Willkommen zum „IBM Informix Newsletter“

Inhaltsverzeichnis

Aktuelles

Liebe Leserinnen und Leser,

nachdem die Sommerferien auch im Süden vorüber sind, und in München mit dem Oktoberfest die fünfte Jahreszeit begonnen hat (Kölner kennen diesen Zustand zu anderer Zeit), haben wir uns in die Arbeit gestürzt und diesmal das neue Feature der „Time-Moving-Objects“ für Sie näher untersucht. Wer bisher bereits die Beiträge über NoSQL verfolgt hat, der erhält weitere, wichtige Tipps für die Performance. Zudem gehen wir aus aktuellem Anlass auf die Limits von Tabellen (bzw. Fragmenten) ein, und wie eine Überwachung dazu aussehen könnte.
Für die nächste Zeit stehen einige wichtige Ereignisse an, wie z.B. der 67. IUG Workshop in Hamburg.

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

Ihr TechTeam

Informix und NoSQL (9): Indexe, Teil 1

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

Nach einem Diskurs in den Bereich des REST-API und der beschleunigten Abfrage von NoSQL-Daten mittels IWA wenden wir uns in diesem Artikel wieder dem MongoDB-API zu, um Indexe auf NoSQL Collections anzulegen.

Genauso wie für relationale Daten in regulären Tabellen dienen Indexe auf NoSQL-Collections dazu, Abfragen zu beschleunigen oder sicherzustellen, dass Dokumente eindeutig sind. Indexe, die mit Befehlen der Mongo Shell erstellt werden, baut der Informix Server intern als Index auf die zugrundeliegende Tabelle auf. Somit sind Indexe auf Collections im Systemkatalog der Datenbank zu sehen. Es gibt jedoch auch in der Mongo Shell die Collection-Funktion getIndexes(), mit der wir uns Informationen zu existierenden Indexen einer Collection anzeigen lassen können. Führen wir die Funktion auf unsere Collection "rating" aus, so sehen wir, dass es schon einen Index gibt:

```
> db.rating.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : [
        1,
        "$string"
      ]
    },
    "ns" : "stores_demo.rating",
    "name" : "_id_",
    "unique" : true,
    "index" : "rating__id_"
  }
]
>
```

Wir erinnern uns, dass das Feld "_id" jeder Collection immer eindeutige Werte enthalten muss. Genau dazu dient dieser Index, der beim Erzeugen einer Collection automatisch angelegt wird. In der Ausgabe von getIndexes() bestätigt die Zeile "unique" : true, dass der Index auf das "_id" Feld ein 'unique index' ist.

Versuchen wir nun, einen neuen Index auf das Feld "rating_value" anzulegen. Hierzu verwenden wir die Funktion createIndex() und übergeben als Parameter {rating_value: 1}. Hierbei bedeutet die 1, dass der Index in aufsteigender Reihenfolge sortiert sein soll.

Als optionalen Parameter geben wir zusätzlich einen Namen für den Index an: "rating_value_idx":

```
> db.rating.createIndex({rating_value: 1}, {name: "rating_value_idx"})
>
```

Kontrollieren wir sofort mit einem erneuten Aufruf von getIndexes(), ob der neue Index nach unseren Wünschen angelegt wurde:

```
> db.rating.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : [
        1,
        "$string"
      ]
    },
    "ns" : "stores_demo.rating",
    "name" : "_id_",
    "unique" : true,
    "index" : "rating__id_"
  },
  {
    "ns" : "stores_demo.rating",
    "key" : {
      "rating_value" : [
        1,
        "$bson"
      ]
    },
    "name" : "rating_value_idx",
    "v" : 1,
    "index" : "rating_rating_value_idx"
  }
]
>
```

Die Ausgabe enthält alle erwarteten Informationen zu unserem Index "rating_value_idx" auf die Collection "rating" in unserer Datenbank "stores_demo". Insbesondere entnehmen wir dem Wert von "key", dass es ein Index auf ein einzelnes Feld, "rating_value" in aufsteigender Reihenfolge, ist. Das Fehlen des Feldes "unique" zeigt an, dass es sich um einen Index handelt, der Duplikate als Schlüsselwerte enthalten kann.

