichen wir den obigen 'explain'-Text mit dem relevanten Teil des 'explain'-Textes für den allgemeinen Index (siehe auch die entsprechende Ausführung von find({rating_value: 7},{_id:0}).explain() im letzten Artikel):

```
Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) informix.rating: INDEX PATH

    (1) Index Name: informix.rating_rating_value_1_catalog_num__1
        Index Keys: :informix.bson_get(data,'rating_value')
                    :informix.bson_get(data,'catalog_num') (desc)
                    (Serial, fragments: ALL)
        Lower Index Filter: informix.equal(
                      BSON_GET (informix.rating.data , 'rating_value' ),UDT )
```

```
UDRs in query:
--------------
    UDR id  :    -1654
    UDR name:    equal
    UDR id  :    506
    UDR name:    json_to_bson
    UDR id  :    473
    UDR name:    json_in
    UDR id  :    503
    UDR name:    compare
    UDR id  :    -1686
    UDR name:    bson_get
    UDR id  :    503
    UDR name:    compare
    UDR id  :    -1686
    UDR name:    bson_get
```

Während die geschätzten Kosten und Anzahl der erwarteten Ergebnisdokumente gleichgeblieben sind, hat sich für den 'Lower Index Filter' und die Liste der 'UDRs in query' doch einiges verändert. (Wir können den Unterschied zwischen dem 'composite index' im allgeimen Fall vom letzten Artikel und dem 'simple index' im typ-spezifischen Fall getrost ignorieren.) Die Funktionen des 'Lower Index Filter' sind im typ-spezifischen Fall wesentlich einfacher, was sich auch in der Liste der 'UDRs in query' widerspiegelt: es wird nur die Funktion "bson_value_int" benötigt. Damit wird ein grosser Teil des Overhead für die allgemeine Verarbeitung beliebiger JSON key-value Paare eingespart, was sich bei grösseren Datenmengen auf jeden Fall positiv auf die Schnelligkeit der Abfrage auswirken sollte.

Die Benutzung der Funktion NumberInt() ist nur deshalb notwendig, weil die Mongo Shell unser Client ist. Mit einem anderen Client (z.B. über das REST API) wird die Umwandlung von einem Integer-Wert in ein Float von vornherein nicht stattfinden. Wollen wir auch mit der Mongo Shell nicht daran denken müssen, dass die Funktion NumberInt() für den Indexzugriff zu benutzen ist, so können wir als alternative Lösung einen weiteren typ-spezifischen Index anlegen, diesmal mit "$double". Danach schauen wir uns mit getIndexes() gleich die dann vorhandenen Indexe an:

```
> db.rating.createIndex({rating_value: [1, "$double"]})
> db.rating.getIndexes()
[
  {
  ...
  },
  {
    "ns" : "stores_demo.rating",
    "key" : {
      "rating_value" : [
        1,
        "$int"
```

```
      ]
    },
    "name" : "rating_value_1,$int",
    "v" : 1,
    "index" : "rating_rating_value_1$int"
  },
  {
    "ns" : "stores_demo.rating",
    "key" : {
      "rating_value" : [
        1,
        "$double"
      ]
    },
    "name" : "rating_value_1,$double",
    "v" : 1,
    "index" : "rating_rating_value_1$double"
  }
]
>
```

Wie erwartet haben wir nun auf das Feld "rating_value" der Collection "rating" zwei Indexe, einen vom Typ Integer und einen weiteren vom Typ Double. Versuchen wir also nochmal unsere Abfrage mit der Mongo Shell ohne die Funktion NumberInt():

```
> db.rating.find({rating_value: 7},{_id:0}).explain()
{
  "version" : 1,
  "explain" :
    "QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 16:11:18)
    ------
    SELECT LIMIT 101 bson_new( data, '{ "_id" : 0.0 }')::bson as data
    FROM rating
    WHERE bson_value_double(data, 'rating_value') = 7.0

    Estimated Cost: 1
    Estimated # of Rows Returned: 1

      1) informix.rating: INDEX PATH

        (1) Index Name: informix.rating_rating_value_1$double
            Index Keys: :informix.bson_value_double(data,'rating_value')
                        (Serial, fragments: ALL)
            Lower Index Filter: BSON_VALUE_DOUBLE
                                (informix.rating.data , 'rating_value' )
                                = 7.0000000000000000

    UDRs in query:
    --------------
        UDR id  :    -1660
        UDR name:    bson_value_double"
}
>
```

