



Exasol Benutzerhandbuch

Version 6.0.10

**Empowering
analytics.**

Experience the world's fastest,
most intelligent, in-memory analytics
database.

Copyright © 2019 Exasol AG. Alle Rechte vorbehalten.

Überarbeitungen und andere Änderungen der in diesem Handbuch enthaltenen Informationen sind ohne Vorankündigung vorbehalten. Das Handbuch dient ausschließlich als Informationsquelle und enthält keine Verpflichtung. EXASOL ÜBERNIMMT KEINE HAFTUNG FÜR ETWAIGE FEHLER IN DIESEM HANDBUCH ODER DARAUS RESULTIERENDE SCHÄDEN. Die teilweise oder vollständige Vervielfältigung, Übersetzung, digitale Übertragung oder sonstige Konvertierung dieses Dokuments, ohne ausdrückliche schriftliche Zustimmung von Exasol, ist nicht gestattet.

Die verwendeten Marken- und Produktnamen sind durch Warenzeichen oder eingetragenen Warenzeichen geschützt und als solche gekennzeichnet. Die Warenzeichen oder eingetragene Warenzeichen sind Eigentum der entsprechenden Firmen.

Inhaltsverzeichnis

Vorwort	ix
Konventionen	xi
Änderungen in Version 6.0	xiii
1. Was ist Exasol?	1
2. SQL-Referenz	5
2.1. Grundlegende Sprachelemente	5
2.1.1. Kommentare in SQL	5
2.1.2. SQL-Bezeichner	5
2.1.3. Reguläre Ausdrücke	7
2.2. SQL-Befehle	11
2.2.1. Definition der Datenbank (DDL)	12
2.2.2. Manipulation der Datenbank (DML)	37
2.2.3. Zugriffssteuerung mittels SQL (DCL)	62
2.2.4. Anfragesprache (DQL)	75
2.2.5. Überprüfung der Datenqualität	86
2.2.6. Sonstige Befehle	91
2.3. Datentypen	107
2.3.1. Überblick über Exasol Datentypen	108
2.3.2. Details zu den Datentypen	108
2.3.3. Datentyp-Aliase	112
2.3.4. Typkonvertierungsregeln	113
2.3.5. Default-Werte	115
2.3.6. Identity-Spalten	117
2.4. Geodaten	119
2.4.1. Geo-Objekte	120
2.4.2. Geo-Funktionen	121
2.5. Literale	124
2.5.1. Numerische Literale	125
2.5.2. Boolesche Literale	126
2.5.3. Datum/Zeit Literale	126
2.5.4. Intervall Literale	126
2.5.5. Zeichenketten-Literale	128
2.5.6. NULL Literal	128
2.6. Format-Modelle	129
2.6.1. Datum/Zeit Format-Modelle	130
2.6.2. Numerische Format-Modelle	132
2.7. Operatoren	134
2.7.1. Arithmetische Operatoren	135
2.7.2. Konkatenationsoperator 	136
2.7.3. CONNECT BY Operatoren	137
2.8. Prädikate	137
2.8.1. Einleitung	138
2.8.2. Liste der Prädikate	138
2.9. Built-in-Funktionen	143
2.9.1. Skalare Funktionen	144
2.9.2. Aggregatsfunktionen	148
2.9.3. Analytische Funktionen	148
2.9.4. Alphabetische Liste aller Funktionen	151
3. Konzepte	267
3.1. Transaktions-Management	267
3.1.1. Allgemeines Konzept	267
3.1.2. Unterschiede zu anderen Systemen	268
3.1.3. Empfehlungen für den Anwender	268
3.2. Rechteverwaltung	268
3.2.1. User	269

3.2.2. Rollen	269
3.2.3. Privilegien	270
3.2.4. Zugriffskontrolle bei SQL-Befehlen	270
3.2.5. Metainformationen zur Rechteverwaltung	271
3.2.6. Rechteverwaltung und Transaktionen	271
3.2.7. Beispiel zur Rechteverwaltung	271
3.3. Prioritäten	272
3.3.1. Einleitung	273
3.3.2. Prioritäten in Exasol	273
3.3.3. Beispiel	274
3.4. ETL-Prozesse	274
3.4.1. Einführung	275
3.4.2. SQL-Befehle IMPORT und EXPORT	275
3.4.3. Scripting von ETL-Prozessen	276
3.4.4. Benutzerdefinierter IMPORT mittels UDFs	276
3.4.5. Benutzerdefinierter EXPORT mittels UDFs	278
3.4.6. Hadoop und andere Systeme	278
3.4.7. Virtual Schemas und ETL	279
3.4.8. Definition der Dateiformate (CSV/FBV)	279
3.5. Scripting-Programmierung	281
3.5.1. Einleitung	282
3.5.2. Generelle Elemente der Scripting-Sprache	283
3.5.3. Interaktion mit der Datenbank	291
3.5.4. Bibliotheken	298
3.5.5. Systemtabellen	305
3.6. UDF Skripte	305
3.6.1. Was sind UDF Skripte?	306
3.6.2. Einführende Beispiele	307
3.6.3. Details für die verschiedenen Sprachen	312
3.6.4. Das synchrone Cluster-Dateisystem BucketFS	339
3.6.5. Skript-Sprachen erweitern mittels BucketFS	341
3.7. Virtuelle Schemas	345
3.7.1. Virtuelle Schemas und Tabellen	346
3.7.2. Adapter und Properties	347
3.7.3. Zugriff auf virtuelle Tabellen ermöglichen	348
3.7.4. Privilegien zur Administration	349
3.7.5. Meta-Daten	350
3.7.6. Details für Experten	350
3.8. SQL-Präprozessor	352
3.8.1. Wie arbeitet der SQL-Präprozessor?	353
3.8.2. Bibliothek sqlparsing	353
3.8.3. Best Practice	356
3.8.4. Beispiele	357
3.9. Profiling	363
3.9.1. Was ist Profiling?	364
3.9.2. Aktivierung und Auswertung	364
3.9.3. Beispiel	365
3.10. Skyline	366
3.10.1. Motivation	367
3.10.2. So funktioniert Skyline	367
3.10.3. Beispiel	368
3.10.4. Syntaxelemente	369
4. Clients und Schnittstellen	371
4.1. EXAplus	371
4.1.1. Installation	371
4.1.2. Die grafische Benutzeroberfläche	372
4.1.3. Der Konsolenmodus	376
4.1.4. EXAplus-spezifische Befehle	379

4.2. ODBC-Treiber	396
4.2.1. Unterstützte Standards	397
4.2.2. Windows-Version des ODBC-Treibers	397
4.2.3. Linux/Unix-Version des ODBC-Treibers	400
4.2.4. Verbindungsauflaufbau in einer ODBC-Anwendung	401
4.2.5. Connection-String Parameter	402
4.2.6. Zeichensatz-Unterstützung	405
4.2.7. Best-Practice für Entwickler	405
4.3. JDBC-Treiber	405
4.3.1. Unterstützte Standards	406
4.3.2. Systemvoraussetzungen	406
4.3.3. Benutzung des Treibers	407
4.3.4. Best-Practice für Entwickler	411
4.4. ADO.NET Data Provider	411
4.4.1. Unterstützte Standards, Systemvoraussetzungen und Installation	412
4.4.2. Benutzung des Data Providers	413
4.4.3. Exasol Data Destination	417
4.4.4. Exasol Data Processing Extension	419
4.5. WebSockets	420
4.6. SDK	421
4.6.1. Call Level Interface (CLI)	421
A. Systemtabellen	425
A.1. Allgemeine Informationen	425
A.2. Liste der Systemtabellen	425
A.2.1. Katalog-Systemtabellen	425
A.2.2. Systemtabellen zu Metadaten	425
A.2.3. Statistische Systemtabellen	458
A.2.4. Oracle-kompatible Systemtabellen	478
B. Details zur Rechteverwaltung	481
B.1. Liste der System- bzw. Objektprivilegien	481
B.2. Benötigte Privilegien für SQL-Befehle	484
B.3. Systemtabellen zur Rechteverwaltung	488
C. Konformität zum SQL-Standard	491
C.1. SQL 2008 Standard Mandatory Features	491
C.2. SQL 2008 Standard Optional Features	496
D. Unterstützte Zeichensätze	497
E. Customer Service	499
Abkürzungsverzeichnis	501
Index	505

Tabellenverzeichnis

2.1. Sprachelemente in regulären Ausdrücken	9
2.2. Überblick über Exasol Datentypen	108
2.3. Zusammenfassung der Exasol Aliase	113
2.4. Mögliche implizite Konvertierungen	114
2.5. Elemente von Datum/Zeit Format-Modellen	131
2.6. Elemente numerischer Format-Modelle	133
2.7. Präzedenz von Prädikaten	138
3.1. Hinweise zu UDF Skripten	307
4.1. Informationen zur Arbeit mit EXAplus	374
4.2. Kommandozeilen-Parameter von EXAplus (nur für den Konsolenmodus)	377
4.3. Bekannte Probleme im Umgang mit dem ODBC-Treiber unter Windows	400
4.4. Bekannte Probleme im Umgang mit dem ODBC Treiber für Linux/Unix	401
4.5. Unterstützte DriverProperties des JDBC-Treibers	410
4.6. Schlüsselwörter in der ADO.NET Data Provider-Verbindungszeichenkette	414
B.1. Systemprivilegien in Exasol	482
B.2. Objektprivilegien in Exasol	484
B.3. Benötigte Privilegien zur Ausführung von SQL-Befehlen	485
C.1. SQL 2008 Mandatory Features	492
C.2. Von Exasol unterstützte SQL 2008 Optional Features	496

Vorwort

Das vorliegende Benutzerhandbuch liefert einen Überblick über Exasol und dokumentiert anwenderseitige Schnittstellen einschließlich des unterstützten **SQL**-Sprachumfangs. Weitere technische Informationen zu Exasol finden Sie in unserem [Online Solution Center](https://wwwexasolcom/portal/display/SOL) [<https://wwwexasolcom/portal/display/SOL>].

Wir sind stets bemüht, Ihnen einen möglichst hohen Qualitätsstandard zu gewährleisten. Exasol möchte Sie daher ganz herzlich einladen, an der Verbesserung der Qualität dieses Dokuments mitzuwirken.

Deshalb fragen wir Sie:

- Fehlen Ihnen Informationen?
- Haben Sie Fehler in der Dokumentation gefunden?
- Sind Abschnitte unklar formuliert oder zu kurz?
- Sind die Beispiele nicht aussagekräftig genug oder unzureichend?

Senden Sie uns Ihre Anregungen und Kommentare an untenstehende Adresse oder tragen Sie Verbesserungsvorschläge gleich online in unserem [IDEA-Projekt](https://wwwexasolcom/support/projects/IDEA/issues/) [<https://wwwexasolcom/support/projects/IDEA/issues/>] ein. Wir bedanken uns herzlich und werden uns bemühen, Ihre Vorschläge in der nächsten Version zu berücksichtigen.



Weitere Informationen zum Support finden Sie in unserem [Support Dashboard](https://wwwexasolcom/portal/display/EXA/Support+Dashboard) [<https://wwwexasolcom/portal/display/EXA/Support+Dashboard>]

Konventionen

Symbole

Folgende Symbole werden in diesem Benutzerhandbuch verwendet:

-
-  Anmerkung: z.B. "Bitte beachten Sie, dass innerhalb der Datenbank leere Zeichenketten ebenfalls als NULL interpretiert werden."
 -  Tip: z.B. "Wir empfehlen, stets lokale Variablen zu verwenden, um die Variablen-Deklaration explizit sichtbar zu machen."
 -  Wichtig: z.B. "Reguläre Bezeichner sind case-insensitiv."
 -  Achtung: z.B. "Dieser Befehl kann eine lange Ausführungszeit haben."
-

Änderungen in Version 6.0

Neue Features und Verbesserungen

- "EXASOL goes Open Source":
 - EXASOL hat sich entschieden, diverse Technologien im Zuge von Open Source Projekten zu veröffentlichen. Diese beinhalten vor allem die Themen Integration, Nutzung und analytische Erweiterungen unserer kommerziellen Datenbank. Letztlich haben wir mit diesem Konzept schon früher begonnen, indem Sie für die UDF-Skriptsprachen beliebige Open-Source Pakete im Cluster installieren können. Anstatt starr implementierter Verbesserungen ist unsere Vision, Ihnen regelrechte Frameworks zur Erweiterung der Funktionalität zu bieten. Die Vorteile hiervon sind eine agilere Weiterentwicklung der Features, und individuell auf Ihre Bedürfnisse anpassbare Features.
 - Unser Open-Source GitHub Repository finden Sie unter <https://www.github.com/EXASOL>. Wir würden uns sehr freuen, wenn auch Sie aktiv in unserer Open Source Community mitwirken und unsere Open Source Tools nutzen, erweitern und ergänzen. Und natürlich sind wir auch stets offen für ähnliche, neue Projekte. Lassen Sie uns die Zukunft von EXASOL gemeinsam formen.
 - Sie werden bereits in Version 6.0 eine Vielzahl von Produkt-Features entdecken, die Open-Source-Elemente beinhalten und die wir im weiteren Verlauf noch erläutern werden:
 - Adapter für externe Datenquellen zur Umsetzung des Konzepts der [virtuellen Schemas](#) [<https://github.com/EXASOL/virtual-schemas>]
 - [UDF Skripte](#) [<https://github.com/EXASOL/hadoop-ctl-udfs>] zur einfachen Daten-Beladung aus Hadoop Systemen unter Nutzung des `IMPORT FROM SCRIPT` Befehls (siehe auch weiter unten im Abschnitt ETL)
 - Native Treiber für unser neues, offenes Client/Server Protokoll ["JSON via WebSockets"](#) [<https://github.com/EXASOL/websocket-api>]
 - API-Spezifikation zur Integration neuer [Skript-Sprachen](#) [<https://github.com/EXASOL/script-languages>] sowie fertige Skript-Container (z.B. C++)
 - Eine beispielhafte Nutzung der [XMLRPC-Schnittstelle](#) [<https://github.com/EXASOL/exaoperation-xmlrpc>] zur automatischen Installation und Betrieb von EXASOL außerhalb der EXAoperation Web-Schnittstelle
 - Projekte zur Integration von EXASOL in diverse Abstraktions-Frameworks, wie z.B. SQLAlchemy, Django oder Teiid
 - Diverse SQL-Skripte zum [Migrieren von Daten](#) [<https://github.com/EXASOL/database-migration>] aus anderen Datenbanken (u.a. Oracle, DB2, MS SQLServer, Teradata, MySQL, Postgres, Redshift, Vertica), indem durch Scripting automatisch `IMPORT`-Kommandos erzeugt und ausgeführt werden
 - EXASOL stellt ein öffentliches Maven Repository zur Verfügung (<https://mavenexasol.com>). Dieses enthält den JDBC-Treiber sowie die Skript API, um Java Code für EXASOL-Skripte bequem in Ihrer Java IDE entwickeln zu können.
 - Ein [Nagios Monitoring System](#) [<https://github.com/EXASOL/nagios-monitoring>] innerhalb eines Docker Containers, welches innerhalb von Minuten installierbar ist. Dies ist entweder eine einfache Lösung für ein funktionierendes Monitoring Systems für Ihre EXASOL Datenbank, oder kann für Ihre existierende Monitoring-Software benutzt werden (durch Extraktion der Nagios Konfigurations-Dateien).
 - Sie finden weitergehende Dokumentation und Benutzungs-Tipps in den jeweiligen Open Source Projekten, aber ebenfalls in unserem Online Solution Center: <https://wwwexasol.com/portal/display/SOL>
- Produkt-Editionen und Features:
 - Anstatt einzelner zusätzlicher zu bezahlenden Features werden ab Version 6.0 zwei verschiedene Editionen von EXASOL angeboten. Im Gegensatz zur **Standard Edition** enthält die **Advanced Edition** folgende zusätzlichen Funktionalitäten:
 - Virtuelle Schemas (siehe nächster Punkt)
 - UDF Skripte (siehe [Abschnitt 3.6, UDF Skripte](#))
 - Geodaten (siehe [Abschnitt 2.4, Geodaten](#))
 - Skyline (siehe [Abschnitt 3.10, Skyline](#))

Ab sofort kostenfrei in der Standard Edition sind hingegen die Features Query Cache, LDAP Authentifizierung sowie die IMPORT/EXPORT Schnittstellen JDBC (JDBC-Datenquellen) und ORA (native Oracle-Schnittstelle).

Bei Fragen hierzu nehmen Sie bitte Kontakt mit Ihrem EXASOL Account Manager auf oder fragen unseren Support.

- Virtual Schemas bzw. virtuelle Schemas bilden ein mächtiges Werkzeug, um beliebige Datenquellen an EXASOL anbinden zu können. Virtuelle Schemas sind eine Art read-only Link zu einer externen Quelle und enthalten virtuelle Tabellen, die wie reguläre Tabellen aussehen, ohne lokal gespeichert zu sein. Details finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#).
- Über das neue synchrone Dateisystem BucketFS können Daten für den lokalen Zugriff aus UDF Skripten heraus repliziert im Cluster gespeichert werden. Details hierzu finden Sie unter [Abschnitt 3.6.4, Das synchrone Cluster-Dateisystem BucketFS](#).
- Das Skript-Framework von EXASOL wurde erweitert, so dass neue Skript-Sprachen auf dem EXASOL Cluster installierbar werden. Somit können Sie entweder mehrere Versionen einer Sprache (z.B. Python 2 und Python 3) nutzen, zusätzliche Bibliotheken selbst hinzufügen (auch für R), oder ganz neue Sprachen integrieren (Julia, C++, ...). Weitere Details finden Sie in [Abschnitt 3.6.5, Skript-Sprachen erweitern mittels BucketFS](#).
- In den Befehlen `CREATE SCHEMA`, `CREATE TABLE` und `ALTER TABLE (column)` wurde die Option `IF NOT EXISTS` eingeführt. Hierdurch können automatische DDL-Skripte ausgeführt werden, die keine Fehlermeldungen erzeugen, sofern die Objekte bereits existieren.
- `PRELOAD` lädt bestimmte Tabellen bzw. Spalten sowie die entsprechenden internen Indizes von der Festplatte in den Hauptspeicher der Datenbank, sofern diese nicht bereits im Cache vorliegen. Aufgrund des intelligenten Speicher-Managements von EXASOL empfehlen wir Nutzern allerdings, diesen Befehl lediglich für spezielle Ausnahmefälle zu benutzen. Ansonsten besteht die Gefahr einer schlechteren Gesamt-Performance.
- Wie bereits in UDF Skripten werden diverse Metadaten für die Scripting-Sprache eingeführt. Die Metadaten wurden zudem erweitert und enthalten nun auch das Schema des Skriptes sowie den aktuellen Nutzer. Zusätzliche Infos hierzu finden Sie in den entsprechenden Abschnitten in [Abschnitt 3.5, Scripting-Programmierung](#) und [Abschnitt 3.6, UDF Skripte](#).
- UDF Skripte können dynamische Eingabe- und Ausgabe-Parameter verarbeiten und werden hiermit nochmals viel flexibler in der Nutzung. Details siehe Abschnitt [Dynamische Eingabe- und Ausgabe-Parameter](#) in [Abschnitt 3.6, UDF Skripte](#).
- Bei skalaren UDF Skripten (Input-Typ SCALAR) können jetzt EMITS-Ergebnisse in der Select-Liste mit anderen Ausdrücken kombiniert werden, ohne Notwendigkeit eines separaten Subselects.
- Die Funktion `APPROXIMATE_COUNT_DISTINCT` berechnet eine Näherung für die Anzahl an disjunkten Elementen. Vorteil ist die hohe Beschleunigung im Vergleich zur exakten Funktion `COUNT`.
- Die Funktion `TO_CHAR (datetime)` erhält mit dem optionalen Parameter `NLS_DATE_LANGUAGE` die Möglichkeit, die Session-Einstellung im Aufruf zu überschreiben.
- `NVL2` wurde in Ergänzung zur Funktion `NVL` hinzugefügt und enthält im Vergleich hierzu einen zusätzlichen Parameter für die Ersetzung aller Werte, die nicht NULL sind.
- Die Bitfunktionen `BIT_LSHIFT`, `BIT_RSHIFT`, `BIT_LROTATE` und `BIT_RROTATE` wurden hinzugefügt. Zusätzlich wurden für die anderen Bitfunktionen der Wertebereich von 59 auf 64 Bit erweitert.
- Der neue Session/System-Parameter `TIMESTAMP_ARITHMETIC_BEHAVIOR` definiert das Verhalten der +/- Operatoren:
 - `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'INTERVAL'` - Die Differenz zweier Datumswerte liefert ein Interval, und beim Addieren einer Dezimalzahl wird diese zu einer Ganzzahl gerundet, so dass stets eine gewisse Zahl ganzer Tage addiert wird.
 - `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'DOUBLE'` - Die Differenz zweier Datumswerte liefert einen Double, und beim Addieren einer Dezimalzahl werden die Nachkommastellen als Anteil eines Tages umgerechnet (Stunden, Minuten, ...).
- Werden Prioritäten mittels `GRANT PRIORITY` verändert, so wird diese Änderung sofort aktiv. Bisher galten die veränderten Prioritäten erst für neue Transaktionen.
- Das Default-Limit für die erlaubte Anzahl an Datenbank-Objekten wurde von 50000 auf 250000 erhöht.
- ETL Prozesse:
 - Mit Hilfe von UDF-Skripten lassen sich Daten aus fast jeder beliebigen externen Datenquelle laden. Die neue Syntax `IMPORT FROM SCRIPT` macht es extrem einfach, Daten aus solchen Systemen mit einem

einfachen Befehl zu laden, sofern die entsprechenden UDF-Skripte entwickelt wurden. Weitere Details hierzu finden Sie in [Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs](#), und wir stellen bereits Open Source Skripte zur Verfügung (siehe <https://www.github.com/EXASOL>), die eine sehr leichte Integration von Hadoop Systeme in EXASOL ermöglichen. In naher Zukunft werden wir zusätzliche Open Source Skripte für weitere Produkte veröffentlichen.

- Die Befehle **IMPORT** und **EXPORT** unterstützen für JDBC-Datenquellen jetzt auch Kerberos Authentifizierung. Sie können die entsprechende Kerberos Konfiguration und die keytab Daten über die IDENTIFIED BY Klausel der Statements bzw. der benutzten Connection (siehe auch [CREATE CONNECTION](#)) definieren.
- Bitte beachten Sie, dass der Oracle Instant Client ab dieser Version manuell in EXAoperation installiert werden muss, sofern Sie die native Oracle-Schnittstelle für IMPORT/EXPORT Befehle nutzen wollen. Die vorinstallierten JDBC-Treiber für andere Datenbanken wurden aktualisiert.
- Performance Verbesserungen:
 - Der Verbindungsaufbau zur Datenbank wurde signifikant beschleunigt.
 - Metadaten-Abfragen von Treibern sind jetzt deutlich schneller.
 - Das Einfügen von kleinen Datenmengen mittels **INSERT** (z.B. nur eine einzige Zeile) wurde durch eine automatische, hybride Speicherung signifikant schneller. Die "letzten" Daten werden in zeilen-weisen Blöcken abgespeichert und dann im Hintergrund automatisch in die spaltenbasierte Speicherung überführt. Der Nutzer bekommt hiervon nichts mit, außer dass das Einfügen kleiner Mengen hierdurch deutlich weniger Daten-Blöcke auf Festplatte übertragen muss.
 - In Szenarien mit vielen parallelen Schreib-Transaktionen wurde der gesamte Systemdurchsatz verbessert.
 - Die Optimierung von innerhalb eines **SELECT**-Statements mehrfach genutzten Views bzw. **WITH** Klauseln wurde deutlich verbessert. Hierdurch wird in mehr Fällen als bisher die bessere Alternative gewählt, ob solch eine View (bzw. **WITH** Klausel) vor-materialisiert wird oder nicht. Insgesamt sollte sich daher die Performance in solchen Situationen verbessern, auch wenn aufgrund der Komplexität der Optimizer nicht immer richtig liegen kann. Kunden, die die View-Optimierung explizit mittels dem Datenbank-Parameter `-disableviewoptimization` abgeschalten haben, könnten jetzt versuchen, ob dies nicht mehr nötig ist.
 - Durch die Optimierung der internen Datenübertragung für verteilte Tabellen-Zeilen erreichen mehrere Operationen eine höhere Performance (u.a. **MERGE**, **IMPORT** und eine Cluster-Vergroßerung).
 - Der **MERGE** Befehl wurde für einige Situationen beschleunigt, u.a. falls die Zieltabelle sehr groß ist oder die Daten der Ziel- und Quell-Tabelle nicht nach den Spalten der ON-Bedingung verteilt sind (**DISTRIBUTED BY**).
 - Werden Daten einer Tabelle gelöscht, so werden die betroffenen Zeilen intern zunächst nur als gelöscht markiert. Erst nach einem bestimmten Schwellwert werden diese Daten endgültig gelöscht und die Tabelle reorganisiert. Diese Reorganisation wurde durch Parallelisierung erheblich beschleunigt.
 - Durch einen intelligenteren Replikations-Mechanismus von kleinen Tabellen werden diese nicht mehr persistent repliziert gehalten, sondern inkrementell im Cache gehalten. Hierdurch beschleunigen sich **INSERT**-Operationen auf kleinen Tabellen signifikant.
 - GROUP BY Berechnungen mit sehr wenigen Gruppen können in gewissen Fällen schneller werden, weil der Verteilungs-Mechanismus der Berechnung auf die Cluster-Knoten verbessert wurde.
 - Die Berechnung von BETWEEN Filtern auf DATE-Spalten nutzte bereits existierende interne Index-Strukturen zur Beschleunigung der Ausführung. Diese Fähigkeit wurde auf die Datentypen **TIMESTAMP**, **DECIMAL**, **DOUBLE** und **INTERVAL** erweitert.
 - Ein smarter Rekompressions-Mechanismus wurde implementiert, so dass nur diejenigen Tabellen und Spalten komprimiert werden, für die eine substantielle Verbesserung zu erwarten ist. Dies wird automatisch während der Ausführung ermittelt und führt ebenfalls zu einer Beschleunigung des **DELETE** Befehls, der ab einem bestimmten Schwellwert an gelöschten Zeilen eine interne Rekompression auslöst. Sie können eine komplette Rekompression jedoch immer noch mit der neuen Option **ENFORCE** des **RECOMPRESS** Befehls ausführen. Zudem ist es jetzt möglich, nur bestimmte Spalten einer Tabelle für das **RECOMPRESS** Statement zu definieren und somit nur selektiv bestimmte Spalten neu zu komprimieren.
 - Anstatt bei einem **ALTER TABLE ADD COLUMN** sämtliche Constraints der Tabelle zu überprüfen, werden jetzt nur noch die Constraints für neu hinzugefügten Spalten geprüft.
 - Das Erstellen und Wiederherstellen von lokalen Backups wurde erheblich beschleunigt. In internen Tests erreichten wir für lokale Backups eine Beschleunigung um Faktor 4-5 und für das Restore Faktor 2-3. Bitte beachten Sie jedoch, dass der entsprechende Faktor auf Kundensystemen stark von der konkreten Hardware-Konfiguration abhängt.
 - Für neu (!) erzeugte EXAStorage Volumes wurde die Lese-Performance aufgrund eines optimierten Speicher-Layouts verbessert. Einerseits wurden Lese-Operationen auf großen und stark fragmentierten Volumes

verbessert. Andererseits steigt die Performance für Szenarien, bei denen der Hauptspeicher viel zu gering dimensioniert ist, also ständig beliebige Daten von diversen Tabellen nachgeladen werden müssen.

- Intelligente Recovery-Mechanismen reduzieren die Datenmenge signifikant, die von EXAStorage z.B. nach einem Knotenausfall wiederhergestellt werden müssen.
- Der (Re-)Start einer Datenbank wurde beschleunigt, besonders für größere Cluster.
- System-Informationen:
 - In der Systemtabelle **EXA_VOLUME_USAGE** werden detaillierte Informationen zur Nutzung der verschiedenen Storage-Volumes zur Verfügung gestellt.
 - Die DURATION Spalte wurde in die EXA_DB_Audit_SQL und EXA_SQL_LAST_DAY System Tabellen eingefügt. Sie enthält die Laufzeit des Statements in Sekunden.
 - Die Anzahl an Zeilen einer Tabellen wird in der Spalte TABLE_ROW_COUNT der Systemtabellen **EXA_ALL_TABLES**, **EXA_DB_TABLES** und **EXA_USER_TABLES** angezeigt.
- Schnittstellen:
 - Bitte beachten Sie, dass ab Version 6.0 der Support für AIX, HP-UX und Solaris eingestellt wird. Weitere Details zu diesem Thema finden Sie unter: <https://wwwexasol.com/portal/display/DOWNLOAD/EXASolution+Life+Cycle>
 - Ein verbindungsorientiertes JSON via WebSockets Protokoll wurde implementiert, mit dem es möglich ist, EXASOL in nahezu jede beliebige Programmiersprache Ihrer Wahl zu integrieren. Ein weiterer Vorteil ist die Unterstützung einer für Client/Server-Kompression. Anfangs stellen wir einen Open Source EXASOL Python Treiber zur Verfügung, dem aber sicher noch weitere Implementierungen folgen werden. Weitere Details finden Sie unter [Abschnitt 4.5, WebSockets](#).
 - Im Falle einer schwerwiegenden Systemüberlast gibt es für den Nutzer SYS die Möglichkeit, das System zu analysieren und die verantwortlichen Prozesse zu beenden. Hierfür muss der Verbindungs-Parameter SUPERCONNECTION (ODBC) bzw. superconnection (JDBC und ADO.NET) gesetzt werden.
 - Der Parameter CONNECTTIMEOUT (ODBC) bzw. connecttimeout (JDBC und ADO.NET) definieren die maximale Zeit in Millisekunden (Default: 2000), die der Treiber für den Aufbau einer TPC-Verbindung zu einem Server wartet. Dieser Timeout ist gerade für größere Cluster mit mehreren Ersatzservern interessant, um die Login-Dauer zu minimieren.
 - Verschlüsselte Client/Server Verbindungen sind ab sofort defaultmäßig aktiviert und basieren auf dem Algorithmus ChaCha20 (RFC 7539).
 - Die Bestimmung der Datentypen von Prepared Parametern im PREPARE-Schritt wurde verbessert. Bisher wurde hier stets der generische Typ VARCHAR(2000000) benutzt.
 - ODBC-Treiber
 - Falls in den Verbindungsparametern kein Schema angegeben wurde, so wird ab sofort nicht mehr versucht, ein Schema mit dem Nutzernamen zu öffnen. Dieses Verhalten wurde aus historischen Gründen implementiert, aber jetzt aus Konsistenzgründen wieder entfernt.
 - JDBC
 - Der JDBC Treiber unterstützt jetzt die java.sql.Driver Datei. Diese Option wurde im JDBC 4.0 Standard hinzugefügt und erlaubt Java Applikationen, JDBC Treiber zu verwenden, ohne sie explizit mittels Class.forName oder DriverManager.registerDriver laden zu müssen. Jetzt reicht es aus, den Treiber in den Klassen-Pfad einzutragen.
 - Bei Ergebnissen einer Query werden die Datentypen besser ermittelt. Im Detail bedeutet dies, dass die Funktion getColumnType() der Klasse ResultSetMetadata für Dezimal-Typen ohne Nachkommastellen den kleinstmöglichen Integer-Typ zurückliefert (smallint, int bzw. bigint).
 - Der JDBC Treiber kann mittels sogenannten Subconnections parallel von mehreren Cluster-Knoten Lesen und Schreiben. Details und Beispiele hierzu finden Sie in unserem Solution Center: <https://wwwexasol.com/support/browse/SOL-546>
 - Die Klasse EXAPreparedStatement enthält die neue Methode setFixedChar(), die den Parameter-Typ auf CHAR setzt. Da die Standard-Methode setString() hierfür VARCHAR verwendet, ist diese neue Funktion nützlich bei Vergleichen mit CHAR Werten.
 - ADO.NET driver
 - Der Parameter querytimeout definiert, wie lange eine Query maximal in Sekunden dauern darf, bevor sie automatisch abgebrochen wird.
 - Die DDEX Provider und Pluggable SQL Cartridge für SQL Server Analysis Services unterstützt den SQL Server 2016 (v 13.0).
 - EXAplus
 - Die neuen grafischen Systemnutzungs-Statistiken werden geöffnet, indem Sie das neue Kuchendiagramm-Symbol auf der Werkzeugeiste anklicken.

- Der Balken in der rechten unteren Ecke zeigt den Speicher-Allokation des aktuellen EXAplus-Prozesses. Durch ein Doppelklick wird das Programm die Garbage Collection starten.
- Zusätzliche JDBC Parameter können über den Parameter - jdbcp param oder den erweiterten Einstellungen eines Verbindungsprofils definiert werden.
- Die Konsolen-Version hat neue Parameter zum Verwalten von Nutzer-Profilen (-lp zum Anzeigen existierender Profile, -dp <profile> zum Löschen eines Profiles und -wp <profile> zum Schreiben bestimmter Optionen in ein Profil)
- **SDK**
 - Bisher gab es zwei Varianten der SDK Bibliothek: libexacli und libexacli_c. Die erste Bibliothek enthielt Abhängigkeiten zu C++, weshalb auf Kundenanfrage die zweite Bibliothek in Version 5 hinzugefügt wurde ohne Abhängigkeiten zu C++. Nun haben wir die C++ Abhängigkeiten in der libexacli eliminiert und liefern nur noch diese Variante aus.
- **Betrieb:**
 - Ausführliche Informationen zum Update auf Version 6 finden Sie in folgendem Beitrag unseres Solution Centers: <https://wwwexasol.com/support/browse/SOL-504>



Ein Downgrade auf Version 5 nach der Installation von Version 6 funktioniert nur über eine Neuinstallation und ein Restore eines entsprechenden Backups.



Beim Upgrade auf 6.0 empfehlen wir aus Performance-Gründen, die Datenbank-Volumes in EXAStorage neu anzulegen, auch wenn dies zu einer längeren Downtime führt. Werden die Volumes nicht neu angelegt, dann wird bei jedem Datenbank-Start eine Warning ausgegeben, dass noch das alte EXAStorage-Format benutzt wird.

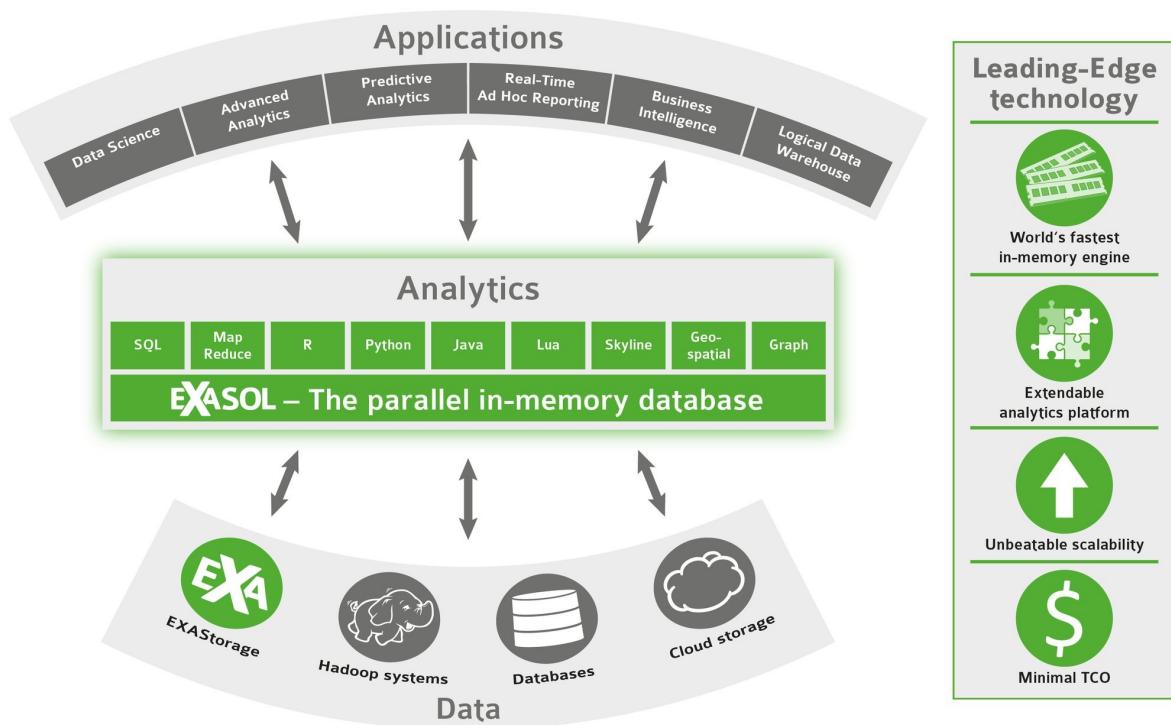
- In EXAStorage wird es jetzt ermöglicht, Disks ohne die Neuinstallation eines Knotens hinzuzufügen.
- EXAStorage Volumes können jetzt den Status DEGRADED haben, was auf ausgefallene Disks hinweist. Bisher war in solchen Fällen der Status ONLINE, solange die Daten weiterhin zur Verfügung stehen (z.B. aufgrund von RAID-Einstellungen).
- Persistente Daten-Volumes können explizit verkleinert werden (Shrink). Diese nicht-blockierende Operation kann über EXAoperation gestartet werden, wobei entweder eine Zielgröße angegeben werden kann, oder die Operation das Volume so viel wie möglich verkleinert.
- Die XMLRPC Administration-Schnittstelle wurde erweitert, so dass nun die gesamte Funktionalität von EXAoperation zur Verfügung steht.
- Es ist jetzt möglich, WebHDFS als Backend für Remote Archive Volumes zu nutzen und dort Backups abzulegen. Eine Kerberos Authentifizierung ist möglich, indem unter den Optionen des Remote Volumes ein so genanntes Delegation Token spezifiziert wird.
- Backups auf Remote Archive Volumes werden von nun an automatisch gelöscht, sobald Ihre Expiration Time abgelaufen ist. Dieser Aufräum-Mechanismus wird nach jedem Backup-Lauf gestartet und war bisher nur für lokale Backups umgesetzt.
- Bisher werden für die Kommunikation mit Syslog-Servern die Protokolle UDP und TCP unterstützt. Ab sofort wird auch verschlüsselte TLS-basierte Kommunikation unterstützt (wie in RFC 5425 definiert).
- Wird ein Syslog Server in EXAoperation definiert, so wird ab sofort die Priorität einer Nachricht in das Tag-Feld hinzugefügt. Da Syslog dies bei der Ausgabe als Präfix für die eigentliche Nachricht verwendet, lassen sich Meldungen somit leichter klassifizieren. Der neue Präfix ist jetzt "EXAoperation [Priority], [Service]" anstatt wie vorher "EXAoperation, [Service]".
- Der IP-Bereich für das interne, private Cluster-Netzwerk kann jetzt in Grenzen konfiguriert werden. Details hierzu finden Sie im Roadmap-Issue [EXASOL-1750](https://wwwexasol.com/support/browse/EXASOL-1750) [<https://wwwexasol.com/support/browse/EXASOL-1750>].
- Der Lizenzserver konnte bisher entweder über ein DVD Image oder halbautomatisch über Netzwerk installiert werden. Nun geht die zweite Variante auch voll-automatisch. Zusätzlich kann EXASOL auch über ein USB Gerät installiert werden (siehe hierzu auch [EXASOL-1792](https://wwwexasol.com/support/browse/EXASOL-1792) [<https://wwwexasol.com/support/browse/EXASOL-1792>])
- Der Zugriff auf den Support-Bereich in EXAoperation, in dem man Debug-Informationen herunterladen kann, kann jetzt über Nutzer-Privilegien gesteuert werden. Bisher konnten dies nur Nutzer mit den Rollen Administrator oder Master machen. Bitte beachten Sie, dass solche Log-Dateien auch sensible, kontext-spezifische Informationen wie SQL Statements enthalten können.

- Im Monitor-Bereich unter EXAoperation wurde ein Konfigurations-Feld hinzugefügt, mit dem man definieren kann, wie lange im Default-Fall Datenbank Log-Dateien aufgehoben werden.
- Der Zugriff zum Support-Bereich in EXAoperation kann jetzt mittels Rollen vergeben werden. Bisher konnte nur ein Administrator darauf zugreifen.
- Es ist möglich, eigene TLS Zertifikate über EXAoperation auf dem Cluster hochzuladen.



Exasol ist die erste Wahl für alle analytischen Herausforderungen der heutigen Unternehmenswelt. Dabei hebt Exasol Analysen auf ein vollkommen neues Level, ganz egal, ob es sich um unternehmenskritische operative Datenanwendungen, interaktive Customer Analytics oder (Logical) Data Warehousing handelt.

Mit der In-Memory-Plattform von Exasol optimieren Unternehmen durch die Analyse großer Datenmengen ihr Tagesgeschäft. Sie können große Mengen historischer Daten ohne Performanceverluste verarbeiten, Analysen präziser durchführen, in Echtzeit reagieren und so gleichzeitig die User Experience steigern sowie die Zahl ihrer zufriedenen Nutzer und Kunden erhöhen.



Die direkte Integration von R, Python, Java und Lua (siehe [Abschnitt 3.6, UDF Skripte](#)) und die Option, SQL Queries mit MapReduce-Jobs direkt in der Datenbank zu kombinieren, bieten ein sehr mächtiges und flexibles In-Database-Analytics Werkzeug. Graph-Analysefähigkeiten (siehe CONNECT BY in der Beschreibung zum [SELECT](#)-Statement), unsere Skyline-Erweiterung (siehe [Abschnitt 3.10, Skyline](#)) sowie der Support von Geodaten-Auswertungen (siehe [Abschnitt 2.4, Geodaten](#)) vervollständigen das Analyse-Spektrum.

Exasol vereint die Vorteile von In-Memory- Technologie, Spaltenorientierung sowie massiv paralleler Verarbeitung und ermöglicht eine verbesserte Performance, große Flexibilität und eine hohe Skalierbarkeit. Exasol kann reibungslos in jede IT-Infrastruktur und in alle heterogenen analytischen Plattformen integriert werden.

Bitte beachten Sie, dass Exasol in zwei unterschiedlichen Editionen verfügbar ist. Im Gegensatz zur **Standard Edition** enthält die **Advanced Edition** folgende zusätzlichen Funktionalitäten:

- Virtuelle Schemas (siehe [Abschnitt 3.7, Virtuelle Schemas](#))
- UDF Skripte (siehe [Abschnitt 3.6, UDF Skripte](#))
- Geodaten (siehe [Abschnitt 2.4, Geodaten](#))
- Skyline (siehe [Abschnitt 3.10, Skyline](#))

Interfaces

Es werden folgende standardisierte Schnittstellen für Exasol zur Verfügung gestellt: [ODBC](#), [JDBC](#) und [ADO.NET](#). Zusätzlich gibt es die Möglichkeit, mit Hilfe des OLE DB Provider for ODBC Drivers™ von Microsoft über eine [OLE DB](#)-Schnittstelle auf Exasol zuzugreifen.

Das [JSON](#) via WebSockets Protokoll erlaubt es Ihnen, Exasol in nahezu jede beliebige Programmiersprache Ihrer Wahl zu integrieren. Entsprechend darauf aufbauende Pakete wie zum Beispiel ein nativer Python-Treiber stehen als Open Source Projekte zur Verfügung. Und um Exasol speziell in C/C++Anwendungen einbinden zu können, stellt Exasol zusätzlich ein Client [SDK](#) (Software Development Kit) zur Verfügung.

Durch die Treiber-Anbindung ist es möglich, SQL-Anfragen zu stellen und in der angebundenen Applikation die Ergebnisse darzustellen, sowie Daten in Exasol zu laden. Für größere Ladeoperationen wird allerdings aus Performancegründen der [IMPORT](#)-Befehl empfohlen.

Schnittstelle	Kapitel im Handbuch
ODBC-Treiber	Abschnitt 4.2, ODBC-Treiber
JDBC-Treiber	Abschnitt 4.3, JDBC-Treiber
ADO.NET Data Provider	Abschnitt 4.4, ADO.NET Data Provider
JSON via WebSockets	Abschnitt 4.5, WebSockets
Client SDK	Abschnitt 4.6, SDK

EXAplus

EXAplus ist eine Konsole zur Eingabe von [SQL](#)-Befehlen. Mit EXAplus ist es möglich, auf einfache Weise SQL-Skripte auszuführen oder administrative Aufgaben zu erledigen. EXAplus wurde in Java implementiert, ist daher plattformunabhängig und kann sowohl als grafische Benutzeroberfläche als auch als reine Textkonsole benutzt werden.

Eine ausführliche Dokumentation ist in [Abschnitt 4.1, EXAplus](#) zu finden.

1. Was ist Exasol?

The screenshot shows the EXAplus interface. On the left is a navigation sidebar with tabs for Datenbankbrowser, Chronik, Favoriten, and Vorlagen. The main area has a title bar "EXAplus - GUEST@tsdb [bar] (on ws64-2)". Below the title bar is a toolbar with various icons. The central part of the screen contains a SQL Editor window titled "* SQL-Editor 1". The SQL code in the editor is:

```

1 SELECT
2   l_orderkey,
3   SUM(l_expectedprice * (1 - l_discount)) as revenue,
4
5   FROM
6   SUM(expr)
7   WHERE
8     Liefer die Summe.
9
10  Beispiel(e)
11  GRN
12  SELECT SUM(salary) SUM FROM staff WHERE age b
13
14  SUM
15  ORDER BY
16    revenue DESC,
17    o_orderdate
18  LIMIT 10
19 ;

```

A tooltip "Beispiel(e)" is shown over the "GRN" line. Below the editor is a results table with columns ID and TESTID, containing several rows of data. At the bottom of the screen are status bars for Log (10 Zeilen / 0.1 s), Off, Tabellen (ersetzen), memory usage (1.000 MB, 19:2, 235/455 MB), and a progress bar.

2. SQL-Referenz

Diese [SQL](#)-Referenz dokumentiert den Großteil des von Exasol unterstützten SQL-Sprachumfangs. Einige Features sind hier noch nicht dokumentiert, werden aber in den nächsten Versionen ergänzt.

Eine detaillierte Übersicht der unterstützten Features des SQL-Standards ist in [Anhang C, Konformität zum SQL-Standard](#) zu finden.

2.1. Grundlegende Sprachelemente

2.1.1. Kommentare in SQL

Kommentare dienen in erster Linie der Übersichtlichkeit und Wartbarkeit von SQL-Skripten. Sie können vom Anwender an beliebiger Stelle im SQL-Skript eingefügt werden.

Es gibt zwei verschiedene Arten von Kommentaren in Exasol:

- Zeilenkommentare beginnen mit den Zeichen `--` und markieren den restlichen Teil der aktuellen Zeile als Kommentar.
- Blockkommentare sind durch die Zeichen `/*` bzw. `*/` markiert und können auch über mehrere Zeilen reichen. Alle Zeichen zwischen den Trennsymbolen werden dabei ignoriert.

Beispiele

```
-- Dies ist ein Kommentar
SELECT * FROM dual;

-- Zeige die Anzahl an Zeilen an
SELECT count(*) FROM dual;

SELECT avg(salary)      -- Durchschnittsgehalt
FROM staff                -- aus Tabelle staff
WHERE age>50;            -- Einschränkung auf über 50-jährige

/*
 Zwischen den beiden Trennsymbolen können beliebig
 viele Kommentarzeilen eingefügt werden
*/
SELECT * FROM dual;

SELECT /*Dieser Teil wird nicht ausgewertet!*/ * FROM dual;
```

2.1.2. SQL-Bezeichner

Jedes Datenbankobjekt besitzt einen eindeutigen Namen. Innerhalb eines SQL-Befehls werden diese Namen über SQL-Bezeichner referenziert. Zu den Datenbankobjekten gehören Schemas, Tabellen, Views, Spalten, Funktionen, User und Rollen. SQL-Bezeichner dürfen in Exasol höchstens 128 Zeichen lang sein. Man unterscheidet zwischen regulären Bezeichnern und Bezeichnern in Anführungszeichen. Bezeichner werden häufig auch als Identifier bezeichnet.

Reguläre Bezeichner

Reguläre Bezeichner werden ohne Anführungszeichen angegeben. In ihnen ist nur eine Teilmenge aller Unicode-Zeichen erlaubt, wobei das Anfangszeichen größeren Beschränkungen unterliegt als die weiteren Zeichen. Entsprechend dem SQL-Standard können in Exasol für Anfangs- bzw. Fortsetzungszeichen die folgenden Zeichengruppen verwendet werden (welche Zeichen genau in diese Gruppen fallen, ist im Unicode-Standard festgelegt):

- **Für das Anfangszeichen:** Alle Zeichen aus den Unicode Zeichengruppen Lu (upper-case letters, Großbuchstaben), Ll (lower-case letters, Kleinbuchstaben), Lt (title-case letters), Lm (modifier letters), Lo (other letters) und Nl (letter numbers).
- **Für Fortsetzungszeichen:** Alle Anfangszeichen und zusätzlich alle Zeichen aus den Unicode Zeichengruppen Mn (nonspacing marks), Mc (spacing combining marks), Nd (decimal numbers, Dezimalzahlen), Pc (connector punctuations) und Cf (formatting codes) sowie das Unicode-Zeichen U+00B7 (Mittlerer Punkt).

Für den einfachen ASCII-Zeichensatz bedeuten diese Punkte beispielsweise, dass Bezeichner mit einem Zeichen aus der Menge {a-z,A-Z} beginnen müssen und folgende Zeichen aus der Menge {a-z,A-Z,0-9,_} stammen dürfen.

Weiterhin existiert die Einschränkung, dass keine reservierten Wörter (siehe weiter unten) als reguläre Bezeichner benutzt werden dürfen.



Falls Sie Zeichen verwenden wollen, die für reguläre Bezeichner nicht erlaubt sind, können Sie begrenzte Bezeichner benutzen (siehe nächsten Abschnitt). Soll beispielsweise das Wort `table` als Bezeichner verwendet werden, muss es mit Anführungszeichen geschrieben werden ("`table`"), weil es ein reserviertes Wort ist.

Reguläre Bezeichner werden in der Datenbank immer in Großbuchstaben gespeichert. Sie sind *case-insensitiv*. Daher sind beispielsweise die beiden Bezeichner ABC und aBc äquivalent.



Reguläre Bezeichner werden in Großbuchstaben gespeichert.

Beispiele

Bezeichner	Name in der DB
ABC	ABC
aBc	ABC
a123	A123

Begrenzte Bezeichner

Diese Bezeichner sind in doppelten Anführungszeichen eingeschlossene Namen. Innerhalb der Anführungszeichen können beliebige Zeichen stehen außer dem Punkt (' . '). Nur bei Benutzern und Rollen ist der Punkt erlaubt, weil damit z.B. Email-Adressen verwendet werden können. Dies ist möglich, weil Benutzer und Rollen in keinem Konflikt zu schemaqualifizierten Objekten stehen können (siehe nächster Abschnitt).

Eine Besonderheit bildet die Verwendung doppelter Anführungszeichen: Will ein Benutzer innerhalb des Namens dieses Zeichen verwenden, so muss er zwei doppelte Anführungszeichen hintereinander schreiben (also bedeutet "ab" "c" den Namen ab"c").

Bezeichner in Anführungszeichen werden grundsätzlich *case-sensitiv* in der Datenbank gespeichert, mit der Ausnahme von Benutzern und Rollen.



Begrenzte Bezeichner sind case-sensitiv (außer bei Benutzern und Rollen).

Beispiele

Bezeichner	Name in der DB
"ABC"	ABC
"abc"	abc
"_x_"	_x_
"ab" "c"	ab"c

Schemaqualifizierte Namen

Werden Datenbankobjekte über schemaqualifizierte Namen angesprochen, so werden die Namen mit einem Punkt getrennt. Dies ermöglicht den Zugriff auf Schemaobjekte, die sich nicht im aktuellen Schema befinden.

Beispiele

```
SELECT my_column FROM my_schema.my_table;
SELECT "my_column" FROM "my_schema"."my_table";
SELECT my_schema.my_function(my_column) from my_schema.my_table;
```

Reservierte Wörter

Es existieren eine Reihe von reservierten Wörtern, die nicht als reguläre Bezeichner verwendet werden dürfen. Beispielsweise ist das Keyword SELECT solch ein reserviertes Wort. Soll eine Tabelle mit dem Namen SELECT angelegt werden, so ist dies nur möglich, wenn der Name in doppelte Anführungszeichen gesetzt wird. "SELECT" als Tabellenname ist daher wohlunterschieden von Tabellennamen wie "Select" oder "seLect".

Die Liste der in Exasol reservierten Wörter kann in der Systemtabelle [EXA_SQL_KEYWORDS](#) eingesehen werden.

2.1.3. Reguläre Ausdrücke

Reguläre Ausdrücke dienen der Filterung oder der Ersetzung von Zeichenketten nach bestimmten Mustern (Pattern Matching). Exasol unterstützt den **PCRE** Dialekt, welcher eine sehr mächtige Syntax umfasst und über alternative Dialekte wie **POSIX BRE** oder **POSIX ERE** deutlich hinausgeht.

Dieses Kapitel soll die grundlegenden Möglichkeiten von regulären Ausdrücken in Exasol erläutern. Detailliertere Informationen zum **PCRE** Dialekt finden Sie unter www.pcre.org [<http://www.pcre.org>].

Reguläre Ausdrücke können in folgenden skalaren Funktionen und Prädikaten eingesetzt werden:

- [REGEXP_INSTR](#)
- [REGEXP_REPLACE](#)
- [REGEXP_SUBSTR](#)
- [\[NOT\] REGEXP_LIKE](#)

Beispiele

In der Beschreibung zu den entsprechenden skalaren Funktionen und Prädikaten finden Sie Beispiele zur Verwendung von regulären Ausdrücken in SQL Befehlen. Die folgenden Beispiele sollen die generellen Möglichkeiten von regulären Ausdrücken aufzeigen:

Bedeutung	Regulärer Ausdruck ^a
American Express Kreditkartennummern (haben 15 Ziffern und starten mit 34 oder 37)	3[47][0-9]{13}
IP-Adressen wie z.B. 192.168.0.1	\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}
Fließkommazahlen wie z.B. -1.23E-6 oder 40	[+-]?[0-9]*\.[0-9]+([+-]?[eE][0-9]+)?
Datumswerte im Format YYYY-MM-DD wie z.B. 2010-12-01 (eingeschränkt auf 1900-2099)	(19 20)\d\d-(0[1-9] 1[012])-(0[1-9] 1[2][0-9] 3[01])
Email-Adressen wie z.B. hello.world@yahoo.com	(?i)[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}

^aBitte beachten Sie, dass die Beispiel-Ausdrücke zur besseren Anschaulichkeit simple gewählt wurden. Produktive Ausdrücke zur Identifikation von z.B. beliebigen gültigen Email-Adressen sind deutlich komplexer.

Sprachelemente

Folgende Elemente können in regulären Ausdrücken verwendet werden:

Tabelle 2.1. Sprachelemente in regulären Ausdrücken

Element	Bedeutung im regulären Ausdruck
Allgemeine Elemente	
.	Beliebiges einzelnes Zeichen außer Zeilenumbruch - bei Modifikator (?s) auch Zeilenumbruch
^	String-Anfang - bei Modifikator (?m) auch Zeilen-Anfang (also nach einem Zeilenumbruch)
\$	String-Ende - bei Modifikator (?m) auch Zeilen-Ende (also vor einem Zeilenumbruch)
(?#)	Kommentar, der nicht ausgewertet wird (z.B. (?#This is a comment))
Quantoren und Operatoren	
{}	Minimum/Maximum Quantor (z.B. {1,5} für 1-5 Vorkommen oder {3} für exakt drei Vorkommen oder {,7} für maximal 7 Vorkommen)
{ }?	wie {}, aber non-greedy (d.h. minimale Anzahl an Vorkommen wird benutzt)
?	0 oder 1 Quantor (<i>höchstens 1 Mal</i>) - äquivalent zu {0,1}
*	0 oder mehr Quantor (<i>beliebig oft</i>) - äquivalent zu {0,}
*?	Äquivalent wie *, aber non-greedy (d.h. minimale Anzahl an Vorkommen wird benutzt)
+	1 oder mehr Quantor (<i>mindestens 1 Mal</i>) - äquivalent zu {1,}
+?	Äquivalent wie +, aber non-greedy (d.h. minimale Anzahl an Vorkommen wird benutzt)
	Alternativ-Operator (z.B. a b c für 'a', 'b' oder 'c'). Bitte beachten Sie, dass die erste passende Alternative verwendet wird. Daher kann es bei bestimmten Ausdrücken auf die Reihenfolge der Alternativen ankommen.
()	Definition eines Teilmusters (zur Gruppierung, z.B. (abc)+ für mindestens 1 Mal das Wort 'abc'). Außerdem werden hierdurch Captures definiert (siehe unten).
(?({=pattern1})pattern2 pattern3)	Abhängig, ob das Pattern in den ersten Klammern gefunden wird, wird das weitere Pattern bestimmt (z.B. (?({=[0-9])[0-9]* [a-z]*) sucht nur nach Buchstaben von a-z, falls das erste Zeichen keine Ziffer ist).
Zeichenklassen mittels []	
[]	Definition einer Zeichenklasse (z.B. [abc] für 'a', 'b', oder 'c')
\	Escape Zeichen innerhalb von Zeichenklassen (z.B. [a\\] für 'a' oder '\\')
-	Definiert einen Bereich (z.B. [0-9] für eine beliebige Ziffer)
^	Negiert eine Zeichenklasse (z.B. [^0-9] für alles außer Ziffern)
Verwendung des Backslash (\)	
\	Escape Zeichen, um Sonderzeichen verwenden zu können (z.B. \+ für das Zeichen '+')
\R	Beliebiger Zeilenumbruch (Newline oder Carriage Return)
\n	Newline
\r	Carriage Return
\f	Formfeed
\t	Tab
\e	Escape
\d	Beliebige Ziffer (0-9)
\D	Alles außer einer Ziffer
\w	Beliebiges Wortzeichen (Alphanumerische Zeichen sowie _)
\W	Alles außer einem Wortzeichen
\s	Beliebiges Leerzeichen (Whitespace)
\S	Alles außer einem Leerzeichen

Element	Bedeutung im regulären Ausdruck
Modifikatoren (stellen Optionen ab der aktuellen Position ein/aus)	
(?i)	Case-insensitiv (Groß-/Kleinschreibung egal)
(?m)	^ und \$ beachten auch Zeilenumbrüche
(?s)	. erkennt auch Zeilenumbrüche
(?x)	Ignoriere Leerzeichen komplett
(?U)	Non-greedy Suche (versucht, den kürzest möglichen Match zu finden)
(?X)	Liefert Fehlermeldung, wenn innerhalb eines Pattern nach einem Backslash kein Sonderzeichen angegeben ist. Z.B. ist \a nicht erlaubt. Ohne diese Option wird \a als 'a' interpretiert.
(?ims)	Einschalten mehrerer Modifikatoren (hier werden i, m und s eingeschaltet)
(?i-ms)	Ausschalten von Modifikatoren (i wird eingeschaltet, m und s ausgeschaltet)
(*CR)	Mit jedem dieser drei Modifikatoren lassen sich Zeilenumbruchzeichen definieren, die von den Sonderzeichen ^, \$ und . ausgewertet werden.
(*LF)	(*CR) Nur Carriage Return (\r) wird als Zeilenumbruch ausgewertet
(*CRLF)	(*LF) Nur Newline (\n) wird als Zeilenumbruch ausgewertet (*CRLF) Beide Alternativen werden als Zeilenumbruch ausgewertet
	Beispiel: Das Pattern (*LF) (?ms)^abc . def\$ würde auf den String 'abc\ndef' passen.
Captures	
\1	Back-References auf das erste bis neunte Capture.
\2	Die Captures werden von links nach rechts durch die runden Klammern definiert, die Nummerierung durch die Position der öffnenden Klammer. Wollen Sie mehr als 9 Captures benutzen, so müssen Sie Namen dafür vergeben (siehe nächster Punkt).
...	
\9	Beispiel: Das Pattern (a(b c))([0-9]*) auf den String 'ab123' liefert die folgenden Captures: \1 ab \2 b \3 123
\0	Dieses spezielle Capture referenziert stets den kompletten String.
(?P<name>)	Durch diese Notation kann ein Capture auch einen Namen erhalten, was zur Übersichtlichkeit beiträgt und die Verwendung von mehr als 9 Captures ermöglicht. Das Capture kann dann wiederum mittels \g<name> referenziert werden. Wichtig: name darf nicht mit einer Zahl beginnen. Die fortlaufende Nummerierung (siehe oben) wird durch benannte Captures nicht unterbrochen. Beispiel: Das Pattern (?P<all>([a-zA-Z]*)(?P<digits>[0-9]*)) angewendet auf den String 'user123' liefert folgende Captures: \1 bzw. \g<all> user123 \2 user \3 bzw. \g<digits> 123

Element	Bedeutung im regulären Ausdruck
(?:)	<p>Diese Klammern werden nicht als Capture gespeichert.</p> <p>Beispiel: Das Pattern <code>(a(?:b c))([0-9])*</code> auf den String 'ab123' liefert die folgenden Captures:</p> <pre>\1 ab \2 123</pre>
(?:)	Modifikatoren können zwischen ? und : gesetzt werden, um die Syntax zu vereinfachen.

2.2. SQL-Befehle

2.2.1. Definition der Datenbank (DDL)

Mit Hilfe der Data Definition Language (DDL) wird die Struktur der Datenbank definiert. Dies betrifft insbesondere das Anlegen von Schemas und allen darin enthaltenen Schemaobjekte wie z.B. Tabellen, Views, Funktionen und Skripten.

CREATE SCHEMA

Zweck

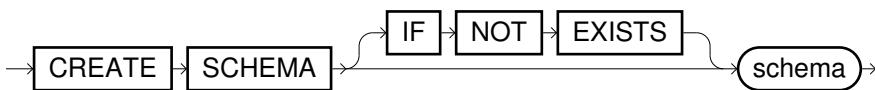
Mit diesem Kommando wird ein neues Schema erzeugt. Dies kann entweder ein physikalisches Schema sein, in dem normale Schemaobjekte erzeugt werden können, oder eine Art virtueller Link zu einer externen Datenquelle, bei dem dessen Daten und Metadaten in das virtuelle Schema abgebildet werden.

Vorbedingung(en)

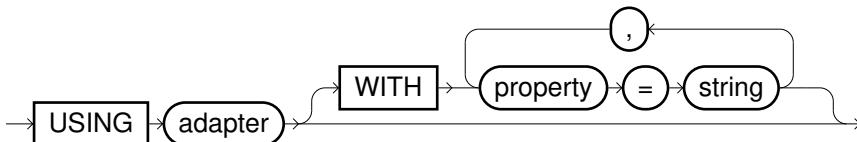
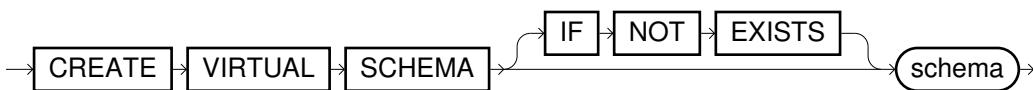
- System-Privileg CREATE SCHEMA bzw. CREATE VIRTUAL SCHEMA.
- Für virtuelle Schema benötigen Sie zusätzlich das EXECUTE-Privileg für das entsprechende Adapter-Skript.

Syntax

create_schema ::=



create_virtual_schema ::=



Anmerkung(en)

- Ein Benutzer kann mehrere Schemas anlegen.
- Das angelegte Schema gehört dem aktuellen Nutzer, kann aber mit dem Befehl [ALTER SCHEMA](#) einem anderen Benutzer oder einer Rolle zugewiesen werden.
- Beim Anlegen eines Schemas wechselt man ins neue Schema. Das bedeutet, dass es zum CURRENT_SCHEMA gesetzt wird, was vor allem die Namensauflösung beeinflusst.
- Bei Angabe der Option IF NOT EXISTS wird keine Fehlermeldung geworfen, falls das Schema bereits existiert. Zusätzlich wird in dieses Schema gewechselt.
- Details zu virtuellen Schemas finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#). Die USING Option spezifiziert das benutzte Adapter UDF Skript, welches den Inhalt des virtuellen Schemas definiert. Mit der WITH Klausel können bestimmte Parameter angegeben werden, die vom Adapter Skript benutzt werden.



Bitte beachten Sie, dass virtuelle Schemas Teil der Advanced Edition von Exasol ist.

Beispiel(e)

```
CREATE SCHEMA my_schema;
CREATE VIRTUAL SCHEMA hive
  USING adapter.jdbc_adapter
  WITH
    SQL_DIALECT      = 'HIVE'
    CONNECTION_STRING = 'jdbc:hive2://localhost:10000/default'
    SCHEMA_NAME       = 'default'
    USERNAME          = 'hive-usr'
    PASSWORD          = 'hive-pwd';
```

DROP SCHEMA

Zweck

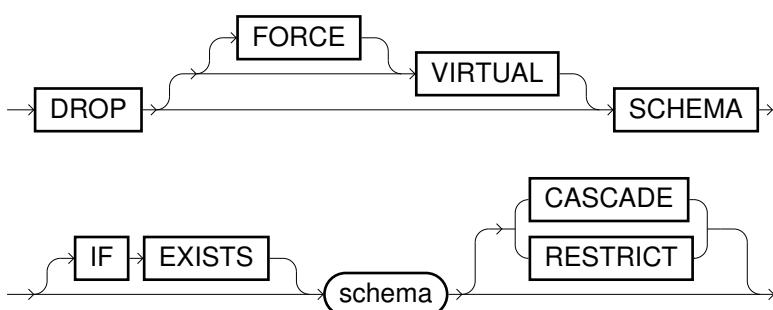
Mit diesem Kommando kann ein Schema und die darin enthaltenen Objekte vollständig gelöscht werden.

Vorbedingung(en)

- Physikalisches Schema: System-Privileg DROP ANY SCHEMA oder das Schema gehört dem aktuellen Benutzer oder einer seiner Rollen.
- Virtuelles Schema: System-Privileg DROP ANY VIRTUAL SCHEMA Systemprivileg oder das Schema gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

drop_schema ::=



Anmerkung(en)

- RESTRICT: Löscht das Schema nur dann, wenn es leer ist.
- CASCADE: Löscht das Schema und alle darin enthaltenen Schemaobjekte. Zudem werden alle Foreign Keys gelöscht, die Tabellen dieses Schemas referenzieren.
- Falls weder CASCADE noch RESTRICT angegeben wird, so wird RESTRICT angenommen.
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls das Schema nicht existiert.
- Ein Benutzer kann eine Vielzahl von Schemas besitzen.

- Details zu virtuellen Schemas finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#). Bei Angabe der FORCE Option wird das zugehörige Adapter Skript nicht über diese Operation informiert. Ansonsten wird das Adapter Skript aufgerufen und kann auf die Aktion je nach Implementierung reagieren.



Bitte beachten Sie, dass virtuelle Schemas Teil der Advanced Edition von Exasol ist.

Beispiel(e)

```
DROP SCHEMA my_schema;
DROP SCHEMA my_schema CASCADE;
DROP VIRTUAL SCHEMA my_virtual_schema;
```

ALTER SCHEMA

Zweck

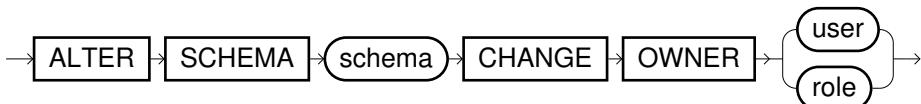
Mit diesem Befehl ist es möglich, ein Schema einem anderen Benutzer oder einer Rolle zuzuordnen. Das Schema sowie alle darin enthaltenen Schemaobjekte gehören dann dem angegebenen Benutzer bzw. der angegebenen Rolle.

Vorbedingung(en)

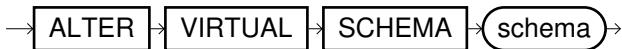
- Das Systemprivileg ALTER ANY SCHEMA oder das Objektprivileg ALTER auf s. Es reicht nicht, der Besitzer des Schemas zu sein!
- Virtual schemas:
 - CHANGE OWNER: Systemprivileg ALTER ANY VIRTUAL SCHEMA oder das Objektprivileg ALTER auf dem Schema. Es reicht nicht, der Besitzer des Schemas zu sein!
 - SET: Systemprivileg ALTER ANY VIRTUAL SCHEMA, Objektprivileg ALTER auf s oder das Schema gehört dem aktuellen Nutzer oder einer seiner Rollen. Zusätzlich sind Zugriffsrechte auf das entsprechende Adapter-Skript nötig.
 - REFRESH: Systemprivileg ALTER ANY VIRTUAL SCHEMA oder Systemprivileg ALTER ANY VIRTUAL SCHEMA REFRESH, Objektprivileg ALTER oder REFRESH auf s, oder s gehört dem aktuellen Nutzer oder einer seiner Rollen. Zusätzlich sind Zugriffsrechte auf das entsprechende Adapter-Skript nötig.

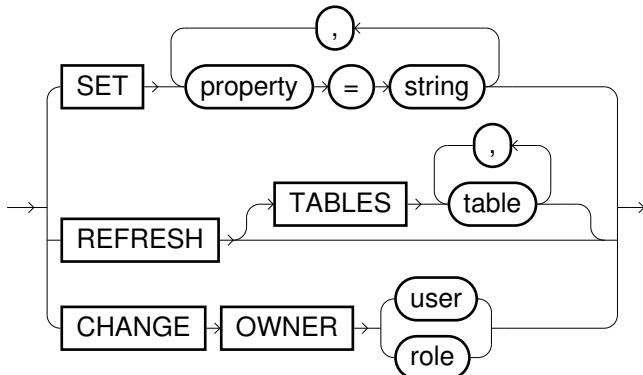
Syntax

alter_schema ::=



alter_virtual_schema ::=





Anmerkung(en)

- Dieses Statement ist nicht im SQL-Standard enthalten.
- Details zu virtuellen Schemas finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#). Beim REFRESH werden die Metadaten eines virtuellen Schemas aktualisiert, also die darunter liegenden Objekte im externen System. Dies wird nur explizit über diesen Befehl gemacht und nicht implizit bei Anfragen auf das virtuelle Schema, weil ansonsten Transaktionskonflikte bei lesenden Transaktionen entstehen könnten.

Bitte beachten Sie, dass virtuelle Schemas Teil der Advanced Edition von Exasol ist.

Beispiel(e)

```

ALTER SCHEMA s1 CHANGE OWNER user1;
ALTER SCHEMA s1 CHANGE OWNER role1;
ALTER VIRTUAL SCHEMA s2
  SET CONNECTION_STRING = 'jdbc:hive2://localhost:10000/default';
ALTER VIRTUAL SCHEMA s2 REFRESH;
  
```

CREATE TABLE

Zweck

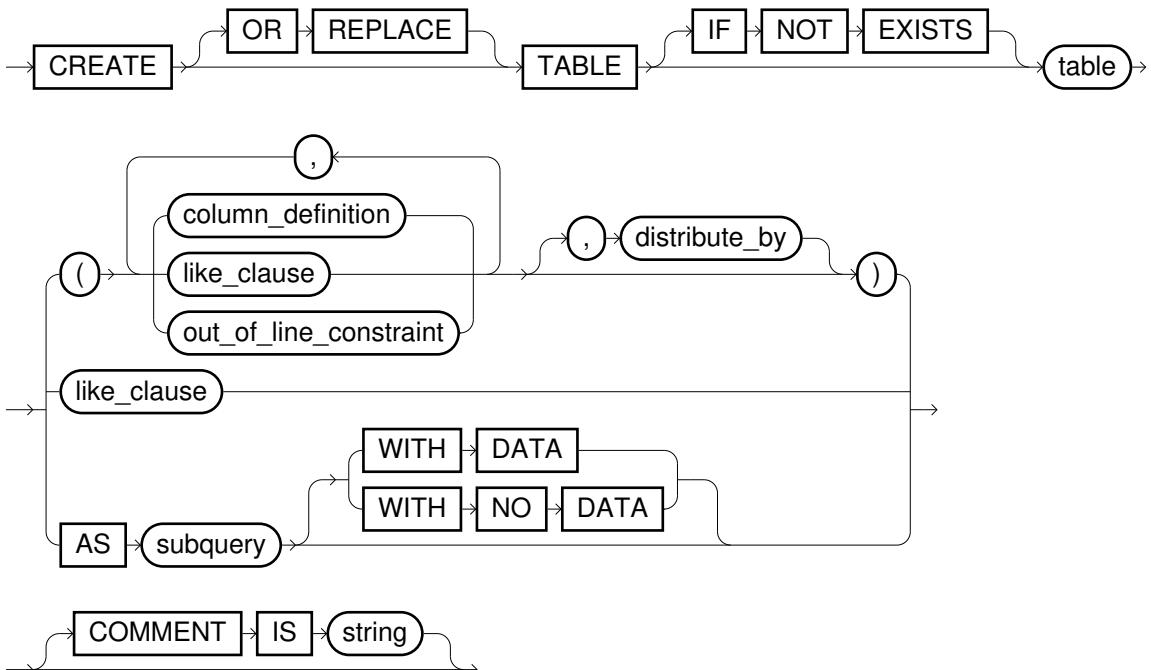
Mit diesem Kommando kann eine Tabelle erzeugt werden. Dies geschieht entweder durch die Angabe der Spaltentypen bzw. Tabellen, die als Vorlage dienen, oder durch die Zuweisung einer Subquery.

Vorbedingung(en)

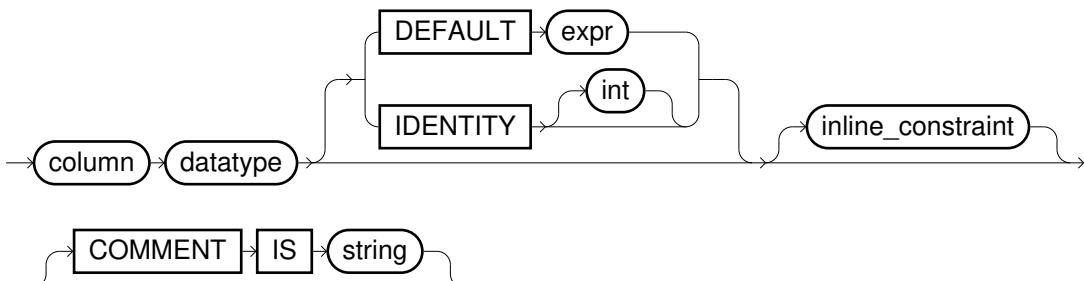
- System-Privileg CREATE TABLE, falls die Tabelle in einem eigenen Schema oder dem einer zugewiesenen Rolle angelegt werden soll, oder CREATE ANY TABLE. Falls die Tabelle über eine Subquery definiert wird, so benötigt der Benutzer die entsprechenden SELECT-Privilegien auf alle in der Subquery referenzierten Objekte.
- Falls die OR REPLACE-Option angegeben wurde und die Tabelle bereits existiert, so sind zudem die für [DROP TABLE](#) benötigten Rechte erforderlich.

Syntax

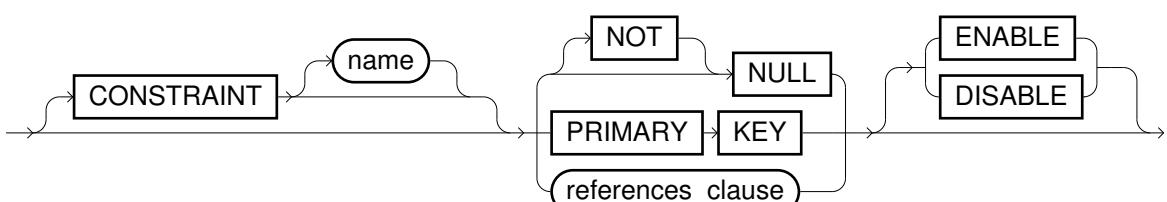
create_table ::=



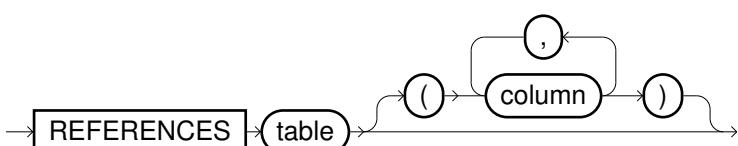
column_definition::=



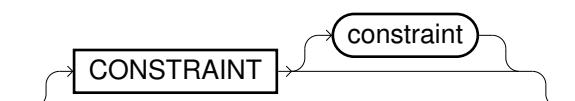
inline_constraint::=

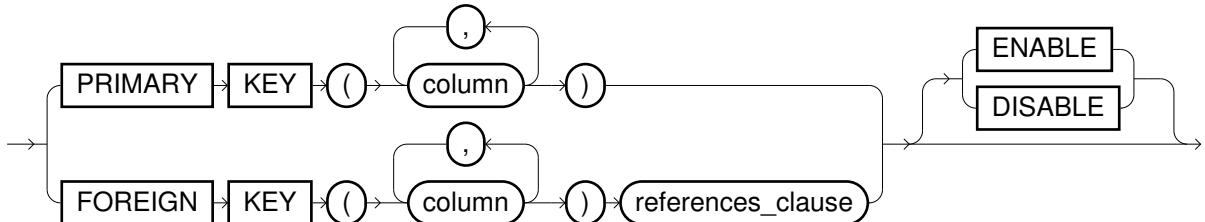


references_clause::=

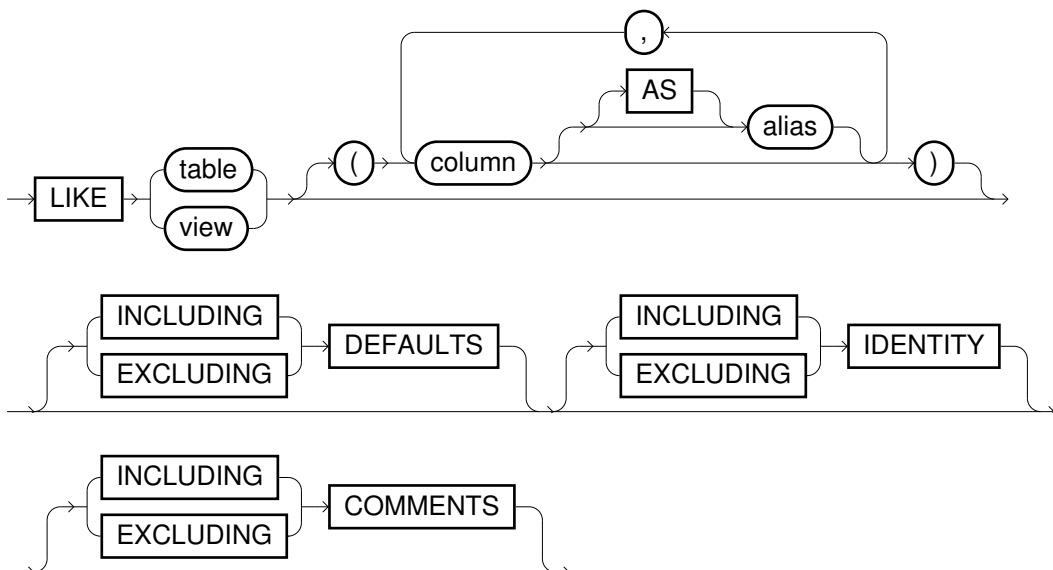


out_of_line_constraint::=

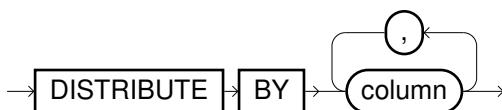




like_clause ::=



distribute_by ::=



Anmerkung(en)

- Mit der OR REPLACE-Option kann eine bereits existierende Tabelle ersetzt werden, ohne diese explizit mittels **DROP TABLE** löschen zu müssen.
- Bei Angabe der Option IF NOT EXISTS wird keine Fehlermeldung geworfen, falls die Tabelle bereits existiert.
- Detaillierte Informationen zu den verwendbaren Datentypen finden Sie in [Abschnitt 2.3, Datentypen](#).
- Details zu Constraints finden Sie im Befehl [ALTER TABLE \(constraints\)](#).
- Ein angegebener Default-Wert muss zum Datentyp seiner Spalte passen. Detaillierte Informationen zu den erlaubten Ausdrücken für Default-Werte finden Sie in [Abschnitt 2.3.5, Default-Werte](#).
- Details zur Identity-Eigenschaft finden Sie unter [Abschnitt 2.3.6, Identity-Spalten](#).
- Bei der Tabellendefinition durch eine Subquery (CREATE TABLE AS) ist folgendes zu beachten: falls in der Subquery keine direkten Spalten referenziert sind, sondern zusammengezogene Ausdrücke, so müssen für diese Spaltenalias angegeben werden.
- Wird CREATE TABLE AS mit der Option WITH NO DATA verwendet, so wird die Query nicht berechnet, sondern lediglich eine Tabelle mit den Spaltennamen und Datentypen angelegt, die das Ergebnis der Query hätte.
- Alternativ zu CREATE TABLE AS kann auch die Formulierung SELECT INTO TABLE verwendet werden (siehe [SELECT INTO](#)).
- Mit LIKE können alle oder ausgewählte Spalten einer angegebenen Tabelle bzw. View übernommen werden. Bei Angabe von INCLUDING DEFAULTS, INCLUDING IDENTITY bzw. INCLUDING COMMENTS werden zu diesen Spalten außerdem ihre Default-Werte, Identity-Spalten und Spaltenkommentare mit übernom-

men. Ein NOT NULL Constraint wird immer automatisch übernommen, Primary oder Foreign Key Constraints hingegen nie.

 Wird die `like_clause`, aber keine `distribution_clause` angegeben, so wird der Verteilungsschlüssel immer dann von den abgeleiteten Tabellen übernommen, falls alle Verteilungsschlüssel-Spalten einer Tabelle ausgewählt wurden und keine weitere Verteilungsschlüssel-Spalte einer anderen Tabelle.

- LIKE-Ausdrücke und normale Spaltendefinitionen können beliebig gemischt verwendet werden, müssen dann aber in runden Klammern eingeschlossen werden.
- Mit der `distribution_clause` wird die explizite Verteilung der Tabelle auf die Clusterknoten definiert. Nähere Informationen zu diesem Thema finden Sie auch im Befehl [ALTER TABLE \(distribution\)](#).
- Kommentare zu Tabelle und Spalten können Sie ebenfalls nachträglich mit dem Befehl `COMMENT` setzen bzw. ändern.

Beispiel(e)

```
CREATE TABLE t1 (a VARCHAR(20),
                 b DECIMAL(24,4) NOT NULL,
                 c DECIMAL DEFAULT 122,
                 d DOUBLE,
                 e TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                 f BOOL);
CREATE TABLE t2 AS SELECT * FROM t1;
CREATE OR REPLACE TABLE t2 AS SELECT a,b,c+1 AS c FROM t1;
CREATE TABLE t3 AS SELECT count(*) AS my_count FROM t1 WITH NO DATA;
CREATE TABLE t4 LIKE t1;
CREATE TABLE t5 (id int IDENTITY PRIMARY KEY,
                 LIKE t1 INCLUDING DEFAULTS,
                 g DOUBLE,
                 DISTRIBUTIVE BY a,b);
SELECT * INTO TABLE t6 FROM t1;
```

SELECT INTO

Zweck

Mit diesem Kommando kann eine Tabelle erzeugt werden, ähnlich wie bei CREATE TABLE AS (siehe [CREATE TABLE](#)).

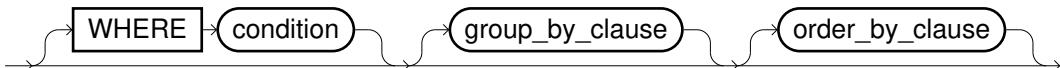
Vorbedingung(en)

- System-Privileg `CREATE TABLE`, falls die Tabelle in einem eigenen Schema oder dem einer zugewiesenen Rolle angelegt werden soll, oder `CREATE ANY TABLE`. Außerdem benötigt der Benutzer die entsprechenden `SELECT`-Privilegien auf alle referenzierten Objekte.

Syntax

`select_into ::=`





Anmerkung(en)

- Details zu den einzelnen Syntax-Elementen (z.B. select_list,...) siehe [SELECT](#).
- Falls in der Select-Liste keine direkten Spalten referenziert sind, sondern zusammengeetzte Ausdrücke, so müssen für diese Spaltenalias angegeben werden.

Beispiel(e)

```
SELECT * INTO TABLE t2 FROM t1 ORDER BY 1;
```

DROP TABLE

Zweck

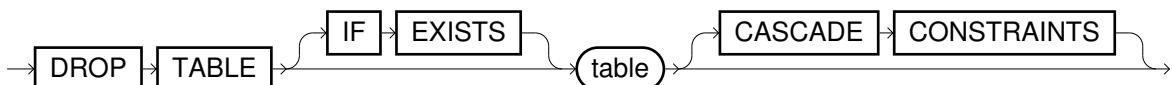
Mit diesem Kommando kann eine Tabelle gelöscht werden.

Vorbedingung(en)

- System-Privileg DROP ANY TABLE oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

drop_table ::=



Anmerkung(en)

- Falls ein Foreign Key die zu löschennde Tabelle referenziert, so muss die Option CASCADE CONSTRAINTS angegeben werden. In diesem Fall wird der Foreign Key ebenfalls gelöscht, auch wenn die referenzierende Tabelle nicht dem aktuellen Nutzer gehört.
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die Tabelle nicht existiert.

Beispiel(e)

```
DROP TABLE my_table;
```

ALTER TABLE (column)

Zweck

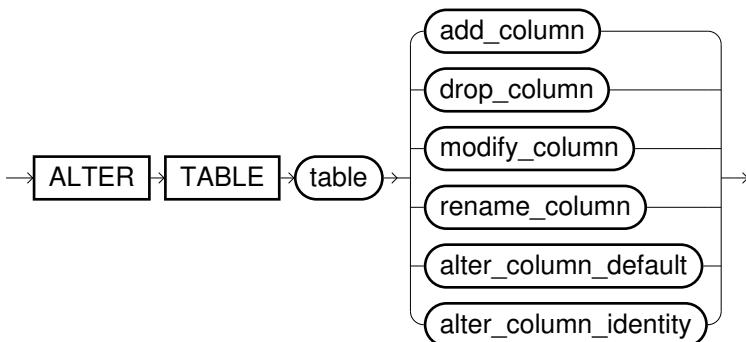
Hinzufügen, Löschen, Datentypänderung und Umbenennen einer Spalte sowie das Definieren von Default-Werten und Identity-Spalten.

Vorbedingung(en)

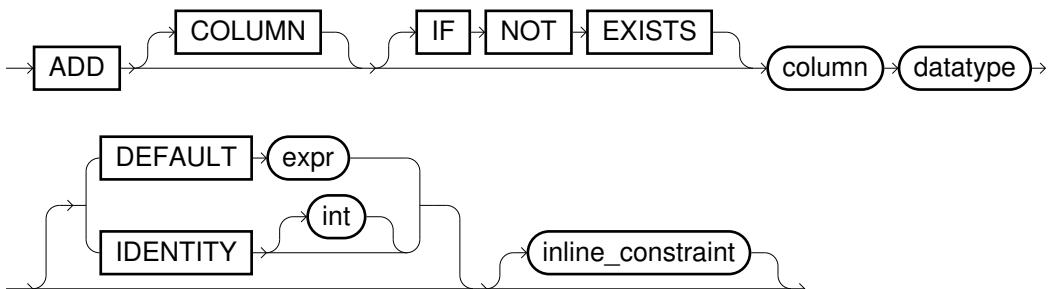
- System-Privileg ALTER ANY TABLE, Objekt-Privileg ALTER auf die Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

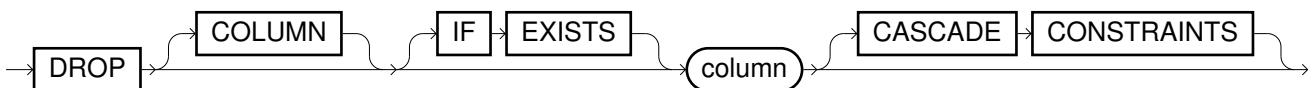
`alter_column ::=`



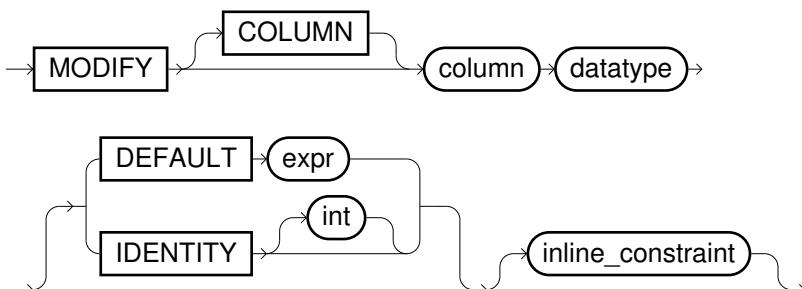
`add_column ::=`



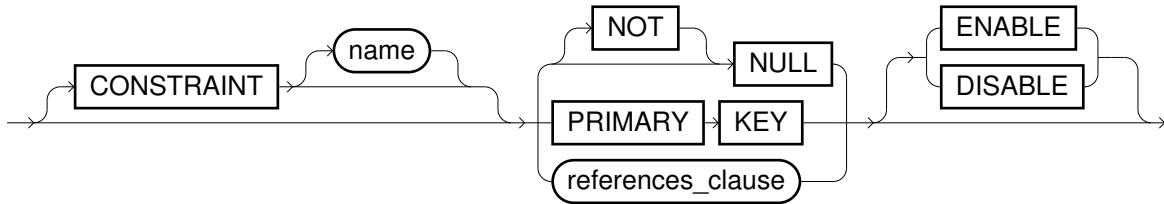
`drop_column ::=`



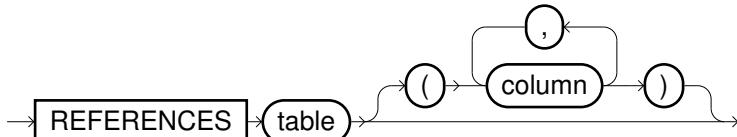
`modify_column ::=`



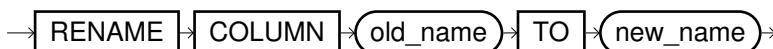
`inline_constraint ::=`



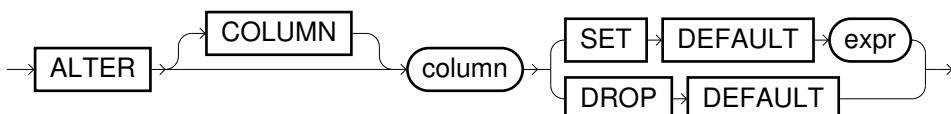
references_clause ::=



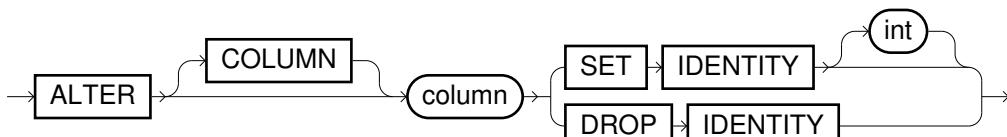
rename_column ::=



alter_column_default ::=



alter_column_identity ::=



Anmerkung(en)

ADD COLUMN

- Falls die Tabelle bereits Zeilen besitzt, so werden die Inhalte der hinzugefügten Spalte auf den Default-Wert gesetzt, falls ein solcher angegeben wurde, anderenfalls werden sie mit NULL initialisiert.
- Ein angegebener Default-Wert muss zum Datentyp seiner Spalte passen. Detaillierte Informationen zu den erlaubten Ausdrücken für Default-Werte finden Sie in [Abschnitt 2.3.5, Default-Werte](#).
- Beim Hinzufügen einer Identity-Spalte werden bei bereits existierenden Zeilen entsprechende Einträge generiert. Details zur Identity-Eigenschaft finden Sie unter [Abschnitt 2.3.6, Identity-Spalten](#).
- Bei Angabe der Option IF NOT EXISTS wird keine Fehlermeldung geworfen, falls die Spalte bereits existiert.
- Details zu Constraints finden Sie im Befehl [ALTER TABLE \(constraints\)](#).

DROP COLUMN

- Falls ein Foreign Key die zu löschen Spalte referenziert, so muss die Option CASCADE CONSTRAINTS angegeben werden. In diesem Fall wird der Foreign Key ebenfalls gelöscht, auch wenn die referenzierende Tabelle nicht dem aktuellen Nutzer gehört.
- Falls eine Spalte gelöscht wird, die Teil des Verteilungsschlüssels ist (siehe auch [ALTER TABLE \(distribution\)](#)), so werden die Verteilungsschlüssel

- komplett gelöscht und neue Zeilen der Tabelle wieder zufällig über die Knoten verteilt.
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die Spalte nicht existiert.
- MODIFY COLUMN**
- Die angegebenen Typen müssen konvertierbar sein. Dies betrifft sowohl die Datentypen (z.B. ist BOOL nicht in TIMESTAMP konvertierbar) als auch die Inhalte der Spalte (CHAR(3) mit Inhalt '123' kann in DECIMAL konvertiert werden, aber 'ABC' nicht).
 - Falls ein Default-Wert angegeben ist, muss dieser zum Datentyp passen. Details zu erlaubten Ausdrücken für den Default-Wert *expr* finden sich unter [Abschnitt 2.3.5, Default-Werte](#). Wurde kein Default-Wert angegeben, wird ein eventuell vorhandener alter Default-Wert weiter verwendet und muss zum neuen Datentyp passen.
 - Details zur Identity-Eigenschaft finden Sie unter [Abschnitt 2.3.6, Identity-Spalten](#). Wurde keine Identity-Eigenschaft angegeben, wird die eventuell vorhandene Identity-Eigenschaft weiter verwendet und muss zum neuen Datentyp passen.
 - Details zu Constraints finden Sie im Befehl [ALTER TABLE \(constraints\)](#). Falls ein Constraint definiert wird, so kann es lediglich hinzugefügt werden, existierende können hiermit nicht geändert werden. Ausnahme: Wird ein NULL Constraint angegeben, so wird ein eventuell vorhandenes NOT NULL Constraint entfernt. Wurde kein Constraint angegeben, wird ein eventuell vorhandenes Constraint nicht verändert.
 - Falls eine Spalte modifiziert wird, die Teil des Verteilungsschlüssels ist (siehe auch [ALTER TABLE \(distribution\)](#)), so wird die Tabelle anschließend umverteilt.
- RENAME COLUMN**
- Die Inhalte der Tabelle werden durch diesen Befehl nicht verändert.
 - Die Tabelle darf keine Spalte mit Namen *new_name* besitzen.
- ALTER COLUMN DEFAULT**
- Die Inhalte der Tabelle werden durch diesen Befehl nicht verändert.
 - Ein angegebener Default-Wert muss zum Datentyp seiner Spalte passen.
 - Details zu erlaubten Ausdrücken für den Default-Wert *expr* finden sich unter [Abschnitt 2.3.5, Default-Werte](#).
 - DROP DEFAULT sollte gegenüber SET DEFAULT NULL bevorzugt werden.
- ALTER COLUMN IDENTITY**
- Die Inhalte der Tabelle werden durch diesen Befehl nicht verändert.
 - Bei Angabe des optionalen Parameters *int* wird der Zahlengenerator auf diese Zahl gesetzt, ansonsten auf 0. Beginnend mit dieser Zahl werden bei INSERT-Statements, die keinen expliziten Wert für die Identity-Spalte einfügen, aufsteigende Zahlenwerte erzeugt. Bitte beachten Sie, dass diese Zahlen zwar monoton aufsteigend sind, aber Lücken im Zahlenbereich entstehen können. Mit diesem Befehl lässt sich der Zahlengenerator übrigens auch auf bestimmte Werte "zurücksetzen".
 - Details zu Identity-Spalten finden Sie unter [Abschnitt 2.3.6, Identity-Spalten](#).

Beispiel(e)

```
ALTER TABLE t ADD COLUMN IF NOT EXISTS new_dec DECIMAL(18,0);
ALTER TABLE t ADD new_char CHAR(10) DEFAULT 'some text';

ALTER TABLE t DROP COLUMN i;
ALTER TABLE t DROP j;

ALTER TABLE t MODIFY (i DECIMAL(10,2));
```

```

ALTER TABLE t MODIFY (j VARCHAR(5) DEFAULT 'text');
ALTER TABLE t MODIFY (k INTEGER IDENTITY(1000));

ALTER TABLE t RENAME COLUMN i TO j;

ALTER TABLE t ALTER COLUMN v SET DEFAULT CURRENT_USER;
ALTER TABLE t ALTER COLUMN v DROP DEFAULT;

ALTER TABLE t ALTER COLUMN id SET IDENTITY(1000);
ALTER TABLE t ALTER COLUMN id DROP IDENTITY;

```

ALTER TABLE (distribution)

Zweck

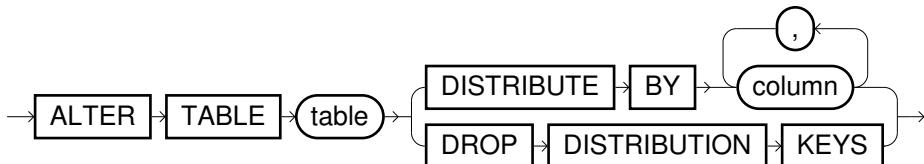
Mit diesen Kommandos wird die Verteilung der Zeilen einer Tabelle über die Clusterknoten gesteuert.

Vorbedingung(en)

- System-Privileg ALTER ANY TABLE, Objekt-Privileg ALTER auf die Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

distribute_by::=



Anmerkung(en)

- Durch die explizite Datenverteilung einer Tabelle können bestimmte SQL-Anfragen wesentlich beschleunigt werden. Sind z.B. bei einem Join zwischen zwei Tabellen **beide** Tabellen nach den Join-Spalten verteilt, so kann der Join auf den Cluster-Knoten lokal durchgeführt werden. Ebenfalls wird ein GROUP BY auf eine einzelne Tabelle beschleunigt, wenn alle bei der Verteilung angegebenen Spalten in den Gruppierungselementen vorhanden sind. Bei kleinen Tabellen mit einigen tausend Zeilen ist eine explizite Verteilung jedoch nicht sinnvoll.



Dieser Befehl kann eine lange Ausführungszeit haben und sollte nur dann benutzt werden, wenn Sie sich über die Auswirkungen bewusst sind. Insbesondere können Querylaufzeiten auch negativ beeinflusst werden.

- Wurde die Verteilung einer Tabelle nicht explizit definiert, so verteilt das System die Zeilen gleichmäßig über die Cluster-Knoten.
- Bei Ausführung dieses Statements wird die Verteilung der Zeilen über die Cluster-Knoten durch einen Hashwert über die entsprechenden Spalten berechnet. Die Reihenfolge der angegebenen Spalten ist hierfür irrelevant. Nach der Verteilung bleibt die Tabelle immer in diesem explizit verteiltem Zustand, auch nach Einfüge-, Lösch- und Update-Operationen.
- Die konkrete Zuordnung zwischen Werten und Datenbank-Knoten können Sie mit der Funktion **VALUE2PROC** nachvollziehen.

- Bei der Angabe von ungünstigen Verteilungsspalten, die eine starke Ungleichverteilung hervorrufen würden, wird der Befehl abgebrochen und eine entsprechende Fehlermeldung ausgegeben.
- Eine explizite Verteilung einer Tabelle kann auch direkt im **CREATE TABLE** Befehl gesetzt werden.
- Nach Setzen oder Entfernen des Verteilungsschlüssels werden alle internen Indizes neu gebaut.
- Durch den Befehl **DROP DISTRIBUTION KEYS** lässt sich die explizite Verteilung einer Tabelle wieder aufheben. Anschließend werden die Zeilen wieder zufällig, jedoch gleichmäßig vom System verteilt.
- Mit dem **DESC[RIBE]** Befehl, aber auch über die Systemtabelle **EXA_ALL_COLUMNS**, kann man die Verteilungsschlüssel einer Tabelle abfragen. In der Systemtabelle **EXA_ALL_TABLES** lässt sich darüber hinaus anzeigen, welche Tabellen explizit verteilt sind.

Beispiel(e)

```

CREATE TABLE my_table (i DECIMAL(18,0), vc VARCHAR(100), c CHAR(10));
ALTER TABLE my_table DISTRIBUTE BY i, c;

DESCRIBE my_table;
COLUMN_NAME SQL_TYPE           NULLABLE DISTRIBUTION_KEY
-----
I          DECIMAL(18,0)        TRUE      TRUE
VC         VARCHAR(100)        UTF8     FALSE
C          CHAR(10)           UTF8     TRUE

ALTER TABLE my_table DROP DISTRIBUTION KEYS;

```

ALTER TABLE (constraints)

Zweck

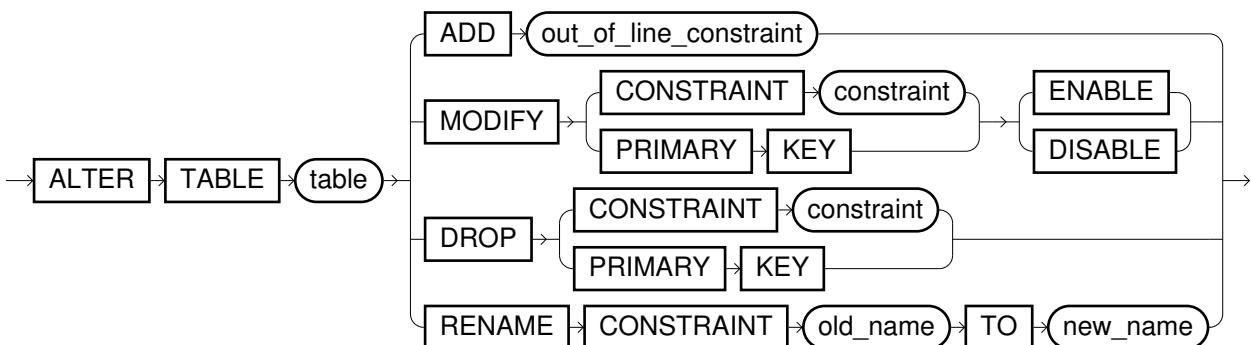
Mit diesen Kommandos können Constraints definiert, gelöscht oder modifiziert werden.

Vorbedingung(en)

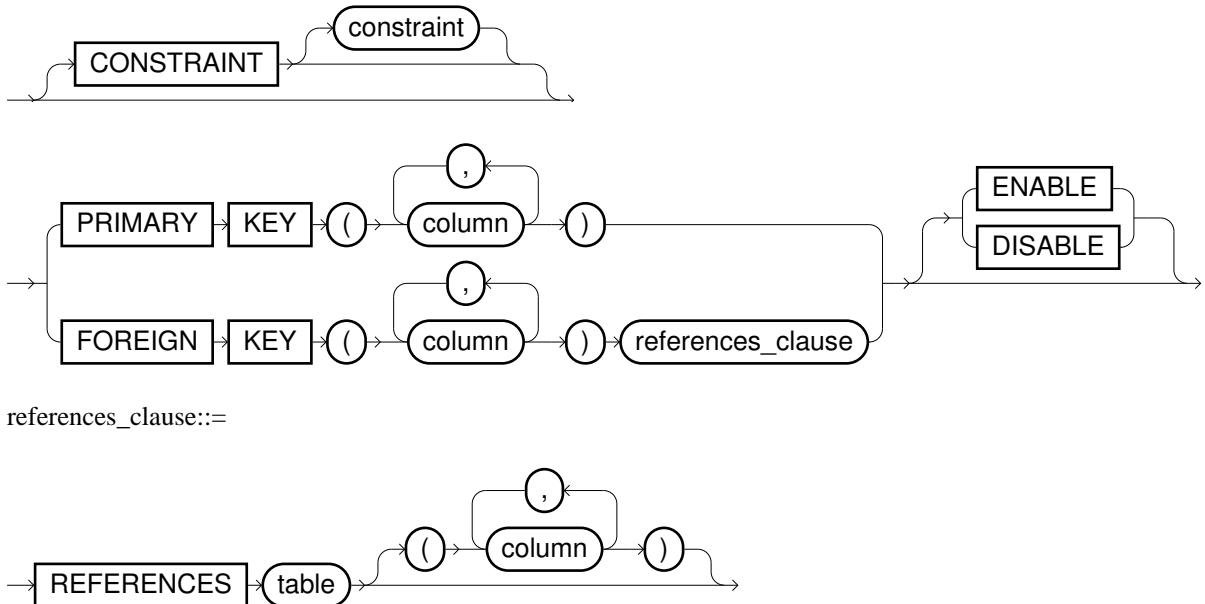
- System-Privileg **ALTER ANY TABLE**, Objekt-Privileg **ALTER** auf die Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

`alter_table_constraint ::=`



`out_of_line_constraint ::=`



Anmerkung(en)

- Folgende Constraints können für Spalten definiert werden:

PRIMARY KEY Alle Werte müssen eindeutig sein, NULL-Werte sind nicht erlaubt. Eine Tabelle darf nur einen Primary Key besitzen.

FOREIGN KEY Ein Foreign Key referenziert stets den Primary Key einer zweiten Tabelle. Die Spalteninhalte müssen entweder im Primary Key vorkommen oder aber NULL sein (bei zusammengesetzten Keys mindestens eine der Spalten). Die Datentypen von Foreign Key und entsprechendem Primary Key müssen identisch sein. Foreign Keys können nicht auf virtuelle Objekte verweisen.

NOT NULL Es können keine NULL-Werte eingefügt werden. Ein NOT NULL Constraint kann nur direkt bei der Tabellendefinition oder mittels ALTER TABLE MODIFY COLUMN definiert werden.

- Constraints können zur leichteren Identifizierung einen Namen erhalten und besitzen einen der beiden Zustände:

ENABLE Das Constraint wird direkt nach DML-Statements (siehe [Abschnitt 2.2.2, Manipulation der Datenbank \(DML\)](#)) geprüft. Dieser Prozess kostet Zeit, gewährleistet aber die Datenintegrität.

DISABLE Das Constraint ist deaktiviert und wird nicht geprüft. Dieser Zustand ist zum Beispiel nützlich, wenn aus Performance-Gründen nur die Metadaten in der Datenbank hinterlegt werden sollen.

Falls kein expliziter Zustand definiert wurde, so wird der Session-Wert CONSTRAINT_STATE_DEFAULT gewählt (siehe auch [ALTER SESSION](#)).

- Die entsprechenden Metadaten zu Primary und Foreign Key Constraints finden Sie in den Systemtabellen [EXA_USER_CONSTRAINTS](#), [EXA_ALL_CONSTRAINTS](#) bzw. [EXA_DBAA_CONSTRAINTS](#), die Metadaten zu NOT NULL Constraints in [EXA_USER_CONSTRAINT_COLUMNS](#), [EXA_ALL_CONSTRAINT_COLUMNS](#) und [EXA_DBAA_CONSTRAINT_COLUMNS](#). Wurde kein Name für ein Constraint angegeben, so wird vom System ein eindeutiger Name generiert.
- Constraints können auch direkt im [CREATE TABLE](#) definiert und mittels [ALTER TABLE \(constraints\)](#) modifiziert werden.

Beispiel(e)

```
ALTER TABLE t1 ADD CONSTRAINT my_primary_key PRIMARY KEY (a);
ALTER TABLE t2 ADD CONSTRAINT my_foreign_key FOREIGN KEY (x) REFERENCES t1;
ALTER TABLE t2 MODIFY CONSTRAINT my_foreign_key DISABLE;
ALTER TABLE t2 RENAME CONSTRAINT my_foreign_key TO my_fk;
ALTER TABLE t2 DROP CONSTRAINT my_fk;
```

CREATE VIEW

Zweck

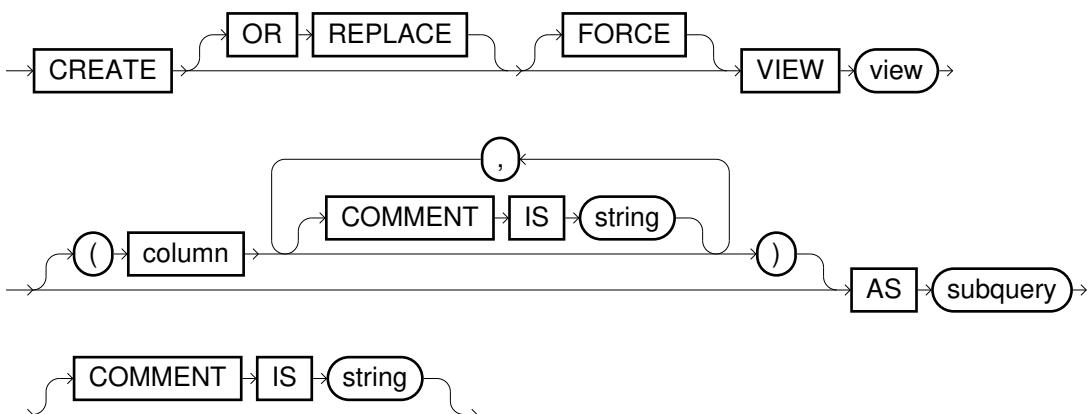
Mit diesem Kommando kann eine View erzeugt werden.

Vorbedingung(en)

- System-Privileg CREATE VIEW, falls die View in einem eigenen Schema oder dem einer zugewiesenen Rolle angelegt werden soll, oder CREATE ANY VIEW. Außerdem benötigt der Besitzer der View (dies ist nicht unbedingt der Erzeuger) die entsprechenden SELECT-Privilegien auf alle in der Subquery referenzierten Objekte.
- Falls die OR REPLACE-Option angegeben wurde und die View bereits existiert, so benötigt der Benutzer zudem die Rechte wie für [DROP VIEW](#).

Syntax

create_view ::=



Anmerkung(en)

- Mit der OR REPLACE-Option kann eine bereits existierende View ersetzt werden, ohne diese explizit mittels [DROP VIEW](#) löschen zu müssen.
- Mit Hilfe der FORCE-Option kann eine View erzeugt werden, ohne den Text zu komplizieren. Dies ist beispielsweise dann nützlich, wenn sehr viele Views angelegt werden sollen, die voneinander abhängig sind und dadurch in einer bestimmten Reihenfolge erzeugt werden müssten. Falls jedoch Syntax- oder sonstige Fehler enthalten sind, werden diese erst beim ersten Benutzen der View gefunden.
- Durch Angabe von Spaltenaliasen werden die Spaltennamen der View definiert. Falls diese nicht angegeben sind, so werden die Spaltennamen von der Subquery abgeleitet. Falls in der Subquery keine direkten Spalten referenziert sind, sondern zusammengezogene Ausdrücke, so müssen für diese entweder innerhalb der Subquery oder für die gesamte View Aliase angegeben werden.
- Der Erzeugungstext einer View ist auf 2.000.000 Zeichen begrenzt.

- Falls unterliegende Objekte einer View geändert werden, wird sie auf INVALID gesetzt (siehe auch Spalte STATUS in den Systemtabellen wie EXA_ALL_VIEWS). Beim nächsten lesenden Zugriff auf die View wird versucht, sie automatisch neu zu komplizieren. Ist dies erfolgreich möglich, so ist die View ab diesem Zeitpunkt wieder valide. Ansonsten erhält der Nutzer eine entsprechende Fehlermeldung. Dies bedeutet aber insbesondere, dass alle invaliden Views nicht unbedingt unbenutzbar sind.

Beispiel(e)

```
CREATE VIEW my_view as select x from t;
CREATE OR REPLACE VIEW my_view as select y from t;
CREATE OR REPLACE VIEW my_view (col_1) as select max(y) from t;
```

DROP VIEW

Zweck

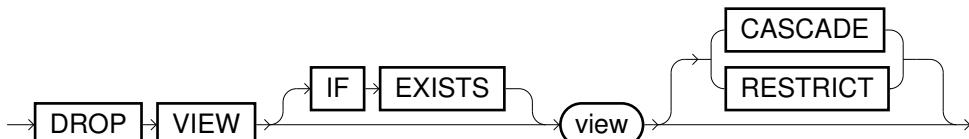
Mit diesem Kommando kann eine View gelöscht werden.

Vorbedingung(en)

- System-Privileg DROP ANY VIEW oder die View gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

drop_view ::=



Anmerkung(en)

- RESTRICT bzw. CASCADE sind aktuell nicht von Bedeutung, werden aber syntaktisch aus Kompatibilitätsgründen unterstützt.
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die View nicht existiert.

Beispiel(e)

```
DROP VIEW my_view;
```

CREATE FUNCTION

Zweck

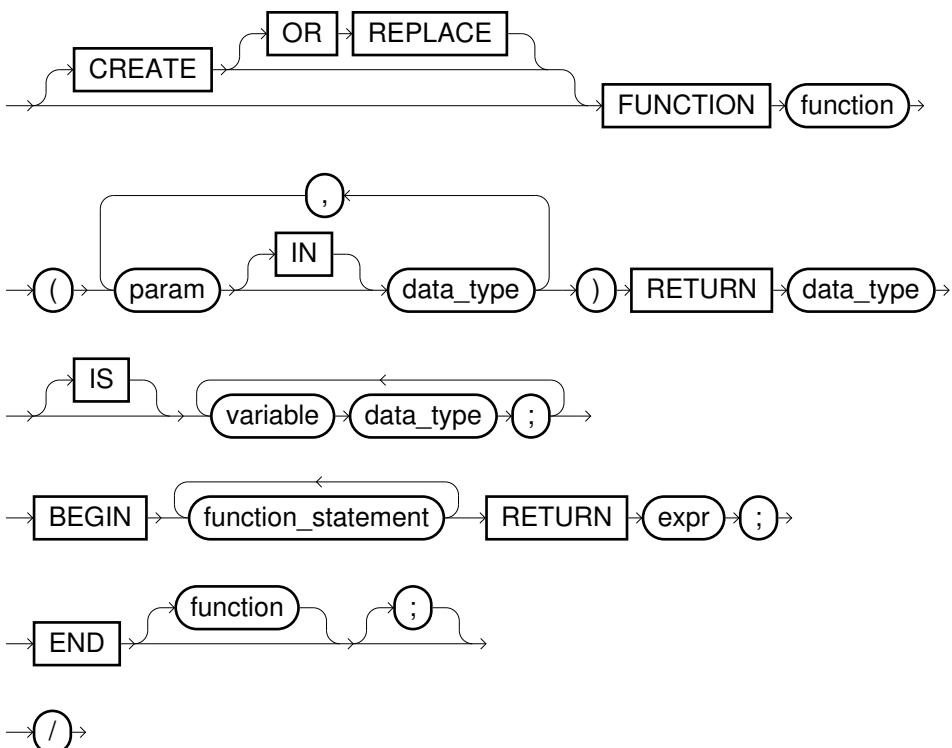
Mit diesem Kommando kann eine benutzerdefinierte Funktion erzeugt werden.

Vorbedingung(en)

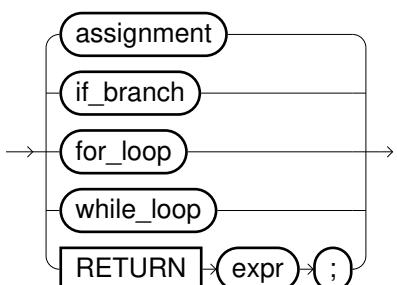
- System-Privileg CREATE FUNCTION, falls die Funktion in einem eigenen Schema oder dem einer zugewiesenen Rolle angelegt werden soll, oder CREATE ANY FUNCTION.
- Falls die OR REPLACE-Option angegeben wurde und die Funktion bereits existiert, so sind zudem die für DROP FUNCTION benötigten Rechte erforderlich.
- Um eine Funktion verwenden zu können, benötigt man das System-Privileg EXECUTE ANY FUNCTION, das Objekt-Privileg EXECUTE auf die Funktion bzw. deren Schema oder muss der Besitzer sein.

Syntax

create_function ::=



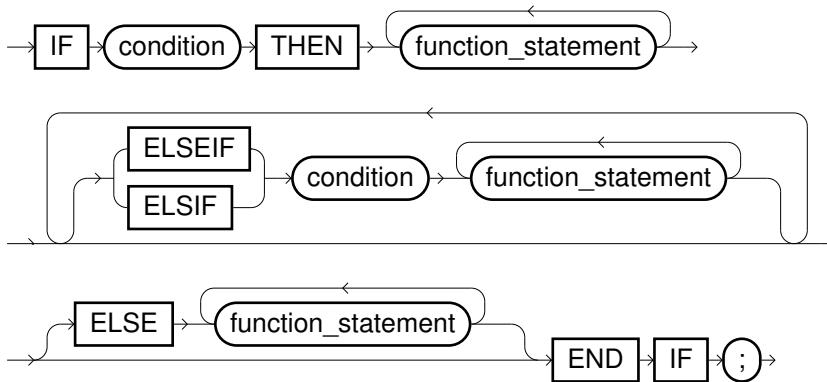
function_statement ::=



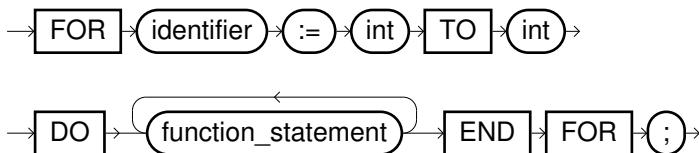
assignment ::=



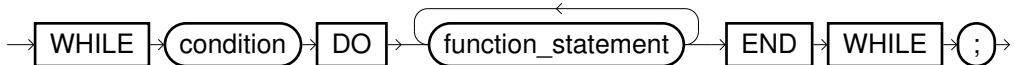
if_branch ::=



for_loop ::=



while_loop ::=



Anmerkung(en)

- Der abschließende Schrägstrich ('/') ist nur bei der Verwendung von EXAplus erforderlich.
- Mit der OR REPLACE-Option kann eine bereits existierende Funktion ersetzt werden, ohne diese explizit mittels **DROP FUNCTION** löschen zu müssen.
- Für Variablen Deklarationen sind die normalen SQL-Datentypen gültig (siehe auch [Abschnitt 2.3, Datentypen](#)).
- Variablen-Zuweisungen können nicht in der Variablen Deklaration durchgeführt werden, sondern nur im Funktionsrumpf.
- Für Ausdrücke (expr) können beliebige skalare SQL-Ausdrücke verwendet werden. Z.B. stehen sämtliche skalare Built-in-Funktionen (siehe [Abschnitt 2.9.1, Skalare Funktionen](#)) und vorher erzeugten Funktionen zur Verfügung.
- Die Zählvariable der FOR-Schleife durchläuft alle Ganzzahlen, die durch die Parameter definiert werden. Die Iterationszahl wird stets vor der ersten Rumpf-Ausführung ausgewertet und ist dann konstant.
- Wie in normalen SQL-Befehlen können innerhalb einer Funktionsdefinition beliebige Kommentare eingefügt werden (siehe auch [Abschnitt 2.1.1, Kommentare in SQL](#)).
- Der Definitionstext einer benutzerdefinierten Funktion steht u.a. in der Systemtabelle **EXA_ALL_FUNCTIONS** (siehe [Anhang A, Systemtabellen](#)).
- Skalare Subqueries innerhalb Funktionen sind nicht parametrisierbar, können also keine Variablen oder Parameter beinhalten.

Beispiel(e)

```

CREATE OR REPLACE FUNCTION percentage (fraction DECIMAL,
                                         entirety DECIMAL)
  RETURN VARCHAR(10)
  IS
    res DECIMAL;
  BEGIN
    IF entirety = 0
      THEN res := NULL;
    END IF;
    IF fraction < 0
      THEN res := '-' || LPAD(FLOOR(FRACTION * 100), 2, '0') || '%';
    ELSE
      res := LPAD(FLOOR(FRACTION * 100), 2, '0') || '%';
    END IF;
  END;
  
```

```

    ELSE
        res := (100*fraction)/entirety;
    END IF;
    RETURN res || ' %';
END percentage;
/

SELECT fraction, entirety, percentage(fraction,entirety) AS PERCENTAGE
FROM my_table;

FRACTION ENTIRETY PERCENTAGE
-----
      1          2      50 %
      1          3      33 %
      3         24     12 %

```

```

-- Beispiele für Funktions-Statements
-- assignment
    res := CASE WHEN input_variable < 0 THEN 0 ELSE input_variable END;

-- if-branch
    IF input_variable = 0 THEN
        res := NULL;
    ELSE
        res := input_variable;
    END IF;

-- for loop
    FOR cnt := 1 TO input_variable
    DO
        res := res*2;
    END FOR;

-- while loop
    WHILE cnt <= input_variable
    DO
        res := res*2;
        cnt := cnt+1;
    END WHILE;

```

DROP FUNCTION

Zweck

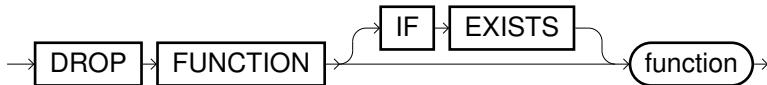
Mit diesem Kommando kann eine benutzerdefinierte Funktion gelöscht werden.

Vorbedingung(en)

- System-Privileg DROP ANY FUNCTION oder die Funktion gehört dem aktuellen User.

Syntax

drop_function::=



Anmerkung(en)

- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die Funktion nicht existiert.

Beispiel(e)

```
DROP FUNCTION my_function;
```

CREATE SCRIPT

Zweck

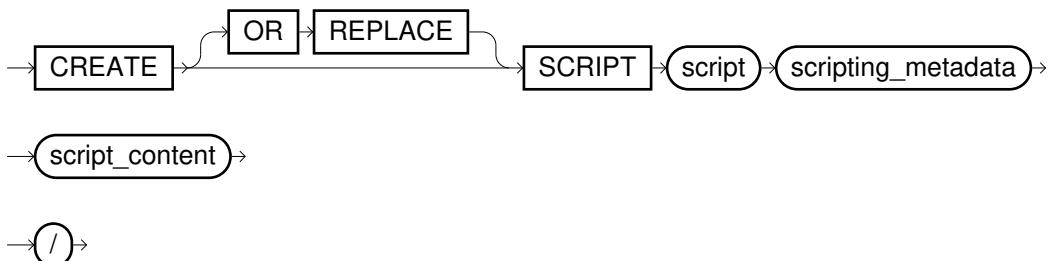
Mit diesem Kommando kann ein Datenbank-Skript erzeugt werden, wobei zwischen benutzerdefinierten Funktionen (UDF), Scripting-Programmen und Adapter-Skripten unterschieden wird.

Vorbedingung(en)

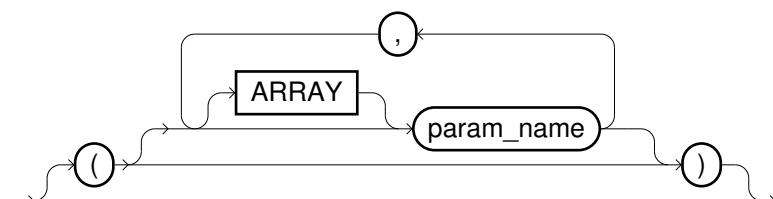
- System-Privileg `CREATE SCRIPT`, falls das Skript in einem eigenen Schema oder dem einer zugewiesenen Rolle angelegt werden soll, oder `CREATE ANY SCRIPT`.
- Falls die `OR REPLACE`-Option angegeben wurde und das Skript bereits existiert, so sind zudem die für `DROP SCRIPT` benötigten Rechte erforderlich.
- Um ein Skript (UDF oder Scripting) verwenden zu können, benötigt man das System-Privileg `EXECUTE ANY SCRIPT`, das Objekt-Privileg `EXECUTE` auf dem Skript bzw. dessen Schema oder muss der Besitzer sein.

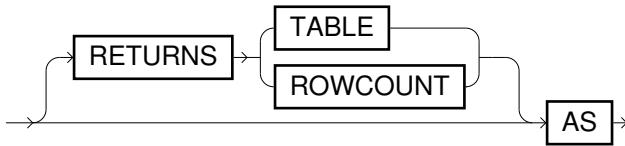
Syntax

`create_scripting_script ::=`

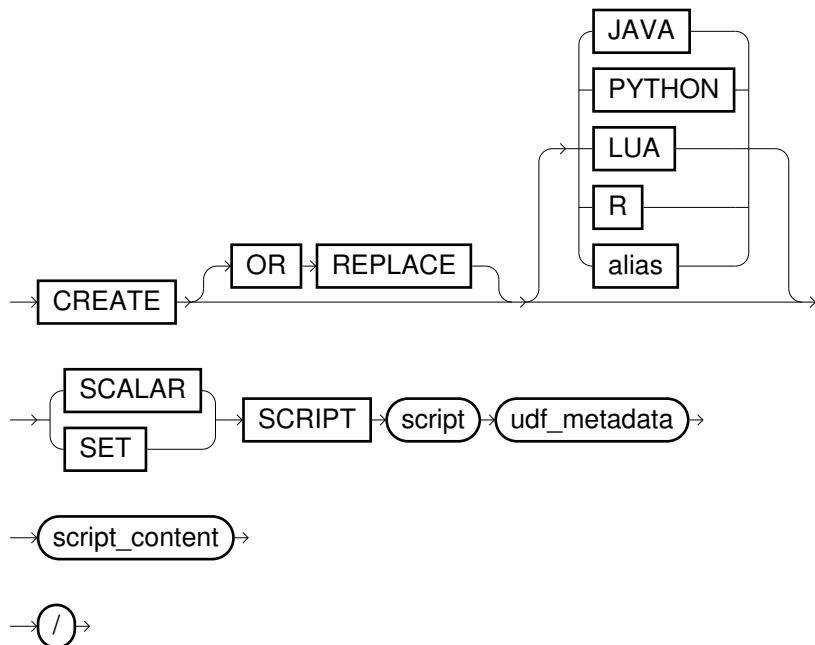


`scripting_metadata ::=`

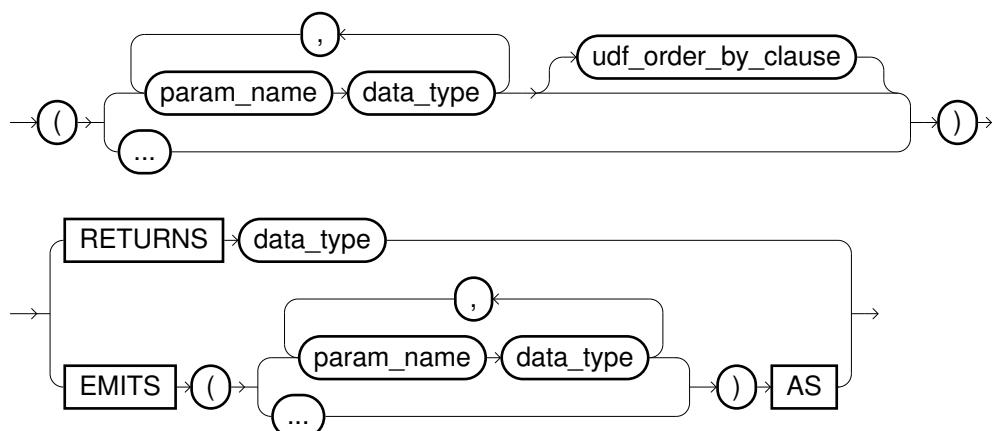




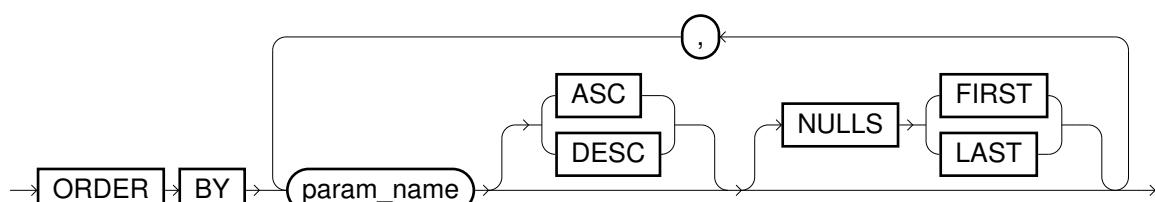
create_udf_script::=



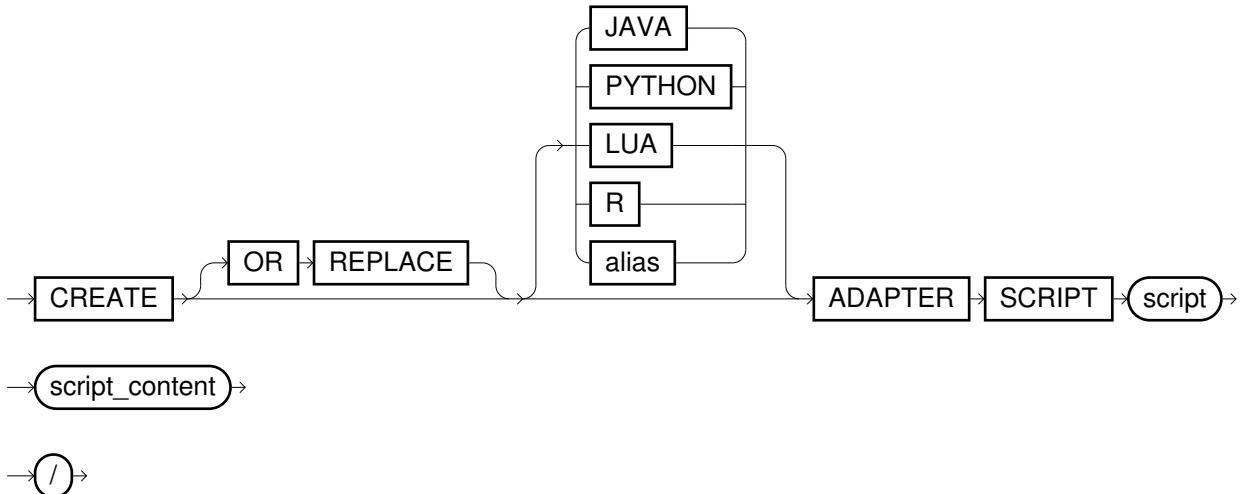
udf_metadata ::=



udf_order_by_clause ::=



create_adapter_script ::=



Anmerkung(en)

- Mit der **OR REPLACE**-Option kann ein bereits existierendes Skript ersetzt werden, ohne dieses explizit mittels **DROP SCRIPT** löschen zu müssen.
- Der abschließende Schrägstrich ('/') ist nur bei der Verwendung von EXAplus erforderlich.
- Der Inhalt des Skriptes steht u.a. in der Systemtabelle **EXA_ALL_SCRIPTS** (siehe [Anhang A, Systemtabellen](#)). Dieser beginnt bei der ersten Zeile nach dem Schlüsselwort AS, die keine Leerzeile ist.
- Falls Sie weitere Skript-Sprachen für benutzerdefinierten Funktionen (UDF-Skripte) und Adapter-Skripte integrieren wollen, beachten Sie bitte auch [Abschnitt 3.6.5, Skript-Sprachen erweitern mittels BucketFS](#).
- Hinweise zu Scripting-Programmen:
 - Scripting-Programme dienen der Programmablaufsteuerung von mehreren SQL-Befehlen (beispielsweise für ETL-Jobs) und werden mit dem Befehl **EXECUTE SCRIPT** ausgeführt. Details siehe [Abschnitt 3.6, UDF Skripte](#).
 - Für Scripting-Programme wird nur die Sprache Lua unterstützt. Eine ausführliche Erläuterung des Sprachumfangs finden Sie in [Abschnitt 3.5, Scripting-Programmierung](#).
 - Falls keiner der Optionen RETURNS TABLE bzw. RETURNS ROWCOUNT angegeben ist, so wird implizit die Option RETURNS ROWCOUNT verwendet (zur Option selbst siehe auch [Abschnitt Rückgabewerte eines Skriptes](#)).
 - Ausgeführt wird ein Skript durch den Befehl **EXECUTE SCRIPT**.
- Hinweise zu benutzerdefinierten Funktionen (UDF-Skripte):
 - Benutzerdefinierte Funktionen (**UDF**) können innerhalb eines SELECT-Befehls verwendet werden und große Datenmengen verarbeiten.



Bitte beachten Sie, dass UDF Skripte Teil der Advanced Edition von Exasol ist.

- Neben skalaren Funktionen können Aggregats- und analytische Funktionen, aber auch MapReduce-Programme erstellt werden. Weitere Infos hierzu finden Sie in [Abschnitt 3.6, UDF Skripte](#).
- Wird die ORDER BY Klausel angegeben, so werden die Gruppen bei SET-Eingabewerte sortiert verarbeitet. Diese Klausel kann ebenfalls beim Aufruf eines Skriptes angegeben werden.
- Hinweise zu Adapter-Skripten:
 - Details zu Adapter-Skripten und virtuellen Schemas finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#).



Bitte beachten Sie, dass virtuelle Schemas Teil der Advanced Edition von Exasol ist.

- Die existierenden Open Source Adapter finden Sie in unserem GitHub Repository: <https://www.github.com/exasol>
- Adapter-Skripte können nur in Java oder Python implementiert werden.

Beispiel(e)

```
-- define a reusable function definition
CREATE SCRIPT function_lib AS
    function min_max(a,b,c)
        local min,max
        if a>b then max,min=a,b
            else max,min=b,a
        end
        if c>max then max=c
        else if c<min then min=c
        end
    end
    return min,max
end
/

-- scripting program example for data insert
CREATE SCRIPT insert_low_high (param1, param2, param3) AS
    import('function_lib') -- accessing external function
    lowest, highest = function_lib.min_max(param1, param2, param3)
    query([[INSERT INTO t VALUES (:x, :y)]], {x=lowest, y=highest})
/


EXECUTE SCRIPT insert_low_high(1, 3, 4);
EXECUTE SCRIPT insert_low_high(2, 6, 4);
EXECUTE SCRIPT insert_low_high(3, 3, 3);

-- UDF example
CREATE LUA SCALAR SCRIPT my_average (a DOUBLE, b DOUBLE)
    RETURNS DOUBLE AS
function run(ctx)
    if ctx.a == nil or ctx.b==nil
        then return NULL
    end
    return (ctx.a+ctx.b)/2
end
/
SELECT x,y,my_average(x,y) FROM t;

X          Y          MY_AVERAGE(T.X,T.Y)
-----  -----  -----
      1          4          2.5
      2          6          4
      3          3          3

-- Adapter script example
CREATE JAVA ADAPTER SCRIPT my_script AS
    %jar hive_jdbc_adapter.jar
/
```

DROP SCRIPT

Zweck

Mit diesem Kommando kann ein Datenbank-Skript gelöscht werden (UDF, Scripting oder Adapter).

Vorbedingung(en)

- System-Privileg DROP ANY SCRIPT oder das Skript gehört dem aktuellen User (also das gesamte Schema).

Syntax

drop_script ::=



Anmerkung(en)

- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls das Skript nicht existiert.
- Bei Adapter Skripten wird eine Fehlermeldung geworfen, falls noch virtuelle Schemas existieren, die auf dieses Skript verweisen.

Beispiel(e)

```
DROP SCRIPT my_script;
```

RENAME

Zweck

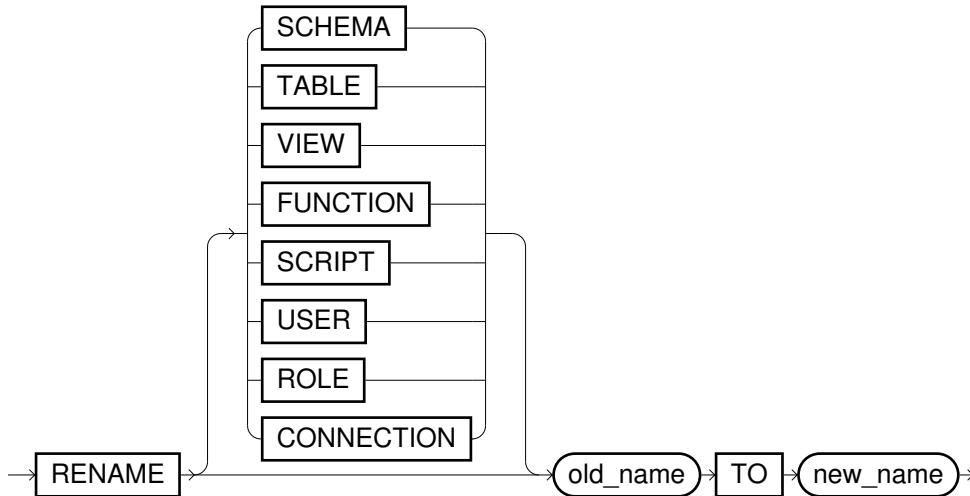
Dieser Befehl ermöglicht die Umbenennung von Schemas und Schemaobjekten.

Vorbedingung(en)

- Falls das Objekt ein Schema ist, so muss es dem Nutzer oder einer seiner Rollen gehören.
- Falls das Objekt ein Schemaobjekt ist, so muss das Objekt dem Benutzer oder einer seiner Rollen gehören (also in einem eigenen Schema oder dem einer zugewiesenen Rolle liegen).
- Falls das Objekt ein Nutzer bzw. eine Rolle ist, so benötigt der Nutzer das Systemprivileg CREATE USER bzw. CREATE ROLE.
- Falls das Objekt eine Verbindung ist, so benötigt der Nutzer entweder das Systemprivileg ALTER ANY CONNECTION oder er muss die Verbindung mit der WITH ADMIN OPTION erhalten haben.

Syntax

rename ::=



Anmerkung(en)

- Schemaobjekte können durch den RENAME-Befehl nicht in andere Schemas verschoben werden. D.h. "RENAME TABLE s1.t1 TO s2.t2" ist nicht erlaubt.
- Die Unterscheidung zwischen Schema, Tabelle, etc. ist optional und nur dann notwendig, falls zwei gleiche Schemaobjekte mit identischem Namen existieren.

Beispiel(e)

```
RENAME SCHEMA s1 TO s2;
RENAME TABLE t1 TO t2;
RENAME s2.t3 TO t4;
```

COMMENT

Zweck

Dieser Befehl ermöglicht das Setzen von Kommentaren zu Schemas und Schemaobjekten.