Die Existenz des Indexes können wir uns durch eine SQL-Abfrage auf den Systemkatalog noch bestätigen. Auch hier sehen wir am Wert 'D' der Spalte "idxtype", dass es sich bei "rating_value_idx" um einen 'duplicate index' handelt. Dem von uns angegebenen Indexnamen wird intern der Collection-Name vorangestellt. Ausserdem sehen wir auch den 'unique index' auf das "_id" Feld, sowie einen zusätzlichen internen Index:

```
> select idxname, i.owner, i.idxtype
  from systables t, sysindexes i
  where t.tabid = i.tabid and t.tabname = 'rating';

Database selected.

idxname   114_26
owner     informix
idxtype   U

idxname   rating__id_
owner     informix
idxtype   U

idxname   rating_rating_value_idx
owner     informix
idxtype   D

3 row(s) retrieved.

Database closed.
```

Es gibt in der Mongo Shell auch die Funktion ensureIndex(), die gleichbedeutend ist mit createIndex(). Seit Version 3 der MongoDB API ist ensureIndex() als veraltet eingestuft und nur noch ein Synonym für createIndex(). Daher verwenden wir in diesem Artikel nur createIndex().

Um die Funktionalität zur Verwaltung von Indexen in der Mongo Shell abzurunden, gibt es natürlich auch eine Funktion zum Entfernen von Indexen: dropIndex(). Als Parameter geben wir den Namen des Indexes an, der entfernt werden soll:

```
> db.rating.dropIndex("rating_value_idx")
{ "ok" : 1, "nIndexesWas" : 2 }
>
```

Die Antwort bestätigt, dass der Index ordentlich entfernt wurde. Wir können dies sofort durch einen Aufruf von getIndexes kontrollieren.

Es waren zwei Indexe, nach dem dropIndex() sollte also nur noch der Index auf das Feld "_id" übrig sein:

```
> db.rating.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : [
        1,
        "$string"
      ]
    },
    "ns" : "stores_demo.rating",
    "name" : "_id_",
    "unique" : true,
    "index" : "rating__id_"
  }
]
>
```

Bei unserer ersten Verwendung von createIndex() haben wir einen Namen für den Index angegeben. Da dieser Parameter aber optional ist, geht es auch ohne diese Angabe. Das probieren wir bei unserem nächsten Versuch aus. Damit dies nicht zu langweilig wird, erstellen wir diesmal einen sogenannten 'composite index', also einen Index, der auf mehrere Felder - in unserem Fall "rating_value" und "catalog_num" - angelegt wird. Wie zuvor wollen wir die Werte des Feldes "rating_value" im Index wieder aufsteigend sortiert haben. Die Werte des Feldes "catalog_num" wollen wir zur Abwechslung jedoch absteigend sortieren. Dazu geben wir statt einer "1" eine "-1" an:

```
> db.rating.createIndex({rating_value: 1, catalog_num: -1})
>
```

Zur Kontrolle rufen wir wieder getIndexes() auf:

```
> db.rating.getIndexes()
[
  {
  ...
  },
  {
    "ns" : "stores_demo.rating",
    "key" : {
      "rating_value" : [
        1,
```

```
          "$bson"
        ],
        "catalog_num" : [
          -1,
          "$bson"
        ]
      },
      "name" : "rating_value_1_catalog_num__1",
      "v" : 1,
      "index" : "rating_rating_value_1_catalog_num__1"
    }
]
>
```

Die Ausgabe zeigt, dass der 'composite index' auf die zwei angegebenen Felder erstellt wurde, wobei die Werte des Feldes "catalog_num" absteigend sortiert sind, erkenntlich an der "-1". Da wir keinen expliziten Namen spezifiziert haben, wurde intern ein Name erzeugt, der sich im wesentlichen aus den Namen der Felder und den Sortierreihenfolgen ergibt. Beim Blick in den Systemkatalog sehen wir den entsprechenden Indexnamen:

```
> select idxname, i.owner, i.idxtype
  from systables t, sysindexes i
  where t.tabid = i.tabid and t.tabname = 'rating';

idxname   114_26
owner     informix
idxtype   U

idxname   rating__id_
owner     informix
idxtype   U

idxname   rating_rating_value_1_catalog_num__1
owner     informix
idxtype   D

3 row(s) retrieved.

>
```

