Fehlerbehandlung in SQL – Was jedermann wissen sollte

Hermann Bär
Oracle USA
Redwood Shores, CA

Schlüsselworte
Datenbank, Fehlerbehandlung, set-based, SQL, error logging, exceptions

Einleitung
Eine der fundamentalen Grundregeln in der Datenanalyse ist „Garbage in – garbage out“, was nichts mehr bedeutet als dass die Qualität der Datenanalyse maximal nur so gut sein kann wie die Qualität der verwendeten Daten. Und wer kennt nicht die Situation wo anscheinend saubere Daten versucht werden in ein System zu laden und ein signifikanter Anteil der Daten aufgrund von Fehlern nicht ins System eingefügt werden können. Nicht umsonst ist ein großer Anteil der „Transformationen“ in ETL Werkzeugen damit beschäftigt Dateninkonsistenzen und Fehler in der einen oder anderen Weise zu bearbeiten .. .

Klassische Ansätze der Fehlerbehandlung sind oft in prozeduraler Natur. Grundsätzlich ist nichts gegen derartige Lösungen vorzubringen, jedoch sind viele dieser angewendeten Ansätze aus Performance-Gesichtspunkten langsamer als sogenanntes set-based processing in SQL. Und genau aus diesen Gründen haben Hersteller wie Oracle über die Jahre hinweg kontinuierlich in die Verbesserung der SQL Funktionalität investiert um genau dieses Problem zu adressieren.

Der folgende Artikel gibt einen Überblick über bestehende Funktionalität in der Oracle Datenbank um Fehlerbehandlungen direkt in SQL zu implementieren und diskutiert deren Vor- und Nachteile. Es ist auch wichtig darauf hinzuweisen dass „Fehlerbehandlung“ in diesem Kontext lediglich die Fähigkeit beschreibt dass die Datenverarbeitung in SQL ohne Fehler zu Ende geführt wird. Eine manuelle Nachbearbeitung der Daten um eventuelle Fehler semantisch zu bereinigen ist in vielen Fällen notwendig.

Der Artikel strebt weder an einen Komplett-Überblick über ETL Verarbeitung in der Datenbank zu geben noch eine Diskussion der Implementierung dieser Core-Funktionalität in ETL Werkzeugen. Er dient lediglich dazu um weniger bekannte Funktionalität sowohl als auch neue Funktionalität in der Oracle Datenbank 12c Release 2 vorzustellen[1].

Kategorisierung von Fehlern
Grundsätzlich können wir zwei Kategorien von Fehlern bei der Datenverarbeitung. Fehler die von vorneweg antizipiert werden können – jeder Auftrag muss einem existierenden Kunden zugeordnet sein – und Fehler welche entweder nicht planbar sind oder nicht mit Geschäftslogik abgebildet werden können. Die erste Kategorie der planbaren Fehler werden wir im Folgenden als *Business Rule Validations* beschreiben während die zweite Kategorie der nicht planbaren Fehler lediglich *Data Rule Validations* genannt werden. Innerhalb dieser Kategorien gibt es weitere Unterschiede.

[1] Oracle Datenbank 12c Release 2 ist in der Cloud als Exadata Express Edition für jedermann verfügbar.

Business Rule Validations
Jeder theoretisch möglicher Fehler in dieser Kategorie kann programmatisch abgefangen und bearbeitet werden kann. Wenn man im Voraus weiß welche Fehler auftreten können dann sind solche Fehler vermeidbar.
Viele solcher einfachen Geschäftsregeln sind als Constraints in der Datenbank implementiert. Der große Vorteil von Constraints im Allgemeinen ist diese direkt in der Datenbank implementiert sind. Es gibt keinen Weg aktive Constraints einer Tabelle zu umgehen; dies ist im Gegensatz zu Geschäftsregeln welche im Applikations-Tier implementiert werden. Jeder direkte Datenbankzugriff umgeht solche Regeln.