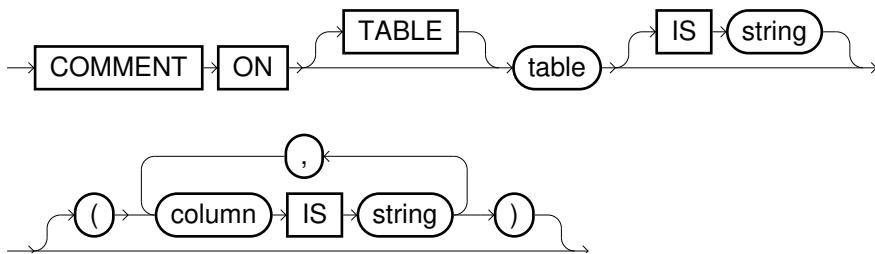
Vorbedingung(en)

- Voraussetzung, dass ein Nutzer ein Objekt kommentieren kann:

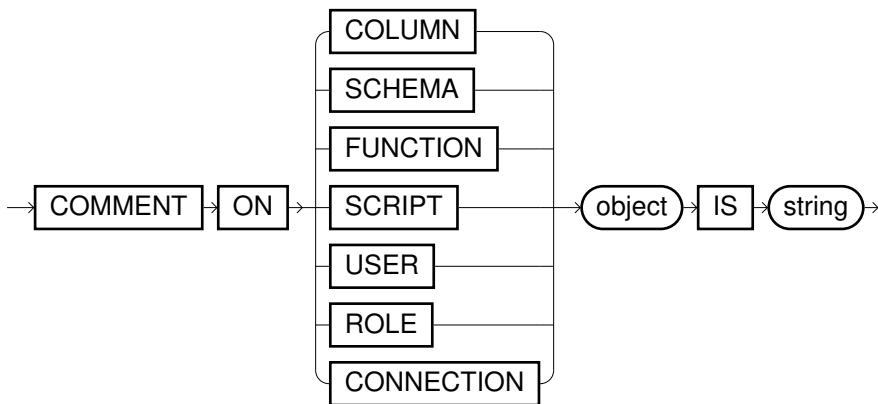
Objekttyp	Vorbedingung
Schema	Besitzer oder Rolle DBA
Tabelle bzw. Tabellenspalte	Besitzer oder Rolle DBA, ALTER-Objektpivileg auf Tabelle oder ALTER ANY TABLE.
Funktion	Besitzer oder Rolle DBA
Skript	Besitzer oder Rolle DBA
Nutzer	Rolle DBA, Systemprivileg ALTER USER oder Systemprivileg CREATE USER
Rolle	Rolle DBA oder Systemprivileg CREATE ROLE
Verbindung	Rolle DBA oder Systemprivileg CREATE CONNECTION

Syntax

comment_table ::=



comment_object ::=



Anmerkung(en)

- In den entsprechenden Systemtabellen zu den Objekten (z.B. [EXA_ALL_OBJECTS](#), [EXA_ALL_TABLES](#), ...) sowie mit dem Befehl [DESC\[RIBE\]](#) (mit der FULL-Option) werden Kommentare angezeigt.
- Kommentare können gelöscht werden, indem einem Object NULL bzw. der leere String zugewiesen wird.
- Kommentare können auch im [CREATE TABLE](#) Statement definiert werden.
- View-Kommentare können nur im [CREATE VIEW](#) Statement definiert werden.

Beispiel(e)

```

COMMENT ON SCHEMA s1 IS 'My first schema';
COMMENT ON TABLE t1 IS 'My first table';
COMMENT ON t1 (id IS 'Identity column', zip IS 'Zip code');
COMMENT ON SCRIPT script1 IS 'My first script';
  
```

2.2.2. Manipulation der Datenbank (DML)

Durch die Data Manipulation Language (DML) können die Inhalte von Tabellen geändert werden.

INSERT

Zweck

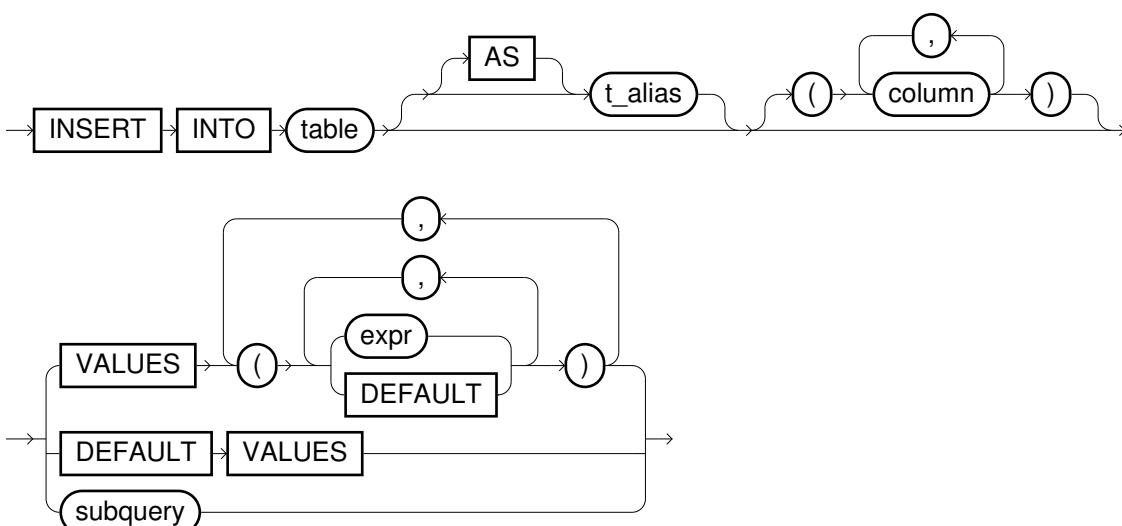
Mit diesem Befehl ist es dem Benutzer möglich, in eine Tabelle sowohl konstante Werte als auch das Ergebnis einer Subquery einzufügen.

Vorbedingung(en)

- System-Privileg INSERT ANY TABLE oder Objekt-Privileg INSERT auf die Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.
- Falls das Ergebnis einer Subquery eingefügt werden soll, so benötigt der Nutzer entsprechende SELECT-Rechte auf die in der Subquery referenzierten Objekte.

Syntax

insert ::=



Anmerkung(en)

- Die Anzahl an Ziel-Spalten der Ziel-Tabelle muss mit der Anzahl an Konstanten bzw. mit der Anzahl der SELECT-Spalten der Subquery übereinstimmen. Ansonsten wird eine Exception geworfen.
- Wird nur eine bestimmte Teilmenge von Ziel-Spalten (<column_list>) für die Zieltabelle angegeben, so werden die restlichen Spalten automatisch aufgefüllt. Bei Identity-Spalten wird ein aufsteigender numerischer Wert und bei Spalten mit Default-Wert der entsprechende Default-Wert eingefügt. Bei allen sonstigen Spalten wird der Wert NULL eingefügt.
- Ist für eine Spalte bei INSERT INTO t VALUES der 'Wert' DEFAULT angegeben, so verhält sich dies wie das implizite Einfügen (siehe oben).
- INSERT INTO t DEFAULT VALUES verhält sich so, als ob für jeden einzelnen Spalte das Literal DEFAULT angegeben worden wäre.
- Details zu Default-Werten finden Sie unter [Abschnitt 2.3.5, Default-Werte](#), zu Identity-Spalten unter [Abschnitt 2.3.6, Identity-Spalten](#).

- Details zur Syntax von Subqueries finden Sie in der Dokumentation zum **SELECT** Befehl in [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Beispiel(e)

```
INSERT INTO t VALUES (1, 2.34, 'abc');
INSERT INTO t VALUES (2, 1.56, 'ghi'), (3, 5.92, 'pqr');
INSERT INTO t VALUES (4, DEFAULT, 'xyz');
INSERT INTO t (i,k) SELECT * FROM u;
INSERT INTO t (i) SELECT max(j) FROM u;
INSERT INTO t DEFAULT VALUES;
```

UPDATE

Zweck

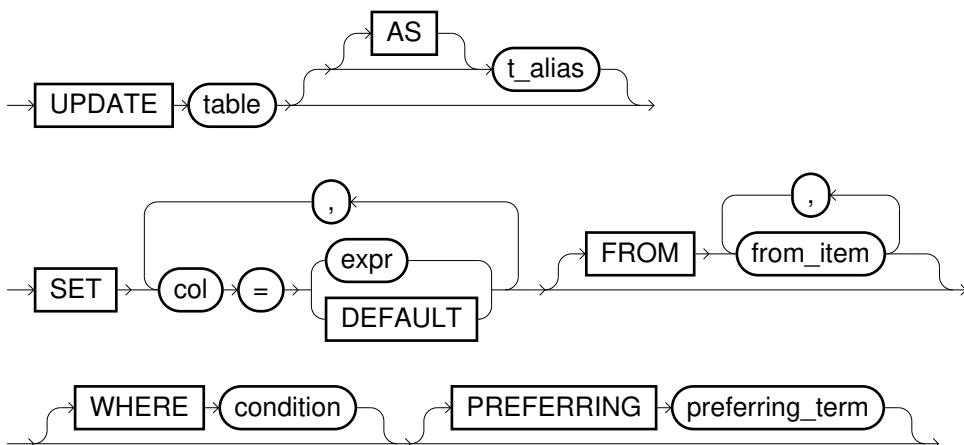
Der UPDATE-Befehl ermöglicht es, die Inhalte einer Tabelle gezielt zu verändern. Durch die Einschränkung der WHERE-Bedingung ist es sogar möglich, nur einen Spalteneintrag einer einzigen Zeile zu ändern.

Vorbedingung(en)

- System-Privileg **UPDATE ANY TABLE** oder Objekt-Privileg **UPDATE** auf der Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.
- Entsprechende **SELECT**-Privilegien auf die in der optionalen **FROM**-Klausel verwendeten Objekte.

Syntax

update ::=



Anmerkung(en)

- Durch die **FROM**-Klausel können mehrere Tabellen definiert werden, die über die **WHERE**-Bedingung verknüpft werden. Dadurch lassen sich komplexere Update-Bedingungen definieren, analog zu einem entsprechenden komplexen **SELECT**-Statement. Bitte beachten sie, dass in diesem Fall die zu verändernde Tabelle in der **FROM**-Klausel angegeben sein muss.

Intern wird dieses Statement in ein **MERGE**-Statement umgewandelt, daher können sich Fehlermeldungen auf das **MERGE**-Kommando beziehen. Würde eine Zeile

mehrfach verändert werden, so muss der neue Wert stets gleich sein, ansonsten führt dies zu der Fehlermeldung "Unable to get a stable set of rows in the source tables".

- Wird eine Spalte auf den 'Wert' DEFAULT gesetzt, wird in den Zeilen, die durch das UPDATE betroffen sind, bei Identity-Spalten ein aufsteigender numerischer Wert und bei Spalten mit Default-Wert der entsprechende Default-Wert eingesetzt.
- Details zu Default-Werten finden Sie unter [Abschnitt 2.3.5, Default-Werte](#), zu Identity-Spalten unter [Abschnitt 2.3.6, Identity-Spalten](#).
- Die PREFERRING Klausel definiert einen Skyline Präferenzausdruck. Details hierzu finden Sie in [Abschnitt 3.10, Skyline](#).

Beispiel(e)

```
--Gehaltserhöhung um 10 %
UPDATE staff SET salary=salary*1.1 WHERE name='SMITH';

--Euro-Umstellung
UPDATE staff AS U SET U.salary=U.salary/1.95583, U.currency='EUR'
    WHERE U.currency='DM';

--Komplexes UPDATE mit Join auf andere Tabelle
UPDATE staff AS U
SET U.salary=V.salary, U.currency=V.currency
FROM staff AS U, staff_updates AS V
WHERE U.name=V.name;
```

MERGE

Zweck

Mit dem MERGE-Befehl ist es möglich, die Inhalte einer Änderungstabelle in eine Zieltabelle zu übernehmen. Die Zeilen der Aktualisierungstabelle bestimmen, welche Zeilen geändert, gelöscht bzw. eingefügt werden. Somit vereint der MERGE-Befehl die drei Befehle [UPDATE](#), [DELETE](#) und [INSERT](#).

Beispielsweise kann die Aktualisierungstabelle sowohl die Daten neuer bzw. zu löschernder Kunden oder die Änderungsinformationen bereits existierender Kunden beinhalten. Durch den MERGE-Befehl können nun die neuen Kunden in die Zieltabelle eingefügt, invalide Kunden gelöscht und die Änderungen der existierenden Kunden übernommen werden.

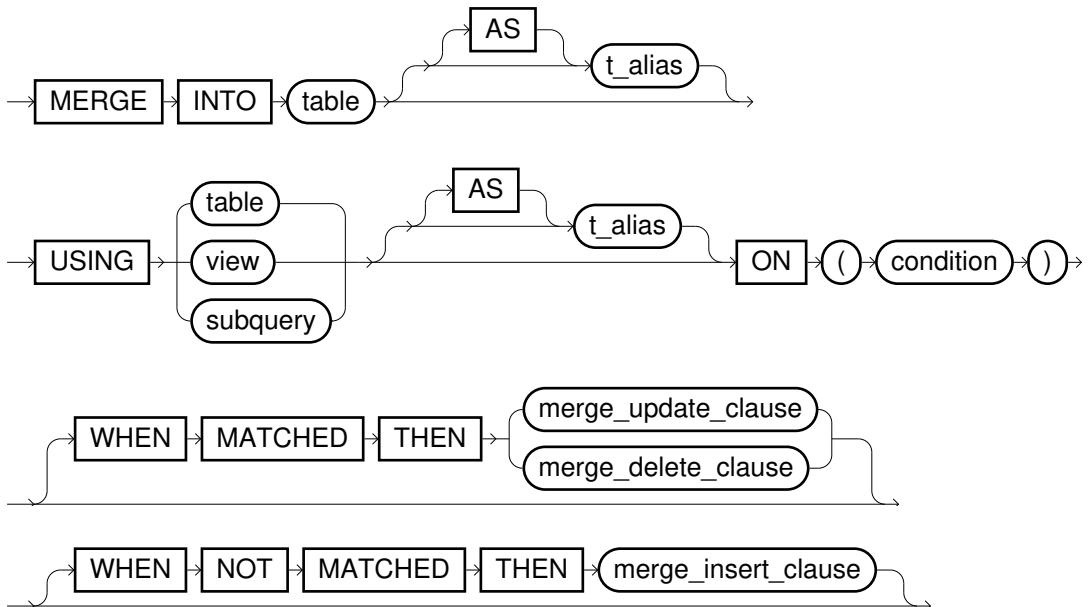
Die UPDATE- bzw. DELETE- und INSERT-Klauseln sind optional, so dass jederzeit nur ein Teil der oben beschriebenen Aktionen möglich ist. Insgesamt bildet das MERGE-Kommando somit ein mächtiges Instrument zur Manipulation der Datenbank.

Vorbedingung(en)

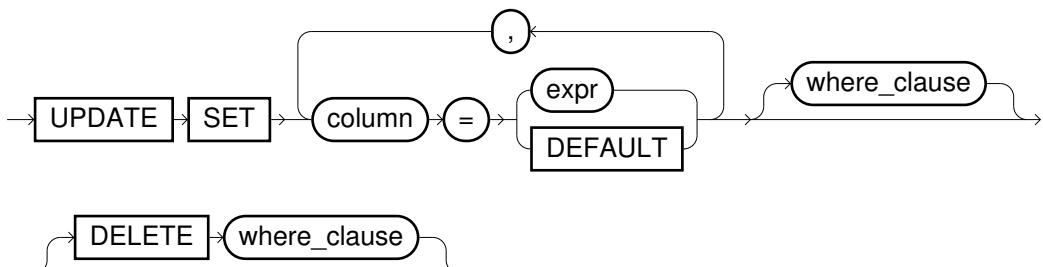
- Entsprechende INSERT-, DELETE- und UPDATE-Privilegien auf die Zieltabelle.
- Entsprechende SELECT-Privilegien auf die Aktualisierungstabelle.

Syntax

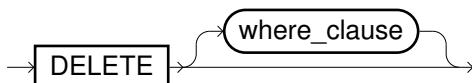
```
merge::=
```



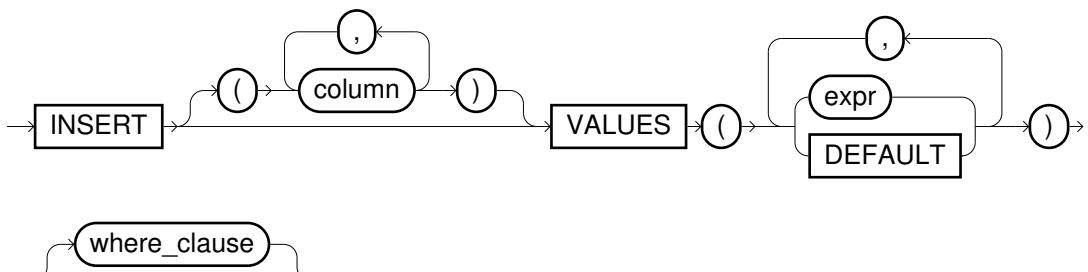
merge_update_clause ::=



merge_delete_clause ::=



merge_insert_clause ::=



Anmerkung(en)

- Die **ON-Bedingung** beschreibt die Verknüpfung zwischen den beiden Tabellen (ähnlich einem Join). Für die übereinstimmenden Zeilenpaare wird die MATCHED-Klausel angewendet, für nicht übereinstimmende die NOT MATCHED-Klausel. In der ON-Bedingung sind nur Äquivalenzbedingungen (=) erlaubt.
- UPDATE-Klausel:** Die optionale WHERE-Bedingung gibt an, unter welchen Umständen das UPDATE durchgeführt wird, wobei hierfür sowohl die Zieltabelle als auch die Änderungstabelle referenziert werden

darf. Mit Hilfe der optionalen DELETE-Bedingung können Zeilen der Zieltabelle gelöscht werden. Es werden nur die geänderten Zeilen betrachtet und zur Überprüfung die Werte *nach* dem UPDATE verwendet.

- **DELETE-Klausel:** Die optionale WHERE-Bedingung gibt an, unter welchen Umständen das DELETE durchgeführt wird.
- **INSERT-Klausel:** Die optionale WHERE-Bedingung gibt an, unter welchen Umständen das INSERT durchgeführt wird. Hierfür dürfen nur Spalten der Änderungstabelle referenziert werden.
- Die Änderungstabelle kann eine materielle Tabelle, eine View oder eine Subquery sein.
- Die UPDATE-, DELETE- bzw. INSERT-Klauseln sind optional mit der Einschränkung, dass mindestens eine angegeben wird. Die Reihenfolge der Klauseln kann auch vertauscht werden.
- Default-Werte und Identity-Spalten werden von der INSERT- und UPDATE-Klausel genauso behandelt wie vom INSERT- und UPDATE-Statement (siehe dort), mit der einzigen Ausnahme, dass hier kein INSERT DEFAULT VALUES erlaubt ist.
- Existieren in der Änderungstabelle mehrere Einträge, die für ein UPDATE einer Zeile in der Zieltabelle in Frage kommen, so führt dies zu der Fehlermeldung "Unable to get a stable set of rows in the source tables", wenn der Originalwert der Zieltabelle durch die UPDATE-Kandidaten verändert werden würde.
- Existieren in der Änderungstabelle mehrere Einträge, die für ein DELETE einer Zeile in der Zieltabelle in Frage kommen, so führt dies zu der Fehlermeldung "Unable to get a stable set of rows in the source tables".

Beispiel(e)

```
/* Sample tables
staff:                                changes:          deletes:
 name   | salary | lastChange      name   | salary      name
-----+-----+-----+-----+-----+-----+-----+
 meier | 30000 | 2006-01-01    schmidt | 43000      meier
 schmidt | 40000 | 2006-05-01   hofmann | 35000
 mueller | 50000 | 2005-08-01   meier   | 29000
 */

-- Merging the table updates
MERGE INTO staff T
    USING changes U
    ON (T.name = U.name)
    WHEN MATCHED THEN UPDATE SET T.salary = U.salary,
                                T.lastChange = CURRENT_DATE
                                WHERE T.salary < U.salary
    WHEN NOT MATCHED THEN INSERT VALUES (U.name,U.salary,CURRENT_DATE);

SELECT * FROM staff;

NAME           SALARY        LASTCHANGE
-----+-----+-----+
meier          30000 2006-01-01
schmidt        43000 2010-10-06
mueller        50000 2005-08-01
hofmann        35000 2010-10-06

-- Merging the table deletes
MERGE INTO staff T
    USING deletes U
    ON (T.name = U.name)
    WHEN MATCHED THEN DELETE;

SELECT * FROM staff;

NAME           SALARY        LASTCHANGE
-----+-----+-----+
```

hofmann	35000	2010-10-06
schmidt	43000	2010-10-06
mueller	50000	2005-08-01

DELETE

Zweck

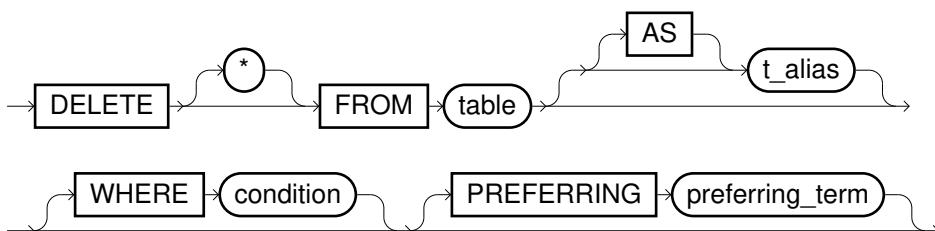
Mit dem DELETE-Befehl ist es möglich, Zeilen einer Tabelle zu löschen.

Vorbedingung(en)

- System-Privileg DELETE ANY TABLE oder Objekt-Privileg DELETE auf der Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

delete::=



Anmerkung(en)

- Zeilen werden intern nicht sofort gelöscht, sondern lediglich intern als gelöscht markiert. Erst nach Erreichen eines gewissen Schwellwertes (standardmäßig bei 25% der Zeilen) werden diese Daten endgültig entfernt. Daher können DELETE-Statements unterschiedlich lange dauern. Den aktuellen Prozentsatz der als gelöscht markierten Zeilen finden Sie in den Systemtabellen EXA_*_TABLES (z.B. EXA_ALL_TABLES) über die Spalte DELETE_PERCENTAGE.
- Die PREFERRING Klausel definiert einen Skyline Präferenzausdruck. Details hierzu finden Sie in [Abschnitt 3.10, Skyline](#).

Beispiel(e)

```
DELETE FROM staff WHERE name='SMITH';
DELETE FROM staff;
```

TRUNCATE

Zweck

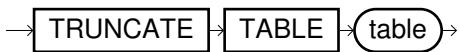
Mit dem TRUNCATE-Befehl ist es möglich, den Inhalt einer Tabelle komplett zu löschen.

Vorbedingung(en)

- System-Privileg DELETE ANY TABLE oder Objekt-Privileg DELETE auf der Tabelle bzw. deren Schema oder die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

truncate ::=



Beispiel(e)

```
TRUNCATE TABLE staff;
```

IMPORT

Zweck

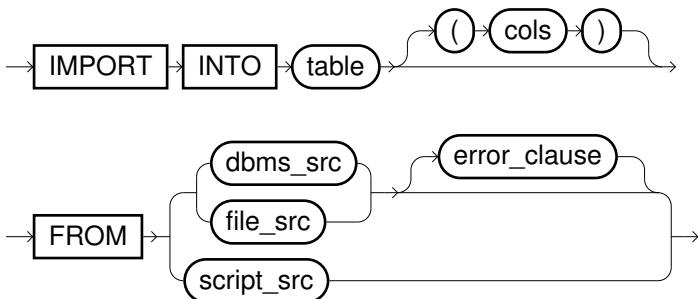
Mit Hilfe des IMPORT-Befehls lassen sich Daten aus externen Dateien oder Datenbanksystemen in eine Tabelle importieren.

Vorbedingung(en)

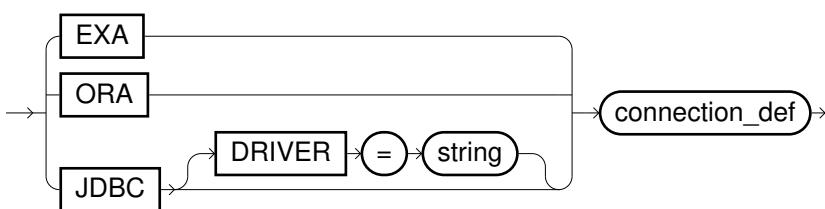
- Im Quellsystem: Entsprechende Privilegien zum Lesen der Tabelleninhalte bzw. Dateien
- In Exasol: Entsprechende Privilegien zum Einfügen von Zeilen (siehe [INSERT](#))
- Bei Verwendung einer Verbindung (siehe auch [CREATE CONNECTION](#) in Abschnitt 2.2.3, Zugriffssteuerung mittels SQL (DCL)) entweder das Systemprivileg USE ANY CONNECTION oder die Verbindung muss mittels [GRANT](#) an den Nutzer oder eine seiner Rollen zugewiesen worden sein
- Bei Verwendung der Fehler-Tabelle müssen im Zielsystem entsprechende Rechte zum Einfügen bzw. Schreiben bestehen

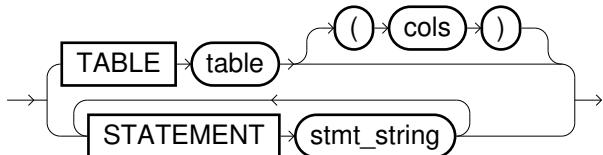
Syntax

import ::=

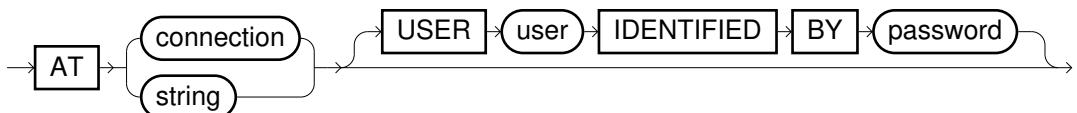


dbms_src ::=

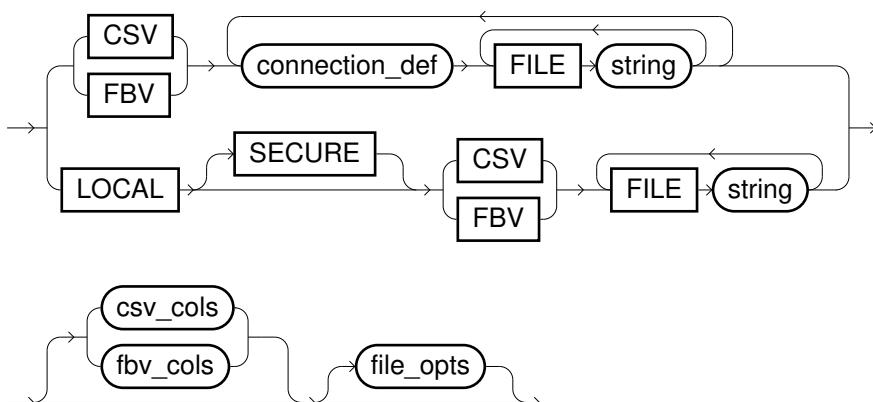




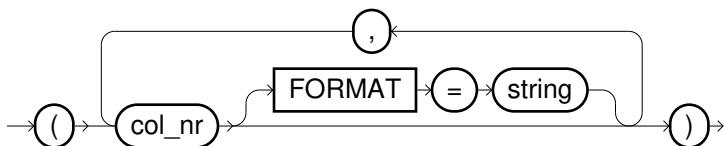
connection_def:=



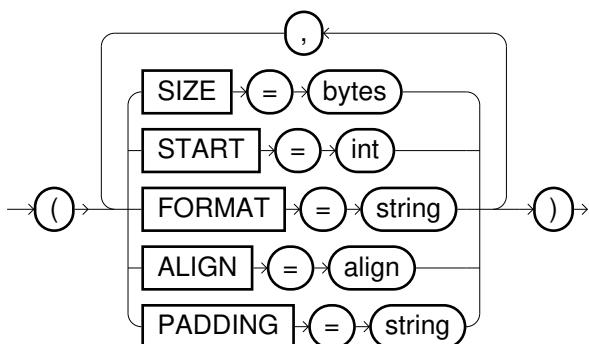
file_src:=



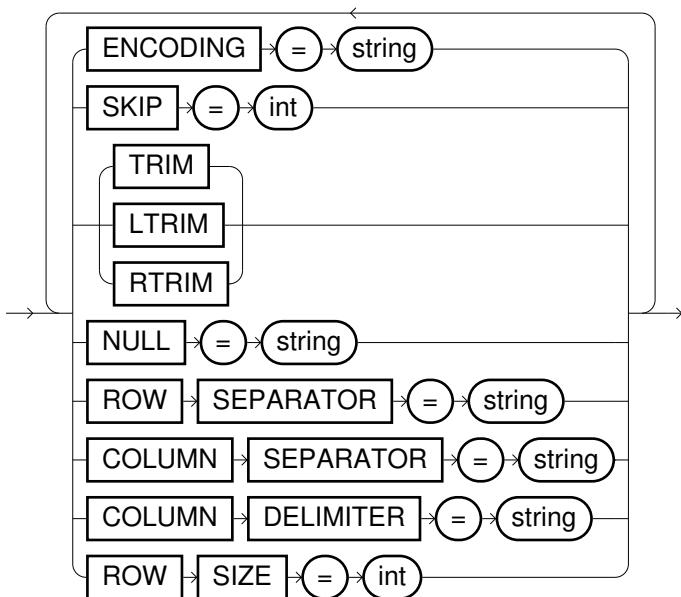
csv_cols:=



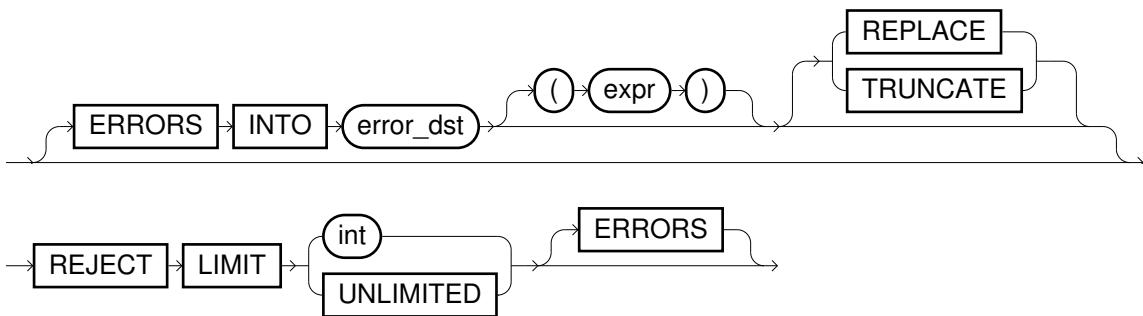
fbv_cols:=



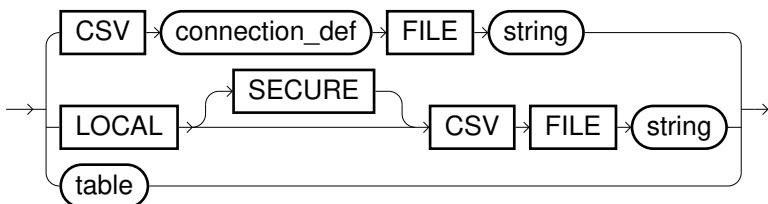
file_opts:=



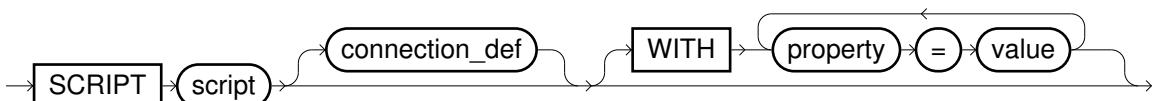
error_clause:=



error_dst:=



script_src:=



Anmerkung(en)

- Weitere Information zum Thema ETL finden Sie unter [Abschnitt 3.4, ETL-Prozesse](#).
- Der aktuelle Status des Ladevorgangs kann in einer zweiten Verbindung zur Datenbank über die Tabelle `EXA_USER_SESSIONS` ausgelesen werden (Spalte ACTIVITY).
- Import-Befehle können auch transparent in SELECT-Anfragen integriert werden. Weitere Infos hierzu finden sie unter [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

- Bitte beachten Sie, dass beim IMPORT von JDBC- und CSV-Quellen Dezimalzahlen abgerundet werden (truncate), sofern der Ziel-Datentyp weniger Präzision hat als der Quell-Datentyp.
- Übersicht über die verschiedenen Elemente und deren Bedeutung:

Element	Beschreibung
dbms_src	<p>Definiert die Datenbankquelle, aus der importiert werden soll. Die Verbindungsdaten werden über die connection_def spezifiziert (siehe unten). Zur Verfügung stehen entweder eine Exasol-Instanz (EXA), eine native Anbindung zu einer Oracle-Datenbank (ORA) oder eine JDBC-Verbindung zu beliebigen Datenbanken (JDBC).</p> <p>Defaultmäßig stehen bereits einige JDBC-Treiber zur Verfügung, die Sie unter EXAoperation einsehen und im Verbindungs-String adressieren können (z.B. jdbc:mysql, jdbc:postgres). Sie können aber über EXAoperation auch eigene Treiber konfigurieren und mittels der DRIVER-Option auswählen, falls dessen Präfix nicht schon eindeutig sein sollte.</p> <p> Nur die bereits vorinstallierten, in EXAoperation grau markierten JDBC-Treiber werden von uns getestet und offiziell unterstützt. Bei Problemen mit anderen Treibern wird unser Support versuchen, Ihnen entsprechend zu helfen.</p> <p>Für die Quelldaten kann entweder eine Datenbank-Tabelle (als Identifier, z.B. MY_SCHEMA.MY_TABLE) oder ein DB-Statement (als String, z.B. 'SELECT "TEST" FROM DUAL') angegeben werden. Im zweiten Fall wird dieser Ausdruck auf der Datenbank ausgeführt, also z.B. eine SQL-Query oder der Aufruf einer Datenbankprozedur.</p> <p> Bitte beachten Sie, dass im Fall einer Tabelle der Identifier genauso wie Exasol-Identifier behandelt werden. Daher müssen sie bei case-sensitiven Tabellen den Namen in Anführungszeichen setzen.</p> <p>Bitte beachten Sie folgendes zur optimalen Parallelisierung des Ladens:</p> <ul style="list-style-type: none"> • Beim Importieren von Exasol Datenbanken wird immer parallel geladen. Bitte beachten Sie, dass hier das Laden von Tabellen deutlich schneller ist als die Verwendung von DB-Statements. • Wenn Sie aus Oracle-Quellen laden, wird für partitionierte Tabellen automatisch parallel geladen. • Die Angabe von mehreren Statements ist nur für JDBC- und Oracle-Quellen möglich.
file_src	<p>Spezifiziert die Dateiquelle.</p> <ol style="list-style-type: none"> 1. Remote Dateien <p>Es werden FTP-, FTPS-, SFTP, HTTP- und HTTPS-Server unterstützt, deren Verbindungsdaten über die connection_def angegeben werden (siehe unten).</p> <p>Anmerkungen:</p> <ul style="list-style-type: none"> • Bei verschlüsselten Verbindungen findet keine Überprüfung der Zertifikate statt. • Falls anstatt einer Datei ein Verzeichnis angegeben wird, so wird die Liste der darin enthaltenen Dateien importiert, sofern der Server dies unterstützt. • Bei URLs beginnend mit 'ftps:// ' wird von der Verwendung der impliziten Verschlüsselung ausgegangen. • Bei URLs beginnend mit 'ftp:// ' verschlüsselt Exasol Nutzernamen und Passwort (explizite Verschlüsselung), sofern dies vom Server unterstützt wird.

Element	Beschreibung
	<p>Im Fall, dass der Server zusätzlich eine verschlüsselte Datenübertragung erfordert, werden sämtliche Daten verschlüsselt übertragen.</p> <ul style="list-style-type: none"> • Für HTTP- bzw. HTTPS-Server unterstützt Exasol nur Basis-Authentifizierung. • Für HTTP- bzw. HTTPS-Server können HTTP Parameter angegeben werden, indem sie dem Dateinamen angefügt werden (z.B. FILE 'file.csv?op=OPEN&user.name=user'). <p>2. Lokale Dateien</p> <p>Mit dieser Option können auch lokale Dateien von einem Client-System importiert werden. Bei Angabe der SECURE-Option werden die Daten verschlüsselt, wodurch sich allerdings auch die Ladegeschwindigkeit verringert.</p> <p> Diese Funktionalität steht nur für EXAplus und den JDBC Treiber zur Verfügung und kann nicht als Prepared Statement oder aus Datenbank-Skripten heraus aufgerufen werden. Falls Sie eine lokale Datei über ein explizites Programm verarbeiten wollen, so können Sie das Tool EXAjload benutzen, welches im JDBC-Treiber-Paket mit ausgeliefert wird. Rufen Sie dieses Programm einfach ohne Parameter auf, um Informationen über die Verwendung zu erhalten.</p> <p> Zum Importieren von lokalen Dateien öffnet der JDBC Treiber eine interne Verbindung zum Cluster und emuliert einen HTTP- bzw. HTTPS- (SECURE-Option) Server, der genau diese Datei bereitstellt. Für den Nutzer ist dies allerdings völlig transparent.</p> <p>Die Quell-Dateien können CSV- bzw. FBV-Dateien sein, deren Format-Spezifikationen in Das CSV Datenformat und Das Fixblock Datenformat (FBV) zu finden sind. Im Dateinamen sind nur ASCII-Zeichen erlaubt. BOM wird nicht unterstützt.</p> <p>Komprimierte Daten werden anhand der Dateiendung erkannt, unterstützt werden dabei die Endungen .zip, .gz (gzip) und .bz2 (bzip2).</p> <p>Lautet der Dateiname System.in, dann werden die Daten vom Standardeingabestrom gelesen (System.in).</p>
script_src	<p>Spezifiziert das UDF Skript, das für den benutzerdefinierten Import verwendet werden soll. Sie können optional Verbindungsinformationen oder eine Liste von Parametern angeben, die an das Skript weitergegeben werden. Das angegebene Skript muss eine spezielle Funktion implementieren die von der Datenbank aufgerufen wird und die Spezifikation des Imports erhält (z.B. Informationen zu Verbindung und Parametern). Das Skript generiert dann einen SQL Befehl, der intern ein <code>INSERT INTO SELECT</code> ausführt. Eine ausführliche Erklärung von benutzerdefinierten Imports mit Beispielen finden Sie in Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs.</p> <p>connection_def</p> <p>Optionale Angabe einer Verbindung zur Möglichkeit, die Verbindungsdaten wie das Passwort zu kapseln. Bitte lesen Sie den separaten Eintrag in dieser Tabelle über die genaue Syntax.</p> <p>WITH parameter=value ...</p>

Element	Beschreibung								
	<p>Optionale Angabe von Parametern, die an das Skript weitergegeben werden. Jedes Skript kann selbst festlegen, welche Parameter es erwartet. Parameter sind Schlüssel-Wert Paare, wobei der Wert ein String ist, z.B.:</p> <pre data-bbox="509 377 1160 406">... WITH PARAM_1='val1' PARAM_2 = 'val2';</pre>								
connection_def	<p>Definiert eine Verbindung zu einer Datenbank- oder einem Datei-Server. Dieser kann über einen Connection-String definiert werden, in der die Verbindungsdaten spezifiziert sind (z.B. 'ftp://192.168.1.1/' sowie die entsprechenden Login-Daten).</p> <p>Zur regelmäßigen Benutzung einer externen Verbindung bietet sich allerdings die Verwendung von Connections an, in denen die Verbindungsdaten sowie Benutzer und evtl. das Passwort bequem gekapselt werden können. Für nähere Informationen und Beispiele zu Connection-Objekten siehe auch CREATE CONNECTION in Abschnitt 2.2.3, Zugriffssteuerung mittels SQL (DCL).</p> <p>Die Angabe von Nutzer und Passwort innerhalb des IMPORT-Befehls sind optional. Falls diese nicht angegeben sind, so werden die Daten des verwendeten Connection-Strings bzw. des Connection-Objekts benutzt.</p> <p>Für JDBC Verbindungen ist eine Kerberos Authentifizierung möglich, indem bestimmte Daten über das IDENTIFIED BY Feld definiert werden. Diese bestehen aus einem Indikator, dass Kerberos benutzt werden soll (ExaAuthType=Kerberos), einer base64 kodierten Konfigurations-Datei und einer base64 kodierten keytab Datei mit den Credentials des Principal. Die Syntax sieht wie folgt aus: IMPORT INTO table1 FROM JDBC AT '<JDBC_URL>' USER '<kerberos_principal>' IDENTIFIED BY 'ExaAuthType=Kerberos;<base64_krb_conf>;<base64_keytab>' TABLE table2; Weitere Details und Beispiele finden Sie in unserem Solution Center: https://wwwexasol.com/portal/display/SOL-512</p>								
csv_cols	<p>Definiert, welche Spalten in der CSV-Datei wie interpretiert werden sollen. Bitte beachten Sie auch Das CSV Datenformat.</p> <table> <tr> <td data-bbox="504 1298 584 1423">col_nr</td><td data-bbox="584 1298 1433 1423">Definiert die Spaltennummer (beginnend bei 1). Alternativ können sie mittels ... auch einen fortlaufenden Bereich von Spalten definieren (z.B. 5..8 für die Spalten 5,6,7,8). Bitte beachten Sie, dass die Spaltennummern stets aufsteigend sein müssen!</td></tr> <tr> <td data-bbox="504 1446 584 1545">FORMAT</td><td data-bbox="584 1446 1433 1545">Optionale Formatangabe für Zahlen oder Zeitstempel-/Datums-Werte (Default: Session-Format). Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle.</td></tr> <tr> <td data-bbox="504 1574 584 1641"></td><td data-bbox="584 1574 1433 1641">Im folgenden Beispiel werden die ersten 4 Spalten der CSV-Datei geladen, wobei für die letzte ein explizites Datumsformat definiert ist:</td></tr> <tr> <td data-bbox="504 1671 584 1700"></td><td data-bbox="584 1671 1433 1700">(1..3,4 FORMAT='DD-MM-YYYY')</td></tr> </table>	col_nr	Definiert die Spaltennummer (beginnend bei 1). Alternativ können sie mittels ... auch einen fortlaufenden Bereich von Spalten definieren (z.B. 5..8 für die Spalten 5,6,7,8). Bitte beachten Sie, dass die Spaltennummern stets aufsteigend sein müssen!	FORMAT	Optionale Formatangabe für Zahlen oder Zeitstempel-/Datums-Werte (Default: Session-Format). Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle .		Im folgenden Beispiel werden die ersten 4 Spalten der CSV-Datei geladen, wobei für die letzte ein explizites Datumsformat definiert ist:		(1..3,4 FORMAT='DD-MM-YYYY')
col_nr	Definiert die Spaltennummer (beginnend bei 1). Alternativ können sie mittels ... auch einen fortlaufenden Bereich von Spalten definieren (z.B. 5..8 für die Spalten 5,6,7,8). Bitte beachten Sie, dass die Spaltennummern stets aufsteigend sein müssen!								
FORMAT	Optionale Formatangabe für Zahlen oder Zeitstempel-/Datums-Werte (Default: Session-Format). Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle .								
	Im folgenden Beispiel werden die ersten 4 Spalten der CSV-Datei geladen, wobei für die letzte ein explizites Datumsformat definiert ist:								
	(1..3,4 FORMAT='DD-MM-YYYY')								
fbv_cols	<p>Definiert, welche Spalten in der FBV-Datei wie interpretiert werden sollen. Bitte beachten Sie auch Das Fixblock Datenformat (FBV).</p> <p>Folgende Elemente können hierbei angegeben werden:</p> <table> <tr> <td data-bbox="504 1888 584 1949">SIZE</td><td data-bbox="584 1888 1433 1949">Definiert die Anzahl an Bytes der Spalte und muss immer angegeben werden.</td></tr> </table>	SIZE	Definiert die Anzahl an Bytes der Spalte und muss immer angegeben werden.						
SIZE	Definiert die Anzahl an Bytes der Spalte und muss immer angegeben werden.								

Element	Beschreibung
	<p>START Anfangsbyte der Spalte (beginnend bei 0). Beachten Sie, dass die START-Werte der einzelnen Spalten aufsteigend sein müssen!</p> <p>FORMAT Optionale Formatangabe für Zahlen oder Zeitstempel-/Datums-Werte (Default: Session-Format). Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle.</p> <p>ALIGN Ausrichtung der Spalte (LEFT bzw. RIGHT), Default ist LEFT.</p> <p>PAD-DING Füllzeichen für Spalten, Default ist ein Leerzeichen (Space). Angeben kann man ein ASCII-Zeichen, entweder im Klartext (Bsp: '+'), als Hexadezimalwert (Bsp: '0x09') oder als Abkürzung ('CR','ESC','LF','NUL','TAB').</p> <p>Im Beispiel werden 4 Spalten der FBV-Datei geladen, wobei die erste Spalte nach rechts ausgerichtet ist und mit + aufgefüllt wird, nach den ersten 12 Bytes eine Lücke gelassen wird und die 4. Spalte dem angegebenen Datumsformat entspricht:</p> <pre>(SIZE=8 PADDING='+' ALIGN=RIGHT, SIZE=4, START=17 SIZE=8, SIZE=32 FORMAT='DD-MM-YYYY')</pre>
file_opts	<p>ENCODING Zeichensatz der CSV bzw. FBV Datei (Default ist UTF8). Alle unterstützten Zeichensätze siehe Anhang D, Unterstützte Zeichensätze.</p> <p>ROW SEPARATOR Zeilenumbruchzeichen: <ul style="list-style-type: none"> 'LF' (Default): entspricht dem ASCII-Zeichen 0x0a 'CR': entspricht dem ASCII-Zeichen 0x0d 'CRLF': entspricht den ASCII-Zeichen 0x0d und 0x0a 'NONE': kein Zeilenumbruchzeichen (nur bei FBV erlaubt) </p> <p>NULL Darstellung der NULL-Werte. Wird nichts explizit gesetzt, so werden NULL-Werte durch den leeren String dargestellt.</p> <p>SKIP Anzahl der zu überspringenden Zeilen. Dies ist sinnvoll, um eventuell vorhandene Header-Informationen zu ignorieren. Zu beachten ist, dass sich SKIP auf die Anzahl an Zeilenumbrüchen (ROW SEPARATOR) bezieht, auch wenn sich diese in Datensätzen befinden.</p> <p>TRIM, LTRIM, RTRIM Definiert, ob bei CSV-Spalten Leerzeichen abgeschnitten werden sollen (LTRIM: von links, RTRIM: von rechts, TRIM: von beiden Seiten). Defaultmäßig werden keine Leerzeichen abgeschnitten.</p> <p>COLUMN SEPARATOR Feldtrennzeichen für CSV-Dateien. Standardmäßig wird das Komma (,) verwendet. Angeben kann man einen String, entweder im Klartext (Bsp: ','), als Hexadezimalwert (Bsp: '0x09') oder als Abkürzung ('NUL', 'TAB', 'LF', 'CR' oder 'ESC'). Ein Klartext-String ist auf 10 Zeichen begrenzt, die zum ENCODING (siehe oben) der Datei automatisch konvertiert werden. Ein Hexadezimalwert ist auf 10 Bytes (nicht Zeichen) begrenzt, die nicht konvertiert werden.</p>

Element	Beschreibung				
	<p>COLUMN DELIMITER Feldbegrenzungszeichen für CSV-Dateien. Standardmäßig wird das Anführungszeichen ("") verwendet. Angeben kann man einen String, entweder im Klartext (Bsp: ""), als Hexadezimalwert (Bsp: '0x09') oder als Abkürzung ('NUL', 'TAB', 'LF', 'CR', 'ESC'). Ein Klartext-String ist auf 10 Zeichen begrenzt, die zum ENCODING (siehe oben) der Datei automatisch konvertiert werden. Ein Hexadezimalwert ist auf 10 Bytes (nicht Zeichen) begrenzt, die nicht konvertiert werden. Soll kein Feldbegrenzungszeichen verwendet werden, so kann ein leerer String angegeben werden ("").</p> <p>ROW SIZE Zeilenlänge für FBV-Dateien. Falls die letzte Spalte der FBV-Datei nicht benutzt wird, muss dieser Wert angegeben werden, um das Ende von Zeilen erkennen zu können. Ansonsten wird das Ende implizit über das Ende der hintersten Spalte berechnet, die im IMPORT-Befehl angegeben wurde (z.B. wird bei (SIZE=4 START=5) angenommen, dass eine Spalte mit 4 Bytes gelesen wird und die Zeile aus insgesamt 9 Bytes besteht).</p>				
error_clause	<p>Bei Angabe dieser Klausel kann die Anzahl an erlaubten fehlerhaften Zeilen in der Quelle definiert werden. Z.B. würde das Statement bei REJECT LIMIT 5 bei bis zu fünf fehlerhaften Zeilen erfolgreich durchlaufen, ab sechs Fehlern allerdings eine Exception werfen.</p> <p>Zusätzlich können Sie die fehlerhaften Zeilen in eine Datei bzw. lokale Tabelle innerhalb Exasol schreiben, um sie anschließend analysieren bzw. weiterverarbeiten zu können:</p> <table> <tr> <td>Tabelle</td> <td>Für jede fehlerhafte Zeile werden folgende Spalten erzeugt: Zeilennummer, Fehlermeldung, (Expression), Truncated Flag und die eigentlichen Daten. Das Truncated Flag gibt an, ob die Daten auf die maximale String-Länge gekürzt wurde.</td> </tr> <tr> <td>CSV Datei</td> <td>Für jede fehlerhafte Zeile wird eine Kommentarzeile mit Zeilennummer, Fehlermeldung sowie (Expression), gefolgt von der eigentlichen Daten-Zeile geschrieben.</td> </tr> </table> <p>Die (optionale) Expression dient der besseren Identifikation, falls Sie mehrfach in die gleiche Fehler-Tabelle bzw. -Datei einfügen. Hier könnten Sie beispielsweise CURRENT_TIMESTAMP verwenden.</p>	Tabelle	Für jede fehlerhafte Zeile werden folgende Spalten erzeugt: Zeilennummer, Fehlermeldung, (Expression), Truncated Flag und die eigentlichen Daten. Das Truncated Flag gibt an, ob die Daten auf die maximale String-Länge gekürzt wurde.	CSV Datei	Für jede fehlerhafte Zeile wird eine Kommentarzeile mit Zeilennummer, Fehlermeldung sowie (Expression), gefolgt von der eigentlichen Daten-Zeile geschrieben.
Tabelle	Für jede fehlerhafte Zeile werden folgende Spalten erzeugt: Zeilennummer, Fehlermeldung, (Expression), Truncated Flag und die eigentlichen Daten. Das Truncated Flag gibt an, ob die Daten auf die maximale String-Länge gekürzt wurde.				
CSV Datei	Für jede fehlerhafte Zeile wird eine Kommentarzeile mit Zeilennummer, Fehlermeldung sowie (Expression), gefolgt von der eigentlichen Daten-Zeile geschrieben.				

Beispiel(e)

```

IMPORT INTO table_1 FROM CSV
  AT 'http://192.168.1.1:8080/' USER 'agent_007' IDENTIFIED BY 'secret'
  FILE 'tab1_part1.csv' FILE 'tab1_part2.csv'
  COLUMN SEPARATOR = ';'
  SKIP = 5;

CREATE CONNECTION my_fileserver
  TO 'ftp://192.168.1.2/' USER 'agent_007' IDENTIFIED BY 'secret';

IMPORT INTO table_2 FROM FBV
  AT my_fileserver
  FILE 'tab2_part1.fbv'

```

```

        ( SIZE=8 PADDING='+' ALIGN=RIGHT,
        SIZE=4,
        SIZE=8,
        SIZE=32 FORMAT='DD-MM-YYYY' ) ;

CREATE CONNECTION my_oracle
  TO '(DESCRIPTION =
    (ADDRESS_LIST = (ADDRESS =
      (PROTOCOL = TCP)
      (HOST = 192.168.0.25)(PORT = 1521)
    )
  )
  (CONNECT_DATA = (SERVICE_NAME = orautf8)))
)';

IMPORT INTO table_3 (col1, col2, col4) FROM ORA
AT my_oracle
USER 'agent_008' IDENTIFIED BY 'secret'
STATEMENT ' SELECT * FROM orders WHERE order_state='OK' '
ERRORS INTO error_table (CURRENT_TIMESTAMP) REJECT LIMIT 10;

IMPORT INTO table_4 FROM JDBC
AT 'jdbc:exa:192.168.6.11..14:8563'
USER 'agent_008' IDENTIFIED BY 'secret'
STATEMENT ' SELECT * FROM orders WHERE order_state='OK' ';

IMPORT INTO table_5 FROM CSV
AT 'http://HadoopNode:50070/webhdfs/v1/tmp'
FILE 'file.csv?op=OPEN&user.name=user';

IMPORT INTO table_6 FROM CSV
AT 'https://testbucket.s3.amazonaws.com'
USER '<AccessKeyID>' IDENTIFIED BY '<SecretAccessKey>'
FILE 'file.csv';

IMPORT INTO table_7 FROM EXA
AT my_exasol
TABLE MY_SCHEMA.MY_TABLE;

IMPORT INTO table_8 FROM SCRIPT etl.import_hcat_table
WITH HCAT_DB      = 'default'
      HCAT_TABLE   = 'my_hcat_table'
      HCAT_ADDRESS = 'hcatalog-server:50111'
      HDFS_USER    = 'hdfs';IMPORT INTO table_9
FROM LOCAL CSV FILE '~/.my_table.csv'
COLUMN SEPARATOR = ';' SKIP = 5;

```

EXPORT

Zweck

Mit Hilfe des EXPORT-Befehls lassen sich Daten aus Exasol in externe Dateien oder Datenbanksysteme exportieren.

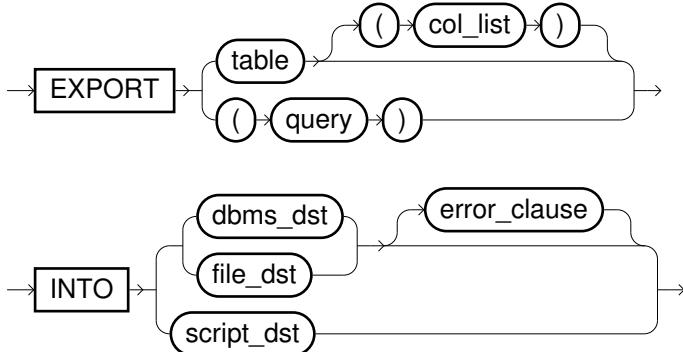
Vorbedingung(en)

- In Exasol: Entsprechende Privilegien zum Lesen der Tabelleninhalte.

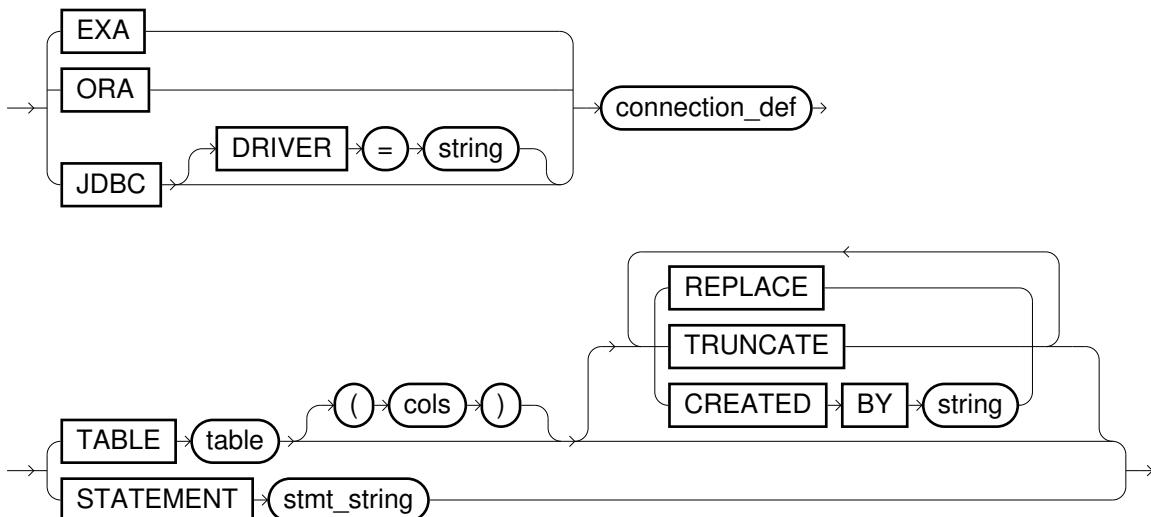
- Im Zielsystem: Entsprechende Privilegien zum Einfügen von Zeilen bzw. zum Schreiben von Dateien. Bei Angabe von gewissen Optionen benötigen Sie zudem Rechte zum Ersetzen des Ziels (REPLACE) bzw. Löschen der Daten (TRUNCATE).
- Bei Verwendung einer Verbindung (siehe auch [CREATE CONNECTION](#) in [Abschnitt 2.2.3, Zugriffssteuerung mittels SQL \(DCL\)](#)) entweder das Systemprivileg USE ANY CONNECTION oder die Verbindung muss mittels GRANT an den Nutzer oder eine seiner Rollen zugewiesen worden sein.

Syntax

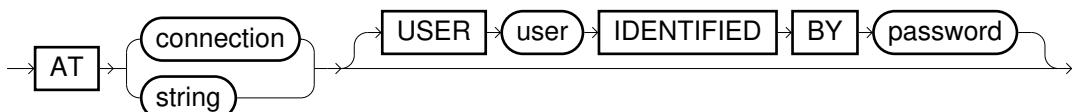
export ::=



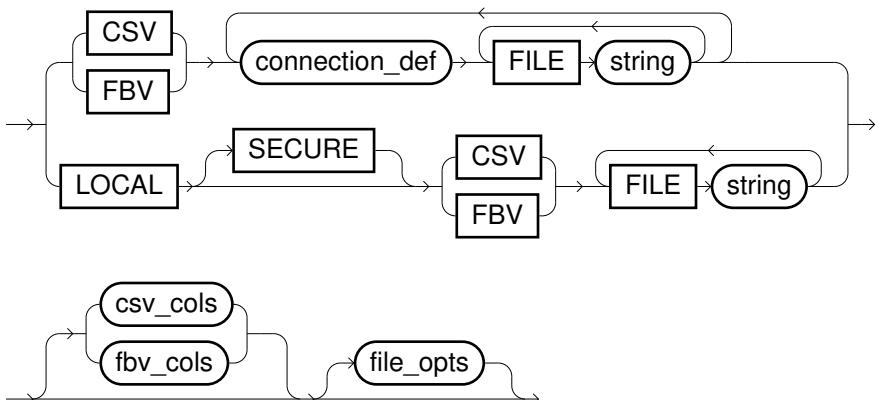
dbms_dst ::=



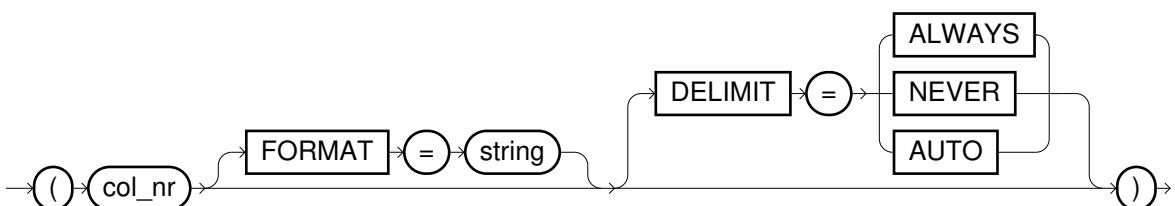
connection_def ::=



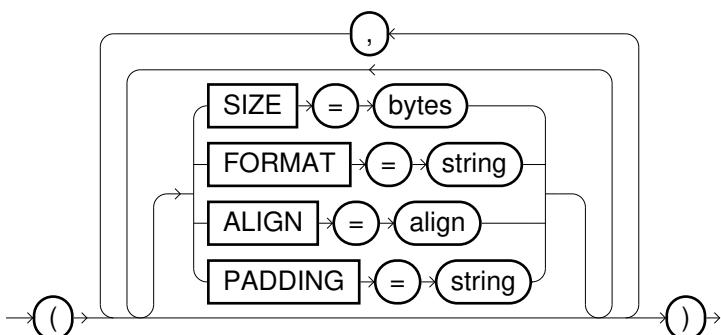
file_dst ::=



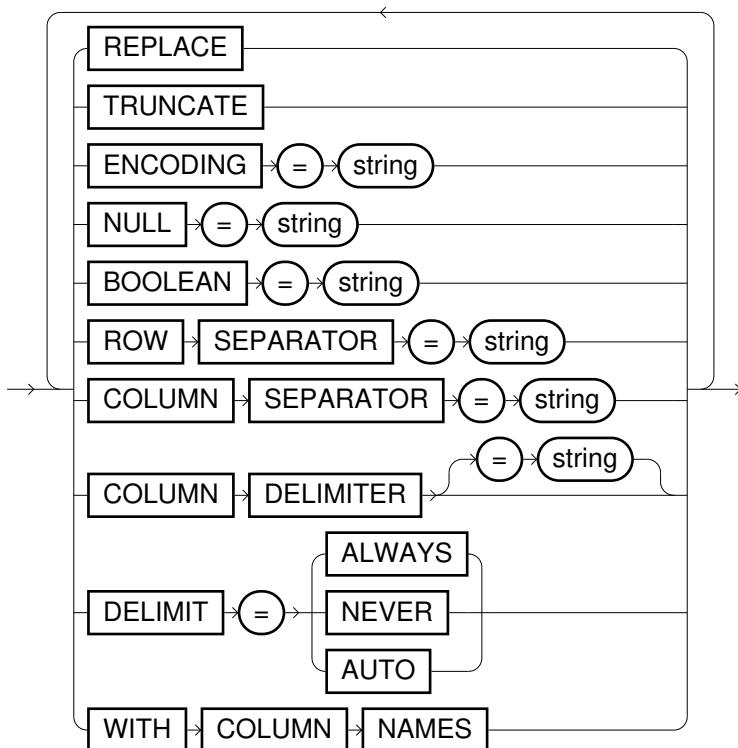
`csv_cols`:



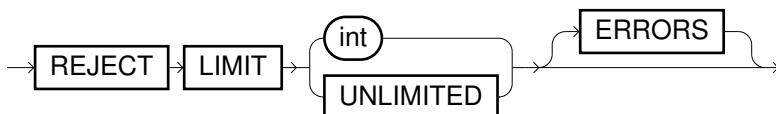
`fbv_cols`:



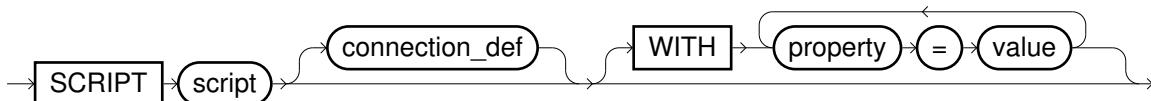
`file_opts`:



error_clause:=



script_dst:=



Anmerkung(en)

- Weitere Information zum Thema ETL finden Sie unter [Abschnitt 3.4, ETL-Prozesse](#).
- Ohne Angabe einer anderen Option (siehe unten) werden die Daten an das Ziel angehängt (Append Modus).
- Der aktuelle Status des Ladevorgangs kann in einer zweiten Verbindung zur Datenbank über die Tabelle [EXA_USER_SESSIONS](#) ausgelesen werden (Spalte ACTIVITY).
- Enthält ein Statement oder eine View eine ORDER BY Klausel auf äußerster Ebene, so werden die entsprechenden Daten in sortierter Reihenfolge exportiert (gilt nur für Dateien).
- Übersicht über die verschiedenen Elemente und deren Bedeutung:

Element	Beschreibung
data_src	Für die Quelldaten kann entweder eine Tabelle (als Identifier, z.B. MY_SCHEMA.MY_TABLE) oder eine Query (als String, z.B. 'SELECT "TEST" FROM DUAL') angegeben werden. Bei einer Tabelle kann man zusätzlich die zu exportierenden Spalten eingrenzen.
dbms_dst	Definiert das Datenbankziel, in das exportiert werden soll. Die Verbindungsdaten werden über die connection_def spezifiziert (siehe unten). Zur Verfügung stehen

Element	Beschreibung
	<p>entweder eine Exasol-Instanz (EXA), eine native Anbindung zu einer Oracle-Datenbank (ORA) oder eine JDBC-Verbindung zu beliebigen Datenbanken (JDBC).</p> <p>Defaultmäßig stehen bereits einige JDBC-Treiber zur Verfügung, die Sie unter EXAoperation einsehen und im Verbindungs-String adressieren können (z.B. jdbc:mysql, jdbc:postgres). Sie können aber über EXAoperation auch eigene Treiber konfigurieren und mittels der DRIVER-Option auswählen, falls dessen Präfix nicht schon eindeutig sein sollte.</p> <p> Nur die bereits vorinstallierten, in EXAoperation grau markierten JDBC-Treiber werden von uns getestet und offiziell unterstützt. Bei Problemen mit anderen Treibern wird unser Support versuchen, Ihnen entsprechend zu helfen.</p> <p>Für das Ziel kann entweder eine Tabelle angegeben werden oder ein Prepared Statement (z.B. ein INSERT-Statement oder ein Prozeduraufgruf). Im letzteren Fall werden die zu exportierenden Daten als Eingabedaten des Prepared Statements übergeben. Bitte beachten Sie, dass bei einer Tabelle der Name schemaqualifiziert angegeben werden muss.</p> <p> Bitte beachten Sie, dass im Fall einer Tabelle der Identifier genauso wie Exasol-Identifier behandelt werden. Daher müssen sie bei case-sensitiven Tabellen den Namen in Anführungszeichen setzen.</p> <p>Weitere Optionen für das Ziel:</p> <ul style="list-style-type: none"> REPLACE Löscht die Zieltabelle vor dem Export. TRUNCATE Löscht alle Zeilen aus der Zieltabelle vor dem Export. TRUNCATE kann nicht mit REPLACE oder CREATED BY kombiniert werden! CREATED BY Definiert einen Erzeugungsstring, mit dem die Tabelle vor dem Export angelegt wird.
file_dst	<p>Spezifiziert den Zielort für die Datei.</p> <ol style="list-style-type: none"> 1. Remote Dateien <p>Es werden FTP-, FTPS-, SFTP, HTTP- und HTTPS-Server unterstützt, deren Verbindungsdaten über die connection_def angegeben werden (siehe unten).</p> <p>Anmerkungen:</p> <ul style="list-style-type: none"> • Bei verschlüsselten Verbindungen findet keine Überprüfung der Zertifikate statt. • Bei URLs beginnend mit '<code>ftps://</code>' wird von der Verwendung der impliziten Verschlüsselung ausgegangen. • Bei URLs beginnend mit '<code>ftp://</code>' verschlüsselt Exasol Nutzernamen und Passwort (explizite Verschlüsselung), sofern dies vom Server unterstützt wird. Im Fall, dass der Server zusätzlich eine verschlüsselte Datenübertragung erfordert, werden sämtliche Daten verschlüsselt übertragen. • Für HTTP- bzw. HTTPS-Server unterstützt Exasol nur Basis-Authentifizierung. • Für HTTP- bzw. HTTPS-Server können HTTP Parameter angegeben werden, indem sie dem Dateinamen angefügt werden (z.B. FILE '<code>file.csv?op=CREATE&user.name=user</code>'). <ol style="list-style-type: none"> 2. Lokale Dateien

Element	Beschreibung
	<p>Mit dieser Option können Daten auch in lokale Dateien auf einem Client-System exportiert werden. Bei Angabe der SECURE-Option werden die Daten verschlüsselt, wodurch sich allerdings auch die Ladegeschwindigkeit verringert.</p> <p> Diese Funktionalität steht nur für EXAplus und den JDBC Treiber zur Verfügung und kann nicht als Prepared Statement oder aus Datenbank-Skripten heraus aufgerufen werden. Falls Sie eine lokale Datei über ein explizites Programm verarbeiten wollen, so können Sie das Tool EXAjload benutzen, welches im JDBC-Treiber-Paket mit ausgeliefert wird. Rufen Sie dieses Programm einfach ohne Parameter auf, um Informationen über die Verwendung zu erhalten.</p> <p> Zum Exportieren von lokalen Dateien öffnet der JDBC Treiber eine interne Verbindung zum Cluster und emuliert einen HTTP- bzw. HTTPS- (SECURE-Option) Server, der genau diese Datei bereitstellt. Für den Nutzer ist dies allerdings völlig transparent.</p>
	<p>Die Ziel-Datei kann eine CSV- bzw. FBV-Datei, deren Format-Spezifikationen in Das CSV Datenformat und Das Fixblock Datenformat (FBV) zu finden sind. In Dateinamen sind nur ASCII-Zeichen erlaubt. BOM wird nicht unterstützt.</p> <p>Komprimierte Daten werden anhand der Dateiendung erkannt, unterstützt werden dabei die Endungen <code>.zip</code>, <code>.gz</code> (gzip) und <code>.bz2</code> (bzip2).</p> <p>Werden mehrere Dateien angegeben, so hängt die Datenverteilung von mehreren Faktoren ab. Es kann auch durchaus passieren, dass einzelne Dateien leer bleiben.</p>
script_dst	<p>Spezifiziert das UDF Skript, das für den benutzerdefinierten Export verwendet werden soll. Sie können optional Verbindungsinformationen oder eine Liste von Parametern angeben, die an das Skript weitergegeben werden. Das angegebene Skript muss eine spezielle Funktion implementieren die von der Datenbank aufgerufen wird und die Spezifikation des Exports erhält (z.B. Informationen zu Verbindung und Parametern). Das Skript generiert dann einen SELECT Befehl, der aufgerufen wird, um den eigentlichen Export umzusetzen. Eine ausführliche Erklärung von benutzerdefinierten Exports mit Beispielen finden Sie in Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs.</p> <p>connection_def</p> <p>Optionale Angabe einer Verbindung zur Möglichkeit, die Verbindungsdaten wie das Passwort zu kapseln. Bitte lesen Sie den separaten Eintrag in dieser Tabelle über die genaue Syntax.</p> <p>WITH parameter=value ...</p> <p>Optionale Angabe von Parametern, die an das Skript weitergegeben werden. Jedes Skript kann selbst festlegen, welche Parameter es erwartet. Parameter sind Schlüssel-Wert Paare, wobei der Wert ein String ist, z.B.:</p> <pre data-bbox="552 1848 1203 1877">... WITH PARAM_1='val1' PARAM_2 = 'val2';</pre>
connection_def	<p>Definiert eine Verbindung zu einer Datenbank- oder einem Datei-Server. Dieser kann über einen Connection-String definiert werden, in der die Verbindungsdaten spezifiziert sind (z.B. <code>'ftp://192.168.1.1/'</code>).</p>

Element	Beschreibung
	<p>Zur regelmäßigen Benutzung einer externen Verbindung bietet sich allerdings die Verwendung von Connections an, in denen die Verbindungsdaten sowie Benutzer und evtl. das Passwort bequem gekapselt werden können. Für nähere Informationen und Beispielen zu Connection-Objekten siehe auch CREATE CONNECTION in Abschnitt 2.2.3, Zugriffssteuerung mittels SQL (DCL).</p> <p>Die Angabe von Nutzer und Passwort innerhalb des EXPORT-Befehls sind optional. Falls diese nicht angegeben sind, so werden die Daten des verwendeten Connection-Strings bzw. des Connection-Objekts benutzt.</p> <p>Für JDBC Verbindungen ist eine Kerberos Authentifizierung möglich, indem bestimmte Daten über das IDENTIFIED BY Feld definiert werden. Diese bestehen aus einem Indikator, dass Kerberos benutzt werden soll (ExaAuthType=Kerberos), einer base64 kodierten Konfigurations-Datei und einer base64 kodierten keytab Datei mit den Credentials des Principal. Die Syntax sieht wie folgt aus: IMPORT INTO table1 FROM JDBC AT '<JDBC_URL>' USER '<kerberos_principal>' IDENTIFIED BY 'ExaAuthType=Kerberos;<base64_krb_conf>;<base64_keytab>' TABLE table2; Weitere Details und Beispiele finden Sie in unserem Solution Center: https://wwwexasolcom/portal/display/SOL-512</p>
csv_cols	<p>Definiert, welche Spalten in der CSV-Datei wie geschrieben werden sollen. Bitte beachten Sie auch Das CSV Datenformat.</p> <p>col_nr Definiert die Spaltennummer (beginnend bei 1). Alternativ können sie mittels . . . auch einen fortlaufenden Bereich von Spalten definieren (z.B. 5 . . 8 für die Spalten 5,6,7,8). Bitte beachten Sie, dass die Spaltennummern stets aufsteigend sein müssen!</p> <p>FORMAT Optionale Formatangabe für Zahlen oder Zeitstempel-/Datums-Werte (Default: Session-Format). Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle.</p> <p>DELIMIT Im Default-Fall (AUTO) werden Feldbegrenzungszeichen immer dann geschrieben, wenn in den Daten der COLUMN SEPARATOR, der COLUMN DELIMITER oder ein Whitespace-Zeichen vorkommt. Durch Angabe von ALWAYS oder NEVER kann definiert werden, dass für diese Spalte das Feldbegrenzungszeichen immer bzw. niemals geschrieben wird. Diese Einstellung auf Spaltenebene überschreibt die globale Einstellung (siehe <code>file_opts</code>).</p> <p>Bitte beachten Sie, dass bei der Benutzung der Option NEVER nicht gewährleistet werden kann, dass die exportierten Daten auch wieder in Exasol importiert werden können.</p> <p>Im folgenden Beispiel werden 4 Spalten in eine CSV-Datei geladen, wobei für die letzte ein explizites Datumsformat definiert ist:</p> <pre>(1 . . 3 , 4 FORMAT= ' DD-MM-YYYY')</pre>
fbv_cols	<p>Definiert, wie die Spalten in die FBV-Datei geschrieben werden sollen. Bitte beachten Sie auch Das Fixblock Datenformat (FBV).</p> <p>Folgende Elemente können hierbei angegeben werden:</p> <p>SIZE Anzahl an Bytes der Spalte, der Default berechnet sich aus dem Quell-Datentyp:</p>

Element	Beschreibung
	<p>'1/0', 'TRUE/FALSE', 'true/false', 'True/False', 'T/F', 't/f', 'y/n', 'Y/N', 'yes/no', 'Yes/No', 'YES/NO'</p> <p> Beim Import werden all diese Paare automatisch erkannt, falls Sie Strings in eine Boolean-Spalte einfügen.</p>
ROW SEPARATOR	<p>Zeilenumbruchzeichen:</p> <ul style="list-style-type: none"> • 'LF' (Default): entspricht dem ASCII-Zeichen 0x0a • 'CR': entspricht dem ASCII-Zeichen 0x0d • 'CRLF': entspricht den ASCII-Zeichen 0x0d und 0x0a • 'NONE': kein Zeilenumbruchzeichen (nur bei FBV erlaubt)
COLUMN SEPARATOR	Feldtrennzeichen für CSV-Dateien. Standardmäßig wird das Komma (,) verwendet. Angeben kann man einen String, entweder im Klartext (Bsp: ','), als Hexadezimalwert (Bsp: '0x09') oder als Abkürzung ('NUL', 'TAB', 'LF', 'CR' oder 'ESC'). Ein Klartext-String ist auf 10 Zeichen begrenzt, die zum ENCODING (siehe oben) der Datei automatisch konvertiert werden. Ein Hexadezimalwert ist auf 10 Bytes (nicht Zeichen) begrenzt, die nicht konvertiert werden.
COLUMN DELIMITER	Feldbegrenzungszeichen für CSV-Dateien. Standardmäßig wird das Anführungszeichen ("") verwendet. Angeben kann man einen String, entweder im Klartext (Bsp: ""), als Hexadezimalwert (Bsp: '0x09') oder als Abkürzung ('NUL', 'TAB', 'LF', 'CR', 'ESC'). Ein Klartext-String ist auf 10 Zeichen begrenzt, die zum ENCODING (siehe oben) der Datei automatisch konvertiert werden. Ein Hexadezimalwert ist auf 10 Bytes (nicht Zeichen) begrenzt, die nicht konvertiert werden. Soll kein Feldbegrenzungszeichen verwendet werden, so kann ein leerer String angegeben werden ("") oder die Option DELIMIT NEVER definiert werden (siehe unten).
DELIMIT	<p>Im Default-Fall (AUTO) wird ein Feldbegrenzungszeichen nur dann geschrieben, wenn in den Daten der COLUMN SEPARATOR, der ROW SEPARATOR, der COLUMN DELIMITER oder ein Whitespace-Zeichen vorkommt. Durch die Verwendung der Optionen ALWAYS bzw. NEVER können Sie definieren, dass dieser immer oder niemals geschrieben wird. Diese globale Option kann innerhalb der einzelnen Spaltendefinitionen überschrieben werden (siehe csv_cols).</p> <p>Bitte beachten Sie, dass bei der Benutzung der Option NEVER nicht gewährleistet werden kann, dass die exportierten Daten auch wieder in Exasol importiert werden können!</p>
WITH COLUMN NAMES	Bei Angabe dieser Option (nur für CSV-Dateien möglich) wird eine zusätzliche Zeile am Anfang der Datei

Element	Beschreibung
	geschrieben, welche die Spaltennamen der zu exportierenden Daten enthält. Bei einem Subselect können dies auch zusammengesetzte Ausdrücke sein. Die restlichen Einstellungen wie z.B. Feldtrennzeichen werden hier ebenfalls angewendet. Wollen Sie die gleiche Datei mittels IMPORT wieder importieren, so empfiehlt sich die Option SKIP 1 .
error_clause	Bei Angabe dieser Klausel kann die Anzahl an erlaubten fehlerhaften Zeilen in der Quelle definiert werden. Z.B. würde das Statement bei REJECT LIMIT 5 bei bis zu fünf fehlerhaften Zeilen erfolgreich durchlaufen, ab sechs Fehlern allerdings eine Exception werfen. REJECT LIMIT 0 entspricht dem Verhalten, wenn diese Klausel gar nicht angegeben wird.

Beispiel(e)

```

EXPORT tab1 INTO CSV
    AT 'ftp://192.168.1.1/' USER 'agent_007' IDENTIFIED BY 'secret'
    FILE 'tab1.csv'
    COLUMN_SEPARATOR = ';'
    ENCODING = 'Latin1'
    WITH COLUMN NAMES;

CREATE CONNECTION my_connection
    TO 'ftp://192.168.1.1/' USER 'agent_007' IDENTIFIED BY 'secret';

EXPORT (SELECT * FROM T WHERE id=3295) INTO FBV
    AT my_connection
    FILE 't1.fbv' FILE 't2.fbv'
    REPLACE;

EXPORT (SELECT * FROM my_view) INTO EXA
    AT '192.168.6.11..14:8563'
    USER 'my_user' IDENTIFIED BY 'my_secret'
    TABLE my_schema.my_table
    CREATED BY 'CREATE TABLE my_table(order_id INT, price DEC(18,2))';

EXPORT tab1 INTO JDBC
    AT 'jdbc:exa:192.168.6.11..14:8563'
    USER 'agent_007' IDENTIFIED BY 'secret'
    TABLE my_schema.tab1;

EXPORT tab1 INTO CSV
    AT 'http://HadoopNode:50070/webhdfs/v1/tmp'
    FILE 'file.csv?op=CREATE&user.name=user';

EXPORT tab1 INTO CSV
    AT 'https://testbucket.s3.amazonaws.com'
    USER '<AccessKeyId>' IDENTIFIED BY '<SecretAccessKey>'
    FILE 'file.csv';

EXPORT tab1 INTO SCRIPT etl.export_hcat_table
WITH HCAT_DB      = 'default'
      HCAT_TABLE   = 'my_hcat_table'
      HCAT_ADDRESS = 'hcatalog-server:50111'

```

```
HDFS_USER      = 'hdfs';

EXPORT tab1 INTO LOCAL CSV FILE '~/.my_table.csv'
COLUMN SEPARATOR = ';' SKIP = 5;
```

2.2.3. Zugriffssteuerung mittels SQL (DCL)

Die SQL-Befehle der Data Control Language (DCL) dienen der Steuerung der Zugriffsrechte in der Datenbank. Durch die Verwaltung von Benutzern und Rollen sowie die Gewährung von Privilegien kann unterschieden werden, wer welche Aktionen in der Datenbank ausführen darf.

Einen Einstieg in die grundlegenden Konzepte zum Thema Rechteverwaltung finden Sie in [Abschnitt 3.2, Rechteverwaltung](#). Weitere Details wie z.B. die Liste aller Privilegien, eine Zusammenfassung der Zugriffsrechte für SQL-Befehle sowie die für die Rechteverwaltung relevanten Systemtabellen befinden sich in [Anhang B, Details zur Rechteverwaltung](#).

Zudem wird innerhalb der SQL-Referenz für jeden SQL-Befehl in den Vorbedingungen angegeben, welche Privilegien für den Befehl nötig sind.

CREATE USER

Zweck

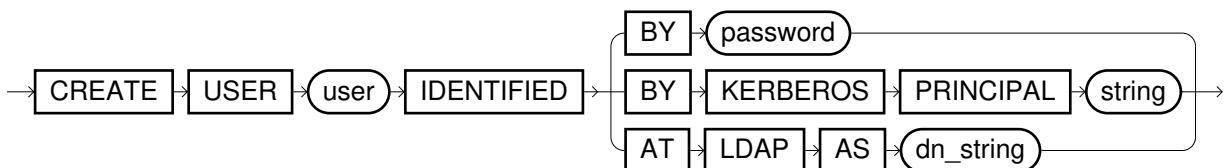
Legt einen Benutzer in der Datenbank an. Das angegebene Passwort wird zur Authentifizierung in Exasol benutzt.

Vorbedingung(en)

- Das Systemprivileg CREATE USER

Syntax

create_user ::=



Anmerkung(en)

- Damit der Benutzer sich anschließend auch einloggen kann, muss ihm noch das Systemprivileg CREATE SESSION gewährt werden.
- Für den Benutzernamen gelten die gleichen Regeln wie für SQL-Bezeichner (siehe [Abschnitt 2.1.2, SQL-Bezeichner](#)). Es wird allerdings selbst bei Bezeichnern in Anführungszeichen nicht auf Groß-/Kleinschreibung geachtet. Das bedeutet, dass die Benutzernamen "Test", "TEST" und test Synonyme sind.
- Es existieren verschiedene Möglichkeiten zur Authentifizierung:

Via Passwort Die Datenbank prüft beim Anmelden direkt das eingegebene Passwort. Bitte beachten Sie, dass Passwörter als Bezeichner eingegeben werden müssen (siehe [Abschnitt 2.1.2, SQL-Bezeichner](#)). Sofern Sie einen begrenzten Bezeichner (in doppelten Anführungszeichen), so wird die Groß-/Kleinschreibung berücksichtigt.



Im Falle eines regulären Bezeichners (ohne doppelte Anführungszeichen) wird das Passwort zu Großbuchstaben konvertiert und muss dann auch so bei der Anmeldung angegeben werden.

Via Kerberos

Die Treiber authentifizieren sich über einen Kerberos-Dienst (Single Sign-On). Typischerweise besteht das angegebene Principal aus der Form <user>@<realm>. Wei-

tere Informationen zur generellen Kerberos-Konfiguration finden Sie in den entsprechenden Kapiteln zu den Treibern ([Kapitel 4, Clients und Schnittstellen](#)) sowie unserem Operational Manual: <https://wwwexasol.com/portal/display/DOC/Operational+Manual>

Via LDAP

Die Datenbank prüft das eingegebene Passwort gegen einen LDAP-Server, welcher in EXAoperation je Datenbank konfiguriert werden kann. Der Parameter *dn-string* (Zeichenkette in einfachen Anführungszeichen) definiert den so genannten *distinguished name*, welcher den Nutzernamen im LDAP-Server darstellt. Nicht unterstützt werden [SASL](#) sowie eine Zertifikatverwaltung.

- Nach der Erstellung des Nutzers existiert kein Schema für diesen Nutzer.
- Ein Nutzer kann durch den [RENAME](#) Befehl umbenannt werden.

Beispiel(e)

```
CREATE USER user_1 IDENTIFIED BY "h12_xhz";
CREATE USER user_2 IDENTIFIED AT LDAP
AS 'cn=user_2,dc=authorization,dc=exasol,dc=com';
```

ALTER USER

Zweck

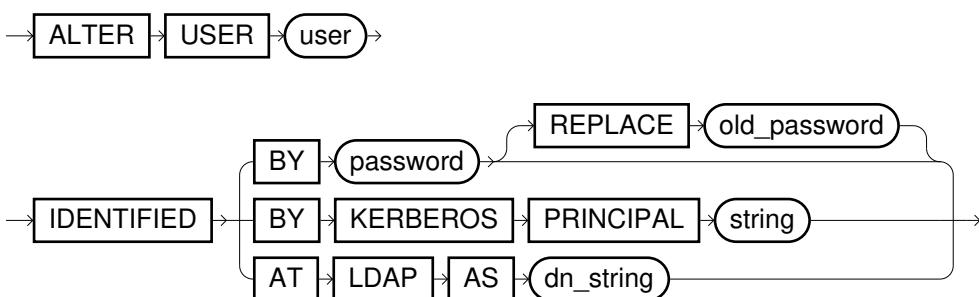
Ändert das Passwort eines Benutzers.

Vorbedingung(en)

- Falls die Authentifizierung über ein Passwort geschieht, so kann ein Nutzer dieses selbst immer ändern.
- Für das Setzen des Passworts anderer User sowie die Definition der Kerberos / LDAP Authentifizierung benötigt man das Systemprivileg ALTER USER.

Syntax

alter_user ::=



Anmerkung(en)

- Falls man das Systemprivileg ALTER USER besitzt, ist die REPLACE-Klausel optional bzw. wird das alte Passwort nicht überprüft.
- Aus Sicherheitsgründen muss auch dann das alte Passwort angegeben werden, wenn ein Benutzer sein eigenes Passwort ändern will (außer er besitzt das Systemprivileg ALTER USER).
- Details zur Kerberos / LDAP Authentifizierung und den Regeln zur Passwortvergabe siehe auch [CREATE USER](#).

Beispiel(e)

```
ALTER USER user_1 IDENTIFIED BY "h22_xhz" REPLACE "h12_xhz";
-- ALTER_USER Privileg nötig für nächste Befehle
ALTER USER user_1 IDENTIFIED BY "h12_xhz";

ALTER USER user_2 IDENTIFIED AT LDAP
AS 'cn=user_2,dc=authorization,dc=exasol,dc=com';
```

DROP USER

Zweck

Löscht einen Benutzer sowie dessen Schemas inklusive aller darin enthaltenen Schemaobjekte.

Vorbedingung(en)

- Das Systemprivileg DROP USER.

Syntax

drop_user ::=



Anmerkung(en)

- Falls CASCADE angegeben ist, so werden alle Schemas des Nutzers sowie deren Inhalt gelöscht! Außerdem werden datenbankweit alle Foreign Keys gelöscht, die Tabellen des Nutzers referenzieren.
- Falls es noch Schemas gibt, die dem Benutzer gehören, so muss CASCADE angegeben werden oder aber diese explizit vorher gelöscht werden (mittels DROP SCHEMA).
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls der Nutzer nicht existiert.
- Falls der zu löschen Benutzer gleichzeitig eingeloggt ist, wird eine Fehlermeldung ausgegeben und der Benutzer nicht gedroppt. Es empfiehlt sich dann, dem Benutzer das Privileg CREATE SESSION zu entziehen und seine Session mittels KILL zu beenden.

Beispiel(e)

```
DROP USER test_user1;
DROP USER test_user2 CASCADE;
```

CREATE ROLE

Zweck

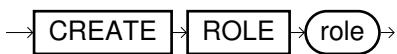
Erzeugt eine Rolle.

Vorbedingung(en)

- Das Systemprivileg CREATE ROLE.

Syntax

create_role ::=



Anmerkung(en)

- Eine Rolle verfügt nach der Erzeugung über keinerlei Privilegien. Diese werden erst durch den GRANT-Befehl zugewiesen. Entweder werden einer Rolle Privilegien direkt erteilt oder ihr werden andere Rollen zugewiesen.
- Für Rollennamen gelten die gleichen Regeln wie für Benutzernamen (siehe [CREATE USER](#)).
- Eine Rolle kann durch den [RENAME](#) Befehl umbenannt werden.

Beispiel(e)

```
CREATE ROLE test_role;
```

DROP ROLE

Zweck

Löscht eine Rolle.

Vorbedingung(en)

- Entweder das Systemprivileg DROP ANY ROLE oder man muss diese Rolle mit der WITH ADMIN OPTION zugewiesen bekommen haben.

Syntax

drop_role ::=



Anmerkung(en)

- Falls CASCADE angegeben ist, so werden alle Schemas der Rolle sowie deren Inhalt gelöscht!
- Falls es noch Schemas gibt, die der Rolle gehören, so muss CASCADE angegeben werden oder aber diese explizit vorher gelöscht werden (mittels [DROP SCHEMA](#)).
- Die Rolle wird durch diesen Befehl auch von allen Benutzern entfernt, die diese Rolle inne hatten. Allerdings sind davon offene Transaktionen von solchen Benutzern nicht betroffen.
- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die Rolle nicht existiert.
- Es reicht nicht aus, die Rolle erzeugt zu haben, um sie löschen zu können.

Beispiel(e)

```
DROP ROLE test_role;
```

CREATE CONNECTION**Zweck**

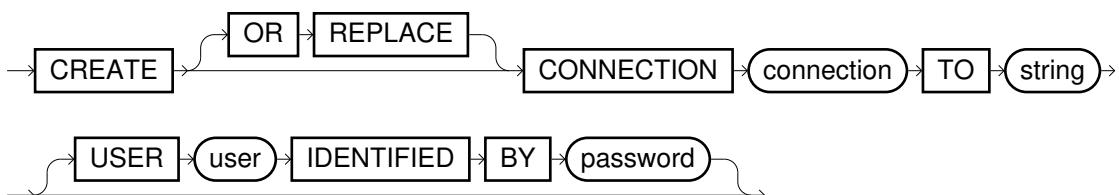
Mit diesem Kommando kann eine externe Verbindung erzeugt werden.

Vorbedingung(en)

- Das Systemprivileg **CREATE CONNECTION**
- Falls die OR REPLACE-Option angegeben wurde und die Connection bereits existiert, so sind zudem die für **DROP CONNECTION** benötigten Rechte erforderlich.

Syntax

create_connection ::=

**Anmerkung(en)**

- Externe Verbindungen können in den Befehlen **IMPORT** und **EXPORT** benutzt werden. Benutzer müssen hierfür entsprechende Zugriffsrechte auf die Verbindung erhalten haben (mittels **GRANT**). Der Erzeuger der Verbindung bekommt diese automatisch mit ADMIN OPTION zugewiesen.
- Zudem können Verbindungen verwendet werden, um Nutzern Lese-Zugriff aus Skripten heraus auf das lokale Buckets im BucketFS zu geben. Details hierzu finden Sie unter [Abschnitt 3.6.4, Das synchrone Cluster-Dateisystem BucketFS](#).
- Definiert werden kann entweder eine Exasol-Instanz, eine native Anbindung zu einer Oracle-Datenbank oder eine JDBC-Verbindung zu beliebigen Datenbanken. Defaultmäßig stehen bereits einige JDBC-Treiber zur Verfügung, die Sie unter EXAoperation einsehen und im Verbindungs-String adressieren können (z.B. **jdbc:mysql**, **jdbc:postgres**). Sie können aber über EXAoperation auch eigene Treiber konfigurieren und mittels der DRIVER-Option auswählen, falls dessen Präfix nicht schon eindeutig sein sollte.

Nur die bereits vorinstallierten, in EXAoperation grau markierten JDBC-Treiber werden von uns getestet und offiziell unterstützt. Bei Problemen mit anderen Treibern wird unser Support versuchen, Ihnen entsprechend zu helfen.f

- Die Angabe von Nutzer und Passwort sind optional und können auch erst in den **IMPORT** und **EXPORT** Befehlen definiert werden.
- Fehlerhaften Verbindungsdaten fallen erst bei der Benutzung in den **IMPORT** und **EXPORT** Befehlen auf.
- Der Liste aller Verbindungen finden Sie in der Systemtabelle **EXA_DBA_CONNECTIONS** (siehe [Anhang A, Systemtabellen](#)).
- Eine Verbindung kann durch den **RENAME** Befehl umbenannt werden.

Beispiel(e)

```

CREATE CONNECTION ftp_connection
  TO 'ftp://192.168.1.1/'
  USER 'agent_007'
  IDENTIFIED BY 'secret';

CREATE CONNECTION exa_connection TO '192.168.6.11..14:8563';

CREATE CONNECTION ora_connection TO '(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.6.54)(PORT = 1521))
  (CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = orcl)))';

CREATE CONNECTION jdbc_connection_1
  TO 'jdbc:mysql://192.168.6.1/my_user';
CREATE CONNECTION jdbc_connection_2
  TO 'jdbc:postgresql://192.168.6.2:5432/my_db?stringtype=unspecified';

```

ALTER CONNECTION**Zweck**

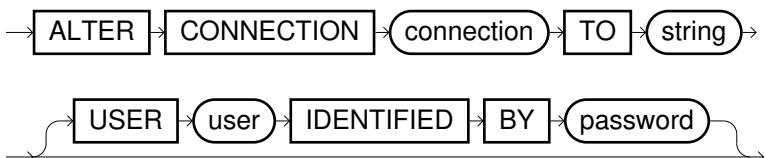
Ändert die Verbindungsdaten einer externen Verbindung.

Vorbedingung(en)

- Systemprivileg ALTER ANY CONNECTION oder man muss die Verbindung mit der WITH ADMIN OPTION zugewiesen bekommen haben.

Syntax

alter_connection::=

**Beispiel(e)**

```

ALTER CONNECTION ftp_connection
  TO 'ftp://192.168.1.1/'
  USER 'agent_008'
  IDENTIFIED BY 'secret';

ALTER CONNECTION exa_connection TO '192.168.6.11..14:8564';

ALTER CONNECTION ora_connection TO '(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.6.54)(PORT = 1522))
  (CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = orcl)))';

```

DROP CONNECTION

Zweck

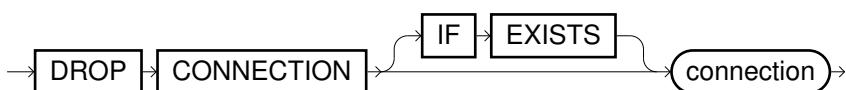
Mit diesem Kommando kann eine externe Verbindung gelöscht werden.

Vorbedingung(en)

- System-Privileg DROP ANY CONNECTION oder man muss die Verbindung mit der WITH ADMIN OPTION zugewiesen bekommen haben.

Syntax

drop_connection::=



Anmerkung(en)

- Wird die optionale IF EXISTS Klausel angegeben, so wirft das Statement keinen Fehler, falls die Verbindung nicht existiert.

Beispiel(e)

```
DROP CONNECTION my_connection;
```

GRANT

Zweck

Mit dem GRANT-Befehl kann man Systemprivilegien, Objektprivilegien, Rollen oder der Zugriff auf Verbindungen an Benutzer oder Rollen vergeben.

Vorbedingung(en)

- Für Systemprivilegien benötigt der Zuweisende das Systemprivileg GRANT ANY PRIVILEGE oder er muss dieses Systemprivileg mit der WITH ADMIN OPTION erhalten haben.
- Bei Objektrechten muss der Zuweisende entweder Besitzer des Objekts sein oder über das Systemprivileg GRANT ANY OBJECT PRIVILEGE verfügen.

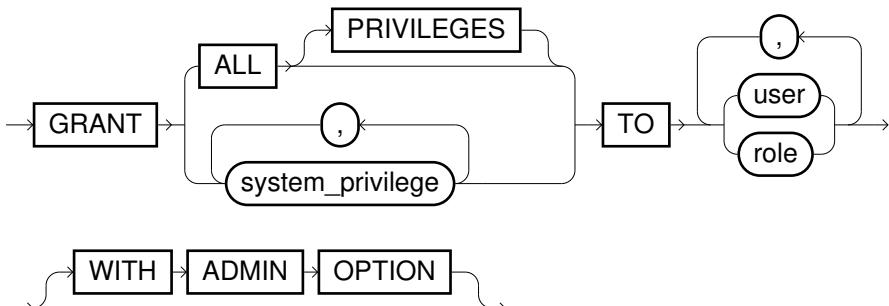
Bei GRANT SELECT auf Views muss beachtet werden, dass der Zuweisende das SELECT auf die View gewähren darf und zusätzlich der Besitzer der View entsprechende SELECT-Privilegien auf die Basistabellen besitzt, die er anderen Benutzern auch weitergeben darf (entweder er ist Besitzer oder hat das Privileg GRANT ANY OBJECT PRIVILEGE). Andernfalls wäre es möglich, den Zugriff auf eine fremde Tabelle über die Erstellung einer View jedem beliebigen Nutzer zu erlauben.

- Für Rollen benötigt der Zuweisende entweder das Systemprivileg GRANT ANY ROLE oder er muss die Rolle mit der WITH ADMIN OPTION erhalten haben.
- Für Prioritäten benötigt der Zuweisende das Systemprivileg GRANT ANY PRIORITY.

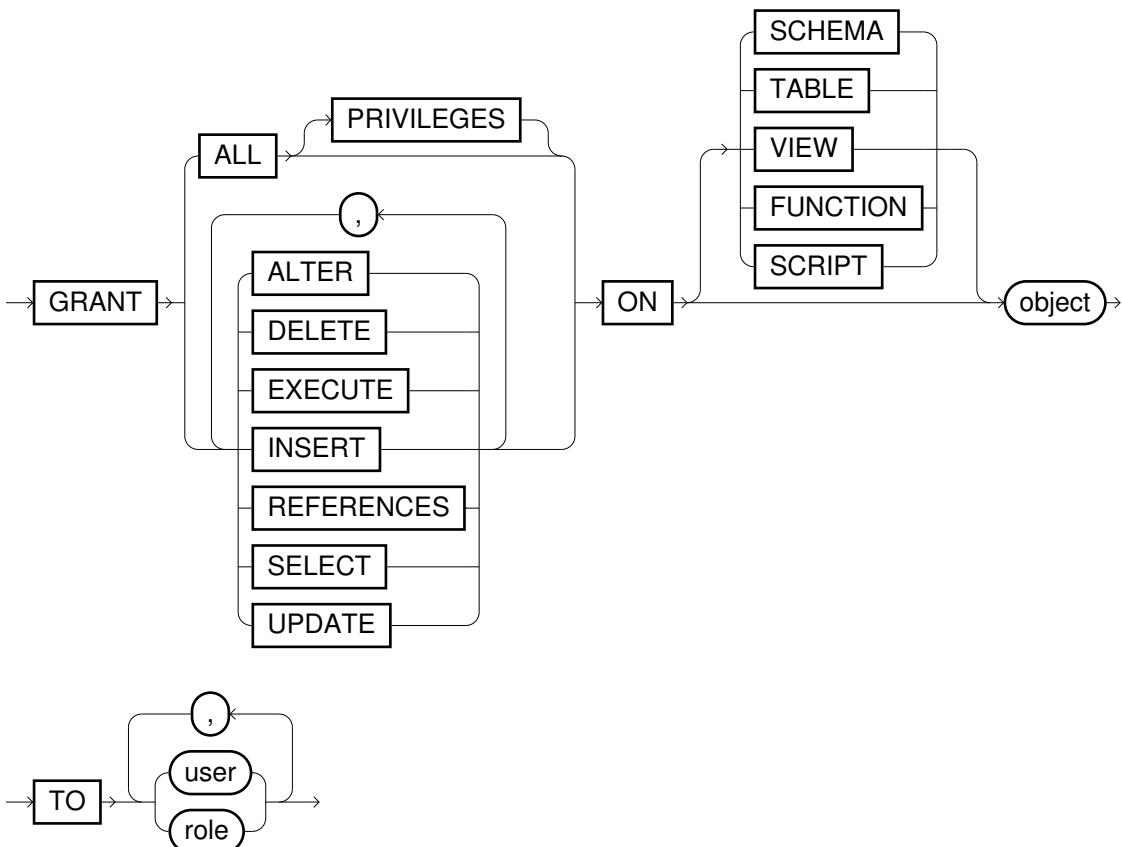
- Für Verbindungen benötigt der Zuweisende entweder das Systemprivileg GRANT ANY CONNECTION oder er muss die Verbindung mit der WITH ADMIN OPTION erhalten haben.

Syntax

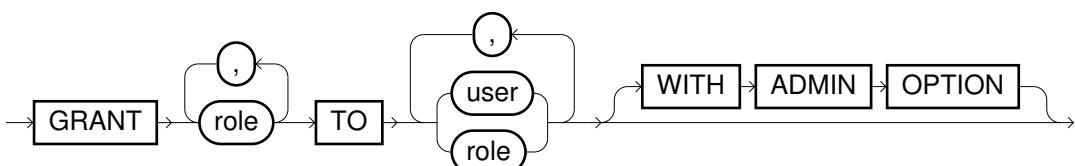
grant_system_privileges ::=



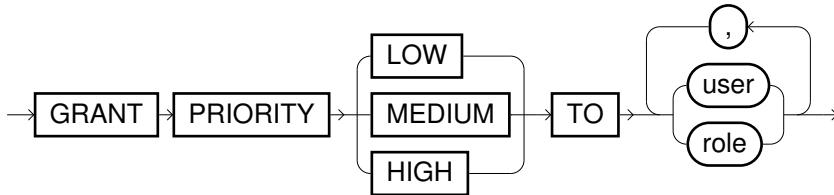
grant_object_privileges ::=



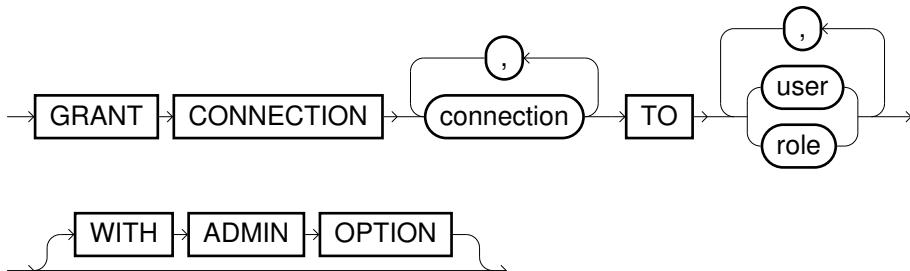
grant_roles ::=



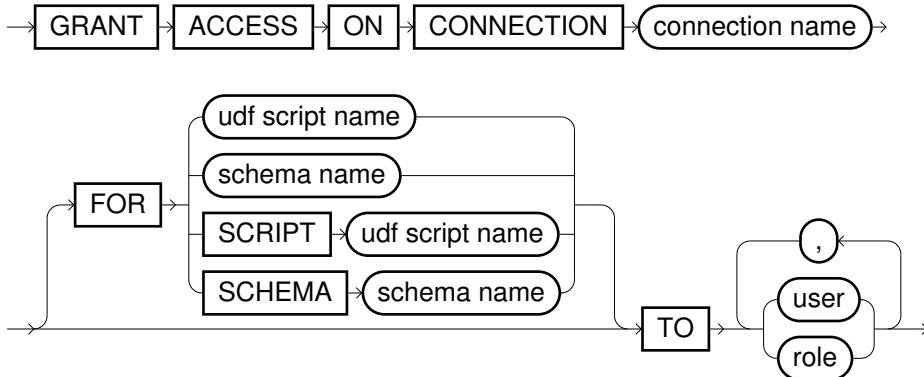
grant_priority ::=



grant_connection ::=



grant_connection_restricted ::=



Anmerkung(en)

- Die Liste der von Exasol unterstützten Systemprivilegien ist in [Tabelle B.1, Systemprivilegien in Exasol](#) zu finden.
- Um die Sicherheit der Datenbank zu gewährleisten, sollte der GRANT-Befehl nur sehr gezielt eingesetzt werden. Einige der Privilegien bzw. Rollen führen zur vollständigen Kontrolle über die Datenbank. Die Rolle DBA besitzt alle möglichen Systemprivilegien mit der ADMIN-Option. Mit dem Privileg GRANT ANY PRIVILEGE kann man alle System-Privilegien vergeben. Mit dem Privileg ALTER USER kann man das Passwort von SYS ändern. Und mit dem Privileg GRANT ANY ROLE kann man alle Rollen gewähren (also auch z.B. die Rolle DBA).
- Bei GRANT ALL werden dem Benutzer alle System- oder Objekt-Privilegien gewährt.
- Bei Zuweisung eines Objektprivilegs an ein Schema gilt dieses Privileg für alle darin enthaltenen Schemaobjekte. Objektprivilegien können nicht auf virtuelle Schemas oder die darin enthaltenen Tabellen vergeben werden.
- Das Objektprivileg REFERENCES kann nicht an Rollen vergeben werden.
- Zugewiesene Rollen können vom Benutzer nicht aktiviert oder deaktiviert werden.
- Das ACCESS Privileg erlaubt den Zugriff auf die Verbindungs-Informationen (auch das Passwort!) aus (bestimmten) UDF-Skripten heraus. Dies wird für Adapter-Skripte von virtuellen Schemas benötigt, siehe auch [Abschnitt 3.7, Virtuelle Schemas](#).
- Details zu Prioritäten finden Sie in [Abschnitt 3.3, Prioritäten](#), zu Verbindungen weiter oben in den Erläuterungen zum Befehl [CREATE CONNECTION](#).

Beispiel(e)

```
-- Systemprivileg
GRANT CREATE SCHEMA TO role1;
GRANT SELECT ANY TABLE TO user1 WITH ADMIN OPTION;

-- Objektprivilegien
GRANT INSERT ON my_schema.my_table TO user1, role2;
GRANT SELECT ON VIEW my_schema.my_view TO user1;

-- Zugriff auf my_view für alle Benutzer
GRANT SELECT ON my_schema.my_view TO PUBLIC;

-- Rollen
GRANT role1 TO user1, user2 WITH ADMIN OPTION;
GRANT role2 TO role1;

-- Priorität
GRANT PRIORITY HIGH TO role1;

-- Verbindung
GRANT CONNECTION my_connection TO user1;

-- Zugriff auf Verbindungsdetails für ein Skript
GRANT ACCESS ON CONNECTION my_connection
FOR SCRIPT script1 TO user1;
```

REVOKE

Zweck

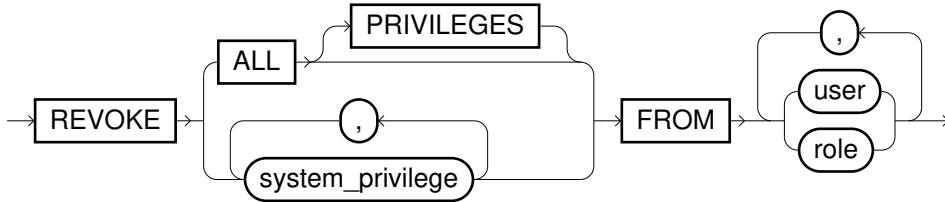
Mit dem REVOKE-Befehl kann man Systemprivilegien, Objektprivilegien, Rollen oder den Zugriff auf Verbindungen von Benutzern oder Rollen entziehen.

Vorbedingung(en)

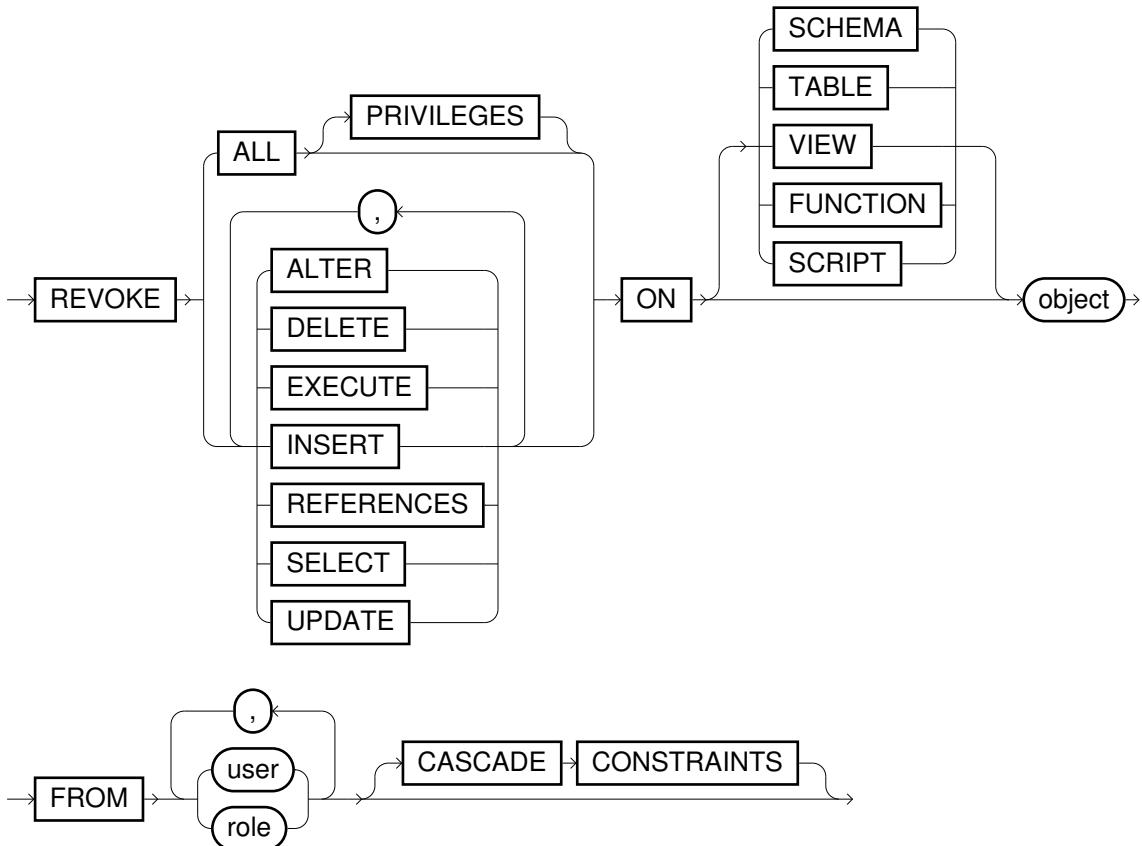
- Bei Systemprivilegien benötigt der Entziehende das Systemprivileg GRANT ANY PRIVILEGE oder er muss dieses Systemprivileg mit der WITH ADMIN OPTION erhalten haben.
- Bei Objektprivilegien benötigt der Entziehende das Systemprivileg GRANT ANY OBJECT PRIVILEGE oder er muss der Besitzer des Objektes sein.
- Bei Rollen benötigt der Entziehende das Systemprivileg GRANT ANY ROLE oder er muss die Rolle mit der WITH ADMIN OPTION erhalten haben.
- Für Prioritäten benötigt der Entziehende das Systemprivileg GRANT ANY PRIORITY.
- Bei Verbindungen benötigt der Entziehende das Systemprivileg GRANT ANY CONNECTION oder er muss die Verbindung mit der WITH ADMIN OPTION erhalten haben.

Syntax

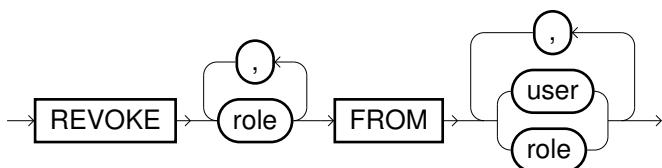
revoke_system_privileges::=



revoke_object_privileges ::=

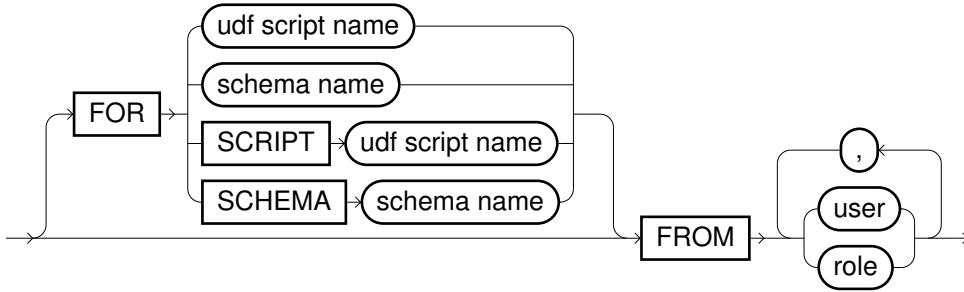


revoke_roles ::=

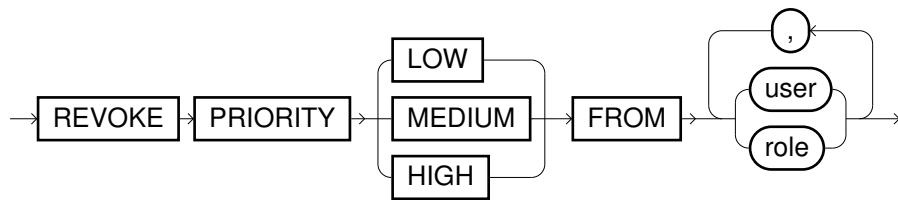


revoke_connection_restricted ::=

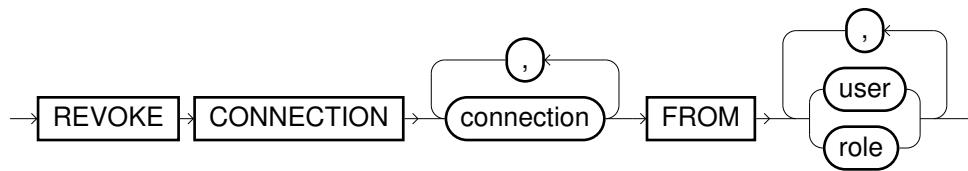




revoke_priority ::=



revoke_connections ::=



Anmerkung(en)

- Falls ein Benutzer das gleiche Privileg oder die gleiche Rolle von mehreren Benutzern erhalten hat, so werden diese bei einem entsprechenden REVOKE alle gelöscht.
- Wurde ein Objekt-Privileg für ein Schemaobjekt und zusätzlich für dessen Schema (also implizit für alle darin enthaltenen Objekte) vergeben und anschließend das Privileg für das Schema wieder entzogen, so gilt das Objekt-Privileg für das einzelne Schemaobjekt weiterhin.
- Das Objekt-Privileg REFERENCES kann einem Nutzer nur entzogen werden, wenn er keine Foreign Keys auf die entsprechenden Tabelle angelegt hat. Ist dies der Fall, können mit der Option CASCADE CONSTRAINTS diese Foreign Keys automatisch gelöscht werden.
- Bei REVOKE ALL [PRIVILEGES] werden im Gegensatz zu Oracle alle System- bzw. Objekt-Privilegien gelöscht, auch wenn der Benutzer vorher nicht mittels GRANT ALL alle Rechte gewährt bekommen hatte.

Beispiel(e)

```

-- Systemprivileg
REVOKE CREATE SCHEMA FROM role1;

-- Objektprivilegien
REVOKE SELECT, INSERT ON my_schema.my_table FROM user1, role2;
REVOKE ALL PRIVILEGES ON VIEW my_schema.my_view FROM PUBLIC;

-- Rolle
REVOKE role1 FROM user1, user2;

-- Priorität
REVOKE PRIORITY FROM role1;
  
```

```
-- Verbindung  
REVOKE CONNECTION my_connection FROM user1;
```

2.2.4. Anfragesprache (DQL)

Mit Hilfe der Data Query Language (DQL) können Inhalte aus der Datenbank abgefragt und analysiert werden.

SELECT

Zweck

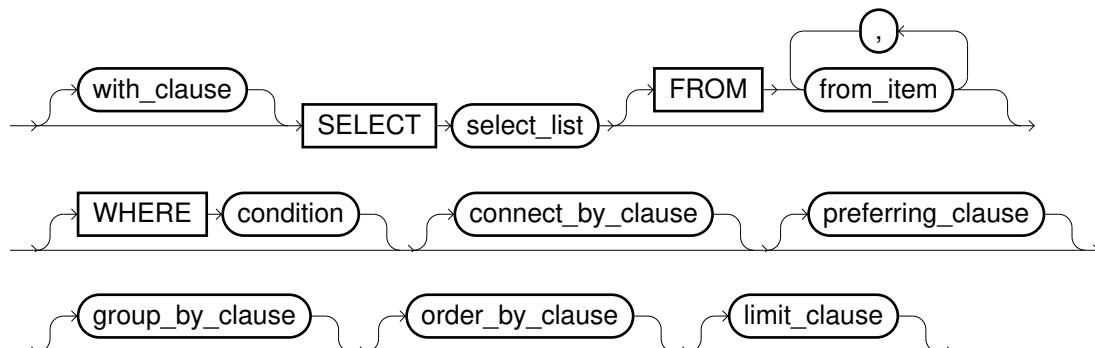
Mit Hilfe des SELECT-Statements können Daten aus Tabellen und Views selektiert werden.

Vorbedingung(en)

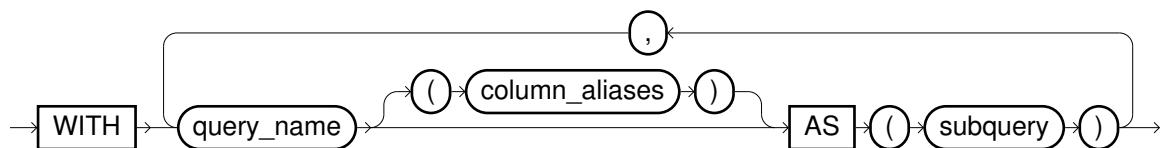
- System-Privileg `SELECT ANY TABLE` oder entsprechende `SELECT`-Rechte auf die in der `SELECT`-Liste referenzierten Tabellen bzw. Views. Entweder gehören diese dem aktuellen Benutzer oder einer seiner Rollen oder er besitzt das Objekt-Privileg `SELECT` auf die Tabelle/View bzw. deren Schema.
- Bei Views ist erforderlich, dass der Besitzer der View entsprechende `SELECT`-Privilegien auf die referenzierten Objekte besitzt.
- Bei Verwendung eines subimport benötigen Sie die entsprechenden Rechte analog zum `IMPORT` Befehl.

Syntax

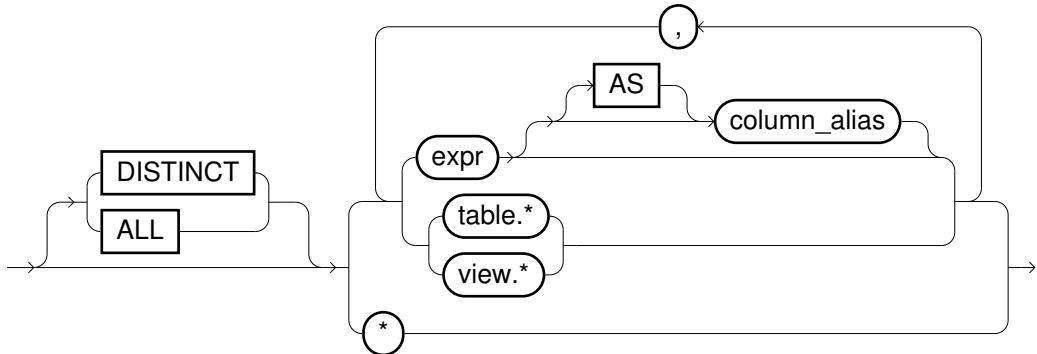
`subquery`::=



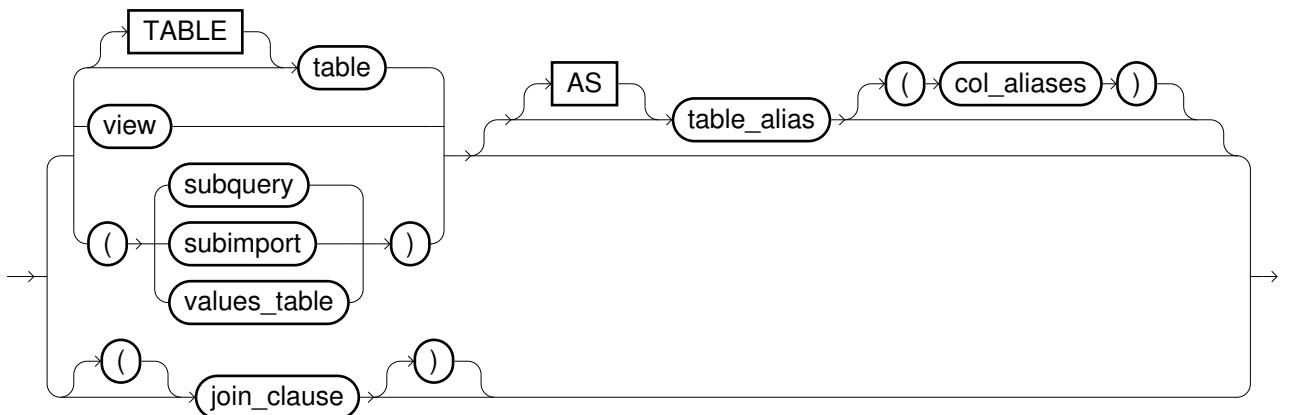
`with_clause`::=



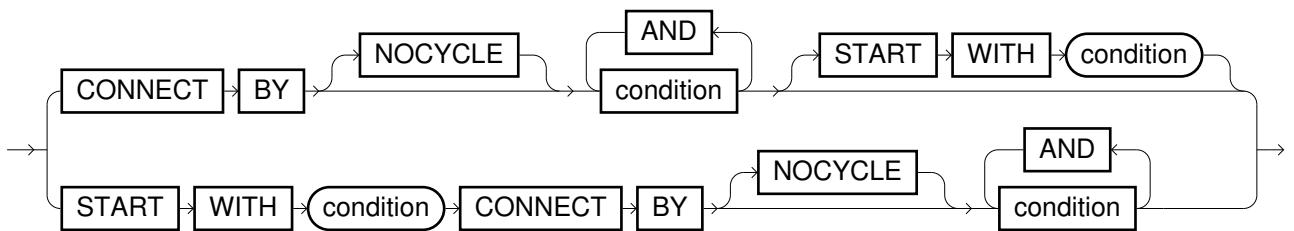
`select_list`::=



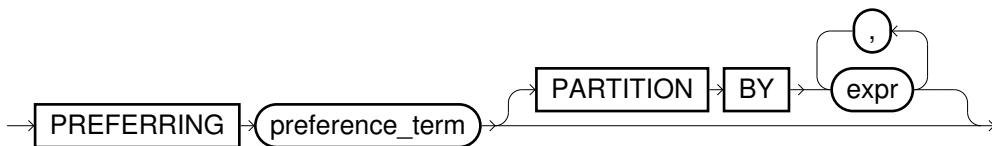
`from_item ::=`



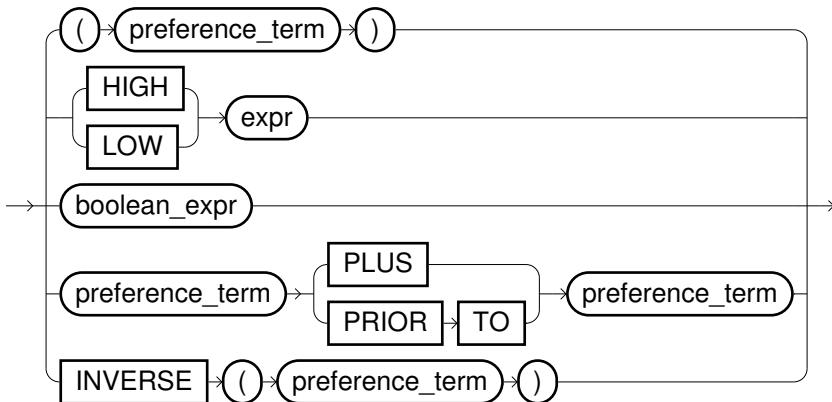
`connect_by_clause ::=`



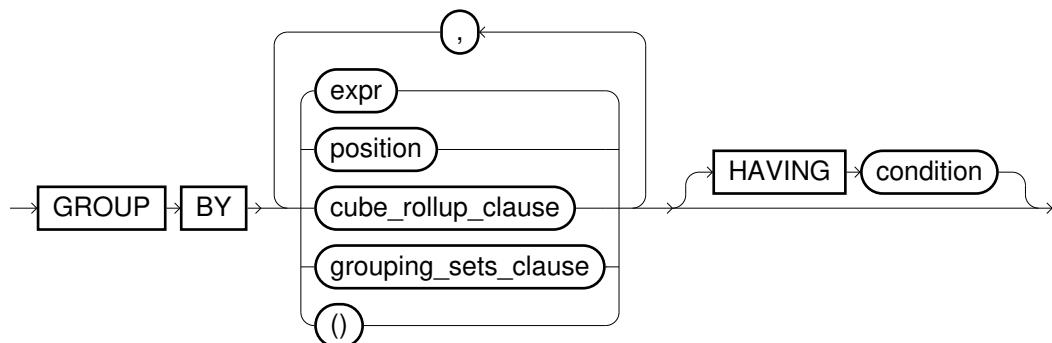
`preferring_clause ::=`



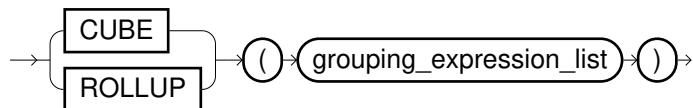
`preference_term ::=`



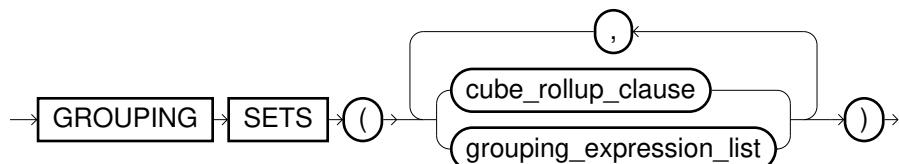
`group_by_clause ::=`



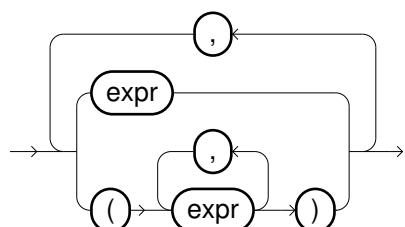
`cube_rollup_clause ::=`



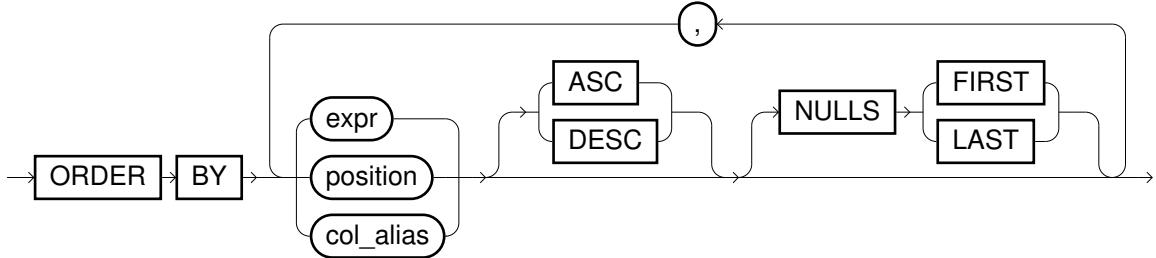
`grouping_sets_clause ::=`



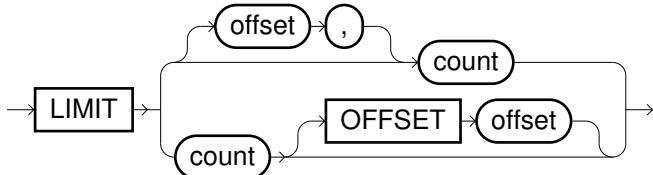
`grouping_expression_list ::=`



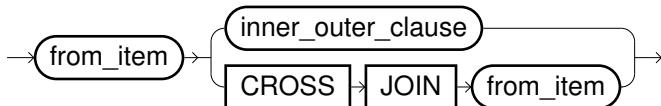
`order_by_clause ::=`



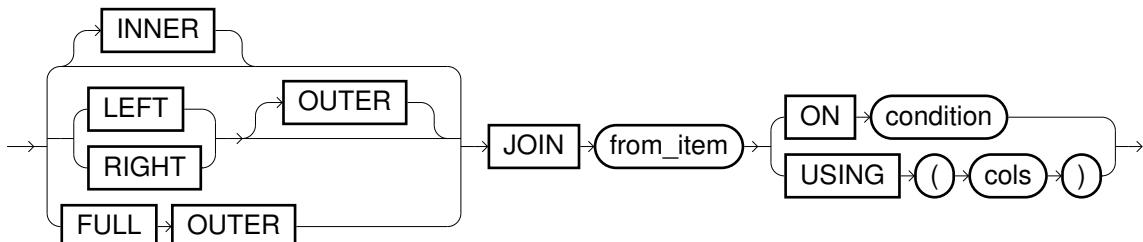
limit_clause ::=



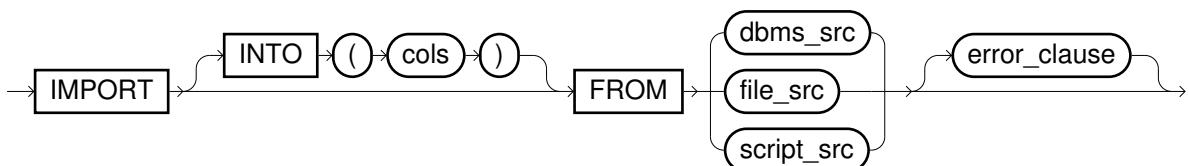
join_clause ::=



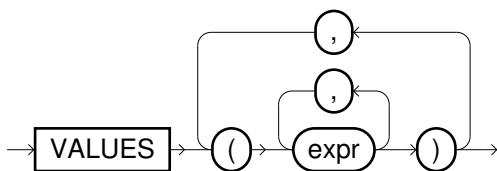
inner_outer_clause ::=



subimport ::=



values_table ::=



Anmerkung(en)

- Skalare Ausdrücke können sie einfach erzeugen, ohne eine FROM Klausel angeben zu müssen (z.B. SELECT 'abc').

- Mit Hilfe der `WITH` Klausel können temporäre Views erzeugt werden, die nur während der Ausführung des eigentlichen `SELECT`s gültig sind.
- Bei Angabe von `DISTINCT` werden identische Zeilen in der Ausgabetafelle eliminiert. Bei der Angabe von `ALL` (Default) werden hingegen alle Zeilen der Ergebnistabelle ausgegeben.
- In der `FROM` Klausel werden die Ausgangs-Tabellen bzw. Views angegeben. Mit der `values_table` können Sie statische Tabellen sehr einfach erzeugen, z.B. erzeugt `(VALUES (1,TRUE), (2,FALSE), (3,NULL)) AS t(i, b)` eine Tabelle mit zwei Spalten und drei Zeilen.
- Die `SELECT`-Liste definiert die Spalten der Ergebnistabelle. Wird `*` angegeben, so werden alle Spalten ausgewählt.
- Bei komplexeren Ausdrücken in der `SELECT`-Liste bietet sich die Verwendung von Spalten-Aliassen an. Diese können nur direkt in der `ORDER BY`-Klausel referenziert werden. Allerdings können Aliase indirekt durch das Keyword `LOCAL` auch in den anderen Klauseln (`WHERE`, `GROUP BY`, `HAVING`) und sogar in der `SELECT` Liste referenziert werden. Ein Beispiel für die indirekte Referenzierung: `SELECT ABS(x) AS x FROM t WHERE local.x>10`. Zudem definieren Spalten-Aliase die Spaltennamen der Ergebnistabelle.
- Die `WHERE` Klausel dient der Einschränkung der Ergebnismenge.
- Join-Bedingungen zwischen Tabellen können in der `WHERE` Klausel definiert werden. Durch `=` werden Äquivalenzbedingungen definiert, durch den Einsatz von `(+)` hinter einem Ausdruck wird eine Outer-Bedingung definiert. Diese Syntax wird häufig der Verwendung der `join_clause` vorgezogen.
- Die `USING` Klausel können Sie in einem Join verwenden, falls die entsprechenden Spaltennamen in beiden Tabellen identisch sind. In diesem Fall geben Sie einfach die Liste der zu verknüpfenden Spaltennamen unqualifiziert an, also ohne Tabellen- bzw. Alias-Namen. Beachten Sie ferner, dass bei einem Outer-Join eine Coalesce-Bedingung erzeugt wird. Das bedeutet, dass im Falle von Nicht-Matches der nicht gefundene Wert ausgegeben wird und nicht `NULL`. Anschließend kann nur noch die so berechnete Spalte referenziert werden, aber nicht mehr die Original-Spalten der beiden Tabellen.
- Mit der `CONNECT BY` Klausel können Sie hierarchische Bedingungen formulieren. Sie wird vor den `WHERE`-Bedingungen ausgewertet - bis auf Join-Bedingungen auf Tabellen, die im `CONNECT BY` referenziert werden.

Folgende Elemente spielen bei der `CONNECT BY` Klausel eine Rolle (ein anschauliches Beispiel finden Sie weiter unten):

START WITH Durch diese Bedingung wird die Anzahl an Start-Knoten (Root-Knoten) im Graphen definiert.

condition(s) Hier können diverse Bedingungen definiert werden. Eine hierarchische Verknüpfung zwischen Vater- und Sohn-Zeilen wird durch das Schlüsselwort `PRIOR` definiert (z.B. `PRIOR employee_id = manager_id`). Wird keine solche `PRIOR`-Bedingung spezifiziert, wird das Kreuzprodukt berechnet. So liefert beispielsweise das Statement `SELECT LEVEL FROM dual CONNECT BY LEVEL<=100` eine Tabelle mit 100 Einträgen, da insgesamt 100 Kreuzprodukte mit der Tabelle `dual` durchgeführt werden.

NOCYCLE Wird diese Option gewählt, so liefert eine Query auch ein Ergebnis zurück, falls ein Zyklus existiert. In diesem Fall wird aber bei jedem Zyklus die Expansion abgebrochen.

Die folgenden Funktionen bzw. Operatoren stehen in der `SELECT`-Liste oder in den `WHERE`-Bedingungen zur Verfügung, um die Ergebnisse einer hierarchischen Query qualifizieren zu können.

SYS_CONNECT_BY_PATH(expr, char) Liefert einen String mit dem kompletten Pfad vom Root-Knoten bis zum aktuellen Knoten, bestehend aus den Werten für `expr` und getrennt durch `char`.

LEVEL Liefert die Hierarchie-Stufe einer Zeile zurück. Diese ist 1 für einen Root-Knoten, 2 für dessen direkte Söhne, usw.

PRIOR Referenziert die Vorgänger-Zeile einer Zeile. Einerseits können hierdurch die Vater-Sohn-Bedingungen definiert werden. Und andererseits kann durch diesen Operator auch auf Werte des Vorgängers zugegriffen werden. Daher sind die folgenden beiden `CONNECT BY` Bedingungen äquivalent:

- PRIOR employee_id = manager_id AND PRIOR employee_id=10.
- PRIOR employee_id = manager_id AND manager_id=10.

CONNECT_BY_ROOT

Anstatt dem Wert der Zeile wird der entsprechende Wert des zugehörigen Root-Knoten eingesetzt (z.B. würde CONNECT_BY_ROOT last_name den Namen des obersten Managers eines Mitarbeiters ausgeben, sofern in der CONNECT BY Klausel die Bedingung PRIOR employee_id = manager_id definiert wurde).

CONNECT_BY_ISLEAF

Dieser Ausdruck liefert 1, falls die Zeile ein Blatt im Graphen darstellt (also keinen Sohn mehr besitzt), ansonsten 0.

CONNECT_BY_ISCYCLE

Gibt zurück, ob die aktuelle Zeile einen Zyklus verursacht. Im Pfad (siehe oben) taucht diese Zeile genau zwei mal auf. Dieser Ausdruck kann nur in Verbindung mit der NOCYCLE-Option benutzt werden.



Bitte beachten Sie, dass sich dieses Verhalten zum Oracle-Verhalten unterscheidet. Dort wird der Vorgänger-Knoten zurückgeliefert anstatt des Verbindungsknotens im Zyklus.

Die Funktion LEVEL und der Operator PRIOR können auch direkt innerhalb der CONNECT BY Klausel benutzt werden.

- Die PREFERRING Klausel definiert einen Skyline Präferenzausdruck. Details hierzu finden Sie in [Abschnitt 3.10, Skyline](#).
- In der GROUP BY Klausel werden Gruppen definiert, über die innerhalb der SELECT-Liste aggregiert werden kann. Falls ein numerischer Wert x angegeben wird (position), so wird nach der x-ten Spalte der SELECT-Liste aggregiert. Wird GROUP BY angegeben, so müssen alle Elemente der SELECT-Liste aggregiert werden mit Ausnahme der Gruppierungsschlüssel.
- CUBE, ROLLUP und GROUPING SETS sind Erweiterungen der GROUP BY Klausel, mit denen zu den normalen Gruppierungsergebnissen weitere Zusammenfassungen berechnet werden können. Diese bilden hierarchische Aggregationslevel wie zum Beispiel Teilsummen für Tage, Monate und Jahre und lassen sich innerhalb eines GROUP BY Statements berechnen, anstatt diese Einzel-Berechnungen mittels UNION verbinden zu müssen.

Es wird unterschieden zwischen **regulären Ergebniszellen** (normale Aggregationen auf der tiefsten Ebene) und den **Superaggregations-Zeilen** (Teilsummen auf höheren Ebenen). Die Gesamtmenge der Argumente liefert die normale Aggregation (der tiefsten Ebene), die Teilmengen entsprechende Überaggregationen. Unterscheiden lassen sich die Ergebniszellen mit der Funktion [GROUPING\[_ID\]](#).