Wie bei SQL so ist bei NoSQL die Reihenfolge der Felder beim Erzeugen eines 'composite index' von Bedeutung. In unserem Beispiel bewirkt die Angabe {rating_value: 1, catalog_num: -1}, dass das Feld "rating_value" im 'composite index' an erster Stelle steht. Das Feld "catalog_num" hingegen steht an zweiter Stelle. Somit kann der Index auch für eine Abfrage genutzt werden, die nur einen Filter auf "rating_value" enthält. Formulieren wir dazu eine Abfrage, deren Ergebnis nur Dokumente enthalten soll mit dem Wert 7 im Feld "rating_value". Bei der geringen Anzahl von Dokumenten in unserer Collection "rating" werden wir allerdings anhand der Laufzeit nicht feststellen können, ob für die Abfrage der Index benutzt wird oder nicht. Daher verwenden wir zusätzlich die Funktion explain(), die wir mit einem '.' an die Abfrage anhängen. Als Antwort auf die mit .explain() erweiterte Abfrage erhalten wir nicht die Ergebnisdokumente der Collection, sondern Informationen zum Query-Plan:

```
> db.rating.find({rating_value: 7},{_id:0}).explain()
{
  "version" : 1,
  "explain" :
    "\nQUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 15:38:58)\n------\nSEL
     ECT LIMIT 101 bson_new( data, '{ \\\"_id\\\" : 0.0 }')::bson as dat
     a FROM rating WHERE bson_get(data, 'rating_value') = '{ \\\"rating_
     value\\\" : 7.0 }'::json::bson\n\nEstimated Cost: 1\nEstimated # of
      Rows Returned: 1\n\n  1) informix.rating: INDEX PATH\n\n    (1) In
     dex Name: informix.rating_rating_value_1_catalog_num__1\n        In
     dex Keys: :informix.bson_get(data,'rating_value') :informix.bson_ge
     t(data,'catalog_num') (desc)   (Serial, fragments: ALL)\n        Lo
     wer Index Filter: informix.equal(BSON_GET (informix.rating.data , '
     rating_value' ) ,UDT )\n\nUDRs in query:\n--------------\n    UDR i
     d  :\t-1654\n    UDR name:\tequal\n    UDR id  :\t506\n    UDR name
     :\tjson_to_bson\n    UDR id  :\t473\n    UDR name:\tjson_in\n    UD
     R id  :\t503\n    UDR name:\tcompare\n    UDR id  :\t-1686\n    UDR
      name:\tbson_get\n    UDR id  :\t503\n    UDR name:\tcompare\n    U
     DR id  :\t-1686\n    UDR name:\tbson_get\n"
}
>
```

Bei der Anzeige des Ergebnisdokuments gibt die Mongo Shell den Inhalt des Feldes "explain" in einer einzigen Zeile aus, die im obigen Beispiel mit Zeilenumbrüchen versehen wurde, damit das Ganze nicht allzu unhandlich aussieht. Besonders leserlich ist es trotzdem nicht. Wenn wir aber die enthaltenen Zeichen '\t' und '\n' ihrer Bedeutung entsprechend in Tabulator und 'line feed' umwandeln, dann tritt ein ziemlich bekanntes Format zum Vorschein:

```
QUERY: (OPTIMIZATION TIMESTAMP: 07-29-2015 15:38:58)
------
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: 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
```

Im Prinzip erhalten wir mit .explain() für eine NoSQL-Abfrage als Antwort im Feld "explain" den Text, der für eine entsprechende SQL-Abfrage mit set explain on; in die Datei *sqexplain.out* geschrieben wird. Wie erwartet sehen wir, dass für den Filter {rating_value: 7} unserer Abfrage der 'composite index' benutzt wird.