Da die Mongo Shell den Integer-Wert 7 in ein Double 7.0 umwandelt, kann nun der typ-spezifische Double-Index für die Abfrage genutzt werden. Da auch hier nur eine UDR nötig ist, in diesem Fall "bson_value_double", wird die Abfrage ähnlich schnell sein wie wenn der Integer-Index genutzt werden kann. Allerdings ist zu bedenken, dass nun zwei Indexe existieren, die bei Änderungen von Dokumenten in der Collection "rating" (insert, update oder delete) entsprechend gepflegt werden müssen. Der Aufwand hierfür ist mit zwei Indexen etwa doppelt so hoch, was sich bei Änderung vieler Dokumente (z.B. bei Batch-Verarbeitung) sicherlich besonders bemerkbar macht. Zudem benötigt der zusätzliche Double-Index auch entsprechend Speicherplatz. Das Anlegen eines (zusätzlichen) Double-Index speziell für Abfragen mit der Mongo Shell sollte also wahrscheinlich nur nach reiflicher Überlegung erfolgen.

Schliesslich wollen wir noch untersuchen, wie sich das Einfügen eines Dokuments auswirkt, wenn das Feld "rating_value" im Dokument einen String-Wert hat. Da "rating" nach wie vor eine NoSQL-Collection ist, sollte ja auch die Existenz von typ-spezifischen Indexen nicht die grundlegende Eigenschaft der Schemalosigkeit einer NoSQL-Collection kompromittieren. Zum Test fügen wir ein entsprechendes Dokument ein, bei dem wir dem Feld "rating_value" den String-Wert "n/a" zuweisen. Wir überzeugen uns auch gleich mit einem geeigneten Aufruf von find(), dass das Dokument dann tatsächlich vorhanden ist:

```
  > db.rating.insert({"catalog_num": 10017, "rating_value": "n/a", "comment":
"Administrator test insert"})
  > 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 }
  { "catalog_num" : 10017, "rating_value" : "n/a", "comment" : "Administrator
test insert" }
  >
```

Das Einfügen hat funktioniert, d.h. die Schemalosigkeit der Collection wurde durch die Existenz der typ-spezifischen Indexe nicht verletzl. Bleibt noch herauszufinden, was das nun für die mögliche Nutzung eines typ-spezifischen Indexes bedeutet. Hierzu stellen wir wieder die Abfrage nach allen Dokumenten mit einem "rating_value" von 7, mit Benutzung von NumberInt(), und explain() für den Query-Plan:

```
  > db.rating.find({rating_value: NumberInt(7)},{_id:0}).explain()
  {
    "version" : 1,
    "explain" :
      "QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 16:18:50)
      ------
```

```
      SELECT LIMIT 101 bson_new( data, '{ "_id" : 0.0 }')::bson as data
      FROM rating
      WHERE bson_value_int(data, 'rating_value') = 7

      Estimated Cost: 1
      Estimated # of Rows Returned: 1

        1) informix.rating: INDEX PATH

          (1) Index Name: informix.rating_rating_value_1$int
              Index Keys: :informix.bson_value_int(data,'rating_value')
                           (Serial, fragments: ALL)
              Lower Index Filter: BSON_VALUE_INT
                                  (informix.rating.data , 'rating_value' ) = 7

      UDRs in query:
      --------------
          UDR id  :    -1669
          UDR name:    bson_value_int"
}
>
```

Wie zuvor wird der typ-spezifische Integer-Index benutzt. Dies bedeutet, dass sich das Einfügen eines Dokuments, das zu dem typ-spezifischen Index gar nicht passt, nicht negativ auf den Query-Plan der Abfrage ausgewirkt hat. Bezüglich der Schnelligkeit von Abfragen, die einen typ-spezifischen Index nutzen können, haben wir also nicht viel zu befürchten.