CUBE

Berechnet die Aggregationslevel für alle möglichen Kombinationen der Argumente (also 2^n Kombinationen).

Beispiel: Mittels **CUBE(countries,products)** lassen sich alle Umsätze summieren, wobei jeweils die Teil-Summen je Land/Produkt Kombination berechnet werden (*reguläre Zeilen*), aber auch die Summe je Land, die Summe je Produkt sowie die Gesamtsumme (3 zusätzliche *Superaggregations-Zeilen*).

ROLLUP

Berechnet die Aggregationslevel für die ersten n, n-1, n-2, ... 0 Argumente (also insgesamt n+1 Kombinationen). Das letzte Level entspricht der Aggregation aller Werte.

Beispiel: Mittels **ROLLUP(year,month,day)** lassen sich alle Umsätze summieren, wobei jeweils die Teil-Summen jedes Einzeldatums berechnet werden (*reguläre Zeilen*), aber auch die Summe je Jahresmonat und je Jahr sowie die Gesamtsumme (*Superaggregations-Zeilen*).

GROUPING SETS Berechnet die Aggregationslevel für die spezifizierten Kombinationen. CUBE bzw. ROLLUP sind Sonderformen der GROUPING SETS und dienen lediglich der bequemen Schreibweise.

() Ist äquivalent zu **GROUPING SETS ()**, also wird die ganze Tabelle als eine Gruppe aggregiert.

Werden mehrere hierarchische Gruppierungen mittels Komma getrennt definiert, so ist das Ergebnis die Menge aller Kombinationen der Teilgruppierungen (Kreuzprodukt). So liefert z.B. **ROLLUP(a,b),ROLLUP(x,y)** insgesamt 9 Kombinationen. Aus **(a,b), (a), ()** und **(x,y), (x), ()** entstehen die Kombinationen **(a,b,x,y), (a,b,x), (a,b), (a,x,y), (a,x), (x,y), (x), ()**.

- Mit Hilfe der HAVING Klausel kann die Auswahl der Gruppen eingeschränkt werden.
- Die ORDER BY Klausel definiert die Sortierung der Ausgabettabelle. Falls ein numerischer Wert x angegeben wird (position), so wird nach der x-ten Spalte der SELECT-Liste sortiert.

Optionen der ORDER BY Klausel:

- ASC (Default) bedeutet aufsteigende Sortierung, DESC hingegen absteigende Sortierung.
- Bei NULLS LAST (Default) werden die NULL-Werte am Schluss ausgegeben, bei NULLS FIRST hingegen am Anfang.

 Zeichenketten werden anhand ihrer binären Darstellung sortiert.

- Mit LIMIT lässt sich die Ausgabemenge auf eine bestimmte Anzahl von Zeilen begrenzen. Das optionale Offset kann nur in Verbindung mit ORDER BY benutzt werden, weil ansonsten die Ergebnismenge nicht deterministisch ist. LIMIT ist nicht erlaubt in aggregierten SELECT-Statements und innerhalb eines korrelierten Subselects eines EXISTS-Prädikates.
- Sie können mit der subimport Klausel den Import von externen Datenquellen direkt in eine Anfrage integrieren. Bitte beachten Sie hierbei folgende Punkte:

- Details zu der Benutzung von externen Datenquellen und deren Optionen finden Sie in der Beschreibung des **IMPORT** Befehls in [Abschnitt 2.2.2, Manipulation der Datenbank \(DML\)](#).
- Es wird dringend empfohlen, die Zieldatentypen explizit anzugeben (siehe Beispiel unten). Falls dies nicht geschieht, werden die Spaltennamen und -datentypen generisch ausgewählt. Für Dateien ist die Angabe jedoch zwingend.
- Lokale Dateien können nicht direkt in Queries eingebunden werden.
- Durch die Definition einer View können externe Datenquellen transparent als eine Art externe Tabelle sehr flexibel in Exasol eingebunden werden.
- Lokale Filter-Bedingungen auf solche Importe werden nicht auf die Quell-Datenbanken propagiert. Sie können dies aber über die STATEMENT Option erreichen.
- SELECT-Statements lassen sich über die [Tabellenoperatoren UNION \[ALL\], INTERSECT, MINUS](#) miteinander verknüpfen.
- Bitte beachten Sie, dass Anfragen von SELECT-Anfragen ggfs. direkt aus dem Query Cache zurückgeliefert werden, falls eine syntaktisch äquivalente Anfrage bereits vorher berechnet wurde. Details hierzu finden Sie in den Anmerkungen zum Befehl [ALTER SYSTEM](#).

Beispiel(e)

Die folgenden Beispiele beziehen sich auf diese Tabellen:

```
SELECT * FROM customers;
```

C_ID	NAME
1	smith
2	jackson

```
SELECT * FROM sales;
```

```

S_ID      C_ID      PRICE      STORE
-----  -----  -----  -----
1          1        199.99  MUNICH
2          1         9.99   TOKYO
3          2        50.00   MUNICH
4          1       643.59  TOKYO
5          2         2.99   MUNICH
6          2      1516.78 NEW YORK

SELECT store, SUM(price) AS volume FROM sales
GROUP BY store ORDER BY store DESC;

STORE      VOLUME
-----  -----
TOKYO      653.58
NEW YORK    1516.78
MUNICH     252.98

SELECT name, SUM(price) AS volume FROM
customers JOIN sales USING (c_id)
GROUP BY name ORDER BY name;

NAME      VOLUME
-----  -----
jackson    1569.77
smith      853.57

WITH tmp_view AS (SELECT name, price, store FROM customers, sales
                  WHERE customers.c_id=sales.c_id)
SELECT sum(price) AS volume, name, store FROM tmp_view
GROUP BY GROUPING SETS (name,store,());

VOLUME      NAME      STORE
-----  -----  -----
1569.77    jackson
853.57     smith
653.58      TOKYO
252.98      MUNICH
1516.78    NEW YORK
2423.34

SELECT * FROM (IMPORT INTO (v VARCHAR(1))
               FROM EXA AT my_connection
               TABLE sys.dual);

SELECT last_name, employee_id id, manager_id mgr_id,
       CONNECT_BY_ISLEAF leaf, LEVEL,
       LPAD(' ', 2*LEVEL-1)||SYS_CONNECT_BY_PATH(last_name, '/') "PATH"
  FROM employees
 CONNECT BY PRIOR employee_id = manager_id
 START WITH last_name = 'Clark'
 ORDER BY employee_id;

LAST_NAME  ID MGR_ID LEAF LEVEL PATH
-----  --  --  --  --  --
Clark      10  9      0     1    /Clark

```

Sandler	11	10	1	2	/Clark/Sandler
Smith	12	10	0	2	/Clark/Smith
Jackson	13	10	0	2	/Clark/Jackson
Taylor	14	10	1	2	/Clark/Taylor
Brown	15	12	1	3	/Clark/Smith/Brown
Jones	16	12	1	3	/Clark/Smith/Jones
Popp	17	12	1	3	/Clark/Smith/Popp
Williams	18	13	1	3	/Clark/Jackson/Williams
Johnson	19	13	1	3	/Clark/Jackson/Johnson

Tabellenoperatoren UNION [ALL], INTERSECT, MINUS

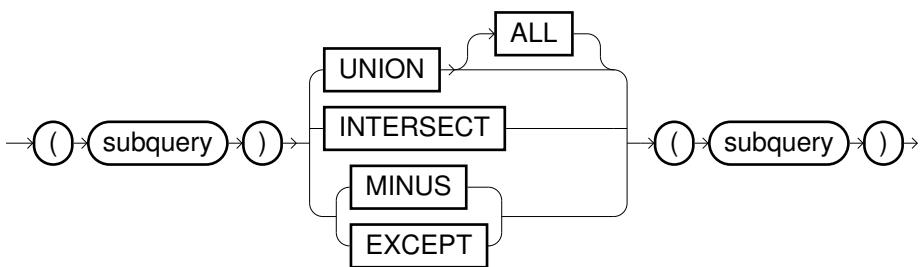
Zweck

Um Ergebnisse verschiedener Anfragen miteinander kombinieren zu können, existieren die Tabellenoperatoren UNION ALL, UNION, INTERSECT und MINUS (=EXCEPT). Diese berechnen aus zwei Subqueries die Vereinigungsmenge, die Vereinigungsmenge ohne Duplikate, die Schnittmenge ohne Duplikate und die Differenzmenge ohne Duplikate.

UNION ALL	Vereinigungsmenge aus beiden Subqueries. Alle Zeilen aus den beiden Operanden werden berücksichtigt.
UNION	Vereinigungsmenge aus beiden Subqueries ohne Duplikate. Alle Zeilen aus den beiden Operanden werden berücksichtigt. Dabei werden doppelte Einträge im Ergebnis eliminiert.
INTERSECT	Schnittmenge aus beiden Subqueries ohne Duplikate. Im Ergebnis werden alle Zeilen berücksichtigt, die in beiden Operanden vorkommen. Dabei werden doppelte Einträge im Ergebnis eliminiert.
MINUS bzw. EXCEPT	Differenzmenge aus beiden Subqueries ohne Duplikate. Das Ergebnis besteht aus denjenigen Zeilen des linken Operanden, die nicht im rechten Operanden existieren. Dabei werden doppelte Einträge im Ergebnis eliminiert.

Syntax

table_operators ::=



Anmerkung(en)

- Die Tabellenoperatoren (bis auf UNION ALL) sind teure Operationen und können gerade bei sehr großen Tabellen zu Performance-Problemen führen. Dies liegt in erster Linie daran, dass aus der Ergebnismenge doppelte Elemente entfernt werden müssen.
- Die Anzahl der Spalten der beiden Operanden müssen übereinstimmen und die Datentypen der Spalten der beiden Operanden kompatibel sein.
- Als Spaltennamen für das Ergebnis werden die Namen des linken Operanden verwendet.
- Es können auch mehrere Tabellenoperatoren hintereinander kombiniert werden. Dabei hat INTERSECT eine höhere Priorität als UNION [ALL] und MINUS. Innerhalb von UNION [ALL] und MINUS wird von links nach rechts ausgewertet. Allerdings sollten aufgrund der Übersichtlichkeit stets Klammern benutzt werden.

- Die Bezeichnung EXCEPT stammt vom SQL Standard, MINUS ist ein Alias und wird z.B. von Oracle unterstützt. Exasol unterstützt beide Alternativen.

Beispiel(e)

Die folgenden Beispiele beziehen sich auf diese Tabellen:

```
SELECT * FROM t1;
```

I1	C1
1	abc
2	def
3	abc
3	abc
5	xyz

```
SELECT * FROM t2;
```

I2	C2
1	abc
	abc
3	
4	xyz
4	abc

```
(SELECT * from T1) UNION ALL (SELECT * FROM T2);
```

I1	C1
1	abc
2	def
3	abc
3	abc
5	xyz
1	abc
	abc
3	
4	xyz
4	abc

```
(SELECT * from T1) UNION (SELECT * FROM T2);
```

I1	C1
1	abc
3	abc
4	abc
	abc
2	def
4	xyz
5	xyz
3	

```
(SELECT * from T1) INTERSECT (SELECT * FROM T2);
```

```
I1          C1  
----- ---  
1 abc
```

```
(SELECT * from T1) MINUS (SELECT * FROM T2);
```

```
I1          C1  
----- ---  
3 abc  
2 def  
5 xyz
```

2.2.5. Überprüfung der Datenqualität

Mit Hilfe der Data Quality Approval Statements kann der Qualitätszustand der Daten in der Datenbank analysiert und gesichert werden.

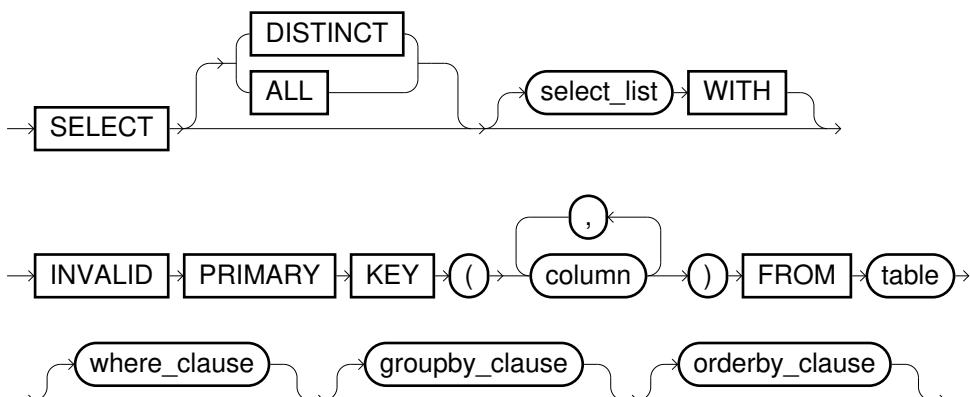
Überprüfung der Primärschlüsseleigenschaft (PRIMARY KEY)

Zweck

Mit diesem Konstrukt kann überprüft werden, ob eine Menge von Spalten die Primärschlüsseleigenschaft besitzt. Dies ist dann der Fall, wenn in den angegebenen Spalten keine Datensätze doppelt vorkommen und keine NULL-Werte vorhanden sind. Zeilen, die die Primärschlüsseleigenschaft verletzen, werden selektiert.

Syntax

select_invalid_primary_key ::=



Anmerkung(en)

- In der Formulierung ohne `select_list` werden die auf die Primärschlüsseleigenschaft zu überprüfenden Spalten ungültiger Zeilen selektiert.
- ROWNUM ist in Kombination mit diesem Statement nicht verwendbar.
- Die Überprüfung der Primärschlüsseleigenschaft findet unmittelbar auf der im `FROM` angegebenen Tabelle statt. WHERE, GROUP BY etc. werden erst danach auf die Tabelle mit den Spalten angewendet, die die Eigenschaft verletzen.

Beispiel(e)

Die folgenden Beispiele beziehen sich auf diese Tabelle:

```

SELECT * FROM t1;

NR      NAME       FIRST_NAME
-----
1  meiser     inge
2  mueller    hans
3  meyer      karl
3  meyer      karl
5  schmidt   ulla
6           benno
2  fleischer jan
  
```

```
SELECT first_name, name WITH INVALID PRIMARY KEY (nr) from T1;
```

FIRST_NAME NAME

hans	mueller
jan	fleischer
karl	meyer
karl	meyer

```
SELECT * WITH INVALID PRIMARY KEY (nr,name) from T1;
```

NR	NAME	FIRST_NAME
3	meyer	karl
3	meyer	karl
6		benno

```
SELECT INVALID PRIMARY KEY (first_name) from T1;
```

FIRST_NAME

karl
karl

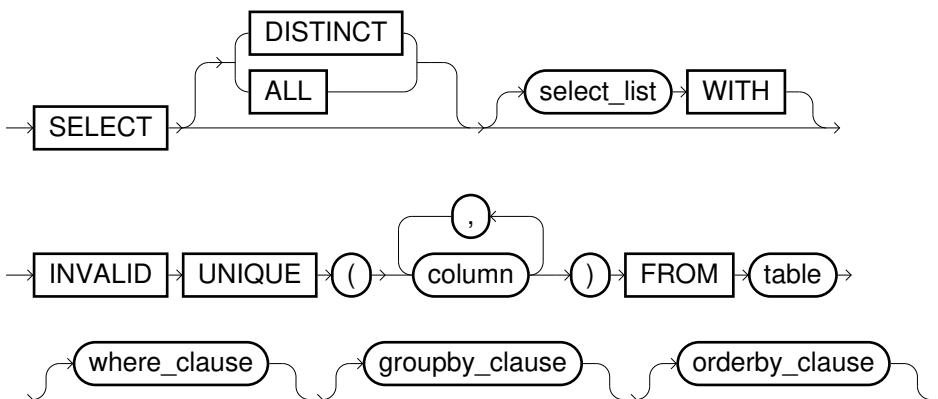
Überprüfung der Eindeutigkeit (UNIQUE)

Zweck

Mit diesem Konstrukt kann überprüft werden, ob die Zeilen einer Menge von Spalten cols eindeutig sind. Dies ist dann der Fall, wenn in den angegebenen Spalten cols keine Datensätze doppelt vorkommen. Zeilen, die in den angegebenen Spalten cols ausschließlich NULL-Werte besitzen, werden als eindeutig gewertet (selbst dann, wenn es mehrere davon gibt). Nicht eindeutige Zeilen werden selektiert.

Syntax

select_invalid_unique ::=



Anmerkung(en)

- In der Formulierung ohne select_list werden die auf Eindeutigkeit zu überprüfenden Spalten ungültiger Zeilen selektiert.

- ROWNUM ist in Kombination mit diesem Statement nicht verwendbar.
- Die Überprüfung der Eindeutigkeit findet unmittelbar auf der im FROM angegebenen Tabelle statt. WHERE, GROUP BY etc. werden erst danach auf die Tabelle mit den Spalten angewendet, die die Eindeutigkeit verletzen.

Beispiel(e)

Die folgenden Beispiele beziehen sich auf diese Tabelle:

```
SELECT * FROM t1;

NR      NAME      FIRST_NAME
-----
1 meiser    inge
2 mueller   hans
3 meyer     karl
3 meyer     karl
5 schmidt   ulla
6           benno
2 fleischer jan
3
```

```
SELECT INVALID UNIQUE (first_name, name) from T1;

FIRST_NAME NAME
-----
karl      meyer
karl      meyer
```

```
SELECT * WITH INVALID UNIQUE (nr,name) from T1;

NR      NAME      FIRST_NAME
-----
3 meyer    karl
3 meyer    karl
```

```
SELECT first_name WITH INVALID UNIQUE (nr, name, first_name) from T1;

FIRST_NAME
-----
karl
karl
```

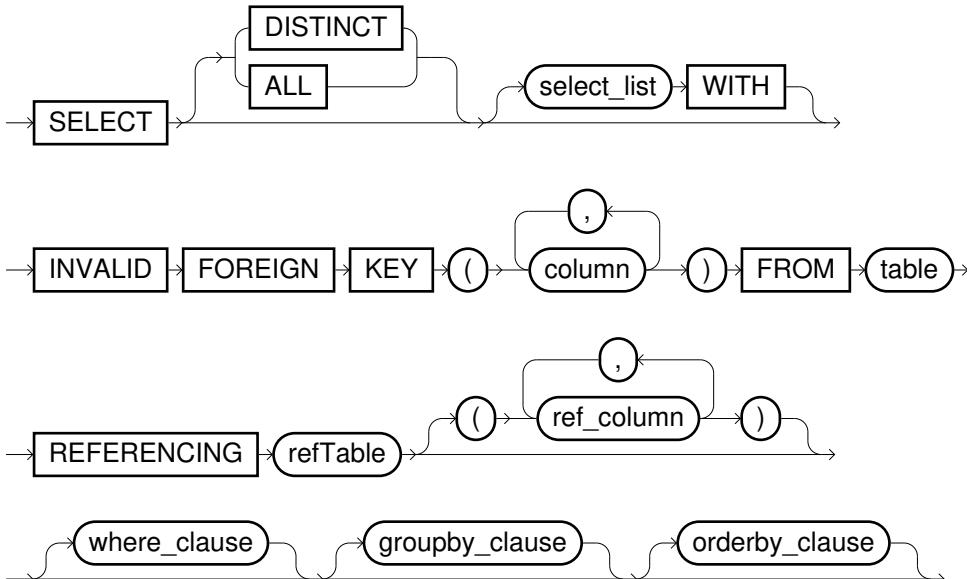
Überprüfung der Fremdschlüsseleigenschaft (FOREIGN KEY)

Zweck

Selektiert diejenigen Zeilen, die die angegebene Fremdschlüsseleigenschaft verletzen. Dies sind Zeilen, deren Werte der spezifizierten Spalten NULL sind oder in den definierten Spalten der referenzierten Tabelle nicht vorkommen.

Syntax

```
select_invalid_foreign_key ::=
```



Anmerkung(en)

- Sinnvollerweise sollten die referenzierten Spalten der Referenztabelle die Primärschlüsseleigenschaft haben. Dies wird durch dieses Statement jedoch nicht überprüft.
- In der Formulierung ohne `select_list` werden die auf Fremdschlüsseleigenschaft zu überprüfenden Spalten ungültiger Zeilen selektiert.
- ROWNUM ist in Kombination mit diesem Statement nicht verwendbar.
- Die Überprüfung der Fremdschlüsseleigenschaft findet unmittelbar auf der im FROM angegebenen Tabelle statt. WHERE, GROUP BY etc. werden erst danach auf die Tabelle mit den Spalten angewendet, die die Eigenschaft verletzen.

Beispiel(e)

Die folgenden Beispiele beziehen sich auf diese Tabellen:

```
SELECT * FROM t1;
```

NR	NAME	FIRST_NAME
1	meiser	inge
2	mueller	hans
3	meyer	karl
3	meyer	karl
5	schmidt	ulla
6		benno
2	fleischer	jan

```
SELECT * FROM t2;
```

ID	NAME	FIRST_NAME
1	meiser	otto
2	mueller	hans
3	meyer	karl
5	schmidt	ulla

```
6          benno
7 fleischer jan
```

```
SELECT first_name, name WITH INVALID FOREIGN KEY (nr) from T1
    REFERENCING T2 (id);

-- Leeres Ergebnis weil alle Werte von nr in der Spalte id existieren

FIRST_NAME NAME
-----
```

```
SELECT * WITH INVALID FOREIGN KEY (first_name, name) from T1
    REFERENCING T2;

NR      NAME      FIRST_NAME
----- 
1   meiser    inge
6   benno     benno
```

```
SELECT INVALID FOREIGN KEY (nr, first_name, name) from T1
    REFERENCING T2 (id, first_name, name);

NR      FIRST_NAME NAME
----- 
1   inge      meiser
6   benno     benno
2   jan       fleischer
```

2.2.6. Sonstige Befehle

In diesem Kapitel werden diejenigen SQL-Befehle erläutert, die sich nicht in eines der vorherigen Kapitel einordnen lassen.

COMMIT

Zweck

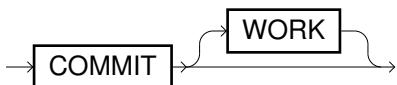
Mit dem COMMIT-Befehl werden die Änderungen der aktuellen Transaktion persistent in der Datenbank gespeichert.

Vorbedingung(en)

- Keine.

Syntax

commit ::=



Anmerkung(en)

- Das Keyword WORK ist optional und wird lediglich aus Konformitätsgründen zum SQL Standard unterstützt.
- Das automatische Ausführen eines COMMITs nach jedem SQL-Statement ist mit dem EXAplus-Befehl **SET AUTOCOMMIT** möglich.
- Weitere Informationen zum Thema Transaktionen finden Sie unter [Abschnitt 3.1, Transaktions-Management](#)

Beispiel(e)

```

CREATE TABLE t (i DECIMAL);
INSERT INTO t values (1);
COMMIT;
SELECT COUNT(*) FROM t;

COUNT(*)
-----
1

-- Tabellendaten sind bereits persistent (COMMIT)
ROLLBACK;

SELECT COUNT(*) FROM t;

COUNT(*)
-----
1

```

ROLLBACK

Zweck

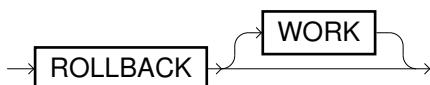
Mit dem ROLLBACK-Befehl werden die Änderungen der aktuellen Transaktion zurückgenommen.

Vorbedingung(en)

- Keine.

Syntax

rollback ::=



Anmerkung(en)

- Das Keyword WORK ist optional und wird lediglich aus Konformitätsgründen zum SQL Standard unterstützt.
- Weitere Informationen zum Thema Transaktionen finden Sie unter [Abschnitt 3.1, Transaktions-Management](#)

Beispiel(e)

```
CREATE TABLE t (i DECIMAL);
COMMIT;

INSERT INTO t values (1);
SELECT COUNT(*) FROM t;

COUNT(*)
-----
1

ROLLBACK;

SELECT COUNT(*) FROM t;

COUNT(*)
-----
0
```

EXECUTE SCRIPT

Zweck

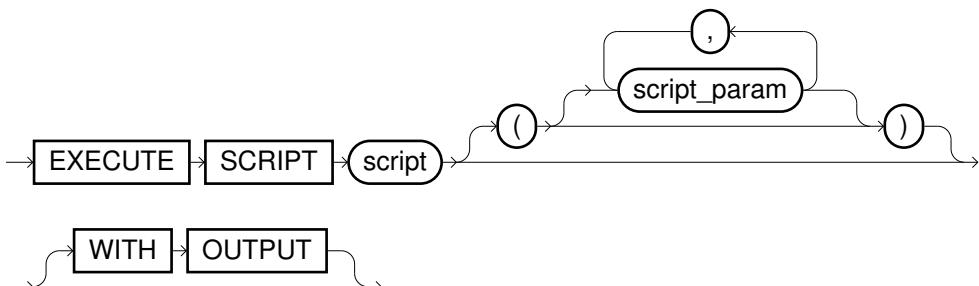
Mit diesem Befehl wird ein Skript ausgeführt.

Vorbedingung(en)

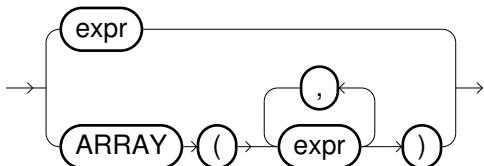
- System-Privileg EXECUTE ANY SCRIPT, Objekt-Privileg EXECUTE auf dem Skript oder muss der Besitzer sein.

Syntax

execute_script ::=



script_param ::=



Anmerkung(en)

- Erzeugt wird ein Skript durch den Befehl [CREATE SCRIPT](#), gelöscht mittels [DROP SCRIPT](#).
- Eine ausführliche Erläuterung des Sprachumfangs von Skripten finden Sie in [Abschnitt 3.5, Scripting-Programmierung](#).
- Der Inhalt sowie die Parameter eines Skriptes stehen u.a. in der Systemtabelle [EXA_ALL_SCRIPTS](#) (siehe [Anhang A, Systemtabellen](#)).
- Der Rückgabewert eines Skriptes kann entweder eine Tabelle oder ein Rowcount sein und wird im [CREATE SCRIPT](#) Befehl spezifiziert (siehe hierzu Abschnitt [Rückgabewerte eines Skriptes](#) in [Abschnitt 3.5, Scripting-Programmierung](#)).
- Bei Angabe der Option `WITH OUTPUT` wird der Rückgabewert des Skriptes ignoriert und stets eine Ergebnistabelle zurückgeliefert, die aus allen Debug-Ausgaben besteht, die mittels der Funktion `output()` während der Ausführung erzeugt wurden (siehe hierzu Abschnitt [Debug-Output](#) in [Abschnitt 3.5, Scripting-Programmierung](#)).
- Im Gegensatz zu Views wird ein Skript mit den Rechten des ausführenden Nutzers ausgeführt und nicht mit den Rechten derjenigen Person, die das Skript mittels [CREATE SCRIPT](#) erzeugt hat.

Beispiel(e)

```

EXECUTE SCRIPT script_1;
EXECUTE SCRIPT script_2 (1,3,'ABC');
EXECUTE SCRIPT script_3 (ARRAY(3,4,5));
  
```

KILL

Zweck

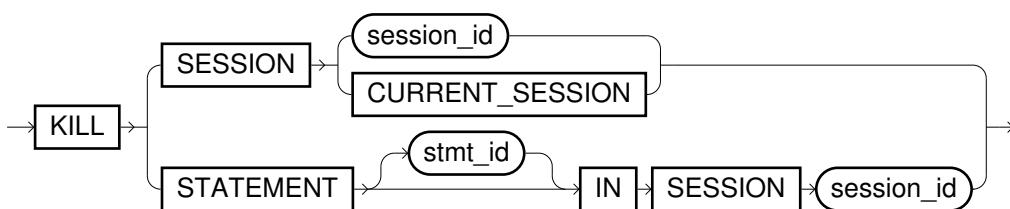
Mit diesem Befehl kann eine Benutzer-Session bzw. -Query beendet werden.

Vorbedingung(en)

- Zum Beenden der Session bzw. Query eines fremden Nutzers wird das System-Privileg KILL ANY SESSION benötigt.

Syntax

kill::=



Anmerkung(en)

- Im Falle von KILL SESSION erhält der entsprechende Nutzer eine Fehlermeldung und wird von der Datenbank auseloggt.
- KILL STATEMENT beendet das aktuelle Statement eines Nutzers, beendet aber nicht dessen Session. Durch Angabe einer Statement-Id (siehe hierzu auch [EXA_ALL_SESSIONS](#)) kann das Beenden auf einen bestimmten Befehl in der Session begrenzt werden. Falls dieses Statement jedoch nicht mehr existiert (z.B. bereits beendet), wird eine entsprechende Exception geworfen. Das Beenden eines Statements verhält sich identisch zum Query Timeout (siehe [ALTER SESSION](#) bzw. [ALTER SYSTEM](#)): Zunächst wird einige Sekunden gewartet, ob die Query einen Abbruchpunkt erreicht und mit einer Exception endet. Schafft sie dies nicht, weil keine Abbruchpunkte definiert sind (z.B. im LUA) oder sie durch Plattenzugriffe verlangsamt ist, wird die Query hart abgebrochen und die Transaktion zurückgerollt (inkl. interner Reconnect). Ist das betroffene Statement Teil einer [EXECUTE SCRIPT](#) Ausführung, so wird damit das gesamte Skript abgebrochen.

Beispiel(e)

```

KILL STATEMENT IN SESSION 7792436882684342285
KILL SESSION 7792436882684342285
  
```

ALTER SESSION

Zweck

Mit diesem Befehl kann die aktuelle Benutzer-Session konfiguriert werden.

Vorbedingung(en)

- Keine.

Syntax

alter_session::=



Anmerkung(en)

- Die sessionbasierten Parameter werden mit den systemweiten Parametern initialisiert (siehe [ALTER SYSTEM](#)), können aber mit ALTER SESSION überschrieben werden. Die aktuellen Einstellungen lassen sich in der Systemtabelle [EXA_PARAMETERS](#) ablesen.
- Sobald sich ein Benutzer ausloggt, gehen die mittels ALTER SESSION geänderten Einstellungen wieder verloren.
- Folgende Parameter können verändert werden:

TIME_ZONE

Definiert die Zeitzone, in der Zeitstempel vom Typ TIMESTAMP WITH LOCAL TIME ZONE interpretiert werden sollen. Informationen hierzu finden Sie im Abschnitt [Datum/Zeit Datentypen](#) in [Abschnitt 2.3, Datentypen](#). Die Liste der unterstützten Zeitzonen finden Sie in der Systemtabelle [EXA_TIME_ZONES](#). Die Funktion [SESSIONTIMEZONE](#) liefert die aktuell gesetzte Session-Zeitzone.

TIME_ZONE_BEHAVIOR

Definiert das Verhalten bei zweideutigen bzw. invaliden Zeitstempeln innerhalb einer bestimmten Zeitzone. Informationen hierzu finden Sie im Abschnitt [Datum/Zeit Datentypen](#) in [Abschnitt 2.3, Datentypen](#).

Definiert as Verhalten der +/- Operatoren:

- INTERVAL - Die Differenz zweier Datumswerte liefert ein Interval, und beim Addieren einer Dezimalzahl wird diese zu einer Ganzzahl gerundet, so dass stets eine gewisse Zahl ganzer Tage addiert wird.
- DOUBLE - Die Differenz zweier Datumswerte liefert einen Double, und beim Addieren einer Dezimalzahl werden die Nachkommastellen als Anteil eines Tages umgerechnet (Stunden, Minuten, ...).

NLS_DATE_FORMAT

Stellt das Datumsformat ein, das für Konvertierungen zwischen Datum und Zeichenketten verwendet wird. Mögliche Formate sind in [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#) zu finden.

NLS_TIMESTAMP_FORMAT

Stellt das Zeitstempelformat ein, das für Konvertierungen zwischen Zeitstempeln und Zeichenketten verwendet wird. Mögliche Formate sind in [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#) zu finden.

NLS_DATE_LANGUAGE

Stellt die Sprache des Datumsformats ein, die bei abgekürzten und ausgeschriebenen Monats- und Tagesformaten (siehe [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#)) verwendet wird. Mögliche Sprachen sind Englisch (ENG=Default) und Deutsch (DEU). Hierbei kann die englische Sprache durch ENG bzw. ENGLISH und die deutsche Sprache mittels DEU, DEUTSCH und GERMAN eingestellt werden.

NLS_FIRST_DAY_OF_WEEK

Definiert den ersten Tag einer Woche (Zahl zwischen 1-7 für Montag-Sonntag).

NLS_NUMERIC_CHARACTERS

Definiert die Komma- und Gruppenzeichen, die für die Darstellung von Zahlen verwendet werden. Dieser Parameter ist auch für die Verwendung von numerischen Format-Modellen relevant (siehe auch [Abschnitt 2.6.2, Numerische Format-Modelle](#)).

DEFAULT_LIKE_ESCAPE_CHARACTER

Definiert das Escape-Zeichen für das LIKE Prädikat (siehe [Abschnitt 2.8, Prädikate](#)), falls dieses nicht explizit angegeben wird.

QUERY_CACHE

Der Parameter QUERY_CACHE definiert die Verwendung eines Caches für SELECT-Anfragen. Wird die syntaktisch identische Anfrage mehrmals abgeschickt (bis auf Groß-/Kleinschreibung, Leerzeichen usw.), so kann die Datenbank das Ergebnis direkt aus einem Puffer lesen, sofern die benutzten Schemaobjekte zwischenzeitlich nicht geändert wurden.

Folgende Werte können definiert werden:

- ON - Der Query-Cache wird aktiv genutzt, d.h. Anfrageergebnisse in den Cache gespeichert und daraus ggfs. gelesen.
- OFF - Der Query-Cache wird nicht genutzt.
- READONLY - Ergebnisse werden aus dem Cache gelesen, aber keine neuen Anfragen im Cache gespeichert.

Ob eine Query über den Query-Cache ausgelesen wurde, können Sie in den entsprechenden Systemtabellen (z.B. [EXA_SQL_LAST_DAY](#)) über die Spalte EXECUTION_MODE finden.

QUERY_TIMEOUT

Mittels QUERY_TIMEOUT kann definiert werden, wie lange eine Query maximal in Sekunden dauern darf. Falls eine Query diese Zeit überschreitet, wird zunächst einige Sekunden gewartet, ob sie einen Abbruchpunkt erreicht und mit einer Exception endet. Schafft sie dies nicht, weil keine Abbruchpunkte definiert sind (z.B. im LUA) oder sie durch Plattenzugriffe verlangsamt ist, wird die Query hart abgebrochen und die Transaktion zurückgerollt (inkl. interner Reconnect). Falls eine Abfrage wegen Transaktionskonflikten blockiert ist (im Zustand Waiting for session), so zählt diese Zeit mit. Bei [EXECUTE SCRIPT](#) wird QUERY_TIMEOUT auf das gesamte Skript angewandt, d.h. mit Erreichen des Timeout wird auch das im Skript ausgeführte Statement abgebrochen. Eine Änderung des QUERY_TIMEOUT im Skript wird erst nach dem Skript aktiv. Im Ausgangszustand ist QUERY_TIMEOUT auf '0' gesetzt, was keine Beschränkung bedeutet.

NICE

Wird der Parameter NICE auf 'ON' gesetzt, so wird die Session niedriger priorisiert. Das Ressourcen-Management teilt das Gewicht des Nutzers in diesem Fall durch die Anzahl aktuell aktiver Queries. Hierdurch wird auch Prioritätsgruppen-übergreifend erreicht, dass die Session möglichst ressourcen-schonend verarbeitet wird. Details zu Prioritäten finden Sie in [Abschnitt 3.3, Prioritäten](#).

CONSTRAINT_STATE_DEFAULT

Dieser Parameter bestimmt, in welchem Status ('ENABLE' bzw. 'DISABLE') Constraints defaultmäßig erstellt werden, falls der Status nicht explizit angegeben wird (siehe auch [CREATE TABLE](#) bzw. [ALTER TABLE \(constraints\)](#)).

PROFILE

Der Parameter aktiviert/deaktiviert das Profiling (Werte 'ON' bzw. 'OFF'). Details hierzu siehe [Abschnitt 3.9, Profiling](#).

SCRIPT_LANGUAGES

Definiert die Aliase für Skript-Sprachen. Details hierzu siehe [Abschnitt 3.6.5, Skript-Sprachen erweitern mittels BucketFS](#).

SQL_PREPROCESSOR_SCRIPT

Definiert ein Präprozessor-Skript. In diesem Fall wird jedes SQL Statement vor der Ausführung von diesem (regulär mittels [CREATE SCRIPT](#) erzeugten) Skript vorverarbeitet. Ausführliche Informationen zu diesem Thema finden Sie in [Abschnitt 3.8, SQL-Präprozessor](#). Details zur Skriptsprache stehen in [Abschnitt 3.5, Scripting-Programmierung](#) zur Verfügung.



Beachten Sie, dass geeignete Privilegien zur Ausführung des Skripts vergeben sein müssen.



Sie können den Präprozessor durch die Angabe des leeren Strings '' bzw. NULL deaktivieren.

Beispiel(e)

```
ALTER SESSION SET TIME_ZONE='EUROPE/BERLIN';
ALTER SESSION SET QUERY_TIMEOUT=120;
ALTER SESSION SET NLS_DATE_FORMAT='DDD-YYYY';
ALTER SESSION SET NLS_DATE_LANGUAGE='ENG';
```

```

SELECT TO_CHAR(TO_DATE('365-2007'), 'DAY-DD-MONTH-YYYY') TO_CHAR1;
TO_CHAR1
-----
MONDAY -31-DECEMBER -2007

ALTER SESSION SET NLS_NUMERIC_CHARACTERS='.,';
SELECT TO_CHAR(123123123.45, '999G999G999D99') TO_CHAR2;
TO_CHAR2
-----
123.123.123,45

```

ALTER SYSTEM

Zweck

Mit diesem Befehl können systemweite Parameter konfiguriert werden.

Vorbedingung(en)

- Das System-Privileg ALTER SYSTEM.

Syntax

alter_system::=



Anmerkung(en)

- Die sessionbasierten Parameter werden mit den systemweiten Parametern (ALTER SYSTEM) initialisiert, können aber mit dem Befehl [ALTER SESSION](#) überschrieben werden. Die aktuellen Einstellungen lassen sich in der Systemtabelle [EXA_PARAMETERS](#) ablesen.
- Die mittels ALTER SYSTEM global geänderten Werte gelten erst für neue Verbindungen zur Datenbank.
- Folgende Parameter können verändert werden:

TIME_ZONE

Definiert die Zeitzone, in der Zeitstempel vom Typ TIMESTAMP WITH LOCAL TIME ZONE interpretiert werden sollen. Informationen hierzu finden Sie im Abschnitt [Datum/Zeit Datentypen](#) in [Abschnitt 2.3, Datentypen](#). Die Liste der unterstützten Zeitzonen finden Sie in der Systemtabelle [EXA_TIME_ZONES](#). Die Funktion [SESSIONTIMEZONE](#) liefert die aktuell gesetzte Session-Zeitzone.

TIME_ZONE_BEHAVIOR

Definiert das Verhalten bei zweideutigen bzw. invaliden Zeitstempeln innerhalb einer bestimmten Zeitzone. Informationen hierzu finden Sie im Abschnitt [Datum/Zeit Datentypen](#) in [Abschnitt 2.3, Datentypen](#).

TIMESTAMP_ARITHMETIC_BEHAVIOR

Definiert das Verhalten der +/- Operatoren:

- INTERVAL - Die Differenz zweier Datumswerte liefert ein Interval, und beim Addieren einer Dezimalzahl wird diese zu einer Ganzzahl gerundet, so dass stets eine gewisse Zahl ganzer Tage addiert wird.
- DOUBLE - Die Differenz zweier Datumswerte liefert einen Double, und beim Addieren einer Dezimalzahl werden die Nachkommastellen als Anteil eines Tages umgerechnet (Stunden, Minuten, ...).

NLS_DATE_FORMAT	Stellt das Datumsformat ein, das für Konvertierungen zwischen Datum und Zeichenketten verwendet wird. Mögliche Formate sind in Abschnitt 2.6.1, Datum/Zeit Format-Modelle zu finden.
NLS_TIMESTAMP_FORMAT	Stellt das Zeitstempelformat ein, das für Konvertierungen zwischen Zeitstempeln und Zeichenketten verwendet wird. Mögliche Formate sind in Abschnitt 2.6.1, Datum/Zeit Format-Modelle zu finden.
NLS_DATE_LANGUAGE	Stellt die Sprache des Datumsformats ein, die bei abgekürzten und ausgeschriebenen Monats- und Tagesformaten (siehe Abschnitt 2.6.1, Datum/Zeit Format-Modelle) verwendet wird. Mögliche Sprachen sind Englisch (ENG=Default) und Deutsch (DEU). Hierbei kann die englische Sprache durch ENG bzw. ENGLISH und die deutsche Sprache mittels DEU, DEUTSCH und GERMAN eingestellt werden.
NLS_FIRST_DAY_OF_WEEK	Definiert den ersten Tag einer Woche (Zahl zwischen 1-7 für Montag-Sonntag).
NLS_NUMERIC_CHARACTERS	Definiert die Komma- und Gruppenzeichen, die für die Darstellung von Zahlen verwendet werden. Dieser Parameter ist auch für die Verwendung von numerischen Format-Modellen relevant (siehe auch Abschnitt 2.6.2, Numerische Format-Modelle).
DEFAULT_LIKE_ESCAPE_CHARACTER	Definiert das Escape-Zeichen für das LIKE Prädikat (siehe Abschnitt 2.8, Prädikate), falls dieses nicht explizit angegeben wird.
QUERY_CACHE	Der Parameter QUERY_CACHE definiert die Verwendung eines Caches für SELECT-Anfragen. Wird die syntaktisch identische Anfrage mehrmals abgeschickt (bis auf Groß-/Kleinschreibung, Leerzeichen usw.), so kann die Datenbank das Ergebnis direkt aus einem Puffer lesen, sofern die benutzten Schemaobjekte zwischenzeitlich nicht geändert wurden. Folgende Werte können definiert werden: <ul style="list-style-type: none"> • ON - Der Query-Cache wird aktiv genutzt, d.h. Anfrageergebnisse in den Cache gespeichert und daraus ggfs. gelesen. • OFF - Der Query-Cache wird nicht genutzt. • READONLY - Ergebnisse werden aus dem Cache gelesen, aber keine neuen Anfragen im Cache gespeichert.
QUERY_TIMEOUT	Ob eine Query über den Query-Cache ausgelesen wurde, können Sie in den entsprechenden Systemtabellen (z.B. EXA_SQL_LAST_DAY) über die Spalte EXECUTION_MODE finden. Mittels QUERY_TIMEOUT kann definiert werden, wie lange eine Query maximal in Sekunden dauern darf. Falls eine Query diese Zeit überschreitet, wird zunächst einige Sekunden gewartet, ob sie einen Abbruchpunkt erreicht und mit einer Exception endet. Schafft sie dies nicht, weil keine Abbruchpunkte definiert sind (z.B. im LUA) oder sie durch Plattenzugriffe verlangsamt ist, wird die Query hart abgebrochen und die Transaktion zurückgerollt (inkl. interner Reconnect). Falls eine Abfrage wegen Transaktionskonflikten blockiert ist (im Zustand Waiting for session), so zählt diese Zeit mit. Bei EXECUTE SCRIPT wird QUERY_TIMEOUT auf das gesamte Skript angewandt, d.h. mit Erreichen des Timeout wird auch das im Skript ausgeführte Statement abgebrochen. Eine Änderung des QUERY_TIMEOUT im Skript wird erst nach dem Skript aktiv. Im Ausgangszustand ist QUERY_TIMEOUT auf '0' gesetzt, was keine Beschränkung bedeutet. Dieser Parameter bestimmt, in welchem Status ('ENABLE' bzw. 'DISABLE') Constraints defaultmäßig erstellt werden, falls der Status nicht explizit angegeben wird (siehe auch CREATE TABLE bzw. ALTER TABLE (constraints)).
PROFILE	Der Parameter aktiviert/deaktiviert das Profiling (Werte 'ON' bzw. 'OFF'). Details hierzu siehe Abschnitt 3.9, Profiling .
SCRIPT_LANGUAGES	Definiert die Aliase für Skript-Sprachen. Details hierzu siehe Abschnitt 3.6.5, Skript-Sprachen erweitern mittels BucketFS .

SQL_PREPROCESSOR_SCRIPT

Definiert ein Präprozessor-Skript. In diesem Fall wird jedes SQL Statement vor der Ausführung von diesem (regulär mittels **CREATE SCRIPT** erzeugten) Skript vorverarbeitet. Ausführliche Informationen zu diesem Thema finden Sie in [Abschnitt 3.8, SQL-Präprozessor](#). Details zur Skriptsprache stehen in [Abschnitt 3.5, Scripting-Programmierung](#) zur Verfügung.



Beachten Sie, dass geeignete Privilegien zur Ausführung des Skripts vergeben sein müssen.



Sie können den Präprozessor durch die Angabe des leeren Strings '' bzw. NULL deaktivieren.

Beispiel(e)

```
ALTER SYSTEM SET TIME_ZONE='EUROPE/BERLIN';
ALTER SYSTEM SET TIME_ZONE_BEHAVIOR='INVALID SHIFT AMBIGUOUS ST';
ALTER SYSTEM SET NLS_DATE_FORMAT='DAY-DD-MM-MONTH-YYYY';
ALTER SYSTEM SET NLS_DATE_LANGUAGE='DEU';
ALTER SYSTEM SET NLS_FIRST_DAY_OF_WEEK=1;
ALTER SYSTEM SET NLS_NUMERIC_CHARACTERS=', .';
ALTER SYSTEM SET QUERY_TIMEOUT=120;
ALTER SYSTEM SET CONSTRAINT_STATE_DEFAULT='DISABLE';
ALTER SYSTEM SET SQL_PREPROCESSOR_SCRIPT=my_schema.my_script;
```

OPEN SCHEMA**Zweck**

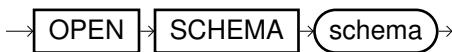
Mit diesem Befehl kann ein Schema geöffnet werden. Dies beeinflusst in erster Linie die Namensauflösung.

Vorbedingung(en)

- Keine.

Syntax

open_schema::=

**Anmerkung(en)**

- Das aktuell geöffnete Schema lässt sich mittels **CURRENT_SCHEMA** ermitteln.
- Mit dem Befehl **CLOSE SCHEMA** kann man das aktuell geöffnete Schema wieder verlassen.
- Ist kein Schema geöffnet, so müssen alle Schemaobjekte durch schemaqualifizierte Namen referenziert werden (siehe auch [Abschnitt 2.1.2, SQL-Bezeichner](#)).

Beispiel(e)

```
OPEN SCHEMA my_schema;
```

CLOSE SCHEMA

Zweck

Mit diesem Befehl wird das aktuelle Schema geschlossen. Dies beeinflusst in erster Linie die Namensauflösung.

Vorbedingung(en)

- Keine.

Syntax

close_schema::=

→ **CLOSE** → **SCHEMA** →

Anmerkung(en)

- Ist kein Schema geöffnet, so müssen alle Schemaobjekte durch schemaqualifizierte Namen referenziert werden (siehe auch [Abschnitt 2.1.2, SQL-Bezeichner](#)).

Beispiel(e)

```
OPEN SCHEMA my_schema;
SELECT * FROM my_table;
CLOSE SCHEMA;
SELECT * FROM my_table; -- Error: object MY_TABLE not found
SELECT * FROM my_schema.my_table;
```

DESC[RIBE]

Zweck

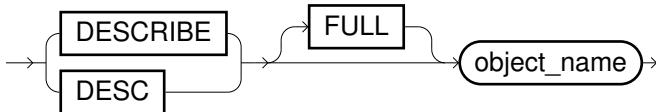
Mit diesem Befehl können Spalteninformationen zu einer gegebenen Tabelle oder View ausgegeben werden.

Vorbedingung(en)

- Ist das zu beschreibende Objekt eine Tabelle, muss eine der folgenden Bedingungen erfüllt sein:
 - Der aktuelle Benutzer besitzt eines der folgenden System-Privilegien: SELECT ANY TABLE (bzw. SELECT ANY DICTIONARY bei Systemtabellen), INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, ALTER ANY TABLE oder DROP ANY TABLE.
 - Der aktuelle Benutzer besitzt mindestens ein Objekt-Privileg auf der Tabelle.
 - Die Tabelle gehört dem aktuellen Benutzer oder einer seiner Rollen.
- Ist das zu beschreibende Objekt eine View, muss eine der folgenden Bedingungen erfüllt sein:
 - Der aktuelle Benutzer besitzt eines der folgenden System-Privilegien: SELECT ANY TABLE oder DROP ANY VIEW.
 - Der aktuelle Benutzer besitzt mindestens ein Objekt-Privileg auf der View.
 - Die View gehört dem aktuellen Benutzer oder einer seiner Rollen.

Syntax

describe ::=



Anmerkung(en)

- Die Spalte SQL_TYPE gibt den Datentyp der betreffenden Spalte an. Bei Stringtypen wird zusätzlich noch der verwendete Zeichensatz (ASCII oder UTF8) angegeben.
- Die Spalte NULLABLE gibt an, ob die Spalte auch NULL-Werte enthalten darf.
- Der Wert in DISTRIBUTION_KEY besagt, ob die Spalte Teil des Verteilungsschlüssels ist (siehe hierzu auch den [ALTER TABLE \(distribution\)](#) Befehl). Bei einer View sind die Werte dieser Spalte stets NULL.
- Wird die Option FULL angegeben, so liefert die zusätzliche Spalte COLUMN_COMMENT den Spaltenkommentar (abgeschnitten auf maximal 200 Zeichen), sofern dieser implizit im Befehl [CREATE TABLE](#) oder explizit mittels [COMMENT](#) dem gesetzt wurde.
- Der DESCRIBE Befehl kann durch DESC abgekürzt werden (z.B. **DESC my_table;**).

Beispiel(e)

```

CREATE TABLE t (i DECIMAL COMMENT IS 'id column',
                d DECIMAL(20,5),
                j DATE,
                k VARCHAR(5),
                DISTRIBUTE BY i);
DESCRIBE FULL t;

COLUMN_NAME  SQL_TYPE          NULLABLE  DISTRIBUTION_KEY  COLUMN_COMMENT
-----  -----  -----  -----
I          DECIMAL(18,0)        TRUE      TRUE            id column
D          DECIMAL(20,5)        TRUE      FALSE
J          DATE                TRUE      FALSE
K          VARCHAR(5)  UTF8    TRUE      FALSE
  
```

EXPLAIN VIRTUAL

Zweck

Dieses Statement ist nützlich, um die resultierenden Anfragen an externe Systeme zu analysieren, die vom Exasol Compiler beim Zugriff auf virtuelle Objekte erzeugt werden.

Vorbedingung(en)

- Die gleichen Rechte, die für die Ausführung der enthaltenen Query benötigt werden.

Syntax

explain_virtual ::=



Anmerkung(en)

- Details zu virtuellen Schemas finden Sie in [Abschnitt 3.7, Virtuelle Schemas](#).
- Wenn Sie in einer Anfrage auf virtuelle Objekte zugreifen und das EXPLAIN Kommando nutzen, wird die Anfrage nicht komplett ausgeführt und keine Daten von den unterliegenden Systemen transferiert. Sie wird lediglich kompiliert und optimiert, so dass möglichst viel Logik auf das externe System verlagert wird. Die Ausgabe des Befehls ist dann eine Ergebnistabelle, die die resultierenden Anfragen für die externen Systeme enthält.
- Sie können das Ergebnis des EXPLAIN Befehls in einem Subselect benutzen und somit mittels SQL weiter verarbeiten.
- Ähnliche Informationen zum Pushdown der Anfragen auf externe Systeme können Sie auch in den Profiling-Informationen finden, aber mit diesem Befehl wird der Zugriff deutlich vereinfacht.

Beispiel(e)

```
SELECT pushdown_id, pushdown_involved_tables, pushdown_sql FROM
(EXPLAIN VIRTUAL SELECT * FROM vs_impala.sample_07 WHERE total_emp>10000);

PUSHDOWN_ID PUSHDOWN_INV... PUSHDOWN_SQL
-----
 1 SAMPLE_07      IMPORT FROM JDBC AT 'jdbc:impala:<shortened>'
                      STATEMENT 'SELECT * FROM `default`.`SAMPLE_07`'
                      WHERE 10000 < `TOTAL_EMP`'
```

RECOMPRESS

Zweck

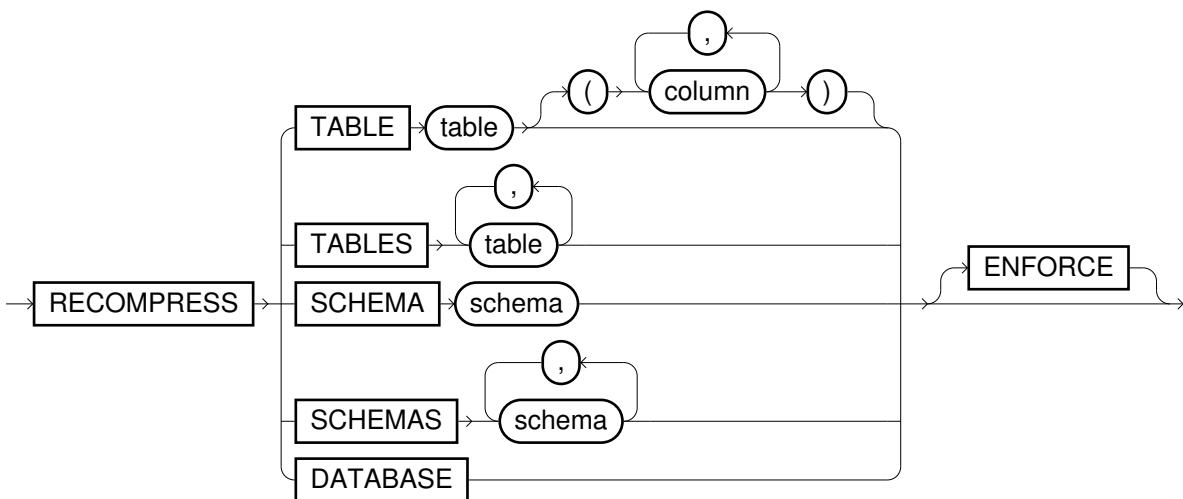
Mit diesem Befehl kann die Kompression von Tabellen verbessert werden.

Vorbedingung(en)

- Die Tabellen gehören dem Nutzer bzw. einer seiner Rollen oder er hat Zugriff auf jede der angegebenen Tabellen durch ein modifizierendes ANY TABLE-Systemprivileg oder ein modifizierendes Objektprivileg. Modifizierende Privilegien sind alle außer SELECT.

Syntax

recompress::=



Anmerkung(en)



Dieser Befehl führt bis auf die Einzeltabelle-Variante **RECOMPRESS TABLE** vor und nach der Kompression jeder einzelnen Tabelle ein implizites **COMMIT** aus!

- Dieser Befehl kann eine lange Ausführungszeit haben. Insbesondere muss jedes COMMIT die kompletten Daten einer Tabelle auf die Festplatten schreiben.
- Eine Rekomprimierung einer Tabelle macht in erster Linie dann Sinn, wenn sehr viele neue Daten eingefügt wurden. Nach Ausführung dieses Befehls muss die Kompressionsrate nicht zwangsläufig signifikant besser sein als vorher.
- Sie können zur zielgerichteten Komprimierung auch einzelne Spalten einer Tabelle selektieren.
- Die komprimierte Datenmenge und die Rohdatenmenge einer Tabelle können Sie in der Systemtabelle **EXA_ALL_OBJECT_SIZES** ablesen.

Beispiel(e)

```
RECOMPRESS TABLE t1 (column_1);
RECOMPRESS TABLES t2,t3 ENFORCE;
```

REORGANIZE

Zweck

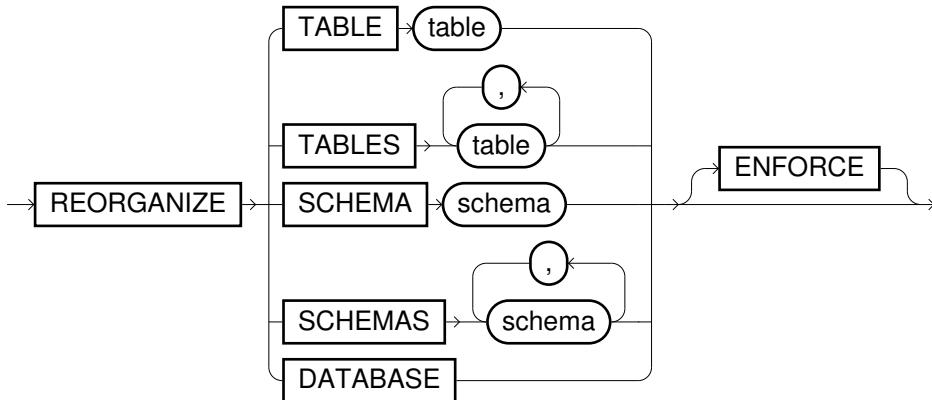
Mit diesem Befehl kann eine Datenbank intern reorganisiert werden. Dieser Befehl ist nötig, falls das DB Cluster vergrößert werden soll. Hierdurch werden die Daten der Tabellen über die neuen Knoten neu verteilt und der Verteilungszustand wiederhergestellt (siehe hierzu auch [ALTER TABLE \(distribution\)](#)). Daher empfiehlt es sich, diesen Befehl sofort nach der Clustervergrößerung durchzuführen. Andernfalls wird die Performance des Systems anfangs sogar langsamer sein als vor der Clustervergrößerung.

Vorbedingung(en)

- Die Tabellen gehören dem Nutzer bzw. einer seiner Rollen oder er hat Zugriff auf jede der angegebenen Tabellen durch ein modifizierendes ANY TABLE-Systemprivileg oder ein modifizierendes Objektpflegeprivileg. Modifizierende Privilegien sind alle außer SELECT.

Syntax

reorganize ::=



Anmerkung(en)

- Im Falle der DATABASE Option werden alle existierenden Tabellen reorganisiert.
- Bei mehreren Tabellen werden diese nacheinander reorganisiert und ein implizites COMMIT nach jeder Tabelle durchgeführt. Dies verringert Transaktionskonflikte und erhöht möglichst frühzeitig die Performance beim Zugriff auf die bereits reorganisierten Tabellen.
- Das Reorganisieren umfasst folgende Dinge:
 - Auffüllen der als gelöscht markierten Zeilen, also eine Art Defragmentierung. Dies passiert grundsätzlich automatisch nach Erreichen eines bestimmten Schwellwertes, kann aber über diesen Befehl explizit durchgeführt werden. Weitere Details werden im [DELETE](#) Befehl erläutert.
 - Rekomprimierung derjenigen Spalten, deren Kompressionsrate über die Zeit signifikant gesunken ist.
 - Neuerzeugung aller internen Indizes.
 - Wiederherstellung des Verteilungszustands (DISTRIBUTE BY Schlüssel) und gleichmäßiges Verteilen der Daten über die Cluster-Knoten nach einer Cluster-Vergrößerung.
- Falls die ENFORCE Option angegeben wurde, werden alle definierten Tabellen reorganisiert. Ansonsten nur diejenigen Tabellen, bei denen dies wirklich nötig ist (z.B. aufgrund einer Cluster-Vergrößerung oder falls sehr viele Zeilen als gelöscht markiert sind).
- Dieser Befehl kann eine lange Ausführungszeit haben. Insbesondere muss jedes COMMIT z.T. große Datenmengen auf die Festplatten schreiben.

Beispiel(e)

```
REORGANIZE DATABASE;
```

TRUNCATE AUDIT LOGS

Zweck

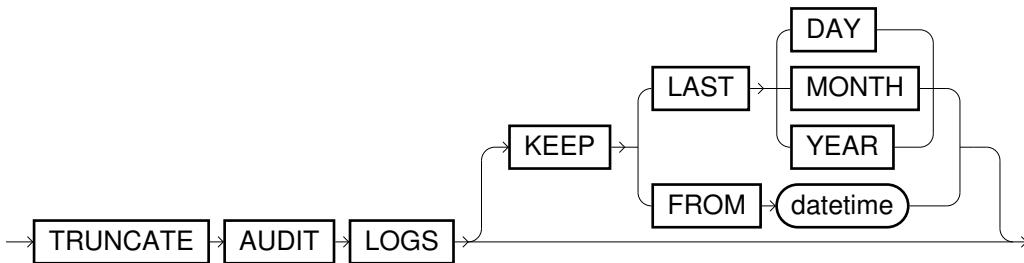
Mit diesem Befehl können die Einträge der Systemtabellen [EXA_DB_AUDIT_SESSIONS](#), [EXA_DB_AUDIT_SQL](#), [EXA_USER_TRANSACTION_CONFLICTS_LAST_DAY](#) und [EXA_DB TRANSACTION CONFLICTS](#) gelöscht werden. Dies kann z.B. dann nötig sein, wenn die gesamte Datenmenge zu groß wird.

Vorbedingung(en)

- Der Nutzer muss die Rolle DBA inne haben.

Syntax

truncate_audit_logs ::=



Anmerkung(en)

- Über die KEEP-Option können die Audit-Logs eines bestimmten Zeitraums vom Löschen ausgenommen werden.

Beispiel(e)

```

TRUNCATE AUDIT LOGS;
TRUNCATE AUDIT LOGS KEEP LAST MONTH;
TRUNCATE AUDIT LOGS KEEP FROM '2009-01-01';
  
```

FLUSH STATISTICS

Zweck

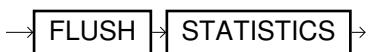
Statistiken für statistische Systemtabellen werden vom System kontinuierlich gesammelt, aber nur in bestimmten Abständen mittels COMMIT für die Systemtabellen übermittelt. Mit dem FLUSH STATISTICS-Befehl kann dieses COMMIT erzwungen werden.

Vorbedingung(en)

- Keine.

Syntax

flush_statistics ::=



Anmerkung(en)

- Um die aktualisierten Statistiken abzufragen, muss im Client ggf. eine neue Transaktion begonnen werden.
- Der Befehl verursacht im DBMS zusätzlichen Aufwand und sollte daher mit Bedacht eingesetzt werden. Die Statistiken werden ohnehin minütlich aktualisiert.
- Die verfügbaren Statistiken sind unter [Abschnitt A.2.3, Statistische Systemtabellen](#) beschrieben.

Beispiel(e)

```
FLUSH STATISTICS;  
COMMIT;  
SELECT * FROM EXA_USAGE_LAST_DAY ORDER BY MEASURE_TIME DESC LIMIT 10;
```

PRELOAD

Zweck

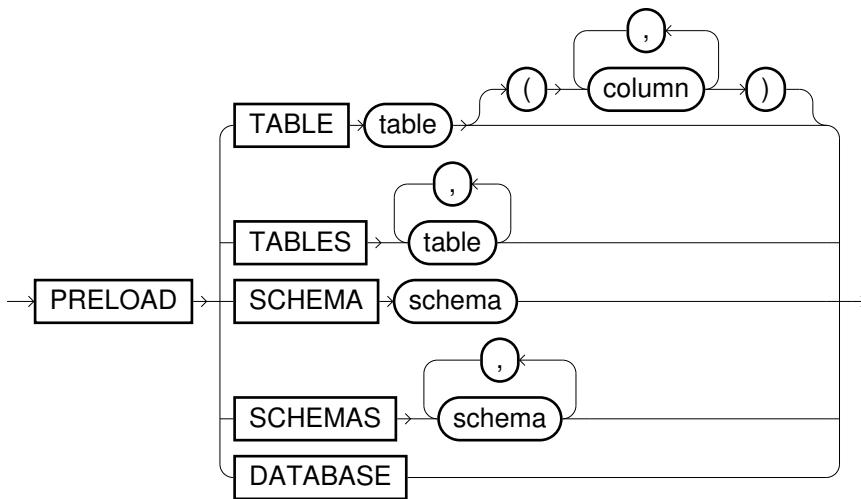
Lt bestimmte Tabellen bzw. Spalten sowie die entsprechenden internen Indizes von der Festplatte in den Hauptspeicher der Datenbank, sofern diese nicht bereits im Cache vorliegen. Aufgrund des intelligenten Speicher-Managements von Exasol empfehlen wir Nutzern allerdings, diesen Befehl lediglich fr spezielle Ausnahmefle zu benutzen. Ansonsten besteht die Gefahr einer schlechteren Gesamt-Performance.

Vorbedingung(en)

- Die Tabellen gehören dem Nutzer bzw. einer seiner Rollen bzw. er hat Lese- oder Schreib-Zugriff auf jede der angegebenen Tabellen mittels System- oder Objektprivileg.

Syntax

preload::=



Anmerkung(en)

- Bei Angabe von Schemas oder der ganzen Datenbank werden alle darin enthaltenen Tabellen in den Speicher geladen.

Beispiel(e)

```
PRELOAD TABLE t(i);
```

2.3. Datentypen

In diesem Abschnitt werden die von Exasol unterstützten Datentypen und deren Aliase erläutert. Zusätzlich zu den ANSI SQL Datentypen werden weitere Datentypen unterstützt, die die Kompatibilität zu anderen Datenbanken gewährleisten sollen.

2.3.1. Überblick über Exasol Datentypen

Die folgende Tabellen liefert eine Übersicht der in Exasol definierten Datentypen.

Tabelle 2.2. Überblick über Exasol Datentypen

Exasol-Typ (ANSI-Typ)	Bemerkung
BOOLEAN	
CHAR(n)	$1 \leq n \leq 2.000$
DATE	
DECIMAL(p,s)	$s \leq p \leq 36$ $p \geq 1; s \geq 0$
DOUBLE PRECISION	
GEOMETRY[(srid)]	srid gibt das Koordinatensystem an (siehe auch EXA_SPATIAL_REF_SYS)
INTERVAL DAY [(p)] TO SECOND [(fp)]	$1 \leq p \leq 9, 0 \leq fp \leq 9$, Präzision auf Millisekunden genau
INTERVAL YEAR [(p)] TO MONTH	$1 \leq p \leq 9$
TIMESTAMP	Zeitstempel mit Präzision auf Millisekunden genau
TIMESTAMP WITH LOCAL TIME ZONE	Zeitstempel, der die Session-Zeitzone berücksichtigt
VARCHAR(n)	$1 \leq n \leq 2.000.000$

2.3.2. Details zu den Datentypen

Numerische Datentypen

In Exasol werden approximative und exakte numerische Typen unterstützt. Der Unterschied ergibt sich bei Verwendung der approximativen Datentypen aus den möglichen Rundungsdifferenzen, die durch die Art der Speicherung und die Art der Durchführung von Berechnungen entstehen können. Bei Verwendung exakter numerischer Typen entstehen keine solchen Informationsverluste aufgrund der Art der Speicherung.

Exasol-Typ (ANSI-Typ)	Wertebereich (Min, Max)	Precision	Bemerkung
<i>Exaktnumerischer Typ</i>			
DECIMAL(p,s)	$\left[-\frac{10^p - 1}{10^s}, +\frac{10^p - 1}{10^s} \right]$	Precision, Scale: (p,s)	$s \leq p \leq 36$ $p \geq 1; s \geq 0$
<i>Approximativer Typ</i>			
DOUBLE PRECISION	$[-1.7 \cdot 10^{308}, +1.7 \cdot 10^{308}]$	~15	Ungefähr 15 Stellen können gespeichert werden, unabhängig vom Scale. Der Sonderwert NaN wird als NULL interpretiert, der Wert Infinity wird nicht unterstützt.

Boolescher Datentyp

Der [ANSI](#) SQL Standard Datentyp BOOLEAN wird in Exasol direkt unterstützt.

Exasol-Typ (ANSI-Typ)	Wertebereich	Bemerkung
BOOLEAN	TRUE, FALSE, NULL/UNKNOWN	Anstatt den booleschen Literalen (siehe Wertebereich) können auch folgende Zahlen/Zeichenketten interpretiert werden: TRUE: 1, '1', 'T', 't', 'TRUE' (case-insensitiv) FALSE: 0, '0', 'F', 'f', 'FALSE' (case-insensitiv)

Datum/Zeit Datentypen

Von den verschiedenen Datentypen des [ANSI](#) SQL-Standards werden in Exasol derzeit die Typen DATE und TIMESTAMP unterstützt. DATE entspricht einem Datum. TIMESTAMP enthält zusätzlich zum Datum die Uhrzeit. Zudem existiert der Datentyp TIMESTAMP WITH LOCAL TIME ZONE, der bei Berechnungen die in der Session eingestellte Zeitzone berücksichtigt (siehe [ALTER SESSION](#) und [SESSIONTIMEZONE](#)).

Exasol-Typ (ANSI-Typ)	Wertebereich	Bemerkung
DATE	01.01.0001 bis 31.12.9999	
TIMESTAMP	01.01.0001 00:00:00.000 bis 31.12.9999 23:59:59.999	Die Präzision ist auf Millisekunden limitiert!
TIMESTAMP WITH LOCAL TIME ZONE	01.01.0001 00:00:00.000 bis 31.12.9999 23:59:59.999	Gleicher Typ wie der normale TIMESTAMP Typ, allerdings berücksichtigt dieser Typ die in der Session eingestellte Zeitzone. Die Präzision ist auf Millisekunden limitiert!

Anmerkungen zum Typ TIMESTAMP WITH LOCAL TIME ZONE

- Bei Spalten vom Typ TIMESTAMP WITH LOCAL TIME ZONE werden die Zeitstempel intern nach UTC normalisiert und beim Ein- und Ausgeben in der Session-Zeitzone interpretiert. Damit können Nutzer aus verschiedenen Zeitzonen bequem Daten einfügen und auslesen, ohne sich um die interne Speicherung kümmern zu müssen. Allerdings sollten Sie beachten, dass die Ausführung des gleichen SQL Statements in Sessions mit unterschiedlicher Zeitzone auch unterschiedliche Ergebnisse liefern kann.
- Während TIMESTAMP eine simple Struktur aus Jahr, Monat, Tag, Stunde, Minute und Sekunde darstellt, repräsentieren Daten vom Typ TIMESTAMP WITH LOCAL TIME ZONE einen spezifischen Zeitpunkt auf der Zeitachse. Weil innerhalb der Zeitzonen auch Zeitsprünge (z.B. beim Wechsel von Winter- auf Sommerzeit) sowie zweideutige Zeiträume (z.B. beim Wechsel von Sommer- auf Winterzeit) existieren, werden intern die Daten stets nach UTC normalisiert. Werden in der lokalen Session-Zeitzone problematische Zeitstempel eingefügt, so wird entsprechend der Session-Option TIME_ZONE_BEHAVIOR vorgegangen (änderbar mittels [ALTER SESSION](#)).

Der String für die Option TIME_ZONE_BEHAVIOR besteht aus zwei Teilen:

Invalide Zeitstempel (INVALID)

Wird in einer Zeitzone die Zeit um eine Stunde nach vorne verschoben, so entsteht eine Lücke. Falls nun Zeitstempel in diese Lücke fallen, können diese unterschiedlich behandelt werden:

SHIFT Korrigiert den Wert, indem die Dauer der Zeitumstellung (typischerweise eine Stunde) aufaddiert wird

ADJUST Rundet den Wert auf den ersten gültigen Zeitstempel nach der Umstellung

NULLIFY Setzt diesen Wert auf NULL
REJECT Wirft eine Fehlermeldung

Mehrdeutige Zeitstempel (AMBIGUOUS)

Wird in einer Zeitzone die Zeit um eine Stunde noch zurück gesetzt, so entstehen mehrdeutige Zeitstempel, die unterschiedlich behandelt werden können:

ST Interpretiert den Wert aus der Standard Time (ST)
DST Interpretiert den Wert aus der Daylight Saving Time (DST)
NULLIFY Setzt diesen Wert auf NULL
REJECT Wirft eine Fehlermeldung

Daher sind folgende Kombinationen möglich:

'INVALID SHIFT AMBIGUOUS ST'
'INVALID SHIFT AMBIGUOUS DST'
'INVALID SHIFT AMBIGUOUS NULLIFY'
'INVALID SHIFT AMBIGUOUS REJECT'
'INVALID ADJUST AMBIGUOUS ST'
'INVALID ADJUST AMBIGUOUS DST'
'INVALID ADJUST AMBIGUOUS NULLIFY'
'INVALID ADJUST AMBIGUOUS REJECT'
'INVALID NULLIFY AMBIGUOUS ST'
'INVALID NULLIFY AMBIGUOUS DST'
'INVALID NULLIFY AMBIGUOUS NULLIFY'
'INVALID NULLIFY AMBIGUOUS REJECT'
'INVALID REJECT AMBIGUOUS ST'
'INVALID REJECT AMBIGUOUS DST'
'INVALID REJECT AMBIGUOUS NULLIFY'
'INVALID REJECT AMBIGUOUS REJECT'

- Bei der Konvertierung zwischen den beiden Typen TIMESTAMP und TIMESTAMP WITH LOCAL TIME ZONE wird stets die Session-Zeitzone ausgewertet, und der TIMESTAMP WITH LOCAL TIME ZONE in einen normalen TIMESTAMP umgewandelt. Dies ist das gleiche Verhalten, als wenn Sie sich z.B. in EXAplus einen TIMESTAMP WITH LOCAL TIME ZONE anzeigen lassen. Denn auch hier wird der Wert, intern ja in UTC gespeichert, unter Berücksichtigung der Session-Zeitzone in einen normalen TIMESTAMP konvertiert.
- Spezielle Literale gibt es für den Typ TIMESTAMP WITH LOCAL TIME ZONE nicht. Es werden normale TIMESTAMP Literale erwartet (siehe [Abschnitt 2.5, Literale](#)), und der entsprechende Zeitpunkt durch die Session-Zeitzone definiert. Wie genau bei der Arithmetik zwischen Datumswerten und bei Datetime-Funktionen vorgegangen wird, entnehmen Sie bitte den Details in den entsprechenden Kapiteln ([Abschnitt 2.9.1, Skalare Funktionen](#) und [Abschnitt 2.7, Operatoren](#)).
- Bitte beachten Sie, dass die protokollierten TIMESTAMP-Werte in den statistischen Systemtabellen in der Datenbank-Zeitzone ([DBTIMEZONE](#)) gesammelt werden, welche über EXAoperation gesetzt wird. Dies ist vor allem dann relevant, wenn Sie die leicht unterschiedlichen Funktionen für die aktuelle Zeit verwenden:

SYSTIMESTAMP	Liefert die aktuelle Zeit als TIMESTAMP, interpretiert in der Datenbank-Zeitzone (DBTIMEZONE)
CURRENT_TIMESTAMP	Liefert die aktuelle Zeit als TIMESTAMP, interpretiert in der Session-Zeitzone (SESSIONTIMEZONE)
LOCALTIMESTAMP	Synonym für CURRENT_TIMESTAMP
NOW	Synonym für CURRENT_TIMESTAMP
- Die Liste der unterstützten Zeitzonen finden Sie in der Systemtabelle [EXA_TIME_ZONES](#).

Intervall Datentypen

Mit Hilfe der zwei Intervall Datentypen können Zeitintervalle definiert werden, welche insbesondere zur Arithmetik mit Datum/Zeit Werten geeignet sind.

Exasol-Typ (ANSI-Typ)	Wertebereich	Bemerkung
INTERVAL YEAR [(p)] TO MONTH	'-999999999-11' bis '999999999-11' (z.B. '5-3')	$1 \leq p \leq 9$ (default: 2)
INTERVAL DAY [(p)] TO SECOND [(fp)]	'-99999999 23:59:59.999' bis '99999999 23:59:59.999' (z.B. '2 12:50:10.123')	$1 \leq p \leq 9$ (default: 2), $0 \leq fp \leq 9$ (default: 3) Die Präzision ist auf Millisekunden limitiert!

Geodaten Datentyp

Der Geodaten Datentyp kann eine Vielzahl von Geometrie-Objekten abspeichern. Details finden Sie auch in [Abschnitt 2.4, Geodaten](#).

Exasol-Typ (ANSI-Typ)	Mögliche Elemente	Bemerkung
GEOMETRY[(srid)]	POINT, LINESTRING, POLYGON, MULTI-POINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION	srid gibt das Koordinatensystem an (siehe auch EXA_SPATIAL_REF_SYS)

Zeichenketten-Datentypen

In Exasol werden zwei ANSI SQL Standard Typen für Schriftzeichen unterstützt: CHAR und VARCHAR.

CHAR(n) ist als Datentyp mit fester, vordefinierter Länge n implementiert. Bei der Speicherung von kürzeren Zeichenketten werden diese mit Leerzeichen aufgefüllt ("Padding").

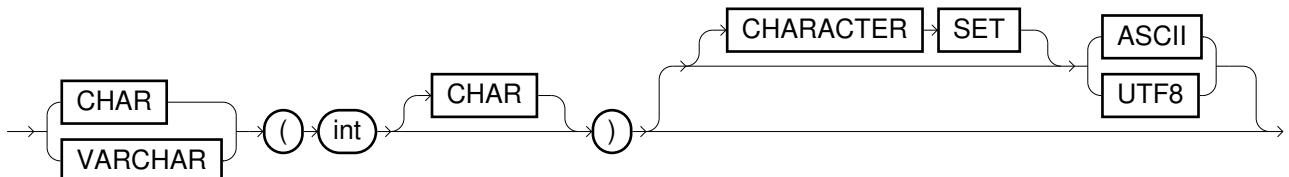
VARCHAR(n) kann beliebige Zeichenketten der Länge n oder kleiner enthalten. Diese Zeichenketten werden in ihrer jeweiligen Länge abgelegt.

Beide Typen sind in der Länge auf 2.000 Zeichen (CHAR) bzw. 2.000.000 Zeichen (VARCHAR) begrenzt und verwenden je nach Einstellung entweder den ASCII- oder UTF8-Zeichensatz (Unicode). Gibt man nicht explizit ein Character Set an, so wird UTF8 verwendet.



Eine leere Zeichenkette wird als NULL ausgewertet.

stringtype_definition ::=



Der Zeichensatz einer Zeichenketten-Spalte, lässt sich mit Hilfe des Befehls [DESC\[RIBE\]](#) in Erfahrung bringen.

Exasol-Typ (ANSI-Typ)	Zeichensatz	Bemerkung
CHAR(n)	ASCII, UTF8	$1 \leq n \leq 2.000$

Exasol-Typ (ANSI-Typ)	Zeichensatz	Bemerkung
VARCHAR(n)	ASCII, UTF8	$1 \leq n \leq 2.000.000$

2.3.3. Datentyp-Aliase

Aus Kompatibilitätsgründen zum SQL Standard und anderen Datenbanken sind in Exasol weitere Datentypen zu den bereits vorgestellten Datentypen als Alias definiert.

Tabelle 2.3. Zusammenfassung der Exasol Aliase

Alias	Exasol-Typ Äquivalent	Bemerkung
BIGINT	DECIMAL(36,0)	
BOOL	BOOLEAN	
CHAR	CHAR(1)	
CHAR VARYING(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
CHARACTER	CHAR(1)	
CHARACTER LARGE OBJECT	VARCHAR(1)	
CHARACTER LARGE OBJECT(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
CHARACTER VARYING(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
CLOB	VARCHAR(2000000)	
CLOB(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
DEC	DECIMAL(18,0)	
DEC(p)	DECIMAL(p,0)	1 ≤ p ≤ 36
DEC(p,s)	DECIMAL(p,s)	s ≤ p ≤ 36 p ≥ 1; s ≥ 0
DECIMAL	DECIMAL(18,0)	
DECIMAL(p)	DECIMAL(p,0)	1 ≤ p ≤ 36
DOUBLE	DOUBLE PRECISION	
FLOAT	DOUBLE PRECISION	
INT	DECIMAL(18,0)	
INTEGER	DECIMAL(18,0)	
LONG VARCHAR	VARCHAR(2000000)	
NCHAR(n)	CHAR(n)	
NUMBER	DOUBLE PRECISION	Möglicher Präzisionsverlust
NUMBER(p)	DECIMAL(p,0)	1 ≤ p ≤ 36
NUMBER(p,s)	DECIMAL(p,s)	s ≤ p ≤ 36 p ≥ 1; s ≥ 0
NUMERIC	DECIMAL(18,0)	
NUMERIC(p)	DECIMAL(p,0)	1 ≤ p ≤ 36
NUMERIC(p,s)	DECIMAL(p,s)	s ≤ p ≤ 36 p ≥ 1; s ≥ 0
NVARCHAR(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
NVARCHAR2(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000
REAL	DOUBLE PRECISION	
SHORTINT	DECIMAL(9,0)	
SMALLINT	DECIMAL(9,0)	
TINYINT	DECIMAL(3,0)	
VARCHAR2(n)	VARCHAR(n)	1 ≤ n ≤ 2.000.000

2.3.4. Typkonvertierungsregeln

An vielen Stellen in SQL-Anfragen werden bestimmte Datentypen erwartet, wie z.B. der Datentyp CHAR oder VARCHAR in der Zeichenketten-Funktion [SUBSTR\[ING\]](#). Wenn ein Benutzer einen anderen Typ einsetzen möchte, sollte er sicherheitshalber mit expliziten Konvertierungsfunktionen arbeiten (siehe auch die Liste der [Konvertierungsfunktionen](#)).

Falls keine expliziten Konvertierungsfunktionen verwendet werden, versucht das System eine implizite Konvertierung durchzuführen. Ist dies nicht möglich oder kann bei der Berechnung ein einzelner Wert nicht erfolgreich umgewandelt werden, so wird vom System eine Fehlermeldung ausgegeben.

Die nachfolgende Tabelle gibt einen Überblick über die zulässigen impliziten Konvertierungen.

Tabelle 2.4. Mögliche implizite Konvertierungen

Quell- \ Zielta- tentyp	DECIMAL	DOUBLE	BOOL- LEAN	DATE / TIME- STAMP	INTER- VAL YEAR [(p)] TO MONTH	INTER- VAL DAY [(p)] TO SECOND [(fp)]	GEOME- TRY	CHAR / VAR- CHAR
DECIMAL	✓	✓	✓	–	–	–	–	✓
DOUBLE	✓	✓	✓	–	–	–	–	✓
BOOLEAN	✓	✓	✓	–	–	–	–	✓
DATE / TIME- STAMP	–	–	–	✓	–	–	–	✓
INTERVAL YEAR [(p)] TO MONTH	–	–	–	–	✓	–	–	✓
INTERVAL DAY [(p)] TO SECOND [(fp)]	–	–	–	–	–	✓	–	✓
GEOMETRY	–	–	–	–	–	–	✓	✓
CHAR / VAR- CHAR	✓	✓	✓	✓	✓	✓	✓	✓

Symbolbedeutung:

- ✓ Implizite Konvertierung funktioniert immer
- ✗ Implizite Konvertierung grundsätzlich möglich, aber es können Fehler auftreten, falls einzelne Werte nicht konvertierbar sind
- Keine implizite Konvertierung möglich

Anmerkungen:

- Bei der Konvertierung von DECIMAL nach DOUBLE kann es zu Rundungsungenauigkeiten kommen.
- Bei der Konvertierung von DECIMAL oder DOUBLE nach BOOLEAN wird 1 nach TRUE, 0 nach FALSE und NULL nach NULL umgeformt. Dabei ist zu beachten das auch andere datentypgerechte Darstellungen der Werte 1 und 0 verwendet werden dürfen. Zum Beispiel ist $1 = 1.00$ (DECIMAL(3,2)) = 1.0 (DECIMAL(2,1)).
- Bei der Umwandlung von BOOLEAN nach DECIMAL oder DOUBLE wird TRUE nach 1, FALSE nach 0 und NULL nach NULL umgeformt.
- Bei der Umwandlung von BOOLEAN nach CHAR(n) bzw. VARCHAR(n) wird TRUE nach 'True' und FALSE nach 'False' umgewandelt.
- Die Umwandlung von GEOMETRY Daten zwischen zwei verschiedenen Koordinatensystemen wird nicht unterstützt.
- Konvertierungen nach CHAR(n) bzw. VARCHAR(n) funktionieren immer dann, wenn n groß genug ist und alle Zeichen im Ziel-Zeichensatz enthalten sind.

- Konvertierungen von CHAR(n) oder VARCHAR(n) in einen anderen Datentyp gelingen immer dann, wenn die zu konvertierenden Daten konform zu dem Zieldatentyp sind.
- Bei der Umwandlung von CHAR(n) bzw. VARCHAR(n) nach DATE oder TIMESTAMP ist das aktuelle Formatmodell zu berücksichtigen (siehe auch [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#)).
- Bei der Umwandlung von CHAR(n) bzw. VARCHAR(n) nach BOOLEAN können die Zeichenketten '0', 'F', 'f' bzw. 'FALSE' (case-insensitiv) für den Wert FALSE sowie '1', 'T', 't', bzw. 'TRUE' (case-insensitiv) für den Wert TRUE benutzt werden.
- In Operationen mit mehreren numerischen Operanden (z.B. bei den Operatoren +,-,/,*) werden diese implizit in den größten vorkommenden Datentyp konvertiert (z.B. ist DOUBLE größer als DECIMAL), bevor die eigentliche Operation durchgeführt wird. In diesem Zusammenhang spricht man auch von numerischer Präzision.

Im folgenden Beispiel wird eine implizite Konvertierung ausgeführt, um die BOOLEAN-Eingabe in die DECIMAL-Spalte der angelegten Tabelle einzufügen zu können.

Beispiel(e)

```
CREATE TABLE t(d DECIMAL);

-- Implizite Konvertierung von BOOLEAN nach DECIMAL
INSERT INTO t VALUES TRUE, NULL, FALSE;

SELECT * FROM t;

D
-----
 1
 0
```

2.3.5. Default-Werte

Einleitung

Default-Werte bezeichnen vordefinierte Werte, die in Tabellenspalten anstelle der NULL immer dann eingefügt werden, wenn bei einer Einfügeoperation keine expliziten Werte angegeben wurden (z.B. [INSERT](#)).

Beispiel

```
CREATE TABLE telephonelist(
    name          VARCHAR(10),
    phone_number VARCHAR(10),
    type          VARCHAR(10) DEFAULT 'home',
    alterationtime TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

INSERT INTO telephonelist (name, phone_number) VALUES ('Meyer', '1234');
INSERT INTO telephonelist (name, phone_number) VALUES ('Müller', '5678');
INSERT INTO telephonelist (name, type, phone_number)
VALUES ('Meyer', 'work', '9999');
UPDATE telephonelist SET name='Meier', alterationtime=DEFAULT
WHERE phone_number='1234';

SELECT * FROM telephonelist;
```

NAME	PHONE_NUMBER	TYPE	ALTERATIONTIME

Meier	1234	home	2010-12-13 16:39:02.393000
Müller	5678	home	2010-12-13 16:39:00.823000
Meyer	9999	work	2010-12-13 16:39:01.62600

Statements, die Default-Werte verwenden und manipulieren

Default-Werte können mit den folgenden Statements erstellt, verändert und gelöscht werden:

- [CREATE TABLE](#) (in der Spaltendefinition)
- [ALTER TABLE \(column\)](#)
 - [ADD COLUMN](#) (in der Spaltendefinition)
 - [MODIFY COLUMN](#) (in der Spaltendefinition)
 - [ALTER COLUMN DEFAULT](#) mit `SET DEFAULT` oder `DROP DEFAULT`

Default-Werte werden in Tabellen mit folgenden Statements explizit oder implizit verwendet:

- [INSERT](#): DEFAULT als 'Wert' für eine Spalte (`INSERT INTO t(i,j,k) VALUES (1, DEFAULT, 5)`) oder DEFAULT VALUES für alle Spalten (`INSERT INTO t DEFAULT VALUES`) bzw. implizit bei Spaltenauswahl (bei den Spalten, die nicht gewählt wurden)
- [IMPORT](#): Implizit bei einer Spaltenauswahl, in der Spalten mit Default-Value nicht enthalten sind
- [UPDATE](#): DEFAULT als 'Wert' für eine Spalte (`SET i=DEFAULT`)
- [MERGE](#): DEFAULT als 'Wert' für eine Spalte in den INSERT- und UPDATE-Teilen bzw. implizit im INSERT-Teil bei Spaltenauswahl (bei den Spalten, die nicht gewählt wurden)
- [ADD COLUMN](#): wenn für die eingefügte Spalte mit DEFAULT ein Default-Wert definiert wurde

Erlaubte Default-Werte

Folgende Ausdrücke sind als Default-Werte erlaubt:

- Konstanten
- Zur Auswertungszeit konstante Werte wie z.B. [CURRENT_USER](#) oder [CURRENT_DATE](#)
- Wertausdrücke, die nur Funktionen der obigen beiden Ausdrücke enthalten

 Default-Werte sind auf 2000 Zeichen begrenzt.

Beispiele

```
CREATE TABLE t (i DECIMAL, vc VARCHAR(100));
ALTER TABLE t ALTER COLUMN i SET DEFAULT 1;
ALTER TABLE t ALTER COLUMN i SET DEFAULT 1+2;
ALTER TABLE t ALTER COLUMN vc SET DEFAULT 'abc' || TO_CHAR(CURRENT_DATE);
ALTER TABLE t ALTER COLUMN vc SET DEFAULT CURRENT_USER;
```

Anzeige von Default-Werten

Die definierten Default-Werte können in den Tabellen [EXA_ALL_COLUMNS](#), [EXA_DBAL_COLUMNS](#) und [EXA_USER_COLUMNS](#) über die Spalte COLUMN_DEFAULT angezeigt werden.

Beispiel:

```
CREATE TABLE t (i DECIMAL DEFAULT 3+4);

SELECT column_default FROM exa_all_columns
    WHERE column_table='t'
```

```

        AND      column_name= 'i' ;

COLUMN_DEFAULT
-----
3+4

```

Mögliche Fehlerquellen

In fast allen Fällen sollten Default-Werte funktionieren, wie es erwartet wird. Es gibt jedoch einige wenige Dinge zu beachten, um einen fehlerfreien Einsatz sicherzustellen:

- Bei **MODIFY COLUMN** wird ein vorhandener Default-Wert bei Änderung des Datentyps übernommen. Passt der alte Default-Wert nicht zum neuen Datentyp, so gibt es eine Fehlermeldung.
- Werden als Default-Werte Einträge verwendet, deren Länge unterschiedlich sein kann und möglicherweise nicht in die Tabellenspalte passt (beispielsweise **CURRENT_USER** oder **CURRENT_SCHEMA**), dann wird zur Einfügezeit eine Fehlermeldung ausgegeben, falls der Wert bei Setzen des Default-Wertes noch passte, nun aber nicht mehr. Die Einfügeoperation wird dann nicht durchgeführt. Soll der Default-Wert im Zweifelsfall auf die Länge der Spalte gekürzt werden, bietet sich die Verwendung von **SUBSTR[ING]** auf den Default-Wert an.
- Die Interpretation von Default-Werten kann von Format-Modellen abhängen (z.B. ein DATE-Wert). In diesem Fall ist es ratsam, das Format explizit zu erzwingen (z.B. mittels **TO_DATE** mit Formatangabe).

Beispiel

```

CREATE TABLE t1 (d DATE DEFAULT '2006-12-31');
CREATE TABLE t2 (d DATE DEFAULT TO_DATE('2006-12-31','YYYY-MM-DD'));
ALTER SESSION SET NLS_DATE_FORMAT='DD.MM.YYYY';

-- beim INSERT in t1 tritt eine Fehlermeldung auf, bei t2 nicht
INSERT INTO t1 DEFAULT VALUES;
INSERT INTO t2 DEFAULT VALUES;

```

2.3.6. Identity-Spalten

Einleitung

Mit Hilfe von Identity-Spalten können IDs erzeugt werden. Sie sind ähnlich zu Default-Werten, werden allerdings dynamisch berechnet.

Beispiel

```

CREATE TABLE actors (id      INTEGER IDENTITY,
                     lastname VARCHAR(20),
                     surname VARCHAR(20));

INSERT INTO actors (lastname, surname) VALUES
  ('Pacino', 'Al'),
  ('Willis', 'Bruce'),
  ('Pitt', 'Brad');

SELECT * FROM actors;

```

ID	LASTNAME	SURNAME
1	Pacino	Al
2	Willis	Bruce
3	Pitt	Brad

Anmerkungen

- Die Inhalte einer Identity-Spalte verhalten sich folgendermaßen:
 - Wird beim Einfügen einer Zeile für die Identity-Spalte ein expliziter Wert angegeben, so wird dieser Wert auch eingefügt.
 - In allen anderen Fällen werden vom System monoton aufsteigende Zahlen generiert, die allerdings Lücken enthalten können.
 - Der aktuelle Wert des Zahlengenerators kann mittels [ALTER TABLE \(column\)](#) jederzeit geändert werden. Explizit eingefügte Werte beeinflussen den Zahlengenerator nicht.
 - Durch DML-Statements ([INSERT](#), [IMPORT](#), [UPDATE](#), [MERGE](#)) können die Werte einer Identity-Spalte jederzeit geändert werden.



Eine Identity-Spalte ist nicht zu verwechseln mit Constraints und daher keine Garantie auf Eindeutigkeit der Werte. Allerdings bleiben die Werte eindeutig, solange nur implizit generierte Werte eingefügt und die Daten anschließend nicht manuell verändert werden.

- Identity-Spalten sind nur für exaktnumerische Datentypen ohne Nachkommastellen erlaubt ([INTEGER](#), [DECIMAL\(x , 0 \)](#)). Der Wertebereich der generierten Zahlen wird durch den Datentyp begrenzt.
- Tabellen dürfen nur eine Identity-Spalte haben.
- Eine Spalte kann nicht gleichzeitig eine Identity-Spalte sein und einen Default-Werte besitzen.

Statements zum Setzen und Ändern von Identity-Spalten

Identity-Spalten können mit den folgenden Statements erstellt, verändert und gelöscht werden:

- [CREATE TABLE](#) (in der Spaltendefinition)
- [ALTER TABLE \(column\)](#)
 - [ADD COLUMN](#) (in der Spaltendefinition)
 - [MODIFY COLUMN](#) (in der Spaltendefinition)
 - [ALTER COLUMN IDENTITY](#) mit `SET IDENTITY` oder `DROP IDENTITY`

Die dynamisch erzeugten Werte einer Identity-Spalte werden bei folgenden Statements explizit oder implizit verwendet:

- [INSERT](#): DEFAULT als 'Wert' für eine Spalte (`INSERT INTO t(i,j,k) VALUES (1,DEFAULT,5)`) bzw. implizit bei einer Spaltenauswahl, in der die Identity-Spalte nicht enthalten ist
- [IMPORT](#): Implizit bei einer Spaltenauswahl, in der die Identity-Spalte nicht enthalten ist
- [UPDATE](#): DEFAULT als 'Wert' für eine Spalte (`SET i=DEFAULT`)
- [MERGE](#): DEFAULT als 'Wert' für eine Spalte in den INSERT- und UPDATE-Teilen bzw. implizit im INSERT-Teil bei Spaltenauswahl (bei den Spalten, die nicht gewählt wurden)
- [ADD COLUMN](#): wenn die eingefügte Spalte eine Identity-Spalte ist

Anzeige von Identity-Spalten

Die definierten Identity-Spalten können in den Tabellen [EXA_ALL_COLUMNS](#), [EXA_DBAL_COLUMNS](#) und [EXA_USER_COLUMNS](#) über die Spalte COLUMN_IDENTITY angezeigt werden, in der die zuletzt vergebene Zahl angegeben ist.

Beispiel:

```
CREATE TABLE t (id INTEGER IDENTITY(30));
SELECT column_identity FROM exa_all_columns
    WHERE column_table='T'
        AND column_name='ID';

COLUMN_IDENTITY
-----
30
```

2.4. Geodaten

Mit Hilfe von Geodaten können raumbezogene Informationen gespeichert und analysiert werden. Über Koordinaten werden Punkte, Linien bzw. Flächen (Polygone) definiert und in Exasol als GEOMETRY-Objekte abgespeichert. GEOMETRY-Spalten können optional eine SRID (ein Koordinaten-Referenzsystem, siehe auch [EXA_SPATIAL_REF_SYS](#)) haben, welche eine Art Constraint darstellt.

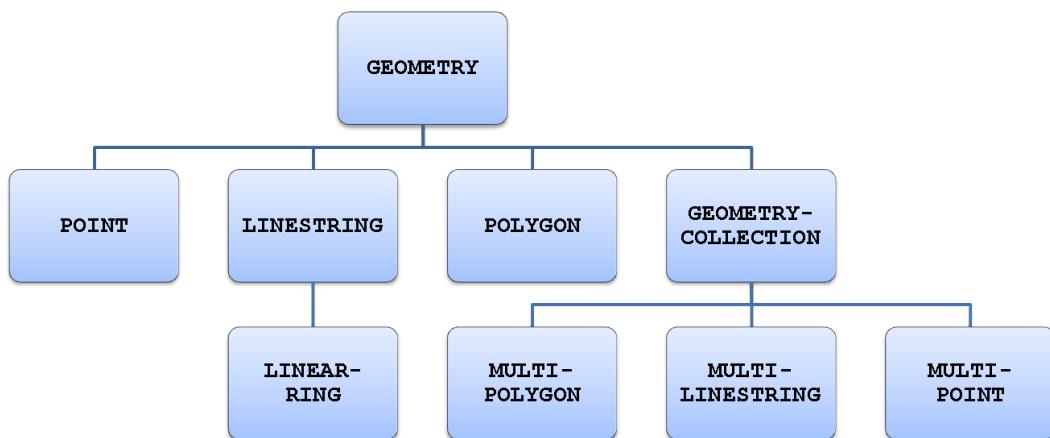
 Bitte beachten Sie, dass Geodaten Teil der Advanced Edition von Exasol ist.

 GEOMETRY Spalten werden durch Zeichenketten befüllt (wie z.B. 'POINT(2 5)'). Zum externen Auslesen durch die Treiber werden diese Daten automatisch zu Zeichenketten konvertiert. Das gleich gilt für die Befehle [IMPORT](#) und [EXPORT](#).

Für Geodaten-Objekte wird eine Vielzahl von Funktionen bereitgestellt, um diverse Berechnungen bzw. Operationen durchführen zu können.

2.4.1. Geo-Objekte

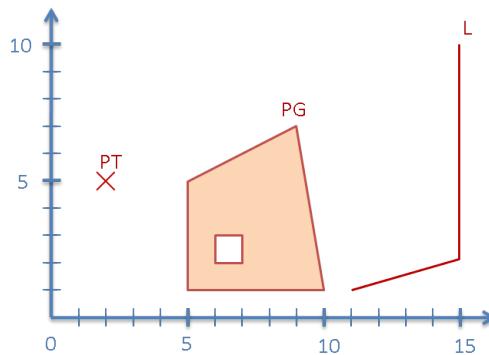
In der nachfolgenden Tabelle finden Sie die unterschiedlichen Typen von Geodaten-Objekten in Exasol.



Konstruktor	Beschreibung
Geometry	Allgemeinster Überbegriff eines beliebigen Geodaten-Objekts
POINT(x y)	Punkt im zweidimensionalen Raum
LINESTRING(x y, x y, ...)	Verbindungslinie zwischen einer Anzahl von zweidimensionalen Punkten
LINEARRING(x y, x y, ...)	Ein linearer Ring ist eine Verbindungslinie, bei der Anfangs- und Endpunkt übereinstimmen.
POLYGON((x y, ...), [(x y, ...), ...])	Fläche, die durch einen geschlossenen Ring definiert wird sowie einer optionalen Liste von Löchern innerhalb dieser Fläche
GEOMETRYCOLLECTION(geometry, ...)	Eine Menge von beliebigen Geodaten-Objekten
MULTIPOINT(x y, ...)	Menge von Punkten
MULTILINESTRING((x y, ...), ...)	Menge von Verbindungslinien
MULTIPOLYGON((x y, ...), ...)	Menge von Polygonen

 Anstatt der numerischen Argumente kann mit dem Keyword `EMPTY` die leere Menge eines Objekts erzeugt werden (z.B. `POLYGON EMPTY`)

Beispiele



```

POINT(2 5)                                -- PT
LINESTRING(10 1, 15 2, 15 10)             -- L
POLYGON((5 1, 5 5, 9 7, 10 1, 5 1),
        (6 2, 6 3, 7 3, 7 2, 6 2))       -- PG

MULTIPOINT(0.1 1.4, 2.2 3, 1 6.4)
MULTILINESTRING((0 1, 2 3, 1 6), (4 4, 5 5))
MULTIPOLYGON(((0 0, 0 2, 2 2, 3 1, 0 0)),
              ((4 6, 8 9, 12 5, 4 6), (8 6, 9 6, 9 7, 8 7, 8 6)))

GEOMETRYCOLLECTION(POINT(2 5), LINESTRING(1 1, 15 2, 15 10))

```

2.4.2. Geo-Funktionen

Exasol stellt eine Vielzahl von Funktionen zur Verfügung, um Operationen und mathematische Berechnungen auf Geodaten-Objekten durchzuführen. Die Argumente der Funktionen bilden Geodatenobjekte wie oben beschrieben ($p=POINT$, $ls=LINESTRING$, $mls=MULTILINESTRING$, $pg=POLYGON$, $mp=MULTIPOLYGON$, $g=GEOMETRY$, $gc=GEOMETRYCOLLECTION$, $n=Integer$).

Funktion	Beschreibung
Point Funktionen	
ST_X(p)	x-Koordinate des Punktes.
ST_Y(p)	y-Koordinate des Punktes.
(Multi-)LineString Funktionen	
ST_ENDPOINT(ls)	Endpunkt des LineString-Objekts.
ST_ISCLOSED(mls)	Gibt an, ob alle enthaltenen LineString-Objekte Ringe sind, also deren Anfangs- und Endpunkte übereinstimmen.
ST_ISRING(ls)	Gibt an, ob das LineString-Objekt ein geschlossener Ring ist, also Anfangs- und Endpunkt übereinstimmen.
ST_LENGTH(mls)	Länge eines LineString-Objekts bzw. die Summe der Längen der Einzelobjekte in einem MultiLineString-Objekt.
ST_NUMPOINTS(ls)	Anzahl an Punkten im LineString-Objekt.
ST_POINTN(ls , n)	Der n -te Punkt des LineString-Objekts, beginnend mit 1. Liefert NULL, falls ST_NUMPOINTS(ls)< n .
ST_STARTPOINT(ls)	Startpunkt des LineString-Objekts.
(Multi-)Polygon Funktionen	
ST_AREA(mp)	Fläche eines Polygons bzw. Summe der Flächen für ein MultiPolygon.

Funktion	Beschreibung	
	MULTIPOINT	Simpel, falls keine Punkte identisch sind.
	LINESTRING	Simpel, falls es keinen Punkt gibt, der von der Linie zweimal berührt wird (außer Anfangs- und Endpunkt).
	MULTILINESTRING	Simpel, falls alle Linien selbst simpel sind und sich die Objekte höchstens in den Endpunkten schneiden.
	POLYGON	Immer simpel.
	MULTIPOLYGON	Immer simpel.
ST_OVERLAPS(<i>g,g</i>)	Gibt an, ob sich zwei Geodaten-Objekte <i>überlappen</i> . Dies ist dann der Fall, wenn die Objekte nicht identisch sind und ihre Schnittmenge nicht leer ist sowie dieselbe Dimension wie die Objekte aufweist.	
ST_SETSRID(<i>g,srid</i>)	Setzt für ein Geodatenobjekt die SRID, also das Koordinatensystem (siehe auch EXA_SPATIAL_REF_SYS).	
ST_SYMDIFFERENCE(<i>g,g</i>)	Symmetrische Differenzmenge zweier Geodaten-Objekte.	
ST_TOUCHES(<i>g,g</i>)	Gibt an, ob sich zwei Geodaten-Objekte <i>berühren</i> . Dies ist dann der Fall, wenn die Schnittmenge nicht leer ist und sich nur auf Rändern (siehe ST_BOUNDARY) der Objekte befindet.	
ST_TRANSFORM(<i>g,srid</i>)	Konvertiert ein Geodatenobjekt in ein bestimmtes Koordinaten-Referenzsystem (siehe auch EXA_SPATIAL_REF_SYS).	
ST_UNION(<i>g,g</i>)	Vereinigungsmenge zweier Geodaten-Objekte. Diese Funktion kann auch als Aggregatsfunktion benutzt werden.	
ST_WITHIN(<i>g,g</i>)	Gibt an, ob sich das erste Objekt vollständig im zweiten befindet (Gegenteil von ST_CONTAINS).	

Beispiele

```

SELECT ST_Y('POINT (1 2)');
--> 2

SELECT ST_ENDPOINT('LINESTRING (0 0, 0 1, 1 1)');
--> POINT (1 1)

SELECT ST_ISRING('LINESTRING (0 0, 0 1, 1 1, 0 0)');
--> TRUE

SELECT ST_LENGTH('LINESTRING (0 0, 0 1, 1 1)');
--> 2

SELECT ST_BOUNDARY('LINESTRING (0 0, 1 1, 2 2)');
--> MULTIPOINT (0 0, 2 2)

SELECT ST_AREA('POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0),
                      (1 1, 1 2, 2 2, 2 1, 1 1))');
--> 15 (=16-1)

SELECT ST_DISTANCE('POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0))',

```

```
'POINT(12 10)' );  
--> 10
```

2.5. Literale

Literale stellen Konstanten dar, die einen bestimmten Wert sowie einen zugehörigen Datentyp besitzen. Literale können als Werte in Tabellen eingefügt werden oder innerhalb von Queries als Konstanten z.B. für Vergleiche oder Funktionsparameter verwendet werden. Falls das Literal einen anderen Typ aufweist als der Zuweisungs- bzw. Vergleichstyp, so wird der "kleinere" Datentyp implizit konvertiert.

Grundsätzlich wird unterschieden zwischen numerischen Literalen, booleschen Literalen, Datum/Zeit Literalen und Zeichenketten-Literalen. Außerdem existiert ein Sonderliteral für den NULL-Wert.

Beispiele

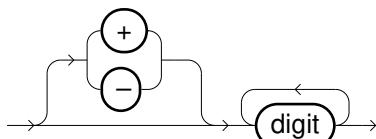
123	Integer Zahl (ganzzahlige Dezimalzahl)
-123.456	Dezimalzahl
1.2345E-15	Double Wert
TRUE	Boolescher Wert
DATE '2007-03-31'	Datum
TIMESTAMP '2007-03-31 12:59:30.123'	Zeitstempel
INTERVAL '13-03' YEAR TO MONTH	Intervall (YEAR TO MONTH)
INTERVAL '1 12:00:30.123' DAY TO SECOND	Intervall (DAY TO SECOND)
'ABC'	Zeichenkette
NULL	NULL-Wert

2.5.1. Numerische Literale

Numerische Literale repräsentieren eine Zahl. Es wird unterschieden zwischen exaktnumerischen Werten und Fließkommazahlen.

Folgende numerischen Literale werden unterschieden:

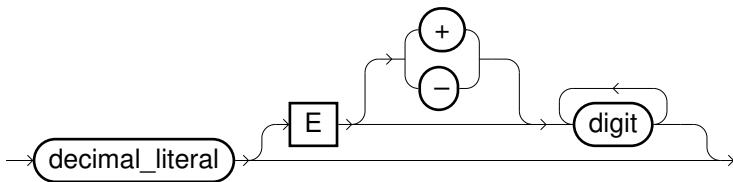
integer_literal ::=



decimal_literal ::=



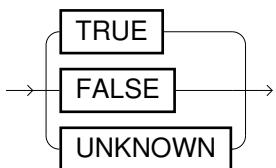
double_literal ::=



2.5.2. Boolesche Literale

Für boolesche Literale gibt es nur drei gültige Werte: TRUE, FALSE und UNKNOWN (=NULL). Anstatt des Literals können aber auch Zeichenketten (z.B. 'T' oder 'True') oder Zahlen (0 bzw. 1) verwendet werden. Details hierzu siehe [Boolescher Datentyp](#).

boolean_literal ::=



2.5.3. Datum/Zeit Literale

Datum/Zeit Literale stellen einen gewissen Zeitpunkt dar. Sie besitzen ein festes Format und sind daher weniger flexibel als die Konvertierungsfunktionen [TO_DATE](#) bzw [TO_TIMESTAMP](#). In den meisten Fällen sind diese Funktionen daher den Literalen zu bevorzugen (nähere Informationen finden Sie in [Abschnitt 2.9, Built-in-Funktionen](#) sowie [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#)).

date_literal ::=



timestamp_literal ::=



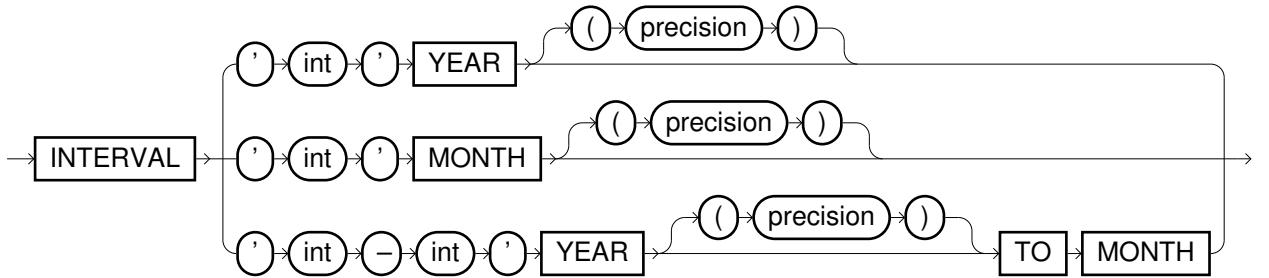
Folgende Formate sind für `string` vorgegeben:

Typ	Format	Erläuterung
DATE	'YYYY-MM-DD'	Festes Format für DATE-Literale ('Jahr-Monat-Tag')
TIMESTAMP	'YYYY-MM-DD HH:MI:SS.FF3'	Festes Format für TIMESTAMP-Literale ('Jahr-Monat-Tag Stunden-Minuten-Sekunden.Millisekunden')

2.5.4. Intervall Literale

Intervall-Literale beschreiben Zeitintervalle, welche besonders zur Arithmetik mit Datum/Zeit Werten und anderen Intervallen geeignet sind.

interval_year_to_month_literal ::=

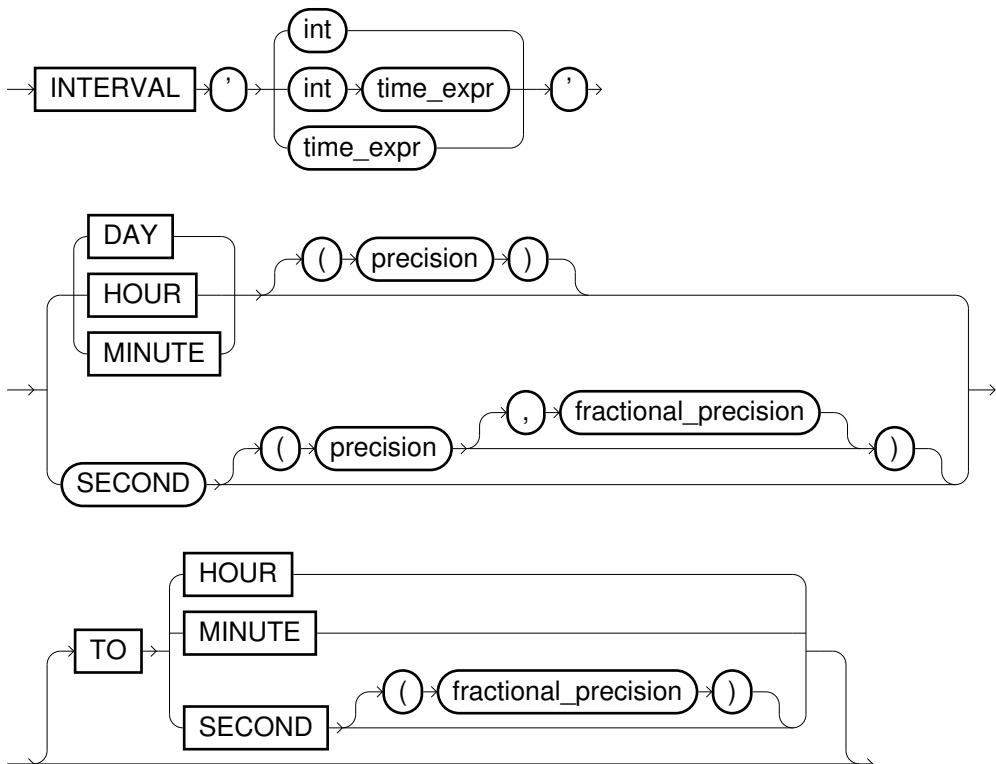


Anmerkungen:

- **int [-int]** definiert die Zahlenwerte für das Intervall. Für **YEAR TO MONTH** muss immer **int-int** angegeben werden.
- Der optionale Parameter **precision** (1-9) definiert die maximale Ziffernzahl. Ohne Angabe dieser Option sind maximal 2 Ziffern erlaubt, bei Monaten Werte von 0 bis 11.
- Beispiele:

INTERVAL '5' MONTH	Fünf Monate
INTERVAL '130' MONTH (3)	130 Monate
INTERVAL '27' YEAR	27 Jahre
INTERVAL '2-1' YEAR TO MONTH	Zwei Jahre und ein Monat
INTERVAL '100-1' YEAR(3) TO MONTH	100 Jahre und ein Monat

interval_day_to_second_literal ::=



Anmerkungen:

- int definiert die Anzahl an Tagen (maximale Ziffernzahl siehe precision).
- time_expr definiert eine Zeitangabe im Format HH[:MI[:SS[.n]]] bzw. MI[:SS[.n]] bzw. SS[.n]. Gültige Werte für Stunden (HH) sind 0-23, für Minuten (MI) 0-59 und für Sekunden (SS) 0-59.999. Für die vorderste Zahl definiert der Parameter precision (siehe unten) die maximale Ziffernzahl, wodurch auch größere Zahlen für Stunden, Minuten und Sekunden möglich sein können (siehe Beispiele). Der Parameter fractional_precision (0-9, Default ist 3) gibt an, ab welcher Stelle gerundet wird. In 'time_expr' muss stets der gesamte Bereich angegeben werden, bei HOUR TO MINUTE also z.B. 'HH:MI'.
- Bitte beachten Sie, dass die Genauigkeit der Sekunden auf drei Nachkommastellen begrenzt ist (wie auch bei Timestamp Werten), auch wenn Sie im Literal mehr Nachkommastellen angeben können.
- Der optionale Parameter precision (1-9) definiert die maximale Ziffernzahl der vorderen Zahl. Ohne Angabe dieser Option sind maximal 2 Ziffern erlaubt.
- Die Intervallgrenzen müssen absteigend sein. Daher ist z.B. SECOND TO MINUTE nicht erlaubt.
- Beispiele:

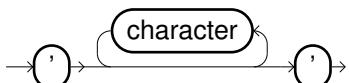
INTERVAL '5' DAY	Fünf Tage
INTERVAL '100' HOUR(3)	100 Stunden
INTERVAL '6' MINUTE	Sechs Minuten
INTERVAL '1.99999' SECOND(2,2)	2.00 Sekunden (es wird nach der zweiten Stelle gerundet)
INTERVAL '10:20' HOUR TO MINUTE	Zehn Stunden und 20 Minuten
INTERVAL '2 23:10:59' DAY TO SECOND	Zwei Tage, 23 Stunden, 10 Minuten und 59 Sekunden
INTERVAL '23:10:59.123' HOUR(2) TO SECOND(3)	23 Stunden, 10 Minuten und 59.123 Sekunden

2.5.5. Zeichenketten-Literale

Zeichenketten-Literale werden verwendet, um Text darzustellen. Die maximale Länge einer Zeichenkette ist durch den Datentyp CHAR begrenzt (siehe [Zeichenketten-Datentypen](#)).

Ein Zeichenketten-Literal verwendet den kleinstmöglichen Zeichensatz. Falls es also nur aus ASCII-Zeichen besteht, dann den ASCII-Zeichensatz, ansonsten den UTF8-Zeichensatz (Unicode).

string_literal::=



Ein leeres String-Literal ('') wird als NULL ausgewertet.



Um innerhalb einer Zeichenkette das einfache Anführungszeichen verwenden zu können, werden zwei einfache Anführungszeichen hintereinander geschrieben. Beispielsweise repräsentiert das Literal 'AB' 'C' den Wert AB'C.

2.5.6. NULL Literal

Das NULL-Literal ist in SQL dafür vorgesehen, einen Wert als "nicht bekannt" zu markieren.

null_literal ::=

→ **NULL** →

2.6. Format-Modelle

Format-Modelle sind String-Literale, die eine Art Konvertierungsregel zwischen Zeichenketten und numerischen bzw. Datums-/Zeit-Werten darstellen. Somit ist es beispielsweise möglich, einer Numerischen-Spalte eine Zeichenkette zuzuweisen oder ein Datum als Zeichenkette in einem gewünschten Format ausgeben zu lassen.

2.6.1. Datum/Zeit Format-Modelle

Datum/Zeit Format-Modelle können in den Funktionen [TO_CHAR \(datetime\)](#)/ [TO_DATE](#)/ [TO_TIMESTAMP](#) sowie den ETL-Befehlen [IMPORT](#)/ [EXPORT](#) verwendet werden.

Falls kein Format angegeben ist, wird das aktuelle Default-Format verwendet. Dies ist in den Session-Parametern [NLS_DATE_FORMAT](#) bzw. [NLS_TIMESTAMP_FORMAT](#) definiert und kann für die aktuelle Session mittels [ALTER SESSION](#) bzw. für die gesamte Datenbank mittels [ALTER SYSTEM](#) geändert werden.

Das Default-Format ist ebenfalls wichtig bei der impliziten Konvertierung (siehe auch [Abschnitt 2.3.4, Typkonvertierungsregeln](#)). Wird beispielsweise ein String-Wert in eine Datums-Spalte eingefügt, so wird ein implizites [TO_DATE](#) ausgeführt.

Falls keine Sprache angegeben ist, wird die aktuelle Session-Sprache zur Datumsdarstellung verwendet. Diese ist in dem Session-Parameter [NLS_DATE_LANGUAGE](#) definiert und kann für die aktuelle Session mittels [ALTER SESSION](#) bzw. für die gesamte Datenbank mittels [ALTER SYSTEM](#) geändert werden.

Die aktuellen Werte für die NLS-Parameter sind in der Systemtabelle [EXA_PARAMETERS](#) zu finden.

Für abgekürzte und ausgeschriebene Monats- und Tagesformate lässt sich über die Groß- und Kleinschreibung der ersten beiden Buchstaben die entsprechende Ausgabedarstellung variieren (siehe hierzu die Beispiele weiter unten).

In der folgenden Tabelle sind alle möglichen Elemente aufgelistet, die in Datum/Zeit Format-Modellen auftreten können. Dabei ist zu beachten, dass jedes der Elemente nur einmal im Format-Modell vorkommen darf. Ausnahmen sind hierbei Trenn- oder Füllzeichen, sowie das Ausgabeformat bei [TO_CHAR \(datetime\)](#). Ferner ist zu beachten, dass die abgekürzten Wochentage bei englischer Spracheinstellung mit drei Buchstaben, bei deutscher allerdings nur mit zwei Buchstaben dargestellt werden.

Tabelle 2.5. Elemente von Datum/Zeit Format-Modellen

Element	Bedeutung
- ; . / \ _	Dienen als Trenn- oder Füllzeichen
CC	Jahrhundert (01-99)
IYYY,vYYY sowie IYY,IY,I,vYY,vY,v	Jahr (0001-9999, nach internationaler Norm ISO 8601) sowie die letzten 3,2 bzw. 1 Ziffern
YYYY sowie YYY,YY,Y	Jahr (0001-9999) sowie die letzten 3,2 bzw. 1 Ziffern
VYYY sowie VYY,VY,V	Jahr passend zum Element VW sowie die letzten 3,2 bzw. 1 Ziffern
Q	Quartal des Jahres (1-4)
MM	Monat (01-12)
MON	Monat abgekürzt (JAN-DEC (ENG) bzw. JAN-DEZ (DEU))
MONTH	Monat ausgeschrieben (JANUARY-DECEMBER (ENG) bzw. JANUAR-DEZEMBER (DEU))
IW, vW	Woche des Jahres (01-53, Mo-So, Woche 01 des Jahres enthält 4. Januar, vorherige Tage zählen zu altem Jahr, nach internationaler Norm ISO 8601)
uW	Woche des Jahres (00-53, Mo-So, Woche 01 des Jahres enthält 4. Januar, vorherige Tage zählen zur Woche 00)
VW	Woche des Jahres (01-53, So-Sa, Woche 01 des Jahres beginnt mit erstem Sonntag, vorherige Tage zählen zu altem Jahr)
UW	Woche des Jahres (00-53, So-Sa, Woche 01 des Jahres beginnt mit erstem Sonntag, vorherige Tage zählen zur Woche 00)
WW	Woche des Jahres (01-53, der erste Tag des Jahres gilt als Wochenanfang)
J	Julianisches Datum (Anzahl Tage seit dem 1. Januar 4713 v. Chr.)
D	Tag der Woche (1-7, beginnend mit demjenigen Tag, der durch den Parameter NLS_FIRST_DAY_OF_WEEK spezifiziert wird - siehe auch ALTER SYSTEM)
ID	ISO-Tag der Woche (1-7, beginnend bei Montag)
DD	Tag (01-31)
DDD	Tag des Jahres (001-365 oder 001-366)
DAY	Wochentag ausgeschrieben (MONDAY-SUNDAY (ENG) bzw. MONTAG-SONNTAG (DEU))
DY	Wochentag abgekürzt (MON-SUN (ENG) bzw. MO-SO (DEU))
HH12	Stunde (01-12)
HH24 (bzw. HH)	Stunde (00-23)
AM, A.M., am, a.m.	Angabe zur 12-Stunden-Zählung (AM= <i>ante meridium</i> =vor Mittags bzw. PM= <i>post meridium</i> =nach Mittags).
PM, P.M., pm, p.m.	Identisch zu AM
MI	Minute (00-59)
SS	Sekunde (00-59)
FF[1-9]	Sekundenbruchteile (Beispielsweise Millisekunden bei FF3)

Beispiele

```
-- Wird interpretiert als 1. Februar 2003
SELECT TO_DATE('01-02-2003','DD-MM-YYYY');

-- Wird interpretiert als 10. Februar 2003
```

```

SELECT TO_DATE('06-2003-MON', 'WW-YYYY-DY');

-- Wird interpretiert als 31. Dezember 2003, 12:59:33
SELECT TO_TIMESTAMP('12:59:33 365-2003', 'HH24:MI:SS DDD-YYYY');

-- Wird interpretiert als 24. Dezember 2009, 23:00:00
SELECT TO_TIMESTAMP('2009-12-24 11:00:00 PM',
    'YYYY-MM-DD HH12:MI:SS AM');

-- Wird interpretiert als 12. Mai 2003, 00:00:10.123
SELECT TO_TIMESTAMP('2000_MAY_12 10.123', 'YYYY_MONTH_DD SS.FF3');

SELECT TO_CHAR(DATE '2007-12-31', 'DAY-Q-DDD; IYYY\IW') TO_CHAR1,
       TO_CHAR(DATE '2000-01-01', 'DAY-Day-day-')           TO_CHAR2;

TO_CHAR1                      TO_CHAR2
-----
MONDAY      -4-365; 2008\01  SATURDAY -Saturday -saturday -

```

2.6.2. Numerische Format-Modelle

Numerische Format-Modelle können in den Funktionen [TO_CHAR \(number\)](#) und [TO_NUMBER](#) sowie den ETL-Befehlen [IMPORT](#) und [EXPORT](#) verwendet werden und definieren die Interpretation bzw. Darstellung von Zeichenketten bzw. Zahlen.

In der folgenden Tabelle sind alle möglichen Elemente aufgelistet, die in numerischen Format-Modellen auftreten können.

Tabelle 2.6. Elemente numerischer Format-Modelle

Element	Beispiele	Beschreibung
Ziffern		
9	9999,999	Jede 9 steht für eine Ziffer der Zahl. Falls die Zahl weniger Ziffern hat, so wird vorne mit Leerzeichen aufgefüllt.
0	0000,000	Jede 0 steht für eine Ziffer der Zahl. Falls die Zahl weniger Ziffern hat, so wird mit 0 aufgefüllt.
Vorzeichen		
S	S9999,999	Schreibt ein + bei positiven Zahlen und - bei negativen Zahlen. S darf nur am Anfang oder Ende des Formatstrings stehen.
MI	9999,999MI	Schreibt ein Leerzeichen bei positiven Zahlen und - bei negativen Zahlen. MI darf nur am Ende des Formatstrings stehen.
FM	FM9999,999	Schneidet alle überflüssigen Nullen ab. Bei positiven Zahlen werden alle führenden und nachfolgenden Leerzeichen entfernt. Bei negativen Zahlen wird ein - geschrieben. FM darf nur am Anfang des Formatstrings stehen.
Dezimaltrennzeichen		
.	999.999	Schreibt einen Dezimalpunkt vor die Nachkommastellen.
D	999D999	Schreibt das Dezimaltrennzeichen aus NLS_NUMERIC_CHARACTERS vor die Nachkommastellen.
Gruppentrennzeichen		
,	999,999.999	Schreibt ein Komma als Gruppentrennzeichen an die angegebene Stelle.
G	999G999D999	Schreibt das Gruppentrennzeichen aus NLS_NUMERIC_CHARACTERS an die angegebene Stelle.
Sonstiges		
eeee	9.99eeee	Ausgabe in wissenschaftlicher Notation.
EEEE	9.99EEEE	Ausgabe in wissenschaftlicher Notation.

Anmerkungen

- Falls der Formatstring zu kurz für die Zahl ist, so wird als Ergebnis ein String zurückgeliefert, der mit dem Zeichen # aufgefüllt ist.
- Die Dezimal- und Gruppentrennzeichen sind im Parameter [NLS_NUMERIC_CHARACTERS](#) definiert und können für die aktuelle Session mittels [ALTER SESSION](#) bzw. für die gesamte Datenbank mittels [ALTER SYSTEM](#) geändert werden.
- Im Default-Fall wird für das Dezimaltrennzeichen ein Punkt und für das Gruppentrennzeichen ein Komma verwendet. Die aktuellen Werte für die NLS-Parameter sind in der Systemtabelle [EXA_PARAMETERS](#) zu finden.

Nachfolgende Beispiele sollen die Verwendung numerischer Format-Modelle für die Funktionen [TO_CHAR](#) ([number](#)) verdeutlichen.

Zahl	Format	Ergebnis	Beschreibung
123	99	'###'	Die Zahl ist zu lang für das Format
123	999	' 123'	Standardformat einer positiven Ganzzahl
-123	999	'-123'	Standardformat einer negativen Ganzzahl
123.456	999.999	' 123.456'	Standardformat einer positiven Dezimalzahl
-123.456	999.999	'-123.456'	Standardformat einer negativen Dezimalzahl

Zahl	Format	Ergebnis	Beschreibung
123.456	0000.0000	'0123.4560'	Format mit aufgefüllten Nullen
123.456	S999.999	'+123.456'	Explizites Vorzeichen
123.456	999.999MI	'123.456 '	Vorzeichen am Ende
-123.456	999.999MI	'123.456-'	Vorzeichen am Ende
123.456	FM999.999	'123.456'	Entfernt führende Leerzeichen und Nullen
123456.789	999,999.999	'123,456.789'	Gruppentrennzeichen
123.4567	9.99eeee	'1.23e2'	Wissenschaftliche Notation
123.4567	9.99EEEE	'1.23E2'	Wissenschaftliche Notation

2.7. Operatoren

Operatoren verknüpfen ein bzw. zwei Werte (Operanden) miteinander, z.B. bei der Addition zweier Zahlen.

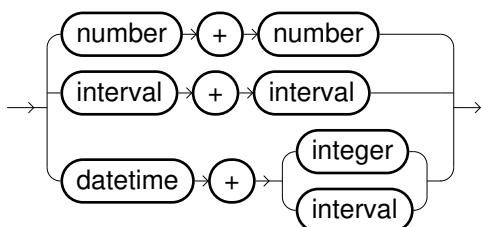
Die Präzedenz, d.h. die Reihenfolge, mit der Operatoren ausgewertet werden, lautet:

+,-, PRIOR, CONNECT_BY_ROOT	Als unäre Operatoren (z.B. bei negativer Zahl)
	Konkatenationsoperator
*,/	Multiplikation bzw. Division
+-	Binäre Operatoren Addition bzw. Subtraktion

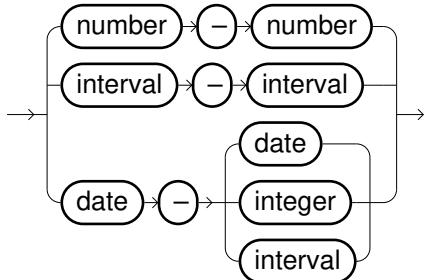
2.7.1. Arithmetische Operatoren

Syntax

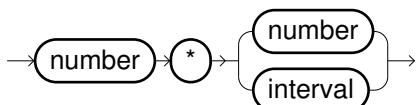
+ operator::=



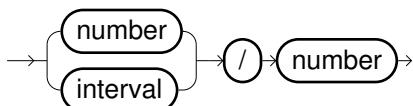
- operator::=



* operator::=



/ operator::=



Anmerkung(en)

- + Operator
 - Das Ergebnis beim Addieren einer Dezimalzahl zu einem Datumswert ist abhängig vom Session-/System-Parameter TIMESTAMP_ARITHMETIC_BEHAVIOR:

- `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'INTERVAL'` - die Dezimalzahl wird zu einer Ganzzahl gerundet und stets eine gewisse Zahl ganzer Tage addiert (wie die Funktion `ADD_DAYS`)
- `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'DOUBLE'` - die Nachkommastellen werden als Anteil eines Tages umgerechnet und ebenso addiert (Stunden, Minuten, ...)
- Falls eine Anzahl an Jahren oder Monaten (z.B. `INTERVAL '3' MONTH`) auf ein Datum addiert wird, dessen Tag der letzte Tag des Monats ist, so wird der letzte Tag des resultierenden Monats zurückgegeben (wie bei `ADD_MONTHS`).
- - Operator
- Die Differenz zweier Datums-Werte hängt vom Session-/System-Parameter `TIMESTAMP_ARITHMETIC_BEHAVIOR` ab:
 - `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'INTERVAL'` - das Ergebnis ist ein Interval
 - `TIMESTAMP_ARITHMETIC_BEHAVIOR = 'DOUBLE'` - das Ergebnis ist ein Double, der der Anzahl an Tagen zwischen den beiden Werten entspricht (ähnlich zur Funktion `DAYS_BETWEEN`)
- Die Differenz eines Datums-Wertes und einer Dezimalzahl entspricht dem Verhalten des + Operators, wobei Tage abgezogen anstatt addiert werden.
- Die Differenz zweier Intervall-Werte liefert wiederum ein Intervall
- Falls eine Anzahl an Jahren oder Monaten (z.B. `INTERVAL '3' MONTH`) von einem Datum subtrahiert wird, dessen Tag der letzte Tag des Monats ist, so wird der letzte Tag des resultierenden Monats zurückgegeben (wie bei `ADD_YEARS` bzw. `ADD_MONTHS`).

Beispiel(e)

```

SELECT 1000 + 234                                ADD1,
        DATE '2000-10-05' + 1                      ADD2,
        DATE '2009-01-05' + INTERVAL '1-2' YEAR TO MONTH ADD3,
        100000 - 1                                  SUB1,
        DATE '2000-10-05' - 1                      SUB2,
        DATE '2009-01-05' - INTERVAL '2' MONTH       SUB3,
        100 * 123                                    MUL,
        100000 / 10                                 DIV;

ADD1    ADD2      ADD3      SUB1     SUB2      SUB3      MUL      DIV
-----  -----
1234   2000-10-06 2010-03-05  99999  2000-10-04  2008-11-05  12300  10000.0

```

2.7.2. Konkatenationsoperator ||

Zweck

Liefert die Konkatenation aus `string1` und `string2`. Dieser Konkatenationsoperator ist äquivalent zur Funktion `CONCAT`.

Syntax

`|| operator::=`

→ (string1) → || → (string2) →

Beispiel(e)

```
SELECT 'abc' || 'DEF';
```

```
'abc' || 'DEF'  
-----  
abcDEF
```

2.7.3. CONNECT BY Operatoren

Zweck

Details zur Verwendung dieser beiden Operatoren finden Sie in der Dokumentation zum [SELECT Befehl](#) in [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Syntax

PRIOR operator::=

→ **PRIOR** → **expr** →

CONNECT_BY_ROOT operator::=

→ **CONNECT_BY_ROOT** → **expr** →

Beispiel(e)

```
SELECT last_name,  
       PRIOR last_name PARENT_NAME,  
       CONNECT_BY_ROOT last_name ROOT_NAME,  
       SYS_CONNECT_BY_PATH(last_name, '/') "PATH"  
  FROM employees  
 CONNECT BY PRIOR employee_id = manager_id  
 START WITH last_name = 'Clark';  
  
LAST_NAME PARENT_NAME ROOT_NAME PATH  
-----  
Clark          Clark      /Clark  
Smith         Clark      /Clark/Smith  
Brown         Smith     /Clark/Smith/Brown
```

2.8. Prädikate

2.8.1. Einleitung

Prädikate sind Ausdrücke, die als Ergebnis stets einen booleschen Wert zurückliefern, also FALSE, TRUE oder NULL (bzw. dessen Alias UNKNOWN).

Prädikate können an folgenden Stellen verwendet werden:

- In der SELECT-Liste sowie in den WHERE- und HAVING-Klauseln einer SELECT-Anfrage
- In der WHERE-Klausel der Befehle **UPDATE** und **DELETE**
- In der ON-Klausel und den WHERE-Klauseln des **MERGE**-Befehls

Folgende Prädikate werden aktuell unterstützt:

[Vergleichsprädikate](#)

[Logische Verknüpfungsprädikate](#)

[\[NOT\] BETWEEN](#)

[EXISTS](#)

[\[NOT\] IN](#)

[IS \[NOT\] NULL](#)

[\[NOT\] REGEXP_LIKE](#)

[\[NOT\] LIKE](#)

In welcher Reihenfolge die Prädikate bei komplexen Ausdrücken ausgewertet werden, bestimmt die Präzedenz der Prädikate. Nachfolgende Tabelle definiert diese in absteigender Reihenfolge, d.h. die Prädikate der ersten Zeile werden als erstes ausgewertet. Die gewünschte Auswertungsreihenfolge kann allerdings stets durch Klammerung der Ausdrücke festgelegt werden.

Tabelle 2.7. Präzedenz von Prädikaten

Prädikate	Bezeichnung
=, !=, <, <=, >, >=	Vergleichsprädikate
[NOT] BETWEEN, EXISTS, [NOT] IN, IS [NOT] NULL, [NOT] REGEXP_LIKE, [NOT] LIKE	Spezialprädikate
NOT	Logische Verneinung
AND	Konjunktion
OR	Disjunktion

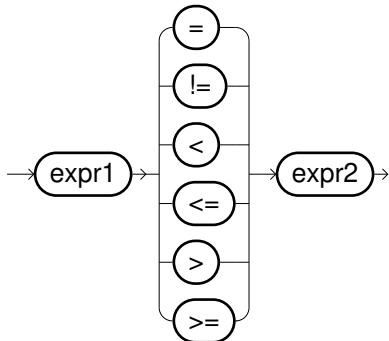
2.8.2. Liste der Prädikate

Vergleichsprädikate

Zweck

Vergleichsprädikate vergleichen zwei Ausdrücke und liefern zurück, ob der Vergleich korrekt ist.

Syntax



Anmerkung(en)

=	Gleichheitsprüfung
!=	Ungleichheitsprüfung (hierfür existieren auch die Aliase <> und ^=)
< bzw. <=	Prüfung auf "Kleiner" bzw. "Kleiner gleich"
> bzw. >=	Prüfung auf "Größer" bzw. "Größer gleich"
NULL-Werte	Falls einer der beiden Ausdrücke der NULL-Wert ist, so ist das Ergebnis ebenso der NULL-Wert.

Beispiel(e)

```

SELECT 1=1, 1<1, 1!=null;

1=1      1<1      1<>NULL
----- -----
TRUE     FALSE

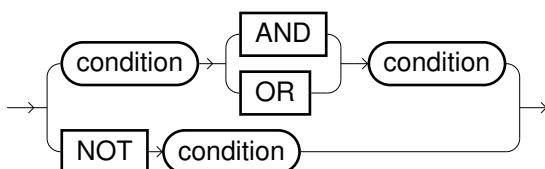
```

Logische Verknüpfungsprädikate

Zweck

Verknüpfungsprädikate bzw. boolesche Operatoren verknüpfen boolesche Werte.

Syntax



Anmerkung(en)

Folgende Tabellen definieren das Verhalten der Verknüpfungsoperatoren:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE

NULL	NULL	FALSE	NULL
OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL
	TRUE	FALSE	NULL
NOT	FALSE	TRUE	NULL

Beispiel(e)

```
SELECT true AND false AS res1,
      NOT false      AS res2,
      true OR null   AS res3;
```

RES1	RES2	RES3

FALSE	TRUE	TRUE

[NOT] BETWEEN**Zweck**

Testet, ob ein Ausdruck zwischen zwei Werten liegt.

Syntax**Anmerkung(en)**

- A BETWEEN B AND C ist äquivalent zu B <= A AND A <= C.

Beispiel(e)

```
SELECT 2 BETWEEN 1 AND 3 AS res;

RES
-----
TRUE
```

EXISTS**Zweck**

Testet, ob die angegebene Subquery Ergebniszeilen beinhaltet.