Um einen 'unique index' zu erstellen, übergeben wir der Funktion createIndex() als weiteren Parameter {unique: true}. Vorraussetzung für solch einen Index ist allerdings, dass in der Collection schon enthaltene Werte der entsprechenden Felder auch tatsächlich eindeutig sind. Andernfalls erhalten wir eine Fehlermeldung wie im folgenden Beispiel:

```
> db.rating.createIndex({rating_value: 1, comment: 1}, {unique: true})
Unable to create index rating_value_1_comment_1 for
namespace stores_demo.rating on table rating with
specification IfxDBObject { "ns" : "stores_demo.rating" ,
"key" : { "rating_value" : [ 1.0 , "$bson" ] ,
  "comment" : [ 1.0 , "$bson" ] } ,
"name" : "rating_value_1_comment_1" , "unique" : true ,
 "v" : 1 , "index" : "rating_rating_value_1_comment_1" }
| Cannot create unique index on column with duplicate data.
  (SQL Code: -371, SQL State: 23000)
| ISAM error: duplicate value for a record with unique key.
  (SQL Code: -100, SQL State: IX000)
>
```

Zum abschliessenden Entfernen des bestehenden Indexes können wir wieder den Namen verwenden, der (wie zuvor von getIndexes() angezeigt) als "rating_value_1_catalog_num__1" generiert wurde. Wenn uns das Nachschauen des generierten Namens mittels getIndexes() zu umständlich ist, dann können wir einen zu entfernenden Index genauso gut mit der Angabe identifizieren, mit der wir den Index angelegt haben. In unserem Fall war dies die Angabe der Felder mit der jeweiligen Sortierreihenfolge, also {rating_value: 1, catalog_num: -1}:

```
> db.rating.dropIndex({rating_value: 1, catalog_num: -1})
{ "ok" : 1, "nIndexesWas" : 2 }
>
```

Im einem der nächsten Informix Newsletter schauen wir uns zum Thema "NoSQL und Indexe" eine Informix-Erweiterung der Mongo Shell Syntax zu createIndex() an und untersuchen dann die Auswirkungen von diversen Aktionen auf die Benutzung der so erstellten Indexe.

TechTipp: Time-Moving-Objects (TimeSeries und Spatialdaten) Teil 1

Mit der Version 12.10.xC5 wurden die beiden Erweiterungen „TimeSeries“ und „Spatialdaten“ des Datenbankservers zu einer neuer Funktionalität kombiniert, den so genannten Time-Moving-Objects. Der Vorteil, der hieraus entsteht, ist die Kombination von Messwerten, die über die Zeit an unterschiedlichen Positionen erfasst werden. Oft ist die Position für Rückschlüsse der Messwerte von Bedeutung (z.B. ob es eine Stadt-, Berg- oder Autobahnfahrt war, bei der die Abgaswerte gemessen wurden).

An folgendem Beispiel wollen wie die Funktionsweise der Kombination aus Spatialdaten und TimeSeries verdeutlichen.

Wie immer bauen wie uns ein Testsystem auf, das wir mit Beispieldaten füllen.

Unsere Beispieldatenbank „moving_obj“ wird im „datadbs“ angelegt:

```
create database moving_obj in datadbs with log;
```

Da wir im Beispiel die Daten und Position jede Sekunde ablegen wollen, müssen wir hierzu einen Kalender mit Sekundenintervall erstellen. Als Start legen wir den ersten September fest:

```
INSERT INTO CalendarPatterns values ('calpat_1s','{1 on} second');

INSERT INTO Calendartable (c_name, c_calendar) values (
        'cal_1sec',
        'startdate(2015-09-01 00:00:00.00000),
         pattstart(2015-09-01 00:00:00.00000),
pattname(calpat_1s)'
);
```

Nun erzeugen wir den Rowtype, der die Daten aufnehmen soll:

```
create row type r_position (
        tstamp datetime year to fraction(5),
        longitude       float,
        latitude        float,
        elevation       float,
        speed           float
);
```