Die einfachsten Beispiele der Fehlerbehandlung der einfachsten Geschäftsregel – NOT NULL constraints, min/max check constraints - sind die Verwendung von SQL Standard-Funktionen wie DECODE() oder NVL. Das folgende einfache SQL Statement ermöglicht zum Beispiel dass sämtliche Aufträge des Tages geladen werden können, auch wenn keine Kunden-ID spezifiziert wurde:

```
INSERT INTO final_orders
SELECT nvl(cust_id, 9999), order_id, product_id, quantity, price
FROM orders_of_the_day;
```

Natürlich erfordert ein solcher Ansatz eine manuelle Nachverarbeitung der Aufträge mit der spezifizierten Dummy-Kundennummer 9999, aber das Hauptziel – alle Daten ohne Fehler zu laden – wurde erzielt. Es ist ultimativ wichtiger die Millionen von korrekten Datensätzen geladen zu bekommen und nicht die gesamte Verarbeitung wegen einem falschen Datensatz abzubrechen. Komplexere Regeln sind oft auch in PL/SQL mit entsprechendem Exception Handling implementiert. Derartige Methoden sind wohlbekannt und weit verbreitet und werden im Rahmen dieses Artikels nicht weitergehend behandelt.

Hätten wir das vorige Statement ohne Verwendung der NVL() Funktion durchgeführt wäre unsere Transaktion mit einem Fehler abgebrochen – entweder direkt bei der Durchführung des INSERT Statements oder wenn die Transaktion versucht wurde zu committen. Letzteres Scenario passiert mit sogenannten deferred constraints wo die Validierung der Constraints erst am Ende einer Transaktion durchgeführt wird. Es gelingt nicht falsche Daten ins System permanent einzufügen.

Temporäry Disabled Constraints

Eine elegante Erweiterung der verzögerten Validierung von Geschäftsregeln besteht mit temporär deaktivierten (disabled) constraints. Disabled constraints sind nach wie vor als Metadaten einer Tabelle existent, werden aber nicht bei der Dateneingabe oder -Manipulation angewandt. Der große Unterschied besteht hier darin dass wir mit disabled constraints Daten ins System einfügen und committen können (somit system-weit verfügbar machen), welche nicht unseren Geschäftsregeln (Constraints) entsprechen.

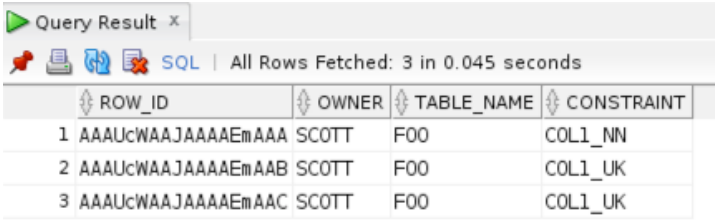
Die Validierung der Daten erfolgt in dem Moment wenn versucht wird den Constraint zu aktivieren mit der zusätzlichen Option falsche Datensätze (oder Datensätze mit Konflikt, z.B. für unique constraints) in eine sogenannte Exceptions-Tabelle einzufügen.

Um falsche Datensätze zu identifizieren muss die Syntax der Constraint-Aktivierung lediglich um eine exceptions clause erweitert werden, z.B.:

```
ALTER TABLE foo ENABLE CONSTRAINT col1_nn EXCEPTIONS INTO exceptions;
```

Sollte die Constraint Aktivierung falsche Datensätze identifizieren wird das Constraint nicht aktiviert. Das Format dieser Exceptions Tabelle ist vordefiniert und enthält alle notwendige Information um die falschen Datensätze zu identifizieren.[2]

Abbildung 1 zeigt exemplarisch welche Information in einer solchen Exceptions Tabelle gespeichert wird wenn sowohl ein NOT NULL constraint als auch ein UNIQUE KEY constraint verletzt werden:


Query Result

SQL | All Rows Fetched: 3 in 0.045 seconds

	ROW_ID	OWNER	TABLE_NAME	CONSTRAINT
1	AAAUcWAAJAAAAEmAAA	SCOTT	FOO	COL1_NN
2	AAAUcWAAJAAAAEmAAB	SCOTT	FOO	COL1_UK
3	AAAUcWAAJAAAAEmAAC	SCOTT	FOO	COL1_UK

Abb. 1: Exemplarischer Inhalt der Exceptions Tabelle

Die grundsätzliche Entscheidung welche Art der Validierung der Geschäftsregeln anwendbar ist bekommt primär eine Entscheidung ob falsche Datensätze im System temporär publiziert werden dürfen oder nicht. Ist eine temporäre Publikation nicht vertretbar müssen derartige Constraints immer im aktivierten Status bestehen. Eine Fehlerbehandlung muss vor der Datenmanipulation erfolgen.