Syntax



Beispiel(e)

```

CREATE TABLE t (i DECIMAL);
INSERT INTO t VALUES 1,2;
CREATE TABLE t2 (j DECIMAL);
INSERT INTO t2 VALUES 2;

SELECT i FROM t WHERE EXISTS (SELECT * FROM t2);

i
-----
1
2

SELECT i FROM t WHERE EXISTS (SELECT * FROM t2 WHERE t.i=t2.j);

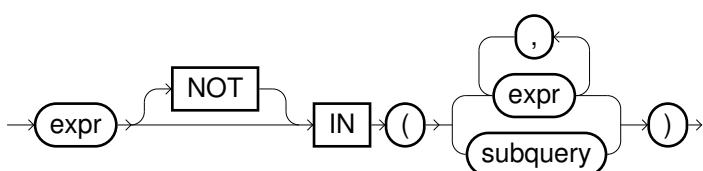
i
-----
2
  
```

[NOT] IN

Zweck

Testet, ob ein Ausdruck in einer Ergebnismenge enthalten ist.

Syntax



Beispiel(e)

```

CREATE TABLE t (x DECIMAL);
INSERT INTO t VALUES 1,2,3,4;

SELECT x FROM t WHERE x IN (2,4);

x
-----
2
4

CREATE TABLE t2 (y DECIMAL);
  
```

```
INSERT INTO t2 VALUES 2,4;

SELECT x FROM t WHERE x IN (SELECT y FROM t2);

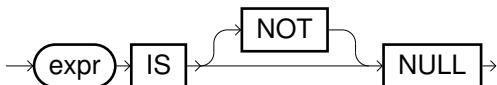
x
-----
2
4
```

IS [NOT] NULL

Zweck

Testet, ob ein Ausdruck der Nullwert ist.

Syntax



Beispiel(e)

```
SELECT null=null, null IS NULL, '' IS NOT NULL;
NULL=NULL NULL IS NULL '' IS NOT NULL
-----
TRUE          FALSE
```

[NOT] REGEXP_LIKE

Zweck

Dieses Prädikat gibt an, ob eine Zeichenkette mit einem regulären Ausdruck übereinstimmt.

Syntax



Anmerkung(en)

- Detaillierte Informationen und Beispiele zu regulären Ausdrücken in Exasol finden Sie in [Abschnitt 2.1.3, Reguläre Ausdrücke](#).
- Siehe auch die Funktionen [SUBSTR\[ING\]](#), [REGEXP_INSTR](#), [REGEXP_SUBSTR](#) und [REGEXP_REPLACE](#).

Beispiel(e)

```
SELECT 'My mail address is my_mail@exasol.com'
      REGEXP_LIKE '(?i).*[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}.*'
      AS contains_email;

CONTAINS_EMAIL
-----
TRUE
```

[NOT] LIKE**Zweck**

Ähnlich wie das Prädikat [\[NOT\] REGEXP_LIKE](#), allerdings mit einfacherem Ausdruck übereinstimmt.

Syntax**Anmerkung(en)**

- Sonderzeichen:
 - _ Wildcard für genau ein Zeichen
 - % Wildcard für eine beliebige Anzahl an Zeichen (auch keine Zeichen)
 - esc_chr Zeichen, mit dem die Zeichen _ und % im regulären Ausdruck verwendet werden können. Defaultmäßig ist dies der Wert der Session-Variable DEFAULT_LIKE_ESCAPE_CHARACTER (siehe auch [ALTER SESSION](#)).

Beispiel(e)

```
SELECT 'abcd' LIKE 'a_d' AS res1, '%bcd' like '\%%d' AS res2;

RES1  RES2
----- -----
FALSE TRUE
```

2.9. Built-in-Funktionen

Dieses Kapitel dokumentiert die von Exasol unterstützten Funktionen. Sie sind nicht zu verwechseln mit benutzerdefinierten Funktionen, die mittels [CREATE FUNCTION](#) angelegt werden können.

In den einzelnen Abschnitten werden die Funktionen nach ihrer Verwendungsart zusammengefasst. Die eigentliche Beschreibung der Funktionen ist in [Abschnitt 2.9.4, Alphabetische Liste aller Funktionen](#) zu finden.

2.9.1. Skalare Funktionen

Skalare Funktionen erhalten einen Eingabewert und liefern daraus einen Ergebniswert. Sie können auf konstante Werte, Spaltelemente einer Tabelle (bzw. View) sowie auf zusammengesetzte Wertausdrücke angewendet werden.

Beispiele:

```
SELECT SIN(1);  
SELECT LENGTH(s) FROM t;  
SELECT EXP(1+ABS(n)) FROM t;
```

Skalare Funktionen erwarten für ihre Argumente in der Regel einen speziellen Datentyp. Wenn dieser nicht gegeben ist, wird eine implizite Konvertierung in diesen Datentyp versucht bzw. eine Fehlermeldung ausgegeben.

Numerische Funktionen

Numerische Funktionen erhalten einen numerischen Wert als Eingabe und liefern in der Regel einen numerischen Wert als Ausgabe.

[ABS](#)
[ACOS](#)
[ASIN](#)
[ATAN](#)
[ATAN2](#)
[CEIL\[ING\]](#)
[COS](#)
[COSH](#)
[COT](#)
[DEGREES](#)
[DIV](#)
[EXP](#)
[FLOOR](#)
[LN](#)
[LOG](#)
[LOG10](#)
[LOG2](#)
[MOD](#)
[PI](#)
[POWER](#)
[RADIAN](#)
[RAND\[OM\]](#)
[ROUND \(number\)](#)
[SIGN](#)
[SIN](#)
[SINH](#)
[SQRT](#)

TAN
TANH
TO_CHAR (number)
TO_NUMBER
TRUNC[ATE] (number)

Zeichenketten Funktionen

Zeichenketten Funktionen können entweder eine Zeichenkette (z.B. LPAD) oder einen numerischen Wert (z.B. LENGTH) zurückliefern.

ASCII
BIT_LENGTH
CHARACTER_LENGTH
CH[A]R
COLOGNE_PHONETIC
CONCAT
DUMP
EDIT_DISTANCE
INSERT
INSTR
LCASE
LEFT
LENGTH
LOCATE
LOWER
LPAD
LTRIM
MID
OCTET_LENGTH
POSITION
REGEXP_INSTR
REGEXP_REPLACE
REGEXP_SUBSTR
REPEAT
REPLACE
REVERSE
RIGHT
RPAD
RTRIM
SOUNDEX
SPACE
SUBSTR[ING]
TO_CHAR (datetime)
TO_CHAR (number)
TO_NUMBER
TRANSLATE
TRIM
UCASE
UNICODE
UNICODECHR
UPPER

Datum/Zeit Funktionen

Datum/Zeit Funktionen behandeln die Datentypen DATE, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE und INTERVAL.

ADD_DAYS
ADD_HOURS
ADD_MINUTES
ADD_MONTHS
ADD_SECONDS
ADD_WEEKS
ADD_YEARS
CONVERT_TZ
CURDATE
CURRENT_DATE
CURRENT_TIMESTAMP
DATE_TRUNC
DAY
DAYS_BETWEEN
DBTIMEZONE
EXTRACT
FROM_POSIX_TIME
HOUR
HOURS_BETWEEN
LOCALTIMESTAMP
MINUTE
MINUTES_BETWEEN
MONTH
MONTHS_BETWEEN
NOW
NUMTODSINTERVAL
NUMTOYMINTERVAL
POSIX_TIME
ROUND (datetime)
SECOND
SECONDS_BETWEEN
SESSIONTIMEZONE
SYSDATE
SYSTIMESTAMP
TO_CHAR (datetime)
TO_DATE
TO_DSINTERVAL
TO_TIMESTAMP
TO_YMINTERVAL
TRUNC[ATE] (datetime)
WEEK
YEAR
YEARS_BETWEEN

Geodaten-Funktionen

Zur Analyse von Geodaten stehen eine Vielzahl von Funktionen zur Verfügung (siehe [ST_*](#) bzw. [Abschnitt 2.4, Geodaten](#)).

Bitfunktionen

Für Bitoperationen stehen diverse Funktionen zur Verfügung.

[BIT_AND](#)
[BIT_CHECK](#)
[BIT_LROTATE](#)
[BIT_LSHIFT](#)
[BIT_NOT](#)
[BIT_OR](#)
[BIT_RROTATE](#)
[BIT_RSHIFT](#)
[BIT_SET](#)
[BIT_TO_NUM](#)
[BIT_XOR](#)

Konvertierungsfunktionen

Mit Hilfe der Konvertierungsfunktionen können Werte in andere Datentypen umgewandelt werden.

[CAST](#)
[CONVERT](#)
[IS_*](#)
[NUMTODSINTERVAL](#)
[NUMTOYMINTERVAL](#)
[TO_CHAR \(datetime\)](#)
[TO_CHAR \(number\)](#)
[TO_DATE](#)
[TO_DSINTERVAL](#)
[TO_NUMBER](#)
[TO_TIMESTAMP](#)
[TO_YMINTERVAL](#)

Funktionen für hierarchische Queries

Diese Funktionen können Sie im Zusammenhang mit CONNECT BY Anfragen benutzen.

[CONNECT_BY_ISCYCLE](#)
[CONNECT_BY_ISLEAF](#)
[LEVEL](#)
[SYS_CONNECT_BY_PATH](#)

Sonstige skalare Funktionen

Hier sind diejenigen Funktionen aufgelistet, die sich nicht in eine der obigen Kategorien einordnen lassen.

[CASE](#)
[COALESCE](#)
[CURRENT_SCHEMA](#)
[CURRENT_SESSION](#)
[CURRENT_STATEMENT](#)
[CURRENT_USER](#)
[DECODE](#)
[GREATEST](#)
[HASH_MD5](#)
[HASH_SHA\[1\]](#)

HASH_TIGER
IPROC
LEAST
NULLIF
NULLIFZERO
NPROC
NVL
NVL2
ROWID
SYS_GUID
USER
VALUE2PROC
ZEROIFNULL

2.9.2. Aggregatsfunktionen

Aggregatsfunktionen beziehen sich stets auf eine Menge von Eingabewerten und liefern einen einzelnen Ergebniswert zurück. Wird über die GROUP BY Klausel die Tabelle in mehrere Gruppen unterteilt, so wird bei Aggregatsfunktionen für jede dieser Gruppen ein Wert berechnet.

Wird keine GROUP BY Klausel angegeben, so bezieht sich eine Aggregatsfunktionen immer auf die gesamte Tabelle. Eine solche Query liefert dann genau eine Ergebnissezeile zurück.

Aggregatsfunktionen werden manchmal auch als *Mengenfunktionen* bezeichnet.

APPROXIMATE_COUNT_DISTINCT
AVG
CORR
COUNT
COVAR_POP
COVAR_SAMP
FIRST_VALUE
GROUP_CONCAT
GROUPING[_ID]
LAST_VALUE
MAX
MEDIAN
MIN
PERCENTILE_CONT
PERCENTILE_DISC
REGR_*
ST_INTERSECTION (siehe [ST_*](#) bzw. [Abschnitt 2.4, Geodaten](#))
ST_UNION (siehe [ST_*](#) bzw. [Abschnitt 2.4, Geodaten](#))
STDDEV
STDDEV_POP
STDDEV_SAMP
SUM
VAR_POP
VAR_SAMP
VARIANCE

2.9.3. Analytische Funktionen

Analytische Funktionen beziehen sich stets auf Teilmengen der Daten. Diese sogenannten Partitionen werden durch die PARTITION BY - Klausel definiert. Wird sie nicht angegeben, bezieht sich die analytische Funktion auf die gesamte Tabelle. Durch die ORDER BY - Klausel kann festgelegt werden, wie die Daten innerhalb der

Partition sortiert werden. Für viele der analytischen Funktionen ist die Angabe der Reihenfolge zwingend, da sie für die Funktionsberechnung wesentlich ist.

Falls für eine analytische Funktion ein ORDER BY spezifiziert wurde, kann die Menge der für die Berechnung relevanten Datensätze durch die Definition eines Fensters (WINDOW) zusätzlich eingeschränkt werden. Normalerweise umfasst das Fenster die Datensätze vom Beginn der Partition bis zum aktuellen Datensatz, es kann aber auch explizit durch ROWS (physische Grenze) eingeschränkt werden. Die Einschränkung über RANGE (logische Grenze) wird von Exasol noch nicht unterstützt.

Analytische Funktionen werden bis auf das ORDER BY zum Schluss ausgeführt, also nach der Auswertung der WHERE-, GROUP BY- und HAVING-Klauseln. Daher dürfen sie nur in der SELECT-Liste oder der ORDER BY-Klausel verwendet werden.

Durch eine Vielzahl an statistischen Funktionen ermöglichen analytische Funktionen komplexe Auswertungen und Analysen und sind eine wertvolle Ergänzung zu den Aggregatsfunktionen (siehe [Abschnitt 2.9.2, Aggregatsfunktionen](#)). Folgende analytischen Funktionen werden unterstützt:

Avg
Corr
Count
Covar_Pop
Covar_Samp
Dense_Rank
First_Value
Lag
Last_Value
Lead
Max
Median
Min
Percentile_Cont
Percentile_Disc
Rank
Ratio_To_Report
Regr_*
Row_Number
StdDev
StdDev_Pop
StdDev_Samp
Sum
Var_Pop
Var_Samp
Variance

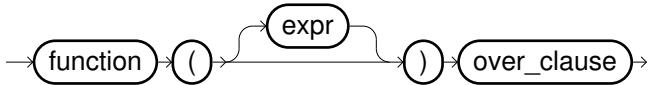
Analytische Query

Zweck

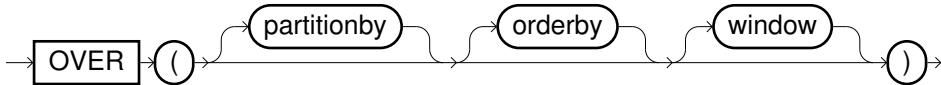
Analytische Funktionen werten eine Menge von Eingabewerten aus. Anders als Aggregatsfunktionen liefern sie aber für jede Datenbankzeile einen Ergebniswert zurück und nicht für jede Gruppe von Zeilen.

Syntax

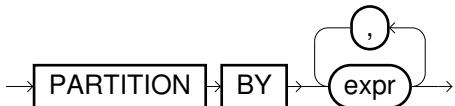
analytical_query :=



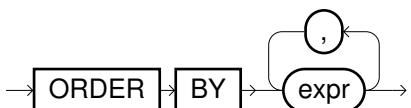
over_clause :=



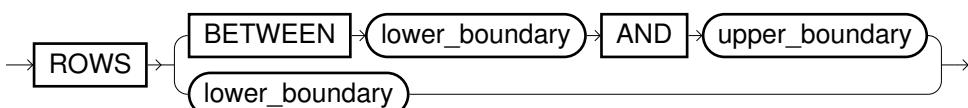
partitionby :=



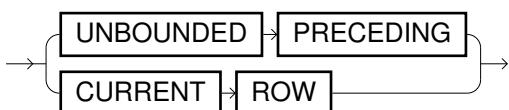
orderby :=



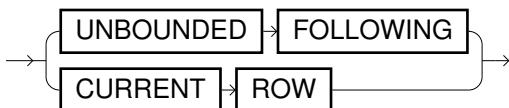
window :=



lower_boundary :=



upper_boundary :=



Anmerkung(en)

- Analytische Funktionen werden immer erst nach WHERE, GROUP BY und HAVING, aber vor einem (globalen) ORDER BY ausgewertet.
- Wird über die PARTITION BY Klausel die Tabelle in mehrere Partitionen unterteilt, so werden die Ergebnisse innerhalb jeder Partition unabhängig vom Rest der Tabelle berechnet. Wird keine PARTITION BY Klausel angegeben, so bezieht sich eine analytische Funktion immer auf die gesamte Tabelle.
- Eine zuverlässige Sortierung der Zeilen für eine analytische Funktion kann nur durch ORDER BY *innerhalb* der OVER - Klausel erreicht werden.
- Falls ein ORDER BY spezifiziert wurde, kann die Menge der für die Berechnung relevanten Datensätze durch die Definition eines Fensters (WINDOW) zusätzlich eingeschränkt werden. Default ist hierbei ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Beispiel(e)

```

SELECT age,
       FLOOR(age/10) || '0ies' AS agegroup,
       COUNT(*) OVER (
           PARTITION BY FLOOR(age/10)
           ORDER BY age
       ) AS COUNT
FROM staff;

AGE  AGEGROUP COUNT
-----
25  20ies      1
26  20ies      2
27  20ies      3
28  20ies      4
31  30ies      1
39  30ies      2

```

2.9.4. Alphabetische Liste aller Funktionen**ABS****Zweck**

Liefert den Absolutbetrag einer Zahl n.

Syntax

abs::=

→ **ABS** → (→ n →) →

Beispiel(e)

```

SELECT ABS(-123) ABS;

ABS
-----
123

```

ACOS**Zweck**

Liefert den Arcus Cosinus einer Zahl n. Das Ergebnis liegt zwischen 0 und π .

Syntax

acos::=



Anmerkung(en)

- Die Zahl n muss im Intervall [-1;1] liegen.

Beispiel(e)

```
SELECT ACOS(0.5) ACOS;  
  
ACOS  
-----  
1.047197551196598
```

ADD_DAYS

Zweck

Addiert auf ein Datum oder einen Zeitstempel eine angegebene Anzahl von Tagen.

Syntax

add_days::=



Anmerkung(en)

- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT ADD_DAYS(DATE '2000-02-28', 1) AD1,  
       ADD_DAYS(TIMESTAMP '2001-02-28 12:00:00', 1) AD2;  
  
AD1          AD2  
-----  -----  
2000-02-29  2001-03-01 12:00:00.000000
```

ADD_HOURS

Zweck

Addiert auf einen Zeitstempel eine angegebene Anzahl von Stunden.

Syntax

add_hours ::=



Anmerkung(en)

- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion intern in UTC berechnet, bevor das Ergebnis in die Session-Zeitzone umgewandelt wird.

Beispiel(e)

```

SELECT ADD_HOURS(TIMESTAMP '2000-01-01 00:00:00', 1) AH1,
       ADD_HOURS(TIMESTAMP '2000-01-01 12:23:45', -1) AH2;

AH1                               AH2
-----
2000-01-01 01:00:00.000000  2000-01-01 11:23:45.000000
  
```

ADD_MINUTES

Zweck

Addiert auf einen Zeitstempel eine angegebene Anzahl von Minuten.

Syntax

add_minutes ::=



Anmerkung(en)

- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion intern in UTC berechnet, bevor das Ergebnis in die Session-Zeitzone umgewandelt wird.

Beispiel(e)

```

SELECT ADD_MINUTES(TIMESTAMP '2000-01-01 00:00:00', -1) AM1,
       ADD_MINUTES(TIMESTAMP '2000-01-01 00:00:00', +2) AM2;

AM1                               AM2
-----
1999-12-31 23:59:00.000000  2000-01-01 00:02:00.000000
  
```

ADD_MONTHS

Zweck

Addiert auf ein Datum oder einen Zeitstempel eine angegebene Anzahl von Monaten.

Syntax

add_months::=

→ **ADD_MONTHS** → (→ **datetime** → , → **integer** →) →

Anmerkung(en)

- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Falls der resultierende Monat weniger Tage hat als der Tag des Eingabedatums, so wird der letzte Tag dieses Monats zurückgeliefert.
- Falls das Eingabedatum der letzte Tag des Monats ist, so wird der letzte Tag des resultierenden Monats zurückgegeben.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT ADD_MONTHS(DATE '2006-01-31', 1) AM1,
       ADD_MONTHS(TIMESTAMP '2006-01-31 12:00:00', 2) AM2;

AM1          AM2
-----
2006-02-28  2006-03-31 12:00:00.000000
```

ADD_SECONDS

Zweck

Addiert auf einen Zeitstempel eine angegebene Anzahl von Sekunden.

Syntax

add_seconds::=

→ **ADD_SECONDS** → (→ **datetime** → , → **decimal** →) →

Anmerkung(en)

- Es können bis zu drei Nachkommastellen verarbeitet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion intern in UTC berechnet, bevor das Ergebnis in die Session-Zeitzone umgewandelt wird.

Beispiel(e)

```
SELECT ADD_SECONDS(TIMESTAMP '2000-01-01 00:00:00', -1) AS1,
       ADD_SECONDS(TIMESTAMP '2000-01-01 00:00:00', +1.234) AS2;
AS1                                AS2
-----
1999-12-31 23:59:59.000000 2000-01-01 00:00:01.234000
```

ADD_WEEKS

Zweck

Addiert auf ein Datum oder einen Zeitstempel eine angegebene Anzahl von Wochen.

Syntax

add_weeks::=

→ **ADD_WEEKS** → (→ **datetime** → , → **integer** →) →

Anmerkung(en)

- ADD_WEEKS(x, n) ist identisch zu ADD_DAYS(x, n*7).
- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT ADD_WEEKS(DATE '2000-02-29', 1) AW1,
       ADD_WEEKS(TIMESTAMP '2005-01-31 12:00:00', -1) AW2;
AW1      AW2
-----
2000-03-07 2005-01-24 12:00:00.000000
```

ADD_YEARS

Zweck

Addiert auf ein Datum oder einen Zeitstempel eine angegebene Anzahl von Jahren.

Syntax

add_years::=

→ **ADD_YEARS** → (→ **datetime** → , → **integer** →) →

Anmerkung(en)

- Bei Dezimalzahlen wird die Zahl vor der Addition gerundet.
- Falls der resultierende Monat weniger Tage hat als der Tag des Eingabedatums, so wird der letzte Tag dieses Monats zurückgeliefert.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT ADD_YEARS(DATE '2000-02-29', 1) AY1,
       ADD_YEARS(TIMESTAMP '2005-01-31 12:00:00', -1) AY2;

AY1          AY2
----- -----
2001-02-28  2004-01-31 12:00:00.000000
```

APPROXIMATE_COUNT_DISTINCT

Zweck

Liefert einen Schätzwert für die Anzahl der disjunktten Werte (ohne NULL) zurück.

Syntax

approximate_count_distinct::=

→ **APPROXIMATE_COUNT_DISTINCT** → (→ expr →) →

Anmerkung(en)

- Das Ergebnis ist zwar nicht exakt wie in der Funktion **COUNT**, aber die Geschwindigkeit dafür erheblich schneller.
- Für die Berechnung wird intern der Algorithmus HyperLogLog verwendet.

Beispiel(e)

```
SELECT COUNT(DISTINCT customer_id) COUNT_EXACT,
       APPROXIMATE_COUNT_DISTINCT (customer_id) COUNT_APPR
  FROM orders WHERE price > 1000;

COUNT_EXACT COUNT_APPR
----- -----
      10000000      10143194
```

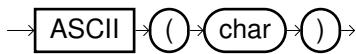
ASCII

Zweck

Liefert den numerischen Wert eines Zeichens im ASCII-Zeichensatz.

Syntax

ascii ::=



Anmerkung(en)

- Falls das Zeichen kein ASCII-Zeichen ist, wird eine Fehlermeldung geworfen.

Beispiel(e)

```
SELECT ASCII('X');

ASCII('X')
-----
88
```

ASIN

Zweck

Liefert den Arcus Sinus einer Zahl n. Das Ergebnis liegt zwischen $-\pi/2$ und $\pi/2$.

Syntax

asin ::=



Anmerkung(en)

- Die Zahl n muss im Intervall [-1;1] liegen.

Beispiel(e)

```
SELECT ASIN(1);

ASIN(1)
-----
1.570796326794897
```

ATAN

Zweck

Liefert den Arcus Tangens einer Zahl n. Das Ergebnis liegt zwischen $-\pi/2$ und $\pi/2$.

Syntax

atan ::=



Beispiel(e)

```

SELECT ATAN(1);
ATAN(1)
-----
0.785398163397448
  
```

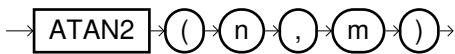
ATAN2

Zweck

Liefert den Arcus Tangens zweier Zahlen n und m. Dieser Ausdruck ist äquivalent zu ATAN(n/m).

Syntax

atan2 ::=



Beispiel(e)

```

SELECT ATAN2(1,1) ATAN2;
ATAN2(1,1)
-----
0.785398163397448
  
```

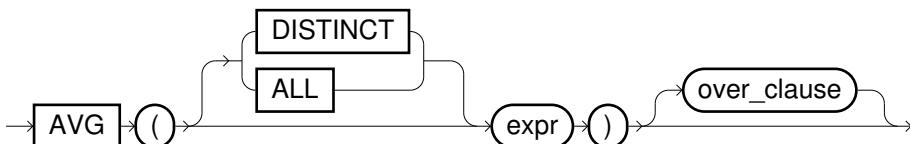
AVG

Zweck

Liefert den Durchschnittswert.

Syntax

avg ::=



Anmerkung(en)

- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Es werden nur numerische Operanden unterstützt.

Beispiel(e)

```
SELECT AVG(age) AVG FROM staff;

AVG
-----
36.25
```

BIT_AND

Zweck

Führt die logische UND-Verknüpfung auf zwei Bitfolgen aus.

Syntax

bit_and ::=

→ **BIT_AND** → (→ integer → , → integer →) →

Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
-- 1001 AND 0011 = 0001
SELECT BIT_AND(9,3);

BIT_AND(9,3)
-----
1
```

BIT_CHECK

Zweck

Prüft, ob ein Bit einer Bitfolge gesetzt ist. Die Nummerierung beginnt beim kleinsten Bit mit der Zahl 0.

Syntax

bit_check ::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Wert für pos darf zwischen 0 und 63 liegen.

Beispiel(e)

```
SELECT BIT_CHECK(3,0) B0,
       BIT_CHECK(3,1) B1,
       BIT_CHECK(3,2) B2,
       BIT_CHECK(3,3) B3;

B0      B1      B2      B3
----- -----
TRUE    TRUE    FALSE   FALSE
```

BIT_LENGTH

Zweck

Liefert die Bitlänge einer Zeichenkette. Verwendet man nur ASCII-Zeichen, dann ist diese Funktion äquivalent zu [CHARACTER_LENGTH](#) * 8.

Syntax

bit_length ::=



Beispiel(e)

```
SELECT BIT_LENGTH('aou') BIT_LENGTH;
BIT_LENGTH
-----
24

SELECT BIT_LENGTH('äöü') BIT_LENGTH;
BIT_LENGTH
-----
48
```

BIT_LROTATE

Zweck

Rotiert die Bits einer Zahl um die angegebene Anzahl an Stellen nach links.

Syntax

bit_lrotate::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der zweite Parameter muss zwischen 0 und 63 sein.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_LROTATE(1024,63);  
  
BIT_LROTATE(1024,63)  
-----  
      512
```

BIT_LSHIFT

Zweck

Verschiebt die Bits einer Zahl um die angegebene Anzahl an Stellen nach links.

Syntax

bit_lshift::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der zweite Parameter muss zwischen 0 und 63 sein.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_LSHIFT(1,10);  
  
BIT_LSHIFT(1,10)  
-----  
      1024
```

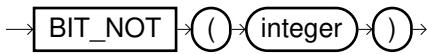
BIT_NOT

Zweck

Berechnet das bitweise Komplement (logische Negation) einer Bitfolge.

Syntax

bit_not::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_NOT(0), BIT_NOT(18446744073709551615);

BIT_NOT(0)          BIT_NOT(18446744073709551615)
-----  -----
18446744073709551615          0

SELECT BIT_AND( BIT_NOT(1), 5);

BIT_AND(BIT_NOT(1),5)
-----
4
```

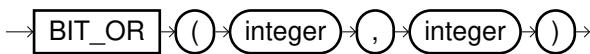
BIT_OR

Zweck

Führt die logische ODER-Verknüpfung auf zwei Bitfolgen aus.

Syntax

bit_or::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
-- 1001 OR 0011 = 1011
SELECT BIT_OR(9,3);

BIT_OR(9,3)
-----
11
```

BIT_RROTATE

Zweck

Rotiert die Bits einer Zahl um die angegebene Anzahl an Stellen nach rechts.

Syntax

bit_rrotate ::=

→ **BIT_RROTATE** → (→ **integer** → , → **integer** →) →

Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der zweite Parameter muss zwischen 0 und 63 sein.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_RROTATE(1024,63);

BIT_RROTATE(1024,63)
-----
2048
```

BIT_RSHIFT

Zweck

Verschiebt die Bits einer Zahl um die angegebene Anzahl an Stellen nach rechts.

Syntax

bit_rshift ::=

→ **BIT_RSHIFT** → (→ **integer** → , → **integer** →) →

Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.

- Der zweite Parameter muss zwischen 0 und 63 sein.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_RSHIFT(1024,10);  
  
BIT_RSHIFT(1024,10)  
-----  
1
```

BIT_SET

Zweck

Setzt ein Bit in einer Bitfolge. Die Nummerierung beginnt beim kleinsten Bit mit der Zahl 0.

Syntax

bit_set::=

→ **BIT_SET** → (→ integer → , → pos →) →

Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Wert für pos darf zwischen 0 und 63 liegen.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
SELECT BIT_SET(8,0);  
  
BIT_SET(8,0)  
-----  
9
```

BIT_TO_NUM

Zweck

Berechnet aus einer Bitfolge einen numerischen Wert.

Syntax

bit_to_num::=

→ **BIT_TO_NUM** → (→ digit → , →) →

Anmerkung(en)

- Jedes Argument der Funktion muss sich als 0 oder 1 auswerten lassen.
- Das erste Argument wird als größtes Bit interpretiert.
- Es können maximal 64 Argumente übergeben werden und somit positive Zahlen zwischen 0 und 18446744073709551615 generiert werden.

Beispiel(e)

```
SELECT BIT_TO_NUM(1,1,0,0);  
  
BIT_TO_NUM(1,1,0,0)  
-----  
12
```

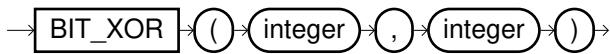
BIT_XOR

Zweck

Führt die logische XOR-Verknüpfung (exklusives ODER) auf zwei Bitfolgen aus. Das Ergebnis-Bit ist 1, wenn die beiden entsprechenden Eingabe-Bits unterschiedlich sind.

Syntax

bit_xor ::=



Anmerkung(en)

- Bitfunktionen sind auf 64 Bits limitiert und somit auf positive Zahlen zwischen 0 und 18446744073709551615.
- Der Ergebnisdatentyp ist ein DECIMAL(20,0).

Beispiel(e)

```
-- 1001 XOR 0011 = 1010  
SELECT BIT_XOR(9,3);  
  
BIT_XOR(9,3)  
-----  
10
```

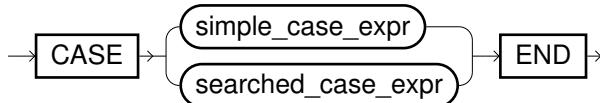
CASE

Zweck

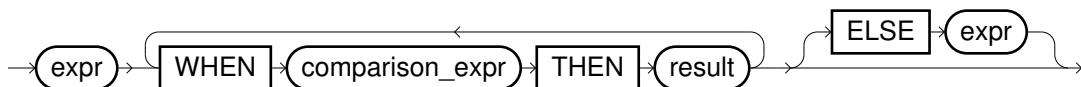
Mit Hilfe der CASE-Funktion kann eine IF THEN ELSE -Logik innerhalb der SQL-Sprache ausgedrückt werden.

Syntax

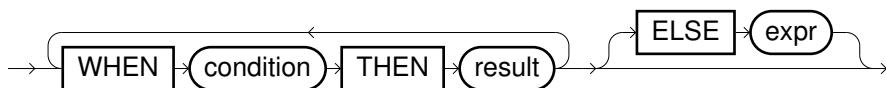
case ::=



simple_case_expr::=



searched_case_expr::=



Anmerkung(en)

- Bei der simple_case_expr wird expr mit den angegebenen Alternativen verglichen. Der THEN-Teil der ersten Übereinstimmung definiert das Ergebnis.
- Bei der searched_case_expr werden der Reihe nach alle Bedingungen ausgewertet, bis eine den Wahrheitswert TRUE ergibt. Der THEN-Teil dieser Bedingung ist dann das Ergebnis.
- Falls jeweils keine der Auswahlmöglichkeiten zutrifft, wird der ELSE-Wert zurückgegeben. Falls dieser nicht angegeben wurde, so wird der NULL-Wert zurückgeliefert.

Beispiel(e)

```

SELECT name, CASE note WHEN 1 THEN 'SEHR GUT'
                      WHEN 2 THEN 'GUT'
                      WHEN 3 THEN 'BEFRIEDIGEND'
                      WHEN 4 THEN 'AUSREICHEND'
                      WHEN 5 THEN 'UNGENUEGEND'
                      WHEN 6 THEN 'MANGELHAFT'
                      ELSE 'INVALID'
END AS NOTE FROM schueler;

```

NAME	NOTE
Fischer	SEHR GUT
Schmidt	AUSREICHEND

```

SELECT name, CASE WHEN umsatz>1000 THEN 'PREMIUM'
                  ELSE 'STANDARD'
END AS CLASS FROM kunden;

```

NAME	CLASS
Meier	STANDARD
Huber	PREMIUM

CAST

Zweck

Wandelt einen Ausdruck in den angegebenen Datentyp um. Falls dies nicht möglich ist, wird eine Exception erzeugt.

Syntax

cast::=



Anmerkung(en)

- **CONVERT** ist ein Alias für diese Funktion.

Beispiel(e)

```
SELECT CAST('ABC' AS CHAR(15)) STRINGCAST;
STRINGCAST
-----
ABC

SELECT CAST('2006-01-01' AS DATE) DATECAST;
DATECAST
-----
2006-01-01
```

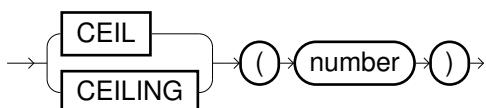
CEIL[ING]

Zweck

Liefert die kleinste ganze Zahl, die größer oder gleich dem angegebenen Parameter ist.

Syntax

ceiling::=



Beispiel(e)

```
SELECT CEIL(0.234) CEIL;
CEIL
-----
1
```

CHARACTER_LENGTH

Zweck

Liefert die Länge einer Zeichenkette (Anzahl der Zeichen).

Syntax

character_length ::=



Beispiel(e)

```
SELECT CHARACTER_LENGTH( 'aeiouäöü' ) C_LENGTH;  
C_LENGTH  
-----  
8
```

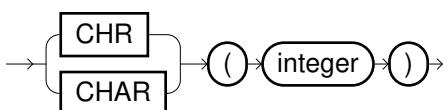
CH[A]R

Zweck

Liefert das ASCII-Zeichen, das dem angegebenen Parameter in der ASCII-Tabelle entspricht.

Syntax

chr ::=



Anmerkung(en)

- Die Zahl n muss zwischen 0 und 127 sein.
- CHR (0) liefert NULL zurück.

Beispiel(e)

```
SELECT CHR( 88 ) CHR;  
CHR  
---  
X
```

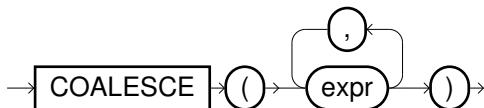
COALESCE

Zweck

Liefert den ersten Wert aus der Argumentliste, der nicht NULL ist. Falls alle Werte NULL sind, so liefert die Funktion den Wert NULL.

Syntax

coalesce::=



Anmerkung(en)

- Die Mindestanzahl an Argumenten beträgt 2.
- Die Funktion COALESCE(expr1, expr2) ist äquivalent zum CASE-Ausdruck CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END

Beispiel(e)

```
SELECT COALESCE(NULL, 'abc',NULL,'xyz') COALES;
COALES
-----
abc
```

COLOGNE_PHONETIC

Zweck

Diese Funktion definiert die Laut-Ähnlichkeit von Zeichenketten. Hiermit können unterschiedlich geschriebene, aber ähnlich ausgesprochene Wörter verglichen werden.

Syntax

cologne_phonetic::=



Anmerkung(en)

- Eine Beschreibung zum verwendeten Algorithmus finden Sie unter http://de.wikipedia.org/wiki/Kölner_Phonetik.
- Das Ergebnis ist stets eine Zeichenkette aus Ziffern, deren Länge maximal die doppelte Länge des Eingabe-Strings ist.
- Diese Funktion ist ähnlich zur Funktion **SOUNDEX**, allerdings besser für deutsche Wörter geeignet.
- Zur Berechnung des Unterschieds zwischen zwei Zeichenketten können Sie die Funktion **EDIT_DISTANCE** benutzen.

Beispiel(e)

```
SELECT COLOGNE_PHONETIC('schmitt'), COLOGNE_PHONETIC('Schmidt');  
CLOGNE_PHONETIC('schmitt') COLOGNE_PHONETIC('Schmidt')  
-----  
862 862
```

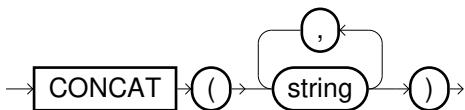
CONCAT

Zweck

Liefert die Konkatenation aus einer Anzahl von Strings.

Syntax

concat ::=



Anmerkung(en)

- Diese Funktion ist äquivalent zum [Abschnitt 2.7.2, Konkatenationsoperator ||](#).

Beispiel(e)

```
SELECT CONCAT('abc', 'def') CONCAT;  
CONCAT  
-----  
abcdef
```

CONNECT_BY_ISCYCLE

Zweck

Liefert zu einer CONNECT BY Anfrage, ob eine Zeile einen Zyklus im Baum verursacht. Details finden Sie in der Dokumentation zum [SELECT Befehl in Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Syntax

connect_by_iscycle ::=

→ **CONNECT_BY_ISCYCLE** →

Beispiel(e)

```
SELECT CONNECT_BY_ISCYCLE,
       SYS_CONNECT_BY_PATH(last_name, '/') "PATH"
  FROM employees WHERE last_name = 'Clark'
CONNECT BY NOCYCLE PRIOR employee_id = manager_id
START WITH last_name = 'Clark';

CONNECT_BY_ISCYCLE PATH
-----
0 /Clark
1 /Clark/Jackson/Johnson/Clark
```

CONNECT_BY_ISLEAF

Zweck

Liefert zu einer CONNECT BY Anfrage, ob eine Zeile ein Blatt im Baum ist. Details finden Sie in der Dokumentation zum [SELECT Befehl](#) in [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Syntax

connect_by_isleaf ::=

→ **CONNECT_BY_ISLEAF** →

Beispiel(e)

```
SELECT last_name, CONNECT_BY_ISLEAF,
       SYS_CONNECT_BY_PATH(last_name, '/') "PATH"
  FROM employees
CONNECT BY PRIOR employee_id = manager_id
START WITH last_name = 'Clark';

LAST_NAME CONNECT_BY_ISLEAF PATH
-----
Clark          0 /Clark
Smith          0 /Clark/Smith
Brown          1 /Clark/Smith/Brown
Jones          1 /Clark/Smith/Jones
```

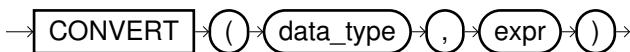
CONVERT

Zweck

Wandelt einen Ausdruck in den angegebenen Datentyp um, falls dies möglich ist.

Syntax

convert ::=



Anmerkung(en)

- Diese Funktion ist ein Alias für [CAST](#).

Beispiel(e)

```

SELECT CONVERT( CHAR(15) , 'ABC' ) STRINGCAST;
STRINGCAST
-----
ABC

SELECT CONVERT( DATE , '2006-01-01' ) DATECAST;
DATECAST
-----
2006-01-01
  
```

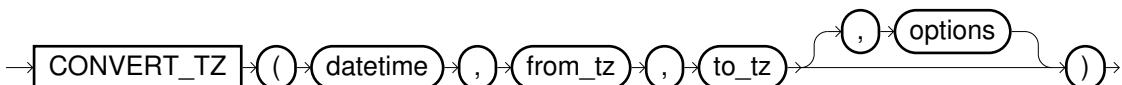
CONVERT_TZ

Zweck

Wandelt einen Zeitstempel zwischen zwei Zeitzonen um. Bitte beachten Sie, dass ein Zeitstempel selbst keine Zeitzoneninformation enthält. Diese Funktion addiert daher lediglich die Zeitverschiebung zwischen zwei Zeitzonen.

Syntax

convert_tz::=



Anmerkung(en)

- Die Liste der unterstützten Zeitzonen finden Sie in der Systemtabelle [EXA_TIME_ZONES](#).
- Falls der Eingabewert vom Typ TIMESTAMP WITH LOCAL TIME ZONE ist, dann ist diese Funktion nur erlaubt, sofern die Session-Zeitzone ([SESSIONTIMEZONE](#)) mit der Ursprungs-Zeitzone (Parameter `from_tz`) übereinstimmt. Das Ergebnis hat allerdings auch dann den Typ TIMESTAMP.
- Der optionale vierte Parameter (String) enthält Optionen zur Behandlung von problematischen Eingabedaten durch Zeitumstellungen. Folgenden Alternativen sind möglich:

```

'INVALID SHIFT AMBIGUOUS ST'
'INVALID SHIFT AMBIGUOUS DST'
'INVALID SHIFT AMBIGUOUS NULLIFY'
'INVALID SHIFT AMBIGUOUS REJECT'
'INVALID ADJUST AMBIGUOUS ST'
'INVALID ADJUST AMBIGUOUS DST'
'INVALID ADJUST AMBIGUOUS NULLIFY'
'INVALID ADJUST AMBIGUOUS REJECT'
  
```

```
'INVALID NULLIFY AMBIGUOUS ST'
'INVALID NULLIFY AMBIGUOUS DST'
'INVALID NULLIFY AMBIGUOUS NULLIFY'
'INVALID NULLIFY AMBIGUOUS REJECT'
'INVALID REJECT AMBIGUOUS ST'
'INVALID REJECT AMBIGUOUS DST'
'INVALID REJECT AMBIGUOUS NULLIFY'
'INVALID REJECT AMBIGUOUS REJECT'
'ENSURE REVERSIBILITY'
```

Details zu den einzelnen Optionen finden Sie im Abschnitt [Datum/Zeit Datentypen](#) in [Abschnitt 2.3, Datentypen](#). Der letzte Eintrag ist eine spezielle Option, mit der die Rückwärtskonvertierbarkeit sichergestellt bleibt. Es wird dann stets eine Fehlermeldung geworfen, falls die Eingabedaten invalid oder mehrdeutig sind, sowie falls der Ergebnis-Zeitstempel mehrdeutig wäre (dadurch wäre eine eindeutige Rückkonvertierung nicht mehr möglich).

Falls kein vierter Parameter angegeben wurde, so ist das Default-Verhalten durch die Session-Variable `TIME_ZONE_BEHAVIOR` bestimmt (siehe [ALTER SESSION](#)).

Beispiel(e)

```
SELECT CONVERT_TZ(TIMESTAMP '2012-05-10 12:00:00',
                  'UTC',
                  'Europe/Berlin') CONVERT_TZ;

CONVERT_TZ
-----
2012-05-10 14:00:00

SELECT CONVERT_TZ(TIMESTAMP '2012-03-25 02:30:00',
                  'Europe/Berlin',
                  'UTC',
                  'INVALID REJECT AMBIGUOUS REJECT') CONVERT_TZ;

Error: [22034] data exception -
       unable to convert timestamp between timezones: invalid timestamp
```

CORR

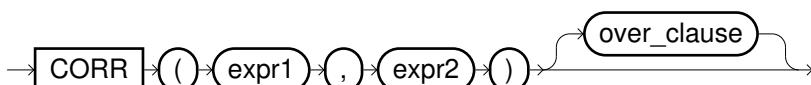
Zweck

Liefert den Korrelationskoeffizienten von Zahlenpaaren innerhalb einer Population (eine Art Zusammenhangsmaß), was folgender Formel entspricht:

$$\text{CORR(expr1, expr2)} = \frac{\text{COVAR_POP(expr1, expr2)}}{\text{STDDEV_POP(expr1) · STDDEV_POP(expr2)}}$$

Syntax

`corr ::=`



Anmerkung(en)

- Besitzt `expr1` oder `expr2` den Wert `NULL`, so wird das entsprechende Zahlenpaar bei der Berechnung ignoriert.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```
SELECT industry, CORR(age, salary) CORR
FROM staff GROUP BY industry;

INDUSTRY      CORR
-----
Finance      0.966045513268967
IT           0.453263345203583
```

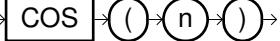
COS

Zweck

Liefert den Cosinus einer Zahl n.

Syntax

`cos ::=`

→ 

Beispiel(e)

```
SELECT COS(PI() / 3);

COS(PI() / 3)
-----
0.5
```

COSH

Zweck

Liefert den Cosinus Hyperbolicus einer Zahl n.

Syntax

`cosh ::=`

→ 

Beispiel(e)

```
SELECT COSH(1);

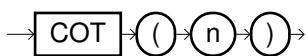
COSH(1)
-----
1.543080634815244
```

COT**Zweck**

Liefert den Kotangens einer Zahl n.

Syntax

cot::=

**Beispiel(e)**

```
SELECT COT(1);

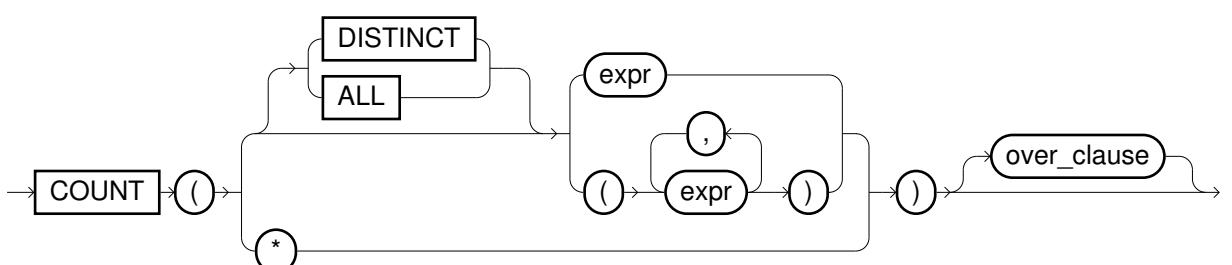
COT(1)
-----
0.642092615934331
```

COUNT**Zweck**

Liefert die Anzahl der Zeilen in der Ergebnismenge einer SQL-Anfrage zurück.

Syntax

count::=

**Anmerkung(en)**

- Falls * angegeben ist, so werden alle Zeilen gezählt.

- Falls ein Ausdruck angegeben ist, so werden die NULL-Werte nicht gezählt. Bei Tupeln werden diejenigen nicht gewertet, die nur aus NULL-Werten bestehen.
- Bei DISTINCT werden doppelte Einträge nur einmal gezählt, bei ALL mehrfach.
- Wird weder ALL noch DISTINCT angegeben, so ist ALL der Default-Wert.
- Eine schnelle, aber nicht exakte Alternative zu COUNT(DISTINCT) ist die Funktion [APPROXIMATE_COUNT_DISTINCT](#).

Beispiel(e)

```
SELECT COUNT(*) CNT_ALL FROM staff;
CNT_ALL
-----
10

SELECT COUNT(DISTINCT salary) CNT_DIST FROM staff;
CNT_DIST
-----
8
```

COVAR_POP

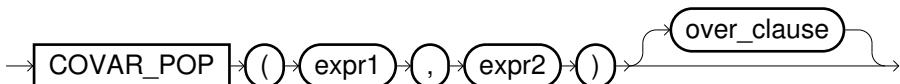
Zweck

Liefert die Kovarianz von Zahlenpaaren innerhalb einer Population (eine Art Zusammenhangsmaß), was folgender Formel entspricht:

$$\text{COVAR_POP(expr1, expr2)} = \frac{\sum_{i=1}^n (\text{expr}_1 - \bar{\text{expr}}_1)(\text{expr}_2 - \bar{\text{expr}}_2)}{n}$$

Syntax

covar_pop ::=



Anmerkung(en)

- Besitzt expr1 oder expr2 den Wert NULL, so wird das entsprechende Zahlenpaar bei der Berechnung ignoriert.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```
SELECT industry, COVAR_POP(age, salary) COVAR_POP
FROM staff GROUP BY industry;
INDUSTRY    COVAR_POP
-----
```

Finance	209360
IT	31280

COVAR_SAMP

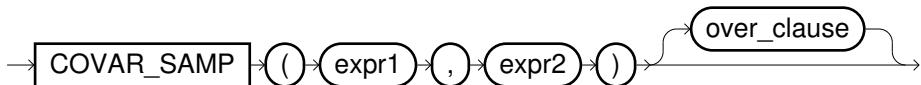
Zweck

Liefert die Kovarianz von Zahlenpaaren innerhalb einer Stichprobe (eine Art Zusammenhangsmaß), was folgender Formel entspricht:

$$\text{COVAR_SAMP(expr1, expr2)} = \frac{\sum_{i=1}^n (\text{expr1}_i - \overline{\text{expr1}})(\text{expr2}_i - \overline{\text{expr1}})}{n - 1}$$

Syntax

covar_samp ::=



Anmerkung(en)

- Besitzt expr1 oder expr2 den Wert NULL, so wird das entsprechende Zahlenpaar bei der Berechnung ignoriert.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT industry, COVAR_SAMP(age, salary) COVAR_SAMP
FROM staff GROUP BY industry;

INDUSTRY      COVAR_SAMP
-----
Finance        261700
IT             39100
  
```

CURDATE

Zweck

Liefert das aktuelle Datum, indem TO_DATE(CURRENT_TIMESTAMP) ausgewertet wird.

Syntax

curdate ::=



Anmerkung(en)

- Diese Funktion ist ein Alias auf [CURRENT_DATE](#).

Beispiel(e)

```
SELECT CURDATE( ) CURDATE;  
  
CURDATE  
-----  
1999-12-31
```

CURRENT_DATE

Zweck

Liefert das aktuelle Datum, indem TO_DATE(CURRENT_TIMESTAMP) ausgewertet wird.

Syntax

current_date ::=

→ **CURRENT_DATE** →

Anmerkung(en)

- Siehe auch Funktion [CURDATE](#).

Beispiel(e)

```
SELECT CURRENT_DATE;  
  
CURRENT_DATE  
-----  
1999-12-31
```

CURRENT_SCHEMA

Zweck

Liefert das aktuell geöffnete Schema. Falls kein Schema geöffnet ist, so ist das Ergebnis der NULL-Wert.

Syntax

current_schema ::=

→ **CURRENT_SCHEMA** →

Beispiel(e)

```
SELECT CURRENT_SCHEMA;

CURRENT_SCHEMA
-----
MY_SCHEMA
```

CURRENT_SESSION

Zweck

Liefert die ID der aktuellen Session. Diese ID wird beispielsweise auch in der Systemtabelle [EXA_ALL_SESSIONS](#) referenziert.

Syntax

current_session ::=

→ **CURRENT_SESSION** →

Beispiel(e)

```
SELECT CURRENT_SESSION;

CURRENT_SESSION
-----
7501910697805018352
```

CURRENT_STATEMENT

Zweck

Liefert die ID des aktuellen Statements. Dies ist eine fortlaufende Nummer innerhalb der aktuellen Session.

Syntax

current_statement ::=

→ **CURRENT_STATEMENT** →

Beispiel(e)

```
SELECT CURRENT_STATEMENT;

CURRENT_STATEMENT
-----
26
```

CURRENT_TIMESTAMP

Zweck

Liefert den aktuellen Zeitstempel zurück, interpretiert in der aktuellen Session-Zeitzone.

Syntax

current_timestamp ::=

→ **CURRENT_TIMESTAMP** →

Anmerkung(en)

- Der Rückgabewert dieser Funktion ist vom Typ TIMESTAMP WITH LOCAL TIME ZONE.
- Als Alias für CURRENT_TIMESTAMP kann auch die Funktion **NOW** verwendet werden.
- Weitere Funktionen zum aktuellen Zeitpunkt:
 - **LOCALTIMESTAMP**
 - **SYSTIMESTAMP**

Beispiel(e)

```
SELECT CURRENT_TIMESTAMP;  
  
CURRENT_TIMESTAMP  
-----  
1999-12-31 23:59:59
```

CURRENT_USER

Zweck

Gibt den aktuellen Benutzer zurück.

Syntax

current_user ::=

→ **CURRENT_USER** →

Beispiel(e)

```
SELECT CURRENT_USER;  
  
CURRENT_USER  
-----  
SYS
```

DATE_TRUNC

Zweck

PostgreSQL-kompatible Funktion zum Abrunden von Datums und Zeitstempeln.

Syntax

date_trunc::=

$\rightarrow \boxed{\text{DATE_TRUNC}} \rightarrow (\rightarrow \text{format} \rightarrow , \rightarrow \text{datetime} \rightarrow) \rightarrow$

Anmerkung(en)

- Für das Format können die folgenden Elemente verwendet werden: 'microseconds', 'milliseconds', 'second', 'minute', 'hour', 'day', 'week', 'month', 'quarter', 'year', 'decade', 'century', 'millennium'
- Der erste Tag einer Woche (für 'week') wird über den Parameter NLS_FIRST_DAY_OF_WEEK definiert (siehe [ALTER SESSION](#) und [ALTER SYSTEM](#)).
- Fast die gleiche Funktionalität bietet die Oracle-kompatible Funktion [TRUNC\[ATE\] \(datetime\)](#).
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT DATE_TRUNC( 'month' , DATE '2006-12-31' ) DATE_TRUNC;
DATE_TRUNC
-----
2006-12-01

SELECT DATE_TRUNC( 'minute' , TIMESTAMP '2006-12-31 23:59:59' ) DATE_TRUNC;
DATE_TRUNC
-----
2006-12-31 23:59:00.000000
```

DAY

Zweck

Liefert den Tag eines Datums.

Syntax

day::=

$\rightarrow \boxed{\text{DAY}} \rightarrow (\rightarrow \text{date} \rightarrow) \rightarrow$

Anmerkung(en)

- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.

- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT DAY(DATE '2010-10-20');
```

```
DAY
---
20
```

DAYS_BETWEEN

Zweck

Liefert die Anzahl der Tage zwischen zwei Datumswerten.

Syntax

days_between ::=

→ **DAY(BETWEEN)** → (→ **datetime1** → , → **datetime2** →) →

Anmerkung(en)

- Bei der Eingabe eines Zeitstempels geht nur das in ihm enthaltene Datum in die Berechnung ein.
- Falls der erste Datumswert vor dem zweiten Datumswert liegt, so ist das Ergebnis negativ.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT DAYS_BETWEEN(DATE '1999-12-31', DATE '2000-01-01') DB1,
       DAYS_BETWEEN(TIMESTAMP '2000-01-01 12:00:00',
                     TIMESTAMP '1999-12-31 00:00:00') DB2;
```

```
DB1      DB2
-----
-1          1
```

DBTIMEZONE

Zweck

Liefert die Datenbank-Zeitzone zurück, welche systemweit in EXAoperation eingestellt wird. Dies ist die lokale Zeitzone auf den EXASOL Rechnern.

Syntax

dbtimezone ::=

→ DBTIMEZONE →

Anmerkung(en)

- Siehe auch [SESSIONTIMEZONE](#).

Beispiel(e)

```
SELECT DBTIMEZONE;

DBTIMEZONE
-----
EUROPE/BERLIN
```

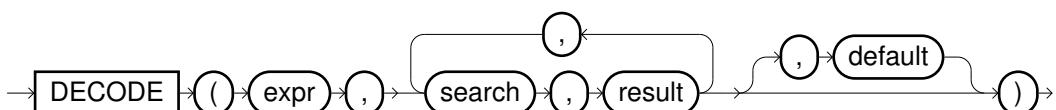
DECODE

Zweck

Die Decode-Funktion liefert denjenigen result-Wert zurück, für den der Ausdruck expr mit dem Ausdruck search übereinstimmt. Falls keine Übereinstimmung gefunden wird, so wird NULL oder – falls angegeben – der default-Wert zurückgeliefert.

Syntax

decode ::=



Anmerkung(en)

- Decode besitzt eine ähnliche Funktionalität wie [CASE](#), verhält sich jedoch geringfügig anders (aus Kompatibilitätsgründen zu anderen Datenbanken):
 - Der Ausdruck expr kann direkt mit NULL verglichen werden (z.B. DECODE(my_column, NULL, 0, my_column))
 - Vergleiche auf Zeichenketten "non-padded" durchgeführt (daher liefert DECODE(my_column, 'abc', TRUE, FALSE) auf einer CHAR(10)-Spalte stets das Ergebnis false)
 - Aus Gründen der besseren Lesbarkeit wird die Verwendung von [CASE](#) empfohlen.

Beispiel(e)

```
SELECT DECODE('abc', 'xyz', 1, 'abc', 2, 3) DECODE;

DECODE
-----
2
```

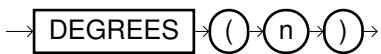
DEGREES

Zweck

Konvertiert die Zahl n vom Bogen- ins Gradmaß.

Syntax

degrees ::=



Anmerkung(en)

- Siehe auch die Funktion [RADIAN](#).
- Eingabewerte sind beliebige Zahlen.
- Der Wertebereich der Funktion beträgt (-180;180].

Beispiel(e)

```

SELECT DEGREES(PI());
DEGREES(PI())
-----
180
  
```

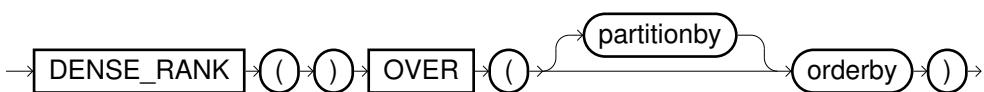
DENSE_RANK

Zweck

Liefert den Rang einer Zeile innerhalb einer geordneten Partition.

Syntax

dense_rank ::=



Anmerkung(en)

- DENSE_RANK kann nur als analytische Funktion (in Verbindung mit OVER(...)) verwendet werden (siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)).
- Die OVER-Klausel muss einen ORDER BY-Teil und darf keine Window-Klausel enthalten.
- Für gleichrangige Zeilen wird derselbe Wert geliefert. Folgende Werte werden aber nicht - wie bei [RANK](#) übersprungen.

Beispiel(e)

```
SELECT name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) dense_rank
FROM staff ORDER BY dense_rank;
```

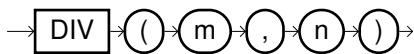
NAME	SALARY	DENSE_RANK
Schmidt	81000	1
Müller	75000	2
Huber	48000	3
Schultze	37000	4
Schulze	37000	4
Meier	25000	5

DIV**Zweck**

Liefert als Ergebnis die Ganzzahldivision von m durch n.

Syntax

div ::=

**Beispiel(e)**

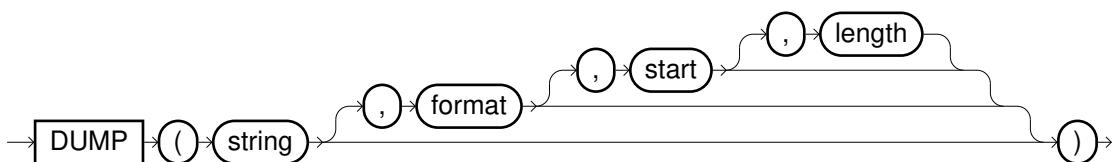
```
SELECT DIV(15,6) DIV;
DIV
---
2
```

DUMP**Zweck**

Liefert die Länge der Zeichenkette string in Bytes, den verwendeten Zeichensatz und den internen Speicherinhalt der mittels start und length ausgewählten Zeichen, in der mit format angegebenen Repräsentation.

Syntax

dump ::=



Anmerkung(en)

- Es gibt vier gültige Repräsentations-Formate:
 - 8: Oktal
 - 10: Dezimal (Default)
 - 16: Hexadezimal
 - 17: ASCII-Zeichen werden direkt ausgegeben, Mehr-Byte-Zeichen hingegen hexadezimal.
- Gibt man keine Länge an oder length=0, dann werden alle Zeichen ab der Startposition (`start`) dargestellt. Bei negativen Längenangaben wird der Betrag genommen.
- Bei der Eingabe einer negativen Startposition `start=-n` werden bis zu `length` Zeichen beginnend mit dem `n`-letzten Zeichen dargestellt. Ist allerdings `n` größer als die Zeichenanzahl von `string`, so wird NULL ausgegeben. Wird keine Startposition angegeben, dann wird ab dem ersten Zeichen begonnen (`n=1`).
- Ist `string=NULL`, dann wird NULL als Zeichenkette ausgegeben.

Beispiel(e)

```
SELECT DUMP( '123abc' ) DUMP;
DUMP
-----
Len=6 CharacterSet=ASCII: 49,50,51,97,98,99

SELECT DUMP( 'üäö45' ,16 ) DUMP;
DUMP
-----
Len=8 CharacterSet=UTF8: c3,bc,c3,a4,c3,b6,34,35
```

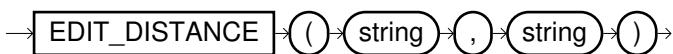
EDIT_DISTANCE

Zweck

Diese Funktion definiert den Abstand zwischen zwei Zeichenketten. Hiermit können beschrieben werden, wie ähnlich sich zwei Zeichenketten sind.

Syntax

`edit_distance ::=`



Anmerkung(en)

- Zur Prüfung auf phonetische Äquivalenz bieten sich die Funktionen `SOUNDEX` und `COLOGNE_PHONETIC` an.
- Es wird die Anzahl an Änderungen berechnet, mit denen die erste Zeichenkette in die andere umgewandelt werden kann.
- Das Ergebnis ist eine Zahl zwischen 0 und der Länge der längeren Zeichenkette.

Beispiel(e)

```
SELECT EDIT_DISTANCE('schmitt', 'Schmidt');

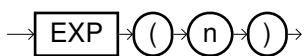
EDIT_DISTANCE('schmitt', 'Schmidt')
-----
2
```

EXP**Zweck**

Liefert die n-te Potenz der Zahl e (Eulersche Zahl).

Syntax

exp ::=

**Beispiel(e)**

```
SELECT EXP(1);

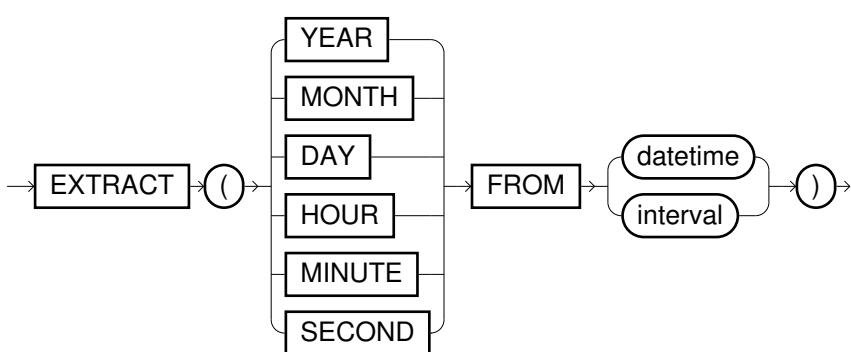
EXP(1)
-----
2.718281828459045
```

EXTRACT**Zweck**

Extrahiert bestimmte Werte aus einem Zeitstempel, Datum oder Intervall.

Syntax

extract ::=



Anmerkung(en)

- Erlaubte Parameter für die verschiedenen Datentypen:

DATE	YEAR, MONTH, DAY
TIMESTAMP	YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
TIMESTAMP WITH LOCAL TIME ZONE	YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
INTERVAL YEAR TO MONTH	YEAR, MONTH
INTERVAL DAY TO SECOND	DAY, HOUR, MINUTE, SECOND

- Bei der Extrahierung von Sekunden werden die im Zeitstempel bzw. Intervall enthaltenen Millisekunden mit-extrahiert.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT EXTRACT(SECOND FROM TIMESTAMP '2000-10-01 12:22:59.123') EXS,
       EXTRACT(MONTH FROM DATE '2000-10-01') EXM,
       EXTRACT(HOUR FROM INTERVAL '1 23:59:30.123' DAY TO SECOND) EXH;

EXS      EXM EXH
-----  ---
59.123  10   23
```

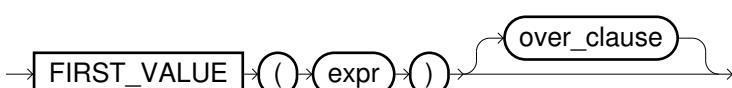
FIRST_VALUE

Zweck

Liefert die erste Zeile in einer Gruppe.

Syntax

first_value ::=



Anmerkung(en)

- Da die Zeilen in EXASOL im Cluster verteilt sind, ist FIRST_VALUE als Aggregatsfunktion nicht deterministisch. Somit dient FIRST_VALUE in erster Linie als Hilfsfunktion, falls innerhalb einer Gruppe sowieso nur gleiche Elemente existieren.
- Bei Verwendung als analytische Funktion (siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)) gilt dasselbe, wenn die OVER - Klausel keinen ORDER BY -Teil enthält.

Beispiel(e)

```
SELECT name,
       hire_date,
       FIRST_VALUE(hire_date) OVER (ORDER BY hire_date) FIRST_VAL
```

```
FROM staff;

NAME      HIRE_DATE      FIRST_VAL
-----
mueller   '2013-05-01'  '2013-05-01'
schmidt   '2013-08-01'  '2013-05-01'
meier     '2013-10-01'  '2013-05-01'
```

FLOOR

Zweck

Liefert die größte ganze Zahl, die kleiner oder gleich n.

Syntax

floor ::=



Beispiel(e)

```
SELECT FLOOR(4.567) FLOOR;
```

```
FLOOR
```

```
-----
```

```
4
```

FROM_POSIX_TIME

Zweck

Die Posix Time (oder auch Unix Time) ist im POSIX Standard festgelegt und liefert die Anzahl der Sekunden seit dem 1. Januar 1970 um 0 Uhr zurück (UTC). Mit dieser Funktion lässt sich aus der Posix Time (also einem numerischen Wert) ein Zeitstempel berechnen.

Syntax

from_posix_time ::=



Anmerkung(en)

- `FROM_POSIX_TIME(<number>)` ist äquivalent zum Funktionsaufruf `ADD_SECONDS('1970-01-01 00:00:00', <number>)`, sofern die Session-Zeitzone auf UTC gesetzt ist.
- Bei negativen Werten liefert die Funktion Zeitpunkte vor dem 1. Januar 1970 (UTC).
- Mit der Funktion `POSIX_TIME` lässt sich die Posix Time berechnen, also ein Zeitstempel in einen numerischen Wert umwandeln.

Beispiel(e)

```
ALTER SESSION SET TIME_ZONE='UTC';SELECT FROM_POSIX_TIME(1) FPT1,  
      FROM_POSIX_TIME(1234567890) FPT2;
```

FPT1 FPT2

1970-01-01 00:00:01.000000 2009-02-13 23:31:30.000000

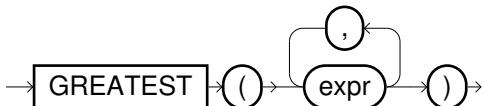
GREATEST

Zweck

Liefert den größten der angegebenen Ausdrücke.

Syntax

greatest::=



Anmerkung(en)

- Der Datentyp BOOLEAN wird nicht unterstützt.

Beispiel(e)

```
SELECT GREATEST(1,5,3) GREATEST;
```

GREATEST

5

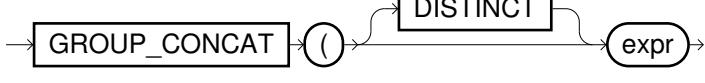
GROUP CONCAT

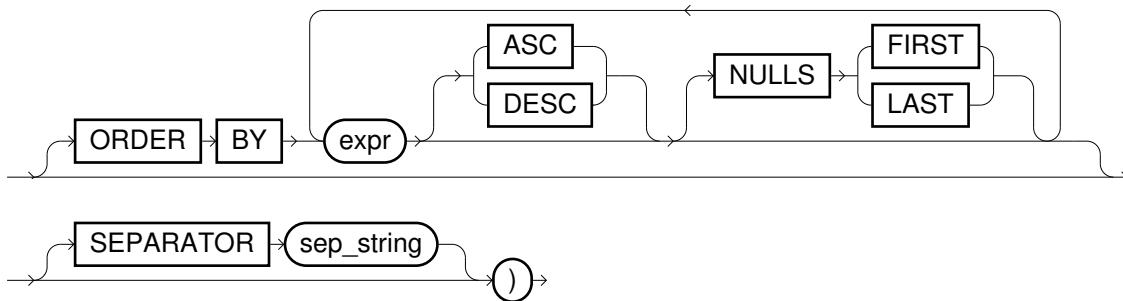
Zweck

Konkateniert die Werte von concat Expr jeweils für alle Elemente in einer Gruppe.

Syntax

group_concat:=





Anmerkung(en)

- Falls DISTINCT angegeben wird, so werden identische Zeichenketten eliminiert, sofern sie im Ergebnis aufeinander folgen würden.
- Bei Angabe von ORDER BY werden die Zeilen innerhalb einer Gruppe sortiert, bevor die Werte konkatenernt werden. Die Defaultsortierung benutzt die Optionen ASC NULLS LAST (also aufsteigend, mit NULL-Werten am Ende).
- Falls die ORDER BY Option nicht definiert wird, so wird bei Angabe von DISTINCT implizit nach der concat_expr sortiert. Dies stellt sicher, dass das Ergebnis deterministisch ist.
- Bei Angabe von SEPARATOR definiert der sep_string das Trennzeichen zwischen den konkatenierten Elementen. Im Defaultfall ist das Trennzeichen ', '. Mittels leerem String ('') kann es auch ganz weggelassen werden.
- Als Ergebnisdatentyp wird stets der maximale Zeichenkettentyp erzeugt (VARCHAR(2000000)).

Beispiel(e)

```

SELECT department , GROUP_CONCAT(name ORDER BY name)
FROM staff GROUP BY department;

DEPARTMENT      GROUP_CONCAT
-----  -----
sales            carsten,joe,thomas
marketing        alex,monica
  
```

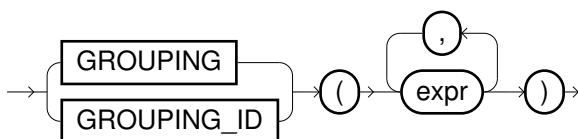
GROUPING[_ID]

Zweck

Diese Funktion ermöglicht es, zwischen regulären Ergebniszellen und Superaggregations-Zeilen unterscheiden zu können, welche bei der Verwendung von GROUPING SETS, CUBE bzw. ROLLUP erzeugt werden.

Syntax

grouping::=



Anmerkung(en)

- Alle Argumente der Funktion müssen jeweils mit einem Ausdruck der GROUP BY Klausel übereinstimmen.
- Bei einem Argument ist der Rückgabewert 0, falls in der entsprechenden Gruppierung der Ausdruck berücksichtigt ist, und ansonsten 1 (Superaggregation).
- Bei mehreren Argumenten ist der Rückgabewert eine Zahl, deren Binärdarstellung mit GROUPING(arg₁),GROUPING(arg₂),..., GROUPING(arg_n) übereinstimmt. Zum Beispiel gilt:

$$\text{GROUPING}(a,b,c) = 4 \times \text{GROUPING}(a) + 2 \times \text{GROUPING}(b) + 1 \times \text{GROUPING}(c)$$

- Details zu GROUPING SETS, CUBE und ROLLUP finden Sie in den Anmerkungen zum Befehl **SELECT** in [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Beispiel(e)

```
SELECT SUM(volume) revenue, y, m,
       DECODE(GROUPING(y,m),1,'yearly',3,'total',NULL) superaggregate
    FROM sales GROUP BY ROLLUP(y,m) ORDER BY y,revenue;
```

REVENUE	Y	M	SUPERAGGREGATE
1725.90	2010	December	
1725.90	2010		yearly
735.88	2011	April	
752.46	2011	February	
842.32	2011	March	
931.18	2011	January	
3261.84	2011		yearly
4987.74			total

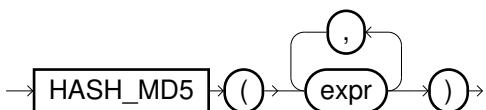
HASH_MD5

Zweck

Berechnet einen Hashwert. Hierfür wird der MD5-Algorithmus verwendet (128 Bit).

Syntax

hash_md5 ::=



Anmerkung(en)

- Der Rückgabewert ist stets vom Typ CHAR(32) und enthält Hexadezimal-Werte.
- Der Datentyp der Eingabeparameter ist signifikant. Daher ist HASH_MD5(123) unterschiedlich zu HASH_MD5('123').
- Bei mehreren Eingabeparametern werden die Parameter hintereinander kopiert (in ihren internen Byte-Representationen) und dann der Hashwert berechnet. Bitte beachten Sie, dass daher in der Regel HASH_MD5(c1, c2) nicht identisch ist mit HASH_MD5(c1 || c2).
- Falls alle Eingabeparameter den Wert NULL besitzen, so ist der Rückgabewert ebenfalls NULL.

Beispiel(e)

```
SELECT HASH_MD5( 'abc' ) ;

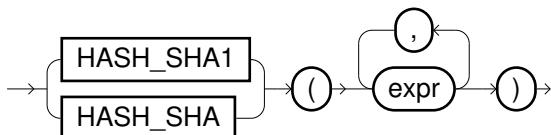
HASH_MD5( 'abc' )
-----
900150983cd24fb0d6963f7d28e17f72
```

HASH_SHA[1]**Zweck**

Berechnet einen Hashwert. Hierfür wird der SHA1-Algorithmus verwendet (160 Bit).

Syntax

hash_sha1 ::=

**Anmerkung(en)**

- Der Rückgabewert ist stets vom Typ CHAR (40) und enthält Hexadezimal-Werte.
- Der Datentyp der Eingabeparameter ist signifikant. Daher ist HASH_SHA1(123) unterschiedlich zu HASH_SHA1('123').
- Bei mehreren Eingabeparametern werden die Parameter hintereinander kopiert (in ihren internen Byte-Representationen) und dann der Hashwert berechnet. Bitte beachten Sie, dass daher in der Regel HASH_SHA1(c1,c2) nicht identisch ist mit HASH_SHA1(c1|c2).
- Falls alle Eingabeparameter den Wert NULL besitzen, so ist der Rückgabewert ebenfalls NULL.
- HASH_SHA() ist ein Alias für HASH_SHA1().

Beispiel(e)

```
SELECT HASH_SHA1( 'abc' ) ;

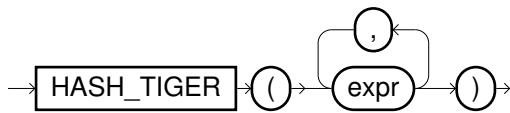
HASH_SHA1( 'abc' )
-----
a9993e364706816aba3e25717850c26c9cd0d89d
```

HASH_TIGER**Zweck**

Berechnet einen Hashwert. Hierfür wird der Tiger-Algorithmus verwendet (192 Bit).

Syntax

hash_tiger ::=

**Anmerkung(en)**

- Der Rückgabewert ist stets vom Typ CHAR(48) und enthält Hexadezimal-Werte.
- Der Datentyp der Eingabeparameter ist signifikant. Daher ist HASH_TIGER(123) unterschiedlich zu HASH_TIGER('123').
- Bei mehreren Eingabeparametern werden die Parameter hintereinander kopiert (in ihren internen Byte-Representationen) und dann der Hashwert berechnet. Bitte beachten Sie, dass daher in der Regel HASH_TIGER(c1,c2) nicht identisch ist mit HASH_TIGER(c1||c2).
- Falls alle Eingabeparameter den Wert NULL besitzen, so ist der Rückgabewert ebenfalls NULL.

Beispiel(e)

```

SELECT HASH_TIGER('abc');

HASH_TIGER('abc')
-----
2aab1484e8c158f2bfb8c5ff41b57a525129131c957b5f93
  
```

HOUR**Zweck**

Liefert die Stunden eines Zeitstempels.

Syntax

hour ::=

**Anmerkung(en)**

- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```

SELECT HOUR(TIMESTAMP '2010-10-20 11:59:40.123');

HOU
---
11
  
```

HOURS_BETWEEN

Zweck

Liefert die Anzahl der Stunden zwischen dem Zeitstempel `timestamp1` und dem Zeitstempel `timestamp2`.

Syntax

`hours_between`::=

→ **HOURS_BETWEEN** → (→ `datetime1` → , → `datetime2` →) →

Anmerkung(en)

- Falls der Zeitstempel `timestamp1` vor dem Zeitstempel `timestamp2` liegt, so ist das Ergebnis negativ.
- Für den Datentyp `TIMESTAMP WITH LOCAL TIME ZONE` wird diese Funktion intern in UTC berechnet.

Beispiel(e)

```
SELECT HOURS_BETWEEN(TIMESTAMP '2000-01-01 12:00:00',
                      TIMESTAMP '2000-01-01 11:01:05.1') HB;
```

HB

0.981916666667

INSERT

Zweck

Ersetzt in der Zeichenkette `string` den Teilstring ab Position `position` mit Länge `length` durch die Zeichenkette `new_string`.

Syntax

`insert`::=

→ **INSERT** → (→ `string` → , → `position` → , → `length` → , → `new_string` →) →

Anmerkung(en)

- Das erste Zeichen hat die Position 1. Falls die Variable `position` 0 ist oder außerhalb der Zeichenkette liegt, so wird diese nicht verändert. Falls die Position negativ ist, so zählt die Funktion rückwärts vom Ende der Zeichenkette.
- Falls `length=0`, so wird `new_string` nur eingefügt und nichts ersetzt.
- Falls `position+length>length(string)` oder falls `length<0`, so wird der Rest der Zeichenkette `string` ab der Position `position` ersetzt.
- Falls einer der Parameter `NULL` ist, so wird `NULL` zurückgeliefert.

Beispiel(e)

```
SELECT INSERT( 'abc' ,2 ,2 , 'xxx' ) ,
       INSERT( 'abcdef' ,3 ,2 , 'CD' ) ;

INSERT( 'abc' ,2 ,2 , 'xxx' )  INSERT( 'abcdef' ,3 ,2 , 'CD' )
-----
axxx                      abCDef
```

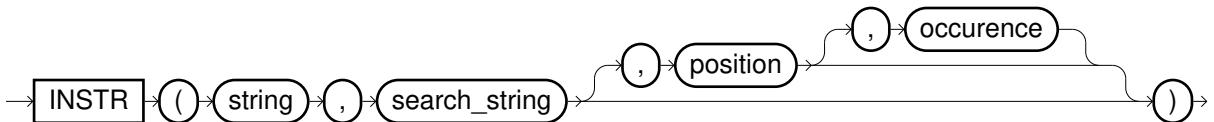
INSTR

Zweck

Liefert die Position in der Zeichenkette `string` zurück, an der die Zeichenkette `search_string` vorkommt. Falls diese gar nicht enthalten ist, so wird der Wert 0 zurückgeliefert.

Syntax

`instr ::=`



Anmerkung(en)

- Der optionale Parameter `position` definiert die Position, ab der gesucht werden soll (das erste Zeichen hat Position 1). Falls ein negativer Wert eingegeben wird, so wird von hinten gezählt *und* rückwärts gesucht (`INSTR(string, 'abc', -3)` sucht also ab dem drittletzten Buchstaben von hinten nach vorne).
- Die optionale positive Zahl `occurrence` definiert, nach welchem Auftreten des Suchworts gesucht wird.
- `INSTR(string,search_string)` entspricht dem Aufruf `INSTR(string,search_string,1,1)`.
- Die Funktion `POSITION` ist ähnlich zu dieser Funktion.

Beispiel(e)

```
SELECT INSTR('abcababcabc','cab') INSTR1,
       INSTR('user1,user2,user3,user4,user5','user', -1, 2) INSTR2;

INSTR1      INSTR2
-----
3           19
```

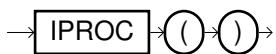
IPROC

Zweck

Liefert die lokale Knotennummer im Cluster zurück. Hiermit kann visualisiert werden, welche Zeilen auf welchen Knoten gespeichert werden.

Syntax

iproc::=



Anmerkung(en)

- Der Ergebniswert ist eine Ganzzahl zwischen 0 und [NPROC](#)-1.
- Reserveknoten werden nicht mit eingerechnet, sondern nur aktive Datenbank-Knoten.
- Bitte beachten Sie in diesem Zusammenhang auch die Funktionen [NPROC](#) und [VALUE2PROC](#).

Beispiel(e)

```
SELECT c1, IPROC() IPROC FROM t ORDER BY c1;

C1 IPROC
-- -----
1 0
2 1
3 2
4 3
5 0
6 1

SELECT IPROC() IPROC, COUNT(*) CNT FROM t GROUP BY 1;

IPROC CNT
----- ---
0      2
1      2
2      1
3      1
```

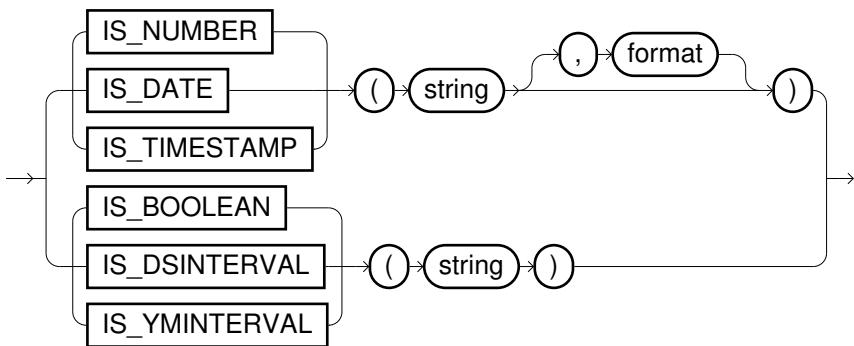
IS_*

Zweck

Liefert den Wert TRUE, falls string in den entsprechenden Datentyp konvertiert werden kann. Falls z.B. ein [IS_NUMBER](#) TRUE liefert, ist ein entsprechendes [TO_NUMBER](#) möglich.

Syntax

is_datatype::=



Anmerkung(en)

- Ist ein Argument NULL, so wird NULL zurückgegeben.
- Wird ein Format angegeben, so gelten die entsprechenden Regeln wie bei den entsprechenden TO-Funktionen ([TO_NUMBER](#), [TO_DATE](#), [TO_TIMESTAMP](#), [TO_DSINTERVAL](#), [TO_YMINTERVAL](#)). Siehe hierzu auch [Abschnitt 2.6.2, Numerische Format-Modelle](#).

Beispiel(e)

```

SELECT IS_BOOLEAN('xyz') IS_BOOLEAN,
       IS_NUMBER('+12.34') IS_NUMBER,
       IS_DATE('12.13.2011', 'DD.MM.YYYY') IS_DATE;

IS_BOOLEAN IS_NUMBER IS_DATE
----- -----
FALSE      TRUE      FALSE
  
```

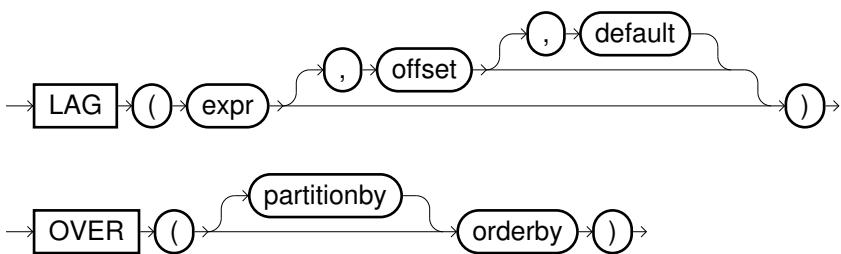
LAG

Zweck

LAG bietet die Möglichkeit, in einer analytischen Funktion auf vorherige Zeilen innerhalb einer Partition zugreifen zu können. Es wird der Ausdruck `expr` auf derjenigen Zeile berechnet, die genau die durch `offset` angegebene Anzahl Zeilen vor der aktuellen Zeile liegt.

Syntax

`lag`::=



Anmerkung(en)

- LAG kann nur als analytische Funktion verwendet werden (in Verbindung mit OVER(...), siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)).
- Die OVER-Klausel muss einen ORDER BY-Teil und darf keine Window-Klausel enthalten.
- Stellt der ORDER BY-Teil keine eindeutige Sortierung sicher, ist das Ergebnis nicht deterministisch.
- Für Zeilen außerhalb der aktuellen Partition liefert LAG den Wert default bzw. NULL, wenn default nicht angegeben ist.
- Wird offset nicht angegeben, so wird der Wert 1 verwendet. Negative Werte sind im offset nicht erlaubt. Zeilen mit offset NULL werden wie Zeilen außerhalb der aktuellen Partition behandelt.
- Für Zugriff auf nachfolgende Zeilen kann die Funktion [LEAD](#) verwendet werden.

Beispiel(e)

```
SELECT airplane, maintenance_date, last_maintenance,
       DAYS_BETWEEN(maintenance_date, last_maintenance) unmaintained_time
FROM
(
    SELECT airplane, maintenance_date,
           LAG(maintenance_date,1)
              OVER (PARTITION BY airplane
                     ORDER BY maintenance_date) last_maintenance
    FROM maintenance
    WHERE TRUNC(maintenance_date,'MM')='2008-06-01'
)
ORDER BY airplane, maintenance_date;

AIRPLANE          MAINTENANC LAST_MAINT UNMAINTAIN
-----          -----
Banana airlines - jet 1 2008-06-03
Banana airlines - jet 1 2008-06-29 2008-06-03      26
Safe travel - jet 1   2008-06-02
Safe travel - jet 1   2008-06-09 2008-06-02      7
Safe travel - jet 1   2008-06-16 2008-06-09      7
Safe travel - jet 1   2008-06-23 2008-06-16      7
Safe travel - jet 1   2008-06-30 2008-06-23      7
```

LAST_VALUE

Zweck

Liefert die letzte Zeile in einer Gruppe.

Syntax

last_value::=



Anmerkung(en)

- Da die Zeilen in EXASOL im Cluster verteilt sind, ist LAST_VALUE als Aggregatsfunktion nicht deterministisch. Somit dient LAST_VALUE in erster Linie als Hilfsfunktion, falls innerhalb einer Gruppe sowieso nur gleiche Elemente existieren.
- Bei Verwendung als analytische Funktion (siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)) gilt dasselbe, wenn die OVER - Klausel keinen ORDER BY -Teil enthält.

Beispiel(e)

```
SELECT name,
       hire_date,
       LAST_VALUE(hire_date) OVER (ORDER BY hire_date
                                     ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) LAST_VAL
FROM staff;

NAME      HIRE_DATE      LAST_VAL
-----  -----
mueller   '2013-05-01'  '2013-10-01'
schmidt   '2013-08-01'  '2013-10-01'
meier     '2013-10-01'  '2013-10-01'
```

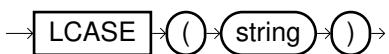
LCASE

Zweck

Wandelt die angegebene Zeichenkette in Kleinbuchstaben um.

Syntax

lcase ::=



Anmerkung(en)

- LCASE ist ein Alias für [LOWER](#).

Beispiel(e)

```
SELECT LCASE( 'AbCdEf' ) LCASE;

LCASE
-----
abcdef
```

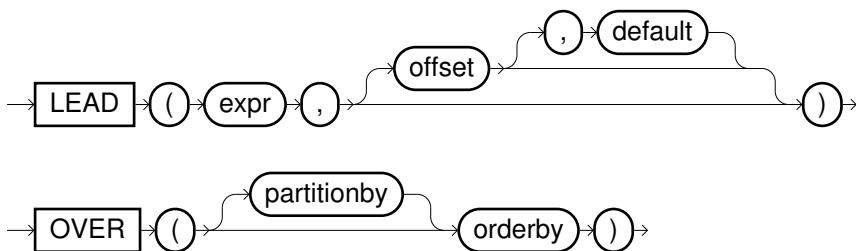
LEAD

Zweck

LEAD bietet die Möglichkeit, in einer analytischen Funktion auf nachfolgende Zeilen innerhalb einer Partition zugreifen zu können. Es wird der Ausdruck `expr` auf derjenigen Zeile berechnet, die genau die durch `offset` angegebene Anzahl Zeilen hinter der aktuellen Zeile liegt.

Syntax

`lead`::=



Anmerkung(en)

- LEAD kann nur als analytische Funktion verwendet werden (in Verbindung mit `OVER(...)`, siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)).
- Die `OVER`-Klausel muss einen `ORDER BY`-Teil und darf keine Window-Klausel enthalten.
- Stellt der `ORDER BY`-Teil keine eindeutige Sortierung sicher, ist das Ergebnis nicht deterministisch.
- Für Zeilen außerhalb der aktuellen Partition liefert LEAD den Wert `default` bzw. `NULL`, wenn `default` nicht angegeben ist.
- Wird `offset` nicht angegeben, so wird der Wert 1 verwendet. Negative Werte sind im `offset` nicht erlaubt. Zeilen mit `offset NULL` werden wie Zeilen außerhalb der aktuellen Partition behandelt.
- Für Zugriff auf vorherige Zeilen kann die Funktion [LAG](#) verwendet werden.

Beispiel(e)

```

SELECT company, fiscal_year, market_value, expected_growth,
       CAST( (next_market_value - market_value) / market_value
             AS DECIMAL(5,2) ) real_growth,
       next_market_value - market_value net_increase
FROM
  ( SELECT company, fiscal_year, market_value, expected_growth,
          LEAD( market_value, 1 )
          OVER ( PARTITION BY company
                  ORDER BY fiscal_year ) next_market_value
    FROM yearly_market_value )
ORDER BY company, fiscal_year;
    
```

COMPANY	FISCAL_YEAR	MARKET_VALUE	EXPECTE	REAL_GR	NET_INCREASE
Bankers R US	2004	20.00	0.15	0.25	5.00
Bankers R US	2005	25.00	0.25	0.20	5.00
Bankers R US	2006	30.00	0.17	0.33	10.00
Bankers R US	2007	40.00	0.22	-0.55	-22.00
Bankers R US	2008	18.00	-0.25		
Service Inc.	2004	102.00	0.05	0.08	8.00

Service Inc.	2005	110.00	0.07	0.09	10.00
Service Inc.	2006	120.00	0.05	0.04	5.00
Service Inc.	2007	125.00	0.03	-0.02	-3.00
Service Inc.	2008	122.00	-0.02		

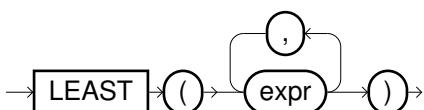
LEAST

Zweck

Liefert den kleinsten der angegebenen Ausdrücke.

Syntax

least::=



Anmerkung(en)

- Der Datentyp BOOLEAN wird nicht unterstützt.

Beispiel(e)

```

SELECT LEAST(3,1,5) LEAST;
LEAST
-----
1
  
```

LEFT

Zweck

Liefert den linksbündigen Teil-String der Zeichenkette `string` mit Länge `length`.

Syntax

left::=



Anmerkung(en)

- Falls `length` oder `string` NULL sind, so wird NULL zurückgeliefert.
- Auch wenn `length` größer ist als die Länge von `string`, so wird der Original-String zurückgeliefert. Falls `length` negativ oder 0 ist, so wird NULL zurückgeliefert.
- Siehe auch die Funktionen [SUBSTR\[ING\]](#), [MID](#) und [RIGHT](#).

Beispiel(e)

```
SELECT LEFT( 'abcdef' , 3 ) LEFT_SUBSTR;  
  
LEFT_SUBSTR  
-----  
abc
```

LENGTH

Zweck

Liefert die Länge einer Zeichenkette zurück (Anzahl an Zeichen).

Syntax

length::=

→ **LENGTH** → (→ string →) →

Beispiel(e)

```
SELECT LENGTH( 'abc' ) LENGTH;  
  
LENGTH  
-----  
3
```

LEVEL

Zweck

Liefert zu einer CONNECT BY Anfrage das Level eines Knotens im Baum. Details finden Sie in der Dokumentation zum [SELECT Befehl in Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Syntax

level::=

→ **LEVEL** →

Beispiel(e)

```
SELECT last_name,  
       LEVEL,  
       SYS_CONNECT_BY_PATH(last_name, '/') "PATH"  
  FROM employees  
CONNECT BY PRIOR employee_id = manager_id  
START WITH last_name = 'Clark';
```

```
LAST_NAME LEVEL PATH
-----
Clark      1 /Clark
Smith      2 /Clark/Smith
Brown      3 /Clark/Smith/Brown
```

LN

Zweck

Liefert den natürlichen Logarithmus einer Zahl n zurück. Die Funktion LN(n) ist äquivalent zu LOG(EXP(1),n).

Syntax

ln ::=

→ **[LN]** → (→ n →) →

Anmerkung(en)

- Die angegebene Zahl n muss größer als 0 sein.

Beispiel(e)

```
SELECT LN(100) LN;
LN
-----
4.6051701859881
```

LOCALTIMESTAMP

Zweck

Liefert den aktuellen Zeitstempel zurück, interpretiert in der aktuellen Session-Zeitzone.

Syntax

localtimestamp ::=

→ **LOCALTIMESTAMP** →

Anmerkung(en)

- Der Rückgabewert dieser Funktion ist vom Typ TIMESTAMP.
- Weitere Funktionen zum aktuellen Zeitpunkt:
 - CURRENT_TIMESTAMP bzw. NOW
 - SYSTIMESTAMP

Beispiel(e)

```
SELECT LOCALTIMESTAMP;

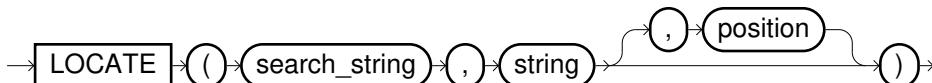
LOCALTIMESTAMP
-----
2000-12-31 23:59:59.000
```

LOCATE**Zweck**

Liefert die Position in der Zeichenkette `string` zurück, an der die Zeichenkette `search_string` vorkommt. Falls diese gar nicht enthalten ist, so wird der Wert 0 zurückgeliefert.

Syntax

`locate ::=`

**Anmerkung(en)**

- Der optionale Parameter `position` definiert die Position, ab der gesucht werden soll (beginnend mit 1). Falls ein negativer Wert eingegeben wird, so wird von hinten gezählt *und* rückwärts gesucht (`LOCATE('abc', string, -3)` sucht also ab dem drittletzten Buchstaben von hinten nach vorne).
- `LOCATE(search_string, string)` entspricht dem Aufruf `LOCATE(search_string, string, 1)`.
- Die Funktionen `POSITION` und `INSTR` sind ähnlich zu dieser Funktion.

Beispiel(e)

```
SELECT LOCATE('cab', 'abcababcabc') LOCATE1,
       LOCATE('user', 'user1,user2,user3,user4,user5', -1) LOCATE2;

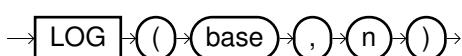
LOCATE1      LOCATE2
-----      -----
            3          25
```

LOG**Zweck**

Liefert den Logarithmus von `n` zur Basis `base`.

Syntax

`log ::=`



Anmerkung(en)

- Die Zahl base muss positiv sein und darf nicht 1 sein.
- Die Zahl n muss positiv sein.

Beispiel(e)

```
SELECT LOG(2,1024);  
  
LOG(2,1024)  
-----  
          10
```

LOG10

Zweck

Liefert den Logarithmus von n zur Basis 10.

Syntax

$\text{log10} ::=$

$\rightarrow \boxed{\text{LOG10}} \rightarrow (\circlearrowleft \rightarrow \circlearrowright n \circlearrowright) \rightarrow$

Anmerkung(en)

- Die Zahl n muss positiv sein.

Beispiel(e)

```
SELECT LOG10(10000) LOG10;  
  
LOG10  
-----  
          4
```

LOG2

Zweck

Liefert den Logarithmus von n zur Basis 2.

Syntax

$\text{log2} ::=$

$\rightarrow \boxed{\text{LOG2}} \rightarrow (\circlearrowleft \rightarrow \circlearrowright n \circlearrowright) \rightarrow$

Anmerkung(en)

- Die Zahl n muss positiv sein.

Beispiel(e)

```
SELECT LOG2(1024) LOG2;  
  
LOG2  
-----  
          10
```

LOWER

Zweck

Wandelt die angegebene Zeichenkette in Kleinbuchstaben um.

Syntax

lower ::=



Anmerkung(en)

- LOWER ist ein Alias für [LCASE](#).

Beispiel(e)

```
SELECT LOWER( 'AbCdEf' );  
  
LOWER( 'AbCdEf' )  
-----  
abcdef
```

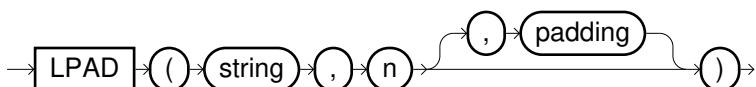
LPAD

Zweck

Liefert eine Zeichenkette der Länge n zurück, die ausgehend von der Zeichenkette string von der linken Seite her mit dem Ausdruck padding aufgefüllt wird.

Syntax

lpad ::=



Anmerkung(en)

- Falls `padding` nicht angegeben ist, so wird mit Leerzeichen aufgefüllt.
- Auch falls `n` größer als 2.000.000 ist, wird das Ergebnis auf 2.000.000 Zeichen begrenzt.
- Um eine Zeichenkette von der rechten Seite her aufzufüllen, können Sie die Funktion `RPAD` verwenden.

Beispiel(e)

```
SELECT LPAD( 'abc' , 5 , 'X' ) ;  
  
LPAD( 'abc' , 5 , 'X' )  
-----  
XXabc
```

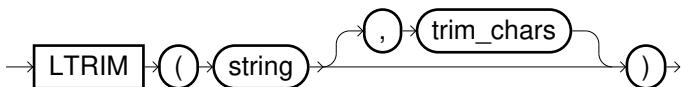
LTRIM

Zweck

`LTRIM` entfernt von der linken Grenze der Zeichenkette `string` alle im Ausdruck `trim_chars` angegebenen Zeichen.

Syntax

`ltrim`::=



Anmerkung(en)

- Falls der Parameter `trim_chars` nicht angegeben ist, werden Leerzeichen entfernt.

Beispiel(e)

```
SELECT LTRIM( 'ab cdef' , ' ab' ) ;  
  
LTRIM( 'ab cdef' , ' ab' )  
-----  
cdef
```

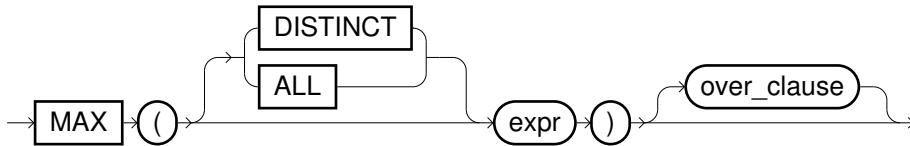
MAX

Zweck

Liefert das Maximum.

Syntax

`max`::=



Anmerkung(en)

- Für das Ergebnis ist es irrelevant, ob ALL oder DISTINCT angegeben ist.

Beispiel(e)

```

SELECT MAX(age) MAX FROM staff;

MAX
-----
57
  
```

MEDIAN

Zweck

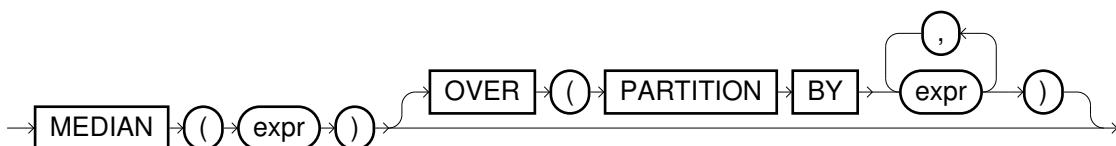
MEDIAN ist eine inverse Verteilungsfunktion. Im Gegensatz zum Durchschnitt (siehe [AVG](#)) liefert der Median den mittleren Eintrag bzw. einen interpolierten Wert, welcher bei Sortierung der Gruppenelementen genau in der Mitte stehen würde (NULL-Werte werden ignoriert).

Die folgende Formel wird hierzu ausgewertet:

$$\text{MEDIAN(expr)} = \begin{cases} \text{expr}_{(n+1)/2} & \text{für } n \text{ ungerade} \\ \frac{\text{expr}_{n/2} + \text{expr}_{n/2+1}}{2} & \text{für } n \text{ gerade} \end{cases}$$

Syntax

median ::=



Anmerkung(en)

- `MEDIAN(<expr>)` ist ein Alias für `PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY <expr>)`.
- Siehe auch die inversen Verteilungsfunktionen [PERCENTILE_CONT](#) und [PERCENTILE_DISC](#).

Beispiel(e)

```

COUNT
-----
  
```

```

50
100
200
900

SELECT MEDIAN(count) MEDIAN FROM sales;

MEDIAN
-----
150

```

MID

Zweck

Liefert einen Teilstring der Länge `length` ab der Position `position` aus der Zeichenkette `string`.

Syntax

`mid ::=`



Anmerkung(en)

- Falls `length` nicht angegeben ist, so werden alle Zeichen bis zum Ende der Zeichenkette verwendet.
- Das erste Zeichen der Zeichenkette hat Position 1. Falls der Parameter `position` negativ ist, so wird vom Ende der Zeichenkette gezählt.
- Siehe auch die Funktionen [LEFT](#) und [RIGHT](#).
- `MID` ist ein Alias für die Funktion [SUBSTR\[ING\]](#).

Beispiel(e)

```

SELECT MID('abcdef', 2, 3) MID;

MID
---
bcd

```

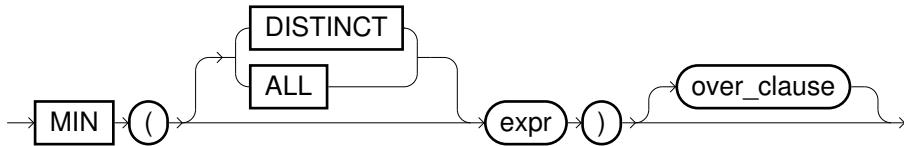
MIN

Zweck

Liefert das Minimum.

Syntax

`min ::=`



Anmerkung(en)

- Für das Ergebnis ist es irrelevant, ob ALL oder DISTINCT angegeben ist.

Beispiel(e)

```

SELECT MIN(age) MIN FROM staff;

MIN
-----
25
  
```

MINUTE

Zweck

Liefert die Minuten eines Zeitstempels.

Syntax

minute ::=



Anmerkung(en)

- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```

SELECT MINUTE(TIMESTAMP '2010-10-20 11:59:40.123');

MIN
---
59
  
```

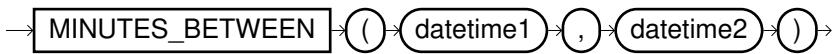
MINUTES_BETWEEN

Zweck

Liefert die Anzahl der Minuten zwischen zwei Zeitstempeln `timestamp1` und `timestamp2`.

Syntax

minutes_between::=



Anmerkung(en)

- Falls der Zeitstempel timestamp1 vor dem Zeitstempel timestamp2 liegt, so ist das Ergebnis negativ.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion intern in UTC berechnet.

Beispiel(e)

```
SELECT MINUTES_BETWEEN(TIMESTAMP '2000-01-01 12:01:00',
                      TIMESTAMP '2000-01-01 12:00:02') MINUTES;
MINUTES
-----
0.96666666666667
```

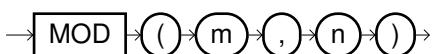
MOD

Zweck

Liefert den Rest bei der Division von m durch n.

Syntax

mod::=



Beispiel(e)

```
SELECT MOD(15,6) MODULO;
MODULO
-----
3
```

MONTH

Zweck

Liefert den Monat eines Datums.

Syntax

month::=



Anmerkung(en)

- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT MONTH(DATE '2010-10-20');

MON
---
10
```

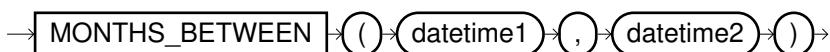
MONTHS_BETWEEN

Zweck

Liefert die Anzahl an Monaten zwischen zwei Datumswerten.

Syntax

months_between::=



Anmerkung(en)

- Bei der Eingabe eines Zeitstempels geht nur das in ihm enthaltene Datum in die Berechnung ein.
- Falls die Tage identisch sind oder beide die letzten Tage im Monat sind, so ist das Ergebnis eine ganze Zahl.
- Falls der erste Datumswert vor dem zweiten Datumswert liegt, so ist das Ergebnis negativ.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT MONTHS_BETWEEN(DATE '2000-01-01', DATE '1999-12-15') MB1,
       MONTHS_BETWEEN(TIMESTAMP '2000-01-01 12:00:00',
                      TIMESTAMP '1999-06-01 00:00:00') MB2;

MB1          MB2
-----
0.548387096774194    7
```

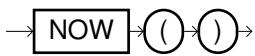
NOW

Zweck

Liefert den aktuellen Zeitstempel zurück, interpretiert in der aktuellen Session-Zeitzone.

Syntax

now::=



Anmerkung(en)

- Der Rückgabewert dieser Funktion ist vom Typ TIMESTAMP.



Bitte beachten Sie, dass sich der Ergebnistyp zur nächsten Major-Version von TIMESTAMP zu TIMESTAMP WITH LOCAL TIME ZONE ändern wird. Wir haben diese Änderung zum jetzigen Zeitpunkt vermieden, um jegliche Auswirkungen auf existierende Prozesse zu vermeiden. Die Änderung wird sich auch dann in Grenzen halten, da inhaltlich die Werte gleich bleiben werden (aktueller Zeitstempel interpretiert in der Session-Zeitzone) und sich nur der Datentyp ändern wird.

- NOW ist ein Alias für [CURRENT_TIMESTAMP](#).
- Weitere Funktionen zum aktuellen Zeitpunkt:
 - [LOCALTIMESTAMP](#)
 - [SYSTIMESTAMP](#)

Beispiel(e)

```
SELECT NOW( ) NOW;  
  
NOW  
-----  
1999-12-31 23:59:59
```

NPROC

Zweck

Liefert die Anzahl an Datenbank-Knoten im Cluster zurück.

Syntax

nproc::=



Anmerkung(en)

- Reserveknoten werden nicht mit eingerechnet, sondern nur aktiven Datenbank-Knoten.

- Bitte beachten Sie in diesem Zusammenhang auch die Funktionen [IPROC](#) und [VALUE2PROC](#).

Beispiel(e)

```
SELECT NPROC() NPROC;
```

```
NPROC  
-----  
4
```

NULLIF

Zweck

Liefert den Wert NULL, falls die beiden Ausdrücke `expr1` und `expr2` identisch sind. Andernfalls wird der Ausdruck `expr1` zurückgeliefert.

Syntax

`nullif`::=

→ **NULLIF** → (→ **expr1** → , → **expr2** →) →

Anmerkung(en)

- Die Funktion `NULLIF` ist äquivalent zum CASE-Ausdruck `CASE WHEN expr1=expr2 THEN NULL ELSE expr1 END`

Beispiel(e)

```
SELECT NULLIF(1,2) NULLIF1, NULLIF(1,1) NULLIF2;
```

```
NULLIF1 NULLIF2  
----- -----  
1
```

NULLIFZERO

Zweck

Liefert den Wert `NULL`, falls die Zahl `number` den Wert 0 hat. Andernfalls wird die Zahl `number` zurückgeliefert.

Syntax

`nullifzero`::=

→ **NULLIFZERO** → (→ **number** →) →

Anmerkung(en)

- Die Funktion NULLIFZERO entspricht dem CASE-Ausdruck CASE WHEN number=0 THEN NULL ELSE number END.
- Siehe auch [ZEROIFNULL](#).

Beispiel(e)

```
SELECT NULLIFZERO(0) NIZ1, NULLIFZERO(1) NIZ2;  
  
NIZ1 NIZ2  
-----  
      1
```

NUMTODSINTERVAL

Zweck

Wandelt einen numerischen Wert n in ein Intervall um des Typs INTERVAL DAY TO SECOND.

Syntax

numtodsinterval ::=

→ **NUMTODSINTERVAL** → (→ n → , →) → interval_unit → (→) →) →

Anmerkung(en)

- Der Parameter interval_unit ist einer der Werte DAY, HOUR, MINUTE, SECOND.
- Siehe auch [NUMTOYMINTERVAL](#), [TO_DSINTERVAL](#) und [TO_YMINTERVAL](#).

Beispiel(e)

```
SELECT NUMTODSINTERVAL( 3.2, 'HOUR' ) NUMTODSINTERVAL;  
  
NUMTODSINTERVAL  
-----  
+0000000000 03:12:00.000000000
```

NUMTOYMINTERVAL

Zweck

Wandelt einen numerischen Wert n in ein Intervall um des Typs INTERVAL YEAR TO MONTH.

Syntax

numtoyminterval ::=

→ **NUMTOYMINTERVAL** → (→ n → , →) → interval_unit → (→) →) →

Anmerkung(en)

- Der Parameter `interval_unit` ist entweder `YEAR` oder `MONTH`.
- Siehe auch [NUMTODSINTERVAL](#), [TO_DSINTERVAL](#) und [TO_YMINTERVAL](#).

Beispiel(e)

```
SELECT NUMTOYMINTEGER(3.5, 'YEAR') NUMTOYMINTEGER;  
  
NUMTOYMINTEGER  
-----  
+00000003-06
```

NVL

Zweck

Ersetzt NULL-Werte durch den Ausdruck `expr2`.

Syntax

`nvl::=`

→ [NVL] → (→ expr1 → , → expr2 →) →

Anmerkung(en)

- Falls `expr1` NULL ist, so wird `expr2` zurückgegeben, andernfalls `expr1`.
- Die Abkürzung NVL steht für "Null Value".
- Siehe auch [ZEROIFNULL](#).

Beispiel(e)

```
SELECT NVL(NULL, 'abc') NVL_1, NVL('xyz', 'abc') NVL_2;  
  
NVL_1 NVL_2  
----- -----  
abc xyz
```

NVL2

Zweck

Ersetzt NULL-Werte durch den Ausdruck `expr3`. In allen anderen Fällen wird `expr2` verwendet.

Syntax

`nvl2::=`

→ [NVL2] → (→ expr1 → , → expr2 → , → expr3 →) →

Anmerkung(en)

- Falls `expr1` NULL ist, so wird `expr3` zurückgegeben, andernfalls `expr2`.
- Die Abkürzung NVL steht für "Null Value".
- Ist keine implizite Konvertierung der Typen von `expr2` und `expr3` möglich, wird ein Systemfehler ausgegeben.
- Siehe auch [NVL](#).

Beispiel(e)

```
SELECT NVL2(NULL, 2, 3) NVL_1,
       NVL2(1, 2, 3) NVL_2;

NVL_1  NVL_2
----- -----
3      2
```

OCTET_LENGTH

Zweck

Liefert die Oktettlänge einer Zeichenkette. Falls nur ASCII Zeichen benutzt werden, ist diese Funktion äquivalent zu [CHARACTER_LENGTH](#) und [LENGTH](#).

Syntax

`octet_length ::=`

→ **OCTET_LENGTH** → (→ **string** →) →

Beispiel(e)

```
SELECT OCTET_LENGTH('abcd') OCT_LENGTH;

OCT_LENGTH
-----
4

SELECT OCTET_LENGTH('äöü') OCT_LENGTH;

OCT_LENGTH
-----
6
```

PERCENTILE_CONT

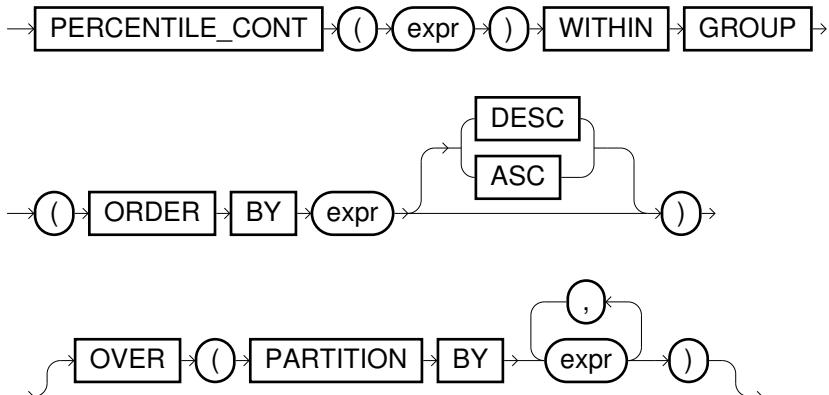
Zweck

`PERCENTILE_CONT` ist eine inverse Verteilungsfunktion und erwartet als Eingabeparameter ein Perzentil und eine Sortierungsspezifikation, welche die Rangfolge jedes Elements definiert. Die Funktion liefert das Perzentil dieser Rangfolge (z.B. bei Perzentil 0.7 und 100 Werten der 70. Wert).

Falls das Perzentil nicht exakt einer Zahl zuordnet werden kann, wird die lineare Interpolation zwischen den beiden nächsten Werten zurückgeliefert (z.B. bei Perzentil 0.71 und 10 Werten die Interpolation zwischen dem 7. und 8. Wert).

Syntax

percentile_cont ::=



Anmerkung(en)

- NULL-Werte werden bei der Berechnung ignoriert.
- Das angegebene Perzentil muss konstant sein (zwischen 0 und 1).
- Wird weder DESC noch ASC angegeben, so ist ASC der Default-Wert.
- Siehe auch die inversen Verteilungsfunktionen [PERCENTILE_DISC](#) und [MEDIAN](#).

Beispiel(e)

COUNT	REGION	COUNTRY
100	EUROPE	NETHERLANDS
500	EUROPE	UK
600	EUROPE	FRANCE
700	EUROPE	GERMANY
900	ASIA	CHINA
300	ASIA	KOREA
700	ASIA	JAPAN


```

SELECT region,
       PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY count),
       PERCENTILE_CONT(0.9) WITHIN GROUP (ORDER BY count)
  FROM sales
 GROUP BY region;
  
```


REGION	PERCENTILE_CONT(0.5)	PERCENTILE_CONT(0.9)
EUROPE	550	670
ASIA	700	860

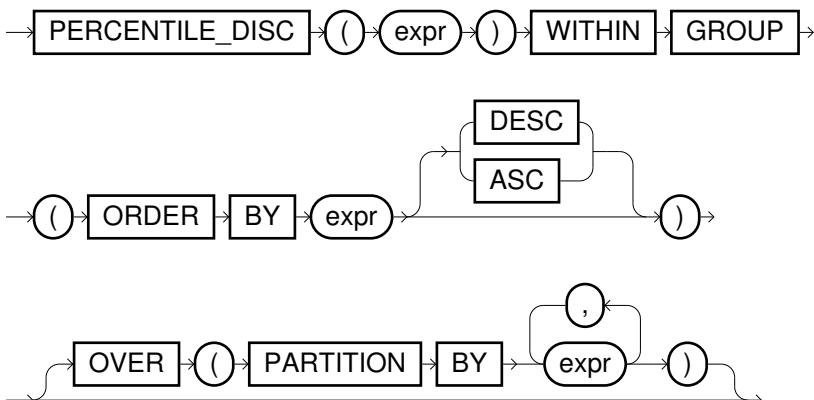
PERCENTILE_DISC

Zweck

PERCENTILE_DISC ist eine inverse Verteilungsfunktion und liefert einen Wert aus der Gruppe. Dieser ist der Wert mit dem kleinsten kumulativen Verteilungswert, der größer oder gleich dem angegebenen Perzentil ist (entsprechend der definierten Sortierung). NULL-Werte werden bei der Berechnung ignoriert.

Syntax

percentile_disc::=



Anmerkung(en)

- Das angegebene Perzentil muss konstant sein (zwischen 0 und 1).
- PERCENTILE_DISC(0) liefert stets den ersten Wert der Gruppe.
- Wird weder DESC noch ASC angegeben, so ist ASC der Default-Wert.
- Siehe auch die inversen Verteilungsfunktionen [PERCENTILE_CONT](#) und [MEDIAN](#).

Beispiel(e)

COUNT	REGION	COUNTRY
100	EUROPE	NETHERLANDS
500	EUROPE	UK
600	EUROPE	FRANCE
700	EUROPE	GERMANY
900	ASIA	CHINA
300	ASIA	KOREA
700	ASIA	JAPAN


```

SELECT region,
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY count),
       PERCENTILE_DISC(0.7) WITHIN GROUP (ORDER BY count)
FROM sales
GROUP BY region;
  
```


REGION	PERCENTILE_DISC(0.5)	PERCENTILE_DISC(0.7)
EUROPE	500	600
ASIA	700	900

PI

Zweck

Liefert die Konstante π (pi; Kreiszahl).

Syntax

pi::=



Beispiel(e)

```
SELECT PI( );  
PI  
-----  
3.141592653589793
```

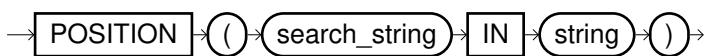
POSITION

Zweck

Liefert die Position in der Zeichenkette `string` zurück, an der das erste Mal die Zeichenkette `search_string` vorkommt. Falls diese gar nicht enthalten ist, so wird der Wert 0 zurückgeliefert.

Syntax

position::=



Anmerkung(en)

- Ist ein Argument NULL, so wird NULL zurückgegeben.
- Die Funktion `INSTR` ist äquivalent zu dieser Funktion.

Beispiel(e)

```
SELECT POSITION( 'cab' IN 'abcabcabc' ) POS;  
POS  
-----  
3
```

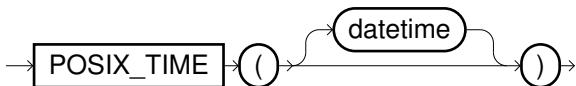
POSIX_TIME

Zweck

Die Posix Time (oder auch Unix Time) ist im POSIX Standard festgelegt und liefert die Anzahl der Sekunden seit dem 1. Januar 1970 um 0 Uhr (UTC) zurück. Mit dieser Funktion lässt sich ein Zeitpunkt in einen Dezimalwert umrechnen.

Syntax

posix_time ::=



Anmerkung(en)

- `POSIX_TIME(<datetime>)` ist äquivalent zum Funktionsaufruf `SECONDS_BETWEEN(<datetime>, '1970-01-01 00:00:00')` falls die Session-Zeitzone auf UTC gesetzt ist.
- Wird kein Parameter angegeben, so bezieht sich die Posix Time auf den aktuellen Zeitpunkt (also `CURRENT_TIMESTAMP`).
- Für Zeitpunkte vor dem 1. Januar 1970 (UTC) liefert die Funktion negative Werte zurück.
- Mit der Funktion `FROM_POSIX_TIME` kann ein numerischer Wert in einen Zeitstempel umgewandelt werden.

Beispiel(e)

```

ALTER SESSION SET TIME_ZONE='UTC';
SELECT POSIX_TIME('1970-01-01 00:00:01') PT1,
       POSIX_TIME('2009-02-13 23:31:30') PT2;

PT1          PT2
-----
1.000      1234567890.000
  
```

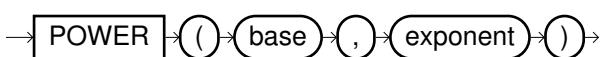
POWER

Zweck

Liefert als Ergebnis Potenz zweier Zahlen.

Syntax

power ::=



Beispiel(e)

```

SELECT POWER(2,10) POWER;
POWER
  
```

```
-----  
1024
```

RADIANS

Zweck

Konvertiert die Zahl n vom Grad- ins Bogenmaß (Radian).

Syntax

radians ::=



Anmerkung(en)

- Siehe auch die Funktion [DEGREES](#).

Beispiel(e)

```
SELECT RADIANS(180) RADIANS;  
  
RADIANS  
-----  
3.141592653589793
```

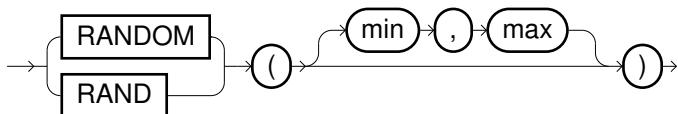
RAND[OM]

Zweck

Erzeugt eine Zufallszahl.

Syntax

random ::=



Anmerkung(en)

- Der Rückgabewert ist stets vom Typ DOUBLE.
- Bei Angabe der optionalen Parameter wird eine Zufallszahl aus dem Intervall [min;max] zurückgeliefert. Ohne Angabe von Parametern beträgt das Intervall [0;1].

Beispiel(e)

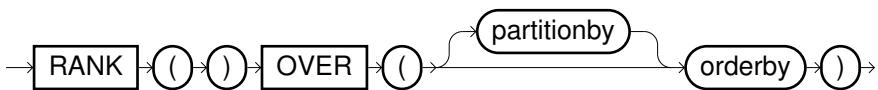
```
SELECT RANDOM() RANDOM_1, RANDOM(5,20) RANDOM_2;
RANDOM_1          RANDOM_2
-----
0.379277567626116 12.7548096816858
```

RANK**Zweck**

Liefert den Rang einer Zeile innerhalb einer geordneten Partition.

Syntax

rank ::=

**Anmerkung(en)**

- RANK kann nur als analytische Funktion verwendet werden (in Verbindung mit OVER(...), siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)).
- Die OVER-Klausel muss einen ORDER BY-Teil und darf keine Window-Klausel enthalten.
- Für gleichrangige Zeilen wird dieselbe Wert geliefert. Dafür werden danach entsprechend viele Werte ausgelassen (im Gegensatz zu [DENSE_RANK](#)).

Beispiel(e)

```
SELECT name, salary, RANK() OVER (ORDER BY Salary DESC) RANK
FROM staff ORDER BY rank;

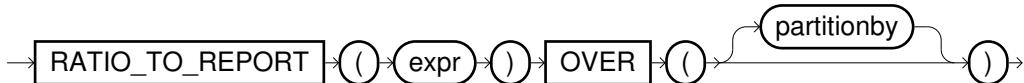
NAME      SALARY      RANK
-----
Schmidt    81000       1
Müller     75000       2
Huber      48000       3
Schultze   37000       4
Schulze    37000       4
Meier      25000       6
```

RATIO_TO_REPORT**Zweck**

Berechnet das Verhältnis eines Wertes zur Gesamtsumme.

Syntax

ratio_to_report ::=



Anmerkung(en)

- Die OVER-Klausel (siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)) kann weder ORDER BY-Teil noch Window-Klausel enthalten.

Beispiel(e)

```
SELECT name, salary, RATIO_TO_REPORT(salary) OVER () RATIO_TO_REPORT
FROM staff ORDER BY RATIO_TO_REPORT;
```

NAME	SALARY	RATIO_TO_REPORT
Meier	25000	0.082508250825083
Schultze	37000	0.122112211221122
Schulze	37000	0.122112211221122
Huber	48000	0.158415841584158
Müller	75000	0.247524752475248
Schmidt	81000	0.267326732673267

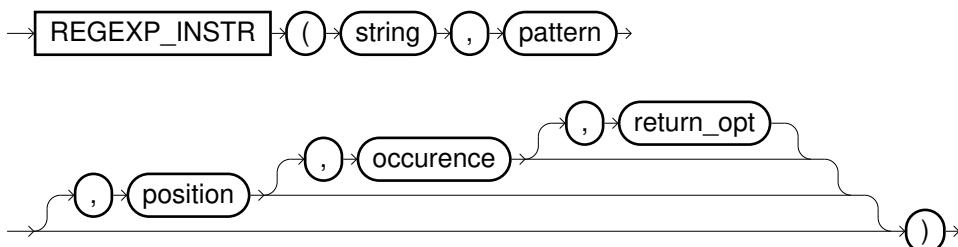
REGEXP_INSTR

Zweck

Sucht den regulären Ausdruck **pattern** in der Zeichenkette **string**. Falls dieser gar nicht enthalten ist, so wird der Wert 0 zurückgeliefert, ansonsten die Stelle des Matches (Details siehe Anmerkungen).

Syntax

regexp_instr ::=



Anmerkung(en)

- Detaillierte Informationen und Beispiele zu regulären Ausdrücken in EXASOL finden Sie in [Abschnitt 2.1.3, Reguläre Ausdrücke](#).
- Der optionale Parameter **position** definiert die Position, ab der gesucht werden soll (beginnend mit 1).

- Die optionale positive Zahl `occurrence` definiert, nach welchem Auftreten des regulären Ausdrucks gesucht wird. Hierbei ist zu beachten, dass die Suche für das zweite Auftreten beim ersten Buchstaben nach dem ersten Auftreten beginnt.
- Der optionale Parameter `return_opt` spezifiziert das Ergebnis der Funktion im Falle eines Matches:

0 (Default)	Die Funktion liefert die Anfangsposition des Treffers zurück (gezählt wird ab 1)
1	Die Funktion liefert die Endposition der Übereinstimmung zurück (das erste Zeichen nach dem Match, gezählt wird ab 1)
- `REGEXP_INSTR(string, pattern)` entspricht dem Aufruf `REGEXP_INSTR(string, pattern, 1, 1)`.
- Siehe auch Funktionen `INSTR`, `REGEXP_REPLACE` und `REGEXP_SUBSTR` sowie das Prädikat `[NOT] REGEXP_LIKE`.

Beispiel(e)

```
SELECT REGEXP_INSTR('Phone: +497003927877678',
                     '\+?\d+'
                   ) REGEXP_INSTR1,
       REGEXP_INSTR('From: my_mail@yahoo.com - To: SERVICE@EXASOL.COM',
                    '(?i)[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}',
                    1,
                    2
                  ) REGEXP_INSTR2;

REGEXP_INSTR1 REGEXP_INSTR2
-----
          8           31
```

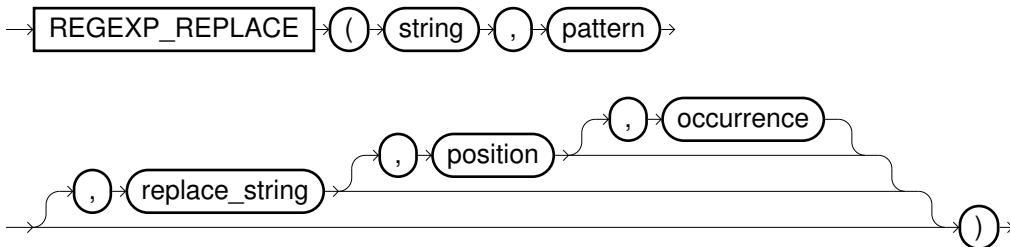
REGEXP_REPLACE

Zweck

Ersetzt Vorkommen von `pattern` in der Zeichenkette `string` durch `replace_string`.

Syntax

`regexp_replace::=`



Anmerkung(en)

- Detaillierte Informationen und Beispiele zu regulären Ausdrücken in EXASOL finden Sie in [Abschnitt 2.1.3, Reguläre Ausdrücke](#).
- Ist `pattern` NULL, so wird `string` zurückgegeben.
- Wird `replace_string` nicht angegeben oder ist er NULL, werden die Vorkommen von `pattern` im Ergebnis gelöscht.

- Im `replace_string` können Captures mittels `\1, \2, ..., \9` bzw. `\g<name>` benutzt werden, die durch `pattern` definiert sind.
- Der optionale Parameter `position` definiert die Position, ab der gesucht werden soll (beginnend mit 1).
- Die optionale positive Zahl `occurrence` definiert, welches Auftreten des regulären Ausdrucks ersetzt werden soll. Hierbei ist zu beachten, dass Treffer nicht überlappend sein können. Daher beginnt die Suche für das zweite Auftreten beim ersten Buchstaben nach dem ersten Auftreten. Bei 0 werden alle Auftreten ersetzt (Default). Bei Angabe der positiven Zahl `n` wird das `n`-te Auftreten ersetzt.
- Siehe auch Funktionen `REPLACE`, `REGEXP_INSTR` und `REGEXP_SUBSTR` sowie das Prädikat `[NOT] REGEXP_LIKE`.

Beispiel(e)

```
SELECT REGEXP_REPLACE(
    'From: my_mail@yahoo.com',
    '(?i)^From: ([a-z0-9._%+-]+)@([a-z0-9.-]+\.[a-z]{2,4}$)' ,
    'Name: \1 - Domain: \2') REGEXP_REPLACE;

REGEXP_REPLACE
-----
Name: my_mail - Domain: yahoo.com
```

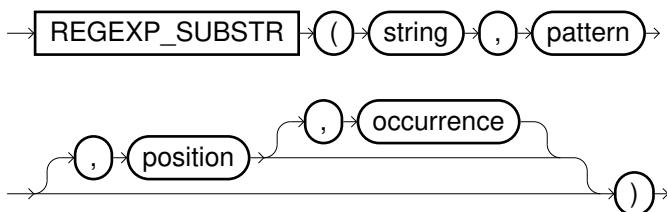
REGEXP_SUBSTR

Zweck

Liefert einen Teilstring aus der Zeichenkette `string`.

Syntax

`regexp_substring`::=



Anmerkung(en)

- Detaillierte Informationen und Beispiele zu regulären Ausdrücken in EXASOL finden Sie in [Abschnitt 2.1.3, Reguläre Ausdrücke](#).
- Die Funktion `REGEXP_SUBSTR` ist ähnlich zur Funktion `REGEXP_INSTR`, nur dass anstatt der entsprechenden Anfangsposition eines Matches der gefundene Teilstring zurückgeliefert wird.
- Der Parameter `pattern` definiert einen regulären Ausdruck, nach dem gesucht wird. Falls kein Match gefunden wird, so liefert die Funktion `NONE` zurück. Ansonsten wird der entsprechend Teilstring zurückgeliefert.
- Der optionale Parameter `position` definiert die Position, ab der gesucht werden soll (beginnend mit 1).
- Die optionale positive Zahl `occurrence` definiert, nach welchem Auftreten des regulären Ausdrucks gesucht wird. Hierbei ist zu beachten, dass die Suche für das zweite Auftreten beim ersten Buchstaben nach dem ersten Auftreten beginnt.
- `REGEXP_SUBSTR(string, pattern)` entspricht dem Aufruf `REGEXP_SUBSTR(string, pattern, 1, 1)`.

- Siehe auch Funktionen `SUBSTR[ING]`, `REGEXP_INSTR` und `REGEXP_REPLACE` sowie das Prädikat `[NOT] REGEXP_LIKE`.

Beispiel(e)

```
SELECT REGEXP_SUBSTR('My mail address is my_mail@yahoo.com',
                      '(?i)[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}') EMAIL
;
EMAIL
-----
my_mail@yahoo.com
```

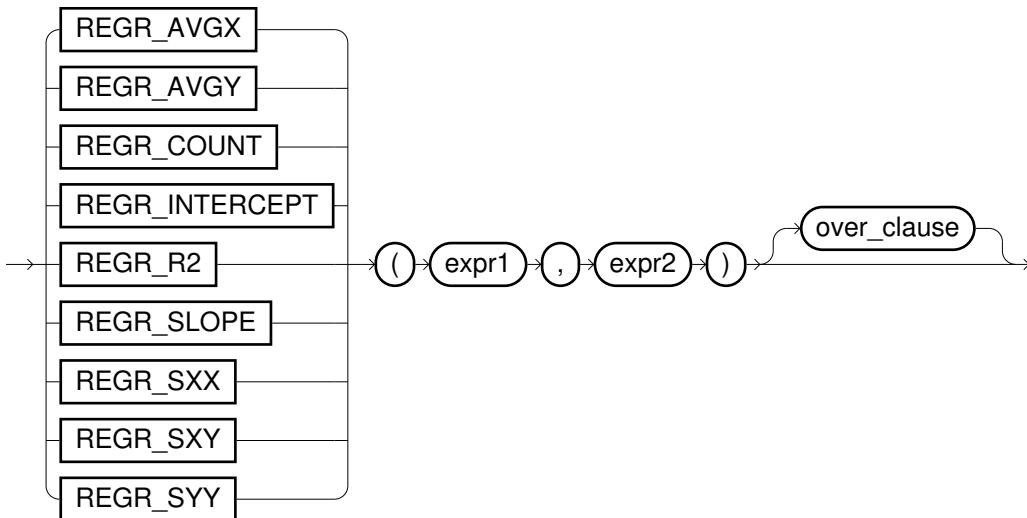
REGR_*

Zweck

Die Regressionsfunktionen dienen der Ermittlung einer linearen Funktion mit minimaler Summe der Abweichungsquadrate.

Syntax

`regr_functions ::=`



Anmerkung(en)

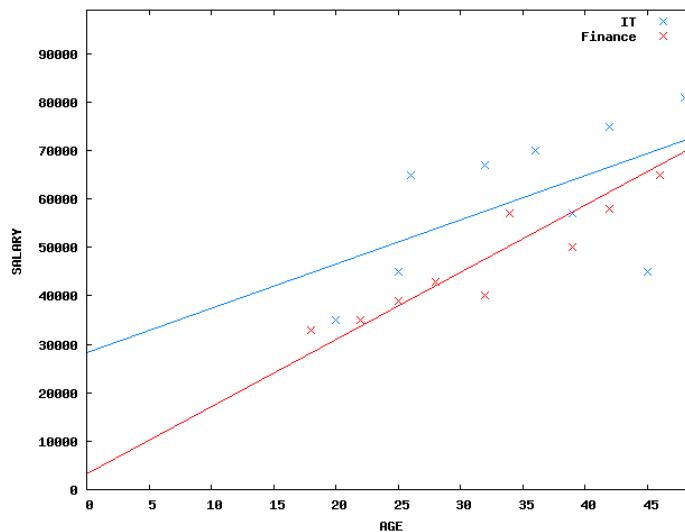
! `expr2` wird als unabhängige Variable ausgewertet (x-Wert), `expr1` als abhängige Variable (y-Wert).

- Besitzt `expr1` oder `expr2` den Wert `NULL`, so wird das entsprechende Zahlenpaar bei der Berechnung ignoriert.
- Beschreibung der verschiedenen Regressionsfunktionen:

Funktion	Beschreibung	Formel
<code>REGR_SLOPE</code>	Steigung der Regressionslinie	$\text{REGR_SLOPE(expr1, expr2) = } \frac{\text{COVAR_POP(expr1, expr2)}}{\text{VAR_POP(expr2)}}$

Funktion	Beschreibung	Formel
REGR_INTERCEPT	Schnittpunkt mit der y-Achse	REGR_INTERCEPT(expr1, expr2) = $\text{AVG(expr1)} - \text{REGR_SLOPE(expr1, expr2)} \cdot \text{AVG(expr2)}$
REGR_COUNT	Anzahl an Zahlenpaaren, die keine NULL enthalten	
REGR_R2	Regressionskoeffizient (eine Art Gütwert der Berechnung).	REGR_R2(expr1, expr2) = $\begin{cases} \text{NULL} & \text{für } \text{VAR_POP(expr2)} = 0 \\ 1 & \text{für } \text{VAR_POP(expr2)} \neq 0, \text{ VAR_POP(expr1)} = 0 \\ \text{CORR(expr1, expr2)}^2 & \text{sonst} \end{cases}$
REGR_AVGX	Durchschnitt der unabhängigen Werte (x-Werte)	REGR_AVGX(expr1, expr2) = AVG(expr2)
REGR_AVGY	Durchschnitt der abhängigen Werte (y-Werte)	REGR_AVGY(expr1, expr2) = AVG(expr1)
REGR_SXX	Hilfsfunktion	REGR_SXX(expr1, expr2) = REGR_COUNT(expr1, expr2) · VAR_POP(expr2)
REGR_SXY	Hilfsfunktion	REGR_SXY(expr1, expr2) = REGR_COUNT(expr1, expr2) · COVAR_POP(expr1, expr2)
REGR_SYY	Hilfsfunktion	REGR_SYY(expr1, expr2) = REGR_COUNT(expr1, expr2) · VAR_POP(expr1)

- Im folgenden Beispiel werden 2 Regressionslinien erzeugt. Die Punkte entsprechen tatsächlichen Einträgen in der Tabelle staff (rot=Finance, blau=IT). Die beiden Linien entsprechen den ermittelten Regressionslinien. Im Beispiel zeigt sich, dass die Regressionslinie für den Finance-Sektor von höherer Güte ist (siehe REGR_R2-Wert), also die Entwicklung der Gehälter stärker an das Alter des Mitarbeiters gekoppelt ist.



Beispiel(e)

```
SELECT industry,
       REGR_SLOPE(salary,age) AS REGR_SLOPE,
       REGR_INTERCEPT(salary,age) AS REGR_INTERCEPT,
       REGR_R2(salary,age) AS REGR_R2
```

```
FROM staff GROUP BY industry;
```

INDUSTRY	REGR_SLOPE	REGR_INTERCEPT	REGR_R2
Finance	1385.778395127685	3314.563521743759	0.92187386620889
IT	913.8748888230062	28217.46219982212	0.325366059690104

REPEAT

Zweck

Fügt die Zeichenkette `string` mehrmals hintereinander (`n` mal).

Syntax

`repeat::=`

→ **REPEAT** → (→ **string** → , → **n** →) →

Anmerkung(en)

- Ist eines der Argumente `NULL` oder `n=0`, so wird `NULL` zurückgegeben.
- Der Parameter `n` muss eine positive Integer-Zahlen im Bereich von 0 bis 999999999 sein.
- Die Ergebnis-Zeichenkette darf höchstens 2000000 Zeichen enthalten.
- Ist der übergebene Wert keine Zeichenkette, so wird er automatisch in eine konvertiert.
- Siehe auch Funktion [SPACE](#).

Beispiel(e)

```
SELECT REPEAT( 'abc' , 3 );
```

```
REPEAT( 'abc' , 3 )
-----
abcabcabc
```

REPLACE

Zweck

Liefert als Ergebnis den String, der entsteht, wenn in `string` alle Vorkommen von `search_string` durch `replace_string` ersetzt werden.

Syntax

`replace::=`

→ **REPLACE** → (→ **string** → , → **search_string** → , → **replace_string** →) →

Anmerkung(en)

- Wird `replace_string` nicht angegeben oder ist er `NULL`, werden die Vorkommen von `search_string` im Ergebnis gelöscht.
- Ist `search_string` `NULL`, so wird `string` zurückgegeben.
- Sind die übergebenen Daten keine Zeichenketten, werden sie automatisch in solche konvertiert.
- Der Rückgabetyp ist immer eine Zeichenkette, auch wenn alle Parameter anderen Typ besitzen.

Beispiel(e)

```
SELECT REPLACE('Apple juice is great', 'Apple', 'Orange') REPLACE_1;

REPLACE_1
-----
Orange juice is great

SELECT REPLACE( '-TEST-Text-TEST-' , ' -TEST-' ) REPLACE_2;

REPLACE_2
-----
Text
```

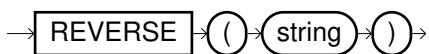
REVERSE

Zweck

Dreht eine Zeichenkette um.

Syntax

reverse ::=



Anmerkung(en)

- Ist `string` `NULL`, so wird `NULL` zurückgegeben.
- Sind die übergebenen Daten keine Zeichenketten, werden sie automatisch in solche konvertiert.
- Der Rückgabetyp ist immer eine Zeichenkette der gleichen Länge wie der Eingabeparameter.

Beispiel(e)

```
SELECT REVERSE( 'abcde' ) REVERSE;

REVERSE
-----
edcba
```

RIGHT

Zweck

Liefert den rechtsbündigen Teil-String der Zeichenkette `string` mit Länge `length`.

Syntax

`right ::=`



Anmerkung(en)

- Falls `length` oder `string` NULL sind, so wird NULL zurückgeliefert.
- Auch wenn `length` größer ist als die Länge von `string`, so wird der Original-String zurückgeliefert. Falls `length` negativ oder 0 ist, so wird NULL zurückgeliefert.
- Siehe auch die Funktionen [SUBSTR\[ING\]](#), [MID](#) und [LEFT](#).

Beispiel(e)

```

SELECT RIGHT( 'abcdef' , 3 ) RIGHT_SUBSTR;
RIGHT_SUBSTR
-----
def

```

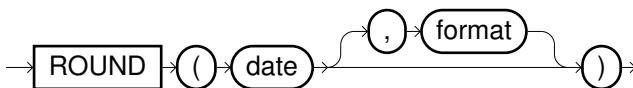
ROUND (datetime)

Zweck

Rundet einen Datums- bzw. Zeitstempelwert auf die vorgegebene Einheit.

Syntax

`round (datetime) ::=`



Anmerkung(en)

- Für das Format können 'YYYY', 'MM', 'DD', 'HH', 'MI', 'SS' für Jahr, Monat, Tag, Stunde, Minute und Sekunde verwendet werden.

CC, SCC	Jahrhundert
YYYY, SYYY, YEAR, SYEAR,	Jahr
YY, YY, Y	
IYYY, IYY, IY, I	Jahr nach internationaler Norm ISO 8601
Q	Quartal
MONTH, MON, MM, RM	Monat

WW	Der gleiche Wochentag wie der erste Tag des Jahres
IW	Der gleiche Wochentag wie der erste Tag des ISO-Jahres
W	Der gleiche Wochentag wie der erste Tag des Monats
DDD, DD, J	Tag
DAY, DY, D	Erster Tag der Woche. Der erste Tag einer Woche wird über den Parameter NLS_FIRST_DAY_OF_WEEK definiert (siehe ALTER SESSION und ALTER SYSTEM).
HH, HH24, HH12	Stunde
MI	Minute
SS	Sekunden
• Aufgerundet wird bei Jahr ab dem 1. Juli, bei Monat ab dem 16. eines Monats, bei Tag ab 12 Uhr, bei Stunden ab 30 Minuten, bei Minuten ab 30 Sekunden und bei Sekunden ab 500 Millisekunden. Ansonsten wird abgerundet.	
• Falls kein Format angegeben ist, so wird der Wert auf Tage gerundet.	
• Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.	

Beispiel(e)

```
SELECT ROUND(DATE '2006-12-31', 'YYYY') ROUND;

ROUND
-----
2007-01-01

SELECT ROUND(TIMESTAMP '2006-12-31 12:34:58', 'MI') ROUND;

ROUND
-----
2006-12-31 12:35:00.000
```

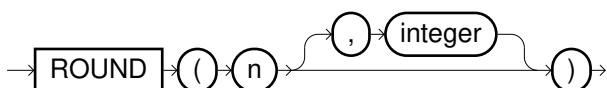
ROUND (number)

Zweck

Rundet die Zahl n auf integer Ziffern hinter dem Komma (kaufmännisches Runden).

Syntax

round (number)::=



Anmerkung(en)

- Falls das zweite Argument nicht angegeben wird, so wird auf eine ganze Zahl gerundet.
- Falls das zweite Argument negativ ist, so wird auf integer Ziffern vor dem Komma gerundet.
- Falls n vom Typ DECIMAL(p,s) ist und das zweite Argument positiv, so ist der Ergebnistyp vom Typ DECIMAL(p,integer).
- Falls n vom Typ DOUBLE ist, so ist der Ergebnistyp ebenfalls DOUBLE. Dies kann aufgrund der Darstellungsproblematik von DOUBLE-Werten dazu führen, dass gerundete Werte mehr Nachkommastellen besitzen, als

durch das zweite Argument definiert. Wir empfehlen daher, bei DOUBLE-Werten das Ergebnis nach der Rundung stets auf einen geeigneten DECIMAL-Typ zu konvertieren.

Beispiel(e)

```
SELECT ROUND(123.456,2) ROUND;
ROUND
-----
123.46
```

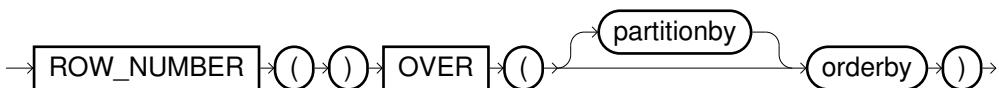
ROW_NUMBER

Zweck

Liefert den Nummer einer Zeile innerhalb einer geordneten Partition.

Syntax

row_number ::=



Anmerkung(en)

- ROW_NUMBER kann nur als analytische Funktion verwendet werden (in Verbindung mit OVER(...), siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#)).
- Die OVER-Klausel muss einen ORDER BY-Teil und darf keine Window-Klausel enthalten.
- Bei gleichrangige Zeilen ist der Wert nicht deterministisch.

Beispiel(e)

```
SELECT name, ROW_NUMBER() OVER (ORDER BY Salary DESC) ROW_NUMBER
FROM staff ORDER BY name;

NAME      ROW_NUMBER
-----
Huber          3
Meier          6
Müller         2
Schmidt        1
Schultze       4
Schulze        5
```

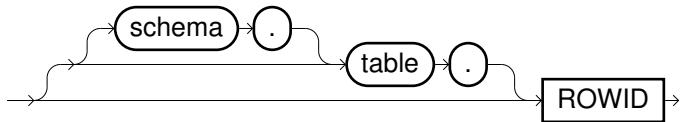
ROWID

Zweck

Jede Zeile einer Basistabelle in der Datenbank hat eine eindeutige Adresse, die sogenannte ROWID. Auf diese Adresse kann über die ROWID-Pseudospalte vom Datentyp DECIMAL(36,0) lesend zugegriffen werden.

Syntax

rowid::=



Anmerkung(en)

Die ROWID-Spalte darf u.a. in folgenden SQL-Konstrukten verwendet werden:

- in allen Spaltenlisten von SELECT-Statements
- in VIEW-Definitionen (siehe auch [CREATE VIEW](#)), wobei in diesen Fällen ein Alias gesetzt werden muss
- in den Bedingungen von [INSERT](#), [UPDATE](#), [MERGE](#) und [DELETE](#) Statements

Die ROWIDs einer Tabelle werden vom DBMS verwaltet. Dieses stellt sicher, dass die ROWIDs innerhalb einer Tabelle disjunkt sind - ROWIDs verschiedener Tabellen können dagegen durchaus gleich sein. Durch DML-Statements wie [INSERT](#), [UPDATE](#), [DELETE](#), [TRUNCATE](#) oder [MERGE](#) werden alle ROWIDs der betreffenden Tabellen invalidiert und vom DBMS neu vergeben. Strukturelle Tabellenänderungen wie das Hinzufügen von Spalten lassen die ROWIDs dagegen unverändert.

ROWIDs sind nur gültig für Basistabellen und können daher nicht für den Zugriff auf eine View verwendet werden. Ein Anwendungsbeispiel für ROWIDs ist das gezielte Löschen bestimmter Zeilen einer Tabelle, um beispielsweise die UNIQUE-Eigenschaft wiederherzustellen.

Beispiel(e)

```

SELECT ROWID, i FROM t;

ROWID          I
-----
318815196395658560306020907325849600      1
318815196395658560306020907325849601      1
318815196395658560306302382302560256      2
318815196395658560306302382302560257      3

-- Wiederherstellung der Eindeutigkeit von i
DELETE FROM t WHERE NOT EXISTS (
    SELECT r FROM (SELECT MIN(ROWID) r FROM t GROUP BY i) h
    WHERE t.ROWID=h.r);

CREATE VIEW v AS SELECT ROWID r, i FROM t;
SELECT * FROM v;

R          I
-----
318815196395658578752764981035401216      1
318815196395658578753046456012111872      2
318815196395658578753046456012111873      3

-- Fehlermeldung, da nur Basistabellen eine ROWID-Spalte haben
SELECT ROWID FROM v;

Error: [42000] ROWID is invalid for non-material tables

```

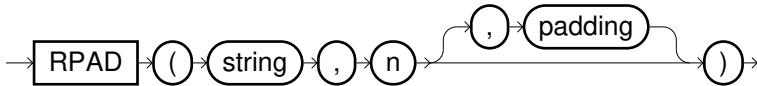
RPAD

Zweck

Liefert eine Zeichenkette der Länge n zurück, die ausgehend von der Zeichenkette `string` von der rechten Seite her mit dem Ausdruck `padding` aufgefüllt wird.

Syntax

`rpad ::=`



Anmerkung(en)

- Falls die Variable `padding` nicht angegeben ist, so wird mit Leerzeichen aufgefüllt.
- Auch falls n größer als 2.000.000 ist, wird das Ergebnis auf 2.000.000 Zeichen begrenzt.
- Um eine Zeichenkette von der linken Seite her aufzufüllen, können Sie die Funktion [LPAD](#) verwenden.

Beispiel(e)

```

SELECT RPAD( 'abc' , 5 , 'X' ) ;

RPAD( 'abc' , 5 , 'X' )
-----
abcXX
  
```

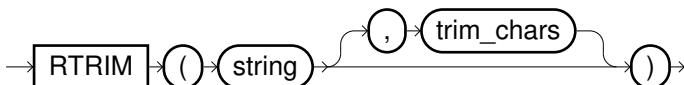
RTRIM

Zweck

`RTRIM` entfernt von der rechten Grenze der Zeichenkette `string` alle im Ausdruck `trim_chars` angegebenen Zeichen.

Syntax

`rtrim ::=`



Anmerkung(en)

- Falls der Parameter `trim_chars` nicht angegeben ist, werden Leerzeichen entfernt.

Beispiel(e)

```

SELECT RTRIM( 'abcdef' , 'afe' ) ;
  
```

```
RTRIM( 'abcdef' , 'afe' )
-----
abcd
```

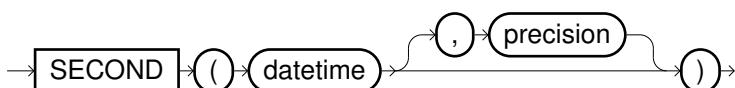
SECOND

Zweck

Liefert die Sekunden eines Zeitstempels.

Syntax

second ::=



Anmerkung(en)

- Der optionale zweite Parameter definiert, wie viele Stellen hinter dem Komma angegeben werden sollen.
- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT SECOND(TIMESTAMP '2010-10-20 11:59:40.123', 2);

SECOND
-----
40.12
```

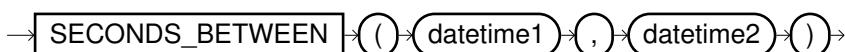
SECONDS_BETWEEN

Zweck

Liefert die Anzahl der Sekunden zwischen zwei Zeitstempeln.