Eine Abfrage, die sowieso keinen Integer-Index nutzen kann, wird hingegen auf einen 'sequential scan' zurückgreifen müssen und zusätzlich die verschiedenen BSON UDRs benötigen. Dies wird am Beispiel der folgenden Suche nach Dokumenten mit einem "rating_value" von "n/a" deutlich:

```
> db.rating.find({rating_value: "n/a"}, {_id:0}).explain()
{
  "version" : 1,
  "explain" :
    "QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 16:23:20)
    ------
    SELECT LIMIT 101 bson_new( data, '{ "_id" : 0.0 }')::bson as data
    FROM rating
    WHERE bson_get(data, 'rating_value')
          = '{ "rating_value" : "n/a" }'::json::bson

    Estimated Cost: 4
    Estimated # of Rows Returned: 1

      1) informix.rating: SEQUENTIAL SCAN  (Serial, fragments: ALL)

            Filters: informix.equal(
                     BSON_GET (informix.rating.data , 'rating_value' )
                     ,UDT )
```

```
        UDRs in query:
        --------------
            UDR id  :    -1654
            UDR name:    equal
            UDR id  :    506
            UDR name:    json_to_bson
            UDR id  :    473
            UDR name:    json_in"
    }
    >
```

Das Erstellen von typ-spezifischen Indexen mittels der Informix-Erweiterung von createIndex() kann also die Schnelligkeit entsprechender Abfragen durchaus positiv beeinflussen. Besonders sinnvoll sind typ-spezifische Indexe dann, wenn ein Grossteil der Dokumente einer Collection typ-konform ist, d.h. die indizierten Felder haben überwiegend Werte von dem Datentyp, für den der Index erstellt wird.

Hiermit haben wir den Exkurs über die Informix-Erweiterungen für typ-spezifische Indexe auf NoSQL-Collections beendet. Der Vollständigkeit halber wollen wir unsere Collection "rating" wieder in den ursprünglichen Zustand versetzen. Wir entfernen das "Demonstrations-Dokument" mit dem "rating_value" von "n/a" und die beiden typ-spezifischen Indexe:

```
> db.rating.remove({rating_value: "n/a"})
> db.rating.dropIndex({rating_value: [1, "$double"]})
{ "ok" : 1, "nIndexesWas" : 3 }
> db.rating.dropIndex({rating_value: [1, "$int"]})
{ "ok" : 1, "nIndexesWas" : 2 }
>
```

TechTipp: Time-Moving-Objects (STS_GetFirstTimeByPoint) Teil 6

Nachdem wir in der Ausgabe September 2015 bereits die Funktionen zur Ermittlung der Endposition (z.B. wichtig bei CarSharing) und zur aktuellen Position zu einem bestimmten Zeitpunkt kennengelernt haben, stellen wir diesmal die Funktion vor, die angibt wann sich ein Objekt erstmals im Nahbereich eines vorgegebenen Punktes befand. Die Syntax der Funktion „STS_GetFirstTimeByPoint“ ist:

```
STS_GetFirstTimeByPoint(
      ts_tabname LVARCHAR,
      obj_id LVARCHAR,
      ts TimeSeries,
      starttime DATETIME YEAR TO FRACTION(5),
      endtime DATETIME YEAR TO FRACTION(5),
      geometry ST_Point,
      max_distance REAL
)
```

Dabei wird eine maximale Distanz angegeben, die als Nahbereich betrachtet werden soll, der Bezugspunkt und die Zeitspanne, für die die Abfrage ausgeführt werden soll.

Als Ergebnis wird der Zeitpunkt ausgegeben, zu dem das Objekt erstmals näher als „max_distance“ am Referenzpunkt war.