DML Error Logging

Eine elegantere Lösung für eine bessere Fehlerbehandlung bietet die sogenannte DML Error Logging Funktionalität der Oracle Datenbank, eine leider weitgehend unbekannte und unterschätzte Funktionalität. Während einer DML oder Ladeoperation werden falsche Datensätze automatisch in eine Exceptions (DML Error Log) Tabelle geschrieben anstatt in die Zieltabelle. Es kombiniert quasi die Vorteile der beiden vorhergehend diskutierten Ansätze: es werden nur korrekte Datensätze eingefügt und falsche Datensätze kommen automatisch in Quarantäne. Dies bedeutet nichts Anderes als das falsche Datensätze in eine spezielle Exceptions (Error Logging) Tabelle geschrieben werden.

Zusätzliche Optionen der DML Error Logging Syntax sind die Spezifikation einer Transaktions-beschreibung und die Spezifikation einer maximalen Fehleranzahl. Speziell Letzteres ist eine sehr sinnvolle Ergänzung um neuen Code zu verifizieren.

Ein exemplarisches SQL Statement mit DML Error Logging ist z.B.:

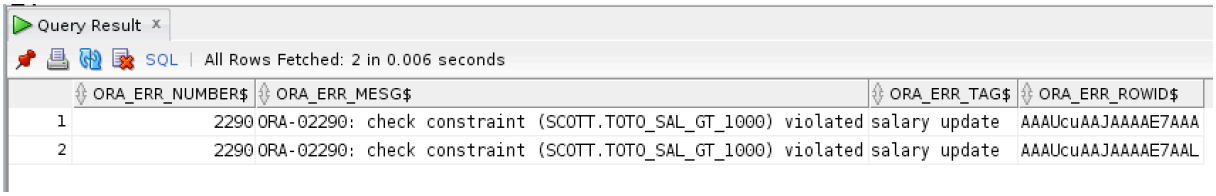
```
UPDATE toto SET sal=sal/10 LOG ERRORS ('salary update') REJECT LIMIT
UNLIMITED;
```

Mit der Spezifikation von REJECT LIMIT UNLIMITED obiges Statement wird komplett durchgeführt, selbst wenn sämtliche Update-Operationen in einen Fehler laufen würden; es werden zwischen null und allen Datensätze in der Tabelle upgedated.

Abbildung 2 zeig exemplarisch welche Art der Information in einer DML Error Logging Tabelle gespeichert werden. Zusätzlich zu dieser Metadateninformation werden auch die aktuellen Spaltenwerte abgespeichert.

---

[2] Template Tabelle in `$ORACLE_HOME/rdbms/admin/utlexpt.sql`



Query Result

All Rows Fetched: 2 in 0.006 seconds

	ORA_ERR_NUMBER$	ORA_ERR_MESG$	ORA_ERR_TAG$	ORA_ERR_ROWID$
1	2290	ORA-02290: check constraint (SCOTT.TOTO_SAL_GT_1000) violated	salary update	AAAUcuAAJAAAAE7AAA
2	2290	ORA-02290: check constraint (SCOTT.TOTO_SAL_GT_1000) violated	salary update	AAAUcuAAJAAAAE7AAL

Abb. 2: Exemplarischer Inhalt einer DML Error Logging Tabelle

DML Error Logging ist ein mächtiges Werkzeug um gewisse Fehlersituation speziell bei der Massendatenverarbeitung abzufangen ohne die Datenintegrität zu verletzen. Obwohl nicht alle DML Operationen diese Funktionalität einsetzen können (aus technischen Gründen, Details in der Dokumentation) ermöglicht es in vielen Anwendungsfällen robusteren Code zu schreiben und eventuelle Fehler in einem zusätzlichen Verarbeitungsschritt zu korrigieren. Die Identifizierung von falschen Datensätzen ist oft der schwierigste Schritt in einer Fehlerdiagnose in SQL; aufgrund des Paradigmas von „Alles-oder-nichts“ sind solche Daten in Millionen von Datensätzen nur schwer zu identifizieren.