Syntax

seconds_between ::=



Anmerkung(en)

- Falls der Zeitstempel timestamp1 vor dem Zeitstempel timestamp2 liegt, so ist das Ergebnis negativ.
- Das Ergebnis enthält zusätzlich die Differenz der Millisekunden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion intern in UTC berechnet.

Beispiel(e)

```
SELECT SECONDS_BETWEEN(TIMESTAMP '2000-01-01 12:01:02.345',  
                      TIMESTAMP '2000-01-01 12:00:00') SB;  
  
SB  
-----  
62.345
```

SESSIONTIMEZONE

Zweck

Liefert die in der Session eingestellte Zeitzone zurück (siehe auch [ALTER SESSION](#)).

Syntax

sessiontimezone ::=

→ **SESSIONTIMEZONE** →

Anmerkung(en)

- Siehe auch [DBTIMEZONE](#).

Beispiel(e)

```
SELECT SESSIONTIMEZONE;  
  
SESSIONTIMEZONE  
-----  
EUROPE/BERLIN
```

SIGN

Zweck

Liefert das Vorzeichen einer Zahl n. Für negative/positive Zahlen wird -1/1 zurückgegeben, für die Zahl 0 der Wert 0.

Syntax

sign ::=

→ **SIGN** → (→ n →) →

Beispiel(e)

```
SELECT SIGN(-123);
```

```
SIGN(-123)
-----
-1
```

SIN

Zweck

Liefert den Sinus einer Zahl n.

Syntax

sin::=

→ [SIN] → (→ n →) →

Beispiel(e)

```
SELECT SIN(PI() / 6);

SIN(PI() / 6)
-----
0.5
```

SINH

Zweck

Liefert den Sinus Hyperbolicus einer Zahl n.

Syntax

sinh::=

→ [SINH] → (→ n →) →

Beispiel(e)

```
SELECT SINH(0) SINH;

SINH
-----
0
```

SOUNDEX

Zweck

SOUNDEX definiert die Laut-Ähnlichkeit von Zeichenketten. Hiermit können unterschiedlich geschriebene, aber ähnlich ausgesprochene Wörter verglichen werden.

Syntax

soundex ::=

→ [SOUNDEX] → (→ string →) →

Anmerkung(en)

- Zur Berechnung wird der Algorithmus verwendet, der beschrieben wird in: *Donald Knuth, The Art of Computer Programming, Vol. 3.*
- Das Ergebnis ist stets eine Zeichenkette der Länge 4 (mit einem Buchstaben und drei Ziffern).
- Diese Funktion ist ähnlich zu [COLOGNE_PHONETIC](#), welche besser für deutsche Wörter geeignet ist.

Beispiel(e)

```
SELECT SOUNDEX('smythe'), SOUNDEX('Smith');  
  
SOUNDEX('smythe') SOUNDEX('Smith')  
----- -----  
S530          S530
```

SPACE

Zweck

SPACE erzeugt eine Zeichenkette aus n Leerzeichen.

Syntax

space ::=

→ [SPACE] → (→ integer →) →

Anmerkung(en)

- Siehe auch die Funktion [REPEAT](#).

Beispiel(e)

```
SELECT 'x' || SPACE(5) || 'x' my_string;  
  
MY_STRING
```

```
-----  
x     x
```

SQRT

Zweck

Liefert die Quadratwurzel einer Zahl n.

Syntax

sqrt::=



Anmerkung(en)

- Für den Parameter muss gelten: $n \geq 0$.

Beispiel(e)

```
SELECT SQRT( 2 ) ;  
  
SQRT( 2 )  
-----  
1.414213562373095
```

ST_*

Zweck

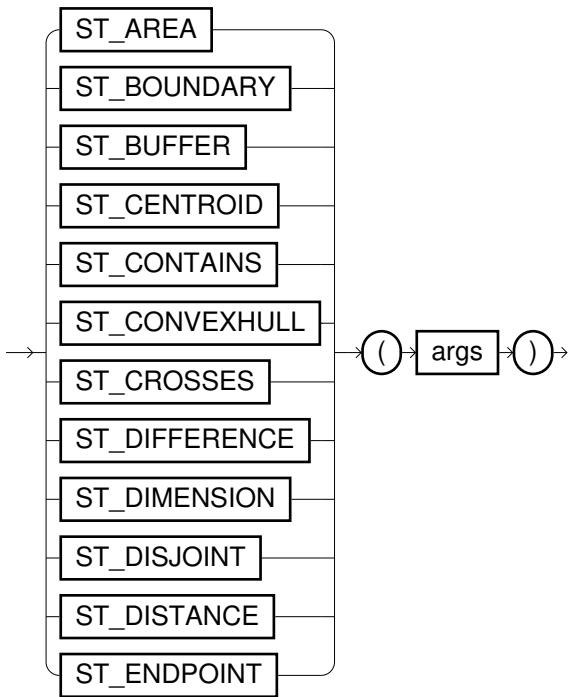
Diverse Funktionen für Geodaten-Objekte. Bitte berücksichtigen Sie [Abschnitt 2.4, Geodaten](#) für weitere Details zu Geodaten-Objekten und deren Funktionen.



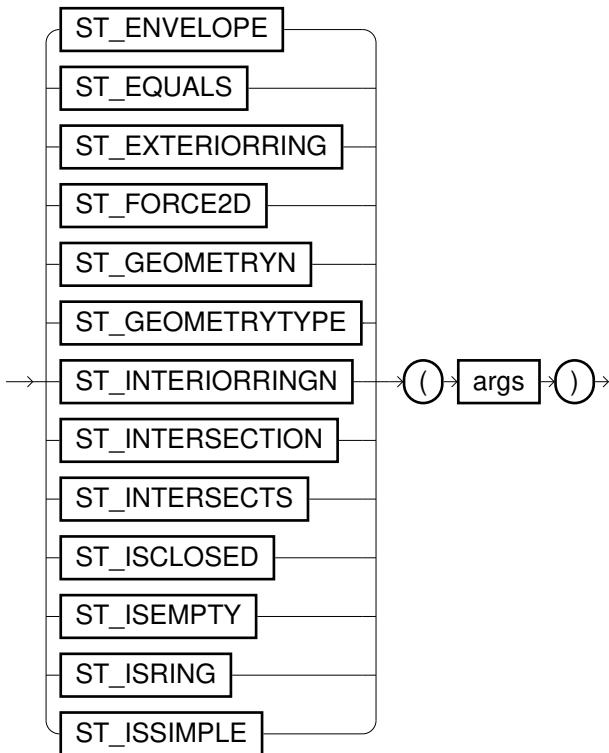
Bitte beachten Sie, dass Geodaten Teil der Advanced Edition von EXASOL ist.

Syntax

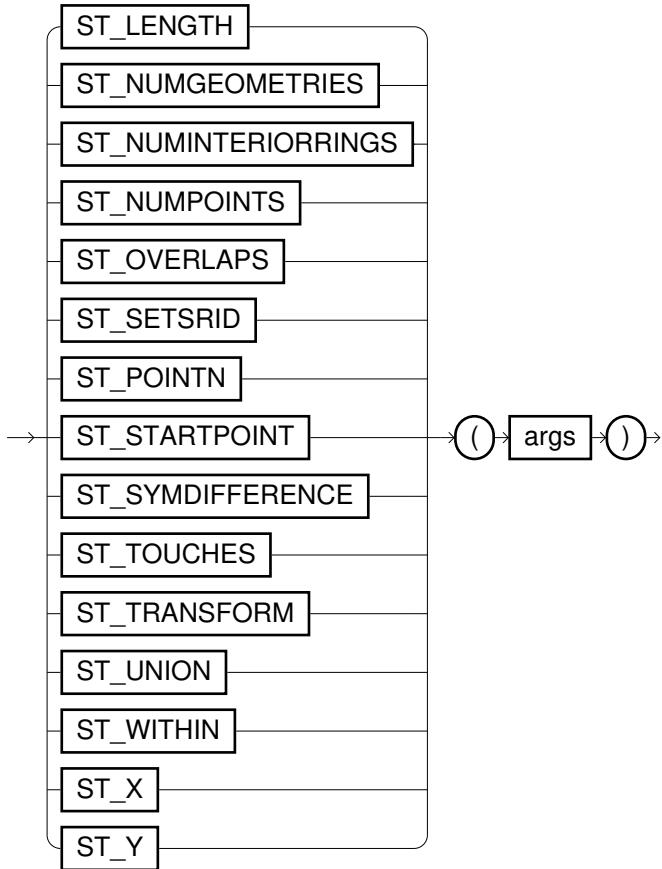
geospatial_functions_1::=



geospatial_functions_2::=



geospatial_functions_3::=



Beispiel(e)

```

SELECT ST_DISTANCE('POLYGON ((0 0, 0 4, 2 4, 2 0, 0 0))',
                   'POINT(10 10)' ) ST_DISTANCE;

ST_DISTANCE
-----
10

```

STDDEV

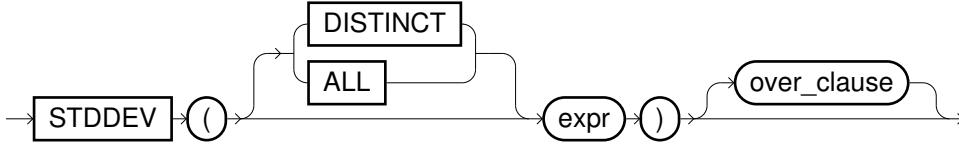
Zweck

Liefert die Standardabweichung innerhalb einer Stichprobe. Dies entspricht der folgenden Formel:

$$\text{STDDEV(expr)} = \sqrt{\frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n - 1}}$$

Syntax

stddev::=



Anmerkung(en)

- Bei Stichproben mit genau einem Element ist das Ergebnis 0.
- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT STDDEV(salary) STDDEV FROM staff WHERE age between 20 and 30;

STDDEV
-----
19099.73821810132
  
```

STDDEV_POP

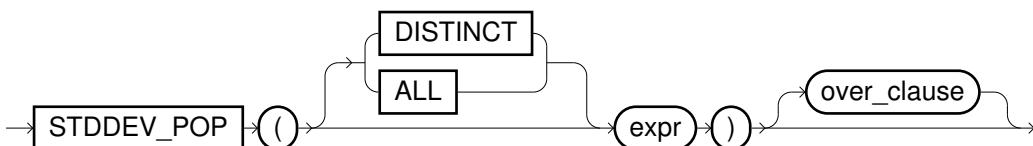
Zweck

Liefert die Standardabweichung innerhalb einer Population. Dies entspricht der folgenden Formel:

$$\text{STDDEV_POP(expr)} = \sqrt{\frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n}}$$

Syntax

stddev_pop::=



Anmerkung(en)

- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT STDDEV_POP(salary) STDDEV_POP FROM staff;

STDDEV_POP
-----
```

```
20792.12591343175
```

STDDEV_SAMP

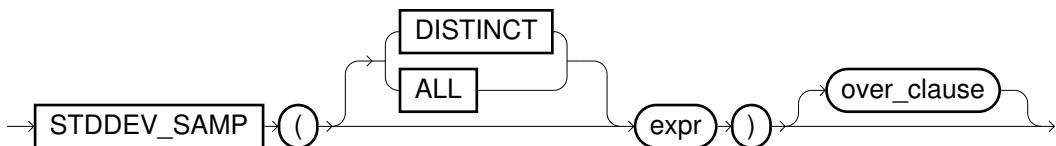
Zweck

Liefert die Standardabweichung innerhalb einer Stichprobe. Dies entspricht der folgenden Formel:

$$\text{STDDEV_SAMP(expr)} = \sqrt{\frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n - 1}}$$

Syntax

`stddev_samp ::=`



Anmerkung(en)

- `STDDEV_SAMP` ist identisch zur Funktion `STDDEV`. Falls allerdings die Stichprobe nur ein Element umfasst, ist das Ergebnis `NULL` anstatt 0.
- Wird `ALL` oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird `DISTINCT` angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT STDDEV_SAMP(salary) AS STDDEV_SAMP
FROM staff WHERE age between 20 and 30;

STDDEV_SAMP
-----
19099.73821810132
  
```

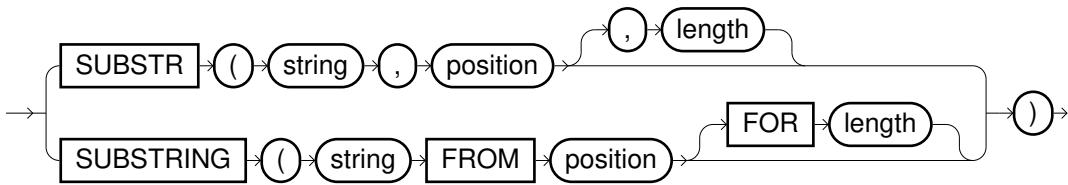
SUBSTR[ING]

Zweck

Liefert einen Teilstring der Länge `length` ab der Position `position` aus der Zeichenkette `string`.

Syntax

`substring ::=`



Anmerkung(en)

- Falls `length` nicht angegeben ist, so werden alle Zeichen bis zum Ende der Zeichenkette verwendet.
- Falls `position` negativ ist, so wird vom Ende der Zeichenkette gezählt.
- Falls `position` 0 oder 1 ist, so beginnt das Ergebnis vom ersten Zeichen der Zeichenkette.
- `MID` ist ein Alias für diese Funktion.
- Siehe auch [REGEXP_SUBSTR](#).

Beispiel(e)

```

SELECT SUBSTR('abcdef', 2, 3) S1,
       SUBSTRING('abcdef' FROM 4 FOR 2) S2
;

S1   S2
--- ---
bcd  de
  
```

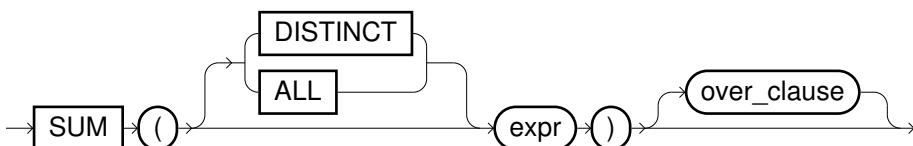
SUM

Zweck

Liefert die Summe.

Syntax

`sum ::=`



Anmerkung(en)

- Wird `ALL` oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird `DISTINCT` angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Es werden nur numerische Operanden unterstützt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT SUM(salary) SUM FROM staff WHERE age between 20 and 30;
  
```

```
SUM
-----
220500

SELECT SUM(salary) over (order by salary) SUM, salary FROM staff;

SUM          SALARY
-----
25000        25000
60500        35500
97500        37000
145500       48000
220500       75000
301500       81000
```

SYS_CONNECT_BY_PATH

Zweck

Liefert für eine CONNECT BY Anfrage einen String mit dem kompletten Pfad vom Root-Knoten bis zum aktuellen Knoten, bestehend aus den Werten für expr und getrennt durch char. Details finden Sie in der Dokumentation zum [SELECT Befehl](#) in [Abschnitt 2.2.4, Anfragesprache \(DQL\)](#).

Syntax

sys_connect_by_path::=

$$\rightarrow \boxed{\text{SYS_CONNECT_BY_PATH}} \rightarrow (\rightarrow \boxed{\text{expr}} \rightarrow , \rightarrow \boxed{\text{char}} \rightarrow) \rightarrow$$

Beispiel(e)

```
SELECT SYS_CONNECT_BY_PATH(last_name, '/') "PATH"
  FROM employees
 WHERE last_name = 'Johnson'
 CONNECT BY PRIOR employee_id = manager_id
 START WITH last_name = 'Clark';

PATH
-----
/Clark/Jackson/Johnson
```

SYS_GUID

Zweck

Liefert eine systemweit eindeutige Hexadezimalzahl vom Typ CHAR(48).

Syntax

sys_guid::=



Beispiel(e)

```
SELECT SYS_GUID();  
  
SYS_GUID()  
-----  
069a588869dfcaf8baf520c801133ab139c04f964f047f1
```

SYSDATE

Zweck

Liefert das aktuelle Systemdatum, indem TO_DATE(SYSTIMESTAMP) ausgewertet wird (also interpretiert in der aktuellen Datenbank-Zeitzone).

Syntax

sysdate ::=



Anmerkung(en)

- Siehe auch [CURRENT_DATE](#).

Beispiel(e)

```
SELECT SYSDATE;  
  
SYSDATE  
-----  
2000-12-31
```

SYSTIMESTAMP

Zweck

Liefert den aktuellen Zeitstempel zurück, interpretiert in der aktuellen Datenbank-Zeitzone.

Syntax

systimestamp ::=



Anmerkung(en)

- Der Rückgabewert dieser Funktion ist vom Typ TIMESTAMP.
- Weitere Informationen zur Datenbank-Zeitzone finden Sie in der Funktion [DBTIMEZONE](#).
- Weitere Funktionen zum aktuellen Zeitpunkt:
 - [CURRENT_TIMESTAMP](#) bzw. [NOW](#)
 - [LOCALTIMESTAMP](#)

Beispiel(e)

```
SELECT SYSTIMESTAMP;

SYSTIMESTAMP
-----
2000-12-31 23:59:59
```

TAN

Zweck

Liefert den Tangens einer Zahl n.

Syntax

tan ::=

→ **TAN** → (→ n →) →

Beispiel(e)

```
SELECT TAN(PI() / 4);

TAN(PI() / 4)
-----
1
```

TANH

Zweck

Liefert den Tangens Hyperbolicus einer Zahl n.

Syntax

tanh ::=

→ **TANH** → (→ n →) →

Beispiel(e)

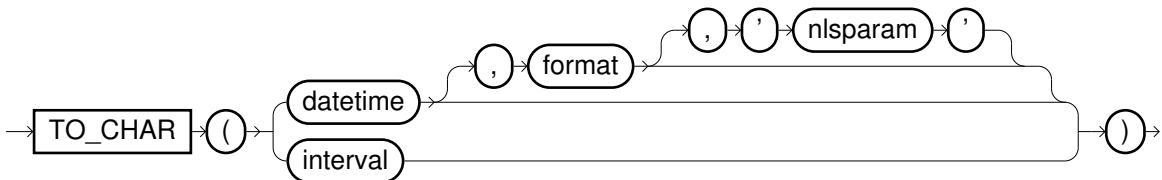
```
SELECT TANH( 0 ) TANH;
TANH
-----
0
```

TO_CHAR (datetime)**Zweck**

Wandelt ein Datum, Zeitstempel oder ein Intervall in eine Zeichenkette um.

Syntax

`to_char (datetime) ::=`

**Anmerkung(en)**

- Wird kein Ausgabe-Format angegeben, so wird das Standardformat verwendet, das im Session-Parameter `NLS_DATE_FORMAT` bzw. `NLS_TIMESTAMP_FORMAT` definiert ist.
- Welche Formate verwendbar sind, finden Sie in [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#).
- Im optionalen dritten Parameter können Sie die Spracheinstellung für das Format einstellen (z.B. '`NLS_DATE_LANGUAGE=GERMAN`'). Unterstützte Sprachen hierfür sind Deutsch (DEU, DEUTSCH bzw. GERMAN) und Englisch (ENG, ENGLISH).
- Für den Datentyp `TIMESTAMP WITH LOCAL TIME ZONE` wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT TO_CHAR(DATE '1999-12-31') TO_CHAR;
TO_CHAR
-----
1999-12-31

SELECT TO_CHAR(TIMESTAMP '1999-12-31 23:59:00',
               'HH24:MI:SS DD-MM-YYYY') TO_CHAR;
TO_CHAR
-----
23:59:00 31-12-1999

SELECT TO_CHAR(DATE '2013-12-16',
               'DD. MON YYYY',
               'NLS_DATE_LANGUAGE=DEU') TO_CHAR;
```

TO_CHAR

16. DEZ 2013

TO_CHAR (number)

Zweck

Wandelt eine Zahl in eine Zeichenkette um.

Syntax

to_char (number)::=



Anmerkung(en)

- Welche Formate verwendbar sind, finden Sie in [Abschnitt 2.6.2, Numerische Format-Modelle](#).

Beispiel(e)

```
SELECT TO_CHAR(12345.6789) TO_CHAR;  
  
TO_CHAR  
-----  
12345.6789  
  
SELECT TO_CHAR(12345.67890, '9999999.99999999') TO_CHAR;  
  
TO_CHAR  
-----  
12345.678900000  
  
SELECT TO_CHAR(-12345.67890, '000G000G000D000000MI') TO_CHAR;  
  
TO_CHAR  
-----  
000,012,345.678900-
```

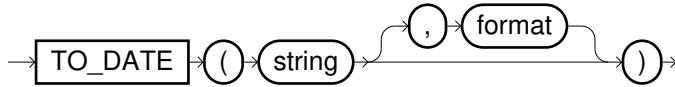
TO_DATE

Zweck

Wandelt eine Zeichenkette in ein Datum um.

Syntax

to_date::=



Anmerkung(en)

- Wird kein Format angegeben, so wird zur Interpretation von `string` das Standardformat verwendet, das im Session-Parameter [NLS_DATE_FORMAT](#) definiert ist.
- Welche Formate verwendbar sind, finden Sie in [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#).
- ISO-Formate (YYYY, IW, ID) dürfen nicht mit Nicht-ISO-Formaten gemischt werden.
- Falls einzelne Elemente ausgelassen werden, so wird für diese der kleinste mögliche Wert angenommen (z.B. wird `TO_DATE('1999-12', 'YYYY-MM')` als 1. Dezember 1999 interpretiert).
- Wie das Datum ausgegeben wird, bestimmt der Session-Parameter [NLS_DATE_FORMAT](#).

Beispiel(e)

```

SELECT TO_DATE( '1999-12-31' ) TO_DATE;

TO_DATE
-----
1999-12-31

SELECT TO_DATE( '31-12-1999' , 'DD-MM-YYYY' ) TO_DATE;

TO_DATE
-----
1999-12-31
  
```

TO_DSINTERVAL

Zweck

Wandelt eine Zeichenkette in ein Intervall um (INTERVAL DAY TO SECOND).

Syntax

`to_dsinterval ::=`



Anmerkung(en)

- Die Zeichenkette ist stets vom Format `[+ | -]DD HH:MI:SS[.FF]`. Gültige Werte für Tage (DD) sind 0-999999999, für Stunden (HH) 0-23, für Minuten (MI) 0-59 und für Sekunden (SS[.FF]) 0-59.999.
- Bitte beachten Sie, dass zwar mehr Nachkommastellen für die Sekundenbruchteile angegeben werden können, aber nach der dritten Stelle abgeschnitten wird.
- Siehe auch [TO_YMINTERVAL](#), [NUMTODSINTERVAL](#) und [NUMTOYMINTEGRAL](#).

Beispiel(e)

```
SELECT TO_DSINTERVAL('3 10:59:59.123') TO_DSINTERVAL;
TO_DSINTERVAL
-----
+000000003 10:59:59.123000000
```

TO_NUMBER

Zweck

Wandelt eine Zeichenkette in eine Zahl um.

Syntax

to_number ::=



Anmerkung(en)

- Das Format beeinflusst den Wert der Zahl nicht, sondern lediglich dessen Darstellung.
- Wenn man ein Format angibt, darf die zugehörige Zeichenkette nur Ziffern, sowie die Zeichen Plus, Minus, **NLS_NUMERIC_CHARACTERS**, Dezimalpunkt (Dezimaltrennzeichen) und Komma (Gruppentrennzeichen) enthalten. Plus und Minus dürfen dabei nur am Anfang der Zeichenkette vorkommen.
- Das Format muss für jede Ziffer der Zeichenkette mindestens einmal das Format-Element Neun oder Null enthalten, siehe hierzu auch [Abschnitt 2.6.2, Numerische Format-Modelle](#). Beinhaltet die Zeichenkette ein Dezimaltrennzeichen, so muss auch das Format ein passendes Dezimaltrennzeichen (D oder .) enthalten.
- Wird keine Zeichenketten für **string** übergeben, so wird der entsprechende Wert implizit konvertiert.
- Der Datentyp des Funktionsergebnisses richtet sich nach dem Format und ist in der Regel ein **DECIMAL**-Typ. Falls kein Format angegeben wird, ist der Ergebnistyp **DECIMAL(1, 0)** für boolesche Werte und ansonsten immer **DOUBLE**.

Beispiel(e)

```
SELECT TO_NUMBER('+123') TO_NUMBER1,
       TO_NUMBER('-123.45', '99999.999') TO_NUMBER2;
TO_NUMBER1          TO_NUMBER2
-----
123      -123.450
```

TO_TIMESTAMP

Zweck

Wandelt eine Zeichenkette in einen Zeitstempel um.

Syntax

to_timestamp::=



Anmerkung(en)

- Wird kein Format angegeben, so wird das Standardformat verwendet, das im Session-Parameter [NLS_TIMESTAMP_FORMAT](#) definiert ist.
- Welche Formate verwendbar sind, finden Sie in [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#).
- Wie der Zeitstempel ausgegeben wird, bestimmt der Session-Parameter [NLS_TIMESTAMP_FORMAT](#).

Beispiel(e)

```

SELECT TO_TIMESTAMP('1999-12-31 23:59:00') TO_TIMESTAMP;
TO_TIMESTAMP
-----
1999-12-31 23:59:00.000000

SELECT TO_TIMESTAMP('23:59:00 31-12-1999',
                   'HH24:MI:SS DD-MM-YYYY') TO_TIMESTAMP;
TO_TIMESTAMP
-----
1999-12-31 23:59:00.000000
  
```

[TO_YMINTERVAL](#)

Zweck

Wandelt eine Zeichenkette in ein Intervall um (INTERVAL YEAR TO MONTH).

Syntax

to_yminterval::=



Anmerkung(en)

- Die Zeichenkette ist stets vom Format [+ | -]YY-MM. Gültige Werte für Jahre (YY) sind Ganzzahlen von 0 bis 999999999, für Monate (MM) von 0 bis 11.
- Siehe auch [TO_DSINTERVAL](#), [NUMTODSINTERVAL](#) und [NUMTOYMINTEGRAL](#).

Beispiel(e)

```

SELECT TO_YMINTERVAL('3-11') TO_YMINTERVAL;
  
```

```
TO_YMINTERVAL
-----
+000000003-11
```

TRANSLATE

Zweck

Ersetzt in der Zeichenkette `expr` die Zeichen aus `from_string` durch die korrespondierenden Zeichen aus `to_string`.

Syntax

`translate ::=`

$$\rightarrow \boxed{\text{TRANSLATE}} \rightarrow (\rightarrow \text{expr} \rightarrow , \rightarrow \text{from_string} \rightarrow , \rightarrow \text{to_string} \rightarrow) \rightarrow$$

Anmerkung(en)

- Diejenigen Zeichen in `expr`, die nicht in `from_string` vorkommen, werden nicht ersetzt.
- Falls der `from_string` länger ist als der `to_string`, so werden die entsprechenden Zeichen gelöscht und nicht ersetzt.
- Falls der `to_string` länger ist als der `from_string`, so werden die entsprechenden Zeichen bei der Ersetzung ignoriert.
- Ist einer der Parameter ein leerer String, so ist das Ergebnis NULL.

Beispiel(e)

```
SELECT TRANSLATE( 'abcd' , 'abc' , 'xy' ) TRANSLATE;
TRANSLATE
-----
xyd
```

TRIM

Zweck

TRIM entfernt sowohl von der linken als auch von der rechten Grenze der Zeichenkette `string` alle im Ausdruck `trim_string` angegebenen Zeichen.

Syntax

`trim ::=`

$$\rightarrow \boxed{\text{TRIM}} \rightarrow (\rightarrow \text{string} \rightarrow , \rightarrow \text{trim_string} \rightarrow) \rightarrow$$

Anmerkung(en)

- Falls `trim_string` nicht angegeben ist, werden Leerzeichen entfernt.

Beispiel(e)

```
SELECT TRIM( 'abcdef' , 'acf' ) "TRIM";
-----  
bcde
```

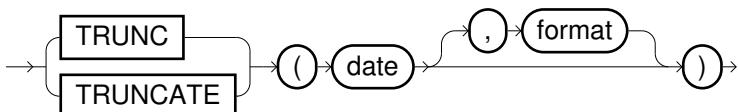
TRUNC[ATE] (datetime)

Zweck

Liefert ein Datum bzw. einen Zeitstempel, das je nach Formatangabe abgeschnitten ist. Somit verhält sich `TRUNC(datetime)` genauso wie [ROUND\(datetime\)](#) mit dem Unterschied, dass bei `TRUNC` stets abgerundet wird.

Syntax

`trunc(datetime) ::=`



Anmerkung(en)

- Für das Format können 'YYYY', 'MM', 'DD', 'HH', 'MI', 'SS' für Jahr, Monat, Tag, Stunde, Minute und Sekunde verwendet werden.

CC, SCC	Jahrhundert
YYYY, SYYY, YEAR, SYEAR,	Jahr
YY, YY, Y	
IYYY, IYY, IY, I	Jahr nach internationaler Norm ISO 8601
Q	Quartal
MONTH, MON, MM, RM	Monat
WW	Der gleiche Wochentag wie der erste Tag des Jahres
IW	Der gleiche Wochentag wie der erste Tag des ISO-Jahres
W	Der gleiche Wochentag wie der erste Tag des Monats
DDD, DD, J	Tag
DAY, DY, D	Erster Tag der Woche. Der erste Tag einer Woche wird über den Parameter NLS_FIRST_DAY_OF_WEEK definiert (siehe ALTER SESSION und ALTER SYSTEM).
HH, HH24, HH12	Stunde
MI	Minute
SS	Sekunden
• Fast die gleiche Funktionalität bietet die PostgreSQL-kompatible Funktion DATE_TRUNC .	
• Falls kein Format angegeben ist, so wird der Wert auf Tage abgerundet.	
• Für den Datentyp <code>TIMESTAMP WITH LOCAL TIME ZONE</code> wird diese Funktion in der Session-Zeitzone berechnet.	

Beispiel(e)

```
SELECT TRUNC(DATE '2006-12-31', 'MM') TRUNC;  
  
TRUNC  
-----  
2006-12-01  
  
SELECT TRUNC(TIMESTAMP '2006-12-31 23:59:59', 'MI') TRUNC;  
  
TRUNC  
-----  
2006-12-31 23:59:00.000000
```

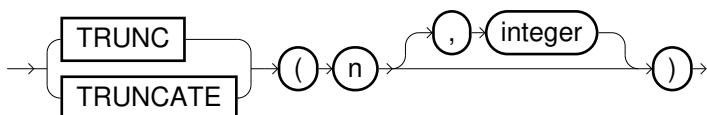
TRUNC[ATE] (number)

Zweck

Schneidet die Zahl n auf integer Stellen hinter dem Komma ab.

Syntax

trunc (number)::=



Anmerkung(en)

- Falls das zweite Argument nicht angegeben wird, so wird auf eine ganze Zahl abgeschnitten.
- Falls das zweite Argument negativ ist, so wird auf integer Stellen vor dem Komma abgeschnitten.

Beispiel(e)

```
SELECT TRUNC(123.456,2) TRUNC;  
  
TRUNC  
-----  
123.45
```

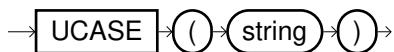
UCASE

Zweck

Wandelt die angegebene Zeichenkette in Großbuchstaben um.

Syntax

ucase::=



Anmerkung(en)

- UCASE ist ein Alias für [UPPER](#).

Beispiel(e)

```
SELECT UCASE( 'AbCdEf' )  UCASE;  
  
UCASE  
-----  
ABCDEF
```

UNICODE

Zweck

Liefert den numerischen Unicode-Wert eines Zeichens.

Syntax

unicode ::=



Anmerkung(en)

- Siehe auch [UNICODECHR](#).

Beispiel(e)

```
SELECT UNICODE( 'ä' )  UNICODE;  
  
UNICODE  
-----  
228
```

UNICODECHR

Zweck

Liefert das Unicode-Zeichen, das dem numerischen Wert n entspricht.

Syntax

unicodechr ::=



Anmerkung(en)

- Die Zahl n muss zwischen 0 und 2097151 sein.
- UNICODECHR (0) liefert NULL zurück.
- Siehe auch [UNICODE](#).

Beispiel(e)

```
SELECT UNICODECHR( 252 )  UNICODECHR ;  
  
UNICODECHR  
-----  
ü
```

UPPER

Zweck

Wandelt die angegebene Zeichenkette in Großbuchstaben um.

Syntax

upper ::=



Anmerkung(en)

- UPPER ist ein Alias für [UCASE](#).

Beispiel(e)

```
SELECT UPPER( 'AbCdEf' )  UPPER ;  
  
UPPER  
-----  
ABCDEF
```

USER

Zweck

Liefert den aktuellen Benutzer zurück.

Syntax

user ::=

→ **USER** →

Anmerkung(en)

- Diese Wert ist äquivalent zu [CURRENT_USER](#).

Beispiel(e)

```
SELECT USER;
USER
-----
SYS
```

VALUE2PROC

Zweck

Funktion, die für einen Wert einen zugehörigen Datenbankknoten zurückliefert. Diese Zuordnung entspricht der Datenverteilung, die über ein **DISTRIBUTE BY** über diesen Wert erzielt werden würde.

Syntax

value2proc ::=

→ **VALUE2PROC** → (→ **expr** →) →

Anmerkung(en)

- Diese Funktion kann dafür benutzt werden, die konkrete Verteilung der Daten über den Cluster nachvollziehen zu können, welche mittels [ALTER TABLE \(distribution\)](#) definiert wird.
- Es wird ein Integer-Wert zwischen 0 und [NPROC-1](#) zurückgeliefert.
- Bitte beachten Sie in diesem Zusammenhang auch die Funktionen [IPROC](#) und [NPROC](#).

Beispiel(e)

```
SELECT IPROC(),
       c1, VALUE2PROC(c1) V2P_1,
       c2, VALUE2PROC(c2) V2P_2 FROM t;

IPROC C1 V2P_1 C2          V2P_2
----- -- ----- -----
0      1   3     abc      3
1      2   2     abcd     0
2      3   1     abcde    1
3      4   0     abcdef   3
0      5   3     abcdefg  0
1      6   2     abcdefgh 2
2      7   2     abcdefghi 1
3      8   1     abcdefghij 1
```

VAR_POP

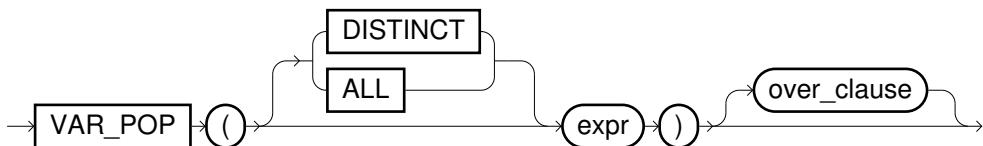
Zweck

Liefert die Varianz innerhalb einer Population. Dies entspricht der folgenden Formel:

$$\text{VAR_POP(expr)} = \frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n}$$

Syntax

`var_pop ::=`



Anmerkung(en)

- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```

SELECT VAR_POP(salary) AS VAR_POP FROM staff;
VAR_POP
-----
432312500

```

VAR_SAMP

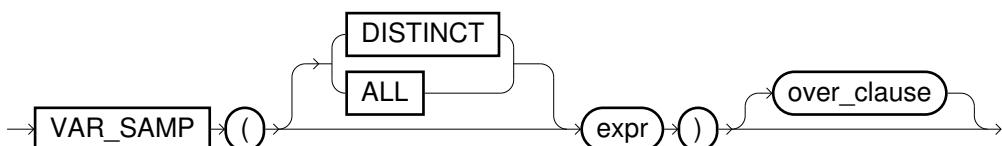
Zweck

Liefert die Varianz innerhalb einer Stichprobe. Dies entspricht der folgenden Formel:

$$\text{VAR_SAMP(expr)} = \frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n - 1}$$

Syntax

`var_samp ::=`



Anmerkung(en)

- VAR_SAMP ist identisch zur Funktion **VARIANCE**. Falls allerdings die Stichprobe nur ein Element umfasst, ist das Ergebnis NULL anstatt 0.
- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```
SELECT VAR_SAMP(salary) AS VAR_SAMP FROM staff WHERE age between 20 and 30;
VAR_SAMP
-----
364800000
```

VARIANCE

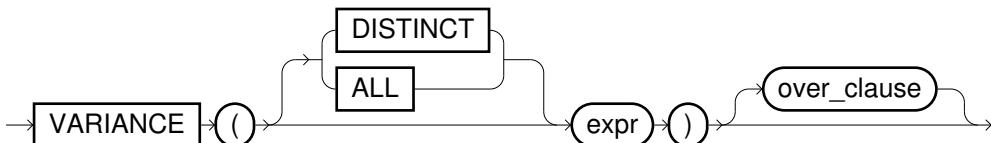
Zweck

Liefert die Varianz innerhalb einer Stichprobe. Dies entspricht der folgenden Formel:

$$\text{VARIANCE(expr)} = \frac{\sum_{i=1}^n (\text{expr}_i - \bar{\text{expr}})^2}{n - 1}$$

Syntax

variance ::=



Anmerkung(en)

- Bei Stichproben mit genau einem Element ist das Ergebnis 0.
- Wird ALL oder nichts angegeben, so werden alle Einträge berücksichtigt. Wird DISTINCT angegeben, so werden doppelte Einträge nur einmal berücksichtigt.
- Zur Verwendung als analytische Funktion siehe auch [Abschnitt 2.9.3, Analytische Funktionen](#).

Beispiel(e)

```
SELECT VARIANCE(salary) VARIANCE FROM staff WHERE age between 20 and 30;
VARIANCE
-----
364800000
```

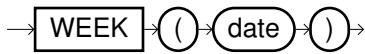
WEEK

Zweck

Liefert die Woche eines Datums (Werte 1-53, definiert in ISO-8601 Standard).

Syntax

week ::=



Anmerkung(en)

- Eine neue Woche beginnt in der Regel am Montag.
- Die erste Woche des Jahres fängt am 1. Januar an, sofern dies ein Montag, Dienstag, Mittwoch oder Donnerstag ist - ansonsten erst am darauf folgenden Montag.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT WEEK(DATE '2012-01-05') WEEK;  
WEEK  
----  
1
```

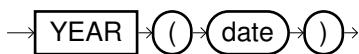
YEAR

Zweck

Liefert das Jahr eines Datums.

Syntax

year ::=



Anmerkung(en)

- Diese Funktion kann im Gegensatz zur Funktion [EXTRACT](#) auch auf Zeichenketten angewendet werden.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT YEAR(DATE '2010-10-20');
```

```
YEAR(
-----
2010
```

YEARS_BETWEEN

Zweck

Liefert die Anzahl an Jahren zwischen zwei Datumswerten.

Syntax

years_between ::=

→ **YEARS_BETWEEN** → (→ **datetime1** → , → **datetime2** →) →

Anmerkung(en)

- Bei der Eingabe eines Zeitstempels geht nur das in ihm enthaltene Datum in die Berechnung ein.
- Falls die Monate und Tage identisch sind, so ist das Ergebnis eine ganze Zahl.
- Falls der erste Datumswert vor dem zweiten Datumswert liegt, so ist das Ergebnis negativ.
- Für den Datentyp TIMESTAMP WITH LOCAL TIME ZONE wird diese Funktion in der Session-Zeitzone berechnet.

Beispiel(e)

```
SELECT YEARS_BETWEEN(DATE '2001-01-01', DATE '2000-06-15') YB1,
       YEARS_BETWEEN(TIMESTAMP '2001-01-01 12:00:00',
                      TIMESTAMP '2000-01-01 00:00:00') YB2;

YB1          YB2
-----
0.5456989247312  1
```

ZEROIFNULL

Zweck

Liefert den Wert 0, falls die Zahl *number* den Wert NULL hat. Andernfalls wird die Zahl *number* zurückgeliefert.

Syntax

zeroifnull ::=

→ **ZEROIFNULL** → (→ **number** →) →

Anmerkung(en)

- Die Funktion ZEROIFNULL entspricht dem CASE-Ausdruck CASE WHEN *number* is NULL THEN 0 ELSE *number* END.

- Siehe auch die Funktion [NULLIFZERO](#).

Beispiel(e)

```
SELECT ZEROIFNULL(NULL) ZIN1, ZEROIFNULL(1) ZIN2;
```

ZIN1	ZIN2
-----	-----
0	1

3. Konzepte

Dieses Kapitel stellt grundsätzliche Konzepte in Exasol vor.

3.1. Transaktions-Management

Durch die Verwendung eines Transaktions-Management-Systems (TMS) gewährleistet Exasol den Mehrbenutzer-Betrieb. Dies bedeutet, dass verschiedene Anfragen von unterschiedlichen Benutzern parallel abgearbeitet werden können. In diesem Kapitel wird das allgemeine Konzept von Transaktionen erklärt sowie Hinweise und Empfehlungen für die Arbeit mit dem TMS von Exasol gegeben.

3.1.1. Allgemeines Konzept

Eine Transaktion besteht aus mehreren SQL-Befehlen. Sie wird entweder mit einem **COMMIT**-Statement abgeschlossen oder durch ein **ROLLBACK**-Statement rückgängig gemacht.

Beispiel

```
-- Transaktion 1
CREATE SCHEMA my_schema;
COMMIT;

-- Transaktion 2
CREATE TABLE t (i DECIMAL);
SELECT * FROM t;
ROLLBACK;

-- Transaktion 3
CREATE TABLE t (i VARCHAR(20));
COMMIT;
```

Ziel eines transaktionsbasierten Systems ist die Bewahrung vollständiger Transactionssicherheit, d.h. jede Transaktion soll ein korrektes Ergebnis liefern und die Datenbank in einem konsistenten Zustand belassen. Um dies zu gewährleisten, müssen Transaktionen die sogenannten ACID-Prinzipien erfüllen:

- Atomicity – Die Transaktion wird entweder vollständig ausgeführt oder gar nicht.
- Consistency – Die Transaktion erhält die interne Konsistenz der Datenbank.
- Isolation – Die Transaktion wird so ausgeführt, als ob sie die einzige Transaktion im System wäre.
- Durability – Alle Veränderungen einer durch ein COMMIT beendeten Transaktion bleiben erhalten.

Um das Einhalten der ACID-Prinzipien zu gewährleisten, unterliegt jede Transaktion einer Überprüfung durch das TMS. Das TMS greift, falls erforderlich selbsttätig ein und behebt Konflikte durch das Erzwingen von Wartezeiten oder durch das Zurückrollen (Rollback) von kollidierenden Transaktionen.



Informationen zu aufgetretenen Transaktionskonflikten finden Sie in den Systemtabellen **EXA_USER_TRANSACTION_CONFLICTS_LAST_DAY** bzw. **EXA_DBATRANSACTION_CONFLICTS**.

Um die Anzahl der kollidierenden Transaktionen so gering wie möglich zu halten, unterstützt Exasol das sog. "MultiCopy-Format". Dies bedeutet, dass von jedem Datenbankobjekt (temporär) mehrere Versionen existieren können. Auf diese Weise kann der Systemdurchsatz (Anzahl der vollständig ausgeführten Transaktionen pro Zeiteinheit) gegenüber von Datenbanken im SingleCopy-Format erheblich gesteigert werden.

Die einzelnen Transaktionen werden vom TMS durch ein Sperrverfahren voneinander isoliert. Die Granularität bei diesem Sperrverfahren umfasst dabei immer ein ganzes Datenbankobjekt wie z.B. ein Schema oder eine Tabelle. Dies bedeutet, dass zwei Transaktionen z.B. nicht gleichzeitig verschiedene Zeilen einer Tabelle aktualisieren können.

Für den Anwender entsteht durch das TMS für jede gestartete Transaktion eine der folgenden Szenarien:

1. Die Transaktion wird bis zum Ende ausgeführt.
2. Die Transaktion wird bis zum Ende ausgeführt, es treten aber Wartezeiten auf, da zwischenzeitlich auf die Beendigung anderer Transaktionen gewartet werden muss.
3. Die Transaktion kann aufgrund von Kollisionen mit anderen Transaktionen nicht beendet werden und wird zurückgerollt. In diesem Fall kann der Anwender die Transaktion zu einem späteren Zeitpunkt wiederholen.

3.1.2. Unterschiede zu anderen Systemen

Einige andere Datenbanksysteme haben das Transaktionsmodell nur partiell implementiert und verbergen teilweise Transaktionen vor dem Nutzer. So werden z.B. Schema-Statements (z.B. "CREATE SCHEMA", "CREATE TABLE") unmittelbar vom System persistent in der Datenbank gespeichert.

Dies verringert bei gleichzeitiger Ausführung von Transaktionen mit Schema-Statements die Kollisionsgefahr, hat aber den Nachteil, dass z.B. ein "ROLLBACK" die durchgeführten Schema-Statements nicht wieder rückgängig machen kann.

Das TMS in Exasol verbirgt im Gegensatz hierzu keine Transaktionen vor dem Nutzer und speichert keine Statements automatisch persistent in der Datenbank.

3.1.3. Empfehlungen für den Anwender

Um bei parallelem Zugriff von sehr vielen Benutzern die Gefahr von Transaktionskonflikten möglichst gering zu halten, ist in den Schnittstellen zu Exasol (EXAplus sowie die Treiber) die Option "Autocommit=ON" standardmäßig eingestellt. Im AUTOCOMMIT-Modus werden erfolgreich abgeschlossene SQL-Kommandos automatisch persistent gespeichert.

Für performance-kritische Teile kann es allerdings sinnvoll sein, nicht nach jedem SQL-Befehl ein COMMIT durchzuführen. Dies gilt insbesondere, wenn in Skripten Zwischentabellen berechnet werden, die nicht persistent gehalten werden sollen. Daher hat der Anwender in EXAplus die Möglichkeit, diese Option mit dem Kommando "SET AUTOCOMMIT OFF" auszuschalten.



AUTOCOMMIT = OFF bewirkt ein erhöhtes Kollisionsrisiko und kann zu Behinderungen anderer Anwender führen.

Falls der Autocommit-Modus ausgeschaltet wird, so empfiehlt sich die Option "-x" in der EXAplus-Konsole. Dies bewirkt einen Abbruch des SQL-Skripts bei Auftreten eines Fehlers, insbesondere bei einem automatischen Rollback nach einem Transaktionskonflikt. Beim Starten von Batch-Skripten ohne diese Option würde die Abarbeitung der SQL-Befehlsfolge trotz des Fehlers fortgeführt werden, was zu fehlerhaften Resultaten führen könnte und daher vermieden werden sollte.

3.2. Rechteverwaltung

Mit Hilfe der **Data Control Language (DCL)** ist es möglich, die Sicherheit und den Zugriff auf die Datenbank zu steuern. Durch die Verwaltung von Benutzern und Rollen sowie die Gewährung von Privilegien kann feingranular unterschieden werden, wer welche Aktionen in der Datenbank ausführen darf.

Folgende SQL-Befehle sind Bestandteil der DCL:

CREATE USER	Anlegen eines Benutzers
ALTER USER	Ändern des Passworts
DROP USER	Löschen eines Benutzers
CREATE ROLE	Erzeugen einer Rolle
DROP ROLE	Löschen einer Rolle
GRANT	Gewährt Rollen, Systemprivilegien oder Objektpflegeprivilegien
REVOKE	Entzieht Systemprivilegien, Objektpflegeprivilegien oder Rollen
ALTER SCHEMA	Ändert den Besitzer eines Schemas (und damit aller darin enthaltenen Schemaobjekte)

Dieses Kapitel soll lediglich einen Einstieg in die grundlegenden Konzepte zum Thema Rechteverwaltung vermitteln. Ausführliche Details befinden sich in [Anhang B, Details zur Rechteverwaltung](#) sowie in [Abschnitt 2.2.3, Zugriffssteuerung mittels SQL \(DCL\)](#).

3.2.1. User

Für jeden Benutzer, der sich mit der Datenbank verbinden will, muss von einem Administrator ein USER-Account angelegt worden sein (mit dem SQL-Befehl **CREATE USER**). Hierbei erhält der Benutzer ein Passwort, das er jederzeit ändern kann, und mit dem er sich gegenüber der Datenbank authentifiziert.

Die Namenskonventionen für Benutzernamen und Passwörter sind die gleichen wie für SQL-Identifier (Bezeichner für Datenbank-Objekte wie z.B. Tabellennamen, siehe hierzu auch [Abschnitt 2.1.2, SQL-Bezeichner](#)). Allerdings wird hier nicht zwischen Groß- und Kleinschreibung unterschieden.



Benutzernamen und Rollen sind case-insensitiv

Damit ein Benutzer auch Aktionen in der Datenbank ausführen darf, benötigt er entsprechende Privilegien. Diese werden ihm von einem Administrator oder anderen Benutzern gewährt bzw. entzogen. Zum Beispiel muss ein Benutzer das Systemprivileg CREATE SESSION besitzen, um sich mit der Datenbank verbinden zu können. Will man einen Nutzer temporär deaktivieren, so entzieht man ihm dieses Systemprivileg.

In der Datenbank gibt es einen speziellen Benutzer **SYS**, der nicht gelöscht werden kann und über alle Privilegien verfügt.



Initial lautet das SYS-Passwort *exasol*, sollte aber sofort nach der ersten Anmeldung geändert werden, um mögliche Sicherheitsrisiken zu vermeiden.

3.2.2. Rollen

Rollen dienen der Gruppierung von Benutzern und können die Rechteverwaltung erheblich vereinfachen. Sie werden mit dem Befehl **CREATE ROLE** erzeugt.

Einem Benutzer kann eine oder mehrere Rollen durch den SQL-Befehl **GRANT** zugewiesen werden. Dadurch erhält der Benutzer die Rechte der entsprechenden Rollen. Anstatt vielen "ähnlichen" Benutzern die gleichen Privilegien vergeben zu müssen, kann man einfach eine Rolle erzeugen und ihr die entsprechenden Privilegien gewähren. Durch die Zuweisung von Rollen zu Rollen ist sogar ein hierarchischer Aufbau von Privilegien möglich.

Rollen können nicht deaktiviert werden (wie z.B. bei Oracle). Soll die Zuweisung zu einer Rolle wieder rückgängig gemacht werden, so kann man mit dem SQL-Befehl **REVOKE** die Rolle wieder entziehen.

Die Rolle **PUBLIC** hat eine Sonderstellung, da jeder Benutzer diese Rolle automatisch einnimmt. Hierdurch ist es sehr einfach, bestimmte Privilegien allen Benutzern der Datenbank zu gewähren und später wieder zu entziehen. Dies sollte allerdings nur dann geschehen, wenn man sich ganz sicher ist, dass die entsprechenden Rechte unbedenklich sind bzw. die freigegebenen Daten öffentlich einsehbar sein sollen. Die Rolle PUBLIC kann nicht gelöscht werden.

Eine weitere vordefinierte Rolle ist die Rolle **DBA**. Sie steht für Datenbankadministrator und verfügt über alle möglichen Privilegien. Sie sollten diese Rolle nur sehr wenigen Benutzern zuweisen, da diese hierdurch den vollen Zugriff auf die Datenbank bekommen. Die Rolle DBA kann ebenso wie PUBLIC nicht gelöscht werden.

3.2.3. Privilegien

Privilegien steuern den Zugriff auf die Datenbank. Mit den SQL-Befehlen **GRANT** und **REVOKE** werden Privilegien gewährt bzw. entzogen. Es wird zwischen zwei verschiedenen Privilegien-Arten unterschieden:

Systemprivilegien steuern generelle Rechte wie z.B. "Schema anlegen", "Benutzer anlegen" oder "auf beliebige Tabellen zugreifen".

Objektprivilegien erlauben den Zugriff auf einzelne Schemaobjekte (z.B. "SELECT-Zugriff auf die Tabelle t im Schema s"). Als Schemaobjekte werden Tabellen, Views, Funktionen und Skripte bezeichnet. Dabei ist zu beachten, dass jedes Schema und alle darin enthaltenen Schemaobjekte genau einem Benutzer oder einer Rolle gehören. Dieser Benutzer hat bzw. alle Inhaber dieser Rolle haben stets das Recht, diese Objekte zu löschen und anderen Nutzern darauf Zugriff zu erteilen. Wird ein Objektprivileg für ein Schema vergeben, so gilt dieses Privileg für alle im Schema enthaltenen Schemaobjekte.

Eine detaillierte Auflistung aller in Exasol zur Verfügung stehenden Privilegien ist in [Abschnitt B.1, Liste der System- bzw. Objektprivilegien](#) zu finden.

Um einem Benutzer oder einer Rolle Privilegien gewähren oder entziehen zu dürfen, muss der Benutzer selbst über bestimmte Privilegien verfügen. Die genauen Regeln werden in der detaillierten Beschreibung der Befehle GRANT und REVOKE innerhalb der SQL-Referenz erläutert (siehe hierzu auch [Abschnitt 2.2.3, Zugriffssteuerung mittels SQL \(DCL\)](#)).

Man sollte sich stets darüber im Klaren sein, wem man welche Privilegien gewährt und welche Auswirkungen dadurch entstehen können. Die Systemprivilegien **GRANT ANY ROLE**, **GRANT ANY PRIVILEGE** und **ALTER USER** seien hierbei besonders hervorgehoben. Sie erlauben den vollen Zugriff auf die Datenbank und sollten nur wenigen Nutzern gewährt werden. Durch das Systemprivileg GRANT ANY ROLE kann ein Benutzer jedem anderen Benutzer (natürlich auch sich selbst) die Rolle DBA zuweisen und hätte somit Vollzugriff auf die Datenbank. Verfügt der Benutzer über das Systemprivileg GRANT ANY PRIVILEGE, so kann er sich selbst oder jedem anderen das Systemprivileg GRANT ANY ROLE gewähren und somit wiederum Vollzugriff erhalten. Mit dem Privileg ALTER USER ist es möglich, dass Passwort von SYS zu ändern und damit ebenfalls vollen Zugriff zu erhalten.



Die Systemprivilegien GRANT ANY ROLE, GRANT ANY PRIVILEGE und ALTER USER liefern Vollzugriff auf die Datenbank.

3.2.4. Zugriffskontrolle bei SQL-Befehlen

Bei jedem SQL-Statement wird vor dessen Ausführung überprüft, ob der aktuelle Benutzer über die entsprechenden Rechte verfügt und, falls dies nicht der Fall ist, eine Fehlermeldung ausgegeben.

Eine Übersicht über alle von Exasol unterstützten SQL-Befehle sowie die dazu benötigten Privilegien befindet sich in [Abschnitt B.2, Benötigte Privilegien für SQL-Befehle](#). Innerhalb [Kapitel 2, SQL-Referenz](#) wird in der ausführlichen Beschreibung jedes SQL-Befehls diese Information ebenfalls angegeben.

Zugriffsrechte auf einzelne Spalten einer Tabelle oder View werden nicht unterstützt. Soll nur ein Ausschnitt einer Tabelle für bestimmte Benutzer/Rollen sichtbar sein, so kann dies mit Hilfe von Views geschehen, die diesen Teil selektieren. Anstatt Zugriff auf die eigentliche Tabelle zu gewähren, wird dies nur für die erzeugte View erlaubt. Dadurch kann der Zugriffsschutz für einzelne Spalten und/oder Zeilen realisiert werden.

3.2.5. Metainformationen zur Rechteverwaltung

Der Zustand der Rechteverwaltung in der Datenbank lässt sich über eine Vielzahl von **Systemtabellen** abfragen. In ihnen stehen Informationen über die existierenden Benutzer und Rollen sowie deren Rechte. Zudem kann ein Benutzer ermitteln, welche Rollen er einnimmt, auf welche Schemaobjekte er Zugriff hat, welche Privilegien er anderen gewährt hat und welche er selbst besitzt.

Eine Auflistung aller für die Rechteverwaltung relevanten Systemtabellen befindet sich in [Abschnitt B.3, Systemtabellen zur Rechteverwaltung](#).

Wer auf welche Systemtabellen zugreifen darf, ist ebenso durch Privilegien gesteuert. So gibt es einige, bei denen aufgrund sicherheitsrelevanter Informationen nur ein DBA Zugriff hat. Außerdem existieren Systemtabellen, die für alle sichtbar sind, aber nur die individuell erlaubten Informationen auflisten (Beispiel **EXA_ALL_OBJECTS**: alle Schemaobjekte, auf die der Benutzer zugreifen kann).

3.2.6. Rechteverwaltung und Transaktionen

Benutzer, Rollen sowie Privilegien sind genauso transaktionsbasiert wie die Schemaobjekte der Datenbank.

Das bedeutet, dass Änderungen erst dann sichtbar werden, wenn die Transaktion durch ein COMMIT abgeschlossen wird. Weil bei allen SQL-Befehlen lesend auf den eigenen Benutzer zugegriffen wird, um dessen Privilegien abzufragen, sollten DCL-Befehle (Data Control Language) möglichst immer mit eingeschaltetem AUTOCOMMIT durchgeführt werden. Ansonsten steigt die Gefahr von Konflikten zwischen Transaktionen.

Details zu dieser Problematik finden sie in [Abschnitt 3.1, Transaktions-Management](#).

3.2.7. Beispiel zur Rechteverwaltung

Anhand des folgenden Beispiel-Szenarios sollen die grundlegenden Mechanismen der Rechteverwaltung veranschaulicht werden. Folgende Anforderungen sollen realisiert werden:

- Rolle ANALYST: Führt Analysen auf der Datenbank durch und darf daher alle Tabellen lesen und eigene Schemas und Schemaobjekte anlegen
- Rolle HR: Verwaltet das Personal und darf daher die Tabelle STAFF bearbeiten
- Rolle DATA_ADMIN: Gewährt Vollzugriff auf das Schema data
- Tabelle STAFF: Beinhaltet Informationen zum Personal der Firma
- Benutzer SCHNEIDER: Ist ein Administrator und darf alles
- Benutzer SCHMIDT: Ist im Marketing tätig und besitzt die Rolle ANALYST
- Benutzer MAIER: Ist im Personalbüro tätig und besitzt daher die Rolle HR
- Benutzer MUELLER: Ist ein Administrator für das DATA-Schema und kann nur auf dieses zugreifen. Deshalb besitzt er die Rolle DATA_ADMIN

Um das Szenario in der Datenbank zu implementieren, könnte folgendes SQL-Skript benutzt werden:

```
--create table
CREATE SCHEMA infos;
CREATE TABLE infos.staff (id          DECIMAL,
                          last_name  VARCHAR(30),
                          name       VARCHAR(30),
                          salary     DECIMAL);
CREATE SCHEMA data_schema;
```

```
--create roles
CREATE ROLE analyst;
CREATE ROLE hr;
CREATE ROLE data_admin;

--create users
CREATE USER schneider IDENTIFIED BY s56_f;
CREATE USER schmidt IDENTIFIED BY x234aj;
CREATE USER maier IDENTIFIED BY jd89a2;
CREATE USER mueller IDENTIFIED BY lk9a4s;

--for connecting to db
GRANT CREATE SESSION TO schneider, schmidt, maier, mueller;

--grant system privileges to role analyst
GRANT CREATE SCHEMA, CREATE TABLE, SELECT ANY TABLE TO analyst;

--grant object privileges on one table to role hr
GRANT SELECT,UPDATE,DELETE ON TABLE infos.staff TO hr;

--grant system privileges to role data_admin
GRANT CREATE TABLE, CREATE VIEW TO data_admin;

--make data_admin the owner of schema data
ALTER SCHEMA data_schema CHANGE OWNER data_admin;

--grant roles to users
GRANT dba TO schneider;
GRANT analyst TO schmidt;
GRANT hr TO maier;
GRANT data_admin TO mueller;
```

3.3. Prioritäten

3.3.1. Einleitung

Mit Hilfe von Prioritäten können die Ressourcen von Exasol gezielt auf Nutzer bzw. Rollen verteilt werden.

Exasol versucht bereits bei Ausführung einer einzelnen Anfrage, durch interne Parallelisierung (Multithreading) so viel wie möglich HW-Ressourcen (CPU, Hauptspeicher, Netzwerk, I/O) optimal auszunutzen. Da aber nicht alle Ausführungsschritte parallelisierbar sind, kann eine vollständige HW-Auslastung erst bei mehreren parallelen Anfragen erreicht werden. Bei mehr parallelen Anfragen als die Anzahl an Rechnerkernen pro Server, greift das Ressourcen-Management von Exasol ein und sichert wie ein Betriebssystem durch Scheduling eine konstante System-Performance.

Über ein transparentes Zeitscheiben-Modell erreicht das Ressourcen-Management, dass jede der parallelen Queries gleichmäßig Ressourcen erhält, während der Gesamtverbrauch nach oben beschränkt wird. Dabei werden Anfragen regelmäßig kurzzeitig pausiert und wieder gestartet. Ohne solch ein Scheduling könnte es passieren, dass die Gesamtperfomance des Systems stark reduziert wird. Ohne das Setzen von expliziten Prioritäten verarbeitet Exasol alle Anfragen gleichberechtigt. Lediglich kurze Anfragen unter 5 Sekunden erhalten eine etwas höhere Gewichtung, um die Reaktionsgeschwindigkeit des Systems zu maximieren.

Wollen Sie die Verarbeitung vieler paralleler Queries beeinflussen, können Sie dies über die Vergabe von Prioritäten erreichen, welche mit dem [GRANT](#)-Befehl definiert werden können. Nötig sollte dieses Eingreifen nur bei einem hohen Parallelisierungsgrad Ihres Systems sein sowie einer bestimmten Nutzergruppe, die maximale Performance benötigt. Ein Anwendungsbeispiel ist die Anbindung einer Webapplikation, die möglichst schnelle Reaktionszeiten haben soll, während das Gesamtsystem regelmäßig mehr als 10 oder mehr parallele Anfragen verarbeiten muss. Zeitintensive ETL-Prozesse oder Anfragen von unwichtigen Applikationen könnten in diesem Fall weniger Priorität erhalten.



Wertvolle Informationen zur Auslastung Ihrer Datenbank finden Sie übrigens in den statistischen Systemtabellen (z.B. [EXA_USAGE_DAILY](#) und [EXA_SQL_DAILY](#), siehe auch [Abschnitt A.2.3, Statistische Systemtabellen](#)).

3.3.2. Prioritäten in Exasol

Im Default-Zustand haben alle Sessions aller Nutzer die gleiche Priorität. Haben Sie jedoch bestimmte Nutzer bzw. Rollen identifiziert, die höher oder niedriger priorisiert werden sollen, so stehen Ihnen drei Prioritätsgruppen **LOW**, **MEDIUM** und **HIGH** mit den Gewichtungen 1, 3 und 9 zur Auswahl. Diese Gruppen definieren die Ressourcen-Vergabe durch Zusicherung von jeweils mindestens ca. 8%, 23% bzw. 69% der Ressourcen (entsprechend Ihrer Gewichte). Nutzer innerhalb einer Gruppe teilen sich diese Ressourcen gleichmäßig auf, wobei dabei im Wesentlichen die CPU-Nutzung sowie der Hauptspeicherverbrauch berücksichtigt wird. Falls eine Prioritätsgruppe überhaupt nicht vertreten ist, so werden die Ressourcen entsprechend der Gewichte auf die Gruppen verteilt. Gibt es z.B. nur LOW und MEDIUM Sessions, so erhalten diese 25% bzw. 75% der Ressourcen (entsprechend der Gewichte 1:3).

Anmerkungen

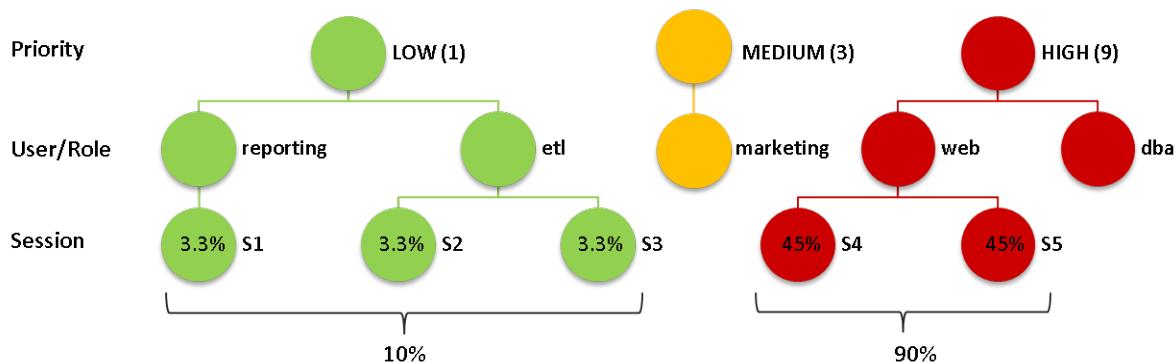
- Nutzer ohne Priorität erhalten per Default die Stufe MEDIUM.
- Ein Nutzer erbt die höchste Priorität seiner Rollen. Eine direkt an den Nutzer vergebene Priorität überschreibt diese jedoch.
- Mehrere Sessions eines Nutzers erhalten alle dieselbe Priorität.
- Beachten Sie, dass sich alle Sessions einer Prioritätsgruppe die Ressourcen gleichmäßig aufteilen. Dadurch kann z.B. jede einzelne von sehr vielen parallelen HIGH-Sessions weniger Ressourcen erhalten als eine einzelne LOW-Session. Auf der anderen Seite ist diese Art der Verteilung von großem Vorteil, da bestimmten Nutzergruppen Mindestressourcen zugesichert werden können (z.B. mindestens 69% für die HIGH-Nutzer).

- Die Ressourcen-Aufteilung lässt sich nicht exakt auf die Ausführungszeit von Anfragen übertragen. Eine identische Anfrage mit doppelt soviel Ressourcen wie eine andere wird deutlich schneller ausgeführt werden, aber nicht exakt im Verhältnis 1:2.
- Ein Nutzer kann in seiner Session mittels **ALTER SESSION** das NICE-Attribut setzen (`ALTER SESSION SET NICE='ON'`). Hiermit wird dem System signalisiert, dass andere Benutzer durch diese Session möglichst wenig gestört/belastet werden sollen. Das Ressourcen-Management teilt das Gewicht des Nutzers in diesem Fall durch die Anzahl der systemweit aktiven Queries. Hierdurch wird auch Prioritätsgruppen-übergreifend erreicht, dass die Session möglichst ressourcen-schonend verarbeitet wird.
- Die Priorität sowie die konkrete Ressourcen-Zuteilung von Nutzern, Rollen bzw. deren Sessions können über diverse Systemtabellen abgefragt werden (u.a. `EXA_ALL_USERS`, `EXA_ALL_ROLES`, `EXA_ALL_SESSIONS`).

3.3.3. Beispiel

Im folgenden Beispiel sind einige Rollen mit unterschiedlicher Priorität erzeugt. In der Grafik sehen Sie, wie sich die Prioritäten auf die Ressourcenverteilung für laufende Sessions auswirkt.

```
GRANT PRIORITY LOW      TO reporting, etl;
GRANT PRIORITY MEDIUM   TO marketing;
GRANT PRIORITY HIGH     TO web, dba;
```



3.4. ETL-Prozesse

3.4.1. Einführung

Unter **ETL**-Prozessen versteht man das Übertragen von Daten aus Quellsystemen in ein Zielsystem:

- **Extract:** Lesen von Daten aus den Quellsystemen
- **Transform:** diverse Datenmodifikationen, Anpassung von Datentypen, Duplikat-Elimination, Aggregation, Vereinheitlichung heterogener Datenstrukturen
- **Load:** Daten in das Zielsystem schreiben

In der Regel handelt es sich bei den Quellsystemen um eine oder mehrere operative Datenbanken, deren Daten in einem Zielsystem zusammengefasst werden sollen. Meist wird zunächst eine Initialladung durchgeführt und später periodisch Daten inkrementell nachgeladen, um das Data-Warehouse auf einem aktuellen Stand zu halten.

ETL oder ELT?

Transformationen, die über Datentypkonvertierungen hinaus gehen, können anstatt in den ETL-Tools auch mittels SQL-Befehlen direkt in der Datenbank durchgeführt werden. Dazu zählen unter anderem Datenänderungen, Aggregatsbildungen oder Schemaanpassungen.

Man kann in diesem Zusammenhang auch von ELT-Prozessen (anstatt ETL) sprechen, da der Transformationsschritt erst nach dem Extract und Load geschieht. Dieses Vorgehen wird für Exasol empfohlen, weil hierdurch die hohe Performance des parallelen Clusters für komplexe Verarbeitungsschritte ausgeschöpft werden kann.

3.4.2. SQL-Befehle IMPORT und EXPORT

Ladeprozesse in Exasol werden mit den Befehlen **IMPORT** und **EXPORT** gesteuert. Eine detaillierte Beschreibung zur Syntax sowie Beispiele finden Sie unter [Abschnitt 2.2.2, Manipulation der Datenbank \(DML\)](#).

Anstatt eines externen Bulk-Loading-Tools ist das Exasol Lade-Framework direkt im Cluster integriert. Hiermit wird durch die Ausnutzung der parallelen Rechenkapazitäten optimale Performance erreicht und zugleich der ganze ETL-Prozess vereinfacht. Denn Sie müssen lediglich einfache SQL Befehle ausführen und keine Client Tools auf mehreren Systemen und Plattformen installieren und pflegen.

Zugangsdaten zu externen Systemen können durch die Verwendung von Verbindungen sehr einfach gekapselt werden (siehe [CREATE CONNECTION](#) in [Abschnitt 2.2.3, Zugriffssteuerung mittels SQL \(DCL\)](#)).

Integration von Datenmanagement-Systemen

Sie können nahezu jedes Datenmanagement-System direkt in die IMPORT und EXPORT Befehle integrieren, indem Sie den entsprechenden JDBC-Treiber in das Exasol Cluster hochladen (über EXAoperation) und die generische JDBC-Schnittstelle für IMPORT/EXPORT nutzen. Und durch den Support von nativen Schnittstellen zu Exasol und Oracle erreichen wir sogar eine noch höhere Ladeperformance.

Datei Up-/Download

Für die reine Verarbeitung von Dateien unterstützt Exasol die beiden Datenformate **CSV** (Comma separated Value) und **FBV** (Fix Block Value), welche weiter unten erklärt werden. Die maximale Performance erreichen Sie unter Verwendung eines HTTP- bzw. FTP-Servers und optionales Aufsplitten der Daten in mehrere Dateien.

Zum Laden/Extrahieren von lokalen Dateien auf Ihrem Computer können Sie ebenfalls die Befehle **IMPORT** und **EXPORT** benutzen, allerdings nur in Verbindung mit EXAplus bzw. des JDBC-Treibers.

In manchen Fällen wird jedoch trotzdem ein externes Bulk-Loading-Werkzeug benötigt, weshalb wir zusätzlich ein Tool namens `exajload` als Teil des JDBC-Treiber-Pakets ausliefern.

3.4.3. Scripting von ETL-Prozessen

IMPORT und **EXPORT** Befehle werden in den meisten Fällen nur ein Teil einer komplexeren Logik sein, um Daten in Exasol zu aktualisieren. Normalerweise sollen alte Daten gleichzeitig gelöscht oder neue und geänderte Daten integriert werden (MERGE), oder sogar noch komplexere Prozesse wie eine Datenbereinigung ausgeführt werden. Um flexible ETL-Prozesse umsetzen zu können, muss auf Exceptions reagiert oder verschiedene Ausführungs-Zweige nach bestimmten Bedingungen gewählt werden.

Deshalb empfehlen wir Ihnen [Abschnitt 3.5, Scripting-Programmierung](#), falls Sie ausgefeilte ETL-Prozesse entwickeln möchten. Hiermit können Sie eine ganze Reihe von SQL-Befehlen innerhalb eines Skripts ausführen, Exception-Handling umsetzen und sogar kleinere Datenmengen direkt in der Skript-Sprache verarbeiten.

3.4.4. Benutzerdefinierter IMPORT mittels UDFs

Sofern die Standard-Schnittstellen - also Dateien, JDBC-Quellen oder nativ Oracle und Exasol - noch nicht ausreichend sind für Ihre Zwecke, können Sie mit Hilfe von UDFs (siehe auch [Abschnitt 3.6, UDF Skripte](#)) benutzerdefinierte ETL-Prozesse programmieren. Wir werden in der Folge erläutern, wie extrem flexibel diese Möglichkeit ist, und wie man fast beliebige Datenquellen und Datenformate an Exasol anbinden kann (u.a. auch Hadoop).

Basis dieser Lösung ist die Erstellung eines UDF-Skriptes mittels **CREATE SCRIPT**, welches Zeilen auswirft (also mit Ausgabe-Typ EMIT). Wird solch ein Skript dann in einem `INSERT INTO <table> (SELECT my_emit_script('param') [FROM ...])` eingebunden, so wird bei der Ausführung das UDF-Skript im Cluster gestartet und die entsprechenden Werte in die Zieltabelle eingefügt.

Innerhalb der Skript-Ausführung können durch die offene Programmierschnittstelle nahezu alle denkbaren Prozesse durchgeführt werden, wie zum Beispiel ein Verbindungsaufbau zu externen Systemen, das Extrahieren von Daten, Datentypkonvertierungen oder sogar komplexere Datentransformationen. Über Parameter können dem Skript alle notwendigen Informationen übergeben werden.

Der große Vorteil dieser Architektur ist die Möglichkeit, in den verschiedenen Programmiersprachen beliebige (Open Source) Bibliotheken einzubinden, die sich zum Beispiel um Datei-Formate kümmern. So können Sie neuartige Dateiformate eines Hadoop-Clusters unmittelbar in Exasol einbinden, ohne auf das nächste Release von Exasol warten zu müssen.

Der Code lässt sich durch Ausnutzung von dynamischen Ein- und Ausgabe-Parametern (siehe auch [Dynamische Eingabe- und Ausgabe-Parameter](#)) sehr generisch gestalten und somit für Tabellen unterschiedlicher Größen und Dateitypen wiederverwenden.

Parallele Lade-Prozesse mit UDFs

Um die optimale Performance von Ladeprozessen zu erreichen, gibt es für UDF ETL-Prozesse die Möglichkeit der Parallelisierung, welche durch Angabe der `GROUP BY` gesteuert wird. Hierdurch wird das UDF-Skript wie auch bei der Nutzung außerhalb von ETL-Prozessen parallel für jede Gruppe aufgerufen.

Das folgende Beispiel veranschaulicht, wie einfach mit Hilfe von UDFs parallel Daten geladen werden können. Dass innere SELECT ruft ein Skript (einmalig) auf, welches sich zu einem Service verbindet und die Liste der JSON-Files zurückliefert, sowie eine Partitions-Id. Das äußere SELECT wiederum verwendet ein UDF-Skript (mit Eingabe-Typ SET), dass dann diese Liste von Dateien nach Partitionen parallelisiert einliest.

```
INSERT INTO my_table (
  SELECT load_my_json(service_address, filepath) EMITS (revenues DECIMAL)
  FROM (SELECT get_json_files('my-service-address', 10))
  GROUP BY partition);
```

IMPORT INTO SCRIPT Syntax

Die Nutzung solcher benutzerdefinierten UDF-Skripte innerhalb von INSERT-Befehlen kann schnell sehr komplex und unübersichtlich werden. Deshalb wurde zur verbesserten Benutzerfreundlichkeit die `IMPORT FROM SCRIPT` Syntax eingeführt (Details zur Syntax siehe [IMPORT](#)).

Die einzige Voraussetzung hierfür ist, dass das UDF-Skript eine bestimmte zusätzliche Methode implementiert, die aus den Import-Informationen ein entsprechendes SELECT-Statement in Form einer Zeichenkette zurückliefert. Im einfachsten Fall wird hierin dieselbe UDF aufgerufen werden, die dann die eigentliche Datenbeladung umsetzt. Wie genau die Methode umgesetzt werden muss, finden Sie in [Abschnitt 3.6.3, Details für die verschiedenen Sprachen](#) für jede Programmiersprache beschrieben.

Somit ist die interne Ausführung zweistufig, also SQL-Text-Generierung mit anschließender Einbettung dieses SELECT-Statements in ein `INSERT INTO SELECT`. Um diese Einbettung mit entsprechender Zieltabelle und -spalten kümmert sich Exasol selbst.

Das nachfolgende einfache Beispiel soll Ihnen das generelle Konzept veranschaulichen. Der benutzerdefinierte Import generiert in diesem Fall schlicht eine Zahlen-Sequenz in einer Integer-Spalte, wobei die Anzahl an Zeilen beim Aufruf angegeben werden kann.

```
CREATE PYTHON SCALAR SCRIPT etl.generate_seq (num INT) EMITS (val INT) AS
def generate_sql_for_import_spec(import_spec):
    if not "RECORDS" in import_spec.parameters:
        raise ValueError('Mandatory parameter RECORDS is missing')
    return "SELECT ETL.GENERATE_SEQ(\"+import_spec.parameters[\"RECORDS\"]+\" )"

def run(ctx):
    for i in range(1, ctx["num"]+1):
        ctx.emit(i)
    /

IMPORT INTO (val int) FROM SCRIPT etl.generate_seq WITH RECORDS='5';

VAL
-----
1
2
3
4
5
```

Im UDF-Skript `generate_seq` wird einerseits die `generate_sql_for_import_spec(import_spec)` definiert, welche ein sehr einfaches SELECT-Statement erzeugt und den Parameter RECORDS weiterreicht. Zum anderen wird im gleichen UDF-Skript auch eine `run()` Funktion implementiert, die sich um die eigentliche Logik der Datenerzeugung kümmert. Natürlich könnte das im IMPORT-Befehl angegebene Skript auch nur das SQL für das SELECT erzeugen (`generate_sql_for_import_spec(import_spec)`) und dabei ganz andere UDF-Skripte einbinden. Bitte beachten Sie grundsätzlich, dass der Aufruf der Skripte im SELECT schemaqualifiziert angegeben werden muss.

Im Beispiel wird auch deutlich, wie der IMPORT-Parameter (siehe WITH-Klausel des `IMPORT` Befehls) an den eigentlichen Skript-Aufruf im SELECT weitergegeben wird. Im Object `import_spec` werden diverse Informationen dem Skript zur Verfügung gestellt, wie z.B. die übergebenen Parameter (WITH-Klausel), die verwendete Connection oder die Spaltennamen der Zieltabelle des IMPORT-Statements.

Mit diesen Informationen kann das Skript sehr dynamisch umgehen und das geeignete SQL generieren, sowie die Interaktion mit dem externen System steuern. Der Name des Metadaten-Objekts und dessen Inhalte kann je nach Programmiersprache variieren, und wird in [Abschnitt 3.6.3, Details für die verschiedenen Sprachen](#) ausführlich spezifiziert.

Zum besseren Verständnis empfehlen wir Ihnen unser Open Source Projekt "Hadoop ETL UDFs", in dem die Anbindung an Hadoop-Systeme mittels IMPORT/EXPORT-Befehlen unter Verwendung von UDF-Skripten implementiert wurde: <https://www.github.com/exasol/hadoop-etl-udfs>

Der IMPORT FROM SCRIPT Befehl hat mehrere Vorteile:

- IMPORT bleibt der zentrale Befehl, um Daten in Exasol zu laden.
- Die Benutzerschnittstelle ist einfach und intuitiv. Es müssen nur die für den IMPORT notwendigen Parameter angegeben werden. Dadurch wird die Komplexität des zugrundeliegenden SQL Befehls, der in realistischen Szenarien häufig durchaus komplex wird, vor dem Nutzer durch Skript-Logik gekapselt.
- IMPORT FROM SCRIPT unterstützt benannte Parameter, wodurch die Lesbarkeit verbessert wird und das Arbeiten mit Parametern erleichtert wird.
- Wie das normale IMPORT kann auch das IMPORT FROM SCRIPT in ein Subselect eingebunden (und somit auch Views), und somit der Daten-Zugriff gekapselt werden.

3.4.5. Benutzerdefinierter EXPORT mittels UDFs



Bitte beachten Sie, dass dieses Feature in Version 6.0 nachträglich als Beta-Version eingeführt wurde und sich die API unter Umständen noch leicht ändern kann.

Analog zum Konzept für benutzerdefinierte IMPORT-Prozesse mittels UDFs können Sie auch benutzerdefinierte EXPORT-Prozesse umsetzen. Sie müssen dafür lediglich ein UDF-Skript erzeugen, welches im Skript-Code eine Verbindung zu einem externen System aufbaut und anschließend die Daten weiterreicht. So könnte z.B. ein schlichtes SELECT my_export_script(...) FROM <table> alle Zeilen zu dem externen System schicken. Empfohlen wird ein SET-Skript und die Steuerung der Parallelisierung mittels GROUP BY, damit der Verbindungs-Overhead möglichst gering bleibt.

```
SELECT my_export_script('<service_address>', filepath)
FROM my_table GROUP BY partition;
```

EXPORT FROM SCRIPT Syntax

Analog zum IMPORT-Befehl können UDFs auch sehr bequem über die EXPORT INTO SCRIPT Syntax verwendet werden (Details zur Syntax siehe [EXPORT](#)).

Auch hier muss das UDF-Skript eine bestimmte zusätzliche Methode implementieren, die aus den Export-Informationen ein entsprechendes SQL Statement in Form einer Zeichenkette zurückliefert. Wie genau die Methode umgesetzt werden muss, finden Sie in [Abschnitt 3.6.3, Details für die verschiedenen Sprachen](#) für jede Programmiersprache beschrieben.

Für ein entsprechendes Beispiel verweisen wir auf das nächste Kapitel, in dem unsere Hadoop Open Source Integration beschrieben wird.

3.4.6. Hadoop und andere Systeme

Die Technologie der UDFs für benutzerdefinierten ETL-Prozesse und die integrierte IMPORT/EXPORT SCRIPT Syntax lässt sich sehr gut nutzen, um diverse externe Systeme an Exasol anzubinden. Gerade durch die Flexibilität und die Mächtigkeit der Skript-Sprachen können Sie selbst bei neuen Dateiformaten oder individuellen Anforderungen an Ihr System die UDFs sehr leicht anpassen.

Wir stellen Ihnen in unserem Open Source Repository (siehe <https://github.com/exasol>) UDF-Skripte zur Verfügung, mit denen Sie externe Datenquellen sehr bequem einbinden können. Laden Sie sich einfach die entsprechenden UDF-Skripte herunter, führen Sie sie auf Ihrer Exasol Datenbank aus und schon können Sie in den meisten Fällen loslegen. Wir würden uns sehr freuen, wenn auch Sie aktiv in unserer Open Source Community mitwirken und unsere Open Source Tools nutzen, erweitern und ergänzen.

Eine der existierenden Skript-Implementierungen ist eine Anbindung an Hadoop (siehe <https://www.github.com/exasol/hadoop-etl-udfs>). Im folgenden Beispiel importieren wir daraus eine Apache HCatalog™ Tabelle unter Nutzung des bereitgestellten UDF-Skripte `import_hcat_table` und `export_hcat_table`. Sie sehen, wie simpel letztendlich die Aufrufe sind.

```
IMPORT INTO my_table_1 FROM SCRIPT etl.import_hcat_table
WITH
  HCAT_DB      = 'default'
  HCAT_TABLE   = 'my_hcat_table_1'
  HCAT_ADDRESS = 'hcatalog-server:50111'
  HDFS_USER    = 'hdfs';

EXPORT my_table_2 INTO SCRIPT etl.export_hcat_table
WITH
  HCAT_DB      = 'default'
  HCAT_TABLE   = 'my_hcat_table_2'
  HCAT_ADDRESS = 'hcatalog-server:50111'
  HDFS_USER    = 'hdfs';
```

3.4.7. Virtual Schemas und ETL

Die Nutzung von virtuellen Schemas kann ebenfalls interessant sein für einfache Beladungs-Prozesse. Ist ein virtuelles Schema erstmal angelegt, so können die darin enthaltenen Tabellen sehr leicht materialisiert werden, durch ein simples `CREATE TABLE materialized_schema.my_table AS SELECT * FROM virtual_schema.my_table`. Natürlich sind auch komplexere Transformationen direkt im Subselect möglich.



Falls Sie das Konzept der virtuellen Schemas noch nicht kennen, empfehlen wir Ihnen [Abschnitt 3.7, Virtuelle Schemas](#).

Falls Sie nicht alle Tabellen materialisieren wollen, können Sie auch Views nutzen, die auf die virtuellen Tabellen verweisen. Somit können Sie von Fall zu Fall unterscheiden, welche Tabellen permanent in Exasol gespeichert werden sollen, um die optimale Performance zu erreichen.

Ein weiterer Vorteil der Verwendung von virtuellen Schemas in ETL-Prozessen ist, dass diese bei neuen oder umbenannten Spalten im Quellsystem überhaupt nicht angepasst werden müssen.

3.4.8. Definition der Dateiformate (CSV/FBV)

Das CSV Datenformat

Da [CSV](#) nicht standardisiert ist, wird hier das von Exasol unterstützte Format definiert.

Folgende Formatierungsregeln für Datensätze sind zu beachten:

Kommentare

Ein Datensatz, dessen erstes Zeichen ein „#“ (Sharp) ist, wird ignoriert. Auf diese Weise können Kommentare, Header und Footer verwendet werden.

Trennung der Felder

Die Felder innerhalb eines Datensatzes werden mit dem Spaltentrennzeichen getrennt (hier Komma).

John, Doe,120 any str., Anytown

Zeilentrennung

Die Datensätze werden mit einem Zeilenumbruchzeichen voneinander getrennt. Jede Zeile entspricht einem Datensatz.

```
John, Doe,120 any str., Anytown, 12a
John, Doe,120 any str., Anytown, 12b
```

Innerhalb eines Feldes gelten folgende Regeln für die Datendarstellung:

Leerzeichen Leerzeichen am Anfang bzw. am Ende eines Feldes können optional über die TRIM-Option im [IMPORT-Befehl](#) abgeschnitten werden. Bei Verwendung von RTRIM wird beispielsweise

```
John , Doe ,120 any str.
```

interpretiert als:

```
John, Doe,120 any str.
```

NULL-Wert NULL wird durch ein leeres Datenfeld dargestellt.

So wird

```
John,,any str.
```

interpretiert als:

```
John,NULL,any str.
```



Bitte beachten Sie, dass innerhalb der Datenbank leere Zeichenketten ebenfalls als NULL interpretiert werden.

Zahlen Zahlen können in Gleitkomma- oder in Exponentialdarstellung verwendet werden. Optional kann ein Format angegeben werden (siehe auch [Abschnitt 2.6.2, Numerische Format-Modelle](#)). Bitte beachten Sie hierbei auch die Session-Einstellungen für die Werte [NLS_NUMERIC_CHARACTERS](#).

Beispiele:

```
John, Doe,120 any str.,123,12.3,-1.23,-1.21E+10,4.1E-12
```

Zeitstempel/Datum Falls kein explizites Format (siehe auch [Abschnitt 2.6.1, Datum/Zeit Format-Modelle](#)) spezifiziert wurde, werden die aktuellen Default-Formate verwendet, also die Session-Parameter [NLS_DATE_FORMAT](#) bzw. [NLS_TIMESTAMP_FORMAT](#). Beachten Sie zusätzlich bei bestimmten Format-Elementen den Session-Parameter [NLS_DATE_LANGUAGE](#).

Boolesche Werte Boolesche Werte sind die Werte TRUE bzw. FALSE. Diese können durch unterschiedliche Werte repräsentiert werden, z.B. 0 für FALSE oder T für TRUE (siehe auch [Boolescher Datentyp](#)).

Spezialzeichen Wenn das Feldtrennzeichen, Feldbegrenzungszeichen oder Zeilenumbruchzeichen innerhalb der Daten vorkommt, muss das betroffene Feld vom Feldbegrenzungszeichen umschlossen werden. Soll das Feldbegrenzungszeichen selbst in den Daten enthalten sein, so muss es zweimal direkt hintereinander geschrieben werden.

Beispiel (Feldtrennzeichen: Komma, Feldbegrenzungszeichen: Anführungszeichen):

```
Conference room 1,"John, " "please" " call me back! ", "
```

wird als ein Datensatz mit drei Feldern gelesen:

Feld #1: Conference room 1

Feld #2: John, "please" call me back!

Feld #3: Leerer String, was in der Datenbank einem NULL-Wert entspricht!

Das gleiche Beispiel ohne Verwendung des Feldbegrenzungszeichens führt zu einem Fehler:

```
Conference room 1,John, "please" call me back!, "
```

Das Fixblock Datenformat (FBV)

Neben dem [CSV](#)-Format unterstützt Exasol auch Textdateien mit einer festen Bytestruktur - das so genannte [FBV](#)-Format. Diese weisen eine fixe Breite pro Spalte auf und sind somit schneller zu parsen. Es wird kein Trennzeichen zwischen den einzelnen Feldern benötigt.

Zur Interpretation der Daten werden folgende Elemente berücksichtigt:

Spaltenbreite	Pro Spalte wird eine feste Anzahl an Bytes verwendet. Ist die Größe des Inhalts kleiner als die Spaltenbreite, so wird der Inhalt mit dem angegebenen Füllzeichen aufgefüllt.
Spaltenausrichtung	Bei Linksausrichtung steht der Inhalt am Anfang der Spalte, die Füllzeichen folgen. Bei Rechtsausrichtung ist der Inhalt am Spaltenende notiert und die Füllzeichen stehen davor.
Zeilentrennung	Am Ende eines Datensatzes folgt optional ein Zeilenumbruch.
NULL-Werte	Um NULL-Werte zu realisieren, wird eine Spalte über die gesamte Breite mit dem Füllzeichen beschrieben.
Zahlen	Zahlen werden in menschenlesbarer Form gespeichert, wobei beim Import sowohl Gleitkomma- als auch Exponentialschreibweise unterstützt werden. Die Spaltenbreite muss jedoch eingehalten werden, eventuell müssen Füllzeichen ergänzt werden.
Explizite Formate	Optional kann ein Format für Zahlen oder Zeitstempel-/Datums-Werte spezifiziert werden. Bitte beachten Sie hierzu Abschnitt 2.6.2, Numerische Format-Modelle und Abschnitt 2.6.1, Datum/Zeit Format-Modelle .

3.5. Scripting-Programmierung

3.5.1. Einleitung

Scripting-Programme im Vergleich zu UDF Skripten

Im Kapitel [Abschnitt 3.6, UDF Skripte](#) wurde bereits eine Programmierschnittstelle beschrieben, mit der Kunden individuelle Auswertungen auf Daten entwickeln können. Diese benutzerdefinierten Funktionen bzw. Skripte dienen der flexiblen Verarbeitung von großen Datenmengen innerhalb eines SELECT-Befehls.

Die in diesem Kapitel erläuterte Scripting-Programmierung stellt hingegen eine Schnittstelle zur Verfügung, mit der verschiedene SQL-Befehle hintereinander ausgeführt und auf Fehler reagiert werden kann. Somit können Sie Steuerungsprozesse innerhalb der Datenbank ausführen (z.B. komplexe Beladungsprozesse) und durch Parametrisierung von Skripten wiederkehrende Aufgaben auf einfache Weise erledigen - wie das Anlegen eines Nutzers mit Passwort und entsprechenden Privilegien.

Zusätzlich können zwar Ergebnistabellen von SQL-Anfragen verarbeitet werden. Allerdings ist ein Scripting-Programm ein sequentielles Programm und läuft auf nur einem Clusterknoten (bis auf die darin ausgeführten SQL-Befehle), weshalb von iterativen Operationen auf großen Datenmengen dringend abzuraten ist. Zu diesem Zweck bietet sich die Verwendung von benutzerdefinierten Funktionen an (siehe [Abschnitt 3.6, UDF Skripte](#)).



- Für Steuerungsprozesse mehrerer SQL-Befehle verwenden Sie Scripting-Programme
- Iterative Operationen auf großen Datenmengen innerhalb einer SQL-Anfrage können Sie mit benutzerdefinierten Funktionen umsetzen (siehe auch [Abschnitt 3.6, UDF Skripte](#)).

Für Scripting-Programme steht aktuell nur die Programmiersprache *Lua* (in Version 5.1) zur Verfügung, welche um ein paar spezifische Funktionalitäten erweitert wurde. Die Erläuterungen zum Lua-Sprachumfang in den nachfolgenden Kapiteln sollten zur Verwendung der Scripting-Sprache in Exasol ausreichen. Falls Sie sich jedoch tiefer in die Grundlagen von *Lua* einarbeiten wollen, empfehlen wir Ihnen die offizielle Dokumentation (siehe auch <http://www.lua.org>).

Erzeugt, ausgeführt und gelöscht werden Scripting-Programme mittels der Befehle **CREATE SCRIPT**, **EXECUTE SCRIPT** und **DROP SCRIPT**. Der Rückgabewert des **EXECUTE SCRIPT** ist entweder eine Zahl (als Rowcount) oder eine Ergebnistabelle.

Beispiel

In diesem Beispiel wird ein Skript erzeugt, das auf einfache Weise alle Tabellen aus einem existierenden Schema in ein neues kopiert.



Der abschließende Schrägstrich ('/') ist nur bei der Verwendung von EXAplus erforderlich.

```
CREATE SCRIPT copy_schema (src_schema, dst_schema) AS

-- define function if anything goes wrong
function cleanup()
    query([[DROP SCHEMA ::s CASCADE]], {s=dst_schema})
    exit()
end

-- first create new schema
query([[CREATE SCHEMA ::s]], {s=dst_schema})
```

```

-- then get all tables of source schema
local success, res = pqquery([[SELECT table_name FROM EXA_ALL_TABLES
                                WHERE table_schema=:s]], {s=src_schema})
if not success then
    cleanup()
end

-- now copy all tables of source schema into destination
for i=1, #res do
    local table_name = res[i][1]
    -- create table identifiers
    local dst_table = join(".", quote(dst_schema), quote(table_name))
    local src_table = join(".", quote(src_schema), quote(table_name))
    -- use protected call of SQL execution including parameters
    local success = pqquery([[CREATE TABLE ::d AS SELECT * FROM ::s]],
                           {d=dst_table, s=src_table})
    if not success then
        cleanup()
    end
end
/

EXECUTE SCRIPT copy_schema ('MY_SCHEMA', 'NEW_SCHEMA');

```

3.5.2. Generelle Elemente der Scripting-Sprache

Lexikalische Konventionen

Die Skriptsprache ist im Gegensatz zum üblichen SQL-Sprachumfang case-sensitiv, d.h. die Groß-/Kleinschreibung muss z.B. bei Variablen-Definitionen beachtet werden. Weiterhin existiert die Einschränkung, dass für Variablen- und Funktionsnamen nur ASCII-Zeichen verwendet werden dürfen. Ein abschließendes Semikolon (;) nach einem Skript-Kommando ist optional.



Skripte sind case-sensitiv, also auch Variablen- und Funktionsnamen



Variablen- und Funktionsnamen dürfen nur aus ASCII-Zeichen bestehen

Kommentare

Es gibt zwei verschiedene Arten von Kommentaren:

- Zeilenkommentare beginnen mit den Zeichen -- und markieren den restlichen Teil der aktuellen Zeile als Kommentar.
- Blockkommentare sind durch die Zeichen /* bzw. */ markiert und können auch über mehrere Zeilen reichen. Alle Zeichen zwischen den Trennsymbolen werden dabei ignoriert.

Beispiele

```

-- This is a single line comment

/*
  This is

```

```
a multiline comment
*/
```

Typen & Werte

Folgende Typen werden in der Skriptsprache unterschieden:

Typ	Wertebereich
nil	nil ist der "unbekannte Typ"
null bzw. NULL	null bzw. NULL stellt die SQL Konstante NULL dar. Diese ist nicht im Lua-Sprachumfang enthalten und wurde von uns hinzugefügt, um Vergleiche auf Ergebnisdaten durchführen oder NULL Werte zurückliefern zu können.
boolean	Boolesche Werte (true, false)
string	Beliebige 8 Bit-Zeichen. Werte vom Typ string werden entweder in einfachen oder doppelten Anführungszeichen ('my_string', "my_string") definiert. Alternativ können Zeichenketten allerdings auch in zweifach eckigen Klammern ([[my_string]]) definiert werden. Diese Schreibweise ist v.a. dann nützlich, wenn im String selbst Anführungszeichen vorkommen und man keine umständlichen Escape-Zeichen verwenden will ('SELECT * FROM "t" WHERE v=\`abc\`;' entspricht "SELECT * FROM \"t\" WHERE v='abc';" oder vereinfacht [[SELECT * FROM "t" WHERE v='abc';]]).
number	Ganz- oder Fließkommazahlen (z.B. 3, 3.1416, 314.16e-2, 0.31416E1)
decimal	Dezimalwerte (z.B. 3, 3.1416)

Bitte beachten Sie, dass der Typ decimal nicht in den Standard-Typen von Lua existiert, sondern ein von Exasol extra eingebauter benutzerdefinierter Typ ist (vom Typ userdata, ähnlich wie der spezielle NULL-Wert). Dieser Dezimaltyp umfasst die folgenden Operatoren und Methoden, um die üblichen mathematischen Operationen und Konvertierungen durchführen zu können:

Konstruktor	
decimal(value [,precision [, scale]])	Der Wert value kann vom Typ string, number oder decimal sein. ☞ Default für Precision und Scale sind (18,0), d.h. decimal(5.1) wird zu dem Wert 5 gerundet!
Operatoren	
+, -, *, /, %	Addition, Subtraktion, Multiplikation, Division und Modulo-Berechnung zweier numerischer Werte. Der Rückgabetyp wird dynamisch ermittelt: decimal oder number
== , <, <=, >, >=	
	Vergleichsoperatoren für numerische Werte. Rückgabetyp: boolean
Methoden	
var:add(), var:sub(), var:mul(), var:mod()	Addition, Subtraktion, Multiplikation und Modulo-Berechnung zweier numerischer Werte. ☞ Es wird hierbei keine neue Variable erzeugt!
var:scale(), var:prec()	Scale bzw. Präzision eines Dezimals. Rückgabetyp: number
var:tonumber()	Konvertierung in einen number-Wert. Rückgabetyp: number

```
var : tostring( ), Konvertierung in eine Zeichenkette. Rückgabetyp: string
tostring(var)
```

Im folgenden finden Sie einige Beispiele zur Verwendung von decimal-Werten:

```
d1 = decimal(10)
d2 = decimal(5.9, 2, 1)
s = d1:scale()      -- s=0
str = tostring(d2)  -- str='5.9'
d1:add(d2)         -- result is 16
```

Einfache Variablen

Skript-Variablen werden dynamisch getypt. Dies bedeutet, dass Variablen keinen Typ besitzen, sondern der ihnen zugewiesene Wert. Eine Zuweisung (Assignment) geschieht über den = Operator.

Die Sichtbarkeit von Variablen ist global und kann durch das Keyword local auf den aktuellen Ausführungsblock eingeschränkt werden.



Wir empfehlen, stets lokale Variablen zu verwenden, um die Variablen-Deklaration explizit sichtbar zu machen.

Beispiele

```
local a = nil      -- nil
local b = false    -- boolean
local c = 1.0      -- number
local d = 'xyz'    -- string
d = 3              -- same variable can be used for different types
local e,f = 1,2    -- two variable assignments at once
g = 0              -- global variable
```

Arrays

Ein Array besteht aus einer einfachen Liste von Werten (`my_array={2,3,1}`). Diese Werte dürfen auch heterogen sein, also mit unterschiedlichen Typen.

Die Elemente des Arrays werden über die Position beginnend mit 1 bestimmt (`my_array[position]`). Die Größe eines Arrays lässt sich über den #‐Operator auslesen (`#my_array`). Im Falle eines nil‐Wertes wird eine Exception geworfen.

Arrays können als Werte wiederum Arrays enthalten, wodurch sich bequem mehrdimensionale Arrays anlegen lassen.

Beispiele

```
local my_array = {'xyz', 1, false}   -- array
local first    = my_array[1]        -- accessing first entry
local size     = #my_array        -- size=3
```

Dictionary Tables

Neben den einfachen Variablen und Arrays können auch Dictionary Tables verwendet werden, welche stets aus einer Sammlung von Schlüssel-Wert Paaren bestehen. Die Schlüssel bzw. Werte können selbst heterogen, also von unterschiedlichen Typen sein.

Die Adressierung der Werte geschieht über die Schlüssel der Dictionary Table, entweder über die Array-Notation (`variable['key']`) oder mittels der Punkt-Notation (`variable.key`).

Mit der Funktion `pairs(t)` kann man leicht über alle Einträge einer Dictionary Table iterieren (`for k,v in pairs(t) do end`).



In der Lua Dokumentation wird kein Unterschied gemacht zwischen Arrays und Dictionary Tables - beide werden als *Table* bezeichnet.

Beispiele

```

local my_contact = {name='support',           -- define key/value pairs
                    phone='unknown'}
local n = my_contact['phone']                -- access method 1
n = my_contact.phone                         -- access method 2
my_contact.phone = '0049911239910'           -- setting single value

-- listing all entries in dictionary
for n,p in pairs(my_contact) do
    output(n..":"..p)
end

-- defining a 2 dimensional table
local my_cube = {{10, 11, 12}, {10.99, 6.99, 100.00}}
local my_value = my_cube[2][3]                 -- -> 100.00

-- defining "column names" for the table
local my_prices = {product_id={10, 11, 12}, price={10.99, 6.99, 100.00}}
local my_product_position = 3
local my_product_id = my_prices.price[my_product_position] -- -> 100.00
local my_price = my_prices.product_id[my_product_position] -- -> 12

```

Ausführungs-Blöcke

Blöcke sind Ausführungseinheiten, die die Sichtbarkeit von lokalen Variablen einschränken. Das Skript an sich bildet den äußersten Ausführungsblock. Darin enthaltene Blöcke werden definiert durch Kontrollstrukturen (siehe nächster Abschnitt) oder Funktionsdeklarationen (siehe Abschnitt [Funktionen](#)).

Explizite Blöcke können mittels `do end` deklariert werden. Sie dienen in erster Linie zur Beschränkung der Sichtbarkeit von lokalen Variablen.

Beispiele

```

-- this is the outermost block
a = 1             -- global variable visible everywhere

if var == false then
    -- the if construct declares a new block
    local b = 2   -- b is only visible inside the if block
    c = 3         -- global visible variable

```

```

end

-- explicit declared block
do
    local d = 4; -- not visible outside this block
end

```

Kontrollstrukturen

Folgende Kontrollstrukturen können verwendet werden:

Element	Syntax
if	<pre> if <condition> then <block> [elseif <condition> then <block>] [else <block>] end </pre>
while	<pre> while <condition> do <block> end </pre>
repeat	<pre> repeat <block> until <condition> </pre>
for	<pre> for <var>=<start>,<end>[,<step>] do <block> end for <var> in <expr> do <block> end </pre>

Anmerkungen

- Die Bedingung `<condition>` wird nur bei `false` und `nil` als `false` ausgewertet - ansonsten immer als `true`. Dies bedeutet insbesondere auch, dass der Wert `0` und ein leerer String als `true` ausgewertet werden!
- Bei der `for`-Schleife werden die Variablen `<start>`, `<end>` und `<step>` nur einmal am Anfang als numerischer Wert ausgewertet. Der Schleifenvariablen `<var>` darf innerhalb von `<block>` kein Wert zugewiesen werden. Wird für `<step>` kein Wert angegeben, so ist die Schrittweite 1.
- Die Schleifen `while`, `repeat` und `for` können mittels `break`-Anweisung aus der aktuellen Schleife springen. Das `break` darf aus syntaktischen Gründen nur am Ende eines Ausführungsblocks stehen. Falls es in der Mitte eines Ausführungsblocks verwendet werden soll, so kann man es in einen expliziten Block einschließen (`do break end`).

Beispiele

```

if var == false
then a = 1
else a = 2
end

```

```

while a <= 6 do
    p = p*2
    a = a+1
end

repeat
    p = p*2
    b = b+1
until b == 6

for i=1,6 do
    if p< 0 then break end
    p = p*2
end

```

Operatoren

Folgende Operatoren stehen in der Skriptsprache zur Verfügung:

Operatoren	Anmerkungen
+, -, *, /, %	Übliche arithmetische Operatoren ☞ Bitte beachten Sie, dass stets mittels Gleitkommazahlenarithmetik gerechnet wird.
^	Potenz ($2^3=8$)
==, ~=	Falls beim Gleichheitsoperator (==) die Typen der Operanden unterschiedlich sind, so wird stets <code>false</code> zurückgegeben! Der Ungleichheitsoperator (~=) ist exakt das Gegenteil des Gleichheitsoperators.
<, <=, >, >=	Vergleichs-Operatoren
and, or, not	<ul style="list-style-type: none"> and liefert den ersten Operanden, falls dieser <code>nil</code> oder <code>false</code> ist - ansonsten den zweiten Operanden or liefert den ersten Operanden, falls dieser nicht <code>nil</code> oder <code>false</code> ist - ansonsten den zweiten Beide Operatoren werten den zweiten Operanden nicht mehr aus, falls dies nicht nötig ist Der not-Operator liefert nur <code>true</code>, falls der Operand <code>nil</code> oder <code>false</code> ist - ansonsten immer <code>false</code>
..	Konkatenationsoperator zwischen Zeichenketten und numerischen Werten.

Falls komplexe Ausdrücke nicht geklammert sind, wird die Auswertungsreihenfolge der Operatoren folgendermaßen festgelegt, von hoher zu niedriger Präzedenz:

1. ^
2. not, - (bei einem Operanden)
3. *, /, %
4. +, -
5. ..
6. <, >, <=, >=, ~=, ==
7. and
8. or

Beispiele

```

local x = 1+5          --> 6
x = 2^5              --> 32

x = 1==1            --> true
x = 1=='1'          --> false
x = 1~='1'          --> true

x = true and 10      --> 10
x = true and nil    --> nil
x = 10 and 20       --> 20
x = false and nil   --> false
x = nil and false   --> nil

x = true or false   --> true
x = 10 or 20        --> 10
x = false or nil    --> nil
x = nil or false    --> false
x = nil or 'a'      --> 'a'

x = not true        --> false
x = not false       --> true
x = not nil         --> true
x = not 10          --> false
x = not 0           --> false

x = 'abc'..'def'   --> 'abcdef'

```

Funktionen

Durch die Verwendung von Funktionen lassen sich Skripte einfach strukturieren.

Syntax

```

function <name> ( [parameter-list] )
  <block>
end

```

Anmerkungen

- Werden einfache Variablen als Funktionsparameter übergeben, so werden sie *per value* übergeben. Dies bedeutet, dass sie innerhalb einer Funktion nicht manipuliert werden können. Arrays und Dictionary Tables werden hingegen *per reference* übergeben, d.h. einzelne Werte können von der Funktion verändert werden. Falls jedoch ein komplett neues Objekts zugewiesen wird, so ändert sich der Original-Parameter nicht.
- Wird eine Funktion mit zu vielen Argumenten aufgerufen, so werden die überzähligen Argumente ignoriert. Bei zu wenigen Argumenten werden die restlichen Parameter mit `nil` initialisiert.
- Mittels `return` kann eine Funktion beendet werden und keinen, einen oder mehrere Rückgabewerte zurück liefern. Das Schlüsselwort `return` darf aus syntaktischen Gründen nur am Ende eines Ausführungsblocks stehen, sofern es einen Wert zurückliefert. Falls es in der Mitte eines Ausführungsblocks verwendet werden soll, so kann man es in einen expliziten Block einschließen (z.B. `do return false end`).
- Funktionen selbst sind Werte und lassen sich in Variablen speichern oder in Parameter-Listen übergeben.

Beispiele

```
function min_max(a,b,c)
    local min,max=a,b
    if a>b then min,max=b,a
    end
    if c>max then max=c
    elseif c<min then min=c
    end
    return min,max
end

local lowest, highest = min_max(3,4,1)
```

ERROR Handling mittels `pcall()` und `error()`

Normalerweise wird ein Skript bei einem beliebigen Fehler beendet. Allerdings gibt es Situationen, bei denen man auf bestimmte Fehler reagieren und gewisse Aktionen durchführen möchte. Hierzu existieren spezielle Funktionen:

`pcall()` Kann zum Schützen von Funktionsaufrufen verwendet werden. Als Parameter werden in diesem Fall der Funktionsname sowie alle Parameter der Funktion übergeben, also z.B. `pcall(my_function,param1,param2)` anstatt `my_function(param1,param2)`. Das *p* in `pcall()` steht für *protected*.

Die Funktion `pcall()` liefert zwei Rückgabewerte zurück:

1. Erfolg der Ausführung: `false` im Falle eines beliebigen Fehlers
2. Ergebnis: das eigentliche Ergebnis, falls kein Fehler aufgetreten ist, ansonsten die Fehlermeldung

Diese Rückgabewerte können nach der Ausführung zu Variablen zugewiesen und anschließend ausgewertet werden (z.B. `success,result=pcall(my_function,param1)`).

`error()` Erzeugt einen Fehler, der eine Funktion oder das Skript selbst abbricht.

Beispiele

```
-- define a function which can throw an error
function divide(v1, v2)
    if v2==0 then
        error()
    else
        return v1/v2
    end
end

-- this command will not abort the script
local success, result=pcall(divide, 1, 0)
if not success then
    result = 0
end
```

3.5.3. Interaktion mit der Datenbank

SQL Befehle ausführen mittels `query()` und `pquery()`

Zur Ausführung von SQL-Befehlen gibt es die Funktionen `query()` und `pquery()`, die als Eingabeparameter einen Wert vom Typ `string` erwarten. Gerade für diese Zeichenketten empfiehlt sich die alternative `string`-Syntax ([[SELECT ...]]), um Probleme mit einfachen oder doppelten Anführungszeichen innerhalb von SQL-Befehlen zu vermeiden. Das in EXAplus obligatorische Semikolon zum Abschließen eines SQL-Befehls ist innerhalb von Skripten in der Regel optional.

Falls während der Ausführung des SQL-Befehls mittels `query()` ein Fehler auftritt, so beendet sich das Skript und liefert die entsprechende Fehlermeldung zurück. Zum geschützten Ausführen von SQL-Befehlen können Sie jedoch entweder die hierfür vorgesehene Funktion `pquery()` oder `pcall()` benutzen (siehe auch Abschnitt [ERROR Handling mittels `pcall\(\)` und `error\(\)`](#)).

Rückgabewerte der `query()`-Funktion

Die `query()`-Funktion liefert diverse Informationen zurück, die vom Typ der Anfrage abhängen:

1. SELECT-Befehle

Rückgabewert ist ein zweidimensionales Array, dessen Werte *read-only* sind. Sollen diese Werte weiterverarbeitet werden, so müssen Sie ein neues zweidimensionales Array erzeugen und die Rückgabedaten kopieren. Allerdings sei hier nochmals betont, dass Skripte für Operationen auf großen Datenmengen nicht geeignet sind.

Die erste Dimension des Arrays sind die Zeilen, die zweite die Spalten. Neben der Adressierung über Nummern können auch Spaltennamen verwendet werden, wobei allerdings auf Groß-/Kleinschreibung geachtet werden muss (z.B. `result[i]["MY_COLUMN"]`).

Die Anzahl der Zeilen lassen sich somit über den #‐Operator (`#result`) bestimmen, die Anzahl der Spalten über den #‐Operator auf die erste Zeile (`#result[1]`).

Weiterhin kann über den Schlüssel `statement_text` der ausgeführte SQL Text ausgelesen werden.

2. Sonstige Befehle

Rückgabewert ist eine Dictionary Table, die folgende Schlüssel beinhaltet:

- `rows_inserted`: Anzahl der eingefügten Zeilen
- `rows_updated`: Anzahl der aktualisierten Zeilen
- `rows_deleted`: Anzahl der gelöschten Zeilen
- `rows_affected`: Summe der anderen drei Werte
- `statement_text`: SQL Text des ausgeführten Statements
- `statement_id`: Statement Id in der Session als string Wert

Bei den Befehlen `IMPORT` und `EXPORT` werden noch folgende zusätzliche Schlüssel gesetzt:

- `etl_rows_written`: Anzahl eingefügter Zeilen
- `etl_rows_with_error`: Anzahl fehlerhafter Zeilen (nur sinnvoll bei Nutzung der `error clause`)
- `etl_rows_read`: Anzahl gelesener Zeilen (entspricht der Summe der beiden anderen Werte)

Rückgabewerte der `pquery()`-Funktion

Die `pquery()`-Funktion liefert analog zu `pcall()` zwei Werte zurück:

1. Erfolg der Ausführung: `false` im Falle eines beliebigen Fehlers

2. Eigentliches Ergebnis: das Ergebnis analog wie bei `query()`, falls kein Fehler aufgetreten ist. Ansonsten eine Dictionary Table mit den Schlüsseln `error_message` (Fehler-Text) sowie `error_code` (Fehler-Code). In beiden Fällen kann über den Schlüssel `statement_text` der ausgeführte SQL Text ausgelesen werden.



Das Ergebnis der Funktionen `query()` bzw. `pquery()` ist stets read-only.

Parametrisierte SQL-Befehle

Mittels parametrisierten SQL-Befehlen ist es möglich, dynamisch SQL auszuführen, ohne den SQL-Text neu erzeugen zu müssen. Beispielsweise könnte in einem Skript ein Befehl zur Erzeugung einer Tabelle mehrmals mit verschiedenen Tabellennamen aufgerufen werden. In diesem Fall könnte der Tabellenname in SQL als Parameter verwendet werden.

Allerdings werden solche Parameter nicht einfach durch Textersetzung gesetzt, sondern entweder als Bezeichner oder Wert ausgewertet, bevor sie in den Text eingefügt werden. Zweck dieser Vorgehensweise ist die Vermeidung von Sicherheitslücken durch *SQL Injection*.

Folgende Arten von Parametern können innerhalb von SQL-Befehlen verwendet werden:

- `:variable` Wertet den Inhalt der Variablen als konstanten Wert aus, z.B. bei einem Filter auf eine Spalte (`query([[SELECT * FROM t WHERE i=:v]], {v=1})`).
- `::variable` Wertet den Inhalt der Variablen als Bezeichner aus (z.B. Tabellennamen). Begrenzte Bezeichner müssen in der Zeichenkette die Anführungszeichen enthalten (z.B. `table_name=["t"]`).

Werden in einer `query()` - bzw. `pquery()` -Funktion Variablen verwendet, so muss im zweiten Funktionsparameter eine Dictionary Table angegeben werden, die die Variablen spezifizieren. Darin können entweder konstante Werte oder vorher definierte Variablen spezifiziert werden (z.B. `query([[SELECT * FROM t WHERE ::c=:v]], {c=column_name, v=1})`). Eine direkte Verwendung von globalen Variablen innerhalb der `query()` - bzw. `pquery()` -Funktion ist nicht möglich.

Wird kein zweiter Funktionsparameter angegeben, so wird der SQL-Text uninterpretiert ausgeführt. Dies ist z.B. dann wichtig, wenn Skripte mittels `query()` bzw. `pquery()` angelegt werden sollen, die im Text wiederum solche Variablen verwenden.

Beispiele

```
/*
 *  just executing simple SQL statements
 */
query([[CREATE TABLE t(c CHAR(1))]])      -- creates table T
query([[CREATE TABLE "t"(c CHAR(1))]])    -- creates table t
local my_sql_text = [[INSERT INTO "T" VALUES 'a','b','c','d']]
query(my_sql_text)

/*
 *  using result tables of a query by concatenating
 *  all table entries into one result string
 */
local res_table = query([[SELECT c FROM t]])
local first_element = res_table[1][1]  -- access first row of first column
local row_size = #res_table           -- row_size=4
if row_size==0 then
    col_size = 0
else
```

```

        col_size=#res_table[1]          -- col_size=1
end
local concat_str = ''
for col=1, col_size do
    for row=1, row_size do
        concat_str = concat_str..res_table[row][col]  --> 'abcd'
    end
end

/*
 *   using result information about manipulated rows
 */
local res = query([[MERGE INTO staff s USING update_table u ON (s.id=u.id)
WHEN MATCHED THEN UPDATE SET s.salary=u.salary DELETE WHERE u.delete_flag
WHEN NOT MATCHED THEN INSERT VALUES (u.id,u.salary)]])
local i, u, d = res.rows_inserted, res.rows_updated, res.rows_deleted

/*
 *   using pquery to avoid abort of script
 */
local success, result = pquery([[DROP USER my_user]])
if not success then
    -- accessing error message
    local error_txt = 'Could not drop user. Error: '..result.error_message
    query([[INSERT INTO my_errors VALUES (CURRENT_TIMESTAMP, :err)]],
          {err=error_txt})
end

/*
 *   now using variables inside the SQL statements
 *   and create 5 tables at once
 */
for i=1,5 do
    local tmp_table_name = 'TABLE_'.i
    query([[CREATE TABLE ::t (c CHAR(1))]], {t=tmp_table_name})
end

-- distribute 3 values to the first 3 tables created before
local values = {'x', 'y', 'z'}
for i=1,#values do
    local table_name='TABLE_'.i
    query([[INSERT INTO ::t VALUES :v]], {t=table_name,v=values[i]})
end

-- using both types of variables at once
query([[SELECT * FROM t WHERE ::x=:y]], {x='c',y='a'}))

```

Skript-Parameter

Um Skripten Parameter übergeben zu können, werden diese bei der Skripterzeugung ([CREATE SCRIPT](#)) einfach als Liste spezifiziert. Die Variablen benötigen auch hier keinerlei Angabe von Typen. Einfache Variablen bestehen daher lediglich aus einem Bezeichner, der wie auch innerhalb des Skripts case-sensitiv ist.

Soll ein Parameter ein Array sein, so kann dies über das Schlüsselwort ARRAY deklariert werden. Beim Aufruf des Skriptes wird das Array dann über das Konstrukt ARRAY(value1, value2, ...) übergeben. Dadurch ist es auch einfach möglich, dynamische Listen zu übergeben. Z.B. könnte ein Skript für eine Liste von Nutzernamen ein Systemprivileg vergeben - unabhängig, wie lang die Liste ist.

Beispiele

```
-- using script parameter for table name
create script create_single_table (table_name) as
    query([[CREATE TABLE ::t (i INT)]],{t=table_name})
/

-- using ARRAY construct for multiple input
create script create_multiple_tables (ARRAY table_names) as
    for i=1,#table_names do
        query([[CREATE TABLE ::t (i INT)]],{t=table_names[i]})
    end
/
EXECUTE SCRIPT create_single_table ('t1');
EXECUTE SCRIPT create_multiple_tables (ARRAY('t2','t3','t4'));
SELECT * FROM CAT;

TABLE_NAME TABLE_TYPE
-----
T1          TABLE
T2          TABLE
T3          TABLE
T4          TABLE
```

IMPORT von externen Skripten

Damit Funktionen von externen Skripten benutzt werden können, existiert die Möglichkeit, andere Skripte über die `import()`-Funktion einzubinden. Diese Funktion führt das eingebundene Skript zur Initialisierung aus und stellt alle darin definierten globalen Variablen und Funktionen in einen Namensraum, der über den Skriptnamen adressiert werden kann. Alternativ kann zur Einführung eines Alias ein String als zweiter Parameter angegeben werden.



Beim Einbinden eines Skriptes mittels der `import()`-Funktion wird das eingebundene Skript ausgeführt und alle Funktion sowie globalen Variablen im entsprechenden Namensraum zur Verfügung gestellt.

Im Beispiel unten wird das Skript `other_script` erzeugt, welches eine Funktion `min_max()` definiert. Diese kann nun im zweiten Skript `my_script` nach Einbinden mittels `import("schema1.other_script", "my_alias")` über die Adressierung `my_alias.min_max()` benutzt werden.

Soll ein Skript importiert werden, das Parameter definiert hat, so müssen diese ebenfalls als Parameter der `import()`-Funktion angegeben werden (am Ende, also nach dem optionalen Alias). Hierbei können den Parametern selbst Werte oder Variablen sein.

Beispiele

```
CREATE SCRIPT schema1.other_script AS
    function min_max(a,b,c)
        local min,max=a,b
        if a>b then min,max=b,a
```

```

    end
    if c>max then max=c
    elseif c<min then min=c
    end
    return min,max
end
/

CREATE SCRIPT schema2.my_script (param1, param2, param3) AS
import('schema1.other_script', 'my_alias')
-- accessing external function through alias
local lowest, highest = my_alias.min_max(param1, param2, param3)
output(lowest...',..highest)
/

```

Rückgabewerte eines Skriptes

Zur Rückgabe von Werten eines Skripts existiert die Funktion `exit()`. Diese beendet das aktuelle Skript und liefert den Wert des angegebenen Parameters zurück. Die `exit()`-Funktion kann an jeder Stelle im Skript stehen

Welche Werte ein Skript überhaupt zurückliefern kann, wird innerhalb des Befehls `CREATE SCRIPT` spezifiziert:

1. Option RETURNS ROWCOUNT bzw. keine Angabe einer Option

In diesem Fall ist der Rückgabewert der Wert, der von den Treibern als *Rowcount* zurückgeliefert wird.

Wird kein Wert bei `exit()` angegeben bzw. wird das Skript gar nicht explizit beendet, so wird der Wert 0 zurückgeliefert. Wird das Ergebnis einer SQL-Anfrage mittels `query()` bzw. `pquery()` direkt zurückgeliefert (z.B. `exit(query([[INSERT INTO...]]))`), so wird der Rowcount dieser Query weitergeleitet (funktioniert nicht für eine `SELECT`-Anfrage). Alternativ kann man den Rowcount explizit mittels einer Dictionary Table mit Schlüssel `rows_affected` (z.B. `exit({rows_affected=30})`) angeben.

2. Option RETURNS TABLE

Bei Angabe der `RETURNS TABLE` Option ist das Ergebnis einer Skriptausführung eine Tabelle.

Wird kein Wert bei `exit()` angegeben bzw. wird das Skript gar nicht explizit beendet, so ist das Ergebnis eine leere Tabelle. Wird das Ergebnis einer SQL-Anfrage mittels `query()` bzw. `pquery()` direkt zurückgeliefert (z.B. `exit(query([[SELECT...]]))`), so wird die Ergebnistabelle dieser Query weitergeleitet (funktioniert allerdings nur für `SELECT`-Anfragen).

Ebenfalls möglich ist die Rückgabe eines zweidimensionalen Arrays. Allerdings müssen in diesem Fall die Spaltentypen der Ergebnistabelle als zweiter Rückgabewert spezifiziert werden (siehe Beispiel - analog zur Spaltendefinition eines `CREATE TABLE`-Befehls).

 Die Ergebnistabelle eines Skriptes kann in anderen Skripten weiterverarbeitet werden. Jedoch ist eine persistente Speicherung dieser Tabelle in der Datenbank über SQL nicht möglich (wie im `CREATE TABLE <table> AS SELECT...`).

 Die Angabe der `WITH OUTPUT` Option im `EXECUTE SCRIPT` Befehl überschreibt die `RETURNS` Option, die beim Anlegen des Skriptes definiert wurde (siehe Abschnitt [Debug-Output](#)).

Beispiele

```
-- script which returns rowcount
CREATE SCRIPT script_1 AS
    function my_function(a, b)
        if a == true then
            exit()          -- exit whole script
        else
            return b, b+1 -- returning result of function
        end
    end
    local x, y = my_function(false, 5) --> y=6
    exit({rows_affected=y}) -- return rows_affected
/

-- script which returns result table of query
CREATE SCRIPT script_2 RETURNS TABLE AS
    exit(query([[SELECT * FROM DUAL]]))
/


-- return explicitly created table
CREATE SCRIPT script_3 RETURNS TABLE AS
    local result_table = {{decimal(1),"abc",true},
                          {decimal(2),"xyz",false},
                          {decimal(3),nil,nil}}
    exit(result_table, "i int, c char(3), b bool")
/
EXECUTE SCRIPT script_3;

I          C   B
-----
1 abc TRUE
2 xyz FALSE
3
```

Verfügbare Metadaten

Innerhalb jeden Skripts stehen diverse Metadaten als globale Variablen zur Verfügung, mit denen die Skriptsteuerung verfeinert werden kann.

exa.meta.database_name	Datenbankname
exa.meta.database_version	Datenbankversion
exa.meta.script_language	Name und Version der Sprache
exa.meta.script_name	Name des Skripts
exa.meta.script_schema	Schema des Skripts
exa.meta.script_code	Text des Skripts
exa.meta.current_user	Identisch zur Funktion CURRENT_USER
exa.meta.session_id	Session-Id
exa.meta.statement_id	Statement-Id des EXECUTE SCRIPT Befehls innerhalb der Session. Bitte beachten Sie, dass importierte sowie mittels pquery() ausgeführte Skripte eine andere Id haben.
exa.meta.node_count	Anzahl an Knoten im Cluster
exa.meta.node_id	Lokale Knoten-Id beginnend bei 0
exa.meta.vm_id	Konstanter Wert 0, da nur eine einzige virtuelle Maschine existiert.

Debug-Output

Gerade bei der Entwicklung von Skripten ist es sehr nützlich, über Debug-Ausgaben den Ablauf des Skriptes analysieren zu können. Hierfür steht die `WITH OUTPUT`-Option des `EXECUTE SCRIPT` Befehls zur Verfügung. Bei Angabe dieser Option werden alle Ausgaben, die über die spezielle Funktion `output()` erzeugt wurden, als Tabelle mit einer Spalte und mehreren Zeilen zurückgegeben, unabhängig vom eigentlichen Rückgabewert des Skriptes.

Die `output()`-Funktion erwartet als einzigen Parameter eine Zeichenkette. Falls andere Werte übergeben werden, wird eine implizite Konvertierung versucht. Ist dies jedoch nicht möglich, so wird eine Zeile mit Wert `NULL` in die Ergebnistabelle eingefügt.

Der Spaltenname der Ergebnistabelle lautet `OUTPUT`, der Spaltentyp ist ein `VARCHAR` der Länge des längsten Strings, der mittels `output()` eingefügt wurde.

Wird die `WITH OUTPUT`-Option beim Ausführen eines Skriptes nicht angegeben, so werden sämtliche Ausgaben über `output()` verworfen.

Beispiele

```
CREATE SCRIPT my_script (param1, param2) AS
    output('SCRIPT started')

    if param1==false then
        output('PARAM1 is false, exit SCRIPT')
        exit()
    else
        output('PARAM2 is '..param2)
    end
    output('End of SCRIPT reached')
/

EXECUTE SCRIPT my_script (true, 5) WITH OUTPUT;

OUTPUT
-----
SCRIPT started
PARAM2 is 5
End of SCRIPT reached
```

Hilfsfunktionen für SQL Bezeichner

Um das Arbeiten mit Skripten zu vereinfachen, werden einige Hilfsfunktionen bereitgestellt:

- **quote(param)**

Setzt Anführungszeichen um den Parameter und verdoppelt darin enthaltene Anführungszeichen. Dies ist in erster Linie für begrenzte Bezeichner in SQL-Befehlen nützlich.

- **join(sep, p1, p2, ...)**

Erzeugt aus den Parametern eine Zeichenkette, wobei `sep` zwischen alle weiteren Parameter eingefügt wird. Somit lassen sich über `join(" . ", ...)` sehr einfach schemaqualifizierte Bezeichner erzeugen (siehe hierzu Abschnitt 2.1.2, [SQL-Bezeichner](#)).

Beispiele

```
output("my_table")          -- regular identifier
output(quote("my_table")) -- delimited identifier
output(join(".", "my_schema", "my_table", "my_column"))
output(join(".", 
            quote("my_schema"),
            quote("my_table"),
            quote("my_column")))
OUTPUT
-----
my_table
"my_table"
my_schema.my_table.my_column
"my_schema"."my_table"."my_column"
```

Sonstige Anmerkungen

Zeilennummern	Der Skript-Text wird intern beginnend mit der ersten Zeile nach dem AS-Schlüsselwort abgespeichert, welche keine Leerzeile ist. Dieser Text kann auch in den entsprechenden Systemtabellen gefunden werden. Daher liefert das folgende Beispiel einen Fehler in "line 5":
---------------	---

```
CREATE SCRIPT my_script AS

    -- this is the first line!
    function do_nothing()
        return
    end
    import(x) -- error: string expected
/
```

Scope-Schema	Während des Ausführens eines Skriptes wird das aktuelle Schema als Scope-Schema berücksichtigt (außer man wechselt im Skript explizit das Schema). Daher sollte man bei der Entwicklung von Skripten beachten, dass Schema-Objekte möglichst qualifiziert ange- sprochen werden.
--------------	--

3.5.4. Bibliotheken

String-Bibliothek

In der String-Bibliothek werden typische Zeichenkettenfunktionen wie Suchen und Ersetzen bereitgestellt. Details zum Thema Pattern finden Sie weiter unten.

Bitte beachten Sie, dass die Position eines Zeichens in einem String mit 1 beginnt und auch negativ sein kann. Im letzteren Fall wird die Position von hinten berechnet, so dass z.B. -1 das letzte Zeichen darstellt.

Funktionen

- **string.find(s, pattern [, init [, plain]])**

Sucht das erste Vorkommen von `pattern` im String `s`. Falls ein Treffer gefunden wurde, so werden die Positionen von Beginn und Ende des Treffers zurückgegeben, ansonsten `nil`. Das dritte, optionale numerische Argument `init` definiert, von welcher Position an gesucht werden soll (Default: 1). Falls im vierten Argument `plain` der Wert `true` angegeben wird, so werden beim Pattern-Matching keine Sonderzeichen berücksichtigt

(z.B. `%w`). Bitte beachten Sie, dass in diesem Fall auch das dritte Argument `init` angegeben werden muss. Falls in `pattern` *captures* enthalten sind (mittels runden Klammern *gespeicherte* Inhalte, siehe Beispiel), so werden diese als weitere Rückgabewerte neben den Treffer-Positionen zurückgeliefert.