Zur Ablage wird ein Container für den Rowtype erstellt und dann mit dem Rowtype die Tabelle definiert:

```
execute procedure TSContainerCreate (
        'moves_cont', 'datadbs', 'r_position', 2048, 2048);

create table moves_ts (
    id serial primary key,
    data timeseries(r_position)
) lock mode row;
```

Um die Daten auch mit herkömmlichen Selects abfragen und einfügen zu können, erstellen wir zudem eine virtuelle Tabelle:

```
execute procedure TSCreateVirtualTab (
        'moves_vt',
        'moves_ts',
        'origin(2015-09-21 00:00:00.00000),
         calendar(cal_1sec),
         container(moves_cont),
         threshold(0),
         regular',0,'data');
```

... und laden über die virtuelle Tabelle einige Beispieldaten aus einer Datei:

```
load from 00_moving_object.data
insert into moves_vt
```

Als Testdaten haben wir einen kleinen Ausflug mit dem Rad genutzt:

```
1|2015-09-21 08:42:10|47.549957|9.692173|408.2|18|
1|2015-09-21 08:42:20|47.549841|9.694383|409.100|15|
1|2015-09-21 08:42:30|47.551173|9.694898|410.300|19|
1|2015-09-21 08:42:40|47.551738|9.696851|409.600|16|
1|2015-09-21 08:42:50|47.551593|9.700563|409.400|14|
1|2015-09-21 08:43:00|47.550840|9.703202|408.100|8|
1|2015-09-21 08:43:10|47.548827|9.704361|406.500|4|
2|2015-09-21 08:42:10|47.547958|9.690174|408.2|28|
2|2015-09-21 08:42:20|47.548843|9.695382|409.100|25|
2|2015-09-21 08:42:30|47.550171|9.695895|410.300|29|
2|2015-09-21 08:42:40|47.553736|9.697855|409.600|26|
2|2015-09-21 08:42:50|47.553594|9.703563|409.400|24|
2|2015-09-21 08:43:00|47.553843|9.706200|408.100|18|
```

Nach dieser Vorbereitung können wir uns nun im folgenden Artikel den einzelnen Funktionen widmen.

TechTipp: Time-Moving-Objects (STS_Init) Teil 2

Um mit Spatial-Moving-Objects zu arbeiten, muss ein so genannter „Spatiotemporal Index“ angelegt werden. Dies geschieht mit der Funktion STS_Init, die optional mit Defaultwerten, oder mit explizit angegebenen Parametern den Index und ggf. eine Reindizierung zusätzlicher Daten erstellen kann.

Die Syntax der Funktion lautet:

```
STS_Init(ts_tabname              VARCHAR(128))
returns INTEGER
```

Bzw. für die Angabe der spezifischen Parameter:

```
STS_Init(ts_tabname              VARCHAR(128)
     task_frequency INTERVAL DAY TO SECOND default "0 01:00:00",
     task_starttime DATETIME HOUR TO SECOND default NULL,
     ts_default_starttime DATETIME YEAR TO SECOND
          default "1970-01-01 00:00:00",
     ts_interval_to_process INTERVAL DAY TO SECOND
          default "0 01:00:00",
     ts_interval_to_avoid   INTERVAL DAY TO SECOND
          default "1 00:00:00"
)
returns INTEGER
```

Da wir alle 10 Sekunden Werte erhalten, triggern wir die Indizierung auf eine Minute. Dieser Wert wird an den Scheduler übergeben, der neue Werte verarbeitet.

```
EXECUTE FUNCTION STS_Init("moves_ts",
         "0 00:01:00",
         NULL,
         "2015-09-21 00:00:00",
         "0 00:10:00",
         "0 00:01:00"
);
```

Ist die Erstellung der Indizes erfolgt, so können Abfragen auf den Daten gestartet werden. Alle folgenden Artikel beziehen sich auf dieses Grundbeispiel.