Beispiel:

```
SELECT id, STS_GetFirstTimeByPoint(
      'moves_ts',
      1,
      data,
      '2015-07-24 08:40:00',
      '2015-07-24 09:00:00',
      '4326 point(47.552740 9.696854)',
      1000) as near_time
FROM moves_ts;
```

Das Ergebnis zeigt, dass Objekt 1 um 08:42:40 den Nahbereich von 120m betreten hat, Objekt 2 war bereits um 08:36:36 in dieser Distanz:

```
id near_time
 1 2015-09-21 08:42:40.00000
 2 2015-09-21 08:42:36.00000
```

TechTipp: Time-Moving-Objects (STS_GetNearestObject) Teil 7

Die Abfrage aus dem vorherigen Artikel lässt sich auch umdrehen, indem danach gefragt wird, welches Objekt zu einem gewissen Zeitpunkt die kleinste Entfernung zu einem Punkt hat. Dies könnte z.B. bei der Taxisuche eingesetzt werden.

Die Syntax der Abfrage lautet:

```
STS_GetNearestObject(
      ts_tabname LVARCHAR,
      ts_colname LVARCHAR,
      timestamp DATETIME YEAR TO FRACTION(5),
      geometry ST_Point,
      max_distance REAL
)
```

Beispiel:

```
SELECT UNIQUE STS_GetNearestObject(
      'moves_ts',
      'data',
      '2015-09-21 08:42:36.00000',
      '4326 point(47.552740 9.696854)',
      500) as nearest_object

FROM moves_ts;
```

Ergebnis:

```
nearest_object  2
```

Da wir aus dem vorherigen Artikel wissen, dass das Objekt 2 bereits früher in den Nahbereich dieses Punktes kam als Objekt 1, entspricht dies dem erwarteten Ergebnis.

In der nächsten Ausgabe des Newsletters werden wir die Funktionen vorstellen, mit denen abgefragt werden kann, ob ein Objekt einen Bereich durchquert (STS_TrajectoryCross), ob es in einem Bereich bleibt (STS_TrajectoryWithin) oder ob es einen Bereich durchquert und wieder verlässt (STS_TrajectoryIntersect).

Bleiben Sie gespannt und schalten Sie schon mal die Tracking Funktion der Handys Ihrer Kinder ein, damit Sie die Beispiele nachvollziehen können (natürlich nur bei gemeinsamen Wanderungen, nicht zur Überwachung).

TechTipp: Optionen des ONMODE (onmode -F)

Aufräumen ist immer eine lästige Arbeit, kostete Kraft und Zeit. Der Datenbank geht es dabei nicht anders. Nicht in allen Fällen werden Memorysegmente im virtuellen Memory der Informix Instanz sofort abgebaut, wenn diese nicht mehr von aktiven Applikationen genutzt werden. Dies kann dazu führen, dass die Informix Instanz täglich etwas mehr Memory belegt. Bei einem Restart werden die genutzten Memoryblöcke automatisch freigegeben.

Viele Informix Instanzen laufen über Monate oder sogar Jahre ohne einen Neustart. Daher ist ein „Aufräumen“ der ungenutzten Blöcke im Shared Memory wichtig, um nicht nach einiger Zeit in Engpässe beim verfügbaren Speicherplatz zu laufen.

Der Befehl „onmode -F“ bewirkt, dass die virtuellen Segmente der Informix Instanz nach Blöcken durchsucht werden, die keiner aktiven Applikation mehr zugeordnet sind. Diese werden dadurch freigegeben. Ist nach dem Lauf von „onmode -F“ ein gesamtes, nach dem Start der Instanz hinzugefügtes Segment des virtuellen Shared Memory ungenutzt, so wird dieses an das Betriebssystem zurückgegeben.