```
local x, y = string.find('hello world', 'world')
--> x=7, y=11

local a, b, c, d = string.find('my street number is 54a', '(%d+)(%a+)')
--> a=21, b=23, c=54, d='a'
```

- **string.match(s, pattern [, init])**

Sucht den ersten Treffer von `pattern` im String `s`. Falls die Suche erfolgreich war, liefert `match` die gespeicherten Werte (*captures*, siehe unten) aus dem `pattern` String, ansonsten `nil`. Falls `pattern` keine *captures* beinhaltet, so wird der gesamte Treffer zurückgeliefert. Das dritte, optionale numerische Argument `init` definiert, von welcher Position an gesucht wird (Default: 1).

```
local x, y = string.match('my street number is 54a', '(%d+)(%a+)')
--> x='54'
--> y='a'
```

- **string.gmatch(s, pattern)**

Liefert eine Iterator-Funktion, mit der durch die gespeicherten Werte (*captures*, siehe unten) des `pattern` Strings iteriert werden kann. Falls keine *captures* gespeichert werden, liefert die Funktion stets den gesamten Match zurück.

Im Beispiel werden in einer Schleife alle Schlüssel=Wert Paare in eine Dictionary Table gespeichert.

```
local t = {}
local s = 'from=world, to=moon'
for k, v in string.gmatch(s, '(%w+)=(%w+)') do
    t[k] = v
end
--> t['from']='world'
--> t['to'] = 'moon'
```

- **string.sub(s, i [, j])**

Liefert den Teilstring von `s` ab der Position `i` bis `j` zurück. Die Argumente `i` und `j` können negativ sein. Falls `j` nicht angegeben wird, so wird `-1` verwendet (was gleichbedeutend mit der String-Länge ist). Daher liefert der Aufruf `string.sub(s, 1, j)` den Präfix von `s` der Länge `j` zurück und `string.sub(s, -i)` den Suffix von `s` der Länge `i`.

```
local x = string.sub('hello world', 3)
--> x='llo world'
```

- **string.gsub(s, pattern, repl [, n])**

Liefert eine Kopie von `s` zurück, in der alle Vorkommen von `pattern` durch den Ersetzungs-String `repl` ersetzt werden. Als zweiten Rückgabewert liefert `gsub` die Gesamtzahl an Ersetzungen zurück. Das dritte, optionale Argument `n` limitiert die Zahl an Ersetzungen. Z.B. wird im Fall `n=1` nur das erste Vorkommen ersetzt.

Der String `repl` wird zum Ersetzen verwendet. Das Zeichen `%` dient hierbei als Escape Zeichen: Eine beliebige Element der Form `%n` (mit `n` zwischen 1 und 9) steht für den `n`-ten gespeicherten Wert (*capture*). Das Element `%0` steht für den gesamten Match, das Element `%%` für ein einfaches `%`.

```
local x = string.gsub('hello world', '(%w+)', '%1 %1')
--> x='hello hello world world'

local x = string.gsub('hello world', '%w+', '%0 %0', 1)
--> x='hello hello world'

local x = string.gsub('hello world from Lua', '(%w+)%s*(%w+)', '%2 %1')
--> x='world hello Lua from'
```

- **string.len(s)**

Liefert die Länge eines Strings zurück. Der leere String `''` hat die Länge 0.

```
local x = string.len('hello world')
--> x=11
```

- **string.rep(s, n)**

Liefert einen String zurück, der aus `n` Kopien des Strings `s` besteht.

```
local x = string.rep('hello',3)
--> x='hellohellohello'
```

- **string.reverse(s)**

Liefert den String `s` in umgedrehter Reihenfolge zurück.

```
local x = string.reverse('hello world')
--> x='dlrow olleh'
```

- **string.lower(s)**

Liefert einen String zurück, bei dem alle Großbuchstaben in Kleinbuchstaben konvertiert werden.

```
local x = string.lower('hELLo World')
--> x='hello world'
```

- **string.upper(s)**

Liefert einen String zurück, bei dem alle Kleinbuchstaben in Großbuchstaben konvertiert werden.

```
local x = string.upper('hELLo World')
--> x='HELLO WORLD'
```

- **string.format(formatstring, e1, e2, ...)**

Ersetzt die im Argument `formatstring` definierten Sonderzeichen durch die weiteren Argumente. Die Formatregeln folgen der `printf()` Familie der Standard C Funktionen. Der einzige Unterschied ist, dass Optionen `*`, `l`, `L`, `n`, `p`, und `h` nicht unterstützt werden es eine Extra-Option `q` gibt. Mit `q` wird ein String so eingefügt,

dass er sicher vom Lua Interpreter gelesen werden kann. Die Optionen `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `x`, und `X` erwarten alle ein numerisches Argument, die Optionen `q` und `s` hingegen einen String.

Optionen:

<code>%d, %i</code>	Ganzzahl mit Vorzeichen
<code>%u</code>	Ganzzahl ohne Vorzeichen
<code>%f, %g, %G</code>	Fließkommazahl
<code>%e, %E</code>	Wissenschaftlichen Notation
<code>%o</code>	Oktalwert
<code>%x, %X</code>	Hexadezimalwert
<code>%c</code>	Zeichen
<code>%s</code>	String
<code>%q</code>	Sicherer String

Beispiel:

```
local x = string.format('d=%d e=%e f=%f g=%g', 1.23, 1.23, 1.23, 1.23)
--> x='d=1 e=1.230000e+00 f=1.230000 g=1.23'
```

Pattern

Pattern sind ähnlich zu regulären Ausdrücken.

Eine **Zeichenklasse** repräsentiert eine bestimmte Zeichenmenge. Die folgenden Elemente können benutzt werden, um eine Zeichenklasse zu definieren:

<code>x</code>	Das Zeichen x selbst (falls x keines der Sonderzeichen <code>^\$()%.[]*+-?</code> ist)
<code>.</code>	Der Punkt definiert ein beliebiges Zeichen
<code>%a</code>	Buchstaben
<code>%c</code>	Kontrollzeichen
<code>%d</code>	Ziffern
<code>%l</code>	Kleinbuchstaben
<code>%p</code>	Interpunktionszeichen
<code>%s</code>	Leerzeichen
<code>%u</code>	Großbuchstaben
<code>%w</code>	Alphanumerische Zeichen
<code>%x</code>	Hexadezimalzeichen
<code>%z</code>	Definiert das Zeichen mit Repräsentation 0
<code>%x</code>	Definiert das Zeichen x (wobei x ein beliebiges nicht-alphanumerisches Zeichen ist). Dies ist das Vorgehen, um Sonderzeichen zu escapen. Dies gilt ebenfalls für beliebige Interpunktionszeichen (auch diejenigen, die keine Sonderzeichen sind).
<code>[set]</code>	Definiert eine Menge der darin definierten Zeichen. Ein Wertebereich kann durch das Zeichen <code>-`</code> erreicht werden. Die oben beschriebenen Zeichenklassen können ebenfalls verwendet werden. Alle sonstigen vorkommenden Zeichen repräsentieren sich selbst. Wertebereiche zwischen Zeichenklassen sind nicht definiert, so dass die Mengen wie <code>[%a-z]</code> oder <code>[a-%%]</code> keine Bedeutung haben.

Beispiele für Zeichenmengen:

<code>[%w_]</code> (bzw. <code>[_%w]</code>)	Alle alphanumerischen Zeichen sowie das Zeichen <code>'_'</code>
<code>[0-7]</code>	Alle Oktalziffern
<code>[0-7%`%-]</code>	Alle Oktalziffern, Kleinbuchstaben sowie das Zeichen <code>'`'</code>
<code>[^set]</code>	Definiert das Komplement zu einer Zeichenmenge.

Für alle Zeichenklassen der Form `%x` existiert die Großbuchstabenvariante als Komplement zu der Klasse. Z.B. definiert die Klasse `%S` alle Zeichen, die keine Leerzeichen sind.

Ein *pattern item* kann folgendes Sein:

- Eine einzelne Zeichenklasse: Liefert einen Treffer für beliebige Zeichen aus der Zeichenklasse
- Eine einzelne Zeichenklasse, gefolgt von einem `*': Liefert einen Treffer bei 0 oder mehr wiederholten Treffern für die Klasse. Dabei wird stets die längst mögliche Treffersequenz verwendet.
- Eine einzelne Zeichenklasse, gefolgt von einem `+': Liefert einen Treffer bei 1 oder mehr wiederholten Treffern für die Klasse. Dabei wird stets die längst mögliche Treffersequenz verwendet.
- Eine einzelne Zeichenklasse, gefolgt von einem `-': Analog zu `*', allerdings wird die kürzest mögliche Treffersequenz verwendet.
- Eine einzelne Zeichenklasse, gefolgt von einem `?': Liefert einen Treffer bei 0 oder 1 Vorkommen eines Zeichens aus der Klasse.
- %n (n zwischen 1 und 9): Liefert einen Treffer bei einer Übereinstimmung mit dem n-ten gespeicherten Treffer (*capture*, siehe unten).
- %bxy (wobei x und y zwei unterschiedliche Zeichen sind): Liefert einen Treffer die mit x beginnen und mit y enden und x und y balanciert sind. Balanciert meint, dass beim Lesen von links nach rechts und Addieren von 1 bei x und Subtrahieren von 1 bei y das letzte y das erste Zeichen ist, bei dem der Zählerwert 0 beträgt. Z.B. können mittels %b() Ausdrücke mit balancierten Klammern gefunden werden.

Ein *pattern* ist eine Folge von pattern items. Ein `^' am Anfang eines pattern verankert den Treffer am Anfang des entsprechenden Such-Strings. Ein `'\$' am Ende des pattern verankert den Treffer am Ende des Such-Strings. An anderen Stellen haben `^' und `'\$' keine besondere Bedeutung und repräsentieren sich selbst.

Ein pattern kann Teil-Patterns in Klammern einschließen, so dass diese im Trefferfall zwischengespeichert werden (so genannte *captures*). Captures sind nach ihrer linken Klammer nummeriert. Z.B. wird in "(a*(.)%w(%s*))" der Teil, der mittels "a*(.)%w(%s*)" übereinstimmt, im ersten capture gespeichert und besitzt daher die Nummer 1. Der Treffer für "." besitzt die Nummer 2 und der Treffer für "%s*" die Nummer 3.

Als Spezialfall speichert das leere capture () die aktuelle Position im Such-String. Daher werden für das pattern "(())aa()", angewendet auf den String "flaaap", die beiden captures 3 und 5 gespeichert.

Unicode

Falls Sie Unicode-Zeichen verarbeiten wollen, können Sie die Bibliothek `unicode.utf8` benutzen. Diese enthält die gleichen Funktionen wie die `string`-Bibliothek, allerdings mit Unicode-Support. Es existiert eine weitere Bibliothek namens `unicode.ascii`, welche jedoch die gleiche Funktionalität wie `string` bereitstellt.

XML-Parsing

Die Bibliothek `lxp` umfasst eine Vielzahl von Funktionen zur Verarbeitung von XML-Daten. Eine ausführliche Referenz zu den einzelnen Funktionen finden Sie unter <http://matthewwild.co.uk/projects/luaexpat>.

Die folgenden Methoden stehen zur Verfügung:

<code>lxp.new(callbacks, [separator])</code>	Ein Parser wird über die Funktion <code>lxp.new()</code> erzeugt. Übergeben wird eine Callback Tabelle sowie optional das Parser-Trennzeichen, welches im Namensraum für zusammengesetzte Element-Namen verwendet wird.
<code>close()</code>	Schließt den Parser und gibt sämtlichen Speicher frei. Ein Aufruf von <code>close()</code> ohne einen vorherigen Aufruf von <code>parse()</code> kann zu einem Fehler führen.
<code>getbase()</code>	Liefert den Basis-String zur Auflösung von relativen URLs.
<code>getcallbacks()</code>	Liefert die callbacks Tabelle zurück.
<code>parse(s)</code>	Führt den Parse-Vorgang des Dokuments fort. Die Zeichenkette s enthält einen Teil des (oder eventuell das ganze) Dokument. Falls die Funktion ohne Argument aufgerufen wird, so wird das Dokument geschlossen (aber der Parser muss noch explizit geschlossen werden). Die Funktion liefert

	einen Wert zurück (nicht nil), falls der Parse-Vorgang erfolgreich war. Falls ein Fehler gefunden wurde, so werden fünf Ergebnisse zurückgeliefert: nil, msg, line, col, und pos (nil, Fehlermeldung, Zeilennummer, Spaltennummer und die absolute Position des Fehlers im XML Dokument).
pos()	Liefert drei Ergebnisse zurück: die aktuelle Zeile, Spalte und absolute Position des Parsers.
setbase(base)	Setzt den Basis-String zur Auflösung von relativen URLs.
setencoding(encoding)	Setzt das vom Parser benutzte Encoding. Es existieren vier eingebaute Encodings, die mittels String übergeben werden können: "US-ASCII", "UTF-8", "UTF-16" und "ISO-8859-1".
stop()	Benutzen Sie diese Funktion, um den Parse-Vorgang eines Dokuments zu beenden, z.B. aufgrund eines Fehlers innerhalb eines Callbacks. Nach dieser Funktion akzeptiert der Parser keine neuen Daten.

SQL-Parsing

Die von uns entwickelte Bibliothek `sqlparsing` umfasst eine Vielzahl von Funktionen zur Verarbeitung von SQL-Statements. Details zu den einzelnen Funktionen und der Benutzung finden Sie in [Abschnitt 3.8, SQL-Präprozessor](#).

Internet Zugriff

Mit der Bibliothek `socket` können http, ftp und smtp Verbindungen hergestellt werden. Weitere Dokumentation hierzu finden Sie unter <http://w3.impa.br/~diego/software/lusocket/>.

Math-Bibliothek

Die Math-Bibliothek beinhaltet die meisten Schnittstellen von Standard C, darunter folgende Funktionen und Werte:

math.abs(x)	Absolutbetrag von x
math.acos(x)	Arcus Cosinus von x (im Bogenmaß)
math.asin(x)	Arcus Sinus von x (im Bogenmaß)
math.atan(x)	Arcus Tangens von x (im Bogenmaß)
math.atan2(y,x)	Arcus Tangens von y/x (im Bogenmaß), aber benutzt den Absolutbetrag der beiden Operanden, um den Quadranten des Ergebnisses zu definieren
math.ceil(x)	Kleinste Ganzzahl, die größer oder gleich zu x ist
math.cos(x)	Cosinus von x (im Bogenmaß)
math.cosh(x)	Cosinus hyperbolicus von x
math.deg(x)	Gradmaß von x in Grad (x angegeben im Bogenmaß)
math.exp(x)	Liefert die natürliche Exponentialfunktion von x
math.floor(x)	Größte Ganzzahl, die kleiner oder gleich zu x ist
math.fmod(x,y)	Modulo

<code>math.frexp(x)</code>	Liefert m und n, so dass gilt: $x=m2^n$. n ist eine Ganzzahl und der Absolutbetrag von m liegt im Intervall [0.5;1] (oder 0, falls $x=0$).
<code>math.huge</code>	Wert <code>HUGE_VAL</code> , der größer ist als jeder numerische Wert
<code>math.ldexp(m,n)</code>	Liefert $m2^n$ (n sollte eine Ganzzahl sein)
<code>math.log(x)</code>	Natürlicher Logarithmus von x
<code>math.log10(x)</code>	Logarithmus von x zur Basis 10
<code>math.max(x, ...)</code>	Maximum von einer Reihe von Werten
<code>math.min(x, ...)</code>	Minimum von einer Reihe von Werten
<code>math.modf(x)</code>	Liefert zwei Zahlen (den Ganzzahlwert und den Nachkommanteil von x)
<code>math.pi</code>	Der Wert der Zahl π
<code>math.pow(x,y)</code>	Liefert x^y
<code>math.rad(x)</code>	Radian von x (in Grad angegeben) im Bogenmaß
<code>math.random([m,[n]])</code>	Entspricht der in ANSI C enthaltenen Zufallszahlengenerator-Funktion <code>rand</code> . Bei der Angabe keines Arguments wird eine Pseudo-Zufallszahl im Bereich [0,1) zurückgeliefert. Bei einem Argument wird eine Pseudo-Ganzzahl aus dem Bereich [1,m] geliefert, bei zwei Argumenten entsprechend aus dem Bereich [m,n].
<code>math.randomseed(x)</code>	Setzt x als Seed für den Pseudo-Zufallszahlengenerator
<code>math.sin(x)</code>	Sinus von x (im Bogenmaß)
<code>math.sinh(x)</code>	Sinus hyperbolicus von x
<code>math.sqrt(x)</code>	Quadratwurzel von x
<code>math.tan(x)</code>	Tangens von x
<code>math.tanh(x)</code>	Tangens hyperbolicus von x

Table-Bibliothek

In dieser Bibliothek werden einige Funktionen zur Tabellen-Manipulation zur Verfügung gestellt. Die meisten Funktionen gehen davon aus, dass die Tabelle ein Array oder eine Liste darstellt. Wenn von der Länge einer Tabelle gesprochen wird, so meint dies das Ergebnis des Längen-Operators (#table).

Funktionen

- **table.insert(table, [pos,] value)**

Fügt ein Element in die Tabelle an der Position pos ein, während die weiteren Elemente ggf. verschoben werden. Der Defaultwert für pos ist n+1, wobei n die Länge der Tabelle ist. Daher fügt der Aufruf `table.insert(t,x)` den Wert x ans Ende der Tabelle t.

- **table.remove(table [, pos])**

Löscht ein Element einer Tabelle von Position pos und verschiebt ggf. die anderen Elemente. Rückgabewert ist der gelöschte Wert selbst. Der Defaultwert für pos ist n, wobei n die Länge der Tabelle ist. Daher löscht der Aufruf `table.remove(t)` das letzte Element der Tabelle t.

- **table.concat(table [, sep [, i [, j]]])**

Liefert einen String zurück, der die Werte der Positionen *i* bis *j* mit dem Separator *sep* konkateniert, also `table[i]..sep..table[i+1] ... sep..table[j]`. Defaultwerte für *sep*, *i* und *j* sind der leere String, 1 und die Länge der Tabelle. Falls *i* größer als *j* ist, so wird ein leerer String zurückgegeben.

- **table.sort(table [, comp])**

Sortiert eine Tabelle. Falls eine Vergleichsfunktion *comp* angegeben wurde, so muss diese 2 Argumente erwarten und den Wert `true` zurückliefern, falls das erste Argument kleiner ist als das zweite. Beachten Sie, dass der Sortieralgorithmus nicht *stabil* ist, d.h. gleiche Werte durch den Sortieralgorithmus die Plätze tauschen können.

- **table.maxn(table)**

Liefert den größten positiven numerischen Schlüsselwert, den die Tabelle besitzt. Falls sie keinen positiven numerischen Schlüssel hat, so wird Null zurückgegeben. Beachten Sie, dass diese Funktion die komplette Tabelle traversiert.

3.5.5. Systemtabellen

In folgenden Systemtabellen können die in der Datenbank angelegten Skripte angezeigt werden:

- [EXA_USER_SCRIPTS](#)
- [EXA_ALL_SCRIPTS](#)
- [EXA_DB_SCRIPTS](#)

Weitere Details hierzu finden Sie in [Anhang A, Systemtabellen](#).

3.6. UDF Skripte

3.6.1. Was sind UDF Skripte?

UDF (=User-defined function) Skripte bieten Ihnen die Möglichkeit, eigene Analyse-, Verarbeitungs- und Generierungsfunktionen zu programmieren und innerhalb von Exasol parallel in einem Hochleistungscluster ausführen zu lassen (*In Database Analytics*). Durch dieses Prinzip lassen sich viele Probleme sehr performant lösen, die bisher in SQL undenkbar waren. Mit UDF Skripten erhalten Sie also eine flexible Schnittstelle, mit der sich nahezu beliebige Anforderungen umsetzen lassen. Sie werden so zum **HPC**-Entwickler, ohne spezielle Kenntnisse zu benötigen.

 Bitte beachten Sie, dass UDF Skripte Teil der Advanced Edition von Exasol ist.

Mit UDF Skripten können Sie folgende benutzerdefinierte Erweiterungen umsetzen:

- Skalare Funktionen
- Aggregatsfunktionen
- Analytische Funktionen
- MapReduce Algorithmen
- Nutzerdefinierte Beladungs-Prozesse

Um die Vielfalt von UDF Skripten zu verwenden, müssen Sie lediglich ein benutzerdefiniertes Skript anlegen ([CREATE SCRIPT](#), siehe [Abschnitt 2.2.1, Definition der Datenbank \(DDL\)](#)) und dieses anschließend in einem SELECT-Statement benutzen. Durch die enge Einbettung innerhalb von SQL lässt sich ideale Performance und Skalierbarkeit erreichen.

Exasol unterstützt die Programmiersprachen Java, Lua, Python und R, um Ihnen den Einstieg so einfach wie möglich zu machen. Zudem bieten die unterschiedlichen Sprachen aufgrund ihrer jeweiligen inhaltlichen Ausrichtung (z.B. statistische Funktionen in R) und der jeweils mitgelieferten Bibliotheken (XML-Parser, etc.) spezifische Vorteile. Bitte beachten Sie daher die Kapitel weiter unten, in denen die speziellen Eigenschaften der verschiedenen Programmiersprachen erläutert werden.

 Die konkreten Versionen der Skriptsprachen können mit entsprechenden Metadaten-Funktionen ausgelesen werden.

Innerhalb des [CREATE SCRIPT](#) Befehls müssen Sie definieren, welche Art von Eingabe- bzw. Rückgabewerten verarbeitet werden. Sie können beispielsweise ein Skript definieren, das aus einer Eingabezeile eine Vielzahl von Ergebniszellen erzeugt (SCALAR . . . EMITS).

Eingabewerte:	
SCALAR	Das Schlüsselwort SCALAR legt fest, dass das Skript skalare Eingabeparameter verarbeitet, also jeweils pro Eingabezeile einmal aufgerufen wird.
SET	Wird SET definiert, so bezieht sich die Ausführung auf eine Menge von Eingabedaten, durch die iteriert werden kann (siehe Abschnitt 3.6.2, Einführende Beispiele).
Rückgabewerte:	
RETURNS	In diesem Fall liefert das Skript einen einzelnen Rückgabewert zurück.
EMITS	Falls das Schlüsselwort EMITS definiert wurde, kann ein Skript mehrere Ergebniszellen (Tupel) erzeugen. Im Falle eines SET Inputs darf der EMITS Output in der Select-Liste nur alleine stehen, also nicht mit weiteren Ausdrücken kombiniert werden. Allerdings können Sie selbstverständlich ein Subselect benutzen, um die Zwischenergebnisse weiter zu verarbeiten.

Die Datentypen der Eingabe- und Rückgabeparameter können spezifiziert werden, damit die Konvertierung aus den internen Datentypen der Programmiersprachen eindeutig ist. Falls Sie dies nicht tun, muss das Skript dynamisch darauf reagieren (siehe Details und Beispiele weiter unten).



Bitte beachten Sie, dass die Eingabeparameter von Skripten immer case-sensitiv behandelt werden, analog zum eigentlichen Skript-Code. Dieses Verhalten unterscheidet sich von normalen SQL Bezeichnern, die nur im Falle von doppelten Anführungszeichen eingeschlossen Bezeichnern (begrenzte Bezeichner) case-sensitiv behandelt werden.

Skripte müssen eine Funktion `run()` enthalten. Diese Funktion wird mit einem Parameter aufgerufen, welcher Zugriff auf die Eingabedaten aus Exasol ermöglicht. Sofern Ihr Skript mehrere Eingabetupel innerhalb eines Aufrufs verarbeiten soll, Ihr Skript also vom SET-Typ ist, können Sie mit Hilfe des Parameters durch die einzelnen Eingabetupel iterieren.

Weitere allgemeine Hinweise finden Sie in folgender Tabelle:

Tabelle 3.1. Hinweise zu UDF Skripten

Thema	Erläuterungen
Interne Prozessverarbeitung	<p>Während der Verarbeitung des SQLs werden für darin verwendete Skripte pro Knoten mehrere virtuelle Maschinen gestartet, die die Daten unabhängig voneinander parallel verarbeiten.</p> <p>Bei skalaren Funktionen werden die Eingabezeilen auf diese Prozesse verteilt, so dass eine möglichst hohe Parallelität erreicht wird.</p> <p> Bei SET-Eingabedaten werden die virtuellen Maschinen pro Gruppe benutzt (bei Verwendung der GROUP BY Klausel). Falls keine GROUP BY Klausel angegeben ist, existiert nur eine Gruppe, weshalb die Daten auch nur von einem Knoten und einer virtuellen Maschine verarbeitet werden können.</p>
ORDER BY Klausel	<p>Entweder in der Skripterzeugung (CREATE SCRIPT) oder beim Aufruf kann eine ORDER BY Klausel angegeben werden, die zu einer sortierten Abarbeitung der Gruppen von SET-Eingabedaten führt. Dies kann je nach Algorithmus sinnvoll sein. Falls dies jedoch notwendig für den Algorithmus ist, so sollte man die Sortierung bereits bei der Erzeugung angeben, um Fehler bei der Verwendung zu vermeiden.</p>
Performance-Vergleich zwischen den Programmiersprachen	<p>Die Performance der verschiedenen Sprachen kann nur schwer verglichen werden, weil die unterschiedlichen Elemente der Sprachen unterschiedliche Eigenschaften aufweisen können. So kann die String-Verarbeitung in einer Sprache schneller sein, während die XML-Verarbeitung in der anderen performanter ist.</p> <p>Grundsätzlich empfiehlt sich jedoch der Einsatz der Sprache Lua, sofern Performance das wichtigste Kriterium ist. Dies liegt daran, dass die Lua-Sprache aus technischen Gründen am tiefsten in Exasol integriert ist und somit der geringste Prozess-Overhead entsteht.</p>

3.6.2. Einführende Beispiele

In diesem Kapitel stellen wir Ihnen einige einführende Lua-Beispiele vor, die Ihnen einen ersten Einblick in die Funktionsweise von benutzerdefinierten Skripten geben sollen. Beispiele für die anderen Programmiersprachen finden Sie weiter unten.

Skalare Funktionen

Benutzerdefinierte skalare Funktionen (Schlüsselwort SCALAR) sind der einfachste Fall von benutzerdefinierten Skripten. Das Skript erzeugt pro Eingabewert (evtl. ein Tupel aus mehreren Werten) einen skalaren Ergebniswert (Schlüsselwort RETURNS) oder mehrere Ergebnis-Tupel (Schlüsselwort SET).

Beachten Sie, dass im Skript immer eine Funktion `run()` definiert sein muss, in der die Ausführung geschieht. Dieser Funktion wird zur Laufzeit ein Kontext übergeben (im Beispiel unten hat dieser den Namen `ctx`), der die eigentliche Schnittstelle zwischen Skript und Datenbank darstellt.

Im folgenden Beispiel wird ein Skript definiert, welches das Maximum zweier Werte zurückliefert. Dieses entspricht dem CASE-Ausdruck `CASE WHEN x>=y THEN x WHEN x<y THEN y ELSE NULL`.



Der abschließende Schrägstrich ('/') ist nur bei der Verwendung von EXAplus erforderlich.

```
CREATE LUA SCALAR SCRIPT my_maximum (a DOUBLE, b DOUBLE)
    RETURNS DOUBLE AS
function run(ctx)
    if ctx.a == null or ctx.b == null
        then return null
    end
    if ctx.a > ctx.b
        then return ctx.a
        else return ctx.b
    end
end
/
SELECT x,y,my_maximum(x,y) FROM t;

X          Y          MY_MAXIMUM(T.X,T.Y)
-----  -----  -----
1            2            2
2            2            2
3            2            3
```

Aggregats- und Analytische Funktionen

Noch wesentlich interessanter werden UDF Skripte, wenn bei der Ausführung eines Skripts mehrere Daten analysiert werden können. Hierdurch lassen sich beliebige Aggregats- oder analytische Funktionen erstellen. Durch das Schlüsselwort `SET` wird spezifiziert, dass eine Menge von Eingabedaten erwartet wird. Diese Daten lassen sich innerhalb der `run()` Funktion durch einen Iterator verarbeiten (Funktion `next()`).

Darüber hinaus können Skripte entweder einen einzelnen skalaren Wert zurückliefern (Schlüsselwort `RETURNS`) oder aber eine Vielzahl von Ergebnissen (Tupeln) erzeugen (`EMITS`).

Im folgenden Beispiel werden zwei Skripte definiert: die Aggregatfunktion `my_average` (simuliert `AVG`) sowie eine analytische Funktion `my_sum`, die pro Eingabezeile drei Werte erzeugt (eine fortlaufende Nummer, den aktuellen Wert sowie die Summe der bisherigen Werte). Die Eingabedaten werden hierbei aufgrund der `ORDER BY` Klausel sortiert verarbeitet.

```
CREATE LUA SET SCRIPT my_average (a DOUBLE)
    RETURNS DOUBLE AS
function run(ctx)
    if ctx.size() == 0
        then return null
    else
        local sum = 0
        repeat
            if ctx.a ~= null then
                sum = sum + ctx.a
            end
        until not ctx.next()
```

```

        return sum/ctx.size()
    end
end
/
SELECT my_average(x) FROM t;

MY_AVERAGE(T.X)
-----
7.75

CREATE LUA SET SCRIPT my_sum (a DOUBLE)
    EMITS (count DOUBLE, val DOUBLE, sum DOUBLE) AS
function run(ctx)
    local sum = 0
    local count = 0
    repeat
        if ctx.a ~= null then
            sum = sum + ctx.a
            count = count + 1
            ctx.emit(count,ctx.a,sum)
        end
    until not ctx.next()
end
/
SELECT my_sum(x ORDER BY x) FROM t;

COUNT          VAL          SUM
-----
1              4              4
2              7              11
3              9              20
4             11              31

```

Dynamische Eingabe- und Ausgabe-Parameter

Wenn Sie in der Skript-Definition anstatt fixer Eingabe- bzw. Ausgabe-Parameter die Syntax (. . .) verwenden, können sehr flexible Skripte definiert werden. Einerseits kann das gleiche Skript anschließend für beliebige Datentypen verwendet werden (z.B. eine Maximum-Funktion unabhängig vom Typ), und andererseits sogar eine flexible Anzahl von Parametern verarbeiten. Gleichermaßen können im Falle von EMITS Skripten sowohl Anzahl als auch Typen der Ausgabe-Parameter dynamisch gestaltet werden.

Für die Arbeit mit dynamischen Eingabe-Parametern können die Anzahl und die Typen der Eingabe-Parameter in den Metadaten abgefragt werden. Über den Index kann dann der Wert eines Eingabe-Parameters gelesen werden. Beispielsweise können Sie die Anzahl der Eingabe-Parameter in Python über die Variable `exa.meta.input_column_count` abrufen. Bitte beachten Sie jeweils die Details zu den Programmiersprachen, die unter [Abschnitt 3.6.3, Details für die verschiedenen Sprachen](#) beschrieben werden.

Für UDFs mit dynamischen Ausgabe-Parametern werden die Ausgabe-Parameter und deren Typen bei jedem Aufruf der UDF individuell bestimmt. Es gibt hierbei drei Möglichkeiten:

1. Sie können die Ausgabe-Parameter direkt im SELECT beim Aufruf der UDF angeben. Ergänzen Sie hierzu hinter dem Aufruf der UDF ein EMITS gefolgt von der Spezifikation der Ausgabe-Parameter (Namen und Typen).

2. Wird die UDF im äußersten SELECT eines `INSERT INTO SELECT` Befehls aufgerufen, werden die Spalten der Zieltabelle als Ausgabe-Parameter verwendet.
3. Falls weder `EMITS` angegeben ist, noch ein `INSERT INTO SELECT`, ruft die Datenbank die Funktion `default_output_columns()` (der Name variiert, hier für Python) auf, welche die Ausgabe-Parameter dynamisch berechnet und zurück gibt. Diese Funktion kann vom Nutzer implementiert werden. Unter [Abschnitt 3.6.3, Details für die verschiedenen Sprachen](#) wird beschrieben wie die Callback Methode in jeder Programmiersprache implementiert werden kann.

Im folgenden sehen sie Beispiele für alle drei Möglichkeiten:

```
-- Definition einer sehr einfachen Sampling UDF, in der der letzte
-- Parameter die Sampling-Rate spezifiziert (in Prozent), mit der die
-- Zeilen selektiert werden sollen
CREATE PYTHON SCALAR SCRIPT sample_simple (...) EMITS (...) AS
from random import randint, seed
seed(1001)
def run(ctx):
    percent = ctx[exa.meta.input_column_count-1]
    if randint(0,100) <= percent:
        currentRow = [ctx[i] for i in range(0, exa.meta.input_column_count-1)]
        ctx.emit(*currentRow)
/
-- Dies ist die gleiche UDF, aber hier werden die Ausgabe-Parameter
-- automatisch generiert, um die explizite EMITS Definition im SELECT
-- zu vermeiden.
-- In default_output_columns() wird ein Präfix 'c' hinzugefügt, weil die
-- Input Spalten nur eine genierte Zahlenfolge ist
CREATE PYTHON SCALAR SCRIPT sample (...) EMITS (...) AS
from random import randint, seed
seed(1001)
def run(ctx):
    percent = ctx[exa.meta.input_column_count-1]
    if randint(0,100) <= percent:
        currentRow = [ctx[i] for i in range(0, exa.meta.input_column_count-1)]
        ctx.emit(*currentRow)
def default_output_columns():
    output_args = list()
    for i in range(0, exa.meta.input_column_count-1):
        name = exa.meta.input_columns[i].name
        type = exa.meta.input_columns[i].sql_type
        output_args.append("c" + name + " " + type)
    return str(", ".join(output_args))
/
-- Beispieldatenebene
ID      USER_NAME PAGE_VISITS
----- -----
    1 Alice          12
    2 Bob            4
    3 Pete           0
    4 Hans          101
    5 John           32
    6 Peter          65
    7 Graham         21
    8 Steve           4
    9 Bill           64
```

10 Claudia 201

```
-- Die erste UDF benötigt die Angabe der Ausgabe-Parameter mittels EMITS.
-- Es werden hier 20% der Zeilen zufällig extrahiert.

SELECT sample_simple(id, user_name, page_visits, 20)
EMITS (id INT, user_name VARCHAR(100), PAGE_VISITS int)
FROM people;

ID      USER_NAME PAGE_VISITS
----- -----
 2 Bob          4
 5 John         32

-- Die zweite UDF ermittelt die Ausgabe-Typen automatisch
SELECT SAMPLE(id, user_name, page_visits, 20)
FROM people;

C0      C1      C2
----- -----
 2 Bob          4
 5 John         32

-- Im Falle von INSERT INTO werden die Ziel-Datentypen implizit aus den Spalten der ...
CREATE TABLE people_sample LIKE people;
INSERT INTO people_sample
  SELECT sample_simple(id, user_name, page_visits, 20) FROM people;
```

MapReduce Programme

Das UDF Skripte Framework ist durch seine Flexibilität in der Lage, nahezu sämtliche vorstellbare Analysen umzusetzen. Um Ihnen diese Mächtigkeit näher zu bringen, folgt hier ein Beispiel eines MapReduce-Programms. In unserem Beispiel sollen die Häufigkeiten einzelner Wörter innerhalb eines Textes ermittelt werden - ein Problem, das mit Standard-SQL nicht lösbar ist.

Dazu wird ein Skript `map_words`, erstellt, welches aus einem Text die einzelnen Wörter extrahiert und emittiert. Dieses Skript wird anschließend in SQL integriert, wobei zur Aggregation (dem typischen Reduce-Schritt in MapReduce) kein weiteres Skript benötigt wird, sondern die SQL-Funktion `COUNT` verwendet werden kann. Dies erleichtert einerseits den Implementierungsaufwand, da eine ganze Reihe von SQL-Funktionen zur Verfügung stehen. Andererseits wird die Analysegeschwindigkeit nochmals gesteigert, weil die native SQL-Ausführung noch performanter ist.

```
CREATE LUA SCALAR SCRIPT map_words(w varchar(10000))
EMITS (words varchar(100)) AS
function run(ctx)
    local word = ctx.w
    if (word ~= null)
    then
        for i in unicode.utf8.gmatch(word, '(%w+)')
        do
            ctx.emit(i)
        end
    end
end
/
```

```
SELECT words, COUNT(*) FROM
  (SELECT map_words(l_comment) FROM tpc.lineitem)
GROUP BY words ORDER BY 2 desc LIMIT 10;
```

WORDS	COUNT(*)
the	1376964
slyly	649761
regular	619211
final	542206
carefully	535430
furiously	534054
ironic	519784
blithely	450438
even	425013
quickly	354712

Zugriff auf externe Dienste

Innerhalb von Skripten lassen sich externe Dienste und Daten einbinden, was die Möglichkeiten der Analyse nochmals deutlich flexibler macht.

Im folgenden Beispiel wird eine Liste von URLs aus einer Tabelle verarbeitet, die entsprechenden Dokumente vom Webserver geladen und schließlich die Länge der Dokumente berechnet. Bitte beachten Sie, dass jede Skriptsprache hierfür eine andere Bibliothek zur Verfügung stellt (siehe weiter unten).

```
CREATE LUA SCALAR SCRIPT length_of_doc (url VARCHAR(50))
  EMITS (url VARCHAR(50), doc_length DOUBLE) AS
http = require("socket.http")
function run(ctx)
  file = http.request(ctx.url)
  if file == nil then error('Cannot open URL ' .. ctx.url) end
  ctx.emit(ctx.url, unicode.utf8.len(file))
end
/
SELECT length_of_doc(url) FROM t;

URL                      DOC_LENGTH
-----
http://en.wikipedia.org/wiki/Main_Page.htm      59960
http://en.wikipedia.org/wiki/Exasol              30007
```

Benutzerdefiniertes ETL mittels UDFs

UDF Skripte können ebenfalls zur Implementierung von sehr flexiblen ETL Prozessen verwendet werden, indem darin definiert wird, wie Daten aus externen Systemen extrahiert und konvertiert werden. Für nähere Details und Beispielen verweisen wir auf [Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs](#) in [Abschnitt 3.4, ETL-Prozesse](#).

3.6.3. Details für die verschiedenen Sprachen



Unabhängig von der Skriptsprache gibt es die Möglichkeit, Umgebungsvariablen für die Ausführung des Skripts zu definieren. Die entsprechende Syntax ist `%env <variable>=<value>`, und ein nützlicher Anwendungsfall ist das Setzen des Suchpfades,

damit bestimmte Bibliotheken im BucketFS (Details hierzu weiter unten) gefunden werden.

Beispiel: %env LD_LIBRARY_PATH=/buckets/bfs-default/hadoop_libs/native_lib

Lua

Neben den folgenden Infos finden Sie in [Abschnitt 3.5, Scripting-Programmierung](#) weitere Informationen zu Lua. Darüber hinaus empfiehlt sich auch die offizielle Lua-Dokumentation (siehe <http://www.lua.org>).

run() und cleanup() Methoden	<p>Die Methode <code>run()</code> wird für jeden Datensatz (SCALAR) bzw. jede Gruppe (SET) aufgerufen. Deren Übergabeparameter ist eine Art Ausführungskontext und ermöglicht den Zugriff auf die Daten sowie den Iterator bei SET-Funktionen.</p> <p>Zum Initialisieren aufwändiger Dinge (z.B. externe Verbindungen öffnen) können Sie Code außerhalb der <code>run()</code>-Methode schreiben. Dieser wird je virtueller Maschine einmal am Anfang der SELECT-Ausführung aufgerufen. Zum Deinitialisieren existiert die Methode <code>cleanup()</code>, welche ganz zum Schluss der Ausführung einmal pro virtueller Maschine aufgerufen wird.</p>																				
Parameter	<p>Beachten Sie, dass die Lua-Datentypen und die Datenbank-SQL-Typen nicht identisch sind. Daher werden bei den Ein- und Ausgabedaten folgende Konvertierungen durchgeführt:</p> <table> <tbody> <tr> <td>DOUBLE</td> <td>number</td> </tr> <tr> <td>DECIMAL bzw. INTEGER</td> <td>decimal</td> </tr> <tr> <td>BOOLEAN</td> <td>boolean</td> </tr> <tr> <td>VARCHAR bzw. CHAR</td> <td>string</td> </tr> <tr> <td>Sonstige</td> <td>Nicht unterstützt</td> </tr> </tbody> </table> <p> Bitte beachten Sie auch die Details zu den Lua-Datentypen in Abschnitt 3.5, Scripting-Programmierung, besonders zum speziellen Typ decimal.</p> <p> NULL-Werte werden durch die spezielle Konstante <code>NULL</code> repräsentiert.</p>	DOUBLE	number	DECIMAL bzw. INTEGER	decimal	BOOLEAN	boolean	VARCHAR bzw. CHAR	string	Sonstige	Nicht unterstützt										
DOUBLE	number																				
DECIMAL bzw. INTEGER	decimal																				
BOOLEAN	boolean																				
VARCHAR bzw. CHAR	string																				
Sonstige	Nicht unterstützt																				
Metadaten	<p>Die Eingabedaten können über den Namen adressiert werden, also z.B. <code>ctx.my_input</code>.</p> <p>Falls Sie eine dynamische Anzahl von Parametern verwenden wollen, können Sie dies über die Notation <code>(...)</code> erreichen, also z.B. <code>CREATE LUA SCALAR SCRIPT my_script (...)</code>. Sie können die einzelnen Parameter dann einfach über einen Index zugreifen, z.B. <code>ctx[1]</code> für den ersten Parameter. Die Anzahl der Parameter sowie ihre Datentypen (werden während dem Aufruf bestimmt) sind Bestandteil der Metadaten.</p>																				
	<p>Folgende Metadaten stehen als globale Variablen zur Verfügung:</p> <table> <tbody> <tr> <td><code>exa.meta.database_name</code></td> <td>Datenbankname</td> </tr> <tr> <td><code>exa.meta.database_version</code></td> <td>Datenbankversion</td> </tr> <tr> <td><code>exa.meta.script_language</code></td> <td>Name und Version der Sprache</td> </tr> <tr> <td><code>exa.meta.script_name</code></td> <td>Name des Skripts</td> </tr> <tr> <td><code>exa.meta.script_schema</code></td> <td>Schema, in dem das Skripts liegt</td> </tr> <tr> <td><code>exa.meta.current_schema</code></td> <td>Schema, das aktuell geöffnet ist</td> </tr> <tr> <td><code>exa.meta.script_code</code></td> <td>Text des Skripts</td> </tr> <tr> <td><code>exa.meta.session_id</code></td> <td>Session-Id</td> </tr> <tr> <td><code>exa.meta.current_user</code></td> <td>Aktueller Nutzer</td> </tr> <tr> <td><code>exa.meta.statement_id</code></td> <td>Statement-Id innerhalb der Session</td> </tr> </tbody> </table>	<code>exa.meta.database_name</code>	Datenbankname	<code>exa.meta.database_version</code>	Datenbankversion	<code>exa.meta.script_language</code>	Name und Version der Sprache	<code>exa.meta.script_name</code>	Name des Skripts	<code>exa.meta.script_schema</code>	Schema, in dem das Skripts liegt	<code>exa.meta.current_schema</code>	Schema, das aktuell geöffnet ist	<code>exa.meta.script_code</code>	Text des Skripts	<code>exa.meta.session_id</code>	Session-Id	<code>exa.meta.current_user</code>	Aktueller Nutzer	<code>exa.meta.statement_id</code>	Statement-Id innerhalb der Session
<code>exa.meta.database_name</code>	Datenbankname																				
<code>exa.meta.database_version</code>	Datenbankversion																				
<code>exa.meta.script_language</code>	Name und Version der Sprache																				
<code>exa.meta.script_name</code>	Name des Skripts																				
<code>exa.meta.script_schema</code>	Schema, in dem das Skripts liegt																				
<code>exa.meta.current_schema</code>	Schema, das aktuell geöffnet ist																				
<code>exa.meta.script_code</code>	Text des Skripts																				
<code>exa.meta.session_id</code>	Session-Id																				
<code>exa.meta.current_user</code>	Aktueller Nutzer																				
<code>exa.meta.statement_id</code>	Statement-Id innerhalb der Session																				

	<p>exa.meta.node_count exa.meta.node_id exa.meta.vm_id</p> <p>exa.meta.input_type</p> <p>exa.meta.input_column_count exa.meta.input_columns[]</p> <p>exa.meta.output_type</p> <p>exa.meta.output_column_count exa.meta.output_columns[]</p>	<p>Anzahl an Knoten im Cluster Lokale Knoten-Id beginnend bei 0 Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander) Typ der Eingabedaten (SCALAR oder SET) Anzahl der Eingabe-Spalten Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length} Typ der Ausgabedaten (RETURNS oder EMITS) Anzahl der Ausgabe-Spalten Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}</p>				
Daten-Iterator, <code>next()</code> , <code>size()</code> und <code>reset()</code>	Für Skripte mit mehreren Eingabewerten pro Aufruf (Schlüsselwort SET) wird mittels der <code>next()</code> -Methode durch die Daten iteriert (auf die Methode <code>next()</code> wird über den Kontext zugegriffen). Initial zeigt der Iterator auf die erste Eingabezeile. Daher bietet sich eine <i>repeat...until</i> Schleife an (siehe Beispiele).	<p> Falls die Eingabemenge leer ist, wird die <code>run()</code>-Methode gar nicht aufgerufen. Als Ergebnis wird dann analog zu Aggregatsfunktionen der Wert NULL zurückgeliefert (wie z.B. in <code>SELECT MAX(x) FROM t WHERE false</code>).</p> <p>Zusätzlich gibt es eine Methode <code>reset()</code>, mit der der Iterator auf den ersten Wert zurückgesetzt werden kann. Damit sind auch mehrere Durchläufe durch die Daten möglich, falls das Ihr Algorithmus benötigt.</p> <p>Die Anzahl der Eingabedaten lässt sich über die Methode <code>size()</code> abfragen.</p>				
<code>emit()</code>	<p>Mehrere Ergebnissezeilen (Schlüsselwort EMITS) können über die Methode <code>emit()</code> erzeugt werden.</p> <p>Die Funktion erwartet so viele Parameter wie Ausgabe-Spalten. Im Falle von dynamischen Ausgabe-Parametern bietet sich in Lua die Nutzung eines Array-Objekts an, welches Sie dann mit <code>unpack()</code> referenzieren können - z.B. <code>ctx.emit(unpack({1, "a"}))</code>.</p>					
Import anderer Skripte	Zum Import anderer Skripte steht die Methode <code>exa.import()</code> zur Verfügung. Auch Skript-Programme (siehe Abschnitt 3.5, Scripting-Programmierung) können importiert werden, allerdings können keine Parameter übergeben werden.					
Zugriff auf Verbindungsdefinitionen	<p>Die Details von Verbindungen, die mit CREATE CONNECTION definiert wurden können innerhalb von LUA UDF Skripten mit Hilfe der Methode <code>exa.get_connection("connection_name")</code> abgefragt werden. Das Ergebnis ist eine Lua Table mit folgenden Einträgen:</p> <table> <tr> <td><code>type</code></td><td>Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.</td></tr> <tr> <td><code>address</code></td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.</td></tr> </table>	<code>type</code>	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.	<code>address</code>	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.	
<code>type</code>	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.					
<code>address</code>	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.					

	user	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.												
	password	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.												
Hilfsbibliotheken	<p>Zu den in Lua zur Verfügung gestellten Bibliotheken finden Sie in Abschnitt 3.5, Scripting-Programmierung ausführliche Informationen. Kurz zusammenfassend stehen folgende Bibliotheken zur Verfügung:</p> <table> <tr> <td>math</td><td>Mathematische Berechnungen</td></tr> <tr> <td>table</td><td>Tabellen-Operationen</td></tr> <tr> <td>string</td><td>Textverarbeitung</td></tr> <tr> <td>unicode.utf8</td><td>Analoge Funktionen wie in string, allerdings für Unicode-Daten</td></tr> <tr> <td>socket</td><td>Internet Support (http, ftp, smtp) (siehe auch http://w3.impa.br/~diego/software/luasocket/)</td></tr> <tr> <td>lxp</td><td>XML Parsing (siehe auch http://matthewwild.co.uk/projects/luaexpat/)</td></tr> </table> <p> Die Module <code>math</code>, <code>table</code>, <code>string</code> und <code>unicode.utf8</code> stehen immer im Namensraum zur Verfügung, die anderen müssen mittels <code>require()</code> geladen werden.</p> <p> Zusätzliche Lua-Pakete können nicht vom Nutzer installiert werden.</p>		math	Mathematische Berechnungen	table	Tabellen-Operationen	string	Textverarbeitung	unicode.utf8	Analoge Funktionen wie in string, allerdings für Unicode-Daten	socket	Internet Support (http, ftp, smtp) (siehe auch http://w3.impa.br/~diego/software/luasocket/)	lxp	XML Parsing (siehe auch http://matthewwild.co.uk/projects/luaexpat/)
math	Mathematische Berechnungen													
table	Tabellen-Operationen													
string	Textverarbeitung													
unicode.utf8	Analoge Funktionen wie in string, allerdings für Unicode-Daten													
socket	Internet Support (http, ftp, smtp) (siehe auch http://w3.impa.br/~diego/software/luasocket/)													
lxp	XML Parsing (siehe auch http://matthewwild.co.uk/projects/luaexpat/)													
Funktion <code>default_output_columns()</code> für dynamische Ausgabe-Parameter	<p>Falls ein UDF Skript mit dynamischen Ausgabe-Parametern definiert wurde und die Ausgabe-Parameter nicht bestimmt werden können (über EMITS in der Abfrage oder über INSERT INTO), ruft die Datenbank die Funktion <code>default_output_columns()</code> auf, die Sie implementieren können. Der erwartete Rückgabewert ist ein String mit den Namen und Typen der Ausgabe-Parameter, z.B. "a int, b varchar(100)". Siehe Dynamische Eingabe- und Ausgabe-Parameter für eine genaue Beschreibung wann diese Methode aufgerufen wird und für Beispiele.</p> <p> Sie können die Metadaten über <code>exa.meta</code> abrufen, z.B. um die Anzahl und die Typen der Eingabe-Parameter zu abzufragen.</p> <p> Diese Methode wird genau einmal auf einem einzigen Knoten ausgeführt.</p>													
Funktion <code>generate_sql_for_import_spec()</code> für benutzerdefinierten Import (siehe Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs)	<p>Damit ein UDF Skript in einem benutzerdefinierten Import (IMPORT FROM SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>generate_sql_for_import_spec(import_spec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>import_spec</code> enthält alle Informationen über den ausgeführten IMPORT FROM SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die zu importierenden Daten liefert. <code>import_spec</code> ist eine Lua Tabelle mit den folgenden Feldern:</p>													

	parameters	Die in dem IMPORT Befehl angegebenen Parameter. Zum Beispiel könnte der Wert für den Parameter FOO durch <code>import_spec.parameters.FOO</code> gelesen werden. Der Wert <code>nil</code> wird zurückgegeben, falls der Parameter nicht vom Nutzer spezifiziert wurde.
	is_subselect	Genau dann wahr, wenn der IMPORT Befehl in einem SELECT benutzt wird, und nicht in einem IMPORT INTO table Befehl.
	subselect_column_names	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Namen der spezifizierten Ausgabespalten zurückgegeben. Der Wert <code>nil</code> wird zurückgegeben wenn die Ausgabespalten nicht spezifiziert wurden.
	subselect_column_types	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Typen der spezifizierten Ausgabespalten zurückgegeben. Die Typen sind im SQL Format angegeben (z.B. <code>"VARCHAR(100)"</code>). Der Wert <code>nil</code> wird zurückgegeben wenn die Ausgabespalten nicht spezifiziert wurden.
	connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.
	connection	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird eine Lua Tabelle zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code> . Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.
		<p> Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das <code>connection_name</code> Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können</p>

	dann über <code>exa.get_connection(name)</code> abgefragt werden.														
Funktion <code>genera - te_sql_for_export_spec()</code> für benutzerdefinierten Export (siehe Abschnitt 3.4.5, Benutzerdefinierter EXPORT mittels UDFs)	<p>Damit ein UDF Skript in einem benutzerdefinierten Export (EXPORT INTO SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>genera - te_sql_for_export_spec(export_spec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>export_spec</code> enthält alle Informationen über den ausgeführten EXPORT INTO SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die Daten (in der Regel mittels UDFs) exportiert. Der FROM-Teil dieses Strings kann eine Dummy-Tabelle sein (z.B. DUAL), denn der Export-Befehl weiß ja bereits, welche Tabelle exportiert werden soll. Sie muss dennoch angegeben werden, damit dieser SQL-String erfolgreich kompiliert werden kann.</p> <p><code>export_spec</code> ist eine Lua Tabelle mit den folgenden Feldern:</p> <table> <tr> <td>parameters</td> <td>Die in dem EXPORT Befehl angegebenen Parameter. Zum Beispiel könnte der Wert für den Parameter <code>FOO</code> durch <code>export_spec.parameters.FOO</code> gelesen werden. Der Wert <code>nil</code> wird zurückgegeben, falls der Parameter nicht vom Nutzer spezifiziert wurde.</td> </tr> <tr> <td>source_column_names</td> <td>Liste der Spaltennamen der zu exportierenden Daten.</td> </tr> <tr> <td>has_truncate</td> <td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.</td> </tr> <tr> <td>has_replace</td> <td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).</td> </tr> <tr> <td>created_by</td> <td>String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.</td> </tr> <tr> <td>connection_name</td> <td>Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</td> </tr> <tr> <td>connection</td> <td>Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird eine Lua Tabelle zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code>. Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</td> </tr> </table> <p>! Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung</p>	parameters	Die in dem EXPORT Befehl angegebenen Parameter. Zum Beispiel könnte der Wert für den Parameter <code>FOO</code> durch <code>export_spec.parameters.FOO</code> gelesen werden. Der Wert <code>nil</code> wird zurückgegeben, falls der Parameter nicht vom Nutzer spezifiziert wurde.	source_column_names	Liste der Spaltennamen der zu exportierenden Daten.	has_truncate	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.	has_replace	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).	created_by	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.	connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.	connection	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird eine Lua Tabelle zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code> . Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.
parameters	Die in dem EXPORT Befehl angegebenen Parameter. Zum Beispiel könnte der Wert für den Parameter <code>FOO</code> durch <code>export_spec.parameters.FOO</code> gelesen werden. Der Wert <code>nil</code> wird zurückgegeben, falls der Parameter nicht vom Nutzer spezifiziert wurde.														
source_column_names	Liste der Spaltennamen der zu exportierenden Daten.														
has_truncate	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.														
has_replace	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).														
created_by	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.														
connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.														
connection	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird eine Lua Tabelle zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code> . Der Wert <code>nil</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.														

(CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das connection_name Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über exa.get_connection(name) abgefragt werden.

Beispiel:

```
/*
This example loads from a webserver
and processes the following file goalies.xml:

<?xml version='1.0' encoding='UTF-8'?>
<users>
    <user active="1">
        <first_name>Manuel</first_name>
        <last_name>Neuer</last_name>
    </user>
    <user active="1">
        <first_name>Joe</first_name>
        <last_name>Hart</last_name>
    </user>
    <user active="0">
        <first_name>Oliver</first_name>
        <last_name>Kahn</last_name>
    </user>
</users>
*/
CREATE LUA SCALAR SCRIPT process_users(url VARCHAR(500))
EMITS (firstname VARCHAR(20), lastname VARCHAR(20)) AS
require("lxp")
http = require("socket.http")
local in_user_tag = false;
local in_first_name_tag = false;
local in_last_name_tag = false;
local current = {}
local users = {}

p = lxp.new(
    {StartElement = function(p, tag, attr)
        if tag == "user" and attr.active == "1" then in_user_tag = true; end
        if tag == "first_name" then in_first_name_tag = true; end
        if tag == "last_name" then in_last_name_tag = true; end
    end,
    EndElement = function(p, tag)
        if tag == "user" then in_user_tag = false; end
        if tag == "first_name" then in_first_name_tag = false; end
        if tag == "last_name" then in_last_name_tag = false; end
    end,
    CharacterData = function(p, txt)
        if in_user_tag then
            if in_first_name_tag then current.first_name = txt; end
            if in_last_name_tag then current.last_name = txt; end
        end
    end
})
```

```

        end
    if current.first_name and current.last_name then
        users[#users+1] = current
        current = {}
    end
end} )

function run(ctx)
    content = http.request(ctx.url)
    p:parse(content); p:parse(); p:close();
    for i=1, #users do
        ctx.emit(users[i].first_name, users[i].last_name)
    end
end
/
SELECT process_users ('http://www.my_valid_webserver/goalies.xml')
FROM DUAL;

```

Java

Neben den folgenden Informationen finden Sie weitere Details zu Java in der offiziellen Dokumentation.

Java Skript-Hauptklasse	<p>Falls nicht anders angegeben, muss die Skript-Hauptklasse mit den unten aufgelisteten Methoden (<code>run()</code> sowie optional <code>init()</code> und <code>cleanup()</code>) denselben Namen besitzen wie der Name des Skripts (bitte beachten Sie die Regeln für Bezeichner und deren Groß-/Kleinschreibung).</p> <p>Sie können die Hauptklasse auch explizit durch das Schlüsselwort <code>%scriptclass</code> angeben (z.B. <code>%scriptclass com.mycompany.MyScriptClass;</code>)</p> <ul style="list-style-type: none">  Alle Klassen, die direkt im Skript angegeben werden, befinden sich implizit im Java package <code>com.exasol</code>, indem der entsprechende Befehl <code>package com.exasol;</code> implizit am Anfang jedes Skriptes hinzugefügt wird.  Alle Callback Funktionen (<code>run()</code>, <code>init()</code>, <code>cleanup()</code>, <code>getDefaultOutputColumns()</code> und <code>generateSqlForImportSpec()</code>) müssen in der Hauptklasse implementiert werden.
Nutzung des Maven-Repositories	<p>Exasol's Java API für Skripte ist in Maven verfügbar, damit Sie Java-Code leichter entwickeln können. Fügen Sie hierfür bitte das folgende Repository und folgende Dependency in der Projekt-Konfiguration Ihres Build-Tools hinzu (z.B. <code>pom.xml</code> bei Maven):</p> <pre> <repositories> <repository> <id>maven.exasol.com</id> <url>https://maven.exasol.com/artifactory/exasol-releases</url> <snapshots> <enabled>false</enabled> </snapshots> </repository> </repositories> <dependencies> <dependency> <groupId>com.exasol</groupId> <artifactId>exasol-script-api</artifactId> <version>6.0.0</version> </dependency> </pre>

	<pre></dependency> </dependencies></pre>																
run(ExaMetadata, ExaIterator), init(ExaMetadata) und cleanup() Methoden	<p>Die Methode <code>static <Type> run(ExaMetadata, ExaIterator)</code> wird für jeden Datensatz (SCALAR) bzw. jede Gruppe (SET) aufgerufen. Deren Überabeparameter sind einerseits ein Metadaten-Objekt (ExaMetadata) sowie der Iterator für den Datenzugriff (ExaIterator).</p> <p>Zum Initialisieren aufwändiger Dinge (z.B. externe Verbindungen öffnen) können Sie Code in der Methode <code>static void init(ExaMetadata)</code> definieren. Dieser wird je virtueller Maschine einmal am Anfang der SELECT-Ausführung aufgerufen. Zum Deinitialisieren existiert die Methode <code>static void cleanup(ExaMetadata)</code>, welche ganz zum Schluss der Ausführung einmal pro virtueller Maschine aufgerufen wird.</p>																
Daten-Iterator, next(), emit(), size() und reset()	<p>Folgende Methoden stehen in der Klasse <code>ExaIterator</code> zur Verfügung:</p> <ul style="list-style-type: none"> • <code>next()</code> • <code>emit(Object... values)</code> • <code>reset()</code> • <code>size()</code> • <code>getInteger(String name)</code> bzw. <code>getInteger(int column)</code> • <code>getLong(String name)</code> bzw. <code>getLong(int column)</code> • <code>getBigDecimal(String name)</code> bzw. <code>getBigDecimal(int column)</code> • <code>getDouble(String name)</code> bzw. <code>getDouble(int column)</code> • <code>getString(String name)</code> bzw. <code>getString(int column)</code> • <code>getBoolean(String name)</code> bzw. <code>getBoolean(int column)</code> • <code>getDate(String name)</code> bzw. <code>getDate(int column)</code> • <code>getTimestamp(String name)</code> bzw. <code>getTimestamp(int column)</code> <p>Für Skripte mit mehreren Eingabewerten pro Aufruf (Schlüsselwort SET) wird mittels der <code>next()</code>-Methode durch die Daten iteriert. Initial zeigt der Iterator auf die erste Eingabezeile. Zum Iterieren bietet sich eine <code>while(true)</code> Schleife an, in der zum Ende im Falle <code>if(!ctx.next())</code> abgebrochen wird.</p> <p> Falls die Eingabemenge leer ist, wird die <code>run()</code>-Methode gar nicht aufgerufen. Als Ergebnis wird dann analog zu Aggregatsfunktionen der Wert NULL zurückgeliefert (wie z.B. in <code>SELECT MAX(x) FROM t WHERE false</code>).</p>																
	<p>Zusätzlich gibt es eine Methode <code>reset()</code>, mit der der Iterator auf den ersten Wert zurückgesetzt werden kann. Damit sind auch mehrere Durchläufe durch die Daten möglich, falls das Ihr Algorithmus benötigt.</p> <p>Die Anzahl der Eingabedaten lässt sich über die Methode <code>size()</code> abfragen.</p> <p>Mehrere Ergebnissezeilen (Schlüsselwort EMITS) können über die Methode <code>emit()</code> erzeugt werden. Die Funktion erwartet so viele Parameter wie Ausgabe-Spalten. Im Falle von dynamischen Ausgabe-Parametern bietet sich in Java die Nutzung eines Object Array an (Beispiel: <code>iter.emit(new Object[] {1, "a"})</code>).</p>																
Parameter	<p>Beachten Sie, dass die Java-Datentypen und die Datenbank-SQL-Typen nicht identisch sind. Daher werden bei den Ein- und Ausgabedaten folgende Konvertierungen durchgeführt:</p> <table> <tbody> <tr> <td>DECIMAL(p,0)</td> <td>Integer, Long, BigDecimal</td> </tr> <tr> <td>DECIMAL(p,s)</td> <td>BigDecimal</td> </tr> <tr> <td>DOUBLE</td> <td>Double</td> </tr> <tr> <td>DATE</td> <td>java.sql.Date</td> </tr> <tr> <td>TIMESTAMP</td> <td>java.sql.Timestamp</td> </tr> <tr> <td>BOOLEAN</td> <td>Boolean</td> </tr> <tr> <td>VARCHAR bzw. CHAR</td> <td>String</td> </tr> <tr> <td>Sonstige</td> <td>Nicht unterstützt</td> </tr> </tbody> </table>	DECIMAL(p,0)	Integer, Long, BigDecimal	DECIMAL(p,s)	BigDecimal	DOUBLE	Double	DATE	java.sql.Date	TIMESTAMP	java.sql.Timestamp	BOOLEAN	Boolean	VARCHAR bzw. CHAR	String	Sonstige	Nicht unterstützt
DECIMAL(p,0)	Integer, Long, BigDecimal																
DECIMAL(p,s)	BigDecimal																
DOUBLE	Double																
DATE	java.sql.Date																
TIMESTAMP	java.sql.Timestamp																
BOOLEAN	Boolean																
VARCHAR bzw. CHAR	String																
Sonstige	Nicht unterstützt																

	<p> Der Wert <code>null</code> stellt das Gegenstück zur SQL-NULL dar.</p> <p>Die Eingabedaten können über get-Methoden adressiert werden, also z.B. <code>ctx.getString("url")</code>.</p> <p>Falls Sie eine dynamische Anzahl von Parametern verwenden wollen, können Sie dies über die Notation <code>(...)</code> erreichen, also z.B. <code>CREATE JAVA SCALAR SCRIPT my_script (...)</code>. Sie können die einzelnen Parameter dann einfach über einen Index zugreifen, z.B. <code>ctx.getString(0)</code> für den ersten Parameter. Die Anzahl der Parameter sowie ihre Datentypen (werden während dem Aufruf bestimmt) sind Bestandteil der Metadaten.</p>																																																								
Metadaten	<p>Folgende Metadaten stehen in dem Objekt <code>ExaMetadata</code> zur Verfügung:</p> <table> <tbody> <tr> <td><code>String getDatabaseName()</code></td><td>Datenbankname</td></tr> <tr> <td><code>String getDatabaseVersion()</code></td><td>Datenbankversion</td></tr> <tr> <td><code>String getScriptLanguage()</code></td><td>Name und Version der Sprache</td></tr> <tr> <td><code>String getScriptName()</code></td><td>Name des Skripts</td></tr> <tr> <td><code>String getScriptSchema()</code></td><td>Schema, in dem das Skript liegt</td></tr> <tr> <td><code>String getCurrentSchema()</code></td><td>Schema, das aktuell geöffnet ist</td></tr> <tr> <td><code>String getScriptCode()</code></td><td>Text des Skripts</td></tr> <tr> <td><code>String getSessionId()</code></td><td>Session-Id</td></tr> <tr> <td><code>long getStatementId()</code></td><td>Statement-Id innerhalb der Session</td></tr> <tr> <td><code>long getCurrentUser()</code></td><td>Aktueller Nutzer</td></tr> <tr> <td><code>long getNodeCount()</code></td><td>Anzahl an Knoten im Cluster</td></tr> <tr> <td><code>longgetNodeId()</code></td><td>Lokale Knoten-Id beginnend bei 0</td></tr> <tr> <td><code>String getVmId()</code></td><td>Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)</td></tr> <tr> <td><code>BigInteger getMemoryLimit()</code></td><td>Speicherlimit des Java-Prozesses</td></tr> <tr> <td><code>String getInputType()</code></td><td>Typ der Eingabedaten (SCALAR oder SET)</td></tr> <tr> <td><code>long getInputColumnCount()</code></td><td>Anzahl der Eingabe-Spalten</td></tr> <tr> <td><code>String getInputColumnName(int column)</code></td><td>Name einer Eingabe-Spalte</td></tr> <tr> <td><code>Class getInputColumnType(int column)</code></td><td>Typ einer Eingabe-Spalte</td></tr> <tr> <td><code>String getInputColumnSqlType(int column)</code></td><td>SQL-Typ einer Eingabe-Spalte</td></tr> <tr> <td><code>long getInputColumnPrecision(int column)</code></td><td>Precision einer Eingabe-Spalte</td></tr> <tr> <td><code>long getInputColumnScale(int column)</code></td><td>Scale einer Eingabe-Spalte</td></tr> <tr> <td><code>long getInputColumnLength(int column)</code></td><td>Länge einer Eingabe-Spalte</td></tr> <tr> <td><code>String getOutputType()</code></td><td>Typ der Ausgabedaten (RETURNS oder EMITS)</td></tr> <tr> <td><code>long getOutputColumnCount()</code></td><td>Anzahl der Ausgabe-Spalten</td></tr> <tr> <td><code>String getOutputColumnName(int column)</code></td><td>Name einer Ausgabe-Spalte</td></tr> <tr> <td><code>Class getOutputColumnType(int column)</code></td><td>Typ einer Ausgabe-Spalte</td></tr> <tr> <td><code>String getOutputColumnSqlType(int column)</code></td><td>SQL-Typ einer Ausgabe-Spalte</td></tr> <tr> <td><code>long getOutputColumnPrecision(int column)</code></td><td>Precision einer Ausgabe-Spalte</td></tr> </tbody> </table>	<code>String getDatabaseName()</code>	Datenbankname	<code>String getDatabaseVersion()</code>	Datenbankversion	<code>String getScriptLanguage()</code>	Name und Version der Sprache	<code>String getScriptName()</code>	Name des Skripts	<code>String getScriptSchema()</code>	Schema, in dem das Skript liegt	<code>String getCurrentSchema()</code>	Schema, das aktuell geöffnet ist	<code>String getScriptCode()</code>	Text des Skripts	<code>String getSessionId()</code>	Session-Id	<code>long getStatementId()</code>	Statement-Id innerhalb der Session	<code>long getCurrentUser()</code>	Aktueller Nutzer	<code>long getNodeCount()</code>	Anzahl an Knoten im Cluster	<code>longgetNodeId()</code>	Lokale Knoten-Id beginnend bei 0	<code>String getVmId()</code>	Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)	<code>BigInteger getMemoryLimit()</code>	Speicherlimit des Java-Prozesses	<code>String getInputType()</code>	Typ der Eingabedaten (SCALAR oder SET)	<code>long getInputColumnCount()</code>	Anzahl der Eingabe-Spalten	<code>String getInputColumnName(int column)</code>	Name einer Eingabe-Spalte	<code>Class getInputColumnType(int column)</code>	Typ einer Eingabe-Spalte	<code>String getInputColumnSqlType(int column)</code>	SQL-Typ einer Eingabe-Spalte	<code>long getInputColumnPrecision(int column)</code>	Precision einer Eingabe-Spalte	<code>long getInputColumnScale(int column)</code>	Scale einer Eingabe-Spalte	<code>long getInputColumnLength(int column)</code>	Länge einer Eingabe-Spalte	<code>String getOutputType()</code>	Typ der Ausgabedaten (RETURNS oder EMITS)	<code>long getOutputColumnCount()</code>	Anzahl der Ausgabe-Spalten	<code>String getOutputColumnName(int column)</code>	Name einer Ausgabe-Spalte	<code>Class getOutputColumnType(int column)</code>	Typ einer Ausgabe-Spalte	<code>String getOutputColumnSqlType(int column)</code>	SQL-Typ einer Ausgabe-Spalte	<code>long getOutputColumnPrecision(int column)</code>	Precision einer Ausgabe-Spalte
<code>String getDatabaseName()</code>	Datenbankname																																																								
<code>String getDatabaseVersion()</code>	Datenbankversion																																																								
<code>String getScriptLanguage()</code>	Name und Version der Sprache																																																								
<code>String getScriptName()</code>	Name des Skripts																																																								
<code>String getScriptSchema()</code>	Schema, in dem das Skript liegt																																																								
<code>String getCurrentSchema()</code>	Schema, das aktuell geöffnet ist																																																								
<code>String getScriptCode()</code>	Text des Skripts																																																								
<code>String getSessionId()</code>	Session-Id																																																								
<code>long getStatementId()</code>	Statement-Id innerhalb der Session																																																								
<code>long getCurrentUser()</code>	Aktueller Nutzer																																																								
<code>long getNodeCount()</code>	Anzahl an Knoten im Cluster																																																								
<code>longgetNodeId()</code>	Lokale Knoten-Id beginnend bei 0																																																								
<code>String getVmId()</code>	Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)																																																								
<code>BigInteger getMemoryLimit()</code>	Speicherlimit des Java-Prozesses																																																								
<code>String getInputType()</code>	Typ der Eingabedaten (SCALAR oder SET)																																																								
<code>long getInputColumnCount()</code>	Anzahl der Eingabe-Spalten																																																								
<code>String getInputColumnName(int column)</code>	Name einer Eingabe-Spalte																																																								
<code>Class getInputColumnType(int column)</code>	Typ einer Eingabe-Spalte																																																								
<code>String getInputColumnSqlType(int column)</code>	SQL-Typ einer Eingabe-Spalte																																																								
<code>long getInputColumnPrecision(int column)</code>	Precision einer Eingabe-Spalte																																																								
<code>long getInputColumnScale(int column)</code>	Scale einer Eingabe-Spalte																																																								
<code>long getInputColumnLength(int column)</code>	Länge einer Eingabe-Spalte																																																								
<code>String getOutputType()</code>	Typ der Ausgabedaten (RETURNS oder EMITS)																																																								
<code>long getOutputColumnCount()</code>	Anzahl der Ausgabe-Spalten																																																								
<code>String getOutputColumnName(int column)</code>	Name einer Ausgabe-Spalte																																																								
<code>Class getOutputColumnType(int column)</code>	Typ einer Ausgabe-Spalte																																																								
<code>String getOutputColumnSqlType(int column)</code>	SQL-Typ einer Ausgabe-Spalte																																																								
<code>long getOutputColumnPrecision(int column)</code>	Precision einer Ausgabe-Spalte																																																								

	<pre>long getOutputColumnScale(int column) long getOutputColumnLength(int column) Class<?> importScript(String name)</pre>	Scale einer Ausgabe-Spalte Länge einer Ausgabe-Spalte Importieren eines anderen Skripts								
Exceptions	Folgende Exasol-spezifischen Exceptions können geworfen werden: <ul style="list-style-type: none"> • ExaCompilationException • ExaDataException • ExaIterationException 									
Import anderer Skripte	Zum Import anderer Skripte steht neben der importScript()-Methode der Klasse ExaMetadata (siehe oben) auch das Schlüsselwort %import zur Verfügung (z.B. %import OTHER_SCRIPT;). Anschließend steht dieses Skript im Namensraum zur Verfügung (z.B. OTHER_SCRIPT.my_method()).									
Zugriff auf Verbindungsdefinitionen	<p>Die Details von Verbindungen, die mit CREATE CONNECTION definiert wurden können innerhalb von Java UDF Skripten mit Hilfe der Methode ExaMetadata.getConnection(String connectionName) abgefragt werden. Das Ergebnis ist ein Objekt, welches das Interface com.exasol.ExaConnectionInformation implementiert. Dieses bietet folgende Methoden:</p> <table> <tr> <td>ExaConnectionInformation.ConnectionType ExaConnectionInformation.getType()</td><td>Der Typ der Verbindungsdefinition. ConnectionType ist eine Aufzählung, die aktuell nur den Eintrag PASSWORD enthält.</td></tr> <tr> <td>String ExaConnectionInformation.getAddress()</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.</td></tr> <tr> <td>String ExaConnectionInformation.getUser()</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.</td></tr> <tr> <td>String ExaConnectionInformation.getPassword()</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.</td></tr> </table>	ExaConnectionInformation.ConnectionType ExaConnectionInformation.getType()	Der Typ der Verbindungsdefinition. ConnectionType ist eine Aufzählung, die aktuell nur den Eintrag PASSWORD enthält.	String ExaConnectionInformation.getAddress()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.	String ExaConnectionInformation.getUser()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.	String ExaConnectionInformation.getPassword()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.	
ExaConnectionInformation.ConnectionType ExaConnectionInformation.getType()	Der Typ der Verbindungsdefinition. ConnectionType ist eine Aufzählung, die aktuell nur den Eintrag PASSWORD enthält.									
String ExaConnectionInformation.getAddress()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.									
String ExaConnectionInformation.getUser()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.									
String ExaConnectionInformation.getPassword()	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.									
Integration eigener JAR-Pakete	Siehe Abschnitt 3.6.5, Skript-Sprachen erweitern mittels BucketFS									
JVM Optionen	<p>Um die Skript-Performance je nach Speicherbedarf des Skripts zu optimieren, können Sie folgende Java VM Optionen setzen:</p> <ul style="list-style-type: none"> • Initiale Heap-Größe (-Xms) • Maximale Heap-Größe (-Xmx) • Thread Stack-Größe (-Xss) <p>Beispiel: %jvmoption -Xms128m -Xmx1024m -Xss512k;</p> <p>Dies setzt die initiale Heap-Größe auf 128 MB, die maximale Heap-Größe auf 1024 MB und die Thread Stack-Größe auf 512 kB.</p> <p>Bitte beachten Sie, dass beim mehrmaligen Setzen (z.B. durch Importieren anderer Skripte) die jeweils letzte Einstellung verwendet wird.</p>									
Funktion getDefaultOutputColumns(ExaMetadata) für	Falls ein UDF Skript mit dynamischen Ausgabe-Parametern definiert wurde und die Ausgabe-Parameter nicht bestimmt werden können (über EMITS in der Abfrage oder über INSERT INTO), ruft die Datenbank die Funktion static String getDefaultOutputColumns(ExaMetadata exa) auf, die Sie implementieren können. Der erwartete Rückgabewert ist ein String mit den Namen und Typen der Ausgabe-Parameter, z.B. "a int, b varchar(100)". Siehe Dynamische Einführung									

dynamische Ausgabe-Parameter	<p>gabe- und Ausgabe-Parameter für eine genaue Beschreibung wann diese Methode aufgerufen wird und für Beispiele.</p>
	 Diese Methode wird genau einmal auf einem einzigen Knoten ausgeführt.
	 Diese Funktion muss in der Hauptklasse implementiert werden.
Funktion generateSqlFor ImportSpec() für benutzerdefinierten Import (siehe Abschnitt 3.4.4 , Benutzerdefinierter IMPORT mittels UDFs)	<p>Damit ein UDF Skript in einem benutzerdefinierten (IMPORT FROM SCRIPT) angegeben werden kann müssen Sie die Funktion <code>public static String generateSqlForImportSpec(ExaMetadata meta, ExaImportSpecification importSpec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>importSpec</code> enthält alle Informationen über den ausführten IMPORT FROM SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die zu importierenden Daten liefert. <code>importSpec</code> ist ein Objekt mit den folgenden Methoden:</p>
	<code>Map<String, String> getParameters()</code>
	<code>boolean isSubselect()</code>
	<code>List<String> getSubselectColumnNames()</code>
	<code>List<String> getSubselectColumnTypes()</code>
	<code>boolean hasConnectionName()</code>
	<code>String getConnectionName()</code>
	<code>boolean hasConnectionInformation()</code>
	<code>ExaConnectionInformation getConnectionInformation()</code>
	 Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das <code>connection_name</code> Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über <code>ExaMetadata.getConnection(connectionName)</code> abgefragt werden.
	 Diese Funktion muss in der Hauptklasse implementiert werden.
Funktion generateSqlFor ExportSpec()	<p>Damit ein UDF Skript in einem benutzerdefinierten Export (EXPORT INTO SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>public static String generateSqlForExportSpec(ExaMetadata meta, ExaExportSpecification export_spec)</code> imple-</p>

<p><code>portSpec()</code> für benutzerdefinierten Export (siehe Abschnitt 3.4.5, Benutzerdefinierter EXPORT mittels UDFs)</p>	<p>mentieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>export_spec</code> enthält alle Informationen über den ausgeführten EXPORT INTO SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die Daten (in der Regel mittels UDFs) exportiert. Der FROM-Teil dieses Strings kann eine Dummy-Tabelle sein (z.B. DUAL), denn der Export-Befehl weiß ja bereits, welche Tabelle exportiert werden soll. Sie muss dennoch angegeben werden, damit dieser SQL-String erfolgreich kompiliert werden kann.</p> <p><code>export_spec</code> ist ein Objekt mit den folgenden Methoden:</p> <table border="0"> <tr> <td><code>Map<String, String> getParameters()</code></td><td>Die in dem EXPORT Befehl angegebenen Parameter.</td></tr> <tr> <td><code>List<String> getSourceColumnNames()</code></td><td>Liste der Spaltennamen der zu exportierenden Daten.</td></tr> <tr> <td><code>boolean hasTruncate()</code></td><td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.</td></tr> <tr> <td><code>boolean hasReplace()</code></td><td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).</td></tr> <tr> <td><code>boolean hasCreatedBy()</code></td><td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob für die Tabelle ein Erzeugungstext angegeben wurde.</td></tr> <tr> <td><code>String getCreatedBy()</code></td><td>String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.</td></tr> <tr> <td><code>boolean hasConnectionName()</code></td><td>Genau dann wahr, wenn ein Verbindung angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>ExaMetadata.getConnection(name)</code> abfragen.</td></tr> <tr> <td><code>String getConnectionName()</code></td><td>Der Name der Verbindung, falls eine solche angegeben wurde.</td></tr> <tr> <td><code>boolean hasConnectionInformation()</code></td><td>Genau dann wahr, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Das UDF Skript kann die Verbindungsinformationen dann über <code>getConnectionInformation()</code> abfragen.</td></tr> <tr> <td><code>ExaConnectionInformation getConnectionInformation()</code></td><td>Falls <code>hasConnectionInformation()</code> wahr ist, werden die vom Nutzer angegebenen Verbindungsinformationen zurückgegeben. Siehe weiter oben in dieser Tabelle für die Definition von <code>ExaConnectionInformation</code>.</td></tr> </table> <p>! Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das <code>connection_name</code> Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über <code>ExaMetadata.getConnection(connectionName)</code> abgefragt werden.</p> <p>! Diese Funktion muss in der Hauptklasse implementiert werden.</p>	<code>Map<String, String> getParameters()</code>	Die in dem EXPORT Befehl angegebenen Parameter.	<code>List<String> getSourceColumnNames()</code>	Liste der Spaltennamen der zu exportierenden Daten.	<code>boolean hasTruncate()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.	<code>boolean hasReplace()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).	<code>boolean hasCreatedBy()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob für die Tabelle ein Erzeugungstext angegeben wurde.	<code>String getCreatedBy()</code>	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.	<code>boolean hasConnectionName()</code>	Genau dann wahr, wenn ein Verbindung angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>ExaMetadata.getConnection(name)</code> abfragen.	<code>String getConnectionName()</code>	Der Name der Verbindung, falls eine solche angegeben wurde.	<code>boolean hasConnectionInformation()</code>	Genau dann wahr, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Das UDF Skript kann die Verbindungsinformationen dann über <code>getConnectionInformation()</code> abfragen.	<code>ExaConnectionInformation getConnectionInformation()</code>	Falls <code>hasConnectionInformation()</code> wahr ist, werden die vom Nutzer angegebenen Verbindungsinformationen zurückgegeben. Siehe weiter oben in dieser Tabelle für die Definition von <code>ExaConnectionInformation</code> .
<code>Map<String, String> getParameters()</code>	Die in dem EXPORT Befehl angegebenen Parameter.																				
<code>List<String> getSourceColumnNames()</code>	Liste der Spaltennamen der zu exportierenden Daten.																				
<code>boolean hasTruncate()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.																				
<code>boolean hasReplace()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).																				
<code>boolean hasCreatedBy()</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob für die Tabelle ein Erzeugungstext angegeben wurde.																				
<code>String getCreatedBy()</code>	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.																				
<code>boolean hasConnectionName()</code>	Genau dann wahr, wenn ein Verbindung angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>ExaMetadata.getConnection(name)</code> abfragen.																				
<code>String getConnectionName()</code>	Der Name der Verbindung, falls eine solche angegeben wurde.																				
<code>boolean hasConnectionInformation()</code>	Genau dann wahr, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Das UDF Skript kann die Verbindungsinformationen dann über <code>getConnectionInformation()</code> abfragen.																				
<code>ExaConnectionInformation getConnectionInformation()</code>	Falls <code>hasConnectionInformation()</code> wahr ist, werden die vom Nutzer angegebenen Verbindungsinformationen zurückgegeben. Siehe weiter oben in dieser Tabelle für die Definition von <code>ExaConnectionInformation</code> .																				

Beispiel:

```
/*
This example loads from a webserver
and processes the following file goalies.xml:

<?xml version='1.0' encoding='UTF-8'?>
<users>
    <user active="1">
        <first_name>Manuel</first_name>
```

```

<last_name>Neuer</last_name>
</user>
<user active="1">
    <first_name>Joe</first_name>
    <last_name>Hart</last_name>
</user>
<user active="0">
    <first_name>Oliver</first_name>
    <last_name>Kahn</last_name>
</user>
</users>
*/



CREATE JAVA SCALAR SCRIPT process_users(url VARCHAR(500))
EMITS (firstname VARCHAR(20), lastname VARCHAR(20)) AS

import java.net.URL;
import java.net.URLConnection;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

class PROCESS_USERS {
    static void run(ExaMetadata exa, ExaIterator ctx) throws Exception {
        URL url = new URL(ctx.getString("url"));
        URLConnection conn = url.openConnection();
        DocumentBuilder docBuilder =
            DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document doc = docBuilder.parse(conn.getInputStream());
        NodeList nodes =
            doc.getDocumentElement().getElementsByTagName("user");
        for (int i = 0; i < nodes.getLength(); i++) {
            if (nodes.item(i).getNodeType() != Node.ELEMENT_NODE)
                continue;
            Element elem = (Element)nodes.item(i);
            if (!elem.getAttribute("active").equals("1"))
                continue;
            Node name = elem.getElementsByTagName("first_name").item(0);
            String firstName = name.getChildNodes().item(0).getNodeValue();
            name = elem.getElementsByTagName("last_name").item(0);
            String lastName = name.getChildNodes().item(0).getNodeValue();
            ctx.emit(firstName, lastName);
        }
    }
}
/
SELECT process_users ('http://www.my_valid_webserver/goalies.xml')
FROM DUAL;

FIRSTNAME          LASTNAME
-----  -----
Manuel             Neuer
Joe               Hart

```

Python

Neben den folgenden Informationen finden Sie weitere Details zu Python in der offiziellen Dokumentation (siehe <http://www.python.org>).

<code>run()</code> und <code>cleanup()</code> Methoden	<p>Die Methode <code>run()</code> wird für jeden Datensatz (SCALAR) bzw. jede Gruppe (SET) aufgerufen. Deren Übergabeparameter ist eine Art Ausführungskontext und ermöglicht den Zugriff auf die Daten sowie den Iterator bei SET-Funktionen.</p> <p>Zum Initialisieren aufwändiger Dinge (z.B. externe Verbindungen öffnen) können Sie Code außerhalb der <code>run()</code>-Methode schreiben. Dieser wird je virtueller Maschine einmal am Anfang der SELECT-Ausführung aufgerufen. Zum Deinitialisieren existiert die Methode <code>cleanup()</code>, welche ganz zum Schluss der Ausführung einmal pro virtueller Maschine aufgerufen wird.</p>																								
Parameter	<p>Beachten Sie, dass die Python-Datentypen und die Datenbank-SQL-Typen nicht identisch sind. Daher werden bei den Ein- und Ausgabedaten folgende Konvertierungen durchgeführt:</p> <table> <tbody> <tr><td><code>DECIMAL(p, 0)</code></td><td><code>int</code></td></tr> <tr><td><code>DECIMAL(p, s)</code></td><td><code>decimal.Decimal</code></td></tr> <tr><td><code>DOUBLE</code></td><td><code>float</code></td></tr> <tr><td><code>DATE</code></td><td><code>datetime.date</code></td></tr> <tr><td><code>TIMESTAMP</code></td><td><code>datetime.datetime</code></td></tr> <tr><td><code>BOOLEAN</code></td><td><code>bool</code></td></tr> <tr><td><code>VARCHAR</code> bzw. <code>CHAR</code></td><td><code>unicode</code></td></tr> <tr><td>Sonstige</td><td>Nicht unterstützt</td></tr> </tbody> </table> <p> Aus Performance-Gründen sollten Sie den DOUBLE-Datentyp einem DECIMAL-Typ bevorzugen.</p> <p> Der Wert <code>None</code> stellt das Gegenstück zur SQL-NULL dar.</p> <p>Die Eingabedaten können über den Namen adressiert werden, also z.B. <code>ctx.my_input</code>.</p> <p>Falls Sie eine dynamische Anzahl von Parametern verwenden wollen, können Sie dies über die Notation <code>(...)</code> erreichen, also z.B. <code>CREATE PYTHON SCALAR SCRIPT my_script (...)</code>. Sie können die einzelnen Parameter dann einfach über einen Index zugreifen, z.B. <code>ctx[0]</code> für den ersten Parameter. Die Anzahl der Parameter sowie ihre Datentypen (werden während dem Aufruf bestimmt) sind Bestandteil der Metadaten.</p>	<code>DECIMAL(p, 0)</code>	<code>int</code>	<code>DECIMAL(p, s)</code>	<code>decimal.Decimal</code>	<code>DOUBLE</code>	<code>float</code>	<code>DATE</code>	<code>datetime.date</code>	<code>TIMESTAMP</code>	<code>datetime.datetime</code>	<code>BOOLEAN</code>	<code>bool</code>	<code>VARCHAR</code> bzw. <code>CHAR</code>	<code>unicode</code>	Sonstige	Nicht unterstützt								
<code>DECIMAL(p, 0)</code>	<code>int</code>																								
<code>DECIMAL(p, s)</code>	<code>decimal.Decimal</code>																								
<code>DOUBLE</code>	<code>float</code>																								
<code>DATE</code>	<code>datetime.date</code>																								
<code>TIMESTAMP</code>	<code>datetime.datetime</code>																								
<code>BOOLEAN</code>	<code>bool</code>																								
<code>VARCHAR</code> bzw. <code>CHAR</code>	<code>unicode</code>																								
Sonstige	Nicht unterstützt																								
Metadaten	<p>Folgende Metadaten stehen als globale Variablen zur Verfügung:</p> <table> <tbody> <tr><td><code>exa.meta.database_name</code></td><td>Datenbankname</td></tr> <tr><td><code>exa.meta.database_version</code></td><td>Datenbankversion</td></tr> <tr><td><code>exa.meta.script_language</code></td><td>Name und Version der Sprache</td></tr> <tr><td><code>exa.meta.script_name</code></td><td>Name des Skripts</td></tr> <tr><td><code>exa.meta.script_schema</code></td><td>Schema, in dem das Skript liegt</td></tr> <tr><td><code>exa.meta.current_schema</code></td><td>Schema, das aktuell geöffnet ist</td></tr> <tr><td><code>exa.meta.script_code</code></td><td>Text des Skripts</td></tr> <tr><td><code>exa.meta.session_id</code></td><td>Session-Id</td></tr> <tr><td><code>exa.meta.statement_id</code></td><td>Statement-Id innerhalb der Session</td></tr> <tr><td><code>exa.meta.current_user</code></td><td>Aktueller Nutzer</td></tr> <tr><td><code>exa.meta.node_count</code></td><td>Anzahl an Knoten im Cluster</td></tr> <tr><td><code>exa.meta.node_id</code></td><td>Lokale Knoten-Id beginnend bei 0</td></tr> </tbody> </table>	<code>exa.meta.database_name</code>	Datenbankname	<code>exa.meta.database_version</code>	Datenbankversion	<code>exa.meta.script_language</code>	Name und Version der Sprache	<code>exa.meta.script_name</code>	Name des Skripts	<code>exa.meta.script_schema</code>	Schema, in dem das Skript liegt	<code>exa.meta.current_schema</code>	Schema, das aktuell geöffnet ist	<code>exa.meta.script_code</code>	Text des Skripts	<code>exa.meta.session_id</code>	Session-Id	<code>exa.meta.statement_id</code>	Statement-Id innerhalb der Session	<code>exa.meta.current_user</code>	Aktueller Nutzer	<code>exa.meta.node_count</code>	Anzahl an Knoten im Cluster	<code>exa.meta.node_id</code>	Lokale Knoten-Id beginnend bei 0
<code>exa.meta.database_name</code>	Datenbankname																								
<code>exa.meta.database_version</code>	Datenbankversion																								
<code>exa.meta.script_language</code>	Name und Version der Sprache																								
<code>exa.meta.script_name</code>	Name des Skripts																								
<code>exa.meta.script_schema</code>	Schema, in dem das Skript liegt																								
<code>exa.meta.current_schema</code>	Schema, das aktuell geöffnet ist																								
<code>exa.meta.script_code</code>	Text des Skripts																								
<code>exa.meta.session_id</code>	Session-Id																								
<code>exa.meta.statement_id</code>	Statement-Id innerhalb der Session																								
<code>exa.meta.current_user</code>	Aktueller Nutzer																								
<code>exa.meta.node_count</code>	Anzahl an Knoten im Cluster																								
<code>exa.meta.node_id</code>	Lokale Knoten-Id beginnend bei 0																								

	<p>exa.meta.vm_id</p> <p>exa.meta.input_type</p> <p>exa.meta.input_column_count exa.meta.input_columns[]</p> <p>exa.meta.output_type</p> <p>exa.meta.output_column_count exa.meta.output_columns[]</p>	<p>Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)</p> <p>Typ der Eingabedaten (SCALAR oder SET)</p> <p>Anzahl der Eingabe-Spalten Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}</p> <p>Typ der Ausgabedaten (RETURNS oder EMITS)</p> <p>Anzahl der Ausgabe-Spalten Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}</p>						
Daten-Iterator, <code>next()</code> , <code>size()</code> und <code>reset()</code>	Für Skripte mit mehreren Eingabewerten pro Aufruf (Schlüsselwort SET) wird mittels der <code>next()</code> -Methode durch die Daten iteriert (auf die Methode <code>next()</code> wird über den Kontext zugegriffen). Initial zeigt der Iterator auf die erste Eingabezeile. Zum Iterieren bietet sich eine while True Schleife an, in der zum Ende im Falle <code>if not ctx.next()</code> abgebrochen wird.	<p> Falls die Eingabemenge leer ist, wird die <code>run()</code>-Methode gar nicht aufgerufen. Als Ergebnis wird dann analog zu Aggregatsfunktionen der Wert NULL zurückgeliefert (wie z.B. in <code>SELECT MAX(x) FROM t WHERE false</code>).</p> <p>Zusätzlich gibt es eine Methode <code>reset()</code>, mit der der Iterator auf den ersten Wert zurückgesetzt werden kann. Damit sind auch mehrere Durchläufe durch die Daten möglich, falls das Ihr Algorithmus benötigt.</p> <p>Die Anzahl der Eingabedaten lässt sich über die Methode <code>size()</code> abfragen.</p>						
<code>emit()</code>	Mehrere Ergebnissezeilen (Schlüsselwort EMITS) können über die Methode <code>emit()</code> erzeugt werden. Die Funktion erwartet so viele Parameter wie Ausgabe-Spalten. Im Falle von dynamischen Ausgabe-Parametern bietet sich in Python die Nutzung eines <code>list</code> Objekts, welches Sie dann mit * referenzieren können (wie im Beispiel oben: <code>ctx.emit(*currentRow)</code>).							
Import anderer Skripte	Zum Import anderer Skripte steht die Methode <code>exa.import_script()</code> zur Verfügung. Der Rückgabewert dieser Methode muss einer Variablen zugewiesen werden, die anschließend das importierte Modul darstellt.							
Zugriff auf Verbindungsdefinitionen	<p>Die Details von Verbindungen, die mit CREATE CONNECTION definiert wurden können innerhalb von Python UDF Skripten mit Hilfe der Methode <code>exa.get_connection("connection_name")</code> abgefragt werden. Das Ergebnis ist ein Python Objekt mit den folgenden Feldern:</p> <table> <tr> <td>type</td><td>Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.</td></tr> <tr> <td>address</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.</td></tr> <tr> <td>user</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.</td></tr> </table>	type	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.	address	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.	user	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.	
type	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.							
address	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.							
user	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.							

	password	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.																
Hilfsbibliotheken		<p>Folgende zusätzliche Bibliotheken stehen zur Verfügung, die nicht bereits im Sprachumfang enthalten sind:</p> <table> <tr> <td>cjson</td><td>Bearbeitung von JSON-Objekten (siehe auch http://pypi.python.org/pypi/python-cjson)</td></tr> <tr> <td>lxml</td><td>Bearbeitung von XML-Strukturen (siehe auch http://pypi.python.org/pypi/lxml)</td></tr> <tr> <td>NumPy</td><td>Numerische Berechnungen (siehe auch http://www.scipy.org)</td></tr> <tr> <td>PyTables</td><td>Hierarchisches Datenbank-Paket (siehe auch http://www.pytables.org). Für diese Bibliothek haben wir zusätzlich das benötigte Paket HDF5 hinzugefügt (http://www.h5py.org).</td></tr> <tr> <td>pytz</td><td>Zeitzonen-Funktionen (siehe auch http://pytz.sourceforge.net)</td></tr> <tr> <td>redis</td><td>Schnittstelle zu Redis (siehe auch http://pypi.python.org/pypi/redis/)</td></tr> <tr> <td>scikit-learn</td><td>Maschinelles Lernen (siehe auch http://scikit-learn.org)</td></tr> <tr> <td>SciPy</td><td>Wissenschaftliche Berechnungen (siehe auch http://www.scipy.org). Für diese Bibliothek haben wir zusätzlich das benötigte Build-Tool atlas hinzugefügt (http://pypi.python.org/pypi/atlas).</td></tr> </table>	cjson	Bearbeitung von JSON-Objekten (siehe auch http://pypi.python.org/pypi/python-cjson)	lxml	Bearbeitung von XML-Strukturen (siehe auch http://pypi.python.org/pypi/lxml)	NumPy	Numerische Berechnungen (siehe auch http://www.scipy.org)	PyTables	Hierarchisches Datenbank-Paket (siehe auch http://www.pytables.org). Für diese Bibliothek haben wir zusätzlich das benötigte Paket HDF5 hinzugefügt (http://www.h5py.org).	pytz	Zeitzonen-Funktionen (siehe auch http://pytz.sourceforge.net)	redis	Schnittstelle zu Redis (siehe auch http://pypi.python.org/pypi/redis/)	scikit-learn	Maschinelles Lernen (siehe auch http://scikit-learn.org)	SciPy	Wissenschaftliche Berechnungen (siehe auch http://www.scipy.org). Für diese Bibliothek haben wir zusätzlich das benötigte Build-Tool atlas hinzugefügt (http://pypi.python.org/pypi/atlas).
cjson	Bearbeitung von JSON-Objekten (siehe auch http://pypi.python.org/pypi/python-cjson)																	
lxml	Bearbeitung von XML-Strukturen (siehe auch http://pypi.python.org/pypi/lxml)																	
NumPy	Numerische Berechnungen (siehe auch http://www.scipy.org)																	
PyTables	Hierarchisches Datenbank-Paket (siehe auch http://www.pytables.org). Für diese Bibliothek haben wir zusätzlich das benötigte Paket HDF5 hinzugefügt (http://www.h5py.org).																	
pytz	Zeitzonen-Funktionen (siehe auch http://pytz.sourceforge.net)																	
redis	Schnittstelle zu Redis (siehe auch http://pypi.python.org/pypi/redis/)																	
scikit-learn	Maschinelles Lernen (siehe auch http://scikit-learn.org)																	
SciPy	Wissenschaftliche Berechnungen (siehe auch http://www.scipy.org). Für diese Bibliothek haben wir zusätzlich das benötigte Build-Tool atlas hinzugefügt (http://pypi.python.org/pypi/atlas).																	
Funktion <code>default_output_columns()</code> für dynamische Ausgabe-Parameter		<p>Falls ein UDF Skript mit dynamischen Ausgabe-Parametern definiert wurde und die Ausgabe-Parameter nicht bestimmt werden können (über EMITS in der Abfrage oder über INSERT INTO), ruft die Datenbank die Funktion <code>default_output_columns()</code> auf, die Sie implementieren können. Der erwartete Rückgabewert ist ein String mit den Namen und Typen der Ausgabe-Parameter, z.B. "a int, b varchar(100)". Siehe Dynamische Eingabe- und Ausgabe-Parameter für eine genaue Beschreibung wann diese Methode aufgerufen wird und für Beispiele.</p> <p> Sie können die Metadaten über <code>exa.meta</code> abrufen, z.B. um die Anzahl und die Typen der Eingabe-Parameter zu abzufragen.</p> <p> Diese Methode wird genau einmal auf einem einzigen Knoten ausgeführt.</p>																
Funktion <code>generate_sql_for_import_spec()</code> für benutzerdefinierten Import (siehe Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs)		<p>Damit ein UDF Skript in einem benutzerdefinierten Import (IMPORT FROM SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>generate_sql_for_import_spec(import_spec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>import_spec</code> enthält alle Informationen über den ausgeführten IMPORT FROM SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die zu importierenden Daten liefert.</p> <p><code>import_spec</code> hat die folgenden Felder:</p> <table> <tr> <td>parameters[]</td> <td>Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO.</td> </tr> </table>	parameters[]	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO.														
parameters[]	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO.																	

	is_subselect	ter FOO. Falls FOO nicht angegeben wurde wird None zurückgegeben. Genau dann wahr, wenn der IMPORT Befehl in einem SELECT benutzt wird, und nicht in einem IMPORT INTO table Befehl.
	subselect_column_names[]	Falls is_subselect wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)) wird eine Liste von Strings mit den Namen der spezifizierten Ausgabespalten zurückgegeben.
	subselect_column_types[]	Falls is_subselect wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)) wird eine Liste von Strings mit den Typen der spezifizierten Ausgabespalten zurückgegeben. Die Typen sind im SQL Format angegeben (z.B. "VARCHAR(100)").
	connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über exa.get_connection(name) abfragen. Der Wert None wird zurückgegeben wenn keine Verbindung angegeben wurde.
	connection	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird ein Objekt zurückgegeben das die gleiche Struktur hat wie der Rückgabewert von exa.get_connection(name). Der Wert None wird zurückgegeben wenn keine Verbindung angegeben wurde.
	<p>! Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das connection_name Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über exa.get_connection(name) abgefragt werden.</p>	
Funktion generate_sql_for_export_spec() für benutzerdefinierten Export (siehe	Damit ein UDF Skript in einem benutzerdefinierten Export (EXPORT INTO SCRIPT) angegeben werden kann, müssen Sie die Funktion generate_sql_for_export_spec(export_spec) implementieren. Bitte	

<p>Abschnitt 3.4.5, Benutzerdefinierter EXPORT mittels UDFs</p>	<p>beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>export_spec</code> enthält alle Informationen über den ausgeführten EXPORT INTO SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die Daten (in der Regel mittels UDFs) exportiert. Der FROM-Teil dieses Strings kann eine Dummy-Tabelle sein (z.B. DUAL), denn der Export-Befehl weiß ja bereits, welche Tabelle exportiert werden soll. Sie muss dennoch angegeben werden, damit dieser SQL-String erfolgreich kompiliert werden kann.</p> <p><code>export_spec</code> hat die folgenden Felder:</p> <table border="0"> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>parameters[]</code></td><td>Die in dem EXPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO. Falls FOO nicht angegeben wurde wird <code>None</code> zurückgegeben.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>source_column_names[]</code></td><td>Liste der Spaltennamen der zu exportierenden Daten.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>has_truncate</code></td><td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>has_replace</code></td><td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>created_by</code></td><td>String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>connection_name</code></td><td>Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</td></tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>connection</code></td><td>Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird ein Objekt zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code>. Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</td></tr> </table> <p>! Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das <code>connection_name</code> Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über <code>exa.get_connection(name)</code> abgefragt werden.</p>	<code>parameters[]</code>	Die in dem EXPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO. Falls FOO nicht angegeben wurde wird <code>None</code> zurückgegeben.	<code>source_column_names[]</code>	Liste der Spaltennamen der zu exportierenden Daten.	<code>has_truncate</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.	<code>has_replace</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).	<code>created_by</code>	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.	<code>connection_name</code>	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.	<code>connection</code>	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird ein Objekt zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code> . Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.
<code>parameters[]</code>	Die in dem EXPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters['FOO']</code> den Wert für den Parameter FOO. Falls FOO nicht angegeben wurde wird <code>None</code> zurückgegeben.														
<code>source_column_names[]</code>	Liste der Spaltennamen der zu exportierenden Daten.														
<code>has_truncate</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.														
<code>has_replace</code>	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).														
<code>created_by</code>	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.														
<code>connection_name</code>	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über <code>exa.get_connection(name)</code> abfragen. Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.														
<code>connection</code>	Dieser Wert ist nur dann definiert, wenn der Nutzer direkt Verbindungsinformationen angegeben hat. Es wird ein Objekt zurückgegeben die die gleiche Struktur hat wie der Rückgabewert von <code>exa.get_connection(name)</code> . Der Wert <code>None</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.														

Beispiel:

```
/*
This example loads from a webserver
and processes the following file goalies.xml:

<?xml version='1.0' encoding='UTF-8'?>
<users>
    <user active="1">
        <first_name>Manuel</first_name>
        <last_name>Neuer</last_name>
    </user>
    <user active="1">
        <first_name>Joe</first_name>
        <last_name>Hart</last_name>
    </user>
    <user active="0">
        <first_name>Oliver</first_name>
        <last_name>Kahn</last_name>
    </user>
</users>
*/



CREATE PYTHON SCALAR SCRIPT process_users(url VARCHAR(500))
EMITS (firstname VARCHAR(20), lastname VARCHAR(20)) AS
import re
import xml.etree.cElementTree as etree
from urllib2 import urlopen

def run(ctx):
    content = etree.parse(urlopen(ctx.url))
    for user in content.findall('user[@active="1"]'):
        ctx.emit(user.find('first_name').text, user.find('last_name').text)
/


SELECT process_users ('http://www.my_valid_webserver/goalies.xml')
FROM DUAL;

FIRSTNAME           LASTNAME
-----  -----
Manuel               Neuer
Joe                  Hart
```

R

Neben den folgenden Infos finden Sie weitere Details zu R in der offiziellen Dokumentation (siehe <http://www.r-project.org>).

run() und cleanup() Methoden	<p>Die Methode <code>run()</code> wird für jeden Datensatz (SCALAR) bzw. jede Gruppe (SET) aufgerufen. Deren Überabeparameter ist eine Art Ausführungskontext und ermöglicht den Zugriff auf die Daten sowie den Iterator bei SET-Funktionen.</p> <p>Zum Initialisieren aufwändiger Dinge (z.B. externe Verbindungen öffnen) können Sie Code außerhalb der <code>run()</code>-Methode schreiben. Dieser wird je virtueller Maschine einmal am Anfang der SELECT-Ausführung aufgerufen.</p>
---------------------------------------	--

	Zum Deinitialisieren existiert die Methode <code>cleanup()</code> , welche ganz zum Schluss der Ausführung einmal pro virtueller Maschine aufgerufen wird.																																				
Parameter	<p>Beachten Sie, dass die R-Datentypen und die Datenbank-SQL-Typen nicht identisch sind. Daher werden bei den Ein- und Ausgabedaten folgende Konvertierungen durchgeführt:</p> <table> <tbody> <tr> <td><code>DECIMAL(p, 0)</code> für $p \leq 9$</td> <td><code>integer</code></td> </tr> <tr> <td><code>DECIMAL(p, s)</code> bzw. <code>DOUBLE</code></td> <td><code>double</code></td> </tr> <tr> <td><code>DATE</code> bzw. <code>TIMESTAMP</code></td> <td><code>POSIXt</code></td> </tr> <tr> <td><code>BOOLEAN</code></td> <td><code>logical</code></td> </tr> <tr> <td><code>VARCHAR</code> bzw. <code>CHAR</code></td> <td><code>character</code></td> </tr> <tr> <td>Sonstige</td> <td>Nicht unterstützt</td> </tr> </tbody> </table> <p> Aus Performance-Gründen sollten Sie den DOUBLE-Datentyp einem <code>DECIMAL(p,s)</code>-Typ bevorzugen.</p> <p> NULL-Werte werden durch NA repräsentiert.</p> <p>Die Eingabedaten können über den Namen adressiert werden, also z.B. <code>ctx\$my_input</code>.</p> <p>Falls Sie eine dynamische Anzahl von Parametern verwenden wollen, können Sie dies über die Notation <code>(...)</code> erreichen, also z.B. <code>CREATE R SCALAR SCRIPT my_script (...)</code>. Sie können die einzelnen Parameter dann einfach über einen Index zugreifen. Bitte beachten Sie, dass in diesem Fall eine spezielle Notation nötig ist, z.B. <code>ctx[[1]]()</code> für den ersten Parameter. Die Anzahl der Parameter sowie ihre Datentypen (werden während dem Aufruf bestimmt) sind Bestandteil der Metadaten.</p>	<code>DECIMAL(p, 0)</code> für $p \leq 9$	<code>integer</code>	<code>DECIMAL(p, s)</code> bzw. <code>DOUBLE</code>	<code>double</code>	<code>DATE</code> bzw. <code>TIMESTAMP</code>	<code>POSIXt</code>	<code>BOOLEAN</code>	<code>logical</code>	<code>VARCHAR</code> bzw. <code>CHAR</code>	<code>character</code>	Sonstige	Nicht unterstützt																								
<code>DECIMAL(p, 0)</code> für $p \leq 9$	<code>integer</code>																																				
<code>DECIMAL(p, s)</code> bzw. <code>DOUBLE</code>	<code>double</code>																																				
<code>DATE</code> bzw. <code>TIMESTAMP</code>	<code>POSIXt</code>																																				
<code>BOOLEAN</code>	<code>logical</code>																																				
<code>VARCHAR</code> bzw. <code>CHAR</code>	<code>character</code>																																				
Sonstige	Nicht unterstützt																																				
Metadaten	<p>Folgende Metadaten stehen als globale Variablen zur Verfügung:</p> <table> <tbody> <tr> <td><code>exa\$meta\$database_name</code></td> <td>Datenbankname</td> </tr> <tr> <td><code>exa\$meta\$database_version</code></td> <td>Datenbankversion</td> </tr> <tr> <td><code>exa\$meta\$script_language</code></td> <td>Name und Version der Sprache</td> </tr> <tr> <td><code>exa\$meta\$script_name</code></td> <td>Name des Skripts</td> </tr> <tr> <td><code>exa\$meta\$script_schema</code></td> <td>Schema, in dem das Skript liegt</td> </tr> <tr> <td><code>exa\$meta\$current_schema</code></td> <td>Schema, das aktuell geöffnet ist</td> </tr> <tr> <td><code>exa\$meta\$script_code</code></td> <td>Text des Skripts</td> </tr> <tr> <td><code>exa\$meta\$session_id</code></td> <td>Session-Id</td> </tr> <tr> <td><code>exa\$meta\$statement_id</code></td> <td>Statement-Id innerhalb der Session</td> </tr> <tr> <td><code>exa\$meta\$current_user</code></td> <td>Current user</td> </tr> <tr> <td><code>exa\$meta\$node_count</code></td> <td>Anzahl an Knoten im Cluster</td> </tr> <tr> <td><code>exa\$meta\$node_id</code></td> <td>Lokale Knoten-Id beginnend bei 0</td> </tr> <tr> <td><code>exa\$meta\$vm_id</code></td> <td>Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)</td> </tr> <tr> <td><code>exa\$meta\$input_type</code></td> <td>Typ der Eingabedaten (SCALAR oder SET)</td> </tr> <tr> <td><code>exa\$meta\$input_column_count</code></td> <td>Anzahl der Eingabe-Spalten</td> </tr> <tr> <td><code>exa\$meta\$input_columns[]</code></td> <td>Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}</td> </tr> <tr> <td><code>exa\$meta\$output_type</code></td> <td>Typ der Ausgabedaten (RETURNS oder EMITS)</td> </tr> <tr> <td><code>exa\$meta\$output_column_count</code></td> <td>Anzahl der Ausgabe-Spalten</td> </tr> </tbody> </table>	<code>exa\$meta\$database_name</code>	Datenbankname	<code>exa\$meta\$database_version</code>	Datenbankversion	<code>exa\$meta\$script_language</code>	Name und Version der Sprache	<code>exa\$meta\$script_name</code>	Name des Skripts	<code>exa\$meta\$script_schema</code>	Schema, in dem das Skript liegt	<code>exa\$meta\$current_schema</code>	Schema, das aktuell geöffnet ist	<code>exa\$meta\$script_code</code>	Text des Skripts	<code>exa\$meta\$session_id</code>	Session-Id	<code>exa\$meta\$statement_id</code>	Statement-Id innerhalb der Session	<code>exa\$meta\$current_user</code>	Current user	<code>exa\$meta\$node_count</code>	Anzahl an Knoten im Cluster	<code>exa\$meta\$node_id</code>	Lokale Knoten-Id beginnend bei 0	<code>exa\$meta\$vm_id</code>	Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)	<code>exa\$meta\$input_type</code>	Typ der Eingabedaten (SCALAR oder SET)	<code>exa\$meta\$input_column_count</code>	Anzahl der Eingabe-Spalten	<code>exa\$meta\$input_columns[]</code>	Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}	<code>exa\$meta\$output_type</code>	Typ der Ausgabedaten (RETURNS oder EMITS)	<code>exa\$meta\$output_column_count</code>	Anzahl der Ausgabe-Spalten
<code>exa\$meta\$database_name</code>	Datenbankname																																				
<code>exa\$meta\$database_version</code>	Datenbankversion																																				
<code>exa\$meta\$script_language</code>	Name und Version der Sprache																																				
<code>exa\$meta\$script_name</code>	Name des Skripts																																				
<code>exa\$meta\$script_schema</code>	Schema, in dem das Skript liegt																																				
<code>exa\$meta\$current_schema</code>	Schema, das aktuell geöffnet ist																																				
<code>exa\$meta\$script_code</code>	Text des Skripts																																				
<code>exa\$meta\$session_id</code>	Session-Id																																				
<code>exa\$meta\$statement_id</code>	Statement-Id innerhalb der Session																																				
<code>exa\$meta\$current_user</code>	Current user																																				
<code>exa\$meta\$node_count</code>	Anzahl an Knoten im Cluster																																				
<code>exa\$meta\$node_id</code>	Lokale Knoten-Id beginnend bei 0																																				
<code>exa\$meta\$vm_id</code>	Eindeutige Id für die lokale virtuelle Maschine (die Id's der virtuellen Maschinen haben aber keinerlei Bezug zueinander)																																				
<code>exa\$meta\$input_type</code>	Typ der Eingabedaten (SCALAR oder SET)																																				
<code>exa\$meta\$input_column_count</code>	Anzahl der Eingabe-Spalten																																				
<code>exa\$meta\$input_columns[]</code>	Array mit folgenden Informationen: {name, type, sql_type, precision, scale, length}																																				
<code>exa\$meta\$output_type</code>	Typ der Ausgabedaten (RETURNS oder EMITS)																																				
<code>exa\$meta\$output_column_count</code>	Anzahl der Ausgabe-Spalten																																				

	<code>exa\$meta\$output_columns[]</code>	Array mit folgenden Informationen: { name, type, sql_type, precision, scale, length}
Daten-Iterator, <code>next_row()</code> , <code>size()</code> und <code>reset()</code>	<p>Für Skripte mit mehreren Eingabewerten pro Aufruf (Schlüsselwort SET) gibt es zwei Zugriffsmethoden auf die Daten der Gruppe:</p> <ol style="list-style-type: none"> 1. Zeilenweises Iterieren über die Daten 2. Laden von allen (oder mehreren) Zeilen einer Gruppe in einen Vektor im Hauptspeicher, um R Operationen darauf auszuführen <p>Mittels der <code>next_row()</code>-Methode durch die Daten iteriert, die über den Kontext zugreifbar ist. Der Name unterscheidet sich zu den anderen Sprachen, da <code>next</code> ein reserviertes Schlüsselwort in R ist.</p> <p>Initial zeigt der Iterator auf die erste Eingabezeile. Daher bietet sich eine <i>repeat...until</i> Schleife an (siehe Beispiele). Falls die Eingabemenge leer ist, wird die <code>run()</code>-Methode gar nicht aufgerufen. Als Ergebnis wird dann analog zu Aggregatfunktionen der Wert NULL zurückgeliefert (wie z.B. in <code>SELECT MAX(x) FROM t WHERE false</code>).</p> <p>Um die Daten als Vektor zu verarbeiten, können Sie die Methode <code>next_row</code> mit einem Parameter aufrufen, der die Anzahl an Elementen definiert, die in den Vektor geladen werden sollen (z.B. <code>next_row(1000)</code> oder <code>next_row(NA)</code>). Geben Sie für den Parameter NA an, so wird die gesamte Gruppe auf einmal übergeben.</p> <p>Bitte beachten Sie, dass der Vektor komplett im Speicher gehalten wird. Bei sehr großen Gruppen könnte dies die Speicher-Grenzen überschreiten, so dass Sie z.B. jeweils nur 1000 Zeilen verarbeiten sollten.</p> <p>Den aktuellen Vektor mit Daten können Sie über Kontext zugreifen, mittels des Input Parameter-Namens (z.B. <code>ctx\$input_word</code>, siehe auch Beispiel unten). Beachten Sie bitte, dass allerdings in diesem Fall die Funktion <code>next_row</code> zunächst aufgerufen haben müssen, bevor Sie auf die Daten zugreifen.</p> <p>Zusätzlich gibt es eine Methode <code>reset()</code>, mit der der Iterator auf den ersten Wert zurückgesetzt werden kann. Damit sind auch mehrere Durchläufe durch die Daten möglich, falls das Ihr Algorithmus benötigt.</p> <p>Die Anzahl der Eingabedaten lässt sich über die Methode <code>size()</code> abfragen.</p>	
<code>emit()</code>	<p>Mehrere Ergebnissezeilen (Schlüsselwort EMITS) können über die Methode <code>emit()</code> erzeugt werden.</p> <p>Die Funktion erwartet so viele Parameter wie Ausgabe-Spalten. Im Falle von dynamischen Ausgabe-Parametern bietet sich in R die Nutzung einer Liste und der Methode <code>do.call</code> an wie im folgenden Beispiel:</p> <pre>do.call(ctx\$emit, list(1, "a"))</pre> <p>Übrigens können Vektoren auch für Skripte vom Typ RETURNS benutzt werden, um die Performance zu optimieren (siehe Beispiele weiter unten).</p>	
Import anderer Skripte	Zum Import anderer Skripte steht die Methode <code>exa\$import_script()</code> zur Verfügung. Der Rückgabewert dieser Methode muss einer Variablen zugewiesen werden, die anschließend das importierte Modul darstellt.	
Zugriff auf Verbindungsdefinitionen	Die Details von Verbindungen, die mit CREATE CONNECTION definiert wurden können innerhalb von R UDF Skripten mit Hilfe der Methode	

	<p><code>exa\$get_connection("<connection_name>")</code> abgefragt werden. Das Ergebnis ist eine R Liste mit den folgenden Einträgen:</p> <table> <tbody> <tr> <td>type</td><td>Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.</td></tr> <tr> <td>address</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.</td></tr> <tr> <td>user</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.</td></tr> <tr> <td>password</td><td>Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.</td></tr> </tbody> </table>	type	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.	address	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.	user	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.	password	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.																						
type	Der Typ der Verbindungsdefinition. Aktuell wird nur der Typ "PASSWORD" verwendet.																														
address	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort TO markiert wurde.																														
user	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit dem Schlüsselwort USER markiert wurde.																														
password	Der Teil der Verbindungsdefinition, der bei CREATE CONNECTION mit den Schlüsselwörtern IDENTIFIED BY markiert wurde.																														
Hilfsbibliotheken	<p>Folgende zusätzliche Bibliotheken stehen zur Verfügung, die nicht bereits im Sprachumfang enthalten sind:</p> <table> <tbody> <tr> <td>bitops</td><td>Bitweise Operationen für Integer-Vektoren (siehe auch http://cran.r-project.org/web/packages/bitops)</td></tr> <tr> <td>class</td><td>Diverse Funktionen zur Klassifikation (siehe auch http://cran.r-project.org/web/packages/class)</td></tr> <tr> <td>data.table</td><td>Erweiterung von <code>data.frame</code> zum Beschleunigen von Indizierung, sortierten Joins, Zuweisungen und Gruppierungen (siehe auch http://cran.r-project.org/web/packages/data.table)</td></tr> <tr> <td>e1071</td><td>Diverse Funktionen des Statistik-Lehrstuhls (e1071) der TU Wien (siehe auch http://cran.r-project.org/web/packages/e1071)</td></tr> <tr> <td>flashClust</td><td>Implementierung von hierarchischem Clustering (siehe auch http://cran.r-project.org/web/packages/flashClust)</td></tr> <tr> <td>gbm</td><td>Generalized Boosted Regression Models (siehe auch http://cran.r-project.org/web/packages/gbm)</td></tr> <tr> <td>Hmisc</td><td>Harrell Miscellaneous (siehe auch http://cran.r-project.org/web/packages/Hmisc)</td></tr> <tr> <td>MASS</td><td>Funktionen und Datenstrukturen, um MASS von Venables und Ripley zu unterstützen (siehe auch http://cran.r-project.org/web/packages/MASS)</td></tr> <tr> <td>randomForest</td><td>Klassifikations- und Regressions-Algorithmen (siehe auch http://cran.r-project.org/web/packages/randomForest)</td></tr> <tr> <td>RCurl</td><td>Internet Support wie z.B. http Verbindungen (siehe auch http://www.omegahat.org/RCurl)</td></tr> <tr> <td>RODBC</td><td>Schnittstelle zu ODBC Datenbanken (siehe auch http://cran.r-project.org/web/packages/RODBC)</td></tr> <tr> <td>rpart</td><td>Rekursive Partitionierung und Regressions-Bäume (siehe auch http://cran.r-project.org/web/packages/rpart)</td></tr> <tr> <td>rredis</td><td>Schnittstelle zu Redis (siehe auch http://cran.r-project.org/web/packages/rredis)</td></tr> <tr> <td>RSXML</td><td>XML Verarbeitung (siehe auch http://www.omegahat.org/RSXML)</td></tr> <tr> <td>survival</td><td>Lebensdauer-Analyse (siehe auch http://cran.r-project.org/web/packages/survival)</td></tr> </tbody> </table>	bitops	Bitweise Operationen für Integer-Vektoren (siehe auch http://cran.r-project.org/web/packages/bitops)	class	Diverse Funktionen zur Klassifikation (siehe auch http://cran.r-project.org/web/packages/class)	data.table	Erweiterung von <code>data.frame</code> zum Beschleunigen von Indizierung, sortierten Joins, Zuweisungen und Gruppierungen (siehe auch http://cran.r-project.org/web/packages/data.table)	e1071	Diverse Funktionen des Statistik-Lehrstuhls (e1071) der TU Wien (siehe auch http://cran.r-project.org/web/packages/e1071)	flashClust	Implementierung von hierarchischem Clustering (siehe auch http://cran.r-project.org/web/packages/flashClust)	gbm	Generalized Boosted Regression Models (siehe auch http://cran.r-project.org/web/packages/gbm)	Hmisc	Harrell Miscellaneous (siehe auch http://cran.r-project.org/web/packages/Hmisc)	MASS	Funktionen und Datenstrukturen, um MASS von Venables und Ripley zu unterstützen (siehe auch http://cran.r-project.org/web/packages/MASS)	randomForest	Klassifikations- und Regressions-Algorithmen (siehe auch http://cran.r-project.org/web/packages/randomForest)	RCurl	Internet Support wie z.B. http Verbindungen (siehe auch http://www.omegahat.org/RCurl)	RODBC	Schnittstelle zu ODBC Datenbanken (siehe auch http://cran.r-project.org/web/packages/RODBC)	rpart	Rekursive Partitionierung und Regressions-Bäume (siehe auch http://cran.r-project.org/web/packages/rpart)	rredis	Schnittstelle zu Redis (siehe auch http://cran.r-project.org/web/packages/rredis)	RSXML	XML Verarbeitung (siehe auch http://www.omegahat.org/RSXML)	survival	Lebensdauer-Analyse (siehe auch http://cran.r-project.org/web/packages/survival)
bitops	Bitweise Operationen für Integer-Vektoren (siehe auch http://cran.r-project.org/web/packages/bitops)																														
class	Diverse Funktionen zur Klassifikation (siehe auch http://cran.r-project.org/web/packages/class)																														
data.table	Erweiterung von <code>data.frame</code> zum Beschleunigen von Indizierung, sortierten Joins, Zuweisungen und Gruppierungen (siehe auch http://cran.r-project.org/web/packages/data.table)																														
e1071	Diverse Funktionen des Statistik-Lehrstuhls (e1071) der TU Wien (siehe auch http://cran.r-project.org/web/packages/e1071)																														
flashClust	Implementierung von hierarchischem Clustering (siehe auch http://cran.r-project.org/web/packages/flashClust)																														
gbm	Generalized Boosted Regression Models (siehe auch http://cran.r-project.org/web/packages/gbm)																														
Hmisc	Harrell Miscellaneous (siehe auch http://cran.r-project.org/web/packages/Hmisc)																														
MASS	Funktionen und Datenstrukturen, um MASS von Venables und Ripley zu unterstützen (siehe auch http://cran.r-project.org/web/packages/MASS)																														
randomForest	Klassifikations- und Regressions-Algorithmen (siehe auch http://cran.r-project.org/web/packages/randomForest)																														
RCurl	Internet Support wie z.B. http Verbindungen (siehe auch http://www.omegahat.org/RCurl)																														
RODBC	Schnittstelle zu ODBC Datenbanken (siehe auch http://cran.r-project.org/web/packages/RODBC)																														
rpart	Rekursive Partitionierung und Regressions-Bäume (siehe auch http://cran.r-project.org/web/packages/rpart)																														
rredis	Schnittstelle zu Redis (siehe auch http://cran.r-project.org/web/packages/rredis)																														
RSXML	XML Verarbeitung (siehe auch http://www.omegahat.org/RSXML)																														
survival	Lebensdauer-Analyse (siehe auch http://cran.r-project.org/web/packages/survival)																														

Funktion <code>defaultOutputColumns()</code> für dynamische Ausgabe-Parameter	<p>Falls ein UDF Skript mit dynamischen Ausgabe-Parametern definiert wurde und die Ausgabe-Parameter nicht bestimmt werden können (über EMITS in der Abfrage oder über INSERT INTO), ruft die Datenbank die Funktion <code>defaultOutputColumns()</code> auf, die Sie implementieren können. Der erwartete Rückgabewert ist ein String mit den Namen und Typen der Ausgabe-Parameter, z.B. "a int, b varchar(100)". Siehe Dynamische Eingabe- und Ausgabe-Parameter für eine genaue Beschreibung wann diese Methode aufgerufen wird und für Beispiele.</p> <ul style="list-style-type: none">  Sie können die Metadaten über <code>exa\$meta</code> abrufen, z.B. um die Anzahl und die Typen der Eingabe-Parameter zu abzufragen.  Diese Methode wird genau einmal auf einem einzigen Knoten ausgeführt. 										
Funktion <code>generate_sql_for_import_spec()</code> für benutzerdefinierten Import (siehe Abschnitt 3.4.4, Benutzerdefinierter IMPORT mittels UDFs)	<p>Damit ein UDF Skript in einem benutzerdefinierten Import (IMPORT FROM SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>generate_sql_for_import_spec(import_spec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>import_spec</code> enthält alle Informationen über den ausgeführten IMPORT FROM SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die zu importierenden Daten liefert.</p> <p><code>import_spec</code> hat die folgenden Felder:</p> <table> <tr> <td>parameters</td> <td>Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.</td> </tr> <tr> <td>is_subselect</td> <td>Genau dann wahr, wenn der IMPORT Befehl in einem SELECT benutzt wird, und nicht in einem IMPORT INTO table Befehl.</td> </tr> <tr> <td>subselect_column_names[]</td> <td>Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Namen der spezifizierten Ausgabespalten zurückgegeben.</td> </tr> <tr> <td>subselect_column_types[]</td> <td>Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Typen der spezifizierten Ausgabespalten zurückgegeben. Die Typen sind im SQL Format angegeben (z.B. "VARCHAR(100)").</td> </tr> <tr> <td>connection_name</td> <td>Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über</td> </tr> </table>	parameters	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.	is_subselect	Genau dann wahr, wenn der IMPORT Befehl in einem SELECT benutzt wird, und nicht in einem IMPORT INTO table Befehl.	subselect_column_names[]	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Namen der spezifizierten Ausgabespalten zurückgegeben.	subselect_column_types[]	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Typen der spezifizierten Ausgabespalten zurückgegeben. Die Typen sind im SQL Format angegeben (z.B. "VARCHAR(100)").	connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über
parameters	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.										
is_subselect	Genau dann wahr, wenn der IMPORT Befehl in einem SELECT benutzt wird, und nicht in einem IMPORT INTO table Befehl.										
subselect_column_names[]	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Namen der spezifizierten Ausgabespalten zurückgegeben.										
subselect_column_types[]	Falls <code>is_subselect</code> wahr ist und der Nutzer die Ausgabespalten angegeben hat (z.B. <code>SELECT * FROM (IMPORT INTO (c1 int, c2 varchar(100)) FROM SCRIPT ...)</code>) wird eine Liste von Strings mit den Typen der spezifizierten Ausgabespalten zurückgegeben. Die Typen sind im SQL Format angegeben (z.B. "VARCHAR(100)").										
connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die Verbindungsinformationen dann über										

	connection	<p><code>exa\$get_connection(name)</code> abfragen. Der Wert <code>NULL</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</p> <p>Wenn der Nutzer direkt Verbindungsinformationen angegeben hat wird ein Objekt zurückgegeben das die gleiche Struktur hat wie der Rückgabewert von <code>exa\$get_connection(name)</code>. Der Wert <code>NULL</code> wird zurückgegeben wenn keine Verbindung angegeben wurde.</p> <p> Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das <code>connection_name</code> Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über <code>exa\$get_connection(name)</code> abgefragt werden.</p>												
Funktion <code>genera - te_sql_for_export_spec()</code> für benutzerdefinierten Export (siehe Abschnitt 3.4.5, Benutzerdefinierter EXPORT mittels UDFs)		<p>Damit ein UDF Skript in einem benutzerdefinierten Export (EXPORT INTO SCRIPT) angegeben werden kann, müssen Sie die Funktion <code>genera - te_sql_for_export_spec(export_spec)</code> implementieren. Bitte beachten Sie auch Dynamische Eingabe- und Ausgabe-Parameter sowie den IMPORT Befehl für die genaue Syntax. Der Parameter <code>export_spec</code> enthält alle Informationen über den ausgeführten EXPORT INTO SCRIPT Befehl. Die Funktion muss einen SELECT SQL Befehl als String zurückgeben, der bei Ausführung die Daten (in der Regel mittels UDFs) exportiert. Der FROM-Teil dieses Strings kann eine Dummy-Tabelle sein (z.B. DUAL), denn der Export-Befehl weiß ja bereits, welche Tabelle exportiert werden soll. Sie muss dennoch angegeben werden, damit dieser SQL-String erfolgreich kompiliert werden kann.</p> <p><code>export_spec</code> hat die folgenden Felder:</p> <table> <tr> <td>parameters</td> <td>Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.</td> </tr> <tr> <td>source_column_names</td> <td>Liste der Spaltennamen der zu exportierenden Daten.</td> </tr> <tr> <td>has_truncate</td> <td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.</td> </tr> <tr> <td>has_replace</td> <td>Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).</td> </tr> <tr> <td>created_by</td> <td>String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.</td> </tr> <tr> <td>connection_name</td> <td>Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die</td> </tr> </table>	parameters	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.	source_column_names	Liste der Spaltennamen der zu exportierenden Daten.	has_truncate	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.	has_replace	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).	created_by	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.	connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die
parameters	Die in dem IMPORT Befehl angegebenen Parameter. Z.B. liefert <code>parameters\$FOO</code> den Wert für den Parameter FOO.													
source_column_names	Liste der Spaltennamen der zu exportierenden Daten.													
has_truncate	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Inhalte der Tabelle vor dem Export gelöscht werden sollen.													
has_replace	Bool-Wert, abgeleitet aus der Option im EXPORT-Befehl, ob die Tabelle durch den Export ersetzt werden soll (also zuvor komplett gelöscht).													
created_by	String, abgeleitet aus der Option im EXPORT-Befehl, mit dem der Erzeugungstext (vor dem Transfer der Daten) für die Tabelle im Zielsystem definiert wird.													
connection_name	Der Name der Verbindung, falls eine solche angegeben wurde. Das UDF Skript kann die													

	connection	<p>Verbindungsinformationen dann über exa\$get_connection(name) abfragen. Der Wert NULL wird zurückgegeben wenn keine Verbindung angegeben wurde. Wenn der Nutzer direkt Verbindungsinformationen angegeben hat wird ein Objekt zurückgegeben das die gleiche Struktur hat wie der Rückgabewert von exa\$get_connection(name). Der Wert NULL wird zurückgegeben wenn keine Verbindung angegeben wurde.</p> <p> Falls ein Passwort in den Verbindungsinformationen angegeben wurde wird dieses Klartext übertragen und erscheint möglicherweise in den Logfiles. Wir empfehlen daher, zuvor eine Verbindung (CONNECTION) anzulegen und dann den Verbindungsnamen anzugeben (dieser kann über das connection_name Feld abgefragt werden). Die tatsächlichen Verbindungsinformationen können dann über exa\$get_connection(name) abgefragt werden.</p>
--	------------	--

Beispiel:

```
-- A simple SCALAR RETURNS function, similar to the SQL function UPPER()
CREATE R SCALAR SCRIPT r_upper(input_word VARCHAR(50))
RETURNS VARCHAR(50) AS
run <- function(ctx) {
    # the last statement in a function is automatically used as return value
    toupper(ctx$input_word)
}
/
SELECT last_name, r_upper(last_name) AS r_upper FROM customer;

LAST_NAME R_UPPER
-----
Smith      SMITH
Miller     MILLER

-- The same function as r_upper, but using a faster vector operation
CREATE R SCALAR SCRIPT r_upper_vectorized(input_word VARCHAR(50))
RETURNS VARCHAR(50) AS
run <- function(ctx) {
    # read all records in the current buffer into a single vector.
    # the buffer holds any number of records from the table
    ctx$next_row(NA) toupper(ctx$input_word)
}
/
-- A simple SET-EMITS function, computing simple statistics on a column
CREATE R SET SCRIPT r_stats(group_id INT, input_number DOUBLE)
EMITS (group_id INT, mean DOUBLE, stddev DOUBLE) AS
run <- function(ctx) {
    # fetch all records from this group into a single vector
```

```

    ctx$next_row(NA)
    ctx$emit(ctx$group_id[1], mean(ctx$input_number), sd(ctx$input_number))
}
/

SELECT r_stats(groupId, num) FROM keyvalues
GROUP BY groupId ORDER BY group_id;

GROUP_ID MEAN          STDDEV
-----
1      1.0333333333 0.152752523
2      42             12.16552506059644

/*
The following example loads from a webserver
and processes the following file goalies.xml:

<?xml version='1.0' encoding='UTF-8'?>
<users>
    <user active="1">
        <first_name>Manuel</first_name>
        <last_name>Neuer</last_name>
    </user>
    <user active="1">
        <first_name>Joe</first_name>
        <last_name>Hart</last_name>
    </user>
    <user active="0">
        <first_name>Oliver</first_name>
        <last_name>Kahn</last_name>
    </user>
</users>
*/
CREATE R SCALAR SCRIPT process_users(url VARCHAR(500))
EMITS (firstname VARCHAR(100), lastname VARCHAR(100)) AS
require('RCurl')
require('XML')

run <- function(ctx) {
    cont <- getURL(ctx$url)
    tree <- xmlTreeParse(cont)
    for (i in 1:length(tree$doc$children$users)) {
        if (tree$doc$children$users[i]$user$attributes['active']==1) {
            firstname <- tree$doc$children$users[i]$user$children$first_name
            lastname  <- tree$doc$children$users[i]$user$children$last_name
            ctx$emit(firstname$children$text$value,
                    lastname$children$text$value)
        }
    }
}
/

SELECT process_users ('http://www.my_valid_webserver/goalies.xml');

FIRSTNAME          LASTNAME
-----
```

Manuel	Neuer
Joe	Hart

3.6.4. Das synchrone Cluster-Dateisystem BucketFS

Wenn Skripte parallel auf dem Exasol-Cluster ausgeführt werden, so gibt es immer wieder den Fall, dass alle Instanzen auf identische externe Daten zugreifen müssen. Beispielsweise könnte ein statistisches Modell benutzt oder Wetterdaten in den Algorithmen verwendet werden. Solche Anforderungen könnte man natürlich leicht mit einem externen Dienst (z.B. einem File-Server) lösen. Doch es wäre vor allem aus Performance-Sicht äußerst praktisch, wenn diese Daten lokal auf den Cluster-Knoten verfügbar wären.

Das Exasol BucketFS-Dateisystem wurde genau für solche Anwendungsszenarien konzipiert, in denen Daten im Cluster synchron und repliziert gespeichert werden sollen. Sie werden im nächsten Kapitel allerdings noch lernen, wie dieses Konzept auch benutzt werden kann, um den Sprachumfang der Skript-Sprachen selbst zu erweitern und sogar komplett neue Skript-Sprachen auf dem Exasol-Cluster zu installieren.

Wenn Sie zusätzlich zu den nachfolgenden Informationen interessiert sind an einem echten Data Science Anwendungsbeispiel, so empfehlen wir Ihnen folgenden Beitrag in unserem Solution Center: <https://wwwexasol.com/support/browse/SOL-257>

Was ist BucketFS?

Das BucketFS-Dateisystem ist ein synchrones Datei-System, welches im Exasol Cluster zur Verfügung steht. Dies bedeutet, dass Sie sich grundsätzlich auf jedem Knoten des Clusters mit diesem Service verbinden können (z.B. über die http Schnittstelle) und jeweils den identischen Inhalt sehen werden.



Die Daten werden lokal auf jedem Rechner repliziert vorgehalten und automatisch synchronisiert. Daher sollten Sie dort aus Platzgründen keine großen Datenmengen abspeichern.



BucketFS-Daten werden nicht über das Datenbank-Backup gesichert und müssen daher manuell gesichert werden, sofern dies nötig ist.

Ein BucketFS-Dienst enthält eine beliebige Anzahl von so genannten Buckets, die wiederum eine beliebige Zahl von Dateien enthalten können. Jedes Bucket kann unterschiedliche Zugriffsrechte haben, wie weiter unten beschrieben wird. Ordner-Strukturen werden darin nicht direkt unterstützt, aber wenn Sie im Zielpfad einen noch nicht existierenden Ordner angeben, so wird dieser implizit angelegt. Sobald alle Dateien aus einem Ordner gelöscht wurden, wird der Ordner automatisch gelöscht.

Sofern Dateien geschrieben werden, so wird dies atomar umgesetzt. Allerdings existieren keine Locks auf die Dateien, so dass letztlich der letzte Schreibzugriff alle vorherigen Zugriffe überschreibt. Im Gegensatz zur Datenbank hat das BucketFS auch keine transaktionale Semantik.

Einrichten von BucketFS und Buckets

Sie finden die Administrations-Seite für BucketFS auf der obersten Ebene von EXAoperation. Dort befindet sich für die konfigurierte data-Disk bereits immer ein vorinstalliertes Default-BucketFS. Sofern Sie neue Dateisysteme erzeugen wollen, müssen Sie als Parameter lediglich die data-Disk sowie entsprechende Ports für http(s) spezifizieren.

Wenn Sie dem Link einer BucketFS-Id folgen, können Sie beliebig viele Buckets innerhalb dieses BucketFS erzeugen und verwalten. Neben dem Bucket-Namen müssen Sie immer auch entsprechende Lese-/Schreib-Passwörter angeben und definieren, ob das Bucket öffentlich zugreifbar sein soll. In diesem Fall können Sie Dateien auch ohne Passwort lesen. Schreibend wird jedoch immer das Write-Passwort benötigt.



Im Default-BucketFS befindet sich bereits ein Default-Bucket für die vorinstallierten Skript-Sprachen (Java, Python, R). Wir empfehlen allerdings, zum Speichern von größeren Nutzerdaten eine eigene BucketFS-Instanz in einer separaten Partition.

Zugriff und Zugriffsschutz

Außerhalb des Clusters ist es möglich, mit http(s)-Clients auf die Buckets und darin liegenden Dateien zuzugreifen (z.B. curl™). Sie müssen hierzu lediglich die IP-Adresse eines Datenbank-Servers, den spezifizierten Port sowie den Bucket-Namen verwenden, sowie ggfs. Ihre interne Firewall konfigurieren.



Für den Zugriff auf ein Bucket über http(s) existieren immer nur die Nutzer `r` und `w`, welche mit den angegebenen Read- und Write-Passwörtern verknüpft sind.

Im folgenden Beispiel wird mittels curl erst die Liste der existierenden Buckets aufgelistet, dann eine Datei `file1` in das Bucket `bucket1` hochgeladen und anschließend die Liste der enthaltenen Dateien in diesem Bucket ausgegeben. Konfiguriert wurde hierfür der Port des BucketFS (1234) und das Bucket selbst mit Namen (`bucket1`) und Passwörtern (`readpw` bzw. `writepw`).