TechTipp: Time-Moving-Objects (STS_GetLastPosition) Teil 3

Eine der einfachsten Fragen bei bewegten Objekten ist:

„Wo ist der Endpunkt der erfassten Strecke ?“

Bei CarSharing ist diese Frage besonders interessant, da hier der nächste Fahrer den Wagen findet. Die Antwort liefert die Funktion „STS_GetLastPosition“, die als Argument die TimeSeries erhält. Wir schränken in der Abfrage nicht auf ein Objekt ein und erhalten somit die Endpunkte aller erfassten Objekte:

```
SELECT id, STS_GetLastPosition(data) as last_pos  FROM moves_ts;
```

Das Ergebnis im Beispiel:

```
id        1
last_pos  4326 point(47.548827 9.704361)

id        2
last_pos  4326 point(47.553843 9.706200)
```

TechTipp: Time-Moving-Objects (STS_GetPosition) Teil 4

Interessanter ist die Frage, wo das Objekt zu einem gewissen Zeitpunkt war. Diese Abfrage erfolgt über die Funktion „STS_GetPosition“, die als Argumente die TimeSeries und einen Zeitpunkt übergeben bekommt:

```
SELECT
     id,
     STS_GetPosition(data, '2015-09-21 08:42:40') as pos_0842
FROM moves_ts;
```

Das Ergebnis liefert die Punkte, an denen die Objekte zum gesuchten Zeitpunkt waren als Koordinaten:

```
id        1
pos_0842  4326 point(47.551738 9.696851)

id        2
pos_0842  4326 point(47.553736 9.697855)
```

TechTipp: Time-Moving-Objects (STS_GetTrajectory) Teil 5

Eine weitere Möglichkeit ist, den Weg eines Objektes in einer gewissen Zeitspanne abzufragen. Hierzu stehen die Funktionen „STS_GetTrajectory“ und „STS_GetCompactTrajectory“ zur Verfügung. Beide Funktionen geben die Punkte aus, an denen sich ein Objekt im angegebenen Zeitraum nacheinander aufgehalten hat. Bei grossen Abfragen werden mit „STS_GetCompactTrajectory“ die Rückgabewerte komprimiert, die Ausgabe der Funktion „STS_GetTrajectory“ beinhaltet immer alle Punkte an denen sich das Objekt aufhielt.

Beispiel:

```
SELECT id,
     STS_GetTrajectory(data,
     '2015-09-21 08:42:10', '2015-09-21 08:42:50')
FROM moves_ts;
```

Ergebnis:

```
id            1
(expression)  4326 multilinestring((47.549957 9.692173, 47.549841
9.694383, 47. 551173 9.694898, 47.551738 9.696851, 47.551593
9.700563))

id            2
(expression)  4326 multilinestring((47.547958 9.690174, 47.548843
9.695382, 47. 550171 9.695895, 47.553736 9.697855, 47.553594
9.703563))
```

In der Ausgabe Oktober des Informix Newsletters lernen Sie weitere Funktionen zu „Time-Moving-Objects“ kennen, wie z.B. die Abfrage in welcher minimalen Entfernung zu einem Punkt sich die Objekte bewegt haben, von wann bis wann sich die Objekte in einem bestimmten Bereich befunden haben, welche Objekte zu einem gewissen Zeitpunkt in der Nähe eines Objektes sind (z.B. interessant für die Paketabholung oder Taxidienste), ob sich Wege kreuzen und viele weitere interessante Fragen. Bleiben sie neugierig ! Wir arbeiten intensiv an der Fortsetzung dieser Serie.

TechTipp: Table Limits – Erkennung und Vermeidung von Problemen

Wie jedes System hat auch eine Informix Datenbank Limits, die man kennen und überwachen sollte. Die hier betrachteten Limits gelten dabei je Fragment bzw. Partition einer Tabelle. Legt man eine Tabelle ohne Partitionierung/Fragmentierung an, so besteht die Tabelle aus einem initialen Fragment.
Fragmentierung und Partitionierung sind als Features nicht in der Workgroup Edition enthalten. Überschreitet eine Tabelle in der Workgroup Edition eines der Limits, dann ist die Installation ein direkter Kandidat für einen Upgrade auf die Enterprise Edition.