Data Rule Validations

Mit DML Error Logging werden nicht nur Verletzungen von Geschäftsregeln abgefangen sondern auch jegliche Art von Verletzungen der Datenintegrität (wie z.B. zu lange Werte oder inkompatible Datentypkonvertierungen). Letztere Art der Fehler werden normalerweise als Data Rule Validations kategorisiert und stellen Fehlersituationen dar welche nicht von generell von vorneherein antizipiert werden.

Mit Oracle Datenbank 12c Release 2 steht für solche Situationen neue und erweiterte Funktionalität zur Verfügung welche im Folgenden vorgestellt wird. Im Kontrast zum DML Error Logging bieten diese Funktionen die Möglichkeit falsche Daten direkt mit Default Werten oder Verhalten zu ersetzen oder Datensätze generell als korrekt oder falsch zu identifizieren. Es findet keine separate Behandlung und Quarantäne falscher Datensätze.

Erweiterte CAST() und Datenkonvertierungsfunktionen

Die SQL Funktionen CAST() und die generellen Konvertierungsfunktionen mit TO_NUMBER, TO_CHAR, etc. erlauben eine Datentypkonvertierung wenn Daten verarbeitet werden. In früheren Versionen der Datenbank waren diese Funktionen allerding nicht in der Lage Fehler bei der Datentypkonvertierung abzufangen. Ein einziger Datensatz in Millionen von Daten führte zum Abbruch der Datenverarbeitung in SQL.

Mit der erweiterten Funktionalität kann nun ein Default Wert definiert werden welcher im Falle eines Fehlers falsche Datenwerte ersetzt. Im folgenden, einfachen SQL Statement werden z.B. sämtliche Werte welche nicht als NUMBER konvertiert werden können als Default Wert -99999 angezeigt.

```
SELECT CAST(r AS NUMBER DEFAULT  -9999 ON CONVERSION ERROR) r
FROM some_data;
```

Dieselbe Funktionalität steht nun auch mit sämtlichen expliziten Datenkonvertierungsfunktionen zur Verfügung. Dasselbe Statement unter Verwendung der TO_NUMBER() Konvertierung würde wie folgt aussehen:

```
SELECT TO_NUMBER(r DEFAULT  -9999 ON CONVERSION ERROR) r
FROM some_data;
```

Mit Hilfe dieser erweiterten Funktion können falsche Werte einfach und effizient bei der Datenkonvertierung durch einen default Wert ersetzt werden. Diese ermöglicht die fehlerfreie Eingabe von Datensätzen ohne die Gefahr das solche Datensätze zum Abbruch der gesamten Verarbeitung führen. Es gilt allerdings darauf hinzuweisen dass mit dieser Methodik die originalen falschen Werte nach der Verarbeitung nicht automatisch zur Verfügung stehen. Eine eventuelle Nachbearbeitung mit Bezugnahme auf die originalen falschen Werte ist daher nicht ohne Zuhilfenahme der Originaldaten möglich (was im Kontrast zur DML Error Logging Funktionalität steht).

Abbildung 3 zeigt ein exemplarischer einfaches SQL Statement mit Verwendung der erweiterten CAST() Funktion.

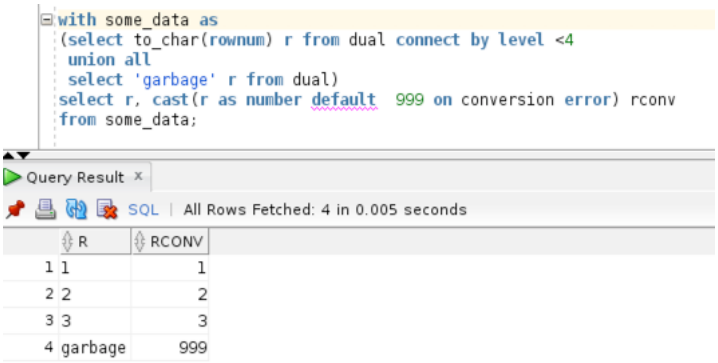


Abb. 3: Einfaches SQL Beispiel mit der erweiterten CAST() Funktionalität

Neue Funktion VALIDATE_CONVERSION()

Mit Oracle Datenbank 12c Release 2 steht uns mit der SQL Funktion VALIDATE_CONVERSION() ein weiteres Tool zur Verfügung um falsche Werte als Bestandteil einer Massendatenverarbeitung einfach und effizient zu identifizieren und zu isolieren.