Die Empfehlung lautet daher, den „onmode -F“ als Cronjob regelmässig laufen zu lassen. Dabei sollte der Zeitpunkt so gewählt werden, dass grössere Verarbeitungen bereits abgeschlossen sind und zumindest ein Checkpoint danach stattgefunden hat.
Statt des Aufrufs im Cron kann diese Befehl auch als Task eingestellt werden.
Der Aufruf im SQL lautet dann: `EXECUTE FUNCTION task("onmode","F");`

Der Befehl „onmode -F“ muss alle Blöcke im virtuellen Shared Memory prüfen, daher hat der Aufruf Auswirkung auf die Performance. Da die Ausführung innerhalb weniger Sekunden abgeschlossen ist, hält sich dieser Nachteil gegenüber dem Vorteil eines bereinigten Memory in Grenzen.

Der Füllungsgrad des virtuellen Memory kann mittels „onstat -g seg“ angezeigt werden:

```
onstat -g seg

IBM Informix Dynamic Server Version 12.10.FC5W1 -- On-Line -- Up 02:33:05 --
904756 Kbytes
Segment Summary:
id          key         addr        size        ovhd     class blkused  blkfree
0           52804801    44000000    6283264     511864   R     1534     0
32769       52804802    445fe000    131072000   1537608  V     28562    3438
65538       52804803    4c2fe000    683634688   1        B     166903   0
98307       52804804    74ef5000    43479040    1        B     10615    0
131076      52804805    7786c000    561152      7848     M     136      1
3768368     52804806    778f5000    20480000    241512   V     59       4941
3801137     52804807    78c7d000    20480000    241512   V     59       4941
3833906     52804808    7a005000    20480000    241512   V     59       4941
Total:      -           -           926470144   -        -     207927   18262
```

Der Aufruf von „onmode -F“ wird im Messagelog (online.log) protokolliert:

```
10/19/15 10:49:29  Attempting to free unused operating system segments.
                        This operation may take several minutes.
10/19/15 10:49:29  Freed 3 shared memory segment(s) (61440000 bytes)
```

Anschliessend sind die nicht mehr genutzten Segmente abgebaut:

```
onstat -g seg

IBM Informix Dynamic Server Version 12.10.FC5W1 -- On-Line -- Up 02:40:39 --
844756 Kbytes
Segment Summary:
id          key         addr        size        ovhd     class blkused  blkfree
0           52804801    44000000    6283264     511864   R     1534     0
32769       52804802    445fe000    131072000   1537608  V     28450    3550
65538       52804803    4c2fe000    683634688   1        B     166903   0
98307       52804804    74ef5000    43479040    1        B     10615    0
131076      52804805    7786c000    561152      7848     M     136      1
Total:      -           -           865030144        -        -     207638   3551
```

Durch den Abbau der zusätzlichen Segmente hat sich auch die gesamte Speichernutzung der Instanz durch den Aufruf verringert.

TechTipp: Optionen des ONMODE (onmode -wf / -wm)

Immer mehr Parameter der Konfiguration können „online“ verändert werden. Die Änderungen erfolgen mittels „onmode -wf“ bzw. „onmode -wm“ und sind nach dem Aufruf sofort wirksam. Die Option „-wm“ ändert die Werte nur für die momentane Laufzeit der Instanz, wohingegen die Option „-wf“ die Werte ändert und die Änderungen in die Konfigurationsdatei „$INFORMIXDIR/etc/$ONCONFIG“ einträgt.

Der Aufruf kann statt über „onmode“ auch im SQL über einen Task erfolgen mit der Syntax:

```
EXECUTE FUNCTION task("onmode","wf","<parameter>=<value>");
```

Die Information, dass Parameter in der Konfiguration geändert wurden, wird allerdings erst beim nächsten Neustart der Instanz im online.log protokolliert.