Je Partition bzw. Fragment sind folgende Limits zu beachten:

- Die maximale Anzahl an Pages beträgt 16777215 dies entspricht (2^16)-1
- Die maximale Anzahl an Datensätzen beträgt 4277659295 dies entspricht (2^24)-1
- Die maximale Anzahl an Extents hängt von der Struktur der Tabelle, den Datentypen und den Indices ab. Kritisch wird der Wert erfahrungsgemäss spätestens ab 250 Extents. Da die Berechnung recht komplex ist, haben wir hierzu eine SPL erstellt, die wir in der Ausgabe Oktober im Detail vorstellen wollen.

Um herauszufinden, ob das Limit der maximalen Anzahl an Datensätzen bald erreicht wird, kann der folgende Select ausgeführt werden:

```
select first 5
        n.tabname[1,20], dbsname[1,8] as db,
        4277659295 - h.nrows as rows_left,
        h.nrows,
        (round(100 * (4277659295 -
h.nrows)/4277659295,2))::char(3) as prc_left
from sysptnhdr h, systabnames n
where h.partnum = n.partnum
and n.tabname[1] != ' '
and h.flags > 0
order by 3
```

Die Ausgabe listet die fünf Tabellen bzw. Fragmente auf, bei denen prozentual die wenigsten Datensätze noch Platz finden würden. Wo die jeweiligen Grenzen für Warnungen oder Alarme liegen, muss individuell je System entschieden werden, da dies stark vom Wachstum der jeweiligen Tabellen abhängt.

Analog kann die Abfrage auf die restlichen Pages erfolgen, die zu einer Tabelle noch hinzugefügt werden können (weitere Extents).

```
select first 5
        n.tabname[1,24], dbsname[1,12] as db,
        16777215 - h.nptotal as pages_left,
        h.nptotal,
        (round(100 * (16777215 - h.nptotal)/16777215,2))::char(3)
as prc_left
from sysptnhdr h, systabnames n
where h.partnum = n.partnum
and n.tabname[1] != ' '
and h.flags > 0
order by 3
```

Die Abfrage betrachtet nur die der Tabelle zugeordneten Pages. Über den Füllungsgrad und ggf. die Menge an leeren Pages, die durch Löschen wieder nutzbar geworden sind, gibt diese erste Näherung keine konkrete Auskunft.

TechTipp: Plattformen für 12.10.xC5

Mit der Version 12.10.xC5 ist die Liste der unterstützten Plattformen erheblich erweitert worden. Folgende Einträge finden sich neuerdings in der Liste:

New 12.10.xC5 Platforms :

- Mac OS X 10.8, 10.9
- ARM v7
- Quark
- Raspberry Pi
- Ubuntu 64-bit for ARM v8 (64-bit)
- RHEL 7 certification for Linux x86-64 (64-bit)
- PPC64

Im Detail sind folgende Hinweise zu lesen:

- Raspberry PI devices certified for V6 ARM 32 with IDS 12.
- IDS Developer Edition 12 for ARM V7 32
- INTEL Quark (DK50) 32 bit and IDS 12
- Smaller runtime embed footprint using revised onconfig settings for devices.
- Mac OS 10.9 client/server support for Mac-based device developers

Termin: IUG Workshop in Hamburg

Am Mittwoch den **21. Oktober findet in Hamburg** der 67. IUG Workshop statt, der unter dem Motto steht „**Informix-Datenbank, Entwicklungsumgebungen und Tools**“.
Wie immer ist die Agenda mit interessanten Beiträgen gefüllt, die das Themengebiet in vielfältiger Weise abdecken. Am Vorabend findet wir immer der Stammtisch für die IUG-Mitglieder statt, an dem wichtige Kontakte bei gutem Essen und Wein geknüpft werden können.
Der Link zur aktuellen Agenda und zur Anmeldung ist unter folgendem Link zu finden:
http://www.iug.de/index.php?option=com_content&task=view&id=312&Itemid=399

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 (Sonnenuntergang - Bregenz)