Als dritte Variante bietet VALIDATE_CONVERSION() die Funktionalität, korrekte und falsche Datensätze zu identifizieren und dies z.B. als Filterkriterium im Rahmen eines SQL Statements zu verwenden. Unser früheres Beispiel einer TO_NUMBER() Konvertierung würde wie folgt formuliert sollten wir nur an den korrekten Werten interessiert sein:

```
SELECT CAST(r AS NUMBER) r FROM some_data
WHERE VALIDATE_CONVERSION(r AS NUMBER) = 1;
```

Unsere Funktion liefert als Returnwert 1 für alle Datensätze die konvertiert werden können und 0 für Datensätze die nicht erfolgreich konvertiert werden können (wo also r nicht als Nummer dargestellt werden kann.

Abbildung 4 zeigt exemplarisch einige Datensätze und den Returnwert der neuen SQL Funktion VALIDATE_CONVERSION() für eine Konvertierung zum Nummer Datentyp

```
with some_data as
(select to_char(rownum) r from dual connect by level <4
 union all
 select 'garbage' r from dual)
select r, validate_conversion(r as number) rconv
from some_data;
```

Query Result

SQL | All Rows Fetched: 4 in 0.007 seconds

	R	RCONV
1	1	1
2	2	1
3	3	1
4	garbage	0

Abb. 4: Einfaches SQL Beispiel mit der neuen VALIDATE_CONVERSION() Funktionalität

Erweiterte LISTAGG() Funktionalität

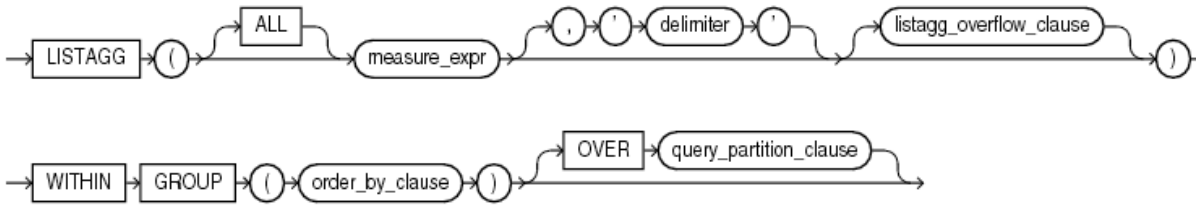
Seit der Version Oracle Database 11g Release 2 steht Oracle Entwickler eine spezielle SQL Funktion - LISTAGG() - zur Verfügung die es ermöglicht, Zeichenketten zu konkatenieren und als zusammengesetzten einzelnen Wert darzustellen. Dies ist eine für viele Entwickler sinnvolle Funktion um Listenwerte zu verketten und als einzelnen Datensatz zurückzuliefern, also eine Art Master-Detail Verknüpfung auf Master Ebene.

Die Verkettung findet zur Laufzeit statt, was zum Vorteil hat dass die Verknüpfungen vollkommen dynamisch sind – allerdings auch den Nachteil haben dass eine solche verknüpfte Liste die Maximallänge des Return Datenyps VARCHAR2 überschreitet (4000 bytes respektive 32767 bytes in Oracle Database 12c mit aktiviertem extended varchar Datentyp). Es gibt sogar eine spezielle Fehlermeldung im Kernel für dieses Problem:

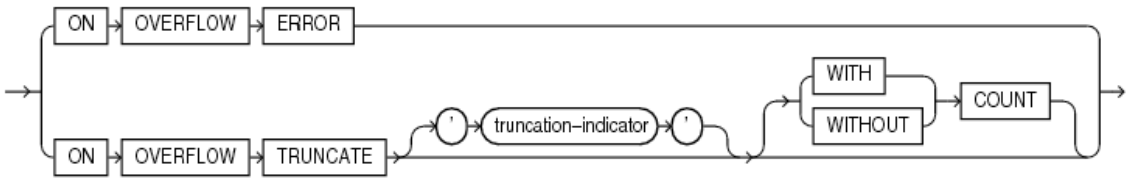
```
ORA-01489: result of string concatenation is too long
```

Oracle Database 12c Release 2 bietet hier Abhilfe mit neuer Funktionalität: mit Hilfe einer speziellen ON OVERFLOW Klausel wird definiert welcher String ein verkürztes Ergebnis repräsentiert und ob die Anzahl der abgeschnittenen Listenwerte mit angezeigt werden soll. Mit dieser einfachen Syntaxerweiterung gehört der oben genannte Laufzeitfehler der Vergangenheit an!

Die erweiterte Syntax von LISTAGG sieht somit folgendermaßen aus:



mit folgender Definition der **listagg_overflow_clause**



Es wurde ebenfalls Alternativsyntax implementiert welche das alte Verhalten – einen Laufzeitfehler bei Überschreitung der Maximallänge – darstellt. Dieses Verhalten ist default, aus Kompatibilitätsgründen zu älteren Versionen.

Ein einfaches exemplarisches Beispiel dieser neuen Funktionalität ist in Abbildung 5 zu sehen.

```
with some_rows
as
  (select rownum r from dual connect by level < 4000),
some_data
as
  (select mod(r,11) col1, rpad(r,mod(r,20),'0123456789') col2
   from some_rows),
some_fun
as
  (select col1, listagg(col2,' - ' on overflow truncate ' TBC ..  ' with count) within group (order by col2) col2
   from some_data group by col1)
select col1, length(col2), substr(col2, length(col2)-50)
from some_fun
order by 2 desc
fetch first 5 rows only;
```

Query Result

SQL | All Rows Fetched: 5 in 0.04 seconds

	COL1	LENGTH(COL2)	SUBSTR(COL2,LENGTH(COL2)-50)
1	6	3976	34567890 - 54501 - 5560123456789012 - TBC .. (61)
2	8	3976	678901234 - 569012345 - 59101234567 - TBC .. (59)
3	3	3975	1234567 - 54 - 5530123456789 - 5640 - TBC .. (58)
4	10	3974	- 57101234567 - 58 - 5930123456789 - TBC .. (57)
5	4	3973	5678 - 543 - 55401234567890 - 56501 - TBC .. (59)

Abb. 5: Exemplarische Verwendung der erweiterten LISTAGG() Funktionalität in Oracle Database 12c Release 2. Es sind sowohl der Qualifier für einen verkürzten Ausgabewert als auch die Anzahl der abgeschnittenen Listenwerte zu sehen

Zusammenfassung

Fehlerbehandlung in SQL ist eine oft vernachlässigte, jedoch geschäftskritische Funktionalität welche nicht nur die Robustheit und Fehleranfälligkeit von SQL Code erhöht, sondern auch eine Maßnahme für erhöhte Datenqualität. Die Tatsache das falsche Datenwerte zum Alltag gehören führt oft zu einer Vernachlässigung der aktiven Qualitätskontrolle in der Datenbank - durch Constraints oder korrekte Datentypen – um Fehlersituationen zu vermeiden. Es besteht die Hoffnung (oder Einbildung?) dass jegliche Art einer solchen Datenkontrolle im Applikationslayer erfolgen sollte, was jedoch keine absolute Garantie der Datenqualität darstellt.

Mit den vorgestellten Funktionen in der Datenbank kann mehr Datenqualität im Kernel gewährleistet werden ohne mit erhöhten Laufzeitfehlern rechnen zu müssen.

Kontaktadresse:
Hermann Bär
Oracle USA
400 Oracle Parkway
Redwood Shores, CA 94065
USA

Telefon:	+1 (0) 650-506-6833
Fax:	+1 (0) 650-506-6833
E-Mail	hermann.baer@oracle.com
Internet:	www.oracle.com