```
$> curl http://192.168.6.75:1234
default
bucket1
$> curl -X PUT -T file1 http://w:writepw@192.168.6.75:1234/bucket1/file1
$> curl -X PUT -T tar1.tgz \
http://w:writepw@192.168.6.75:1234/bucket1/tar1.tgz
$> curl http://r:readpw@192.168.6.75:1234/bucket1
file1
tar1.tgz
```

Innerhalb von Skripten, die im Exasol Cluster ausgeführt werden, kann zur Vereinfachung auf die Dateien über einen lokalen Datei-Pfad lesend zugegriffen werden. Hierdurch müssen Sie keine IP-Adressen angeben und können sicher sein, dass auf die Daten des lokalen Knotens benutzt werden. Den entsprechenden Pfad für ein Bucket finden Sie in EXAoperation in der jeweiligen Übersicht der Buckets eines BucketFS.

Der Zugriffsschutz erfolgt in diesem Fall über ein Datenbank-Objekt CONNECTION (siehe [CREATE CONNECTION](#)), da Buckets sehr ähnlich sind zu normalen externen Datenquellen. Die Verbindung enthält den Bucket-Pfad und kapselt das Lese-Passwort. Sobald einem Nutzer diese Verbindung mittels [GRANT](#) zugewiesen wird, wird das Bucket für den Nutzer sichtbar/lesbar.

Wollen Sie ein Bucket allen Nutzern zur Verfügung stellen, dann können Sie es in EXAoperation auch als *public* definieren.



Ein Schreibzugriff aus Skripten heraus ist wie bei externen Clients nur mittels http(s) möglich, aber Sie müssen in solchen Fällen auch immer auf die Parallelität der Skript-Prozesse achten.

Im folgenden Beispiel wird zunächst solch eine Connection definiert und anschließend ein Skript erzeugt, welches die Dateien aus einem lokalen Pfad ausgibt. Der äquivalente lokale Pfad für das oben angelegte Bucket `bucket1` lautet `/buckets/bfsdefault/bucket1`, wie Sie im Beispiel sehen können.

```
CREATE CONNECTION my_bucket_access TO 'bucketfs:bfsdefault/bucket1'
IDENTIFIED BY 'readpw';

GRANT CONNECTION my_bucket_access TO my_user;

CREATE PYTHON SCALAR SCRIPT ls(my_path VARCHAR(100))
EMITS (files VARCHAR(100)) AS
import subprocess
```

```

def run(c):
    try:
        p = subprocess.Popen('ls '+c.my_path,
                            stdout = subprocess.PIPE,
                            stderr = subprocess.STDOUT,
                            close_fds = True,
                            shell = True)
        out, err = p.communicate()
        for line in out.strip().split('\n'):
            c.emit(line)
    finally:
        if p is not None:
            try: p.kill()
            except: pass
    /
SELECT ls('/buckets/bfsdefault/bucket1');

FILES
-----
file1
tar1

SELECT ls('/buckets/bfsdefault/bucket1/tar1/');

FILES
-----
a
b

```

Wie Sie vielleicht im Beispiel entdeckt haben, werden Archive (.zip, .tar, .tar.gz oder .tgz) für den Zugriff aus Skripten heraus auf das lokale Dateisystem immer ausgepackt. Von außen (z.B. mittels curl) sehen Sie daher das Archiv, während Sie innerhalb des Skripts lokal auf die ausgepackten Dateien und Ordner zugreifen können.



Speichern Sie Archive (.zip, .tar, .tar.gz oder .tgz) in BucketFS, werden diese sowohl im Originalzustand als auch ausgepackt gespeichert und verbrauchen daher auch doppelt Speicherplatz.



Falls Sie direkt mit einem Archive arbeiten und das Auspacken vermeiden wollen, können Sie einfach die Datei-Erweiterung umbenennen (z.B. .zipx anstatt .zip).

3.6.5. Skript-Sprachen erweitern mittels BucketFS

Sofern Ihnen der ausgelieferte Sprachumfang von Exasol ausreicht, dann müssen Sie dieses Kapitel nicht weiter beachten. Wollen Sie jedoch die UDF-Sprachen erweitern (z.B. weitere Paket für R installieren) oder sogar ganz neue Programmiersprachen in das UDF-Framework integrieren, so können Sie dies sehr einfach mittels BucketFS umsetzen.

Existierende Skript-Sprachen erweitern

Die erste Möglichkeit ist es, die existierenden Sprachen zu erweitern, also die darin enthaltenen Pakete zu erweitern. Das entsprechende Vorgehen unterscheidet sich von Sprache zu Sprache und wird daher in den folgenden Unterkapiteln entsprechend vorgestellt.



Die Skript-Sprache Lua ist nicht erweiterbar, weil sie nativ in die Exasol Datenbank-Software kompiliert ist.

Java jar-Dateien

In Java können Sie die gewünschten .jar Dateien sehr einfach in Exasol integrieren. Hierfür speichern Sie die entsprechende Datei in ein Bucket und referenzieren diesen Pfad direkt in Ihrem Java-Skript.

Wenn Sie beispielsweise Google's Bibliothek zur Verarbeitung von Telefonnummern (<http://maven-search.io/repo/com.googlecode.libphonenumber/libphonenumber/4.2>) nutzen wollen, können Sie diese Datei analog zu den Beispielen oben in einem Bucket namens javalib ablegen, so dass der entsprechende Bucket-Pfad folgendermaßen aussieht: /buckets/bfsdefault/javalib/libphonenumber-4.2.jar.

Im Skript unten sehen Sie, wie dieser Pfad eingebunden wird, um die Bibliothek letztlich nutzen zu können.

```
CREATE JAVA SCALAR SCRIPT jphone(num VARCHAR(2000))
RETURNS VARCHAR(2000) AS
%jar /buckets/bfsdefault/javalib/libphonenumber-4.2.jar;

import com.google.i18n.phonenumbers.PhoneNumberUtil;
import com.google.i18n.phonenumbers.NumberParseException;
import com.google.i18n.phonenumbers.Phonenumber.PhoneNumber;

class JPHONE {
    static String run(ExaMetadata exa, ExaIterator ctx) throws Exception {
        PhoneNumberUtil phoneUtil = PhoneNumberUtil.getInstance();
        try {
            PhoneNumber swissNumberProto = phoneUtil.parse(ctx.getString("num"),
                "DE");
            return swissNumberProto.toString();
        } catch (NumberParseException e) {
            System.err.println("NumberParseException thrown: " + e.toString());
        }
        return "failed";
    }
}
```

Python Libraries

Python Bibliotheken werden oftmals in Form von .whl Dateien ausgeliefert. Sie können solche Bibliotheken einfach in den Python-Sprachumfang integrieren, indem Sie die Datei in ein Bucket hochladen und den Suchpfad von Python entsprechend erweitern.

Wenn Sie beispielsweise die Bibliothek von Google zur Verarbeitung von Telefonnummern (<https://pypi.python.org/pypi/phonenumbers>) benutzen wollen, können Sie diese Datei analog zu den Beispielen oben in einem Bucket namens pylib ablegen, so dass der entsprechende Bucket-Pfad folgendermaßen aussieht: /buckets/bfsdefault/pylib/phonenumbers-7.7.5-py2.py3-none-any.whl.

Im Skript unten sehen Sie, wie dieser Pfad eingebunden wird, um die Bibliothek letztlich nutzen zu können.

```
CREATE PYTHON SCALAR SCRIPT phonenumbers(phone_number VARCHAR(2000))
RETURNS VARCHAR(2000) AS
import sys
import glob

sys.path.extend(glob.glob('/buckets/bfsdefault/pylib/*'))
```

```

import phonenumbers

def run(c):
    return str(phonenumbers.parse(c.phone_number, None))
/

SELECT phononenumber('+1-555-2368') AS ghost_busters;

GHOST_BUSTERS
-----
Country Code: 1 National Number: 55552368

```

R Pakete

Im Falle von R-Paketen ist die Installation mitunter etwas komplizierter, da diese oft mit Hilfe des C-Compilers kompiliert werden müssen. Hierfür laden Sie sich den existierenden Exasol Linux-Container herunter, kompilieren darin das gewünschte Paket und laden das so erzeugte Paket ins BucketFS hoch. Details zum Linux-Container finden Sie im nächsten Kapitel.

Eine sehr einfache Möglichkeit ist die Nutzung von Docker, wie im folgenden Beispiel beschrieben.

```

$> bucketfs = http://192.168.6.75:1234
$> cont = /default/EXAClusterOS/ScriptLanguages-2016-10-21.tar.gz
$> docker $bucketfs$cont import my_dock
$> mkdir r_pkg
$> docker run -v `pwd`/r_pkg:/r_pkg --name=my_dock -it my_dock /bin/bash

```

Auch hier wollen wir wieder eine Bibliothek zur Verarbeitung von Telefonnummern nutzen (<https://cran.r-project.org/web/packages/phonenumbers/index.html>), diesmal von Steve Myles. Innerhalb des Docker-Containers kann R gestartet und das entsprechende Paket installiert werden:

```

# export R_LIBS="/r_pkg/"
# R
> install.packages('phonenumbers', repos="http://cran.r-project.org")
Installing package into '/r_pkg'
(as 'lib' is unspecified)
trying URL 'http://cran.r-project.org/src/contrib/phonenumbers_0.2.2.tar.gz'
Content type 'application/x-gzip' length 10516 bytes (10 KB)
=====
downloaded 10 KB

```

Durch die erste Zeile wird das Paket aus dem Linux-Container in das Unterverzeichnis `r_pkg` installiert, welches auch außerhalb des Docker-Containers verfügbar ist.

Anschließend wird das erzeugte tgz-Archiv in dem Bucket namens `rlib` gespeichert:

```

$> bucketfs = http://w:writepw@192.168.6.75:1234
$> curl -vX PUT -T r_pkg.tgz $bucketfs/rlib/r_pkg.tgz

```

Im Skript unten sehen Sie, wie der resultierende Pfad (/buckets/bfsdefault/rlib/r_pkg) eingebunden wird, um die Bibliothek letztlich nutzen zu können.

```

CREATE R SCALAR SCRIPT tophone(letters VARCHAR(2000)) RETURNS INT AS
.libPaths( c( .libPaths(), "/buckets/bfsdefault/rlib/r_pkg" ) )
library(phonenumbers)

run <- function(ctx) {

```

```

    letterToNumber(ctx$letters, qz = 1)
}
/

```

Neue Skript-Sprachen installieren

Durch das offene Skript-Framework von Exasol ist es leicht möglich, komplett neue Skript-Sprachen in Exasol zu integrieren. Hierfür sind grundsätzlich 3 Schritte nötig:

1. Einen funktionsfähigen Sprach-Client bauen
2. Hochladen des fertigen Sprach-Clients in ein Bucket
3. Skript-Sprach-Alias definieren ([ALTER SESSION](#) bzw. [ALTER SYSTEM](#))

Wie man einen Sprach-Client erzeugt, wird im nächsten Abschnitt ausführlicher erklärt. Im Prinzip wird die jeweilige Sprache mit einfachen APIs erweitert, die die Kommunikation zwischen Exasol und der Sprache implementiert. Anschließend wird der Client für den vorinstallierten Exasol Linux Container kompiliert, so dass er in einem gesicherten Prozess im Exasol Cluster gestartet werden kann.

Das Hochladen von Dateien ins BucketFS und wie diese zur Verfügung gestellt werden können wurde bereits in den vorherigen Kapiteln ausführlich erläutert.

Durch den letzten Schritt wird die Verbindung des Sprach-Clients mit der SQL-Sprache in Exasol erzeugt, also letztlich dem Befehl [CREATE SCRIPT](#). Dies ermöglicht sehr viele Optionen, um neue Versionen einer Sprache oder ganz neue Sprachen in Ihrem System zunächst auszuprobieren und dann komplett auszutauschen.

Wenn Sie beispielsweise von Python 2 auf Python 3 umsteigen wollen, können Sie einen neuen Sprach-Client hochladen, anschließend den Alias für PYTHON temporär mittels [ALTER SESSION](#) auf die neue Version umstellen, und nach einer ausgiebigen Testphase den Alias mittels [ALTER SYSTEM](#) für alle Nutzer umstellen.

Andererseits ist es auch möglich, die beiden Sprachen parallel zu nutzen, indem für Skripte zwei Aliase vergeben werden (z.B. PYTHON2 und PYTHON3).

Erzeugung des Sprach-Clients

Die Hauptaufgabe bei der Installation neuer Sprachen ist die Entwicklung des entsprechenden Sprach-Clients. Interessieren Sie sich für eine bestimmte Sprache, so sollten Sie zunächst in unserem Open Source Repository prüfen, ob bereits ein entsprechender Client veröffentlicht wurde (siehe <https://github.com/exasol/script-languages>). Ansonsten würden wir uns natürlich sehr freuen, wenn Sie neu entwickelte Sprachen über diesen Wege auch mit uns und anderen Nutzern teilen.

Ein Sprach-Client basiert auf einem Linux-Container im BucketFS. Der ausgelieferte Sprach-Client für die Sprachen Python, Java und R verwendet dazu den Linux-Container, der sich im Default-Bucket des Default-BucketFS befindet. Mit der Linux-Container Technologie wird innerhalb der Abarbeitung einer SQL-Anfrage mit Skript-Nutzung eine Art gekapselte virtuelle Maschine in einer sicheren Umgebung gestartet. Der Linux-Container enthält eine vollständige Linux-Distribution und benutzt den entsprechenden Skriptsprachen-Client für die Ausführung des Skript-Codes. Grundsätzlich könnten Sie auch einen eigenen Linux-Container in BucketFS speichern und mit Ihrem Client benutzen.

Der Sprach-Client muss ein bestimmtes Protokoll implementieren, das die Kommunikation zwischen Skripten und der Datenbank steuert. Hierfür werden die etablierten Technologien ZeroMQ™ (siehe <http://zeromq.org>) und Google's Protocol Buffers™ (<https://github.com/google/protobuf>) verwendet. Das Protokoll selbst wird nicht im Handbuch beschrieben, weil dies an dieser Stelle zu umfangreich wäre. Wir verweisen daher auf unser Open Source Repository (siehe <https://github.com/exasol/script-languages>), in dem folgende Inhalte enthalten sind:

- Protokoll-Beschreibung
- Notwendige Dateien zum Bauen des Exasol Linux-Containers (Docker Konfigurations-Datei und Build-Skript)
- Beispiel-Implementierungen für weitere Sprachen (u.a. C++ sowie ein nativer Python Client, der Python 2 und Python 3 unterstützt)

Skript-Aliase zur Einbindung in SQL

Haben Sie den Sprach-Client erzeugt und hochgeladen, so muss dieser noch in Exasol über einen Alias konfiguriert werden. Der Datenbank wird hiermit mitgeteilt, welche Skript-Sprache an welcher Stelle im Cluster installiert ist.

Gesetzt werden die Skript-Aliase mit den Befehlen [ALTER SESSION](#) und [ALTER SYSTEM](#). Somit kann die Benutzung der Sprachen Session- oder System-weit konfiguriert werden, was gerade bei der Nutzung von verschiedenen Sprach-Versionen oder der Migration auf eine neuere Version sehr nützlich ist.

Die aktuellen Einstellungen für die Session und System-Parameter finden Sie in der Systemtabelle [EXA_PARAMETERS](#), wobei die im Default-Zustand folgende Werte für den Parameter `SCRIPT_LANGUAGES` gesetzt ist:

```
SELECT session_value FROM exa_parameters
WHERE parameter_name='SCRIPT_LANGUAGES';

PARAMETER_NAME
-----
PYTHON=builtin_python R=builtin_r JAVA=builtin_java
```

Diese Werte sind nicht besonders aussagekräftig, weil Sie nur interne Makros sind, um einerseits den Text kompakt zu halten und dynamisch stets die zuletzt installierte Version zu nutzen. Ausgeschrieben würde zum Beispiel der Java-Alias folgendermaßen lauten:

```
JAVA=localzmq+protobuf:///bfsdefault/default/EXAClusterOS/ScriptLanguages-2016-10-21/?lang=java#buckets/bfsdefault/default/EXASolution-6.0.0/exaudfclient
```

Hiermit wird die Exasol Datenbank derart konfiguriert, dass alle `CREATE JAVA SCRIPT` Statements den Sprach-Client `exaudfclient` im lokalen Verzeichnis `buckets/bfsdefault/default/EXASolution-6.0.0/` verwenden und im Exasol Linux Container aus dem Verzeichnis `bfsdefault/default/EXAClusterOS/ScriptLanguages-2016-10-21` starten. Das verwendete Protokoll für die Kommunikation lautet `localzmq+protobuf` (dies ist aktuell das einzige mögliche Protokoll).

Exasol benutzt für alle drei vorinstallierten Sprachen (Python, R, Java) nur einen einzigen Sprach-Client, der den Parameter `lang=java` auswertet, um zwischen den drei Sprachen unterscheiden zu können. Daher ist das interne Makro z.B. für den Python-Alias nahezu identisch. Vom Nutzer installierte Sprach-Clients können solche optionalen Parameter selbstständig definieren und auswerten.

Im Allgemeinen besteht ein Sprach-Alias also aus folgendem String:

```
<alias>=localzmq+protobuf://<path_to_linux-container>/[?<client_param_list>]#<path_to_client>
```



Bitte beachten Sie, dass Sprach-Aliase keine Leerzeichen enthalten dürfen

Dem genauen Betrachter wird bei dem Java-Alias vielleicht aufgefallen sein, dass der Pfad für den Linux-Container direkt mit dem BucketFS beginnt, während der Client den Präfix `buckets/` enthält. Der Grund hierfür ist, dass der Client innerhalb des Linux-Containers in einer sicheren Umgebung gestartet wird und lediglich auf diesen Container und die vorhandenen (und für den Benutzer sichtbaren) Buckets zugreifen kann. Ein Zugriff auf das echte Datei-Verzeichnis des Rechners ist somit nicht möglich, sondern nur auf die eingebundenen (mittels `mount`) Buckets. Der Pfad `/buckets/` kann dann innerhalb der Skripte rein lesbar benutzt werden, wie dies bereits im vorherigen Kapitel beschrieben wurde.

Sie können grundsätzlich beliebig viele Aliase definieren oder auch umbenennen. Wir empfehlen dabei wie schon erwähnt das Vorgehen, solche Änderungen am besten erstmal nur in der eigenen Session ([ALTER SESSION](#)) auszutesten, bevor die Änderungen mittels [ALTER SYSTEM](#) global sichtbar gemacht werden.

3.7. Virtuelle Schemas

3.7.1. Virtuelle Schemas und Tabellen

Virtual Schemas bzw. virtuelle Schemas bilden ein mächtiges Werkzeug, um beliebige Datenquellen an Exasol anbinden zu können. Virtuelle Schemas sind eine Art read-only Link zu einer externen Quelle und enthalten virtuelle Tabellen, die wie reguläre Tabellen aussehen, ohne lokal gespeichert zu sein.

 Bitte beachten Sie, dass virtuelle Schemas Teil der Advanced Edition von Exasol ist.

Nach der Erzeugung eines virtuellen Schemas können die darin enthaltenen Tabellen in SQL Anfragen benutzt werden und mit den normalen persistenten Tabellen, die in Exasol gespeichert sind, oder aber auch anderen virtuellen Tabellen von anderen virtuellen Schemas kombiniert werden. Der SQL Optimizer übersetzt die virtuellen Objekte in Verbindungen zu den unterliegenden Systemen und transferiert deren Daten implizit. SQL Bedingungen werden nach Möglichkeit zu den Datenquellen propagiert (=Push Down), um minimalen Datentransfer und optimale Performance zu erreichen.

Daher erzeugt dieses Konzept eine Art logische Sicht oberhalb diverser Datenquellen, welche andere Datenbanken oder auch Daten-Services sein könnten. Hierdurch können Sie entweder eine harmonisierte Zugriffsschicht für Ihre Reporting-Tools erzeugen, oder diese Technologie für agile und flexible ETL-Prozesse verwenden. Denn selbst bei Änderungen oder Erweiterungen der unterliegenden Daten-Objekte müssen Sie in Exasol selbst nichts ändern.

Das folgende Beispiel zeigt, wie einfach ein virtuelles Schema angelegt werden kann, indem unser JDBC Adapter eine Verbindung zu einem externen Hive System erzeugt.

```
-- Erzeugen des Schemas durch spezifische Adapter-Parameter
CREATE VIRTUAL SCHEMA hive
  USING adapter.jdbc_adapter
  WITH
    CONNECTION_STRING = 'jdbc:hive://myhost:21050;AuthMech=0'
    USERNAME          = 'hive-user'
    PASSWORD          = 'secret-password'
    SCHEMA_NAME       = 'default';

-- Zugreifen und durchforsten des virtuellen Schemas
OPEN SCHEMA hive;
SELECT * FROM cat;
DESCRIBE clicks;

-- Anfragen gegen die virtuellen Tabellen
SELECT count(*) FROM clicks;
SELECT DISTINCT USER_ID FROM clicks;
-- Anfragen können virtuelle und native Tabellen kombinieren
SELECT * from clicks JOIN native_schema.users ON clicks.userid = users.id;
```

SQL Befehle zum Verwalten von virtuellen Schemas:

CREATE VIRTUAL SCHEMA Erzeugen eines virtuellen Skripts (siehe auch [CREATE SCHEMA](#)).

DROP VIRTUAL SCHEMA Löschen eines virtuellen Schemas und der darin enthaltenen virtuellen Tabellen (siehe auch [DROP SCHEMA](#)).

ALTER VIRTUAL SCHEMA Ändern der Parameter eines virtuellen Schemas, oder aktualisieren der Metadaten mittels der REFRESH Option (siehe auch [ALTER SCHEMA](#)).

EXPLAIN VIRTUAL

Nützlich, um die resultierenden Anfragen an externe Systeme zu analysieren, die vom Exasol Compiler erzeugt werden (siehe [EXPLAIN VIRTUAL](#))

Anstatt nur eine bestimmte Anzahl an Konnektoren (so genannte Adapter) zu anderen Technologien mit auszuliefern, haben wir uns für ein offenes, erweiterbares Framework entschieden, in dem die Verbindungs-Logik über Open Source Projekte öffentlich zur Verfügung gestellt wird. Somit können Sie sehr einfach existierende Adapter nutzen, für die eigenen Bedürfnisse optimieren oder sogar ganz neue Adapter zu allen möglichen Datenquellen erstellen, ohne auf ein neues SW-Release von Exasol warten zu müssen.

In den folgenden Kapiteln werden wir Ihnen erklären, wie dieses Framework funktioniert und wo Sie die existierenden Adapter finden können.

3.7.2. Adapter und Properties

Beim Anlegen eines virtuellen Schemas muss der entsprechende Adapter - genau genommen ein Adapter-Skript - angegeben werden, das die Logik des Datenzugriffs zum externen System definiert. Dieser Zugriff variiert von Technologie zu Technologie, aber enthält immer zwei Hauptaufgaben:

Metadaten-Verarbeitung

Auslesen von Informationen zu den enthaltenen Daten-Objekten des Schemas (Tabellen, Spalten, Typen) sowie die Logik, wie die Datentypen des Quellsystems auf die Datentypen von Exasol abgebildet werden.

Push Down der Anfrage

Propagieren (Push down) von Teilen der Exasol SQL Anfrage zum externen System in Form einer geeigneten Query. Der Adapter definiert jeweils, welche Teile der SQL Logik propagiert werden kann (z.B. Filter und bestimmte Funktionen). Der Exasol Optimizer wird dann so viel wie möglich propagieren und den Rest der Ausführung lokal innerhalb des Exasol Clusters berechnen.

Adapter sind ähnlich wie UDF Skripte (siehe auch [Abschnitt 3.6, UDF Skripte](#) und die Details der nachfolgenden Kapitel). Diese können in einer der unterstützten Programmier-Sprachen implementiert werden, also z.B. Java oder Python, und haben auf die gleichen Metadaten Zugriff wie UDF Skripte. Um einen Adapter zu installieren, laden Sie einfach das entsprechende SQL Skript herunter und führen es aus, wodurch letztlich ein Adapter-Skript in einem der normalen Schemas erzeugt wird.



Die existierenden Open Source Adapter finden Sie in unserem GitHub Repository: <https://www.github.com/exasol/virtual-schemas>



Ein sehr generischer Adapter ist unser JDBC-Adapter. Grundsätzlich können Sie hiermit fast jede Datenquelle integrieren, die über einen JDBC-Treiber für Linux verfügt. Für einige Datenbank-Systeme wurden bereits Dialekte eingebaut, die einen möglichst großen Teil einer SQL-Query auf das unterliegende System propagieren (Push Down). Bitte beachten Sie, dass für den JDBC-Adapter der entsprechende JDBC-Treiber sowohl für den Zugriff aus den Adapter-Skripten heraus auf dem BucketFS gespeichert werden muss (siehe auch [Abschnitt 3.6.4, Das synchrone Cluster-Dateisystem BucketFS](#)), als auch in EXAoperation installiert werden muss (da der JDBC-Adapter einen impliziten IMPORT Befehl ausführt).

SQL Befehle zum Verwalten von Adapter-Skripten:

CREATE ADAPTER SCRIPT

Erzeugen eines Adapters, siehe [CREATE SCRIPT](#)

DROP ADAPTER SCRIPT

Löschen eines Adapters, siehe [DROP SCRIPT](#)

EXPLAIN VIRTUAL

Nützlich, um die resultierenden Anfragen an externe Systeme zu analysieren, die vom Exasol Compiler erzeugt werden, siehe [EXPLAIN VIRTUAL](#)

Das folgende Beispiel zeigt schematisch ein Adapter Skript, welches in Python geschrieben wurde.

```

CREATE SCHEMA adapter;
CREATE OR REPLACE PYTHON ADAPTER SCRIPT adapter.my_adapter AS
def adapter_call(request):
    # Implementieren Sie den Adapter hier
    # ...
    # und erzeugen eine entsprechende Antwort
/

```

Anschließend können Sie virtuelle Schemas erzeugen unter Angabe bestimmter Parameter (Properties), die für das Adapter Skript benötigt werden (siehe initiales Beispiel). Diese Parameter enthalten typischerweise die notwendigen Informationen, um eine Verbindung zum externen System aufzubauen. Im Beispiel war dies der JDBC Verbindungs-String sowie die Zugangsdaten.

Parameter können allerdings flexibel definiert werden und auf alle mögliche Art und Weise die Logik des Adapters kontrollieren. Falls Sie selbst einen Adapter entwickeln oder anpassen wollen, können Sie diese entsprechend erweitern.

Die Liste der definierten Parameter eines virtuellen Schemas finden Sie in der Systemtabelle [EXA_ALL_VIRTUAL_SCHEMA_PROPERTIES](#). Nach dem Anlegen des Schemas können Sie diese Parameter (Properties) mit dem SQL Befehl [ALTER SCHEMA](#) ändern.

Nachdem das virtuelle Schema auf diese Weise angelegt wurde, können Sie die darin enthaltenen Tabellen genauso nutzen wie ganz normale Tabellen in Exasol, und diese sogar miteinander verknüpfen. Sofern Sie diese Technologie für einfache ETL-Aufgaben nutzen möchten, können Sie natürlich einfach eine Abfrage auf die virtuellen Tabellen materialisieren:

```

CREATE VIRTUAL SCHEMA hive
USING adapter.jdbc_adapter
WITH
  CONNECTION_STRING = 'jdbc:hive://myhost:21050;AuthMech=0'
  USERNAME          = 'hive-user'
  PASSWORD          = 'secret-password'
  SCHEMA_NAME        = 'default';

CREATE TABLE my_schema.clicks_copy FROM
  (SELECT * FROM hive.clicks);

```

3.7.3. Zugriff auf virtuelle Tabellen ermöglichen

Der Zugriff auf die virtuellen Daten funktioniert analog wie bei der Erstellung von Views, indem Sie dem Nutzer einfach das SELECT-Privileg zuweisen. Im Falle von virtuellen Schemas ist dies nur auf das ganze Schema möglich (mittels GRANT SELECT ON SCHEMA), oder der Nutzer ist der Besitzer des Schemas.

Intern funktioniert dies, da analog zu Views die Prüfung der Rechte im Namen des Skript-Besitzers ausgeführt wird. Hierdurch werden die Details wie der Zugriff auf das Adapter-Skript sowie die Zugangsdaten zu den externen Systemen vom Nutzer der komplett gekapselt.

Zugriff auf einzelne virtuelle Tabellen einschränken

Wollen Sie Nutzern nicht Komplett-Zugriff auf das virtuelle Schema bzw. die Daten des externen Systems gewähren, sondern nur selektiv auf bestimmte Tabellen, so gibt es verschiedene Möglichkeiten der Umsetzung:

Views	Anstatt direkten Zugriff auf das virtuelle Schema zu gewähren, können Views auf diese Daten angelegt und Nutzern somit indirekt zur Verfügung gestellt werden.
-------	--

Logik im Adapter-Skript

Zudem gibt es die Möglichkeit, direkt im Adapter-Skript eine Logik umzusetzen. In unserem zur Verfügung gestellten JDBC-Adapter gibt es z.B. den Parameter `TABLE_FILTER`, über den nur eine Liste von bestimmten Tabellen sichtbar gemacht werden kann (siehe <https://www.github.com/exasol>). Wird dieser Parameter im virtuellen Schema nicht gesetzt, so werden alle verfügbaren Tabellen sichtbar.

3.7.4. Privilegien zur Administration

Die zum Erstellen und Administrieren von Adapter-Skripten und virtuellen Schemas benötigten Privilegien finden Sie grundsätzlich in den entsprechenden SQL-Befehlen `CREATE SCHEMA`, `ALTER SCHEMA`, `DROP SCHEMA` und `CREATE SCRIPT` bzw. in der Übersicht in Anhang B, *Details zur Rechteverwaltung*.

Zugriffsdaten über Connections kapseln

Wenn Sie das Einführungs-Beispiel noch einmal betrachten, dann werden Sie bemerken, dass in den Parametern des virtuellen Schemas die Verbindungsdaten zum externen System im Klartext stehen. Diese Methode ist zwar grundsätzlich sehr einfach umsetzbar, doch in den meisten Fällen nicht erwünscht - denn diese Informationen tauchen anschließend in bestimmten Systemtabellen auf. Daher empfehlen wir Ihnen die Verwendung von Connection-Objekten, um diese sicherheitsrelevanten Informationen zu kapseln.

Anstelle der Verbindungsdaten wird in den Parametern dann nur noch der Name der entsprechenden Connection gesetzt:

```
CREATE CONNECTION hive_conn
  TO 'jdbc:hive://myhost:21050;AuthMech=0'
  USER 'username'
  IDENTIFIED BY 'secret-password';

CREATE VIRTUAL SCHEMA vs_hive
  USING adapter.jdbc_adapter
  WITH
    SQL_DIALECT = 'HIVE'
    CONNECTION_NAME = 'HIVE_CONN'
    SCHEMA_NAME = 'default';
```

Das Adapter-Skript muss dies natürlich entsprechend umsetzen und die Details der Connection auslesen, um die Verbindung zum externen System aufbauen zu können.

Der Administrator des virtuellen Schemas benötigt folgende Rechte, damit dieses Konzept der Kapselung durch Connections funktionieren kann:

1. Zugriff auf die Connection (`GRANT CONNECTION`), denn bei der zweistufigen Abarbeitung wird in den meisten Fällen von Adapter-Skripten intern ein `IMPORT`-Statement generiert (siehe auch weiter unten [Abschnitt 3.7.6, Details für Experten](#)), welches diese Connection ganz wie normale `IMPORT`-Befehle verwendet. Da der `IMPORT`-Befehl ein Exasol-interner Befehl ist und kein öffentliches Skript, bleiben die Verbindungsdaten selbst geschützt.
2. Zudem benötigt man in den meisten Fällen zusätzlich Zugriff auf die Verbindungsdetails der Connection (also *User* und *Password*), weil das Adapter-Skript bei Befehlen wie `CREATE` oder `REFRESH` eine direkte Verbindung zum externen System aufbaut, um die Metadaten zu lesen. Zu diesem Zweck wurde speziell das `ACCESS` Recht eingeführt, da diese Verbindungsdetails Datenschutz-technisch sehr kritisch sind. Mit dem Befehl `GRANT ACCESS ON CONNECTION [FOR SCRIPT]` können Sie diesen Zugriff auch nur für ein bestimmtes Skript gewähren (FOR SCRIPT Klausel) und stellen dadurch sicher, dass der Administrator eines virtuellen Schemas auf die Zugangsdaten selbst nicht zugreifen kann (z.B. durch Erzeugung eines eigenen Skriptes, welche die Zugangsdaten ausliest und ausgibt). Natürlich sollte er dann das Adapter-Skript nur ausführen, aber auf keinen Fall ändern dürfen.

Im folgenden Beispiel werden einem Nutzer die benötigten Privilegien gewährt, um ein virtuelles Schema unter Nutzung des Adapter-Skripts (`jdbc_adapter`) und einer bestimmten Verbindung (`hive_conn`) zu erzeugen und zu verwalten.

```
GRANT CREATE VIRTUAL SCHEMA TO user_hive_access;
GRANT EXECUTE ON SCRIPT adapter.jdbc_adapter TO user_hive_access;
GRANT CONNECTION hive_conn TO user_hive_access;
GRANT ACCESS ON CONNECTION hive_conn
    FOR SCRIPT adapter.jdbc_adapter TO user_hive_access;
```

3.7.5. Meta-Daten

Für die im System angelegten Adapter-Skripte und virtuellen Schemas finden Sie diverse Systemtabellen mit den entsprechenden Detail-Informationen.

Virtuelle Schemas

- [EXA_VIRTUAL_SCHEMAS](#)
- [EXA_ALL_VIRTUAL_SCHEMA_PROPERTIES](#)
- [EXA_USER_VIRTUAL_SCHEMA_PROPERTIES](#)
- [EXA_DBA_VIRTUAL_SCHEMA_PROPERTIES](#)
- [EXA_ALL_VIRTUAL_TABLES](#)
- [EXA_DBA_VIRTUAL_TABLES](#)
- [EXA_USER_VIRTUAL_TABLES](#)
- [EXA_ALL_VIRTUAL_COLUMNS](#)
- [EXA_DBA_VIRTUAL_COLUMNS](#)
- [EXA_USER_VIRTUAL_COLUMNS](#)

Adapter Skripte

- [EXA_ALL_SCRIPTS](#) (`SCRIPT_TYPE='ADAPTER'`)
- [EXA_DBA_SCRIPTS](#) (`SCRIPT_TYPE='ADAPTER'`)
- [EXA_USER_SCRIPTS](#) (`SCRIPT_TYPE='ADAPTER'`)

Eingeschränkte ACCESS Rechte für Verbindungen

- [EXA_DBA_RESTRICTED_OBJ_PRIVS](#)
- [EXA_ROLE_RESTRICTED_OBJ_PRIVS](#)
- [EXA_USER_RESTRICTED_OBJ_PRIVS](#)

Verbindungen (Connections)

- [EXA_ALL_CONNECTIONS](#)
- [EXA_DBA_CONNECTIONS](#)
- [EXA_SESSION_CONNECTIONS](#)
- [EXA_DBA_CONNECTION_PRIVS](#)
- [EXA_USER_CONNECTION_PRIVS](#)

3.7.6. Details für Experten

Falls Sie existierende Adapter zum Anlegen von virtuellen Schemas einfach nur nutzen wollen, dann sollten Sie an diesem Punkt hoffentlich genug Informationen haben,

1. Wo man die benötigten Adapter Skripte herunterladen kann
2. Wie man dieses Skript in Ihrer Exasol Datenbank installiert
3. Wie Sie virtuelle Schemas anlegen und benutzen können

Sofern Sie jedoch mehr über die zugrunde liegenden Konzepte verstehen wollen oder sogar planen, selbst Adapter zu erzeugen oder anzupassen, dann finden Sie hier zusätzliche Informationen.

Bei jedem Zugriff auf ein virtuelles Schema wird auf einem der Cluster-Knoten ein Container der entsprechenden Skript-Sprache gestartet, also z.B. eine JVM oder ein Python Container. Innerhalb des Containers wird der Code des Adapter Skripts geladen und Exasol kommuniziert mit diesem Skript mit Hilfe eines einfachen Request-Response Protokolls, welches in JSON kodiert ist. Die Datenbank übernimmt hierbei den aktiven Part, indem eine Callback-Methode aufgerufen wird.

Im Falle von Python lautet diese Methode `adapter_call`, aber dies kann in anderen Sprachen variieren.

Um die genauen internen Abläufe zu verstehen, wollen wir uns ein einfaches Beispiel einer Anfrage betrachten:

```
SELECT name FROM my_virtual_schema.users WHERE name like 'A%';
```

Folgendes geschieht dann intern:

1. Exasol erkennt, dass eine virtuelle Tabelle involviert ist und greift auf das entsprechende Adapter Skript zu und startet den entsprechenden Sprach-Container auf einem der Knoten im Exasol Cluster.
2. Exasol sendet einen Anfrage zum Adapter, der die Fähigkeiten des Adapters beschreiben soll.
3. Der Adapter liefert eine Antwort mit den entsprechenden Fähigkeiten, z.B. ob bestimmte WHERE-Bedingungen oder skalare Funktionen unterstützt werden.
4. Exasol sendet einen entsprechenden Push Down, der die spezifischen Fähigkeiten berücksichtigt. Beispielsweise sind hier Informationen zu Spalten-Projektionen (im Beispiel oben wird nur die einzelne Spalte `name` benötigt) oder Filter-Bedingungen enthalten.
5. Der Adapter verarbeitet diese Anfrage und liefert eine SQL Anfrage in Exasol-Syntax zurück, welche anschließend ausgeführt wird. Diese Anfrage ist typischerweise ein IMPORT Statement oder ein SELECT Statement, das ein UDF EMITS Skript enthält, also um die Daten-Verarbeitung kümmert und Zeilen erzeugt.

Das obige Beispiel könnte daher in eine der folgenden zwei Alternativen übersetzt werden (IMPORT und SELECT):

```
SELECT name FROM ( IMPORT FROM JDBC AT ... STATEMENT 'SELECT name from remoteschema.users WHERE name LIKE "A%"' );
```

```
SELECT name FROM ( SELECT GET_FROM_MY_DATA_SOURCE('data-source-address', 'required_column=name') ) WHERE name LIKE 'A%';
```

In der ersten Variante kann der Adapter Filter-Bedingungen verarbeiten und erzeugt ein IMPORT-Befehl, der ein Statement auf dem externen System ausführt. In der zweiten Variante wird ein UDF Skript mit zwei Parametern benutzt und darüber die Adresse der Datenquelle und die entsprechende Spalten-Projektion definiert, jedoch keine Logik für die Filter-Bedingung weiter gibt. Diese würde dann innerhalb Exasol und nicht im externen System angewandt werden.

Bitte beachten Sie, dass bei den Beispielen die vollständig transformierte Query angegeben ist und nicht nur das vom Adapter-Skript gelieferte innere Statement.

6. Die empfangenen Daten werden direkt in die übergeordnete Query-Verarbeitung von Exasol integriert.

Um die vollständige API der Adapter Skripte zu verstehen, verweisen wir auf unser Open Source Repository (<https://www.github.com/exasol>) und unsere existierenden Adapter, die eine API Dokumentation enthalten. Zudem

ist es deutlich einfacher, anhand dieser Beispiel-Implementierungen die Details nachzuvollziehen, als diese in diesem Handbuch zu beschreiben.

Falls Sie existierende Adapter erweitern oder komplett neue entwickeln wollen, empfehlen wir die Nutzung des [EXPLAIN VIRTUAL](#) Befehls. Hiermit können Sie einfach analysieren, was genau der Adapter an die unterliegende Systeme schickt.

3.8. SQL-Präprozessor

3.8.1. Wie arbeitet der SQL-Präprozessor?

In Exasol wird ein SQL-Präprozessor zur Verfügung gestellt, mit dessen Hilfe sämtliche ausgeführte SQL-Statements vorverarbeitet werden können. Auf diese Weise können zum Beispiel nicht unterstützte SQL-Konstrukte automatisch in existierende SQL-Features umgebaut werden (siehe Beispiele unten). Auch der Einbau von syntaktischen Spezialkonstrukten zur einfacheren Schreibweise wird ermöglicht (syntactic sugar).



Falls Sie den SQL-Präprozessor benutzen wollen, lesen Sie sich bitte dieses Kapitel aufmerksam durch, da die Auswirkungen auf Ihr Datenbank-System sehr weitreichend sein können.

Der SQL-Präprozessor ist standardmäßig deaktiviert. Mit den Befehlen [ALTER SESSION](#) bzw. [ALTER SYSTEM](#) kann session- bzw. systemweit ein Skript definiert werden, das ab diesem Zeitpunkt sämtliche SQL-Befehle vorverarbeitet. Bevor der eigentliche Datenbank-Compiler das SQL weitergereicht bekommt, führt der Präprozessor eine Art Textersetzung durch. Innerhalb des Skriptes kann mittels get()- bzw. set()-Methoden das SQL eingelesen und angepasst werden, wobei unsere Hilfsbibliothek (siehe nächster Abschnitt) diese Verarbeitung erleichtert. Details zur Skriptsprache stehen in [Abschnitt 3.5, Scripting-Programmierung](#) zur Verfügung.

Der SQL-Präprozessor ist einerseits ein mächtiges Instrument, um den SQL-Sprachumfang von Exasol flexibel erweitern zu können. Allerdings sollten Sie sehr vorsichtig sein, bevor Sie solch eine SQL-Manipulation systemweit aktivieren. Im schlimmsten Fall wird kein einziges SQL-Statement mehr funktionieren. Allerdings können Sie über [ALTER SESSION](#) und [ALTER SYSTEM](#) jederzeit problemlos den SQL-Präprozessor wieder deaktivieren (durch Setzen des Parameters `SQL_PREPROCESSOR_SCRIPT` auf den `NULL`), da diese beiden Befehle bewusst von der Vorverarbeitung ausgenommen sind. Aus Datenschutz-Gründen werden auch alle Statements, die Passwörter enthalten, von der Vorverarbeitung ausgeschlossen ([CREATE USER](#), [ALTER USER](#), [CREATE CONNECTION](#), [ALTER CONNECTION](#), [IMPORT](#), [EXPORT](#), sofern die `IDENTIFIED BY` Klausel angegeben wurde).

In der Auditing-Tabelle [EXA_DB_AUDIT_SQL](#) erscheint übrigens ein eigener Eintrag mit der Ausführung des Präprozessor-Skripts (EXECUTE SCRIPT mit Original-Text als Kommentar). Das nachher ausgeführte geänderte SQL Statement wird in einem weiteren Eintrag mitprotokolliert.

3.8.2. Bibliothek sqlparsing

Zur möglichst einfachen Verarbeitung liefern wir eine spezielle Bibliothek mit, die diverse Funktionalitäten zum SQL-Parsing vereint.

Übersicht

Zerlegen in Token	<ul style="list-style-type: none"> • <code>tokenize()</code>
Identifizieren von Token-Typen	<ul style="list-style-type: none"> • <code>iswhitespace()</code> • <code>iscomment()</code> • <code>iswhitespaceorcomment()</code> • <code>isidentifier()</code> • <code>iskeyword()</code> • <code>isstringliteral()</code> • <code>isnumericliteral()</code> • <code>isany()</code>
Normalisieren eines Textes	<ul style="list-style-type: none"> • <code>normalize()</code>
Suchen von Token-Folgen	<ul style="list-style-type: none"> • <code>find()</code>

Zugriff auf den SQL-Text

- getsqltext()
- setsqltext()

Details

- **sqlparsing.tokenize(sqlstring)**

Zerlegt den Eingabe-String in ein Array aus einzelnen Strings, die den vom Datenbank-Compiler erkannten Token entsprechen. Alle Token konkateniert ergeben wieder exakt den Eingabestring (inkl. Groß-Kleinschreibung, Zeilenumbrüchen, Leerzeichen, etc.), also gilt: `table.concat(tokens) == sqlstring`.

Folgende Tokentypen sind möglich:

- Gültige SQL-Bezeichner, z.B. `test.tab` oder `"foo"."bar"`
- Schlüsselwörter, z.B. `SELECT`
- String-Literale, z.B. `'abcdef'`
- Numerische Literale ohne Vorzeichen, z.B. `123e4`
- Zusammenhängende Whitespaces entsprechend dem SQL-Standard
- Kommentare entsprechend dem SQL-Standard (Zeilen- und Block-Kommentare)
- Mehr-Zeichen-Operatoren, z.B. `'::', ':::', '||', '>=', '<=', '<>', '!=', '^='`
- Ein-Zeichen-Token, z.B. `'+', '!', '/', '*', '~', '>', '<'`

```
CREATE SCRIPT example(sql_text) AS
    local tokens = sqlparsing.tokenize(sql_text)
    for i=1,#tokens do
        print(tokens[i])
    end
/
EXECUTE SCRIPT example('SELECT dummy FROM dual') WITH OUTPUT;
OUTPUT
-----
SELECT
dummy
FROM
dual
```

- **sqlparsing.iscomment(tokenstring)**

Liefert zurück, ob der Eingabe-String ein Kommentar-Token ist.

- **sqlparsing.iswhitespace(tokenstring)**

Liefert zurück, ob der Eingabe-String ein Whitespace-Token ist.

- **sqlparsing.iswhitespaceorcomment(tokenstring)**

Liefert zurück, ob der Eingabe-String ein Whitespace- oder Kommentar-Token ist. Diese Funktion ist nützlich z.B. für die Verwendung in der Funktion `find()`, da sie alle bzgl. SQL-Standard nicht relevanten Token filtert.

- **sqlparsing.isidentifier(tokenstring)**

Liefert zurück, ob der Eingabe-String ein Bezeichner-Token ist.

- **sqlparsing.iskeyword(tokenstring)**

Liefert zurück, ob der Eingabe-String ein SQL-Keyword-Token ist (z.B. SELECT, FROM, TABLE). Die Funktionen `isidentifier()` und `iskeyword()` liefern für nicht reservierte Schlüsselwörter beide `true`. Daran kann man dann auch die nicht reservierten Schlüsselwörter erkennen.

- **`sqlparsing.isstringliteral(tokenstring)`**

Liefert zurück, ob der Eingabe-String ein String-Literal-Token ist.

- **`sqlparsing.isnumericalliteral(tokenstring)`**

Liefert zurück, ob der Eingabe-String ein numerisches Literal-Token ist.

- **`sqlparsing.isany(tokenstring)`**

Liefert immer `true` zurück. Dies kann z.B. dann sinnvoll sein, wenn man ein beliebiges erstes relevantes Token finden möchte (als Match-Funktion für die Funktion `find()`).

- **`sqlparsing.normalize(tokenstring)`**

Liefert bei Token mit mehreren gültigen Darstellungen (z.B. Bezeichner mit der Groß-Kleinschreibung) eine normalisierte Darstellung, anhand folgender Punkte:

- Reguläre Bezeichner werden in Großbuchstaben umgewandelt, z.B. dual -> DUAL
- Schlüsselwörter werden in Großbuchstaben umgewandelt, z.B. From -> FROM
- Ein beliebig langes Whitespace-Token wird durch ein einzelnes Leerzeichen ersetzt
- Bei numerischen Literalen wird ein eventuell vorhandenes kleines 'e' in ein großes 'E' umgewandelt, z.B 1.2e34 -> 1.2E34

- **`sqlparsing.find(tokenlist, startTokenNr, searchForward, searchSameLevel, ignoreFunction, match1, [match2, ... matchN])`**

Sucht in der Tokenliste ab dem Token an Position `startTokenNr` vorwärts oder rückwärts (`searchForward`) und optional nur in der aktuellen Klammerungsebene (`searchSameLevel`) nach der unmittelbar aufeinander folgenden Tokensequenz, die durch die Parameter `match1, ... matchN` gematched wird. Dabei werden alle Token, die durch die Funktion `ignoreFunction` gematched werden, nicht für einen Match berücksichtigt.

Wird die gesuchte Tokenfolge gefunden, dann wird ein Array der Länge N zurückgeliefert (bei N Match-Elemente), an dessen Position X die Nummer des Token in der Tokenliste steht, das von `matchX` gematched wurde. Wird die gesuchte Tokenfolge nicht gefunden, so liefert die Funktion `nil` zurück.

Details zu den Parametern:

<code>tokenlist</code>	Die Liste der Token, wie sie z.B. von der Funktion <code>tokenize</code> erzeugt wird
<code>startTokenNr</code>	Die Nummer des ersten Token, das bei der Suche berücksichtigt werden soll
<code>searchForward</code>	Definiert, ob die Suche vorwärts (<code>true</code>) oder rückwärts (<code>false</code>) durchgeführt werden soll. Das betrifft nur die Richtung, mit der durch die Token-Liste gewandert wird. Die match-Funktionen selbst werden immer von vorne nach hinten gematched. Sucht man beispielsweise in der Tokenliste 'select' 'abc' 'from' 'dual' die Tokenfolge KEYWORD, IDENTIFIER, dann wird bei Start-Position 3 immer 'from' 'dual' gematched, auch bei Rückwärtssuche, nicht etwa 'from' 'abc'. Bei Suche ab Position 2 liefert die Rückwärtssuche 'select' 'abc' und die Vorwärtssuche 'from' 'dual'.
<code>searchSameLevel</code>	Definiert, ob nur in der aktuellen Klammerungsebene gesucht werden soll (<code>true</code>) oder auch außerhalb (<code>false</code>). Dies bezieht sich nur auf den Match des ersten Token in der Sequenz. Weitere Token können dann auch weiter innen liegen, z.B. ist ein Match der Sequenz '=' '(' 'SELECT' auch mit Suche nur auf der aktuellen Klamme-

rungsebene möglich, obwohl das 'SELECT' dann natürlich eine Klammerungsebene weiter innen liegt.

Die Option `searchSameLevel` ist vor allem interessant, um eine passende schließende Klammer zu finden, z.B. einer Subquery. Beispiel: Suche in der Tokenfolge 'SELECT' 't1.x' '+' '(' 'SELECT' 'min' '(' 'y' ')' 'FROM' 't2' ')' 'FROM' 't1' die passende schließende Klammer zur öffnenden Klammer an Position 4: `sqlparsing.find(tokens, 4, true, true, sqlparsing.iswhitespaceorcomment, ') ')`.

ignoreFunction

Hier wird eine Funktion der Art `function(string) -> bool` erwartet. Alle Token, für die die `ignoreFunction` `true` liefert, werden von den match-Funktionen ignoriert. Das heißt, mit dieser Funktion kann insbesondere angegeben werden, welche Token zwischen den Token der gesuchten Tokenfolge vorkommen dürfen, so dass es trotzdem noch einen Match gibt. Häufig macht hier die Funktion `iswhitespaceorcomment` Sinn.

match1...matchN

Durch `match1..matchN` wird die gesuchte Tokenfolge definiert. Dies sind entweder Funktionen vom Typ `function(tokenstring) -> bool` oder einfache Strings. Es wird eine Tokenfolge gesucht, bei der das erste Token von `match1` gematched wurde, das zweite von `match2`, etc., wobei dazwischen Token ignoriert werden, für die die Funktion `ignoreFunction` `true` liefert. Wenn ein Parameter ein String ist, wird immer der Vergleich `normalize(tokenstring) == normalize(matchstring)` durchgeführt.

- **sqlparsing.getsqltext()**

Liefert den aktuellen SQL Statement Text zurück.



Diese Funktion ist nur innerhalb des SQL-Präprozessor-Skripts benutzbar

- **sqlparsing.setsqltext(string)**

Setzt den SQL Statement Text auf einen neuen Wert. Dieser wird dann nach dem SQL-Präprozessor an den Datenbank-Compiler zur eigentlichen Ausführung weitergegeben.



Diese Funktion ist nur innerhalb des SQL-Präprozessor-Skripts benutzbar

3.8.3. Best Practice

Damit die Entwicklung geeigneter Skripten und die globale Aktivierung des SQL-Präprozessor möglichst reibungslos funktionieren, sollten Sie folgende Hinweise beachten:

- Sie sollten ein Präprozessor-Skript stets in Ihrer eigenen Session ausgiebig austesten, bevor Sie es systemweit aktivieren.
- Die eigentliche SQL-Verarbeitung sollte in separaten Hilfs-Skripten implementiert werden und in einem Hauptskript eingebunden werden, welches lediglich eine Hülle bildet und den SQL-Text übergibt (z.B. `sqlparsing.setsqltext(myscript.preprocess(sqlparsing.getsqltext()))`). Hintergrund ist, dass die Funktionen `getsqltext()` und `setsqltext()` nur innerhalb der Präprozessor-Ausführung zur Verfügung stehen und nicht bei einer normalen Skript-Ausführung. Durch die Separierung ist es möglich, die Verarbeitung vorher auf Test-SQL-Konstrukten auszutesten (z.B. die täglich ausgeführten SQL Statements, gespeichert in einer Tabelle), bevor das umschließende Hauptskript als Präprozessor-Skript aktiviert wird.

- Stellen Sie sicher, dass die nötigen Privilegien zur Ausführung des Präprozessor-Skripts vergeben wurden. Testen Sie nach systemweiter Aktivierung unbedingt mit einem Nutzer ohne spezielle Rechte. Ansonsten könnten bestimmte Nutzergruppen blockiert werden, SQL Befehle auszuführen.
- Die Vorverarbeitung sollte so einfach wie möglich gehalten werden. Besonders `query()` und `pquery()` sollten nur in Ausnahmefällen benutzt werden, sofern der Präprozessor systemweit aktiviert wird. Denn es werden dadurch sämtliche SQL-Anfragen verlangsamt und durch parallelen Zugriff auf gleiche Tabelle das Transaktionskonflikt-Risiko erhöht.

3.8.4. Beispiele

Im Folgenden finden Sie ein paar Beispiele für Präprozessor-Skripte, die Ihnen die Funktionsweise und Möglichkeiten näher bringen sollen.

Beispiel 1: IF()-Funktion

In diesem Beispiel wird die IF()-Funktion, welche aktuell nicht in Exasol existiert, in eine äquivalente CASE-WHEN Anweisung umgewandelt.

```
CREATE SCRIPT transformIf() AS
function processIf(sqltext)
    while (true) do
        local tokens = sqlparsing.tokenize(sqltext)
        local ifStart = sqlparsing.find(tokens,
                                         1,
                                         true,
                                         false,
                                         sqlparsing.iswhitespaceorcomment,
                                         'IF',
                                         '(')

        if (ifStart==nil) then
            break;
        end
        local ifEnd = sqlparsing.find(tokens,
                                         ifStart[2],
                                         true,
                                         true,
                                         sqlparsing.iswhitespaceorcomment,
                                         ')')

        if (ifEnd==nil) then
            error("if statement not ended properly")
            break;
        end
        local commas1 = sqlparsing.find(tokens,
                                         ifStart[2]+1,
                                         true,
                                         true,
                                         sqlparsing.iswhitespaceorcomment,
                                         ',')
        if (commas1==nil) then
            error("invalid if function")
            break;
        end
        local commas2 = sqlparsing.find(tokens,
                                         commas1[1]+1,
                                         true,
                                         true,
```

```

        sqlparsing.iswhitespaceorcomment,
        ', ' )

if (commas2==nil) then
    error("invalid if function")
    break;
end
local ifParam1=table.concat(tokens, '', ifStart[2]+1, commas1[1]-1)
local ifParam2=table.concat(tokens, '', commas1[1]+1, commas2[1]-1)
local ifParam3=table.concat(tokens, '', commas2[1]+1, ifEnd[1]-1)
local caseStmt='CASE WHEN ('..ifParam1..'') != 0 \
                THEN ('..ifParam2..'') \
                ELSE ('..ifParam3..'') END '
sqltext=table.concat(tokens, '', 1,
                     ifStart[1]-1)..caseStmt..table.concat(tokens,
                     '',
                     ifEnd[1]+1)

end
return sqltext
end
/
CREATE SCRIPT sql_preprocessing.preprocessIf() AS
import( 'sql_preprocessing.transformIf', 'transformIf' )
sqlparsing.setsqltext(
    transformIf.processIf(sqlparsing.getsqltext()))
/
SELECT IF( 3+4 > 5, 6, 7 ) from dual;
Error: [42000] syntax error, unexpected IF_ [line 1, column 8]

ALTER SESSION SET sql_preprocessor_script=
    sql_preprocessing.preprocessIf;
SELECT IF( 3+4 > 5, 6, 7 ) AS col1 FROM dual;

COL1
-----
  6

```

Beispiel 2: ls-Kommando

Das aus der Unix-Welt bekannte Kommando **ls** soll auf die Datenbankwelt übertragen werden. So gibt das Kommando entweder die Liste aller Objekte in einem Schema aus oder die Liste aller Schemas, falls keines geöffnet ist. Zusätzlich lassen sich auch Filter (case-insensitiv) anwenden, also z.B. **ls '%name%'** zur Anzeige aller Objekte, die im Namen den Text 'name' enthalten.

```

CREATE SCRIPT sql_preprocessing.addunixcommands( ) AS
function processLS(input, tokens, commandPos)
    local result = query("SELECT CURRENT_SCHEMA")
    local current_schema = result[1][1]
    local returnText = ""
    local searchCol = ""
    if (current_schema==null) then
        returnText = "SELECT schema_name FROM exa_schemas WHERE true"
        searchCol = "schema_name"
    elseif (current_schema=='SYS' or current_schema=='EXA_STATISTICS') then
        returnText = "SELECT object_name, object_type FROM exa_syscat \

```

```

        WHERE schema_name='.." . . . current_schema..'''"
    searchCol = "object_name"
else
    returnText = "SELECT object_name, object_type FROM exa_all_objects \
                  WHERE root_type='SCHEMA' \
                  AND root_name='.." . . . current_schema..'''"
    searchCol = "object_name"
end
local addFilters = {}
local lastValid = commandPos
local foundPos = sqlparsing.find(tokens,
                                  lastValid+1,
                                  true,
                                  false,
                                  sqlparsing.iswhitespaceorcomment,
                                  sqlparsing.isany)
while (not(foundPos==nil) )
do
    local foundToken = tokens[foundPos[1]]
    if (sqlparsing.isstringliteral(foundToken)) then
        addFilters[#addFilters+1] = "UPPER(..searchCol..") \
                                      LIKE UPPER(..foundToken .. "")"
    elseif (not (sqlparsing.normalize(foundToken) == ';')) then
        error("only string literals allowed as arguments for ls,\n\
              but found '..foundToken..'''")
    end
    lastValid = foundPos[1]
    foundPos = sqlparsing.find(tokens,
                                lastValid+1,
                                true,
                                false,
                                sqlparsing.iswhitespaceorcomment,
                                sqlparsing.isany)
end
if ( #addFilters > 0 ) then
    local filterText = table.concat(addFilters, " OR ")
    return returnText.." AND(..filterText..")..." ORDER BY ..searchCol
else
    return returnText.." ORDER BY ..searchCol
end
end

function processUnixCommands(input)
    local tokens = sqlparsing.tokenize(input)
    local findResult = sqlparsing.find(tokens,
                                       1,
                                       true,
                                       false,
                                       sqlparsing.iswhitespaceorcomment,
                                       sqlparsing.isany)
    if (findResult==nil) then
        return input
    end
    local command = tokens[findResult[1]]
    if (sqlparsing.normalize( command )=='LS') then
        return processLS(input, tokens, findResult[1])
    end

```

```
    return input;
end
/
CREATE SCRIPT sql_preprocessing.preprocessWithUnixTools() AS
  import( 'sql_preprocessing.addunixcommands', 'unixCommands' )
  sqlparsing.setsqltext(
    unixCommands.processUnixCommands(sqlparsing.getsqltext()));
/
ALTER SESSION SET sql_preprocessor_script=
  sql_preprocessing.preprocessWithUnixTools;

OPEN SCHEMA sql_preprocessing;
LS '%unix%';

OBJECT_NAME          OBJECT_TYPE
-----
ADDUNIXCOMMANDS      SCRIPT
PREPROCESSWITHUNIXTOOLS  SCRIPT

CLOSE SCHEMA;
LS;

SCHEMA_NAME
-----
SCHEMA_1
SCHEMA_2
SCHEMA_3
SQL_PREPROCESSING
```

Beispiel 3: ANY/ALL

ANY bzw. ALL sind SQL-Konstrukte, welche noch nicht von Exasol unterstützt werden. Mit Hilfe des nachfolgenden Skriptes kann diese Funktionalität jedoch zur Verfügung gestellt werden.

```
CREATE SCRIPT sql_preprocessing.transformAnyAll() AS
function rebuildAnyAll(inputsqltext)
    local sqltext = inputsqltext;
    local tokens = sqlparsing.tokenize(sqltext);
    local found = true;
    local searchStart = 1;
    -- search for sequence >|>=|<|<= ANY|ALL ( SELECT
repeat
    local foundPositions =
        sqlparsing.find(tokens,
            searchStart,
            true,
            false,
            sqlparsing.iswhitespaceorcomment,
            function (token)
                return (token=='<' or token=='<=' or token==">>' or
                    token=='>=' or token=='!=' or token=='<>' or
                    token=='=' );
            end,      -- match <|<=|>|>=|=|=|<>
            function ( token )
```

```

        local normToken = sqlparsing.normalize(token);
        return (normToken=='ANY' or normToken=='SOME'
               or normToken=='ALL');
    end,      -- match ANY|ALL
    '(' ,     -- match (
    'SELECT' -- match SELECT
);
if (foundPositions==nil) then
    found = false;
    break;
end
local operatorPos = foundPositions[1];
local anyAllPos = foundPositions[2];
local openBracketPos = foundPositions[3];
searchStart = anyAllPos + 1
foundPositions = sqlparsing.find(tokens,
                                  openBracketPos,
                                  true,
                                  true,
                                  sqlparsing.iswhitespaceorcomment,
                                  ')');
if (foundPositions ~= nil) then
    local closeBracketPos = foundPositions[1]
    local operatorToken = tokens[operatorPos];
    local anyOrAll = sqlparsing.normalize(tokens[anyAllPos]);
    if (operatorToken=='<' or operatorToken=='<='
        or operatorToken=='>' or operatorToken=='>=')
        then
        -- now we have <|<=>|>= ANY|ALL (SELECT <something> FROM
        -- rebuild to <|<=>|>= (SELECT MIN|MAX(<something>) FROM
        local setfunction = 'MIN';
        if ((anyOrAll=='ANY' or anyOrAll=='SOME') and
            (operatorToken=='<' or operatorToken=='<=')
            ) or
            (anyOrAll=='ALL' and (operatorToken=='>'
                                  or operatorToken=='>=')
            )
        ) then
            setfunction = 'MAX';
    end
    tokens[anyAllPos] = '';
    tokens[openBracketPos] =
        '(SELECT ' .. setfunction .. '(anytab.anycol) FROM (' ;
    tokens[closeBracketPos] = ')' as anytab(anycol) ')';
elseif (operatorToken=='=' and anyOrAll=='ALL') then
    -- special rebuild for = ALL
    -- rebuild to=(SELECT CASE WHEN COUNT(DISTINCT <something>)==1
    --                  THEN FIRST_VALUE(<something>) ELSE NULL END FROM
    tokens[anyAllPos] = '';
    tokens[openBracketPos] =
        '(SELECT CASE WHEN COUNT(DISTINCT anytab.anycol) = 1 \
                  THEN FIRST_VALUE(anytab.anycol) ELSE NULL END FROM (' ;
    tokens[closeBracketPos] = ')' as anytab(anycol) ')';
elseif ((operatorToken=='!=') or (operatorToken=='<>'))
        and anyOrAll=='ALL') then
    -- special rebuild for != ALL
    -- rebuild to NOT IN
    tokens[operatorPos] = ' NOT IN '

```

```

tokens[anyAllPos] = ''
elseif (operatorToken=='!=') and
    (anyOrAll=='ANY' or anyOrAll=='SOME')) then
--special rebuild for != ANY, rebuild to
-- CASE WHEN (SELECT COUNT(DISTINCT <something>) FROM ...) == 1
-- THEN operand != (SELECT FIRST_VALUE(<something>) FROM ...)
-- ELSE operand IS NOT NULL END
--note: This case would normally require to determine the operand
--      which requires full understanding of a value expression
--      in SQL standard which is nearly impossible in
--      preprocessing (and very susceptible to errors)
--      so we evaluate the
--      SELECT COUNT(DISTINCT <something>) FROM ... == 1 here and
--      insert the correct expression
--
-- first preprocess the inner query
local queryText = table.concat(tokens,
                                '',
                                openBracketPos,
                                closeBracketPos)
queryText = rebuildAnyAll( queryText )
-- since the subquery was already processed we can continue
-- searching *after* the SELECT
searchStart = closeBracketPos + 1
local distinctQueryText='SELECT COUNT(DISTINCT anytab.anycol) \
                           FROM '..queryText..'' AS anytab(anycol)'
local success, result = pqquery(distinctQueryText)
if (success) then
    if (result[1][1] == 1) then
        tokens[anyAllPos] = '(SELECT FIRST_VALUE(anytab.anycol) \
                           FROM '..queryText..'' AS anytab(anycol))'
    else
        tokens[operatorPos] = ' IS NOT NULL '
        tokens[anyAllPos] = ''
    end
    -- clear all tokens of the SELECT
    for curTokenNr=openBracketPos,closeBracketPos do
        tokens[curTokenNr] = ''
    end
end
until found == false;
return table.concat(tokens);
end
/
CREATE SCRIPT sql_preprocessing.preprocessAnyAll AS
import('sql_preprocessing.transformAnyAll', 'anyallparser');
sqlparsing.setsqltext(
    anyallparser.rebuildAnyAll(sqlparsing.getsqltext()))
/
CREATE TABLE t1 (i INT);
INSERT INTO t1 VALUES 1,2,3,4,5,6,7,8,9,10;
CREATE TABLE t2 (j INT);
INSERT INTO t2 VALUES 5,6,7;

```

```
SELECT i FROM t1 WHERE i < ALL(SELECT j FROM t2);
Error: [0A000] Feature not supported: comparison with quantifier ALL

ALTER SESSION SET sql_preprocessor_script=
          sql_preprocessing.preprocessAnyAll;
SELECT i FROM t1 WHERE i < ALL(SELECT j FROM t2);

I
-----
1
2
3
4
```

3.9. Profiling

3.9.1. Was ist Profiling?

Exasol verzichtet bewusst auf komplizierte Tuning-Mechanismen für den Kunden wie z.B. die manuelle Angabe von Ausführungsplänen, das Anlegen von diversen Index-Arten, das Berechnen von Statistiken uvm. Query-Anfragen werden in Exasol vom Query-Optimierer analysiert und entsprechende Tuning-Maßnahmen selbstständig und vollautomatisch durchgeführt.

Dennoch existieren Situationen, in denen es interessant sein kann, wie lange die einzelnen Teile der Query-Ausführung dauern. Langlaufende Queries können hierdurch analysiert und eventuell anders formuliert werden. Weiterhin können diese Information Exasol helfen, den Query-Optimierer noch weiter zu verbessern.

Das Konzept der Exasol sieht vor, dass bei Bedarf das Profiling aktiviert werden kann und anschließend die entsprechenden Informationen gesammelt werden. Anschließend werden diese Daten dem Kunden in Systemtabellen zur Verfügung gestellt. Weitere Details hierzu finden Sie in den nächsten Abschnitten.

3.9.2. Aktivierung und Auswertung

Defaultmäßig ist das Profiling nur für die laufenden Anfragen aktiviert und kann über die Systemtabellen [EXA_DB_A_PROFILE_RUNNING](#) und [EXA_USER_PROFILE_RUNNING](#) ausgelesen werden.

Mittels [ALTER SESSION](#) bzw. [ALTER SYSTEM](#) kann durch den Parameter PROFILE das generelle Profiling eingeschalten werden. Ist dies der Fall, werden während der Ausführung von Queries die entsprechenden Profiling-Daten gesammelt und in den Systemtabellen [EXA_USER_PROFILE_LAST_DAY](#) und [EXA_DB_A_PROFILE_LAST_DAY](#) bereitgestellt. Bitte beachten Sie jedoch, dass diese Profiling-Daten Teil der Statistik-Tabellen sind, welche nur periodisch im System geschrieben (COMMIT) werden. Dadurch entsteht eine gewisse Verzögerung, bis die Daten tatsächlich in den Systemtabellen angezeigt werden. Falls Sie die Profiling-Daten direkt nach der Ausführung analysieren wollen, empfiehlt sich der Befehl [FLUSH STATISTICS](#), mit dem sich das COMMIT der statistischen Information erzwingen lässt.

In den Profiling-Systemtabellen werden zu jedem Statement einer Session diverse Informationen zu den verschiedenen Ausführungsschritten (PART) der Anfrage aufgelistet, z.B. die Ausführungsdauer (DURATION), die verbrauchte CPU-Zeit (CPU), der Speicherverbrauch (MEM_PEAK bzw. TEMP_DB_RAM_PEAK) oder die Netzwerk-kommunikation (NET). Zusätzlich werden die in dem Ausführungsschritt zu verarbeitende Zeilenzahl (OBJECT_ROWS) sowie die Ergebniszeilen des Schrittes (OUT_ROWS) sowie weitere Infos (PART_INFO) gesammelt. Bitte beachten Sie, dass bestimmte Informationen nicht für alle Ausführungsschritte Werte beinhalten.

Folgende Ausführungsschritte sind möglich:

COMPILE / EXECUTE	Kompilieren und Ausführen des Statements (inkl. Query-Optimierung und evtl. automatisches Berechnen von Tabellen-Statistiken)
SCAN	Scan einer Tabelle
JOIN	Join mit einer Tabelle
FULL JOIN	Full Outer Join mit einer Tabelle
OUTER JOIN	Outer Join mit einer Tabelle
EXISTS	EXISTS-Berechnung
GROUP BY	Berechnung der GROUP BY Aggregation
GROUPING SETS	Berechnung der GROUPING SETS Aggregation
SORT	Sortieren von Daten (bei ORDER BY, aber auch in analytischen Funktionen)
ANALYTICAL FUNCTION	Berechnung analytischer Funktionen (exklusive der Sortierung von Daten)
CONNECT BY	Berechnung von hierarchischen Queries
PREFERENCE PROCESSING	Berechnung von Skyline-Anfragen (Details siehe Abschnitt 3.10, Skyline)
PUSHDOWN	Vom Adapter generiertes Pushdown SQL Statement für eine Anfrage an ein virtuelles Objekt
QUERY CACHE RESULT	Auslesen des Ergebnisses aus dem Query Cache

CREATE UNION	Unter bestimmten Umständen kann der Optimizer aus mehreren einfachen UNION ALL Verknüpfungen eine Art zusammengesetzte Tabelle erzeugen und sie somit schneller verarbeiten
UNION TABLE	Dieser Eintrag wird pro hinzugefügter zusammengesetzter UNION ALL Tabelle erzeugt (siehe auch "CREATE UNION")
INSERT	Einfügen von Daten (bei INSERT, MERGE oder IMPORT)
UPDATE	Update von Daten (bei UPDATE oder MERGE)
DELETE	Löschen von Daten (bei DELETE, TRUNCATE oder MERGE)
IMPORT	Ausführung des IMPORT-Befehls
EXPORT	Ausführung des EXPORT-Befehls
COMMIT	Commit der Transaktion, persistentes Schreiben auf Festplatte
ROLLBACK	Zurückrollen der Transaktion
WAIT FOR COMMIT	Warten auf das Beenden einer anderen Transaktion
INDEX CREATE	Erzeugen von internen Indices
INDEX INSERT	Insert auf internen Indices
INDEX UPDATE	Update auf internen Indices
INDEX REBUILD	Neuerzeugung von internen Indices
CONSTRAINT CHECK	Überprüfung von Constraint-Eigenschaften (Primary/Foreign Key, NOT NULL)
DISTRIBUTE	Verteilung von Daten über die Cluster-Knoten hinweg
RECOMPRESS	Rekomprimieren von Daten
REPLICATE	Cluster-weites Replizieren von Daten (z.B. Replikation kleiner Tabellen, um globale Joins zu vermeiden)
SYSTEM TABLE SNAPSHOT	Sammeln der Daten für eine Systemtabelle

3.9.3. Beispiel

Im Folgenden finden Sie ein Beispiel, wie die Profiling Information zu einer einfachen SELECT-Anfrage aussehen:

```
-- omit autocommits and switch on profiling
SET AUTOCOMMIT OFF;
ALTER SESSION SET profile='ON';

-- run query
SELECT YEAR(o_orderdate) AS "YEAR", COUNT(*)
FROM orders
GROUP BY YEAR(o_orderdate)
ORDER BY 1 LIMIT 5;

YEAR  COUNT(*)
-----
1992      227089
1993      226645
1994      227597
1995      228637
1996      228626

-- switch off profiling again and avoid transaction conflicts
ALTER SESSION SET profile='OFF';
COMMIT;
-- provide newest information and open new transaction
FLUSH STATISTICS;
COMMIT;

-- read profiling information
SELECT part_id, part_name, object_name, object_rows, out_rows, duration
FROM exa_user_profile_last_day
```

```
WHERE CURRENT_STATEMENT-5 = stmt_id AND CURRENT_SESSION=session_id;
```

PART_ID	PART_NAME	OBJECT_NAME	OBJECT_ROWS	OUT_ROWS	DURATION
1	COMPILE / EXECUTE				0.020672
2	SCAN	ORDERS	1500000	1500000	0.247681
3	GROUP BY	tmp_subselect0	0	7	0.020913
4	SORT	tmp_subselect0	7	5	0.006341

3.10. Skyline

3.10.1. Motivation

Sollen in einer Datenbank optimale Ergebnisse für eine bestimmte Fragestellung gefunden werden, so ergeben sich bei großen Datenmengen und vielen Dimensionen meist folgende schwerwiegende Probleme:

1. Datenüberflutung

Große Datenmengen können manuell kaum analysiert werden. Bei der Navigation durch Millionen von Datensätzen wird meist nur nach einer Dimension sortiert und die Top-N Ergebnisse betrachtet. Dadurch wird das Optimum fast immer verfehlt.

2. Leere Ergebnisse

Um das Problem der großen Datenmengen zu vereinfachen, werden oftmals Filter definiert. Sehr häufig sind diese jedoch zu restriktiv, wodurch die Ergebnismenge komplett leer bleibt. Durch iterative Justierung der Filter muss dann versucht werden, eine beherrschbare Datenmenge zu extrahieren. Auch hier wird das eigentliche Optimum in der Regel stets unberücksichtigt bleiben.

3. Korrelation vieler Dimensionen schwierig

Sind für Daten viele Dimensionen relevant, so kann in SQL kaum ein Optimum gefunden werden. Durch Metriken wird oft versucht, eine halbwegs gute Heuristik zu ermitteln, um die verschiedenen Attribute unterschiedlich zu gewichten. Allerdings wird durch die Simplifizierung auf eine einzelne Kennzahl sehr viel Information und Korrelation zwischen den Daten eliminiert. Auch hierdurch wird das eigentliche Optimum fast immer verfehlt.

Insgesamt navigieren Analysten oft iterativ in den großen Datenmengen und benutzen dabei diverse Filter, Aggregationen und Metriken. Diese sind einerseits zeitaufwändig und liefern andererseits nur schwer nachvollziehbare Ergebnisse, deren Relevanz oft fragwürdig bleibt. Anstatt eines Optimums für eine Fragestellung entsteht so lediglich ein Kompromiss, der die komplexe Korrelation zwischen den Dimensionen vernachlässigt.

3.10.2. So funktioniert Skyline

Skyline ist Exasol's eigens entwickelte SQL-Spracherweiterung, die die oben beschriebenen Probleme der Daten-Analyse löst. Mit Skyline erhalten Nutzer eine intuitive Schnittstelle, um optimale Ergebnisse nach einer Vielzahl von Dimensionen zu extrahieren (Pareto-Optimierung).

Anstelle harter Filter über die WHERE-Klausel sowie Metriken zwischen den einzelnen Spalten werden die relevanten Dimensionen in der PREFERRING-Klausel angegeben, und Exasol wird die nach diesen Attributen tatsächlich optimalen Ergebnisse ermitteln. Optimal bedeutet die Menge der nicht-dominierten Punkte im mehrdimensionalen Suchraum, auch Pareto-Menge genannt. Ein Punkt wird laut Definition von einem anderen dominiert, wenn er in allen Dimensionen schlechter ist.

Stellen Sie zur Veranschauung einen Optimierungsraum mit lediglich zwei Dimensionen vor, zum Beispiel die Entscheidung für ein Auto mit den beiden Attributen "hohe Leistung (PS)" und "niedriger Preis". Ein Auto A wird dann von einem anderen Auto B dominiert, sofern dessen Preis niedriger und gleichzeitig die Leistung höher ist. Ist nur eine Dimension besser, die andere aber schlechter, so kann zwischen den beiden Autos keine klare Entscheidung getroffen werden.

Insgesamt bedeutet die Pareto-Menge in diesem Fall also die Menge aller Autos mit möglichst hoher PS-Zahl bei gleichzeitig minimalem Preis. Ohne Skyline können Sie lediglich die PS-Zahl und den Preis in einer Metrik verknüpfen, oder die Datenmenge durch einen bestimmten Preis- oder Leistungs-Bereich eingrenzen. Die tatsächlich optimalen Autos werden Sie dadurch aber nicht identifizieren können.

Mit Skyline hingegen geben Sie einfach diese beiden Dimensionen in der PREFERRING-Klausel an (PREFERRING HIGH power PLUS LOW price), und Exasol wird Ihnen die optimalen Kombinationen liefern. Dabei sollten Sie sich vergegenwärtigen, dass bei dem Skyline-Algorithmus alle Zeilen einer Tabelle mit allen anderen verglichen werden. Im Falle einer einfachen Metrik für ein mehrdimensionales Optimierungsproblem wird das Ergebnis zwar sehr einfach berechnet werden können (Sortierung nach einer Zahl), allerdings wird die Komplexität der Korrelationen dabei komplett vernichtet. Skyline gibt Ihnen also die Möglichkeit, die Struktur der Daten wirklich berücksichtigen zu können, und dabei die wirklich optimale Ergebnismenge zu extrahieren.

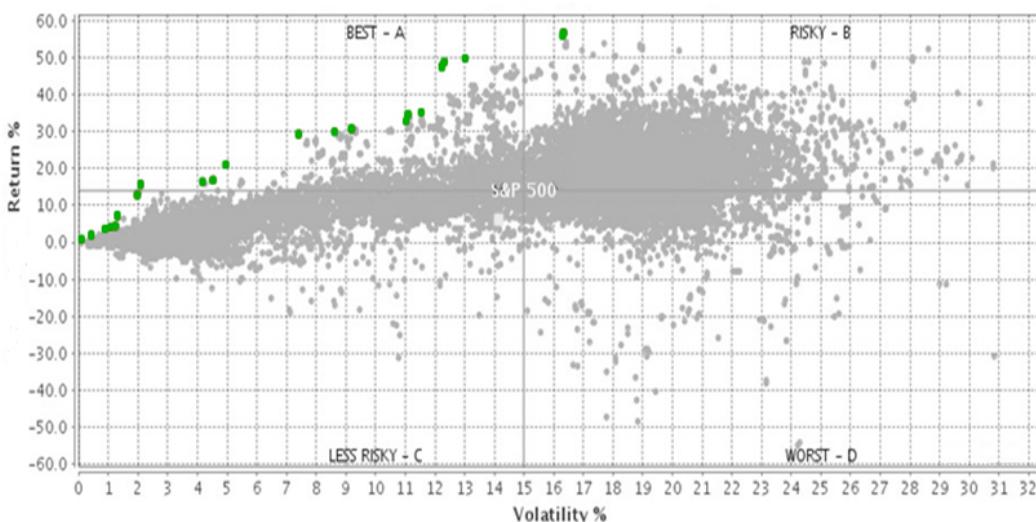
Die Vorteile von Skyline sind Ihnen hoffentlich bereits im sehr einfach gewählten zweidimensionalen Beispiel klar geworden. Der Algorithmus kann aber ebenso eine große Anzahl von Dimensionen verarbeiten, wenn das menschliche Gehirn sich eine optimale Menge nur noch sehr schwer vorstellen kann.

Übrigens kann die PREFERRING-Klausel in den Befehlen `SELECT`, `DELETE` und `UPDATE` benutzt werden.

3.10.3. Beispiel

Als Beispiel zur Veranschaulichung der Mächtigkeit von Skyline wollen wir die Auswahl der besten Fonds optimieren, vor die Finanz-Anleger immer wieder gestellt sind. Bei der Selektion von guten Fonds gibt es eine Vielzahl an Eigenschaften, zum Beispiel Performance, Volatilität, Ausgabegebühr, Rating von Agenturen, jährliche Kosten, usw.

Wir wollen uns aus Vereinfachungsgründen nur auf die ersten beiden Eigenschaften konzentrieren. In der Realität sind die beiden Eigenschaften Performance und Volatilität übrigens oftmals gegenläufig. Je konservativer ein Fonds ist, desto geringer dessen Volatilität, aber desto geringer in der Regel auch seine Performance.



```
SELECT * FROM funds PREFERRING HIGH performance PLUS LOW volatility;
```

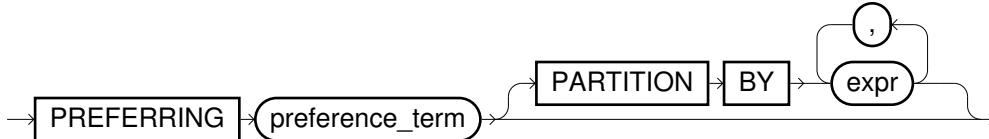
Im obigen Bild sehen Sie eine Grafik mit vielen tausend Fonds mit deren Performance und Volatilität. Die grünen Punkte sind das Ergebnis der im Beispiel angegebenen Skyline-Query. Diese Fonds liefern die optimale Teilmenge hinsichtlich der beiden angegebenen Dimensionen. Ob letztlich ein konservativer oder riskanter Fonds ausgewählt wird, kann in einem zweiten, nachgelagerten und subjektiven Verfahren entschieden werden.

Aber Sie erkennen bereits an diesem einfachen Beispiel, wie sehr sich die Problemstellung vereinfachen lässt. Aus vielen tausend Fonds werden nur noch die 23 objektiv besten extrahiert, und diese Teilmenge ist hinsichtlich der Fragestellung die tatsächlich optimale Ergebnismenge.

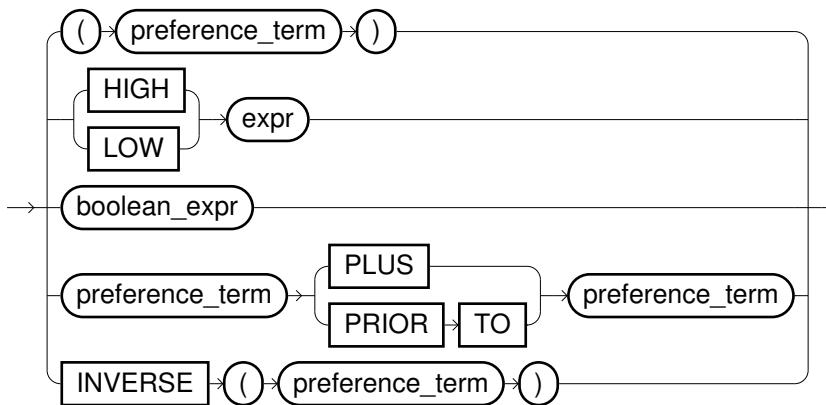
3.10.4. Syntaxelemente

Wie in der Beschreibung des Statements `SELECT` beschrieben wird die PREFERRING-Klausel nach der WHERE-Bedingung definiert und kann folgende Elemente enthalten:

`preferring_clause ::=`



`preference_term ::=`



`PARTITION BY`

Wird diese Option gewählt, so werden die Präferenzen für jede Partition separat berechnet.

`HIGH bzw. LOW`

Hiermit wird definiert, ob der entsprechende Ausdruck möglichst hohe oder niedrige Werte haben soll. Bitte beachten Sie, dass hierfür numerische Ausdrücke erwartet werden.

`Boolesche Ausdrücke`

Im Falle von booleschen Ausdrücken werden die Elemente bevorzugt, welche für die Bedingung TRUE ergeben. Der Ausdruck `x > 0` ist also äquivalent zu `HIGH (x > 0)`. Der letztere Ausdruck würde dabei implizit zu der numerischen Zahl 0 bzw. 1 umgewandelt werden.

`PLUS`

Über das Schlüsselwort `PLUS` können wie im Beispiel oben mehrere gleichwertige Ausdrücke als Liste von Präferenz-Ausdrücken angegeben werden.

`PRIOR TO`

Mit dieser Klausel können zwei Ausdrücke hierarchisch geschachtelt werden. Der zweite Term wird dabei nur dann berücksichtigt, falls zwei Elemente für den ersten Term den gleichen Wert aufweisen.

`INVERSE`

Mittels `INVERSE` kann das Gegenteil des Präferenz-Ausdrucks berechnet werden. Daher ist der Ausdruck `LOW price` äquivalent zu `INVERSE(HIGH price)`.

Das nachfolgende Anfrage zeigt ein komplexeres Beispiel für eine Selektion der besten Autos mit geschachtelten Ausdrücken:

- Die Leistung wird in 10er-Schritten segmentiert
- Der Preis wird in 1000er-Schritten segmentiert
- Bei gleicher Leistung und Preis wird die Farbe "silver" bevorzugt.

```
SELECT * FROM cars
  PREFERRING (LOW ROUND(price/1000) PLUS HIGH ROUND(power/10))
    PRIOR TO (color = 'silver');
```

4. Clients und Schnittstellen



Die Clients und Treiber können im Exasol Kundenportal heruntergeladen werden (wwwexasol.com [<http://wwwexasol.com/>]).

4.1. EXAplus

EXAplus ist eine Benutzerschnittstelle, um [SQL](#)-Befehle in Exasol verarbeiten zu können. EXAplus ist in Java implementiert und steht als grafische Applikation sowie als reine Konsolenanwendung sowohl unter MS Windows als auch unter Linux zur Verfügung.

4.1.1. Installation

Windows

EXAplus wurde auf folgenden Systemen erfolgreich getestet:

- Windows 10 (x86/x64)
- Windows 7, Service Pack 1 (x86/x64)
- Windows Server 2012 R2 (x86/x64)
- Windows Server 2012 (x86/x64)
- Windows Server 2008 R2, Service Pack 1 (x86/x64)
- Windows Server 2008, Service Pack 2 (x86/x64)

Zum Installieren von EXAplus folgen Sie bitte den Anweisungen des Installations-Wizards, der sich bei der Ausführung der Installationsdatei startet.



Für die Installation von EXAplus werden Administrator-Rechte sowie das Microsoft .NET Framework 4.0 Client Profile™ benötigt.

Bitte beachten Sie folgendes:

- EXAplus benötigt für die Ausführung eine Java-Laufzeitumgebung. Zur Nutzung aller Funktionalitäten empfehlen wir mindestens Java 7 inklusive entsprechenden Updates.

Es wird weiterhin empfohlen, dass die *Unterstützung für zusätzliche Sprachen* bei Ihrer Java-Installation ausgewählt wurde. Für die korrekte Formatierung von speziellen Unicode-Zeichen (z.B. japanische Zeichen) muss auf Ihrem Windowssystem eine entsprechende Schriftart installiert sein, die die entsprechenden Zeichen darstellen kann.
- Nach erfolgreicher Installation befindet sich ein entsprechender Startmenü-Eintrag zum Starten der grafischen EXAplus Version. Sie können EXAplus aber auch als Konsolen-Version in der Eingabeaufforderung (cmd.exe) benutzen, indem Sie die Datei `exaplus64.exe` bzw. `exaplus.exe` (32-Bit Variante) aufrufen. Um auf die komplette Pfadangabe verzichten zu können, wählen Sie die Option *Add EXAplus to path* während der Installation.

Linux/Unix

EXAplus wurde auf folgenden Systemen erfolgreich getestet:

- Red Hat / CentOS 7, OpenJDK JVM 1.8.0 (x64)
- Red Hat / CentOS 6, OpenJDK JVM 1.8.0 (x86/x64)
- Debian 8, OpenJDK JVM 1.7.0 (x86/x64)

- Ubuntu 16.04 LTS, OpenJDK JVM 1.8.0 (x86/64)
- Ubuntu 14.04 LTS, OpenJDK JVM 1.7.0 (x86/64)
- SUSE Linux Enterprise Server 12, IBM's JVM 1.7.0 (x64)
- SUSE Linux Enterprise Desktop 12, OpenJDK JVM 1.7.0 (x64)
- SUSE Linux Enterprise Server 11 Service Pack 3, IBM's JVM 1.7.0 (x86/x64)
- openSUSE Leap 42.2, OpenJDK JVM 1.8.0 (x64)

- macOS Sierra, JVM 1.8.0 (64Bit)
- OS X 10.11, JVM 1.8.0 (64Bit)

- FreeBSD 11.0, OpenJDK 1.8.0 (64Bit)
- FreeBSD 10.3, OpenJDK 1.8.0 (64Bit)

Bitte beachten Sie folgendes:

- Java 7 oder höher muss installiert und das Programm „java“ muss im Pfad enthalten sein.

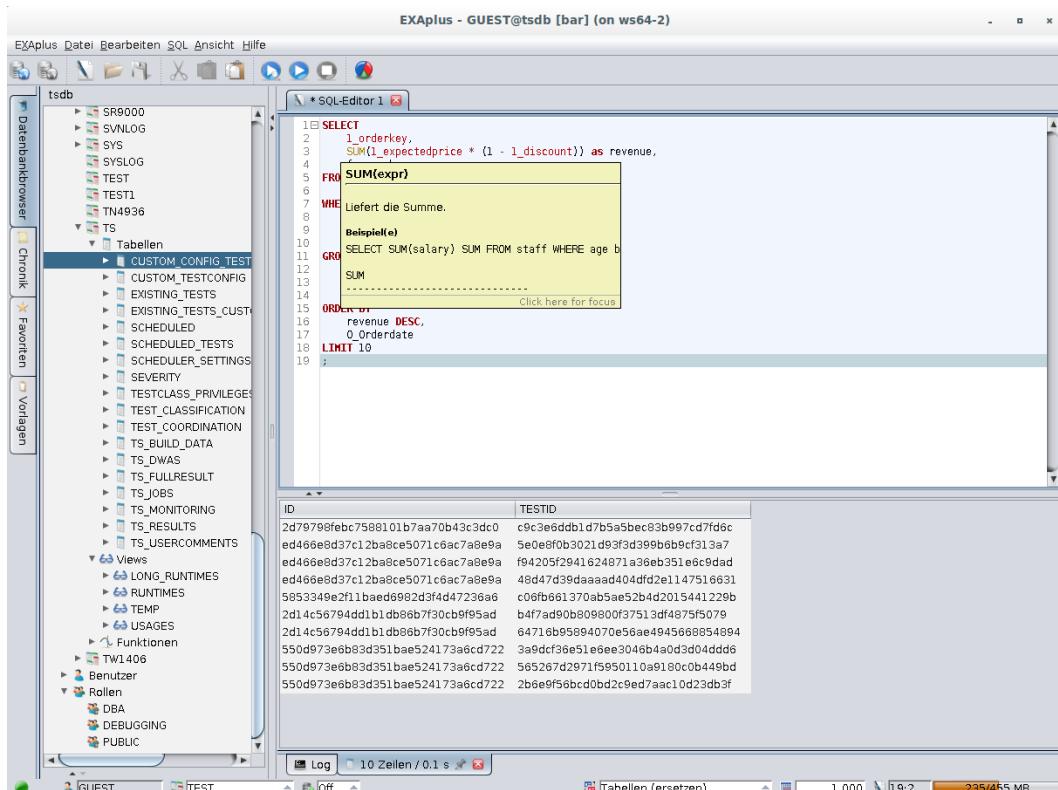
Für die korrekte Formatierung von speziellen Unicode-Zeichen (z.B. japanische Zeichen) muss auf Ihrem Linuxsystem eine entsprechende Schriftart installiert sein, die die entsprechenden Zeichen darstellen kann.

- Falls Sie japanische Zeichen verwenden möchten und Ihr Input Manager Scim ist, empfehlen wir Ihnen, mindestens die Version 1.4.7 von Scim zu installieren. Darüber hinaus sollte Scim-Anthy (japanische Zeichenunterstützung) in der Version 1.2.4 oder höher installiert sein.

Das EXAplus Installationsarchiv kann mit dem Kommando **tar -xfz** entpackt werden. Wenn die oben genannten Software-Anforderungen erfüllt sind, kann EXAplus nun durch Aufruf des Startskripts **exaplusgui** (grafische Version) bzw. **exaplus** (Kommandozeilenversion) gestartet werden. Eine gesonderte Installation ist nicht nötig.

4.1.2. Die grafische Benutzeroberfläche

Die grafische Benutzeroberfläche von EXAplus stellt sich dem Benutzer wie folgt dar:



Folgende Teilbereiche lassen sich unterscheiden:

Menübar	Ausführen verschiedenster Aktionen über das Menü.
Toolbar	Direkte Ausführung der häufigsten Aktionen über Symbole.
Datenbankbrowser	Dient der Anzeige der existierenden Datenbankobjekte wie Schemas, Tabellen, Skripte, Nutzer, usw. Durch Doppelklick auf ein Objekt werden Informationen zum Objekt im Editor-Bereich geöffnet. Im Kontextmenü der Objekte (rechter Mausklick) finden Sie eine Liste möglicher Aktionen auf diesem Objekt.
Chronik	Übersicht über die bisher ausgeführten Kommandos. Wenn Sie eine Anfrage als Favoriten speichern wollen, können Sie dies bequem über das Kontextmenü tun (rechter Mausklick auf das SQL).
Favoriten	In den Favoriten können häufig verwendete Queries gespeichert und mit einem Tastenkürzel verknüpft werden. Ihre Favoriten können Sie über das Kontextmenü exportieren und importieren (rechter Mausklick auf den Ordner).
Vorlagen	Enthält eine Liste von diversen Vorlagen (SQL Statements und Funktionen, Lua Funktionen sowie EXAplus Kommandos). Als eine Art Schnell-Referenz kann man schnell Syntax Details in Erinnerung rufen oder z.B. die Liste der verfügbaren String-Funktionen einsehen. Per Drag & Drop können diese Vorlagen auch bequem in den SQL Editor kopiert werden.
SQL Editor	Arbeitsbereich, in dem SQL Dateien angezeigt, editiert und ausgeführt werden können. Darüber hinaus werden hier Objektinformationen angezeigt.
Ergebnisanzeige	Anzeige von Query-Ergebnissen in Tabellenform einerseits sowie das Log-Protokoll von EXAplus.
Statusleiste	Zeigt die aktuellen Verbindungsinformationen und -einstellungen an. Z.T. können diese direkt in der Statusleiste verändert werden (z.B. Autocommit-Modus oder Limitierung der Ergebnisanzeige). Zudem finden Sie dort auch den aktuellen Speicherverbrauch.

Die meisten Funktionalitäten von EXAplus sind intuitiv. Allerdings sollen an dieser Stelle einige Dinge hervorgehoben werden, mit denen die tägliche Arbeit erleichtert werden kann.

Tabelle 4.1. Informationen zur Arbeit mit EXAplus

Thema	Erläuterungen
Verbindungen	<p>Beim Start von EXAplus öffnet sich ein Fenster, in dem die entsprechenden Verbindungsdaten eingegeben werden können. Diese Daten speichert EXAplus für den nächsten Start ab, so dass diese bequem wiederverwendet werden können (bis auf das Passwort). Alternativ öffnet sich dieser Verbindungsdialog über das Connect-Symbol in der Toolbar.</p> <p>Nutzt ein Nutzer jedoch mehrere Datenbanken oder möchte sich mit unterschiedlichen Nutzerprofilen an eine Datenbank verbinden, so lässt sich dies einfach über Verbindungsprofile bewerkstelligen. Diese können über den Menüeintrag <i>EXAplus -> Verbindungsprofile</i> erzeugt und editiert werden.</p> <p>Anschließend kann man diese Profile über das Connect-Symbol bzw. über den Menübareintrag <i>EXAplus -> Verbinde zu</i> auswählen. Falls einzelne Felder wie z.B. das Passwort vorher nicht abgespeichert wurden, so muss der Nutzer diese beim Verbinden vervollständigen. Sobald ein Profil angelegt wurde, öffnet sich der Dialog für benutzerspezifische Verbindungen nicht mehr.</p> <p>Um den Verbindungsprozess völlig zu automatisieren, lässt sich eine Default-Verbindung definieren. Diese nutzt EXAplus dann automatisch beim Start.</p> <p> Angelegte Verbindungen können auch in der Konsolen-Version verwendet werden (siehe Kommandozeilenparameter <i>-profile</i>)</p>
Mehrere EXAplus-Instanzen	<p>Falls sich ein Nutzer mit mehreren Datenbanken gleichzeitig verbinden oder mehrere Sessions zur gleichen DB öffnen will, so kann er mehrere Instanzen von EXAplus starten. Dies geschieht entweder über mehrmaliges Starten von EXAplus oder aber über den Menüeintrag <i>EXAplus -> Neues Fenster</i>.</p> <p> EXAplus erlaubt pro Instanz nur eine offene Verbindung zu einer Datenbank</p> <p>Ändert man die Einstellungen von EXAplus, so gelten diese für alle offenen Instanzen.</p>
Ergebnistabellen	<p>Über die Statusleiste und das Menü kann definiert werden, ob bei der Ausführung neuer Kommandos die alten Ergebnistabellen aufbewahrt oder gelöscht werden sollen oder die Ausgabe im Textformat (Log) erfolgen soll.</p> <p>Zudem kann der Nutzer in der Statusleiste die maximale Zeilenzahl definieren, die in einer Ergebnistabelle angezeigt werden soll. Durch die Limitierung wird nur die begrenzte Datenmenge von der Datenbank an den Client übertragen, was gerade bei sehr großen Ergebnismengen durchaus sinnvoll sein kann.</p> <p>Tabellen können umsortiert werden, indem man auf eine der Spalten klickt. Der Tabelleninhalt kann mittels Copy&Paste in andere Applikationen wie z.B. Excel kopiert werden.</p>
Aktuelles Schema	<p>In der Statusleiste wird das aktuell geöffnete Schema angezeigt und kann darüber auch geändert werden. Weiterhin lässt sich die Liste der im Datenbankbrowser angezeigten Schemas auf das aktuell geöffnet Schema begrenzen (über den Menüeintrag <i>Ansicht -> Nur aktuelles Schema anzeigen</i>).</p>
Autocommit-Modus	<p>Der Autocommit-Modus wird in der Statusleiste angezeigt und kann darüber auch direkt verändert werden. Folgende Werte sind möglich:</p> <ul style="list-style-type: none"> ON Nach der Ausführung jedes SQL-Statements wird ein implizites COMMIT ausgeführt (Default). OFF Ein implizites COMMIT wird niemals ausgeführt. EXIT Ein implizites COMMIT wird nur beim Beenden der Verbindung ausgeführt.

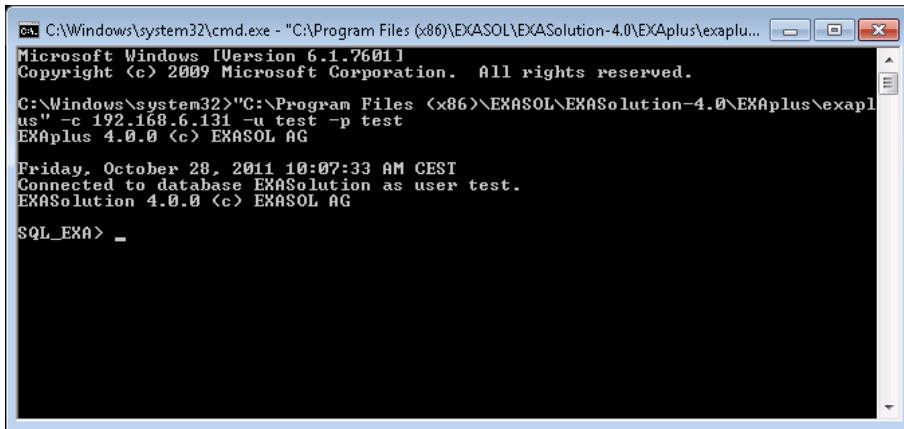
Thema	Erläuterungen
Drag & Drop	<p>Mittels Drag&Drop (Ziehen&Loslassen) können SQL-Befehle aus der Befehlshistorie direkt in den Editorbereich kopiert werden.</p> <p>Analog ist es möglich, schemaqualifizierte Namen eines Datenbankobjekts aus dem Datenbankbrowser in den Editorbereich einzufügen (z.B. MY_SCHEMA.MY_TABLE). Lediglich bei Spalten wird auf die Schemaqualifizierung verzichtet.</p>
Auto vervollständigung	<p>Über die Tastenkombination STRG+SPACE versucht EXAplus das aktuelle Wort im Editor zu vervollständigen. Hierbei werden die Metainformationen der Datenbank berücksichtigt und zum Beispiel Spaltennamen von Tabellen erkannt. Zusätzlich erhalten Sie für z.B. Funktionen weitere Informationen und Beispiele.</p> <p>Über die Tastenkombination SHIFT+STRG+SPACE wird darüber hinaus auf sehr simple Art versucht, das aktuell eingegebene Wort anhand der zuletzt eingegebenen Begriffe zu vervollständigen. Bei innerhalb oft benutzten Wörtern einer Anfrage (z.B. ein Schema-Name) kann dies Ihre Arbeit erleichtern.</p>
Editor Modus	<p>Über das Kontextmenü (Rechts-Klick an beliebiger Stelle im SQL-Editor) kann zwischen den verschiedenen Modi gewechselt werden (SQL, LUA, R, PYTHON). Dies beeinflusst die Auto vervollständigung, die Syntaxhervorhebung sowie die Code Faltung.</p>
Lesezeichen	<p>Sie können Lesezeichen erzeugen, indem Sie im SQL Editor auf die graue Leiste links neben der Zeilennummern klicken bzw. die Tastenkombination STRG+F2 benutzen. Mittels F2 bzw. UMSCHALT+F2 können Sie anschließend zwischen den Lesezeichen springen, was gerade bei großen Dateien sehr hilfreich sein kann.</p>
Parametrisiertes SQL-Skripte	<p>SQL-Skripte können mit Hilfe von Variablen parametrisiert werden, welche bei Bedarf sogar dynamisch bei der Ausführung eingegeben werden können. Details finden Sie in Abschnitt 4.1.4, EXAplus-spezifische Befehle.</p>
Objektinfos	<p>Durch Doppelklick oder über den Kontextmenüeintrag <i>Info</i> von Objekten öffnet sich im Editorbereich die Objektinfo mit diversen Informationen. Hierzu gehören u.a.:</p> <ul style="list-style-type: none"> • Schemas (mit der Liste der enthaltenen Schemaobjekte) • Tabellen (mit der Liste der Spalten sowie Constraints) • Views und Skripte (mit dem Erzeugungstext) • Nutzer und Rollen (mit deren Liste der Systemprivilegien) • Ergebnistabellen (mit Infos zur Ausführung) • SQL-Befehle aus der Befehlshistorie (mit Infos zur Ausführung) <p>Im oberen Bereich der Objektinfo befinden sich einige Icons mit nützlichen Funktionen. Z.B. kann man für ein Schemaobjekt automatisch das entsprechende DDL generieren lassen (auch für ganze Schemas möglich), das Objekt löschen, eine Tabelle bzw. View exportieren oder ein Skript oder eine View editieren.</p> <p>Diese Aktionen lassen sich auch im Kontextmenü zu den Objekten im Datenbankbrowser finden.</p>
System-Monitoring	<p>Wenn Sie das Kuchendiagramm-Symbol auf der Werkzeugleiste anklicken, öffnet sich ein separates Fenster mit visuellen Nutzungs-Statistiken der verbundenen Datenbank. Sie können zusätzliche Graphen hinzufügen, die Anordnung der Graphen definieren sowie das Interval und die Datengruppe justieren. Die Graphen werden ungefähr alle 3 Minuten aktualisiert.</p>
Spracheinstellung	<p>Die verwendete Sprache lässt sich unter <i>EXAplus -> Einstellungen</i> ändern.</p>
Verhalten bei Fehlern	<p>Unter dem Menüeintrag <i>SQL -> Fehlerbehandlung</i> lässt sich das Verhalten bei Fehlern definieren. Weitere Details hierzu siehe Befehl WHENEVER.</p>
Speicher-Verbrauch	<p>Der Balken in der rechten unteren Ecke zeigt den Speicher-Allokation des aktuellen EXAplus-Prozesses. Durch ein Doppelklick wird das Programm die Garbage Collection starten.</p>

Thema	Erläuterungen
Verknüpfung von SQL-Dateien mit EXAplus	Falls Sie diese Option während der Installation auswählen, können unter Windows SQL Dateien per Doppelklick im EXAplus geöffnet werden. Als Encoding wird dann die letzte Einstellung verwendet, mit der Dateien geöffnet bzw. gespeichert wurden.
Migration von EXAplus-Einstellungen	<p>Wollen Sie die Einstellungen z.B. auf einen anderen Computer übernehmen, so kopieren Sie einfach die entsprechenden xml-Dateien aus dem EXAplus-Ordner (exasol bzw. .exasol) in Ihrem Heimverzeichnis. Hier finden Sie die Favoriten (favorites.xml), die Chronik (history*.xml), die Verbindungsprofile (profiles.xml) sowie die EXAplus-Einstellungen (exaplus.xml).</p> <p> Die xml-Dateien sollten nur für EXAplus Instanzen mit der selben Version kopiert werden.</p>

 Bitte beachten Sie, dass EXAplus im Hintergrund stets eine zweite Verbindung zur Datenbank öffnet, in der Metadaten für z.B. den Datenbankbrowser oder die Auto vervollständigung abgefragt werden. Daher sollten Sie beachten, dass Datenbankänderungen erst nach einem Commit in der Hauptverbindung im Datenbankbrowser sichtbar werden.

4.1.3. Der Konsolenmodus

Neben der grafischen Applikation steht EXAplus auch als reine Konsolenanwendung zur Verfügung, welche gerade zur Einbindung in Skriptsprachen sehr nützlich ist. Gestartet wird die Konsolenversion durch den Aufruf von **exaplus64[.exe]** bzw. **exaplus[.exe]** (32-Bit Variante).



```

ex: C:\Windows\system32\cmd.exe - "C:\Program Files (x86)\EXASOL\EXASolution-4.0\EXAplus\exaplus" -c 192.168.6.131 -u test -p test
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>"C:\Program Files (x86)\EXASOL\EXASolution-4.0\EXAplus\exaplus" -c 192.168.6.131 -u test -p test
EXAplus 4.0.0 <c> EXASOL AG

Friday, October 28, 2011 10:07:33 AM CEST
Connected to database EXASolution as user test.
EXASolution 4.0.0 <c> EXASOL AG

SQL_EXA> _

```

Dem Nutzer öffnet sich nach erfolgreichem Verbinden zur Datenbank eine interaktive Session, in der er SQL- (siehe [Kapitel 2, SQL-Referenz](#)) bzw. EXAplus-Befehle (siehe [Abschnitt 4.1.4, EXAplus-spezifische Befehle](#)) zur Datenbank schicken kann. Ergebnistabellen werden hier in textueller Form angezeigt.

Anmerkungen:

- Beenden kann man EXAplus mit den Befehlen [EXIT und QUIT](#)
- Alle Befehle müssen mit einem Semikolon beendet werden (bis auf View-, Funktions- und Skript-Definitionen, die mit einem / in einer Zeile beendet werden, sowie [EXIT und QUIT](#))
- Der Befehlsverlauf ist über die Nutzung der Pfeiltasten verfügbar
- Die Eingabe mehrzeiliger Befehle kann durch die Tastenkombination [\$<STRG>-T\$](#) abgebrochen werden
- Durch Drücken der Taste [\$<TAB>\$](#) versucht EXAplus, die Eingabe zu vervollständigen

Tabelle 4.2. Kommandozeilen-Parameter von EXAplus (nur für den Konsolenmodus)

Parameter	Beschreibung
Hilfsoptionen	
<code>-help</code>	Zeigt eine Kurzübersicht über die Kommandozeilenparameter an.
<code>-version</code>	Zeigt Versionsinformationen an.
Verbindungsoptionen	
<code>-c <connection string></code>	Verbindungsdaten, bestehend aus den Clusterknoten und einem Port (z.B. 192.168.6.11..14:8563).
<code>-u <user></code>	Benutzername zur Anmeldung an Exasol.
<code>-p <passwd></code>	Passwort zur Anmeldung an Exasol.
<code>-s <schema></code>	EXAplus wird versuchen, das angegebene Schema nach der Anmeldung an Exasol zu öffnen.
<code>-L</code>	Es wird nur ein Login-Versuch unternommen. Wenn Benutzername und Passwort auf der Kommandozeile angegeben worden sind, wird auch bei falschem Passwort nicht nachgefragt. Dieser Parameter ist für die Verwendung in Skripten sehr nützlich.
<code>-encryption <ON/OFF></code>	Schaltet die Client/Server Kommunikation ein bzw. aus. Default: ON
<code>-profile <profile name></code>	Name des Verbindungsprofils aus <code><configDir>/profiles.xml</code> . Profile können in der GUI editiert werden oder über die Verwaltungs-Parameter verwaltet werden (siehe unten).
<code>-k</code>	Nutzung der Kerberos-basierten Authentifizierung.
Verbindungsprofil-Verwaltungs-Optionen	
<code>-lp</code>	Listet alle existierenden Profile auf und beendet sich.
<code>-dp <profile name></code>	Löscht das spezifizierte Profil und beendet sich.
<code>-wp <profile name></code>	Speichert ein spezifiziertes Profil und beendet sich. Hierbei werden die Verbindungsoptionen abgespeichert.
Datei-Optionen	
SQL-Dateien können mit den Parametern <code>-f</code> oder <code><-B></code> übergeben werden oder alternativ über die EXAplus-Befehle start , @ und @@ ausgeführt werden (siehe auch Abschnitt 4.1.4, EXAplus-spezifische Befehle). Gesucht werden diese Dateien zunächst relativ zum Arbeitsverzeichnis, aber auch in allen in der Umgebungsvariablen <code>SQLPATH</code> definierten Verzeichnissen. Hierfür wird jeweils bei Bedarf auch die Endung <code>.sql</code> und <code>.SQL</code> angehängt, falls die Dateien nicht gefunden werden. Die Variable <code>SQLPATH</code> enthält ähnlich wie die Variable <code>PATH</code> eine Liste von Verzeichnissen, die durch <code>:</code> unter Linux und durch <code>;</code> unter Windows getrennt werden.	
<code>-init <datei></code>	Datei, die beim Starten initial ausgeführt wird. Default: <code>exaplus.sql</code>
<code>-f <datei></code>	Datei, die EXAplus ausführen und sich dann beenden soll.
<code>-B <datei></code>	Datei, die EXAplus im „Batch“-Modus ausführen und sich dann beenden soll.
<code>-encoding <encoding></code>	Setzt den Zeichensatz für das Lesen eines SQL-Skriptes über <code>-f</code> bzw <code>-B</code> . Die unterstützten Zeichensätze sind unter Anhang D, Unterstützte Zeichensätze aufgelistet. Standardmäßig erwartet EXAplus, dass SQL-Dateien mit dem Zeichensatz UTF8 kodiert sind. Mit dem Kommando SET ENCODING kann der Zeichensatz innerhalb einer Sitzung gewechselt werden. Dabei ist zu beachten, dass sich die Änderung nicht auf bereits geöffnete Dateien auswirkt.

Parameter	Beschreibung
-- <args>	<p>SQL-Dateien können Argumente über die Variablen &1, &2 usw. auswerten, die mittels dem Parameter “--” übergeben werden. Z.B. kann die Datei <code>test.sql</code> mit dem Inhalt</p> <pre>--test.sql SELECT * FROM &1;</pre> <p>folgendermaßen aufgerufen werden:</p> <p>exaplus -f test.sql -- dual</p>
Reconnect-Optionen	
<code>-recoverConnection <ON/OFF></code>	Soll nach einem Verbindungsverlust versucht werden, die Verbindung wiederherzustellen. Default: OFF
<code>-retry <num></code>	Bestimmt die Anzahl der Versuche zum Wiederherstellen einer Verbindung. Bei Angabe von -1 oder UNLIMITED wird unbegrenzt weiter versucht, die Verbindung wieder herzustellen. Default: UNLIMITED
<code>-retryDelay <num></code>	Mindestintervall zwischen zwei Wiederherstellungsversuchen in Sekunden. Default: 5
<code>-closeOnConnectionLost</code>	Beendet EXAplus nach Verlust einer Verbindung, die nicht wiederhergestellt werden konnte. Default: ON
Sonstige Optionen	
<code>-autocommit <ON/OFF/EXIT></code>	Stellt den Autocommit-Modus ein. ON schaltet Autocommit ein, OFF aus. EXIT bewirkt einen Autocommit beim Programmende oder Disconnect. Default: ON
<code>-lang <EN/DE/JA></code>	Definiert die Sprache für EXAplus-Meldungen. Default: Abhängig von den Systemeinstellungen.
<code>-characterwidth <HALF/FULL></code>	Definiert die Breite von Unicode-Spalten bei der Ergebnisanzeige. Default: lang=EN bzw. DE HALF FULL
<code>-q</code>	Quiet Modus, der zusätzliche Ausgaben von EXAplus unterdrückt.
<code>-x</code>	Beendet EXAplus wenn Fehler auftreten.
<code>-F <num></code>	Setzt die Fetchsize in kB. Diese bestimmt, wie viele Daten des Resultsets in einem Kommunikationsschritt mit der Datenbank verschickt werden. Default: 2000
<code>-Q <num></code>	Query Timeout in Sekunden. Query werden abgebrochen, falls das Timeout überschritten wird. Default: -1 (unbegrenzt)

Parameter	Beschreibung
<code>-autoCompletion <ON/OFF></code>	Wenn diese Funktion eingeschaltet ist, kann der Benutzer Vorschläge durch Drücken von <TAB> anfragen. Beim Verwenden von Skripten wird diese Eingabehilfe automatisch ausgeschaltet. Default: ON
<code>-pipe</code>	Diesen Parameter können Sie verwenden, wenn Sie unter Linux/Unix die Eingabeumleitung (Pipe) verwenden wollen (z.B. echo "SELECT * FROM dual;" exaplus -pipe -c ... -u ... -p ...). Default: OFF
<code>-sql <SQL statement></code>	Hiermit kann ein einzelner SQL-Befehl ausgeführt werden. EXAplus wird beendet sich anschließend. Beispiel: <code>-sql "SELECT \"DUMMY\" FROM dual;"</code>
<code>-jdbcparam <JDBC parameter></code>	Setzen zusätzlicher JDBC Parameter.

4.1.4. EXAplus-spezifische Befehle

Überblick

Befehl	Funktion
Datei- und Betriebssystem-Befehle	
<code>@ und START</code>	Lädt eine Textdatei und führt die darin enthaltenen Befehle aus.
<code>@@</code>	Lädt eine Textdatei und führt die darin enthaltenen Befehle aus. Falls dieser Befehl allerdings in einem SQL-Skript benutzt wird, so beginnt der Suchpfad in dem Ordner, in dem das SQL-Skript liegt.
<code>HOST</code>	Führt ein Betriebssystem-Kommando aus und kehrt dann zu EXAplus zurück.
<code>SET SPOOL ROW SEPARATOR</code>	Definiert das Zeilenumbruchzeichen für den <code>SPOOL</code> Befehl.
<code>SPOOL</code>	Speichert die Ein- und Ausgabe in EXAplus in einer Datei.
Steuerung von EXAplus	
<code>BATCH</code>	Schaltet den Batch-Modus ein oder aus.
<code>CONNECT</code>	Stellt eine neue Verbindung zu Exasol her.
<code>DISCONNECT</code>	Trennt die aktuelle Verbindung zur Datenbank.
<code>EXIT und QUIT</code>	Schließt alle offenen Verbindungen und beendet EXAplus.
<code>PAUSE</code>	Gibt Text in der Konsole aus und wartet auf Return.
<code>PROMPT</code>	Gibt Text in der Konsole aus.
<code>SET AUTOCOMMIT</code>	Steuert, ob Exasol automatisch COMMIT-Befehle ausführen soll.
<code>SET AUTOCOMPLETION</code>	Schaltet Auto-Completion ein oder aus.
<code>SHOW</code>	Zeigt die EXAplus-Einstellungen an.
<code>TIMING</code>	Steuert die eingebauten Timer.
<code>WHENEVER</code>	Definiert das Verhalten von EXAplus bei Fehlern.
Formatierungen	
<code>COLUMN</code>	Zeigt die Formatierungseinstellungen an oder ändert sie.
<code>SET COLSEPARATOR</code>	Setzt die Zeichenkette, die zwei Spalten voneinander trennt.

Befehl	Funktion
SET ENCODING	Einstellen des Zeichensatz für verschiedene Dateien. Gibt aktuelle Einstellungen aus.
SET ESCAPE	Setzt das Escape-Zeichen, mit dem spezielle Zeichen eingegeben werden können.
SET FEEDBACK	Steuert, wann Bestätigungen ausgegeben werden.
SET HEADING	Schaltet die Ausgabe von Spaltenüberschriften ein oder aus.
SET LINESIZE	Setzt die Breite der Ausgabezeilen.
SET NULL	Definiert die Zeichenkette, mit der Nullwerte in Tabellen dargestellt werden.
SET NUMFORMAT	Setzt die Formatierung von numerischen Spalten in Tabellen.
SET PAGESIZE	Stellt ein, nach wie vielen Textzeilen die Spaltenüberschriften wiederholt werden sollen.
SET TIME	Schaltet die Ausgabe der aktuellen Uhrzeit des Clientsystems im Eingabeprompt ein oder aus.
SET TIMING	Schaltet die Anzeige der für die Ausführung eines SQL-Kommandos benötigten Zeit ein oder aus.
SET TRUNCATE HEADING	Definiert, ob Spaltenüberschriften abgeschnitten werden oder nicht.
SET VERBOSE	Schaltet zusätzliche Programm-Informationen ein oder aus.

Befehle für die Handhabung von Variablen

Der Benutzer kann beliebig viele Variablen definieren, die für die Dauer der EXAplus-Session gültig bleiben, bzw. bis sie durch ein entsprechendes Kommando explizit gelöscht werden. Auf den Wert einer Variablen kann in allen Befehlen durch &variable zugegriffen werden. Variablen werden immer als Zeichenketten behandelt.

ACCEPT	Liest Eingaben vom Benutzer ein.
DEFINE	Weist einer Variablen einen Wert zu.
SET DEFINE	Setzt das Zeichen, mit dem User-Variablen eingeleitet werden.
UNDEFINE	Löscht eine Variable.

Alphabetische Liste der EXAplus-Befehle

@ und START

Syntax

```
@ <datei> [args];
```

Beschreibung

Lädt die Textdatei <datei> und führt die darin enthaltenen Befehle aus. @ und START sind synonym.

Wenn kein absoluter Pfad angegeben wurde, so wird relativ zum Arbeitsverzeichnis von EXAplus nach der Datei gesucht. Es ist möglich, anstatt lokaler Pfade auch http- und ftp-URLs zu verwenden. Kann die Datei im ersten Versuch nicht geöffnet werden, so wird die Endung .sql an den Namen angehängt und die Datei noch einmal gesucht.

Einem Skript kann über die Kommandozeilenparameter (Konsolen Modus) eine beliebige Anzahl von Argumenten übergeben werden, die aus dem Skript heraus als &1, &2 ... angesprochen werden können. Die Variable &0 enthält den Namen des Skripts. Diese Variablen sind nur während der Laufzeit des Skripts definiert.

Beispiel(e)

```
@test1.sql 2008 5;  
@ftp://frank:swordfish@ftp.scripts/test2.sql;  
@http://192.168.0.1/test3.sql;
```

@@

Syntax

```
@@ <datei> [args];
```

Beschreibung

Wie @ oder START, allerdings wird die Datei in dem Pfad gesucht, in dem sich das aufrufende Skript befindet (falls der Aufruf durch ein Skript erfolgt, sonst wird das Arbeitsverzeichnis von EXAplus verwendet).

Beispiel(e)

```
--Inhalt von /home/mb/test1.sql:  
@@test2;  
  
--Es wird die Datei /home/mb/test2.sql geladen  
SQL_EXA> @/home/mb/test1.sql;
```

ACCEPT

Syntax

```
ACCEPT <variable> [PROMPT <text>] [DEFAULT <text>];
```

Beschreibung

Nimmt den Wert der Variablen <variable> vom Benutzer als Tastatureingabe entgegen. Wird der Parameter *prompt* angegeben, so wird zuvor der spezifizierte Text <text> ausgegeben. Der durch den Parameter <default> übergebene Wert wird als Vorgabe genommen, falls der Benutzer einfach die Return-Taste drückt.

Beispiel(e)

```
SQL_EXA> accept ok prompt "Alles in Ordnung? " default "vielleicht";  
Alles in Ordnung? ja  
SQL_EXA>
```

BATCH

Syntax

```
BATCH BEGIN | END | CLEAR ;
```

Beschreibung

Schaltet den Batch-Modus ein oder aus. In diesem Modus werden SQL-Kommandos nicht sofort ausgeführt, sondern gebündelt, wodurch die Ausführungsgeschwindigkeit erhöht werden kann.

BEGIN Wechselt in den Batch-Modus. Alle von nun an eingegebenen SQL-Kommandos werden gespeichert und erst bei Eingabe von `BATCH END` ausgeführt.

CLEAR Löscht alle gespeicherten Kommandos.

END Führt alle seit dem letzten `BATCH BEGIN` eingegebenen SQL-Kommandos aus und beendet den Batch-Modus.

 Falls ein Statement im Batch-Mode eine Exception erzeugt, werden die nachfolgenden Statements nicht ausgeführt und zum Zustand vor der Batch-Ausführung zurückgesprungen. Selbst bei eingeschaltetem Autocommit werden die Änderungen innerhalb des Batches stets zurückgenommen. Einzige Ausnahme ist ein explizites COMMIT, dass die Transaktion beendet und zur persistenten Speicherung der Änderungen führt.

 Befindet sich EXAplus im Batch-Modus werden nur spezielle Befehle angenommen:

- SQL Befehle (siehe [Kapitel 2, SQL-Referenz](#)) außer Skripte
- DEFINE und UNDEFINE
- START, @ und @@
- PAUSE
- PROMPT
- SHOW
- SPOOL

Beispiel(e)

```
BATCH BEGIN;
insert into t values(1,2,3);
insert into v values(4,5,6);
BATCH END;
```

COLUMN

Syntax

```
COLUMN [<name>]; -- Anzeige der Konfiguration
COLUMN <name> <command>; -- Ändern der Konfiguration
```

Beschreibung

Zeigt entweder die Formatierungseinstellungen für die Spalte `<name>` an oder ändert die Formatierungsoptionen für die Spalte `<name>`. `<command>` kann eine Kombination aus einer oder mehreren der folgenden Optionen sein:

ON	Aktiviert die Formatierungseinstellungen für die betreffende Spalte.
OFF	Deaktiviert die Formatierungseinstellungen für die betreffende Spalte.
FORMAT <format>	Setzt den Formatstring <code><format></code> für die Werte der betreffenden Spalte. Formatierungsoptionen sind abhängig vom Typ der Spalte. Deswegen sollte man Formatstrings für Textspalten nicht auf numerische Spalten anwenden. Numerische Werte werden grundsätzlich rechtsbündig und andere Datentypen

pen linksbündig dargestellt. Formatstrings für alphanumerische Spalten haben die Form `a<num>`, wobei `<num>` die Anzahl der Buchstaben angibt. Die Breite einer einzelnen Spalte kann nicht größer als die maximale Länge einer Zeile sein.

Beispiel: `a10` formatiert einen Spalte mit der Breite 10.

Formatstrings für Zahlen bestehen aus den Elementen `,9'`, `,0'`, `.'` (Punkt) und `,EEEE'`. Diese Breite der Spalte ergibt sich aus der Länge des angegebenen Formatstrings.

9	In dieser Stelle wird eine Ziffer nur dann dargestellt, wenn es sich nicht um eine führende oder dem Nachkomma-Teil folgende Null handelt.
0	An dieser Stelle wird auf jeden Fall eine Ziffer dargestellt, auch wenn es sich um eine führende oder dem Nachkomma-Teil folgende Null handelt.
. (Punkt)	Gibt die Position des Kommas an. Das Komma wird als Punkt dargestellt. Äquivalent kann auch D geschrieben werden.
EEEE	Die Zahl wird in wissenschaftlicher Notation mit Exponenten dargestellt. Es müssen genau vier E's im Formatstring vorhanden sein, denn die Breite der Spalte wird durch den Exponenten um vier Zeichen erhöht.

Es gelten folgende zusätzliche Regeln:

- Es muss mindestens eine Neun oder Null angegeben werden.
- Der Dezimaltrennpunkt ist optional und es ist nur ein Dezimalpunkt erlaubt.
- Tritt ein Dezimalpunkt auf, so muss sowohl davor als auch danach eine Neun oder Null stehen.
- Vor dem Dezimalpunkt sind Neunen nur vor Nullen erlaubt, nach dem Dezimalpunkt verhält es sich genau umgekehrt.
- Die vier E's sind optional und dürfen nur einmal, am Ende der Formatangabe stehen. Wird diese wissenschaftliche Darstellung gewählt, so wird auf eine Vorkommastelle normalisiert.
- Alle Buchstaben sind unabhängig von der Groß- und Kleinschreibung.

Beispiel: `99990.00` formatiert eine Zahl mit der Breite acht (sieben Ziffern und der Dezimalpunkt). Es werden bis zu fünf, aber mindestens eine Vorkomma- und genau zwei Nachkommastellen angezeigt. Im Falle der Zahl Null würde beispielsweise `0.00` angezeigt, im Fall von `987.6543` würde `987.65` ausgegeben, wobei hier mathematisch gerundet wird. Ist ein Wert nicht darstellbar, da er zu viele Vorkommastellen benötigt, so wird für jede Ziffer `#` angezeigt.

CLEAR	Löscht die Formatierungsoptionen für die betreffende Spalte.
JUSTIFY <LEFT RIGHT CENTER>	Bestimmt die Ausrichtung des Spaltennamens.
LIKE <spalte>	Kopiert alle Formatierungsoptionen von der angegebenen Spalte.
WORD_WWRAPPED	Die Werte der betreffenden Spalte werden wenn möglich zwischen den einzelnen Wörtern umgebrochen.
WRAPPED	Die Werte der betreffenden Spalte werden am rechten Rand der Spalte umgebrochen.
TRUNCATED	Die Werte der betreffenden Spalte werden am rechten Rand der Spalte abgeschnitten.
NULL <text>	Nullwerte der entsprechenden Spalte werden als <text> dargestellt.
HEADING <text>	Setzt eine neue Überschrift für die betreffende Spalte.
ALIAS <text>	Setzt einen Aliasnamen für die betreffende Spalte.

Die folgenden "COLUMN"-Kommandos von SQL*Plus werden nicht unterstützt. Aus Kompatibilitätsgründen führen sie allerdings auch nicht zu einem Syntaxfehler:

- NEWLINE
- NEW_VALUE
- NOPRINT
- OLD_VALUE
- PRINT
- FOLD_AFTER
- FOLD_BEFORE

Beispiel(e)

```
SQL_EXA> column A;
COLUMN A ON
FORMAT 9990

SQL_EXA> column A format 90000.0;
SQL_EXA> column b format 99990;
SQL_EXA> column b just left;
SQL_EXA> select a,a as b from ty;

      A   B
----- -----
 0011.0 11
 0044.0 44
 0045.0 45
 0019.0 19
 0087.0 87
 0099.0 99
 0125.0 125
 0033.0 33
 0442.0 442
```

CONNECT

Syntax

```
CONNECT <user>[ /<password> ][@<connection string>];
```

Beschreibung

Mit diesem Befehl kann eine neue Verbindung zu Exasol hergestellt werden.

Besteht bereits eine Verbindung, so wird diese getrennt, falls die neue Verbindung erfolgreich aufgebaut werden kann. Wenn kein Passwort angegeben wurde, fordert EXAplus zur Eingabe auf. Fehlen die Login-informationen, so werden diese aus der letzten Verbindung übernommen. Vor dieser Trennung wird auch ein COMMIT durchgeführt, sofern "SET AUTOCOMMIT EXIT" eingestellt war.

Beispiel(e)

```
CONN scott/tiger;
CONNECT scott/gondor@191.168.2.1:8563;
```

DEFINE

Syntax

```
DEFINE [<variable>[=<wert>]];
```

Beschreibung

Weist der Variablen <variable> den String <wert> zu. Falls diese Variable nicht existiert, wird sie angelegt. Einfache und doppelte Anführungszeichen müssen wie in SQL doppelt angegeben werden. Wird DEFINE nur mit dem Namen einer Variablen aufgerufen, so wird der Wert der Variablen angezeigt. Durch den Aufruf von DEFINE ohne Parameter werden alle Variablen und die zugewiesenen Werte aufgelistet.



Bei der Benutzung von Variablen dient der Punkt als Trennsymbol von Argumentnamen (z.B. nach **define v=t** wird &v.c1 als t.c1 interpretiert). Für das Einfügen eines Punktes müssen Sie deshalb zwei Punkte verwenden (&v..c1 entspricht dann t.c1)

Beispiel(e)

```
define tablename=usernames;
define message='Aktion erfolgreich beendet.';
```

DISCONNECT

Syntax

```
DISCONNECT;
```

Beschreibung

Mit diesem Befehl wird die aktuelle Verbindung von EXAplus zur Datenbank getrennt, sofern eine solche existiert. Hierbei wird auch ein Commit durchgeführt, sofern "SET AUTOCOMMIT EXIT" eingestellt war. Falls keine Verbindung besteht, hat der Befehl keine Auswirkungen.

Beispiel(e)

```
DISCONNECT;
```

EXIT und QUIT

Syntax

```
[EXIT|QUIT][;]
```

Beschreibung

Exit oder Quit schließen gegebenenfalls die Verbindung zur Datenbank und beenden EXAplus (nicht für die GUI-Version verfügbar). Hier wird kein abschließendes Semikolon benötigt.

Beispiel(e)

```
exit;  
quit;
```

HOST

Syntax

```
HOST <command>;
```

Beschreibung

Führt ein Betriebssystem-Kommando auf dem Client-Rechner aus und kehrt dann zu EXAplus zurück. Einfache und doppelte Anführungszeichen müssen wie in SQL doppelt angegeben werden.



Es ist nicht möglich, Programme, die Eingaben von der Tastatur erwarten, mit dem HOST-Befehl zu verwenden.

Beispiel(e)

```
host cat test.sql;
```

PAUSE

Syntax

```
PAUSE [<text>];
```

Beschreibung

Wie **PROMPT**, wartet aber nach dem Anzeigen des Textes darauf, dass der Benutzer die Return-Taste drückt. Einfache und doppelte Anführungszeichen müssen wie in SQL doppelt angegeben werden.

Beispiel(e)

```
prompt 'Bitte beachten Sie die folgende Meldung!';  
pause 'Meldung: &message';
```

PROMPT

Syntax

```
PROMPT [<text>];
```

Beschreibung

Gibt <text> in der Konsole aus. Fehlt der Parameter, wird eine leere Zeile ausgegeben. Einfache und doppelte Anführungszeichen müssen wie in SQL doppelt angegeben werden.

Beispiel(e)

```
prompt Fertig.;
```

SET AUTOCOMMIT

Syntax

```
SET AUTOCOMMIT ON|OFF|EXIT;
```

Beschreibung

Steuert, ob Exasol automatisch COMMIT-Befehle ausführen soll.

- ON Nach jedem SQL-Kommando wird automatisch ein COMMIT-Befehl durchgeführt. Dies ist die Voreinstellung.
- OFF Es werden keine automatischen COMMIT-Befehle ausgeführt.
- EXIT Beim Beenden von EXAplus wird ein COMMIT-Befehl ausgeführt.



Die für normale Anwendungen empfohlene Einstellung ist "ON". Siehe auch [Abschnitt 3.1, Transaktions-Management](#).

Wenn EXAplus durch CTRL-C abgebrochen wird, wird kein automatischer COMMIT-Befehl ausgeführt.

Beispiel(e)

```
set autocommit off;
```

SET AUTOCOMPLETION

Syntax

```
SET AUTOCOMPLETION ON|OFF;
```

Beschreibung

Schaltet das Leistungsmerkmal Auto-Completion ein oder aus.

Beispiel(e)

```
set autocompletion off;
set autocompletion on;
```

SET COLSEPARATOR

Syntax

```
SET COLSEPARATOR <Trennzeichen>;
```

Beschreibung

Setzt die Zeichenkette, die zwei Spalten voneinander trennt. Die Voreinstellung ist ein Leerzeichen.

Beispiel(e)

```
SQL_EXA> set colseparator ' | ';
SQL_EXA> select * from ty;

      A          | B
-----+-----
     11 | Meier
```

SET DEFINE

Syntax

```
SET DEFINE <C> | ON | OFF;
```

Beschreibung

Setzt das Zeichen, mit dem User-Variablen eingeleitet werden. Die Voreinstellung ist das Zeichen "&". Bei <C> muss es sich um ein einzelnes Sonderzeichen handeln. Mittels ON und OFF können User-Variablen generell aktiviert bzw. deaktiviert werden.

Beispiel(e)

```
SQL_EXA> define wert=A;
SQL_EXA> prompt &wert;
A
SQL_EXA> set define %;
SQL_EXA> prompt %wert &wert;
A &wert
```

SET ENCODING

Syntax

```
SET ENCODING [<encoding>];
```

Beschreibung

Stellt den Zeichensatz für verschiedene Dateien ein, welche EXAplus liest oder schreibt.

Standardeinstellung für das Lesen bzw. Schreiben von allen Dateien ist UTF8, kann aber auch mittels dem Kommandozeilenparameter `-encoding` geändert werden. Die unterstützten Zeichensätze finden Sie im Anhang unter [Anhang D, Unterstützte Zeichensätze](#). Wenn ohne Argument aufgerufen, werden die aktuellen Einstellungen aufgelistet.

Beispiel(e)