TechTipp: Optionen des ONMODE (onmode -we)

Die aktuelle Konfiguration einer Informix Instanz kann mittels „onmode -we <filename>“ exportiert werden. Hierbei wird anhand der Vorlage der Datei „onconfig.std“ eine Datei geschrieben, die die Konfigurationsparameter mit den bekannten Kommentaren beinhaltet. Im Header der Datei steht der Hinweis, dass diese Datei generiert wurde:

```
###
# 10/19/15 21:02:51
# Export of current IDS configuration parameters
# Using template:  $INFORMIXDIR/etc/onconfig.std
###
```

Zusätzlich sind am Ende der Datei die Parameter zu finden, die im aktuellen System nicht gesetzt sind und die nur auf Anraten des Supports geändert werden sollten. Diese sind nach folgender Überschrift zu finden:

```
##########################################################################
# The following undocumented and unsupported parameters are recorded
# here in the interest of providing a comprehensive configuration
# snapshot. IBM STRONGLY recommends that you consult with Support
# before modifying any of these values.
##########################################################################
```

TechTipp: Optionen des ONMODE (onmode -wi)

Die Konfiguration einer Informix Instanz kann durch den Import einer Konfigurationsdatei angepasst werden. Parameter, die dynamisch mittels „onmode -wf“ änderbar sind, werden sofort wirksam, die restlichen Parameter werden in die Konfigurationsdatei $ONCONFIG eingetragen, damit diese beim nächsten Neustart wirksam werden.

Die Syntax lautet:

```
onmode -wi <Konfigurationsdatei>
```

Anpassungen mittels „onmode -wi“ sind eine Alternative um eine Vielzahl an dynamischen Änderungen mittels „onmode -wf“ zu einem Aufruf zusammenzufassen.

TechTipp: Änderung beim „Auto Update Statistics AUS“

Mit Version 12.10.xC5 haben sich die Parameter und das Verhalten des Tasks „**auto update statistics**“ geändert.
Der Parameter „AUS_CHANGE“, der den Prozentsatz der Änderungen angab, ab dem die Statistiken einer Tabelle oder eines Index neu gebildet wurden, entfällt. Dieser Schwellwert wird über den Parameter STATCHANGE der Konfiguration ($ONCONFIG) bestimmt.
Der Parameter AUS_AGE bleibt wie bisher erhalten und sorgt dafür, dass die Tabellen und Indexe spätestens nach AUS_AGE Tagen neue Statistiken erhalten. Dies erfolgt, indem dem Aufruf von „update statistics“ das Schlüsselwort „force“ angefügt wird. Der Defaultwert für AUS_AGE beträgt 30 Tage.

Die Reihenfolge der Erstellung der Statistiken wurde optimiert. Der Prozentsatz der Änderungen bestimmt nun die Reihenfolge der Statistikerstellung, so dass die Tabellen mit den meisten prozentualen Änderungen zuerst behandelt werden. Dadurch wird vermieden, dass ggf. durch das Zeitfenster, das für den Statistics Lauf definiert wurde, wichtige Aktualisierungen nicht mehr ausgeführt werden können.

Der Parameter AUS_AUTO_RULES bleibt unverändert mit dem Defaultwert 1 („aktiviert“). Dieser bewirkt, dass der „auto update statistics“ die folgenden Regeln befolgt:

- Alle Tabellen werden mit „UPDATE STATISTICS LOW“ aktualisiert
- Alle führenden Indexspalten werden mit „ UPDATE STATISTICS HIGH“ behandelt.
- Alle restlichen Indexspalten sind im „ UPDATE STATISTICS MEDIUM“ enthalten.
- Die „minimum resolution“ für MEDIUM ist auf 2.0 gesetzt.
- Die „minimum confidence“ für MEDIUM steht bei 0.95.
- Die „minimum resolution“ für HIGH steht auf 0.5.

Referenzen: INFORMIX erhält CISCO Excellence Award

IBM hat den „Software Excellence Award“ von Cisco für INFORMIX erhalten.
In den letzten Jahren verwendet Cisco einen Grossteil der Ausgaben für IBM Software für den Erwerb von Informix Lizenzen, die in die Produkte von Cisco eingebettet werden. Die Zusammenarbeit zwischen Cisco und Informix besteht seit vielen Jahren.
Mehr zu den Hintergründen und zur Zusammenarbeit finden Sie im Artikel der Internationalen Informix User Group unter folgendem Link:
http://www.iiug.org/insider/insider_sep15.php

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 (Wartburg)