```
set encoding Latin1;
```

SET ESCAPE

Syntax

```
SET ESCAPE <C> | ON| OFF;
```

Beschreibung

Definiert das Escape-Zeichen, mit dem spezielle Zeichen wie z.B. & eingegeben werden können. Bei <C> muss es sich um ein einzelnes Sonderzeichen handeln. Mittels SET ESCAPE ON|OFF kann man das Escape-Zeichen aktivieren bzw. deaktivieren. Wird ein Escape-Zeichen gesetzt, ist er automatisch auch aktiviert. Als Voreinstellung ist das Zeichen \ definiert, aber deaktiviert.

Beispiel(e)

```
SQL_EXA> define wert=a;
SQL_EXA> prompt $&wert;
$ a
SQL_EXA> set escape $;
SQL_EXA> prompt $&wert;
&wert
```

SET FEEDBACK

Syntax

```
SET FEEDBACK ON|OFF |<num>;
```

Beschreibung

Steuert, wann Bestätigungen ausgegeben werden. Darunter fallen die Ausgaben nach Statements wie "INSERT INTO" und die Anzeige der Zeilenzahl bei der Ausgabe einer Tabelle. "OFF" unterdrückt die Ausgabe aller Bestätigungen, "ON" schaltet alle Bestätigungen ein. Wenn <num> ein numerischer Wert größer 0 ist, werden bei Tabellen mit mindestens <num> Zeilen die Zeilenzahlen ausgegeben. Der Befehl "SET VERBOSE OFF" und der Kommandozeilen-Parameter "-q" setzen diese Einstellung auf "OFF".

Beispiel(e)

```
SQL_EXA> set feedback off;
SQL_EXA> insert into ty values (44,'Schmitt');
SQL_EXA> select * from ty;

A                  B
-----
11 Meier
44 Schmitt
```

SET HEADING

Syntax

```
SET HEADING ON|OFF;
```

Beschreibung

Schaltet die Ausgabe von Spaltenüberschriften ein oder aus. Die Voreinstellung ist "ON".

Beispiel(e)

```
SQL_EXA> set heading off;
SQL_EXA> select * from ty;
      11 Meier
      44 Schmitt
      45 Huber
      19 Kunze
      87 Mueller
      99 Smith
     125 Dooley
      33 Xiang
     442 Chevaux
```

SET LINESIZE

Syntax

```
SET LINESIZE <n>;
```

Beschreibung

Setzt die Breite der Ausgabezeilen auf "n" Zeichen. Wenn eine Tabelle ausgegeben wird, deren Zeilen breiter sind, fügt EXAplus Zeilenumbrüche zwischen zwei Spalten ein.

Beispiel(e)

```
SQL_EXA> set linesize 20;
SQL_EXA> select * from ty;

A
-----
B
-----
      11
Meier
```

SET NULL

Syntax

```
SET NULL <text>;
```

Beschreibung

Definiert die Zeichenkette, mit der Nullwerte in Tabellen dargestellt werden. Die Voreinstellung ist ein leeres Feld. Diese Einstellung wird nur für Spalten verwendet, für die nicht mit dem Befehl [COLUMN](#) eine andere Darstellung von Nullwerten definiert wurde.

Beispiel(e)

```
SQL_EXA> set null LEER;
SQL_EXA> select NULL from dual;

NULL
-----
LEER
```

SET NUMFORMAT

Syntax

```
SET NUMFORMAT <Formatstring>;
```

Beschreibung

Setzt die Formatierung von numerischen Spalten in Tabellen. Der Formatstring gleicht dem für den **COLUMN**-Befehl. Dort finden Sie auch eine nähere Erläuterung der möglichen Formate. Diese Einstellung wird nur für numerische Spalten verwendet, für die nicht mit dem Befehl **COLUMN** ein anderes Format definiert wurde.

Beispiel(e)

```
SQL_EXA> set numformat 99990.00;
SQL_EXA> select 43 from dual;

43
-----
43.00
```

SET PAGESIZE

Syntax

```
SET PAGESIZE <num> | UNLIMITED;
```

Beschreibung

Stellt ein, nach wie vielen Textzeilen die Spaltenüberschriften wiederholt werden sollen. Die Voreinstellung ist "UNLIMITED", was bewirkt, dass die Spaltenüberschriften nur vor der ersten Zeile angezeigt werden. Wenn die Einstellung **HEADING** den Wert OFF hat, werden keine Spaltenüberschriften angezeigt.

Beispiel(e)

```
SQL_EXA> set pagesize 10;
SQL_EXA> select * from ty;

A           B
-----
11 Meier
44 Schmitt
45 Huber
```

```
19 Kunze
87 Mueller
99 Smith
125 Dooley
33 Xiang
```

A

B

```
442 Chevaux
```

SET SPOOL ROW SEPARATOR

Syntax

```
SET SPOOL ROW SEPARATOR LF | CR | CRLF | AUTO;
```

Beschreibung

Definiert das Zeilenumbruchzeichen für den **SPOOL**-Befehl. Die Option AUTO (Default) benutzt das auf dem lokalen System übliche Zeichen.

Beispiel(e)

```
set spool row separator LF;
```

SET TIME

Syntax

```
SET TIME ON | OFF;
```

Beschreibung

Schaltet die Ausgabe der aktuellen Uhrzeit des Clientsystems im Eingabeprompt ein oder aus.

Beispiel(e)

```
SQL_EXA> set time on;
16:28:36 SQL_EXA>
```

SET TIMING

Syntax

```
SET TIMING ON | OFF;
```

Beschreibung

Schaltet die Anzeige der für die Ausführung eines SQL-Kommandos benötigten Zeit ein oder aus.

Beispiel(e)

```
SQL_EXA> set timing on;
SQL_EXA> create table tx(a DECIMAL);

Timing element: 1
Elapsed: 00:00:00.313
```

SET TRUNCATE HEADING

Syntax

```
SET TRUNCATE HEADING ON|OFF;
```

Beschreibung

Definiert, ob Spaltenüberschriften entsprechend der Länge der Spalten-Datentypen abgeschnitten werden oder nicht. Die Voreinstellung ist "ON".

Beispiel(e)

```
SQL_EXA> select * from dual;
D
-
SQL_EXA> set truncate heading off;
SQL_EXA> select * from dual;
DUMMY
-----
```

SET VERBOSE

Syntax

```
SET VERBOSE ON|OFF;
```

Beschreibung

Schaltet zusätzliche Programm-Informationen ein oder aus. Voreingestellt ist ON. Um den Wert beim Programmstart auf OFF zu setzen kann der Parameter -q verwendet werden.

Beispiel(e)

```
set verbose off;
```

SHOW

Syntax

```
SHOW [<var>];
```

Beschreibung

Zeigt die EXAplus-Einstellungen an (siehe unten). Wenn keine Einstellung angegeben wurde, werden alle angezeigt.

Beispiel(e)

```
SQL_EXA> show;
AUTOCOMMIT = "ON"
AUTOCOMPLETION = "ON"
COLSEPARATOR = " "
DEFINE = "&"
ENCODING = "UTF-8"
ESCAPE = "OFF"
FEEDBACK = "ON"
HEADING = "ON"
LINESIZE = "200"
NULL = "null"
NUMFORMAT = "null"
PAGESIZE = "UNLIMITED"
SPOOL ROW SEPARATOR = "AUTO"
TIME = "OFF"
TIMING = "OFF"
TRUNCATE HEADING = "ON"
VERBOSE = "ON"
```

SPOOL

Syntax

```
SPOOL [<file>|OFF];
```

Beschreibung

Wenn ein Dateiname als Parameter angegeben wurde, wird die Datei geöffnet und es werden die Ausgaben von EXAplus in dieser Datei gespeichert (als Zeichensatz wird der mittels Kommandozeilenparameter -encoding bzw. über den Befehl [SET ENCODING](#) gesetzte Zeichensatz verwendet). Falls die Datei schon existiert, wird sie überschrieben. SPOOL OFF beendet das Speichern und schließt die Datei. Durch Eingabe von SPOOL ohne Parameter wird der Name der Spool-Datei angezeigt.

Beispiel(e)

```
spool log.out;
select * from table1;
spool off;
```

TIMING

Syntax

```
TIMING START | STOP | SHOW;
```

Beschreibung

Mit diesem Befehl werden die eingebauten Timer gesteuert.

TIMING START [name]	Startet einen neuen Timer mit dem angegebenen Namen. Wenn kein Name angegeben wurde, so wird die Nummer des Timers als Name verwendet.
TIMING STOP [name]	Hält den Timer mit dem angegebenen Namen an und zeigt die gemessene Zeit an. Wurde kein Name angegeben, so wird der zuletzt gestartete Timer angehalten.
TIMING SHOW [name]	Zeigt die von dem angegebenen Timer gemessene Zeit an ohne diesen zu beenden. Wurde kein Name angegeben, so werden alle Timer angezeigt.

Beispiel(e)

```
timing start gesamtzeit;
timing show;
timing stop gesamtzeit;
```

UNDEFINE

Syntax

```
UNDEFINE <variable>;
```

Beschreibung

Löscht die Variable <variable>. Falls es keine Variable mit dem angegebenen Namen gibt, wird ein Fehler gemeldet.

Beispiel(e)

```
undefine message;
```

WHENEVER

Syntax

```
WHENEVER SQLERROR|OSERROR EXIT [<exit_code>] |CONTINUE|ABORT [ROLLBACK|COMMIT];
```

Beschreibung

Mit diesem Kommando kann das Verhalten von EXAplus bei Fehlern definiert werden. WHENEVER SQLERROR reagiert auf Fehler in der Ausführung von SQL-Kommandos, WHENEVER OSERROR auf Fehler vom Betriebssystem (Datei nicht gefunden etc.).

Das Verhalten kann folgendermaßen eingestellt werden:

EXIT	Beendet EXAplus.
CONTINUE	Setzt die Programmausführung trotz des Auftretens eines Fehler fort.
ABORT	Bricht die Programmausführung bei Auftretens eines Fehler ab, beendet EXAplus aber nicht.



CONTINUE ist die Voreinstellung für beide Arten von Fehlern (bzw. EXIT, falls EXAplus mit dem Parameter `-x` gestartet wurde).

Beispiel(e)

```
whenever sqlerror continue;
whenever sqlerror exit 4;
```

4.2. ODBC-Treiber

Dieser Abschnitt beschreibt Installation, Konfiguration und Verwendung des **ODBC**-Treibers für Exasol. Weitere Informationen zur Verwendung des Treibers mit speziellen Produkten finden Sie im Knowledge Center auf unserer Homepage wwwexasol.com.

4.2.1. Unterstützte Standards

Der von Exasol bereitgestellte ODBC-Treiber für Exasol unterstützt den ODBC 3.5 Standard (Core Level).

Die folgenden Features werden nicht unterstützt:

- "Positioned update/delete" und "batched updates/deletes" (`SQLSetPos`)
- Asynchrone Ausführung
- Bookmarks



Sollte eine Query einen Rowcount größer als $2^{31}-1$ erzeugen, wird die ODBC-Funktion `SQLRowCount()` trotzdem nur 2147483647 zurückgeliefert. Grund dafür ist die 32-Bit Beschränkung für diesen Rückgabewert im ODBC-Standard.

4.2.2. Windows-Version des ODBC-Treibers

Systemvoraussetzungen

Der ODBC-Treiber ist sowohl für die 32-Bit Version als auch für die 64-Bit Version von Windows verfügbar. Die Voraussetzungen für die Installation des ODBC-Treibers sind nachfolgend aufgeführt:

- Der Windows-Benutzer, der die Installation von Exasol ODBC durchführt, muss auf dem Rechner Administrator oder Mitglied der Administratoren-Gruppe sein.
- Auf dem System muss Microsoft .NET Framework 4.0 Client Profile™ installiert sein.
- Alle Anwendungen und Dienste, die ODBC einbinden, müssen vor der Installation beendet werden. Sollte "Business Objects XI" auf dieser Maschine installiert sein, muss der "Web Intelligence Report Server"-Dienst angehalten werden.

Der ODBC-Treiber wurde auf folgenden Systemen erfolgreich getestet:

- Windows 10 (x86/x64)
- Windows 7, Service Pack 1 (x86/x64)
- Windows Server 2012 R2 (x86/x64)
- Windows Server 2012 (x86/x64)
- Windows Server 2008 R2, Service Pack 1 (x86/x64)
- Windows Server 2008, Service Pack 2 (x86/x64)

Installation

Starten Sie die Setup-Datei, ein unkomplizierter Setup-Wizard führt Sie anschließend durch die Installations-Schritte. Bitte beachten Sie, dass für 32Bit- und 64Bit-Anwendungen zwei separate Installations-Dateien benötigt werden.

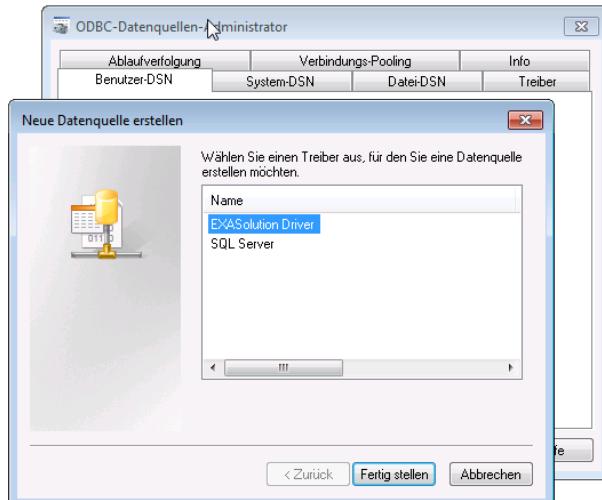


Bereits eingerichtete ODBC-Datenquellen werden bei einem Update auf die neue Treiber-Version gesetzt.

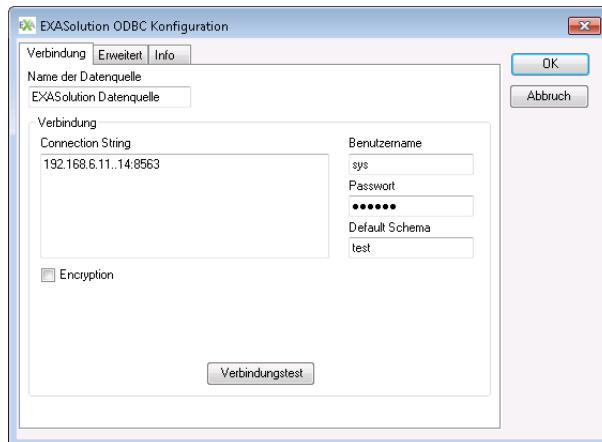
Einrichten des Treibers und der Datenquellen

Zum Einrichten des ODBC-Treibers verwenden Sie das ODBC Konfigurationstool, welches auf Windows-Systemen bereits enthalten ist. Eine Verknüpfung auf dieses Tool finden Sie im Startmenü-Eintrag des Exasol ODBC-Treibers. Falls beide Treiber-Varianten (32Bit und 64Bit) installiert sind, so finden Sie dort je eine Verknüpfung auf die 32-Bit und 64-Bit Varianten des Tools.

Im Tool klicken Sie anschließend in der Registerkarte “Benutzer-DSN” oder “System-DSN” auf das Feld “Hinzufügen” und wählen im Fenster “Neue Datenquelle erstellen” “Exasol Driver” aus.



Im nächsten Schritt werden die Exasol Einstellungen konfiguriert:



Name der Datenquelle
Connection String

Name der neuen ODBC Datenquelle
Liste der Hostnamen bzw. IP-Adressen sowie der passende Port des Exasol-Clusters (z.B. 192.168.6.11..14:8563).



Der Client-Rechner muss in der Lage sein, eventuelle Hostnamen aufzulösen. Ist dies nicht möglich oder sind Sie sich nicht sicher, wenden Sie sich bitte an Ihren Netzwerkadministrator.

Encryption
Benutzername
Passwort
Default Schema

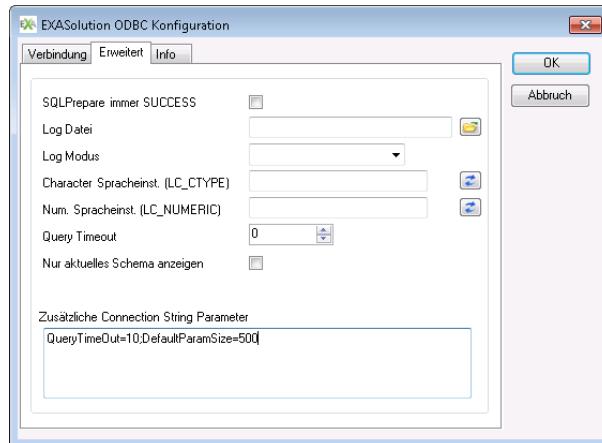
Schaltet die automatische Verschlüsselung ein bzw. aus.
Name des Default Benutzers für diese Verbindung.
Passwort des Benutzers.
Name des Schemas, das beim Zustandekommen einer Verbindung automatisch geöffnet werden soll. Ist dieses Schema während eines Verbindungsversuchs nicht

verfügbar, wird eine Fehlermeldung ausgegeben und die Verbindung mit Exasol kommt nicht zustande.

Verbindungstest

Hiermit kann der Benutzer testen, ob die eingegebenen Verbindungsdaten richtig sind. Es wird versucht, eine Verbindung mit Exasol aufzubauen.

In den erweiterten Einstellungen finden Sie diverse Zusatzoptionen sowie die Möglichkeit, beliebige Connection String Parameter zu setzen (siehe auch [Abschnitt 4.2.5, Connection-String Parameter](#)):



Sobald Sie im Konfigurationsfenster "OK" geklickt haben, erscheint Ihre neue Verbindung in der Liste der Windows Datenquellen.

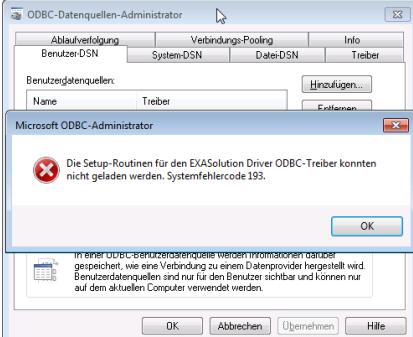
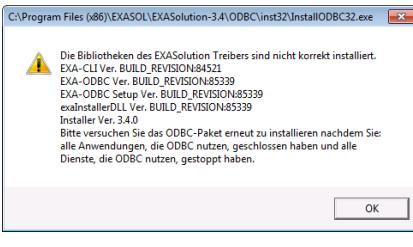
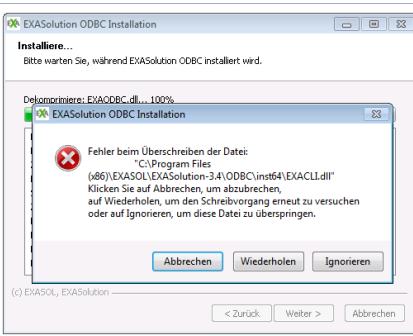
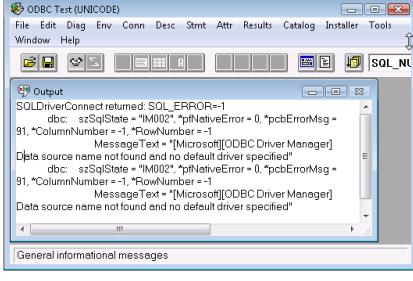


Connection Pooling des Treibermanagers ist per Default nicht eingeschaltet ist. Dies können Sie im Konfigurationstool "Datenquellen (ODBC)" explizit aktivieren. In diesem Fall müssen Sie jedoch beachten, dass wiederverwendete Verbindungen die mittels SQL-Befehlen (siehe [ALTER SESSION](#)) geänderten Session-Einstellungen beibehalten.

Bekannte Probleme

In diesem Kapitel werden bekannte Problem im Umgang mit dem ODBC-Treiber vorgestellt. In den meisten Fällen sind die Ursachen leicht zu beheben.

Tabelle 4.3. Bekannte Probleme im Umgang mit dem ODBC-Treiber unter Windows

Problem	Lösung	Screenshot
Systemfehlercode: 126, 127, 193 bzw. 14001	Wichtige Bestandteile des ODBC-Treibers sind nicht korrekt installiert. Beenden Sie alle Anwendungen und Dienste, die ODBC nutzen könnten, und installieren Sie Exasol ODBC erneut. Sollte der Fehler 14001 aufgetreten sein, installieren Sie die Visual C++ Redistributables und anschließend Exasol ODBC erneut. Fehler 14001 kann auch durch einspielen der neusten Windows-Updates behoben werden.	
Installation error: Incompatible dll versions	Eine ähnliche Meldung kann auch während der Installation auftreten. Diese Meldung tritt auf, wenn eine ältere Exasol ODBC Version schon installiert war und aus irgend einem Grund bei der Installation nicht überschrieben werden konnte. Beenden Sie alle Anwendungen und Dienste, die ODBC nutzen könnten und installieren Sie Exasol ODBC erneut.	
Error opening file for writing	Der Installer hat festgestellt, dass er eine Exasol ODBC Komponente nicht überschreiben kann. Beenden Sie alle Anwendungen und Dienste, die ODBC nutzen könnten, und installieren Sie Exasol ODBC erneut.	
Data source name not found and no default driver specified	Überprüfen Sie den Namen der Datenquelle. Auf einem 64-Bit System überprüfen Sie bitte, ob die angelegte Datenquelle nicht eine 64-Bit Datenquelle ist und die Anwendung möglicherweise eine 32-Bit Datenquelle erwartet oder umgekehrt.	

4.2.3. Linux/Unix-Version des ODBC-Treibers



Bitte lesen Sie zur Installation und Konfiguration unbedingt die im Installationspaket enthaltene Datei `README.txt`.



Die Liste der Optionen, die Sie in der Datei `odbc.ini` setzen können, finden Sie in [Abschnitt 4.2.5, Connection-String Parameter](#).

Der Exasol ODBC-Treiber für Linux/Unix ist so konzipiert, dass er auf möglichst vielen Distributionen lauffähig ist, und unter folgenden Systemen erfolgreich getestet:

- Red Hat / CentOS 7 (x64)
- Red Hat / CentOS 6 (x86/x64)
- Debian 8 (x86/x64)
- Ubuntu 16.04 LTS (x86/64)
- Ubuntu 14.04 LTS (x86/64)
- SUSE Linux Enterprise Server 12 (x64)
- SUSE Linux Enterprise Desktop 12 (x64)
- SUSE Linux Enterprise Server 11 (x86/x64)
- openSUSE Leap 42.2 (x64)

- macOS Sierra (32Bit/64Bit)
- OS X 10.11 (32Bit/64Bit)

- FreeBSD 11.0 (64Bit)
- FreeBSD 10.3 (64Bit)



Der ODBC-Treiber wurde unter Mac OS X mit dem Treibermanager iODBC in zwei Varianten getestet (welche davon benutzt wird, hängt von der Applikation ab):

- Als System-Bibliothek (bereits im System enthalten)
- Als Framework in Version 3.52.7 (siehe auch www.iodbc.org)

Bekannte Probleme

Tabelle 4.4. Bekannte Probleme im Umgang mit dem ODBC Treiber für Linux/Unix

Beschreibung	Lösung
Fehlermeldung "Data source name not found, and no default driver specified"	Eventuell benutzt der unixODBC-Treibermanager die falsche <code>odbc . ini</code> Datei. Über die Umgebungsvariable <code>ODBCINI</code> können Sie die zu verwendende Datei festlegen. Weiterhin könnte die Ursache sein, dass mehrere Versionen von unixODBC installiert sind und eine falsche benutzt wird. Z.B. mittels <code>isql -version</code> können Sie die aktuell verwendete Version ermitteln.
Fehlermeldung "Invalid utf-8 character in input"	Setzen Sie die Variable <code>LC_ALL</code> so, dass die eingegebenen Zeichen in dieser Locale darstellbar sind.
Falsch oder gar nicht dargestellte Zeichen in der Ausgabe der Konsole oder Ihrer ODBC-Applikation	Setzen Sie die Umgebungsvariablen <code>LC_CTYPE</code> bzw. <code>LC_NUMERIC</code> oder die Parameter <code>CONNECTIONLCCTYPE</code> bzw. <code>CONNECTIONLCNUMERIC</code> in Ihrer Datenquelle (<code>odbc . ini</code>) auf die Locale, in der die Zeichen ausgegeben werden sollen. Applikationen, die den DataDirect Treibermanager benutzen, benötigen in der Regel ein UTF8 Locale.

4.2.4. Verbindungsauflaufbau in einer ODBC-Anwendung

Um eine Verbindung zu Exasol mit dem ODBC-Treiber aufzubauen, stehen zwei verschiedene ODBC-Funktionen zur Verfügung:

SQLConnect()

In dieser Methode wird ein [DSN](#)-Eintrag ausgewählt sowie Nutzer und Passwort angegeben.

Beispieldaufruf:

```
SQLConnect(connection_handle,
    (SQLCHAR*)"exa_test", SQL_NTS,
    (SQLCHAR*)"sys", SQL_NTS,
    (SQLCHAR*)"exasol", SQL_NTS);
```

SQLDriverConnect()

Bei der Funktion `SQLDriverConnect()` existieren zwei unterschiedliche Alternativen. Bei der **DSN**-Variante wird eine Verbindung aus den Einträgen der `odbc.ini` gewählt, bei der **DRIVER**-Variante ein Treiber aus der `odbcinst.ini`. In beiden Fällen wird ein Connection-String angegeben, in dem noch weitere Optionen definiert werden können (siehe nächster Abschnitt).

Beispielaufruf:

```
SQLDriverConnect(
    connection_handle, NULL,
    (SQLCHAR*)"DSN=exa_test;UID=sys;PWD=exasol", SQL_NTS,
    NULL, 0, NULL, SQL_DRIVER_NOPROMPT);

SQLDriverConnect(
    connection_handle, NULL,
    (SQLCHAR*)
    "DRIVER={EXASOL Driver};EXAHOST=192.168.6.11..14:8563;UID=sys;PWD=exasol",
    SQL_NTS, NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

4.2.5. Connection-String Parameter

Im Connection-String wird eine Reihe von Key-Value Paaren an eine Connect-Funktion übergeben und der Treibermanager sorgt dafür, dass der passende Treiber geladen wird und die notwendigen Parameter an diesen weiter gegeben werden.

 Die Daten des Connection-Strings haben Vorrang vor der `odbc.ini`-Datei unter Linux/Unix bzw. der Voreinstellungen der Datenquelle unter Windows.

Typische Beispiele für Connection-Strings, mit denen eine Exasol Verbindung aufgebaut werden kann:

```
DSN=exa_test;UID=sys;PWD=exasol;EXASCHEMA=MY_SCHEMA
DRIVER={EXASOL Driver};EXAHOST=192.168.6.11..14:8563;UID=sys;PWD=exasol
```

Zur Zeit werden folgende Schlüsselwörter in Connection-Strings unterstützt:

EXAHOST Verbindungsknoten zum Exasol-Cluster sowie der zugehörige Port (z.B. 192.168.6.11..14:8563).

Beim Öffnen einer Verbindung wählt der Treiber zufällig eine Adresse aus dem Adressbereich aus. Falls dies nicht funktioniert, wird er alle anderen möglichen Adressen ausprobieren.

Anstatt eines einfachen IP-Bereichs können Sie auch eine komma-separierte Liste aus Host:Port-Paaren angeben.

Beispiele:

myhost : 8563

Einzelner Server mit Namen myhost und Port 8563.

myhost1,myhost2:8563

Zwei Server mit Port 8563.

myhost1..4:8563	Vier Server (myhost1, myhost2, myhost3, myhost4) und Port 8563.
192.168.6.11..14:8563	Vier Server von 192.168.6.11 bis 192.168.6.14 und Port 8563.
	Anstelle einer konkreten Liste kann hier auch eine Datei definiert werden, die eine entsprechende Liste enthält (z.B. /c:/mycluster.txt). Die beiden "/" (Slash) signalisieren dem Treiber, dass ein Dateiname folgt.
EXAUID oder UID	Benutzername für den Login. UID wird von einigen Tools aus gefiltert.
EXAPWD oder PWD	Passwort des Benutzers. PWD wird von einigen Tools aus gefiltert.
EXASCHEMA	Schema, das nach dem Verbinden geöffnet wird.
EXALOGFILE	Datei, in die Log-Informationen des Treibers geschrieben werden.
	 Je nach Log-Modus (siehe nächster Punkt) kann das Logfile schnell sehr groß werden, was im Produktiv-Betrieb nicht zu empfehlen ist.
	 Bitte beachten Sie, dass im Logfile unter Umständen sensible Daten mit protokolliert werden (SQL-Befehle oder Daten).
LOGMODE	Gibt den Modus für die Logdaten an. Folgende Werte sind möglich:
DEFAULT	Schreibt die wichtigsten Funktionsaufrufe und SQL-Befehle ins Logfile.
VERBOSE	Es werden zusätzliche Ausgaben geschrieben, unter anderem zu internen Prozessabläufen und Ergebnisdaten.
ON ERROR ONLY	Es werden nur im Falle eines Fehlers Logdaten geschrieben.
DEBUGCOMM	Erweiterte Ausgaben zur Analyse der Client/Server-Kommunikation (wie VERBOSE, aber ohne Protokollierung der Daten- und Parameter-Tabellen).
NONE	Es werden keine Logdaten geschrieben.
KERBEROSSERVICENAME	Principal-Name des Kerberos-Services. Falls hier nichts explizit angegeben wurde, wird der Name "exasol" verwendet.
KERBEROSHOSTNAME	Host-Name des Kerberos-Services. Falls hier nichts explizit angegeben wurde, wird der Host-Name des Parameters EXAHOST verwendet.
ENCRYPTION	Schaltet die automatische Verschlüsselung ein. Werte können "Y" oder "N" sein. Default ist "Y".
AUTOCOMMIT	Autocommit Modus der Verbindung. Werte können "Y" oder "N" sein. Default ist "Y".
QUERYTIMEOUT	Definiert den Query Timeout einer Verbindung in Sekunden. Deaktiviert, falls der Wert auf 0 gesetzt ist. Default ist 0.
CONNECTTIMEOUT	Maximale Zeit in Millisekunden, die der Treiber für den Aufbau einer TPC-Verbindung zu einem Server wartet. Dieser Timeout ist gerade für größere

	Cluster mit mehreren Ersatzservern interessant, um die Login-Dauer zu minimieren. Default: 2000
SUPERCONNECTION	Erlaubt es, Statements auch dann auszuführen, wenn die maximal erlaubte Anzahl an aktiven Sessions (die eine Anfrage ausführen) erreicht wurde. Werte können "Y" oder "N" sein. Default ist "N".
	 Dieser Parameter kann nur durch Nutzer SYS gesetzt werden.
	 SUPERCONNECTION sollte nur im Falle von schwerwiegenden Performance-Problemen benutzt werden, bei denen es nicht mehr möglich ist, sich innerhalb einer vernünftigen Zeit auf dem System anzumelden und Anfragen zu starten. Durch diesen Parameter wird es auch in solchen Fällen ermöglicht, das System zu analysieren und die Prozesse zu beenden, die für die Überlastung verantwortlich sind.
PREPAREAS	Prepare always successful: SQLPrepare und SQLPrepareW liefern immer als Rückgabewert SQL_SUCCESS. Werte können "Y" oder "N" sein. Default ist "N".
MAXPARAMSIZE	Maximale Größe für VARCHAR Parameter Typen in Prepared Statements. Der Wert 0 deaktiviert diese Einstellung. Default ist 0.
DEFAULTPARAMSIZE	Default Größe für Prepared-Statement VARCHAR Parameter Typen, deren exakte Größe im Prepare nicht ermittelt werden kann. Default ist 2000.
COGNOSUPPORT	Falls Sie den Exasol ODBC Treiber mit Cognos verwenden wollen, empfehlen wir Ihnen diesen Parameter auf "Y" zu setzen. Default ist "N".
CONNECTIONLCCTYPE	Setzt LC_CTYPE in der Verbindung auf den angegebenen Wert. Beispielwerte für Windows: "deu", "eng". Beispielwerte für Linux/Unix: "de_DE", "en_US". Unter Linux/Unix können Sie hier auch ein Encoding setzen, z.B. "en_US.UTF-8".
CONNECTIONLCNUMERIC	Setzt LC_NUMERIC in der Verbindung auf den angegebenen Wert. Beispielwerte für Windows: "deu", "eng". Beispielwerte für Linux/Unix: "de_DE", "en_US".
SHOWONLYCURRENTSCHEMA	Definiert, ob der ODBC-Treiber für Metadaten (z.B. Liste der Spalten oder Tabellen) alles Schemas berücksichtigt oder nur das aktuelle Schema. Werte können "Y" oder "N" sein. Default ist "N".
STRINGSNOTNULL	Definiert, ob der ODBC-Treiber NULL-Strings in Tabellendaten als leere Strings zurückliefert anstatt von NULL-Werten (Beachte: In der Datenbank sind leere Strings und NULL-Strings identisch und werden zum Treiber als NULL-Strings geliefert). Werte können "Y" oder "N" sein. Default ist "N".
INTTYPESINRESULTSIFPOSSIBLE	Falls Sie diese Option einschalten, so werden für DECIMAL-Typen ohne Nachkommastellen als SQL_INTEGER (9 Stellen) bzw. SQL_BIGINT (18 Stellen) ausgegeben anstatt SQL_DECIMAL. Werte können "Y" oder "N" sein. Default ist "N".

4.2.6. Zeichensatz-Unterstützung

Der ODBC-Treiber verwendet intern den Zeichensatz UTF8. Eingaben in anderen Zeichensätzen werden vom Treiber in UTF8 konvertiert und dann an Exasol weitergegeben. Daten aus Exasol werden vom Treiber in den Zeichensatz konvertiert, der beim Client eingestellt ist.

Auf Windows-Maschinen müssen die Daten, die an den ODBC Treiber weitergegeben werden, in einem installierten Zeichensatz sein. Der Zeichensatz ist zusätzlich über die Spracheinstellungen für die Verbindung steuerbar (siehe oben).

Auf Linux/Unix-Maschinen kann der Zeichensatz über Umgebungsvariablen gesteuert werden. Um z.B. die Sprache deutsch und das Encoding UTF8 in der Konsole einzustellen, kann der Befehl: `export LC_CTYPE=de_DE.UTF-8` verwendet werden. Für grafische Anwendungen empfiehlt sich die Erstellung eines Wrapper-Skripts.

Es muss immer darauf geachtet werden, dass das zu verwendende Zeichensatz im System installiert sein muss. Auf Linux/Unix finden Sie über den Befehl "locale -a" heraus, welche Zeichensätze installiert sind.

4.2.7. Best-Practice für Entwickler

Lesen großer Datenmengen

Verwenden Sie niemals `SQLGetData`, sondern `SQLBindCol` und `SQLFetch`.

Zur optimalen Performance sollten Sie außerdem die Anzahl der Zeilen je `SQLFetch` so wählen, dass sie einer Datenmenge von 50-100 **MB** entspricht.

Einfügen von Daten in die Datenbank

Anstatt einzelner Einfüge-Operationen wie z.B. "INSERT INTO t VALUES 1, 2, ..." sollten Sie die effizientere Schnittstelle über Prepared Statements mit Parametern nutzen. Prepared Statements erreichen die beste Performance bei Parameter-Sets zwischen 50 und 100 **MB**.

Fügen Sie außerdem nach Möglichkeit die Daten in Ihren nativen Datentypen ein, z.B. die Zahl 1234 als `SQL_C_SLONG` (Integer) anstatt als `SQL_CHAR` ("1234").

Autocommit-Modus

Bitte beachten Sie, dass der Autocommit-Modus unter Windows am besten nur mittels `SQLSetConnectAttr()` ausgeschaltet werden sollte und nicht in den Einstellungen der Datenquelle. Der Windows Treibermanager bekommt diese Änderung sonst nicht mit, geht davon aus, dass Autocommit eingeschaltet ist, und leitet `SQLEndTran()` Aufrufe nicht an die Datenbank weiter. Dieses Verhalten könnte zu Problemen führen.

Probleme bei Verwendung der Lokalisierungs-Attribute

Sollte das Setzen von `CONNECTIONLCCTYPE` bzw. `CONNECTIONLC_NUMERIC` nicht möglich sein, wird die Connect-Funktion `SQL_SUCCESS_WITH_INFO` zurückliefern - mit einem entsprechenden Text im [DiagRec](#).

4.3. JDBC-Treiber

Exasol stellt für die Anbindung von Exasol an Fremdanwendungen auch einen **JDBC**-Treiber bereit.

4.3.1. Unterstützte Standards

Der von Exasol bereitgestellte JDBC-Treiber für Exasol unterstützt die JDBC 3.0 Core [API](#). Dieser Standard umfasst:

- Zugriff auf den Treiber durch die DriverManager-API und Konfiguration des Treibers mit Hilfe der DriverPropertyInfo-Schnittstelle
- Ausführung von SQL-Kommandos direkt, als Prepared Statement und im Batch-Modus
- Unterstützung mehrerer offener ResultSets
- Unterstützung der Metadaten-APIs: DatabaseMetaData und ResultSetMetaData

Die folgenden Features werden nicht unterstützt:

- Savepoints
- Benutzerdefinierte Datentypen und die Typen Blob, Clob, Array und Ref.
- Stored Procedures
- Read-only Verbindungen
- Das API ParameterMetaData



Sollte eine Query einen Rowcount größer als $2^{31}-1$ erzeugen, wird vom JDBC trotzdem nur 2147483647 zurückgeliefert. Grund dafür ist die 32-Bit Beschränkung für diesen Rückgabewert im JDBC-Standard.



Detaillierte Informationen über die unterstützten Schnittstellen stehen in der *API Reference*, die über das Startmenü (Windows) bzw. im Ordner html des Installationsverzeichnis (Linux/Unix) zu finden ist.

4.3.2. Systemvoraussetzungen

Der JDBC-Treiber benötigt eine Java-Umgebung ([JRE](#)) ab Sun Java 1.5. Für die Verwendung anderer JREs kann die korrekte Funktion des Treibers nicht garantiert werden. Der Treiber selbst ist plattformunabhängig. Das heißt, er kann auf jeder Plattform eingesetzt werden, für die eine geeignete [JRE](#) verfügbar ist.

Der JDBC-Treiber wurde auf folgenden Systemen erfolgreich getestet:

- Windows 10 (x86/x64)
- Windows 7, Service Pack 1 (x86/x64)
- Windows Server 2012 R2 (x86/x64)
- Windows Server 2012 (x86/x64)
- Windows Server 2008 R2, Service Pack 1 (x86/x64)
- Windows Server 2008, Service Pack 2 (x86/x64)
- Red Hat / CentOS 7, OpenJDK JVM 1.8.0 (x64)
- Red Hat / CentOS 6, OpenJDK JVM 1.8.0 (x86/x64)
- Debian 8, OpenJDK JVM 1.7.0 (x86/x64)
- Ubuntu 16.04 LTS, OpenJDK JVM 1.8.0 (x86/64)
- Ubuntu 14.04 LTS, OpenJDK JVM 1.7.0 (x86/64)
- SUSE Linux Enterprise Server 12, IBM's JVM 1.7.0 (x64)
- SUSE Linux Enterprise Desktop 12, OpenJDK JVM 1.7.0 (x64)
- SUSE Linux Enterprise Server 11 Service Pack 3, IBM's JVM 1.7.0 (x86/x64)
- openSUSE Leap 42.2, OpenJDK JVM 1.8.0 (x64)

- macOS Sierra, JVM 1.8.0 (64Bit)
- OS X 10.11, JVM 1.8.0 (64Bit)
- FreeBSD 11.0, OpenJDK 1.8.0 (64Bit)
- FreeBSD 10.3, OpenJDK 1.8.0 (64Bit)

Für Windows Systeme existiert ein automatischer Setup-Wizard. In diesem Fall muss das Microsoft .NET Framework 4.0 Client Profile™ installiert sein.

4.3.3. Benutzung des Treibers

JDBC direkt installieren

Nach der Installation auf Windows, liegt der Treiber als .jar-Archiv im Installationsverzeichnis vor. Auf Linux/Unix befindet er sich in der ausgelieferten .tgz-Datei. Je nach Anwendung muss dieses Archiv dem Suchpfad für die Java-Klassen (CLASSPATH) hinzugefügt werden.

Alle Klassen des JDBC-Treibers gehören zum Java-Package "com.exasol.jdbc". Die Hauptklasse des Treibers lautet

```
com.exasol.jdbc.EXADriver
```

JDBC indirekt über Maven einbinden

Der JDBC Treiber ist auch über das Exasol Maven Repository verfügbar (<https://mavenexasol.com>). Fügen Sie hierfür bitte das folgende Repository und folgende Dependency in der Projekt-Konfiguration Ihres Build-Tools hinzu (z.B. pom.xml bei Maven):

```
<repositories>
  <repository>
    <id>maven.exasol.com</id>
    <url>https://mavenexasol.com/artifactory/exasol-releases</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
<dependencies>
  <dependency>
    <groupId>com.exasol</groupId>
    <artifactId>exasol-jdbc</artifactId>
    <version>6.0.0</version>
  </dependency>
</dependencies>
```

URL der Exasol

Die URL zu Exasol hat die folgende Form:

```
jdbc:exa:<host>:<port>[ ;<prop_1>=<value_1>]...[ ;<prop_n>=<value_n> ]
```

Bedeutung der einzelnen URL-Bestandteile:

jdbc:exa

Dieser Präfix ist für den Treiber-Manager notwendig.

<host>:<port> Verbindungsknoten zum Exasol-Cluster sowie der zugehörige Port (z.B. 192.168.6.11..14:8563).

Beim Öffnen einer Verbindung wählt der Treiber zufällig eine Adresse aus dem Adressbereich aus. Falls dies nicht funktioniert, wird er alle anderen möglichen Adressen ausprobieren.

Anstatt eines einfachen IP-Bereichs können Sie auch eine komma-separierte Liste angeben.

Beispiele:

myhost:8563 Einzelner Server mit Namen myhost und Port 8563.

myhost1,myhost2:8563 Zwei Server mit Port 8563.

myhost1..4:8563 Vier Server (myhost1, myhost2, myhost3, myhost4) und Port 8563.

192.168.6.11..14:8563 Vier Server von 192.168.6.11 bis 192.168.6.14 und Port 8563.

Anstelle einer konkreten Liste kann hier auch eine Datei definiert werden, die eine entsprechende Liste enthält (z.B. //c:\\mycluster.txt). Die beiden "/" (Slash) signalisieren dem Treiber, dass ein Dateiname folgt.

<prop_i=value_i> Nach dem Port folgt eine optionale, durch ";" getrennte Liste von Eigenschaften, deren Werte beim Login gesetzt werden sollen. Diese Eigenschaften entsprechen den unterstützten DriverProperties und werden im folgenden Abschnitt beschrieben. Es ist zu beachten, dass die Werte der Eigenschaften innerhalb der URL nur aus alphanumerischen Zeichen bestehen dürfen.

Beispiel Java-Programm.

```
import java.sql.*;
import com.exasol.jdbc.*;

public class jdbcsample
{
    public static void main(String[] args)
    {
        try {
            Class.forName("com.exasol.jdbc.EXADriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection con=null;
        Statement stmt=null;
        try {
            con = DriverManager.getConnection(
                "jdbc:exa:192.168.6.11..14:8563;schema=SYS",
                "sys",
                "exasol"
            );
            stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM CAT");
            System.out.println("Schema SYS contains:");
            while(rs.next())
            {
                String str1 = rs.getString("TABLE_NAME");
                String str2 = rs.getString("TABLE_TYPE");
            }
        }
    }
}
```

```
        System.out.println(str1 + ", " + str2);
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    try {stmt.close();} catch (Exception e) {e.printStackTrace();}
    try {con.close();} catch (Exception e) {e.printStackTrace();}
}
}
```

Das obige Beispiel öffnet eine JDBC-Verbindung zu Exasol, das auf einem Cluster mit den Rechnern 192.168.6.11 bis 192.168.6.14 auf Port 8563 läuft. Es wird der Benutzer "sys" mit dem Passwort "exasol" angemeldet und dann das Schema "sys" geöffnet. Im Anschluss werden alle Tabellen aus dem Schema "sys" angezeigt:

```
Schema SYS contains:
EXA_SQL_KEYWORDS, TABLE
DUAL, TABLE
EXA_PARAMETERS, TABLE
...
```

Unterstützte DriverProperties

Die folgenden Eigenschaften können dem JDBC-Treiber über die URL übergeben werden:

Tabelle 4.5. Unterstützte DriverProperties des JDBC-Treibers

Eigenschaft	Werte	Beschreibung
schema	String	Name des Schemas, das nach Login geöffnet werden soll. Falls das Schema nicht geöffnet werden kann, schlägt der Login mit einer java.sql.SQLException fehl. Default: kein Schema öffnen
autocommit	0=aus, 1=an	Autocommit ein- oder ausschalten. Default: 1
encryption	0=aus, 1=an	Automatische Verschlüsselung ein- oder ausschalten. Default: 1
kerberosprincipalname	String	Principal-Name des Kerberos-Services. Falls hier nichts explizit angegeben wurde, wird der Name "exasol" verwendet.
kerberoshostname	String	Host-Name des Kerberos-Services. Falls hier nichts explizit angegeben wurde, wird der Host-Name des Connection Strings verwendet.
fetchsize	numerisch, >0	Datenmenge in kB, die bei einem Fetch von Exasol übertragen wird. Zu große Werte können dazu führen, dass der JVM der Speicher ausgeht. Default: 2000 <p style="text-align: center;"> Den gleichen Effekt können Sie mit der Funktion <code>setFetchSize()</code> erreichen.</p>
debug	0=aus, 1=an	Schaltet die Log-Funktion des Treibers ein, wodurch für jeden Verbindungsauftbau eine Log-Datei mit dem Namen <code>jdbc_timestamp.log</code> angelegt wird. Diese Dateien beinhalten Informationen zu den aufgerufenen Treiber-Methoden bzw. zum Verlauf der JDBC-Verbindung und können dem Exasol-Support bei der Diagnose von Problemen helfen. <p style="text-align: center;"> Im produktiven Einsatz sollte das Logging aus Performancegründen nicht verwendet werden.</p> <p>Default: 0</p>
logdir	Zeichenkette	Verzeichnis, in das die Log-Dateien geschrieben werden (nur im Debug-Modus wirksam). Beispiel: <code>jdbc:exa:192.168.6.11..14:8563;debug=1;log-dir=/tmp/my_folder/;schema=sys</code> <p style="text-align: center;"> Default ist das aktuelle Arbeitsverzeichnis der Applikation, welches aber nicht immer transparent ist. Daher sollte das Log-Verzeichnis im Debug-Modus stets gesetzt werden.</p>
clientname	Zeichenkette	Teilt dem Server mit, wie die Anwendung heißt. Default: "Generic JDBC client"
clientversion	Zeichenkette	Teilt dem Server die Version der Anwendung mit. Default: leer ("")
logintimeout	numerisch, >=0	Maximale Zeit in Sekunden, die der Treiber bei einer Connect- oder Disconnect-Anfrage auf die Datenbank warten wird. Default ist 0 (unendlich)
connecttimeout	numerisch, >=0	Maximale Zeit in Millisekunden, die der Treiber für den Aufbau einer TPC-Verbindung zu einem Server wartet. Dieser Timeout ist gerade für größere Cluster mit mehreren Ersatzservern interessant, um die Login-Dauer zu minimieren. Default: 2000
querytimeout	numerisch, >=0	Definiert, wie lange eine Query maximal in Sekunden dauern darf, bevor sie automatisch abgebrochen wird. Default ist 0 (unendlich)

Eigenschaft	Werte	Beschreibung
superconnection	0=aus, 1=an	<p>Erlaubt es, Statements auch dann auszuführen, wenn die maximal erlaubte Anzahl an aktiven Sessions (die eine Anfrage ausführen) erreicht wurde.</p> <p> Dieser Parameter kann nur durch Nutzer SYS gesetzt werden.</p> <p> superconnection sollte nur im Falle von schwerwiegenden Performance-Problemen benutzt werden, bei denen es nicht mehr möglich ist, sich innerhalb einer vernünftigen Zeit auf dem System anzumelden und Anfragen zu starten. Durch diesen Parameter wird es auch in solchen Fällen ermöglicht, das System zu analysieren und die Prozesse zu beenden, die für die Überlastung verantwortlich sind.</p> <p>Default: 0</p>
slave	0=aus, 1=an	<p>Bei Verbindungen mit slave=1 handelt es sich um parallele Verbindungen zur Ausführung von Parallel-Read und Parallel-Insert. Details und Beispiele hierzu finden Sie in unserem Solution Center: https://wwwexasol.com/support/browse/SOL-546</p> <p>Default ist 0</p>
slavetoken	numerisch, >=0	Wird zum Aufbau einer parallelen Verbindung benötigt. Default ist 0

4.3.4. Best-Practice für Entwickler

Speicher der JVM

Bei Operationen mit größeren Tabellen kann der Speicher leicht ausgehen, wenn der Speicher der JVM nicht groß genug eingestellt ist.

Lesen großer Datenmengen

Der Parameter "fetchsize" ermöglicht es Ihnen, die Datenmenge in kB zu bestimmen, die pro Lesevorgang aus der Datenbank gelesen wird. Ist dieser Wert zu klein, dauert die Datenübertragung zu lange, ist der Wert zu groß, kann der JVM der Speicher ausgehen. Wir empfehlen eine "fetchsize" von ca. 1000-2000.

Einfügen von Daten in die Datenbank

Anstatt einzelner Einfüge-Operationen wie z.B. "INSERT INTO t VALUES 1, 2, ..." sollten Sie die effizientere Schnittstelle über Prepared Statements mit Parametern nutzen. Prepared Statements erreichen die beste Performance bei Parameter-Sets zwischen 500 kB und 20 MB. Fügen Sie darüber hinaus nach Möglichkeit die Daten in Ihren nativen Datentypen ein.

Ressourcenfreigabe

Nicht mehr benötigte Ressourcen sollten sofort freigegeben werden, beispielsweise Prepared Statements mittels "close()".

Verbindungsknoten

Geben Sie in den Verbindungsdaten keinen unnötig großen IP Adressbereich an. Da diese Adressen zufällig nacheinander ausprobiert werden, bis eine Verbindung geöffnet werden kann, würde ansonsten das "connect" eventuell deutlich verlangsamt werden.

4.4. ADO.NET Data Provider

Für die Anbindung von Exasol an .NET-Anwendungen wird der ADO.NET Data Provider zur Verfügung gestellt.

4.4.1. Unterstützte Standards, Systemvoraussetzungen und Installation

Der Exasol Data Provider unterstützt den ADO.NET 2.0-Standard und wurde auf den folgenden Systemen erfolgreich getestet:

- Windows 10 (x86/x64)
- Windows 7, Service Pack 1 (x86/x64)
- Windows Server 2012 R2 (x86/x64)
- Windows Server 2012 (x86/x64)
- Windows Server 2008 R2, Service Pack 1 (x86/x64)
- Windows Server 2008, Service Pack 2 (x86/x64)

 Auf dem System muss Microsoft .NET Framework 4.0 Client Profile™ installiert sein.

Der Data Provider steht in Form einer ausführbaren Installer-Datei zum Download bereit. Die Installation erfordert Administrator-Rechte und wird durch Doppelklick auf die heruntergeladene Datei gestartet. Hierbei wird der Data Provider im Global Assembly Cache installiert sowie in der globalen Konfigurationsdatei des .NET-Framework eingetragen.

 Bei der Installation wird zusätzlich zum ADO.NET Treiber das Werkzeug Data Provider Metadata View™(DPMV) mitinstalliert, mit dem die Konnektivität zur Datenbank sowie einfache Metadaten-Abfragen ermöglicht werden.

Neben dem eigentlichen ADO.NET Data Provider werden für folgende Software entsprechende Erweiterungen mitinstalliert:

SQL Server Integration Services	Installation der Data Destination als Datenflussziel für Integration Services Projekte für: <ul style="list-style-type: none"> • Visual Studio 2005 mit SQL Server 2005 (v 9.0) • Visual Studio 2008 mit SQL Server 2008 (v 10.0) • Neuere Versionen funktionieren mit dem generischen ADO.NET und ODBC und benötigen daher keine Data Destination
SQL Server Reporting Services	Installation der Data Processing Extension sowie Anpassung der Konfiguration des Report Servers für: <ul style="list-style-type: none"> • Visual Studio 2005 mit SQL Server 2005 (v 9.0) • Visual Studio 2008 mit SQL Server 2008 (v 10.0) • Visual Studio 2010 mit SQL Server 2012 (v 11.0) • Visual Studio 2012 mit SQL Server 2012 (v 11.0) • Visual Studio 2013 mit SQL Server 2014 (v 12.0)
SQL Server Analysis Services	Installation des DDEX Provider und Pluggable SQL Cartridge für: <ul style="list-style-type: none"> • Visual Studio 2008 mit SQL Server 2008 (v 10.0) • Visual Studio 2010 mit SQL Server 2012 (v 11.0) • Visual Studio 2012 mit SQL Server 2012 (v 11.0) • Visual Studio 2013 mit SQL Server 2014 (v 12.0) • Visual Studio 2015 mit SQL Server 2016 (v 13.0) • Visual Studio 2013 mit SQL Server 12.0

4.4.2. Benutzung des Data Providers

Um den Data Provider mit einer Clientanwendung zu benutzen, muss er zunächst in der Anwendung ausgewählt werden. Dies geschieht normalerweise durch Auswahl des Eintrags "Exasol Data Provider" aus der Liste der installierten Data Provider oder durch Eingabe eines *Invariante* genannten Bezeichners, mit dem der Data Provider ausgewählt wird. Die Invariante des Exasol Data Providers ist "Exasol.EXADataProvider".

Um eine Verbindung mit Exasol herstellen zu können, muss dem Data Provider eine Verbindungszeichenkette mit allen Informationen zum Verbindungsauflaufbau von der Clientanwendung übergeben werden. Die Verbindungszeichenkette ist eine Folge von Schlüsselwort-Wert-Paaren, die durch Semikolon voneinander getrennt sind. Ein typisches Beispiel für eine Verbindungszeichenkette für den Exasol Data Provider ist:

```
host=192.168.6.11..14:8563;UID=sys;PWD=exasol;Schema=test
```

Der Exasol Data Provider unterstützt die folgenden Schlüsselwörter:

Tabelle 4.6. Schlüsselwörter in der ADO.NET Data Provider-Verbindungszeichenkette

Schlüsselwort	Bedeutung
server oder host	Verbindungsknoten zum Exasol-Cluster sowie der zugehörige Port (z.B. 192.168.6.11..14:8563)
port	Port für die Exasol-Verbindung. Dieser Port wird verwendet, falls im Parameter host nicht bereits einer angegeben wurde.
user id , username oder uid	Benutzername
password oder pwd	Passwort
schema	Schema, das beim Login geöffnet werden soll.
autocommit	Einstellung für Autocommit: ON (Default) oder OFF.
encryption	Einstellung für die automatische Verschlüsselung: ON (Default) oder OFF.
logfile	Datei, in die Log-Informationen des Treibers geschrieben werden (z.B. <code>logfile='C:\tmp\ado.log'</code>). Da das Logfile schnell sehr groß werden kann, ist dieser Parameter nur temporär zur Analyse von Problemen zu empfehlen.
onconnect	SQL-String, der direkt nach der Verbindung ausgeführt werden soll. Falls bei der Ausführung ein Fehler auftritt, wird die Verbindung abgebrochen.
connecttimeout	Maximale Zeit in Millisekunden, die der Treiber für den Aufbau einer TPC-Verbindung zu einem Server wartet. Dieser Timeout ist gerade für größere Cluster mit mehreren Ersatzservern interessant, um die Login-Dauer zu minimieren. Default: 2000
querytimeout	Definiert, wie lange eine Query maximal in Sekunden dauern darf, bevor sie automatisch abgebrochen wird. Default ist 0 (unendlich)
superconnection	Erlaubt es, Statements auch dann auszuführen, wenn die maximal erlaubte Anzahl an aktiven Sessions (die eine Anfrage ausführen) erreicht wurde. Werte: ON oder OFF (Default)
	 Dieser Parameter kann nur durch Nutzer SYS gesetzt werden.
	 superconnection sollte nur im Falle von schwerwiegenden Performance-Problemen benutzt werden, bei denen es nicht mehr möglich ist, sich innerhalb einer vernünftigen Zeit auf dem System anzumelden und Anfragen zu starten. Durch diesen Parameter wird es auch in solchen Fällen ermöglicht, das System zu analysieren und die Prozesse zu beenden, die für die Überlastung verantwortlich sind.

Ein Codebeispiel für den Exasol Data Provider (in C# geschrieben):

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.Common;

namespace ConnectionTest
{
    class Program
    {
        static void Main(string[] args)
        {
            DbProviderFactory factory=null;
```

```
try
{
    factory = DbProviderFactories.GetFactory("Exasol.EXADataProvider");
    Console.WriteLine("Found Exasol driver");

    DbConnection connection = factory.CreateConnection();
    connection.ConnectionString =
        "host=192.168.6.11..14:8563;UID=sys;PWD=exasol;Schema=sys";

    connection.Open();
    Console.WriteLine("Connected to server");
    DbCommand cmd = connection.CreateCommand();
    cmd.Connection = connection;
    cmd.CommandText = "SELECT * FROM CAT";

    DbDataReader reader = cmd.ExecuteReader();

    Console.WriteLine("Schema SYS contains:");
    while (reader.Read())
    {
        Console.WriteLine("{0}, {1}",
            reader[ "TABLE_NAME" ],
            reader[ "TABLE_TYPE" ]);
    }

    reader.Close();
    connection.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
}
```

Wird dieses Beispiel-Programm erfolgreich ausgeführt, wird folgende Ausgabe erzeugt:

```
Found Exasol driver
Connected to server
Schema SYS contains:
EXA_SQL_KEYWORDS, TABLE
DUAL, TABLE
EXA_PARAMETERS, TABLE
...
```

Schema Collections

Der Exasol ADO.NET Treiber unterstützt folgende Schema Collections:

- MetaDataCollections
- DataSourceInformation
- DataTypes
- Restrictions
- ReservedWords
- Tables
- Columns

- Procedures
- ProcedureColumns
- Views
- Schemas

Best-Practice für Entwickler

• Einfügen von Daten in die Datenbank

Anstatt einzelner Einfüge-Operationen wie z.B. "INSERT INTO t VALUES 1, 2, ..." sollten Sie die effizientere Schnittstelle über Prepared Statements mit Parametern nutzen. Prepared Statements erreichen die beste Performance bei Parameter-Sets zwischen 500 **KB** und 20 **MB**. Fügen Sie darüber hinaus nach Möglichkeit die Daten in Ihren nativen Datentypen ein. Wir empfehlen zum Ausführen von Prepared Statements die Schnittstelle "IParameterTable". Eine "ParameterTable" Instanz wird von einem "Command" erzeugt mit der Methode "CreateParameterTable()".

• Blockweises Update mit Hilfe von Prepared Statements

Da es nicht möglich ist, Zeilen in Exasol mit Hilfe der ADO.NET spezifischen Schnittstellen **DataSet** und **DataAdapter** zu verändern, stellt Exasol ADO.NET den Benutzern eine eigene Schnittstelle zum Ausführen von Prepared Statements zur Verfügung, die Klasse **ParameterTable**.

Das Ausführen von parametrisierten Queries in Exasol geschieht über SQL Statements mit Fragezeichen an den Stellen der Parameter. Die Werte der **ParameterTable** werden dann bei der Ausführung des Statements für die Parameter verwendet. Beispiel einer parametrisierten Anfrage:

```
using (EXACommand command = (EXACommand)connection.CreateCommand())
{
    command.CommandText = "CREATE OR REPLACE TABLE TEST (I INT, TS TIMESTAMP)";
    command.ExecuteNonQuery();
    command.CommandText = "INSERT INTO TEST VALUES (?, ?)";
    command.Prepare();
    IParameterTable tab = command.CreateParameterTable();
    tab.AddColumn(DbType.Int32);
    tab.AddColumn(DbType.String);
    for (int i = 0; i < 10; i++)
    {
        int currentRow = tab.AppendRow();
        tab[0].setInt32(currentRow, i);
        tab[1].setString(currentRow, "2017-06-30 11:21:13." + i);
    }
    command.Execute(tab);
}
```

 Für die Typdefinition von Dezimalwerten existieren zwei Möglichkeiten:

- **AddColumn(DbType.Decimal)**: Bei Verwendung dieser generellen Methode wird intern der Typ DECIMAL(36,8) angelegt.
- **AddDecimalColumn(p,s)**: Bei Verwendung dieser speziellen Methode wird intern der Typ DECIMAL(p,s) verwendet (maximale Präzision ist 38).

• Dezimaltypen

Der ADO.NET Datentyp **DECIMAL** kann nur Werte mit der Präzision von maximal 28 Ziffern speichern. **DECIMAL**-Spalten in Exasol mit größerer Präzision passen nicht in diesen Datentyp und müssen als Strings übergeben werden.

- **Spaltennamen**

Bitte beachten Sie bei der Verwendung von Spaltennamen die Groß- und Kleinschreibung.

- **Batch-Ausführung**

Falls Sie mehrere Statements als Batch verarbeiten wollen, können Sie dies mit den Funktionen `AddBatchCommandText()` und `ExecuteBatch()` der Klasse `EXACommand` erreichen. Beispiel einer Batch-Ausführung:

```
using (EXACommand cmd = (EXACommand)connection.CreateCommand())
{
    cmd.AddBatchCommandText("CREATE OR REPLACE TABLE TEST (I INT)");
    cmd.AddBatchCommandText("INSERT INTO TEST VALUES (1)");
    cmd.AddBatchCommandText("SELECT * FROM TEST");
    DbDataReader reader = cmd.ExecuteBatch();
    do
    {
        if (reader.RecordsAffected>0)
            Console.Out.WriteLine("Records affected: " + reader.RecordsAffected);
    } while (reader.NextResult());
}
```

4.4.3. Exasol Data Destination

Die Exasol Data Destination implementiert die Schnittstelle "PipelineComponent" für Microsoft SQL Server Integration Services. Über die Data Destination können Daten aus beliebigen Datenflussquellen in Exasol eingefügt werden.

Erstellung einer DataReaderSource

- Legen Sie ein neues "Integration Services Project" in Visual Studio an.
- Erstellen Sie einen neuen Datenfluss und fügen Sie eine "Data Reader Quelle" aus dem Toolbox-Abschnitt "Datenflussquellen" in den Datenfluss ein.
- Erstellen Sie eine neue "ADO.NET Verbindung" im "Verbindungsmanager" und wählen Sie aus der Liste der .NET Anbieter den Exasol Data Provider.
- Konfigurieren Sie den Exasol Data Provider und testen Sie anschließend die Verbindung.
- Öffnen Sie die neue "Data Reader Quelle" aus dem Datenfluss mittels Doppelklick.

Verbindungsmanager

Wählen Sie die neu erstellte Verbindung aus

Komponenteneigenschaften

Im Abschnitt "Benutzerdefinierte Eigenschaften" können Sie das SQL eintragen, das die Daten aus Exasol lesen wird (z.B. "select * from my_schema.my_table").

Spaltenzuordnungen

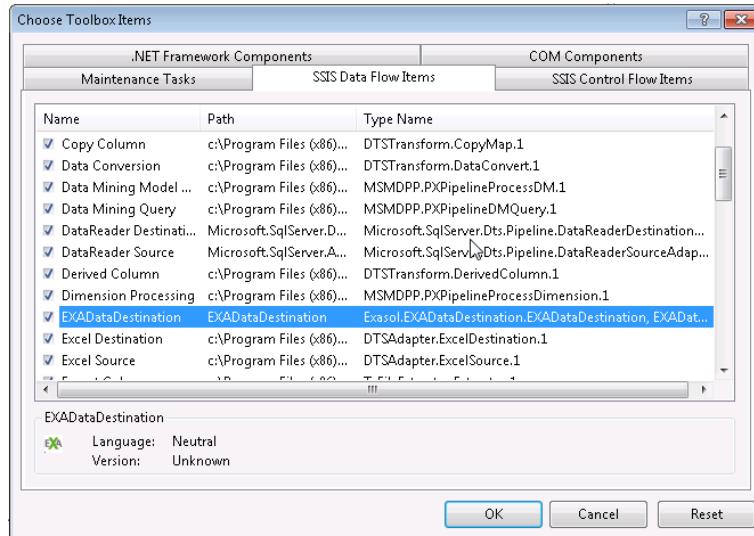
Spaltenzuordnungen betrachten und konfigurieren.

Eingabe- und Ausgabeeigenschaften

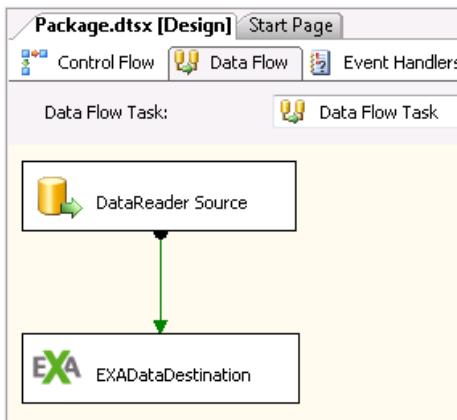
Spalten- Typen und Namen anpassen.

Hinzufügen und Verwenden einer EXADestination

- Öffnen Sie in der Toolbox das Kontextmenü zum Element "Datenflussziele" mittels rechtem Mausklick und wählen Sie den Punkt "Elemente auswählen ..." aus.
- Kreuzen Sie EXADestination an, um die Exasol Data Destination anschließend in der Toolbox verwenden zu können.



- Erzeugen Sie eine Instanz des EXADestination, indem Sie das Objekt mit der Maus aus der Toolbox auf die Arbeitsfläche ziehen.
- Verbinden Sie die Ausgabe der Datenquelle (grün) mit der EXADestination.



- Den Konfigurationsdialog der EXADestination öffnen Sie mittels Doppelklick.

Verbindungsmanager

Selektieren Sie hier eine gültige Exasol ADO.NET Verbindung.

Komponenteneigenschaften

Exasol-spezifische Eigenschaften des Datenflussziels:

- AppendTable: Option, um in existierende Tabellen einzufügen zu können
- Create or replace table: Option, um die Tabelle zuersetzen, falls sie schon existiert
- Insert columns by name: Nutzen Sie diese Option, falls sich die Reihenfolge der Spalten in Quelle und Ziel unterscheiden
- Log file name: Name der Logdatei zu Debug-Zwecken
- TableName: Name der Zieltabelle in Exasol Server (kann auch schemaqualifiziert sein, z.B. my_schema.my_table).

Eingabespalten

Hier werden die zu übertragenen Spalten ausgewählt.

Eingabe- und Ausgabeeigenschaften

Hier können weitere Optionen pro Spalte eingestellt werden, zum Beispiel Name und Datentyp.



Den Datentyp zu ändern kann sehr nützlich sein, falls die ursprünglichen Daten vom Typ Decimal mit Precision größer als 28 sind. Sol-

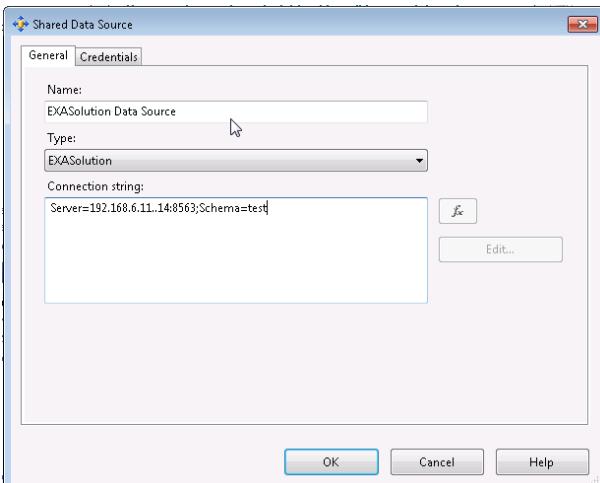
che Daten können nach Exasol transferiert werden, indem der Datentyp z.B. in "varchar(40)" geändert wird.

- Sobald das Datenflussziel konfiguriert ist, kann der Datenfluss getestet bzw. gestartet werden. Bei erfolgreichem Ablauf färben sich die Datenflusselemente grün.

4.4.4. Exasol Data Processing Extension

Die EXADataProcessingExtension implementiert Schnittstellen für die Microsoft SQL Server Reporting Services, um Berichte mit Exasol erstellen und auszuführen zu können. Im folgenden Beispiel wird mit Hilfe von Visual Studio 2005 ein Bericht für die Reporting Services erstellt:

- Starten Sie Visual Studio 2005, erstellen Sie ein neues Berichtserverprojekt und öffnen Sie den Projektmappen-Explorer.
- Legen Sie eine neue Datenquelle (EXADatasource) an und geben dort den Connection String sowie Benutzer und sein Passwort (im Tab "Anmeldeinformationen") an.



- Im Projektmappen-Explorer fügen Sie einen neuen Bericht mit dem Assistenten hinzu. Beim Auswahl der Datenquelle wählen Sie eine ExaDataSource, die zuvor konfiguriert wurde.
- Sobald der Bericht fertig ist, muss dieser erstellt und bereitgestellt werden. Beide Funktionen sind im Menü "Erstellen" enthalten.
- Der Bericht kann jetzt in Ihrem Browser ausgeführt werden. Starten Sie dafür den Browser (z.B. Internet Explorer) und geben Sie die Adresse des Report-Servers ein (z.B. "http://localhost/Reports"). Vergewissern Sie sich bei Ihrem Systemadministrator, dass Sie die nötigen Berechtigungen im Report-Server haben, um Berichte auszuführen.
- Nach der Ausführung könnte die Ausgabe im Browser so aussehen:

The screenshot shows a Windows Internet Explorer window titled "Report Manager - Windows Internet Explorer". The URL in the address bar is <http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fExaBericht%2fReport1>. The page displays the "Report Manager" interface for SQL Server Reporting Services. The breadcrumb navigation shows "Home > ExaBericht > Report1". Below the navigation, there are tabs: View, Properties, History, and Subscriptions, with "View" being the active tab. A sub-header "Report1" is displayed above a table. The table has two columns: "TABLE TYPE" and "TABLE NAME". The data rows are:

TABLE TYPE	TABLE NAME
TABLE	BO_CURR_EX
TABLE	CHARTS
TABLE	PROD

At the bottom of the report area, there is a navigation bar with icons for back, forward, search, and a dropdown menu labeled "Select a format".

4.5. WebSockets

Das [JSON](#) via WebSockets Client Server Protokoll erlaubt es Kunden, Ihre eigenen Treiber für alle möglichen Plattformen zu entwickeln, unter Nutzung eines verbindungsorientierten Web-Protokolls. Hauptvorteil ist die Flexibilität hinsichtlich der verwendeten Programmiersprache, in der Sie Exasol integrieren wollen. Zudem stellt dies einen nativen Zugriff dar als Standard-Protokolle für den Datenbank-Zugriff, wie [JDBC](#), [ODBC](#) oder ADO.NET. Diese sind zumeist alte, statische Standards und verursachen Komplexität aufgrund der Notwendigkeit eines Treiber-Managers.

Falls Sie interessiert sind, mehr über unsere WebSockets API zu erfahren, können Sie Details über die API Spezifikation sowie Beispiel-Implementierungen wie einen nativen Python-Treiber in unserem Open-Source GitHub Repository finden (<https://www.github.com/exasol>). Und wir würden uns sehr freuen, wenn auch Sie aktiv in unserer Open Source Community mitwirken und unsere Open Source Tools nutzen, erweitern und ergänzen.

4.6. SDK

Exasol stellt für Client-Anwendungsprogrammierer ein SDK (Software Development Kit) zur Verfügung, mit dem verschiedene Applikationen auf Exasol zugreifen können. Sie finden das Paket im Download-Bereich auf unserer Webseite und finden nach dem installieren/entpacken folgende Unterordner:

R und Python	R- bzw. Python-Packages zur einfachen Ausführung von R/Python-Code in der Datenbank, gestartet aus Ihrer gewohnten Programmierumgebung heraus. Weitere Infos finden Sie in den entsprechenden Readme-Dateien.
CLI	Das Call Level Interface, mit dessen Hilfe C++ Programme entwickelt werden können. Ausführliche Details hierzu finden Sie im nächsten Unterkapitel.

4.6.1. Call Level Interface (CLI)

Die native C++ Schnittstelle

Die native Schnittstelle (Exasol [CLI](#)) implementiert den größten Teil der ODBC Variante des Call Level Interfaces Standards. Die Schnittstellen-Funktionen sind den ODBC-Funktionen sehr ähnlich. Die meisten unterscheiden sich von den ODBC-Funktionen lediglich durch den Präfix "EXA".

Anmerkungen zur Exasol CLI Schnittstelle:

- Exasol CLI ist schneller als ODBC und JDBC
- Es werden spezielle Exasol-Funktionalitäten unterstützt (z.B. Batch-Execution, Parallel-Read, Parallel-Insert)
- Die Exasol CLI Library ist für Windows und Linux erhältlich
- Jeder ODBC Programmierer wird mit der Exasol CLI Schnittstelle in kürzester Zeit vertraut sein

CLI für Windows

Lieferumfang und Systemvoraussetzungen

Im ausgepackten CLI-Ordner ist Folgendes enthalten:

- Exasol CLI Libraries: EXACLI.lib sowie die dynamischen Libraries (*.dll) im Ordner "lib32" bzw. "lib64", übersetzt für 32- und 64-Bit Windows.
- Die Header-Dateien "exaCInterface.h" und "exaDefs.h". Sie enthalten Deklarationen und Konstanten der Exasol CLI Schnittstelle.

- Ein Beispielprogramm, das die Einbindung der CLI Schnittstelle verdeutlicht. Das Beispiel wird als Quellcode (.cpp) mitgeliefert. Zusätzlich enthält das Paket eine passende Projektdatei für Visual Studio 10 (.vcproj).

Das CLI ist sowohl für die 32-Bit Version als auch für die 64-Bit Version von Windows verwendbar und wurde auf folgenden Systemen erfolgreich getestet:

- Windows 10 (x86/x64)
- Windows 7, Service Pack 1 (x86/x64)
- Windows Server 2012 R2 (x86/x64)
- Windows Server 2012 (x86/x64)
- Windows Server 2008 R2, Service Pack 1 (x86/x64)
- Windows Server 2008, Service Pack 2 (x86/x64)

 Auf dem System muss Microsoft .NET Framework 4.0 Client Profile™ installiert sein.

Übersetzen und Ausführen des Windows-Beispielprogramms

Sie benötigen hierfür Visual C++ von Visual Studio 10 oder neuer.

Öffnen Sie die Projektdatei (.vcproj) aus dem Verzeichnis "examples\sqlExec\" mit Visual Studio. Die Beispiel-Sourcen (.cpp) sind bereits im Projekt eingebunden. Die Lib und Include Pfade zeigen auf die entsprechenden Verzeichnisse, die mit dem CLI installiert wurden. Falls Dateien aus dem CLI-Lieferumfang verschoben oder verändert wurden, muss das Projekt entsprechend angepasst werden.

Das Beispiel-Programm kann für 32- und 64-Bit als Debug- und Release-Version übersetzt werden. Die erzeugte .exe Datei ist eine Windows-Konsolenanwendung. Im Quellcode des Beispielprogramms sind Kommentare enthalten, die die Funktionsweise der Anwendung verdeutlichen.

Ein Aufruf des Beispielprogramms "sqlExec.exe" könnte so aussehen:

```
echo select * from exa_syscat | sqlExec.exe -execDirect -u sys -P exasol
                                         -c 192.168.6.11..14:8563
```

Dabei wird der SQL-String "select * from exa_syscat" über Standard-In an das Beispielprogramm übergeben und über die CLI-Funktion "EXAExecDirect()" in Exasol ausgeführt.

CLI für Linux/Unix

Lieferumfang



Bitte lesen Sie zur Konfiguration unbedingt die im Installationspaket enthaltene Datei README.txt.

Das CLI enthält:

- Exasol CLI Libraries: Bibliotheken für dynamisches Linken (libexacli-uo2212.so bzw. libexacli-uo2214.so, jeweils angelehnt an die beiden unixODBC Versionen 2.2.12 und 2.2.14), übersetzt mit gcc 4.1.0.
- Die Header-Dateien "exaCInterface.h" und "exadefs.h". Sie enthalten die Deklarationen und Konstanten der Exasol CLI-Schnittstelle.
- Ein voll funktionsfähiges Beispielprogramm, das die Einbindung der CLI-Schnittstelle verdeutlicht. Das Beispiel wird als Quellcode (.cpp) samt passendem Makefile mitgeliefert.

Systemvoraussetzungen

Das Exasol CLI wurde auf folgenden Systemen getestet:

- Red Hat / CentOS 7 (x64)
- Red Hat / CentOS 6 (x86/x64)
- Debian 8 (x86/x64)
- Ubuntu 16.04 LTS (x86/64)
- Ubuntu 14.04 LTS (x86/64)
- SUSE Linux Enterprise Server 12 (x64)
- SUSE Linux Enterprise Desktop 12 (x64)
- SUSE Linux Enterprise Server 11 (x86/x64)
- openSUSE Leap 42.2 (x64)

- FreeBSD 11.0 (64Bit)
- FreeBSD 10.3 (64Bit)

Übersetzen und Ausführen des Linux-Beispielprogramms

Sie können das Beispielprogramm mit dem im Ordner `examples` liegenden Makefile übersetzen. Hierzu empfehlen wir GCC 4.1.0 oder neuer.

Ein Aufruf des Beispielprogramms "exaexec" könnte folgendermaßen aussehen:

```
echo "select * from exa_syscat" | ./exaexec -execDirect -u sys -P exasol
                                              -c 192.168.6.11..14:8563
-----
RESULTCOLS=3
ROWS=83
PARAMCOUNT=0
```

Dieses Beispiel führt das angegebene SQL statement (`select * from exa_syscat`) aus und gibt die Anzahl der Zeilen und Spalten des Ergebnisses aus.

Beispiel

Folgendes Beispiel illustriert, wie eine SQL-Anfrage über die CLI-Schnittstelle an die Datenbank gerichtet werden kann. Rückgabewerte der Funktionen werden dabei nicht ausgewertet, um den Code zu vereinfachen.

```
// Analog zur ODBC-Schnittstelle werden vor dem Aufbau einer Verbindung
// zur Exasol Handles angelegt (für Environment und Connection)
SQLHENV henv;
SQLHDBC hdbc;
EXAAllocHandle(SQL_HANDLE_ENV, NULL, &henv);
EXAAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

// Stelle die Verbindung zum Cluster aus 3 Knoten her
EXAServerConnect(hdbc, "192.168.6.11..14:8563", SQL_NTS, NULL,
                  0, "sys", SQL_NTS, "exasol", SQL_NTS);

// Lege Handle für die SQL Befehle an
SQLHSTMT hstmt;
EXAAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

// Erzeuge das Schema "TEST"
EXAExecDirect(hstmt, (SQLCHAR*)"CREATE SCHEMA test", SQL_NTS);

// Gebe die Ressourcen frei und schließe die Verbindung
EXAFreeHandle(SQL_HANDLE_STMT, hstmt);
EXADisconnect(hdbc);
```

```
EXAFreeHandle(SQL_HANDLE_DBC, hdbc);  
EXAFreeHandle(SQL_HANDLE_ENV, henv);
```

Best-Practice für Entwickler

Lesen großer Datenmengen

Verwenden Sie niemals `SQLGetData`, sondern `SQLBindCol` und `SQLFetch`.

Zur optimalen Performance sollten Sie außerdem die Anzahl der Zeilen je `SQLFetch` so wählen, dass sie einer Datenmenge von 50-100 **MB** entspricht.

Einfügen von Daten in die Datenbank

Anstatt einzelner Einfüge-Operationen wie z.B. "INSERT INTO t VALUES 1, 2, ..." sollten Sie die effizientere Schnittstelle über Prepared Statements mit Parametern nutzen. Prepared Statements erreichen die beste Performance bei Parameter-Sets zwischen 50 und 100 **MB**.

Fügen Sie außerdem nach Möglichkeit die Daten in Ihren nativen Datentypen ein, z.B. die Zahl 1234 als `SQL_C_SLONG` (Integer) anstatt als `SQL_CHAR` ("1234").

Anhang A. Systemtabellen

A.1. Allgemeine Informationen

Exasol verfügt über eine Vielzahl an Systemtabellen, die die Metadaten der Datenbank und den aktuellen Zustand des Systems beschreiben. Diese Systemtabellen befinden sich im Schema „SYS“, werden aber automatisch in den Namensraum integriert. Das heißt, dass sie ohne Angabe des Schemanamens „SYS“ abgefragt werden können, sofern im aktuellen Schema kein Objekt mit demselben Namen existiert. Andernfalls kann über den schemaqualifizierten Namen `SYS.<table_name>` auf die Systemtabellen zugegriffen werden (z.B. "SELECT * FROM SYS.DUAL").

Es gibt einige sicherheitskritische Systemtabellen, auf die nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY" Zugriff erhalten (Benutzer mit Rolle DBA besitzen dieses Privileg implizit). Dazu gehören beispielsweise alle Systemtabellen mit Präfix "EXA_DBA_".

Andererseits existieren Systemtabellen, auf die jeder zugreifen kann, deren Inhalt aber vom aktuellen Benutzer abhängig ist. So werden z.B. in EXA_ALL_OBJECTS nur diejenigen Datenbankobjekte angezeigt, auf die der aktuelle Benutzer Zugriff hat.

A.2. Liste der Systemtabellen

Nachfolgend werden alle in Exasol unterstützten Systemtabellen sowie deren Zugriffsregeln spezifiziert.

A.2.1. Katalog-Systemtabellen

EXA_SYSCAT

Diese Systemtabelle listet alle existierenden Systemtabellen auf.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCHEMA_NAME	Name des Systemschema
OBJECT_NAME	Name der Systemtabelle
OBJECT_TYPE	Art des Objekts: TABLE oder VIEW
OBJECT_COMMENT	Kommentar zum Objekt

A.2.2. Systemtabellen zu Metadaten

EXA_ALL_COLUMNS

Diese Systemtabelle enthält Informationen zu allen Tabellenspalten, auf die der Nutzer zugreifen kann.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle

Spalte	Bedeutung
COLUMN_OBJECT_TYPE	Typ des zugehörigen Objekts
COLUMN_NAME	Name der Spalte
COLUMN_TYPE	Datentyp der Spalte
COLUMN_TYPE_ID	Kennung des Datentyps
COLUMN_MAXSIZE	Maximale Anzahl an Zeichen bei Zeichenketten
COLUMN_NUM_PREC	Precision bei numerischen Werten
COLUMN_NUM_SCALE	Scale bei numerischen Werten
COLUMN_ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_IS_VIRTUAL	Gibt an, ob es sich um die Spalte einer virtuellen Tabelle handelt
COLUMN_IS_NULLABLE	Gibt an, ob Nullwerte erlaubt sind (TRUE bzw. FALSE). Bei Views ist dieser Wert stets NULL.
COLUMN_IS DISTRIBUTION KEY	Gibt an, ob die Spalte Teil des Verteilungsschlüssels ist (TRUE bzw. FALSE).
COLUMN_DEFAULT	Default-Wert der Spalte
COLUMN_IDENTITY	Aktueller Wert des Identity-Zahlengenerators, falls diese Spalte die Identity Eigenschaft besitzt.
COLUMN_OWNER	Besitzer des zugehörigen Objekts
COLUMN_OBJECT_ID	ID der Spalte
STATUS	Status des Objekts
COLUMN_COMMENT	Kommentar zur Spalte

EXA_ALL_CONNECTIONS

Eine Auflistung aller dem System bekannten Verbindungen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONNECTION_NAME	Name der Verbindung
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
CONNECTION_COMMENT	Comment on the connection

EXA_ALL_CONSTRAINTS

Diese Systemtabelle enthält Informationen zu allen Constraints, auf die der Nutzer zugreifen kann.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_ENABLED	Gibt an, ob ein Constraint geprüft wird
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)

EXA_ALL_CONSTRAINT_COLUMNS

Diese Systemtabelle enthält Informationen zu referenzierten Tabellenspalten von allen Constraints, auf die der Nutzer zugreifen kann.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)
ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_NAME	Name der Spalte
REFERENCED_SCHEMA	Referenziertes Schema (nur bei Foreign Keys)
REFERENCED_TABLE	Referenzierte Tabelle (nur bei Foreign Keys)
REFERENCED_COLUMN	Name der Spalte in der referenzierten Tabelle (nur bei Foreign Keys)

EXA_ALL_DEPENDENCIES

Beschreibt alle direkten Abhängigkeiten zwischen Schemaobjekten, auf die der aktuelle Nutzer Zugriff hat. Bitte beachten Sie, dass z.B. Abhängigkeiten zwischen Skripten nicht ermittelt werden können. Bei einer View werden unter Umständen Einträge mit REFERENCE_TYPE=NULL ausgegeben, falls sich unterliegende Objekte geändert haben und auf die View seither nicht mehr zugegriffen wurde.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem sich das Objekt befindet
OBJECT_NAME	Name des abhängigen Objekts
OBJECT_TYPE	Objekttyp
OBJECT_OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
REFERENCE_TYPE	Typ der Referenz (VIEW, CONSTRAINT)
REFERENCED_OBJECT_SCHEMA	Schema, in dem sich das referenzierte Objekt befindet
REFERENCED_OBJECT_NAME	Name des referenzierten Objekts
REFERENCED_OBJECT_TYPE	Objekttyp des referenzierten Objekts
REFERENCED_OBJECT_OWNER	Besitzer des referenzierten Objekts
REFERENCED_OBJECT_ID	ID des referenzierten Objekts

EXA_ALL_FUNCTIONS

Diese Systemtabelle beschreibt alle Funktionen der Datenbank, auf die der aktuelle Nutzer Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
FUNCTION_SCHEMA	Schema der Funktion
FUNCTION_NAME	Name der Funktion
FUNCTION_OWNER	Besitzer der Funktion
FUNCTION_OBJECT_ID	ID der Funktion
FUNCTION_TEXT	Erzeugungstext der Funktion
FUNCTION_COMMENT	Kommentar zur Funktion

EXA_ALL_INDICES

Diese Systemtabelle beschreibt alle Indizes auf Tabellen, auf die der aktuelle Nutzer Zugriff hat. Bitte beachten Sie, dass Indizes automatisch vom System erzeugt und verwaltet werden. Diese Tabelle dient daher in erster Linie der Transparenz.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INDEX_SCHEMA	Schema des Index
INDEX_TABLE	Tabelle des Index
INDEX_OWNER	Besitzer des Index
INDEX_OBJECT_ID	ID des Index
INDEX_TYPE	Index-Typ
MEM_OBJECT_SIZE	Größe des Index in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann der Index erzeugt wurde
LAST_COMMIT	Wann wurde der Index das letzte Mal geändert
REMARKS	Zusätzliche Infos zum Index

EXA_ALL_OBJ_PRIVS

Diese Tabelle beinhaltet alle Objektprivilegien, die auf Objekte der Datenbank vergeben worden sind, auf die der aktuelle User Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Objekts

EXA_ALL_OBJ_PRIVS_MADE

Listet alle Objektprivilegien auf, die der aktuelle Benutzer selbst vergeben hat, oder die auf Objekten liegen, die dem Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Objekts

EXA_ALL_OBJ_PRIVS_REC

Listet alle Objektprivilegien auf, die dem aktuellen Benutzer direkt oder PUBLIC gewährt wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Objekts

EXA_ALL_OBJECTS

Diese Systemtabelle beschreibt alle Datenbankobjekte, auf die der aktuelle Benutzer Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt
OBJECT_COMMENT	Kommentar zum Objekt

EXA_ALL_OBJECT_SIZES

Diese Systemtabelle enthält die Größen aller Datenbank-Objekte, auf die der aktuelle Benutzer Zugriff hat. Die Werte sind rekursiv berechnet, so dass z.B. die Größe eines Schemas die Summe der Größen aller darin enthaltenen Schemaobjekte ist.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
RAW_OBJECT_SIZE	Unkomprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
MEM_OBJECT_SIZE	Komprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts

EXA_ALL_ROLES

Eine Auflistung aller dem System bekannten Rollen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
ROLE_NAME	Name der Rolle
CREATED	Zeitstempel, wann die Rolle erzeugt wurde
ROLE_PRIORITY	Priorität der Rolle
ROLE_COMMENT	Kommentar zur Rolle

EXA_ALL_SCRIPTS

Diese Systemtabelle beschreibt alle Datenbank-Skripte, auf die der aktuelle Nutzer Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCRIPT_SCHEMA	Schema, in dem das Skript liegt
SCRIPT_NAME	Name des Skripts
SCRIPT_OWNER	Besitzer des Skripts
SCRIPT_OBJECT_ID	ID des Skripts
SCRIPT_TYPE	Skript-Typ (PROCEDURE, ADAPTER oder UDF)

Spalte	Bedeutung
SCRIPT_LANGUAGE	Skriptsprache
SCRIPT_INPUT_TYPE	Eingabetyp des Skriptes (SCALAR oder SET im Falle von UDF Skripten, sonst NULL)
SCRIPT_RESULT_TYPE	Rückgabetyp des Skripts (ROWCOUNT oder TABLE bzw. RETURNS oder EMITS)
SCRIPT_TEXT	Kompletter Erzeugungstext des Skripts
SCRIPT_COMMENT	Kommentar zum Skript

EXA_ALL_SESSIONS

Diese Systemtabelle enthält Informationen über Benutzersessions. Unter anderem wird das letzte SQL-Kommando ausgegeben, wobei aus Sicherheitsgründen nur der Statementname angegeben ist. Der vollständige SQL-Text ist in den Tabellen [EXA_USER_SESSIONS](#) und [EXA_DB_SESSIONS](#) zu finden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
USER_NAME	Angemeldeter Benutzer
STATUS	Aktueller Zustand der Session. Die wichtigsten davon sind: IDLE Client ist verbunden, führt aber nichts aus. EXECUTE SQL Ausführung einer einzelnen SQL-Anweisung. EXECUTE BATCH Ausführung einer Serie von SQL-Anweisungen. PREPARE SQL Anlegen eines prepared statement. EXECUTE PREPARED Ausführung eines prepared statements. FETCH DATA Client liest Ergebnisse. QUEUED Client wartet, weil zu viele parallele Anfragen ausgeführt werden.
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
STMT_ID	Fortlaufende ID des Nutzers innerhalb der Session
DURATION	Bisherige Dauer der Statement-Ausführung
QUERY_TIMEOUT	Session-spezifischer Wert für QUERY_TIMEOUT, siehe auch ALTER SESSION bzw. ALTER SYSTEM
ACTIVITY	Information über den aktuellen Zustand der SQL-Bearbeitung
TEMP_DB_RAM	Aktueller temporärer Datenbankspeicherverbrauch in MiB (clusterweit)
LOGIN_TIME	Zeitpunkt des Logins
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist
PRIORITY	Prioritätsgruppe
NICE	NICE-Attribut
RESOURCES	Zugeteilte Ressourcen in Prozent

EXA_ALL_TABLES

Diese Systemtabelle beschreibt alle Tabellen in der Datenbank, auf die der aktuelle Benutzer Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OWNER	Besitzer der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
TABLE_IS_VIRTUAL	Zeigt an, ob es sich um eine virtuelle Tabelle handelt
TABLE_HAS_DISTRIBUTION_KEY	Zeigt an, ob die Tabelle explizit verteilt ist
TABLE_ROW_COUNT	Anzahl Zeilen in der Tabelle
DELETE_PERCENTAGE	Anteil der Zeilen, die nur als gelöscht markiert und noch nicht physisch gelöscht sind (in Prozent)
TABLE_COMMENT	Kommentar zur Tabelle

EXA_ALL_USERS

Diese Tabelle bietet eingeschränkte Informationen über alle dem System bekannten Benutzer.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
USER_NAME	Name des Benutzers
CREATED	Zeitpunkt, zu dem der Benutzer angelegt wurde
USER_PRIORITY	Priorität des Nutzers
USER_COMMENT	Kommentar zum Nutzer

EXA_ALL_VIEWS

Listet alle für den aktuellen Nutzer zugreifbaren Views auf.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
VIEW_SCHEMA	Name des Schemas, in dem die View angelegt wurde
VIEW_NAME	Name der View
SCOPE_SCHEMA	Schema, aus dem heraus die View angelegt wurde
VIEW_OWNER	Besitzer der View
VIEW_OBJECT_ID	Interne ID der View
VIEW_TEXT	Text der View, mit der sie erzeugt wurde
VIEW_COMMENT	Kommentar zur View

EXA_ALL_VIRTUAL_COLUMNS

Diese Systemtabelle enthält die Spalten aller virtuellen Tabellen auf die der aktuelle Benutzer Zugriff hat. Sie zeigt Informationen die spezifisch sind für virtuelle Spalten. Virtuelle Spalten werden darüber hinaus auch in der Tabelle [EXA_ALL_COLUMNS](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle
COLUMN_NAME	Name der Spalte
COLUMN_OBJECT_ID	ID der Spalte
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Spalte merken kann

EXA_ALL_VIRTUAL_SCHEMA_PROPERTIES

Diese Systemtabelle enthält die Eigenschaften aller virtuellen Schemas auf die der aktuelle Benutzer Zugriff hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCHEMA_NAME	Name des Schema-Objekts
SCHEMA_OBJECT_ID	ID des Schema-Objekts
PROPERTY_NAME	Name der Eigenschaft des virtuellen Schemas
PROPERTY_VALUE	Wert der Eigenschaft des virtuellen Schemas

EXA_ALL_VIRTUAL_TABLES

Diese Systemtabelle enthält alle virtuellen Tabellen auf die der aktuelle Benutzer Zugriff hat. Sie zeigt Informationen die spezifisch sind für virtuelle Tabellen. Virtuelle Tabellen werden darüber hinaus auch in der Tabelle [EXA_ALL_TABLES](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
LAST_REFRESH	Zeitpunkt der letzten Aktualisierung der Metadaten
LAST_REFRESH_BY	Name des Nutzers der die letzte Aktualisierung der Metadaten initiiert hat
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Tabelle merken kann

EXA_DBA_COLUMNS

Diese Systemtabelle enthält Informationen zu allen Tabellenspalten.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle
COLUMN_OBJECT_TYPE	Typ des zugehörigen Objekts

Spalte	Bedeutung
COLUMN_NAME	Name der Spalte
COLUMN_TYPE	Datentyp der Spalte
COLUMN_TYPE_ID	Kennung des Datentyps
COLUMN_MAXSIZE	Maximale Anzahl an Zeichen bei Zeichenketten
COLUMN_NUM_PREC	Precision bei numerischen Werten
COLUMN_NUM_SCALE	Scale bei numerischen Werten
COLUMN_ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_IS_VIRTUAL	Gibt an, ob es sich um die Spalte einer virtuellen Tabelle handelt
COLUMN_IS_NULLABLE	Gibt an, ob Nullwerte erlaubt sind (TRUE bzw. FALSE). Bei Views ist dieser Wert stets NULL.
COLUMN_IS_DISTRIBUTION_KEY	Gibt an, ob die Spalte Teil des Verteilungsschlüssels ist (TRUE oder FALSE)
COLUMN_DEFAULT	Default-Wert der Spalte
COLUMN_IDENTITY	Aktueller Wert des Identity-Zahlengenerators, falls diese Spalte die Identity-Eigenschaft besitzt.
COLUMN_OWNER	Besitzer des zugehörigen Objekts
COLUMN_OBJECT_ID	ID der Spalte
STATUS	Status des Objekts
COLUMN_COMMENT	Kommentar zur Spalte

EXA_DBA_CONNECTIONS

Eine Auflistung aller dem System bekannten Verbindungen.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
CONNECTION_NAME	Name der Verbindung
CONNECTION_STRING	Definiert das Ziel der Verbindung
USER_NAME	Nutzername, der beim Aufbau der Verbindung benutzt wird
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
CONNECTION_COMMENT	Comment on the connection

EXA_DBA_CONNECTION_PRIVS

Auflistung aller Verbindungen, die einem Benutzer oder einer Rolle gewährt wurden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
GRANTEE	Name des Benutzers/der Rolle, der das Recht gewährt wurde
GRANTED_CONNECTION	Name der Verbindung, die gewährt wurde
ADMIN_OPTION	Angabe, ob GRANTEE den Zugriff auf die Verbindung an andere Benutzer/Rollen weitergeben darf

EXA_DBA_RESTRICTED_OBJ_PRIVS

Auflistung aller Connections, für die einem Benutzer ein eingeschränktes Zugriffsrecht für bestimmte Skripte gewährt wurde.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_NAME	Name des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_TYPE	Typ des Objekts für das das eingeschränkte Privileg gewährt wurde
FOR_OBJECT_SCHEMA	Schema des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_NAME	Name des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_TYPE	Typ des Objekts auf das das Privileg eingeschränkt wurde
PRIVILEGE	Das Privileg, welches eingeschränkt gewährt wurde
GRANTEE	Name des Benutzer/der Rolle der das Privileg eingeschränkt gewährt wurde
GRANTOR	Name des Benutzer/der Rolle von der das Privileg eingeschränkt gewährt wurde
OWNER	Besitzer des Objekts auf das das Privileg eingeschränkt gewährt wurde

EXA_DBA_CONSTRAINTS

Diese Systemtabelle enthält Informationen zu allen Constraints der Datenbank.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_ENABLED	Gibt an, ob ein Constraint geprüft wird
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)

EXA_DBA_CONSTRAINT_COLUMNS

Diese Systemtabelle enthält Informationen zu referenzierten Tabellenspalten von allen Constraints der Datenbank.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)

Spalte	Bedeutung
ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_NAME	Name der Spalte
REFERENCED_SCHEMA	Referenziertes Schema (nur bei Foreign Keys)
REFERENCED_TABLE	Referenzierte Tabelle (nur bei Foreign Keys)
REFERENCED_COLUMN	Name der Spalte in der referenzierten Tabelle (nur bei Foreign Keys)

EXA_DBA_DEPENDENCIES

Beschreibt alle direkten Abhängigkeiten zwischen Schemaobjekten. Bitte beachten Sie, dass z.B. Abhängigkeiten zwischen Skripten nicht ermittelt werden können. Bei einer View werden unter Umständen Einträge mit REFERENCE_TYPE=NULL ausgegeben, falls sich unterliegende Objekte geändert haben und auf die View seither nicht mehr zugegriffen wurde.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem sich das Objekt befindet
OBJECT_NAME	Name des abhängigen Objekts
OBJECT_TYPE	Objekttyp
OBJECT_OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
REFERENCE_TYPE	Typ der Referenz (VIEW, FUNCTION, SCRIPT oder CONSTRAINT)
REFERENCED_OBJECT_SCHEMA	Schema, in dem sich das referenzierte Objekt befindet
REFERENCED_OBJECT_NAME	Name des referenzierten Objekts
REFERENCED_OBJECT_TYPE	Objekttyp des referenzierten Objekts
REFERENCED_OBJECT_OWNER	Besitzer des referenzierten Objekts
REFERENCED_OBJECT_ID	ID des referenzierten Objekts

EXA_DBA_DEPENDENCIES_RECURSIVE

Beschreibt alle direkten und indirekten Abhängigkeiten zwischen Schemaobjekten (also rekursiv). Bitte beachten Sie, dass z.B. Abhängigkeiten zwischen Skripten nicht ermittelt werden können. Views werden nicht angezeigt, falls sich unterliegende Objekte geändert haben und auf die View seither nicht mehr zugegriffen wurde.



Bitte beachten Sie, dass bei Abfrage dieser Tabelle alle Datenbankobjekte gelesen werden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem sich das Objekt befindet
OBJECT_NAME	Name des abhängigen Objekts
OBJECT_TYPE	Objekttyp
OBJECT_OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts

Spalte	Bedeutung
REFERENCE_TYPE	Typ der Referenz (VIEW, FUNCTION, SCRIPT oder CONSTRAINT)
REFERENCED_OBJECT_SCHEMA	Schema, in dem sich das referenzierte Objekt befindet
REFERENCED_OBJECT_NAME	Name des referenzierten Objekts
REFERENCED_OBJECT_TYPE	Objekttyp des referenzierten Objekts
REFERENCED_OBJECT_OWNER	Besitzer des referenzierten Objekts
REFERENCED_OBJECT_ID	ID des referenzierten Objekts
DEPENDENCY_LEVEL	Hierarchie-Level im Abhängigkeits-Graph

EXA_DBA_FUNCTIONS

Diese Systemtabelle beschreibt alle Funktionen der Datenbank.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
FUNCTION_SCHEMA	Schema der Funktion
FUNCTION_NAME	Name der Funktion
FUNCTION_OWNER	Besitzer der Funktion
FUNCTION_OBJECT_ID	ID der Funktion
FUNCTION_TEXT	Erzeugungstext der Funktion
FUNCTION_COMMENT	Kommentar zur Funktion

EXA_DBA_INDICES

Diese Systemtabelle beschreibt alle Indizes auf Tabellen. Bitte beachten Sie, dass Indizes automatisch vom System erzeugt und verwaltet werden. Diese Tabelle dient daher in erster Linie der Transparenz.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
INDEX_SCHEMA	Schema des Index
INDEX_TABLE	Tabelle des Index
INDEX_OWNER	Besitzer des Index
INDEX_OBJECT_ID	ID des Index
INDEX_TYPE	Index-Typ
MEM_OBJECT_SIZE	Größe des Index in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann der Index erzeugt wurde
LAST_COMMIT	Wann wurde der Index das letzte Mal geändert
REMARKS	Zusätzliche Infos zum Index

EXA_DBA_OBJ_PRIVS

Diese Tabelle beinhaltet alle Objektprivilegien, die auf Objekte der Datenbank vergeben worden sind.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Zielobjekts

EXA_DBA_OBJECTS

Diese Systemtabelle beschreibt alle Datenbank-Objekte.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	Eindeutige ID des Objekts
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt
OBJECT_COMMENT	Kommentar zum Objekt

EXA_DBA_OBJECT_SIZES

Diese Systemtabelle enthält die Größen aller Datenbankobjekte. Die Werte sind rekursiv berechnet, so dass z.B. die Größe eines Schemas die Summe der Größen aller darin enthaltenen Schemaobjekte ist.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
RAW_OBJECT_SIZE	Unkomprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
MEM_OBJECT_SIZE	Komprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt

Spalte	Bedeutung
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts

EXA_DBA_ROLES

Eine Auflistung aller dem System bekannten Rollen.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
ROLE_NAME	Name der Rolle
CREATED	Zeitpunkt des Erstellungsdatums
ROLE_PRIORITY	Priorität der Rolle
ROLE_COMMENT	Kommentar zur Rolle

EXA_DBA_ROLE_PRIVS

Auflistung aller Rollen, die einem Benutzer oder einer Rolle gewährt wurden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
GRANTEE	Name des Benutzers/der Rolle, der das Recht gewährt wurde, GRANTED_ROLE anzunehmen
GRANTED_ROLE	Name der Rolle, die gewährt wurde
ADMIN_OPTION	Angabe, ob GRANTEE das Recht zur Rolle selbst an andere Benutzer/Rollen weitergeben darf

EXA_DBA_SCRIPTS

Diese Systemtabelle beschreibt alle Datenbank-Skripte.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SCRIPT_SCHEMA	Schema, in dem das Skripts liegt
SCRIPT_NAME	Name des Skripts
SCRIPT_OWNER	Besitzer des Skripts
SCRIPT_OBJECT_ID	ID des Skripts
SCRIPT_TYPE	Skript-Typ (PROCEDURE, ADAPTER oder UDF)
SCRIPT_LANGUAGE	Skriptsprache
SCRIPT_INPUT_TYPE	Eingabetyp des Skriptes (SCALAR oder SET im Falle von UDF Skripten, sonst NULL)
SCRIPT_RESULT_TYPE	Rückgabetyp des Skripts (ROWCOUNT oder TABLE bzw. RETURNS oder EMITS)
SCRIPT_TEXT	Kompletter Erzeugungstext des Skripts

Spalte	Bedeutung
SCRIPT_COMMENT	Kommentar zum Skript

EXA_DBA_SESSIONS

Diese Systemtabelle enthält Informationen über Benutzersessions. Unter anderem wird das letzte SQL-Kommando ausgegeben.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
USER_NAME	Angemeldeter Benutzer
STATUS	Aktueller Zustand der Session. Die wichtigsten davon sind: IDLE Client ist verbunden, führt aber nichts aus. EXECUTE SQL Ausführung einer einzelnen SQL-Anweisung. EXECUTE BATCH Ausführung einer Serie von SQL-Anweisungen. PREPARE SQL Anlegen eines prepared statement. EXECUTE PREPARED Ausführung eines prepared statements. FETCH DATA Client liest Ergebnisse. QUEUED Client wartet, weil zu viele parallele Anfragen ausgeführt werden.
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
STMT_ID	Fortlaufende ID des Nutzers innerhalb der Session
DURATION	Bisherige Dauer der Statement-Ausführung
QUERY_TIMEOUT	Session-spezifischer Wert für QUERY_TIMEOUT, siehe auch ALTER SESSION bzw. ALTER SYSTEM
ACTIVITY	Information über den aktuellen Zustand der SQL-Bearbeitung
TEMP_DB_RAM	Aktueller temporärer Datenbankspeicherverbrauch in MiB (clusterweit)
LOGIN_TIME	Zeitpunkt des Logins
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist
HOST	Rechnername oder IP-Adresse, von dem aus sich der Benutzer angemeldet hat.
OS_USER	Benutzername, unter dem sich der Benutzer am Betriebssystem des Rechners, von dem der Login kommt, angemeldet hat
OS_NAME	Betriebssystem des Client-Rechners
SCOPE_SCHEMA	Name des Schemas, in dem sich der Benutzer befindet
PRIORITY	Prioritätsgruppe
NICE	NICE-Attribut
RESOURCES	Zugeteilte Ressourcen in Prozent
SQL_TEXT	SQL-Text des Statements

EXA_DBA_SYS_PRIVS

Diese Tabelle zeigt die gewährten Systemprivilegien zu allen Benutzern und Rollen an.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
GRANTEE	Name des Benutzers/der Rolle, dem das Privileg gewährt wurde.
PRIVILEGE	Systemprivileg, das gewährt wurde.
ADMIN_OPTION	Angabe, ob GRANTEE das Recht selbst weitergeben darf.

EXA_DBA_TABLES

Diese Systemtabelle beschreibt alle Tabellen in der Datenbank.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OWNER	Besitzer der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
TABLE_IS_VIRTUAL	Zeigt an, ob es sich um eine virtuelle Tabelle handelt
TABLE_HAS DISTRIBUTION KEY	Zeigt an, ob die Tabelle explizit verteilt ist
TABLE_ROW_COUNT	Anzahl Zeilen in der Tabelle
DELETE_PERCENTAGE	Anteil der Zeilen, die nur als gelöscht markiert und noch nicht physisch gelöscht sind (in Prozent)
TABLE_COMMENT	Kommentar zur Tabelle

EXA_DBA_USERS

Die Tabelle DBA_USERS enthält die vollständige Information über alle dem System bekannten Benutzer.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
USER_NAME	Name des Benutzers
CREATED	Zeitpunkt des Erstellungsdatums
DISTINGUISHED_NAME	Dient zur Identifizierung gegen einen LDAP -Server
KERBEROS_PRINCIPAL	Principal-Name des Kerberos-Services
PASSWORD	Verschlüsselter Hashwerts des Passworts
USER_PRIORITY	Priorität des Nutzers
USER_COMMENT	Kommentar zum Nutzer

EXA_DBA_VIEWS

Listet alle Views in der Datenbank auf.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
VIEW_SCHEMA	Name des Schemas, in dem die View angelegt wurde

Spalte	Bedeutung
VIEW_NAME	Name der View
SCOPE_SCHEMA	Schema, aus dem heraus die View angelegt wurde
VIEW_OWNER	Besitzer der View
VIEW_OBJECT_ID	Interne ID der View
VIEW_TEXT	Text der View, mit der sie erzeugt wurde
VIEW_COMMENT	Kommentar zur View

EXA_DBA_VIRTUAL_COLUMNS

Diese Systemtabelle listet alle Spalten aller virtuellen Tabellen und zeigt Informationen die spezifisch sind für virtuelle Spalten. Virtuelle Spalten werden darüber hinaus auch in der Tabelle [EXA_DBA_COLUMNS](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle
COLUMN_NAME	Name der Spalte
COLUMN_OBJECT_ID	ID der Spalte
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Spalte merken kann

EXA_DBA_VIRTUAL_SCHEMA_PROPERTIES

Diese Systemtabelle listet alle Eigenschaften für alle virtuelle Schemas in dieser Datenbank. Virtuelle Schemas werden in der Tabelle [EXA_VIRTUAL_SCHEMAS](#) gelistet.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SCHEMA_NAME	Name des Schema-Objekts
SCHEMA_OBJECT_ID	ID des Schema-Objekts
PROPERTY_NAME	Name der Eigenschaft des virtuellen Schemas
PROPERTY_VALUE	Wert der Eigenschaft des virtuellen Schemas

EXA_DBA_VIRTUAL_TABLES

Diese Systemtabelle listet alle virtuellen Tabellen und zeigt Informationen die spezifisch sind für virtuelle Tabellen. Virtuelle Tabellen werden darüber hinaus auch in der Tabelle [EXA_DBA_TABLES](#) gelistet.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
LAST_REFRESH	Zeitpunkt der letzten Aktualisierung der Metadaten

Spalte	Bedeutung
LAST_REFRESH_BY	Name des Nutzers der die letzte Aktualisierung der Metadaten initiiert hat
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Tabelle merken kann

EXA_LOADAVG

Diese Systemtabelle enthält Informationen über die aktuelle CPU-Last der einzelnen Exasol Knoten.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
IPROC	Nummer des Knotens
LAST_1MIN	Mittlere Last innerhalb der letzten Minute
LAST_5MIN	Mittlere Last innerhalb der letzten 5 Minuten
LAST_15MIN	Mittlere Last innerhalb der letzten 15 Minuten
RUNNING	Enthält zwei durch „/“ getrennte Zahlen. Die erste gibt an, wie viele Prozesse oder Threads zum Zeitpunkt der Auswertung aktiv sind. Die zweite Zahl gibt die Anzahl aller vom Betriebssystem des Knotens verwalteten Prozesse und Threads an.

EXA_METADATA

Diese Systemtabelle enthält Informationen, die die Eigenschaften der Datenbank beschreiben.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
PARAM_NAME	Name der Eigenschaft
PARAM_VALUE	Wert der Eigenschaft
IS_STATIC	TRUE Der Wert dieser Eigenschaft bleibt konstant FALSE Der Wert dieser Eigenschaft kann sich jederzeit ändern

EXA_PARAMETERS

Diese Systemtabelle liefert die Datenbankparameter, wobei sowohl systemweite als auch sessionbasierte Informationen angezeigt werden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
PARAMETER_NAME	Name des Parameters
SESSION_VALUE	Wert des Session-Parameters
SYSTEM_VALUE	Wert des System-Parameters

EXA_ROLE_CONNECTION_PRIVS

Auflistung aller Verbindungen, die dem Benutzer indirekt über eine seiner Rollen gewährt wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name der Rolle, der das Recht gewährt wurde
GRANTED_CONNECTION	Name der Verbindung, die gewährt wurde
ADMIN_OPTION	Angabe, ob GRANTEE den Zugriff auf die Verbindung an andere Benutzer/Rollen weitergeben darf

EXA_ROLE_RESTRICTED_OBJ_PRIVS

Auflistung aller Connections, für die einem Benutzer ein eingeschränktes Zugriffsrecht für bestimmte Skripte gewährt wurde. Entweder direkt oder indirekt über eine seiner Rollen.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_NAME	Name des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_TYPE	Typ des Objekts für das das eingeschränkte Privileg gewährt wurde
FOR_OBJECT_SCHEMA	Schema des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_NAME	Name des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_TYPE	Typ des Objekts auf das das Privileg eingeschränkt wurde
PRIVILEGE	Das Privileg, welches eingeschränkt gewährt wurde
GRANTEE	Name des Benutzer/der Rolle der das Privileg eingeschränkt gewährt wurde
GRANTOR	Name des Benutzer/der Rolle von der das Privileg eingeschränkt gewährt wurde
OWNER	Besitzer des Objekts auf das das Privileg eingeschränkt gewährt wurde

EXA_ROLE_OBJ_PRIVS

Listet alle Objektprivilegien auf, die an Rollen des Benutzers vergeben wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Name der Rolle
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Zielobjekts

EXA_ROLE_ROLE_PRIVS

Listet alle Rollen auf, die der aktuelle Benutzer indirekt über weitere Rollen inne hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name der Rolle, über die der Benutzer die Rolle indirekt inne hat.
GRANTED_ROLE	Name der Rolle, die der Benutzer indirekt inne hat
ADMIN_OPTION	Angabe, ob die Rolle weitergegeben werden darf

EXA_ROLE_SYS_PRIVS

Listet alle Systemprivilegien auf, die der aktuelle Benutzer über Rollen inne hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name der Rolle
PRIVILEGE	Name des Systemprivilegs
ADMIN_OPTION	Zeigt an, ob das Privileg weitergegeben werden darf

EXA_SCHEMAS

Diese Systemtabelle listet alle Schemas der Datenbank auf.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCHEMA_NAME	Name des Schema-Objekts
SCHEMA_OWNER	Besitzer des Objekts
SCHEMA_OBJECT_ID	ID des Schema-Objekts
SCHEMA_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Schema handelt
SCHEMA_COMMENT	Kommentar zum Objekt

EXA_SCHEMA_OBJECTS

Listet alle Objekte auf, die im aktuellen Schema existieren.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts

EXA_SESSION_CONNECTIONS

Liste aller Verbindungen, die der aktuelle Benutzer benutzen kann. Die Einträge der Spalten CONNECTION_STRING und USER_NAME werden nur dann angezeigt, falls der Nutzer die Connection ändern kann (siehe auch [ALTER CONNECTION](#)).

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONNECTION_NAME	Name der Verbindung

Spalte	Bedeutung
CONNECTION_STRING	Definiert das Ziel der Verbindung
USER_NAME	Nutzername, der beim Aufbau der Verbindung benutzt wird
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
CONNECTION_COMMENT	Comment on the connection

EXA_SESSION_PRIVS

Listet alle Systemprivilegien, die dem Benutzer aktuell zur Verfügung stehen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
PRIVILEGE	Name des Systemprivileges

EXA_SESSION_ROLES

Liste aller Rollen, die der aktuelle Benutzer inne hat.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
ROLE_NAME	Name der Rolle
ROLE_PRIORITY	Priorität der Rolle
ROLE_COMMENT	Kommentar zur Rolle

EXA_SPATIAL_REF_SYS

Liste der verfügbaren Georeferenzsysteme (Koordinatensysteme) für Geodaten.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SRID	Id des Georeferenzsystems
AUTH_NAME	Authority Name des Georeferenzsystems
AUTH_SRID	Authority-spezifische Id für das Georeferenzsystem
SRTEXT	WKT Beschreibung des Georeferenzsystems
PROJ4TEXT	Parameters für Proj4 Projektionen

EXA_SQL_KEYWORDS

Diese Systemtabelle enthält alle Schlüsselwörter von Exasol.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
KEYWORD	Schlüsselwort
RESERVED	Definiert, ob das Schlüsselwort reserviert ist, also z.B. nicht als SQL Bezeichner benutzt werden kann (siehe auch Abschnitt 2.1.2, SQL-Bezeichner)

EXA_SQL_TYPES

Diese Systemtabelle beschreibt die SQL-Datentypen von Exasol.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TYPE_NAME	Name, wie er im SQL-Standard vorgesehen wurde
TYPE_ID	ID des Datentyps
PRECISION	Bei numerischen Werte die Präzision, bei Zeichenketten und anderen Typen die (maximale) Länge in Bytes.
LITERAL_PREFIX	Präfix, mit dem ein Literal dieses Typs eingeleitet werden muss
LITERAL_SUFFIX	Suffix, mit dem ein Literal dieses Typs beendet werden muss
CREATE_PARAMS	Information, welche Information nötig ist, um eine Spalte dieses Typs anzulegen
IS_NULLABLE	Gibt an, ob Nullwerte erlaubt sind (TRUE bzw. FALSE).
CASE_SENSITIVE	Gibt an, ob der Typ Groß- und Kleinschreibung unterscheidet
SEARCHABLE	Gibt an, wie der Typ in einer WHERE-Klausel verwendet werden kann: 0 Kann nicht durchsucht werden 1 Kann nur mit WHERE .. LIKE durchsucht werden 2 Kann nicht mit WHERE .. LIKE durchsucht werden 3 Kann mit beliebiger WHERE-Klausel durchsucht werden
UNSIGNED_ATTRIBUTE	Gibt an, ob es sich um einen vorzeichenlosen Typ handelt
FIXED_PREC_SCALE	Gibt an, ob es sich um einen Typ mit fixer Darstellung handelt
AUTO_INCREMENT	Gibt an, ob es sich um einen automatisch inkrementierenden Typ handelt
LOCAL_TYPE_NAME	lokalisierter Typname
MINIMUM_SCALE	minimale Anzahl Nachkommastellen
MAXIMUM_SCALE	maximale Anzahl Nachkommastellen
SQL_DATA_TYPE	reserviert
SQL_DATETIME_SUB	reserviert
NUM_PREC_RADIX	Zahlenbasis für Angaben der Precision
INTERVAL_PRECISION	reserviert

EXA_STATISTICS_OBJECT_SIZES

Diese Systemtabelle enthält die Größen aller Statistik-Datenbanktabellen aggregiert nach dem Typ von Statistiken (siehe auch [Abschnitt A.2.3, Statistische Systemtabellen](#)).

Zugriff haben alle Benutzer.

Spalte	Bedeutung	
STATISTICS_TYPE	Statistik-Typ	
	AUDIT	Auditing Daten
	DB_SIZE	Daten über die Größe von Datenbankobjekten
	MONITOR	Monitoring Daten

Spalte	Bedeutung
	OTHER Sonstige Daten, z.B. für interne Optimierungen
	PROFILE Profiling Daten
	SQL Daten über die Ausführung von SQL Befehlen
	SYSTEM_EVENTS Informationen über Systemereignisse
	TRANSACTION_CONFLICTS Informationen über Transaktionskonflikte
	USAGE Daten über die Nutzung des Systems
RAW_OBJECT_SIZE	Unkomprimierte Datenmenge dieses Typs von Statistiken in Bytes.
MEM_OBJECT_SIZE	Komprimierte Datenmenge dieses Typs von Statistiken in Bytes.
AUXILIARY_SIZE	Größe der Hilfsstrukturen wie z.B. Indizes dieses Typs von Statistiken in Bytes.

EXA_TIME_ZONES

Diese Systemtabelle enthält die Liste der unterstützten Zeitzonen-Namen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TIME_ZONE_NAME	Name der Zeitzone

EXA_USER_COLUMNS

Diese Systemtabelle enthält Informationen zu den Tabellenspalten derjenigen Tabellen, die dem aktuellen Nutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle
COLUMN_OBJECT_TYPE	Typ des zugehörigen Objekts
COLUMN_NAME	Name der Spalte
COLUMN_TYPE	Datentyp der Spalte
COLUMN_TYPE_ID	Kennung des Datentyps
COLUMN_MAXSIZE	Maximale Anzahl an Zeichen bei Zeichenketten
COLUMN_NUM_PREC	Precision bei numerischen Werten
COLUMN_NUM_SCALE	Scale bei numerischen Werten
COLUMN_ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_IS_VIRTUAL	Gibt an, ob es sich um die Spalte einer virtuellen Tabelle handelt
COLUMN_IS_NULLABLE	Gibt an, ob Nullwerte erlaubt sind (TRUE bzw. FALSE). Bei Views ist dieser Wert stets NULL.
COLUMN_IS DISTRIBUTION KEY	Gibt an, ob die Spalte Teil des Verteilungsschlüssels ist (TRUE bzw. FALSE)
COLUMN_DEFAULT	Default-Wert der Spalte

Spalte	Bedeutung
COLUMN_IDENTITY	Aktueller Wert des Identity-Zahlengenerators, falls diese Spalte die Identity Eigenschaft besitzt.
COLUMN_OWNER	Besitzer des zugehörigen Objekts
COLUMN_OBJECT_ID	ID der Spalte
STATUS	Status des Objekts
COLUMN_COMMENT	Kommentar zur Spalte

EXA_USER_CONNECTION_PRIVS

Auflistung aller Verbindungen, die dem Benutzer direkt gewährt wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name des Benutzers, der das Recht gewährt wurde
GRANTED_CONNECTION	Name der Verbindung, die gewährt wurde
ADMIN_OPTION	Angabe, ob GRANTEE den Zugriff auf die Verbindung an andere Benutzer/Rollen weitergeben darf

EXA_USER_RESTRICTED_OBJ_PRIVS

Auflistung aller Connections, für die der Benutzer ein eingeschränktes Zugriffsrecht für bestimmte Skripte gewährt wurde.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
OBJECT_SCHEMA	Schema des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_NAME	Name des Objekts für das das eingeschränkte Privileg gewährt wurde
OBJECT_TYPE	Typ des Objekts für das das eingeschränkte Privileg gewährt wurde
FOR_OBJECT_SCHEMA	Schema des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_NAME	Name des Objekts auf das das Privileg eingeschränkt wurde
FOR_OBJECT_TYPE	Typ des Objekts auf das das Privileg eingeschränkt wurde
PRIVILEGE	Das Privileg, welches eingeschränkt gewährt wurde
GRANTEE	Name des Benutzer/der Rolle der das Privileg eingeschränkt gewährt wurde
GRANTOR	Name des Benutzer/der Rolle von der das Privileg eingeschränkt gewährt wurde
OWNER	Besitzer des Objekts auf das das Privileg eingeschränkt gewährt wurde

EXA_USER_CONSTRAINTS

Diese Systemtabelle enthält Informationen zu allen Constraints, die dem aktuellen Nutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_ENABLED	Gibt an, ob ein Constraint geprüft wird
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)

EXA_USER_CONSTRAINT_COLUMNS

Diese Systemtabelle enthält Informationen zu referenzierten Tabellenspalten von allen Constraints, die dem aktuellen Nutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
CONSTRAINT_SCHEMA	Zugehöriges Schema
CONSTRAINT_TABLE	Zugehörige Tabelle
CONSTRAINT_TYPE	Typ des Constraints (PRIMARY KEY, FOREIGN KEY bzw. NOT NULL)
CONSTRAINT_NAME	Name des Constraints
CONSTRAINT_OWNER	Besitzer des Constraints (entspricht dem Besitzer der Tabelle)
ORDINAL_POSITION	Position der Spalte in der Tabelle beginnend bei 1
COLUMN_NAME	Name der Spalte
REFERENCED_SCHEMA	Referenziertes Schema (nur bei Foreign Keys)
REFERENCED_TABLE	Referenzierte Tabelle (nur bei Foreign Keys)
REFERENCED_COLUMN	Name der Spalte in der referenzierten Tabelle (nur bei Foreign Keys)

EXA_USER_DEPENDENCIES

Beschreibt alle direkten Abhängigkeiten zwischen Schemaobjekten, die dem aktuellen Nutzer gehören. Bitte beachten Sie, dass z.B. Abhängigkeiten zwischen Skripten nicht ermittelt werden können. Bei einer View werden unter Umständen Einträge mit REFERENCE_TYPE=NULL ausgegeben, falls sich unterliegende Objekte geändert haben und auf die View seither nicht mehr zugegriffen wurde.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem sich das Objekt befindet
OBJECT_NAME	Name des abhängigen Objekts
OBJECT_TYPE	Objekttyp
OBJECT_OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
REFERENCE_TYPE	Typ der Referenz (VIEW, CONSTRAINT)
REFERENCED_OBJECT_SCHEMA	Schema, in dem sich das referenzierte Objekt befindet
REFERENCED_OBJECT_NAME	Name des referenzierten Objekts

Spalte	Bedeutung
REFERENCED_OBJECT_TYPE	Objekttyp des referenzierten Objekts
REFERENCED_OBJECT_OWNER	Besitzer des referenzierten Objekts
REFERENCED_OBJECT_ID	ID des referenzierten Objekts

EXA_USER_FUNCTIONS

Diese Systemtabelle beschreibt alle Funktionen der Datenbank, die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
FUNCTION_SCHEMA	Schema der Funktion
FUNCTION_NAME	Name der Funktion
FUNCTION_OWNER	Besitzer der Funktion
FUNCTION_OBJECT_ID	ID der Funktion
FUNCTION_TEXT	Erzeugungstext der Funktion
FUNCTION_COMMENT	Kommentar zur Funktion

EXA_USER_INDICES

Diese Systemtabelle beschreibt alle Indizes auf Tabellen, die dem aktuellen Nutzer gehören. Bitte beachten Sie, dass Indizes automatisch vom System erzeugt und verwaltet werden. Diese Tabelle dient daher in erster Linie der Transparenz.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INDEX_SCHEMA	Schema des Index
INDEX_TABLE	Tabelle des Index
INDEX_OWNER	Besitzer des Index
INDEX_OBJECT_ID	ID des Index
INDEX_TYPE	Index-Typ
MEM_OBJECT_SIZE	Größe des Index in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann der Index erzeugt wurde
LAST_COMMIT	Wann wurde der Index das letzte Mal geändert
REMARKS	Zusätzliche Infos zum Index

EXA_USER_OBJ_PRIVS

Diese Tabelle beinhaltet alle Objektprivilegien, die auf Objekte der Datenbank vergeben worden sind, auf die der aktuelle User Zugriff hat (außer durch die Rolle PUBLIC).

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts

Spalte	Bedeutung
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Zielobjekts

EXA_USER_OBJ_PRIVS_MADE

Listet alle Objektprivilegien auf, die auf Objekten liegen, die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Zielobjekts

EXA_USER_OBJ_PRIVS_REC'D

Listet alle Objektprivilegien auf, die dem aktuellen Benutzer direkt gewährt wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_SCHEMA	Schema, in dem das Zielobjekt liegt
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Objekttyp
PRIVILEGE	Das vergebene Recht
GRANTEE	Empfänger des Rechts
GRANTOR	Name des Benutzers, der das Recht vergeben hat
OWNER	Besitzer des Zielobjekts

EXA_USER_OBJECTS

Diese Systemtabelle listet alle Objekte auf, die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
CREATED	Zeitstempel, wann das Objekt erzeugt wurde

Spalte	Bedeutung
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt
OBJECT_COMMENT	Kommentar zum Objekt

EXA_USER_OBJECT_SIZES

Enthält die Größen aller Datenbankobjekte, die dem aktuellen Benutzer gehören. Die Werte sind rekursiv berechnet, so dass z.B. die Größe eines Schemas die Summe der Größen aller darin enthaltenen Schemaobjekte ist.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
OBJECT_NAME	Name des Objekts
OBJECT_TYPE	Typ des Objekts
RAW_OBJECT_SIZE	Unkomprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
MEM_OBJECT_SIZE	Komprimierte Datenmenge des Objekts in Bytes (beim letzten COMMIT)
CREATED	Zeitstempel, wann das Objekt erzeugt wurde
LAST_COMMIT	Wann wurde das Objekt das letzte Mal in der DB geändert
OWNER	Besitzer des Objekts
OBJECT_ID	ID des Objekts
OBJECT_IS_VIRTUAL	Gibt an, ob es sich um ein virtuelles Objekt handelt
ROOT_NAME	Name des enthaltenden Objekts
ROOT_TYPE	Typ des enthaltenden Objekts
ROOT_ID	ID des enthaltenden Objekts

EXA_USER_ROLE_PRIVS

Diese Tabelle listet alle Rollen auf, die dem aktuellen Benutzer direkt gewährt wurden (nicht über andere Rollen).

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name des Benutzers
GRANTED_ROLE	Name der Rolle, die er inne hat
ADMIN_OPTION	Angabe, ob die Rolle auch weitergegeben werden darf

EXA_USER_SCRIPTS

Diese Systemtabelle beschreibt alle Datenbank-Skripte, die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCRIPT_SCHEMA	Schema, in dem das Skript liegt
SCRIPT_NAME	Name des Skripts
SCRIPT_OWNER	Besitzer des Skripts
SCRIPT_OBJECT_ID	ID des Skripts
SCRIPT_TYPE	Skript-Typ (PROCEDURE, ADAPTER oder UDF)
SCRIPT_LANGUAGE	Skriptsprache
SCRIPT_INPUT_TYPE	Eingabetyp des Skriptes (SCALAR oder SET im Falle von UDF Skripten, sonst NULL)
SCRIPT_RESULT_TYPE	Rückgabetypr des Skripts (ROWCOUNT oder TABLE bzw. RETURNS oder EMITS)
SCRIPT_TEXT	Kompletter Erzeugungstext des Skripts
SCRIPT_COMMENT	Kommentar zum Skript

EXA_USER_SESSIONS

Diese Systemtabelle enthält Informationen über die eigenen Benutzersessions. Unter anderem wird das letzte SQL-Kommando ausgegeben.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
USER_NAME	Angemeldeter Benutzer
STATUS	Aktueller Zustand der Session. Die wichtigsten davon sind: IDLE Client ist verbunden, führt aber nichts aus. EXECUTE SQL Ausführung einer einzelnen SQL-Anweisung. EXECUTE BATCH Ausführung einer Serie von SQL-Anweisungen. PREPARE SQL Anlegen eines prepared statement. EXECUTE PREPARED Ausführung eines prepared statements. FETCH DATA Client liest Ergebnisse. QUEUED Client wartet, weil zu viele parallele Anfragen ausgeführt werden.
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
STMT_ID	Fortlaufende ID des Nutzers innerhalb der Session
DURATION	Bisherige Dauer der Statement-Ausführung
QUERY_TIMEOUT	Session-spezifischer Wert für QUERY_TIMEOUT, siehe auch ALTER SESSION bzw. ALTER SYSTEM
ACTIVITY	Information über den aktuellen Zustand der SQL-Bearbeitung
TEMP_DB_RAM	Aktueller temporärer Datenbankspeicherverbrauch in MiB (clusterweit)
LOGIN_TIME	Zeitpunkt des Logins
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist

Spalte	Bedeutung
HOST	Rechnername oder IP-Adresse, von dem aus sich der Benutzer angemeldet hat.
OS_USER	Benutzername, unter dem sich der Benutzer am Betriebssystem des Rechners, von dem der Login kommt, angemeldet hat
OS_NAME	Betriebssystem des Client-Rechners
SCOPE_SCHEMA	Name des Schemas, in dem sich der Benutzer befindet
PRIORITY	Prioritätsgruppe
NICE	NICE-Attribut
RESOURCES	Zugeteilte Ressourcen in Prozent
SQL_TEXT	SQL-Text des Statements

EXA_USER_SYS_PRIVS

Listet alle Systemprivilegien auf, die direkt an den Benutzer vergeben wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
GRANTEE	Name des Benutzers
PRIVILEGE	Name des Systemprivilegs
ADMIN_OPTION	Gibt an, ob das Systemprivileg auch weitergegeben werden darf

EXA_USER_TABLES

Diese Systemtabelle beschreibt alle Tabellen in der Datenbank, die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OWNER	Besitzer der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
TABLE_IS_VIRTUAL	Zeigt an, ob es sich um eine virtuelle Tabelle handelt
TABLE_HAS DISTRIBUTION KEY	Zeigt an, ob die Tabelle explizit verteilt ist
TABLE_ROW_COUNT	Anzahl Zeilen in der Tabelle
DELETE_PERCENTAGE	Anteil der Zeilen, die nur als gelöscht markiert und noch nicht physisch gelöscht sind (in Prozent)
TABLE_COMMENT	Kommentar zur Tabelle

EXA_USER_USERS

Diese Tabelle bietet die gleichen Informationen wie EXA_ALL_USERS, ist allerdings auf den aktuell angemeldeten Benutzer beschränkt.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
USER_NAME	Name des Benutzers
CREATED	Zeitpunkt, zu dem der Benutzer eingerichtet wurde
USER_PRIORITY	Priorität des Nutzers
USER_COMMENT	Kommentar zum Nutzer

EXA_USER_VIEWS

Listet alle Views auf, die dem aktuellen Nutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
VIEW_SCHEMA	Name des Schemas, in dem die View angelegt wurde
VIEW_NAME	Name der View
SCOPE_SCHEMA	Schema, aus dem heraus die View angelegt wurde
VIEW_OWNER	Besitzer der View
VIEW_OBJECT_ID	Interne ID der View
VIEW_TEXT	Text der View, mit der sie erzeugt wurde
VIEW_COMMENT	Kommentar zur View

EXA_USER_VIRTUAL_COLUMNS

Diese Systemtabelle listet die Spalten aller virtuellen Tabellen auf die dem aktuellen Benutzer gehören. Sie zeigt Informationen die spezifisch sind für virtuelle Spalten. Virtuelle Spalten werden darüber hinaus auch in der Tabelle [EXA_USER_COLUMNS](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
COLUMN_SCHEMA	Zugehöriges Schema
COLUMN_TABLE	Zugehörige Tabelle
COLUMN_NAME	Name der Spalte
COLUMN_OBJECT_ID	ID der Spalte
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Spalte merken kann

EXA_USER_VIRTUAL_SCHEMA_PROPERTIES

Diese Systemtabelle enthält die Eigenschaften aller virtuellen Schemas auf die dem aktuellen Benutzer gehören.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCHEMA_NAME	Name des Schema-Objekts
SCHEMA_OBJECT_ID	ID des Schema-Objekts
PROPERTY_NAME	Name der Eigenschaft des virtuellen Schemas
PROPERTY_VALUE	Wert der Eigenschaft des virtuellen Schemas

EXA_USER_VIRTUAL_TABLES

Diese Systemtabelle listet alle virtuellen Tabellen auf die dem aktuellen Nutzer gehören und zeigt Informationen die spezifisch sind für virtuelle Tabellen. Virtuelle Tabellen werden darüber hinaus auch in der Tabelle [EXA_USER_TABLES](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLE_SCHEMA	Name des Schemas der Tabelle
TABLE_NAME	Name der Tabelle
TABLE_OBJECT_ID	ID der Tabelle
LAST_REFRESH	Zeitpunkt der letzten Aktualisierung der Metadaten
LAST_REFRESH_BY	Name des Nutzers der die letzte Aktualisierung der Metadaten initiiert hat
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu dieser virtuellen Tabelle merken kann

EXA_VIRTUAL_SCHEMAS

Diese Systemtabelle listet alle virtuellen Schemas und zeigt Informationen die spezifisch sind für virtuelle Schemas. Virtuelle Schemas werden darüber hinaus auch in der Tabelle [EXA_SCHEMAS](#) gelistet.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SCHEMA_NAME	Name des Schema-Objekts
SCHEMA_OWNER	Besitzer des Schema-Objekts
SCHEMA_OBJECT_ID	ID des Schema-Objekts
ADAPTER_SCRIPT	Name des Adapter Skripts welches für dieses virtuelle Schema verwendet wird
LAST_REFRESH	Zeitpunkt der letzten Aktualisierung der Metadaten
LAST_REFRESH_BY	Name des Nutzers der die letzte Aktualisierung der Metadaten initiiert hat
ADAPTER_NOTES	Feld in dem sich der Adapter beliebige Zusatzinformationen zu diesem Schema merken kann

EXA_VOLUME_USAGE

Zeigt Details über die Datenbank-Nutzung der Storage Volumes.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLESPACE	Der Tablespace des Volumes
VOLUME_ID	Bezeichner des Volumes
IPROC	Nummer des Knotens
LOCALITY	Lokalität des Master-Segments. Falls der Wert FALSE ist, muss mit Performance-Einbußen bei I/O Operationen gerechnet werden.
REDUNDANCY	Redundanz-Level des Volumes

Spalte	Bedeutung
HDD_TYPE	HDDs-Typ des Volumes
HDD_COUNT	Anzahl an HDDs des Volumes
HDD_FREE	Physikalisch freier Speicher auf dem Knoten in GiB von allen Disks des Volumes
VOLUME_SIZE	Größe des Volumes auf dem Knoten in GiB
USE	Nutzung des Volumes auf dem Knoten in Prozent, also 100% - (UNUSED_DATA/VOLUME_SIZE)
COMMIT_DATA	Mittels COMMIT gespeicherte Daten auf dem Knoten in GiB
SWAP_DATA	Ausgelagerte Daten auf dem Knoten in GiB
UNUSED_DATA	Unbenutzte Daten auf dem Knoten in GiB. Durch Fragmentierung kann es sein, dass Teile des Volumes nicht benutzt werden können.
DELETE_DATA	Gelöschte Daten auf dem Knoten in GiB. Dies beinhaltet Daten, die nicht sofort gelöscht werden können, z.B. durch eine laufende Backup- oder Shrink-Operation.
MULTICOPY_DATA	Multicopy-Daten auf dem Knoten in GiB. Diese treten im Fall von Transaktionen auf, bei denen parallel alte Daten gelesen und neue Daten der gleichen Tabelle geschrieben werden.

A.2.3. Statistische Systemtabellen

Die folgenden Systemtabellen enthalten historische Informationen zur Nutzung und zum Zustand des DBMS. Sie befinden sich in einem speziellen Systemschema EXA_STATISTICS, welches automatisch in den Namensraum integriert wird. Die Statistik-Systemtabellen sind wie die normalen Systemtabellen in EXA_SYSCAT zu finden. Zeitstempel zu historisierten Statistik-Einträgen werden in der aktuellen Datenbank-Zeitzone gespeichert (DBTIMEZONE).

Die Statistiken werden automatisch in regelmäßigen Abständen aktualisiert. Für eine explizite Aktualisierung steht der Befehl **FLUSH STATISTICS** (siehe [Abschnitt 2.2.6, Sonstige Befehle](#)) zur Verfügung.

Die Statistiktabellen sind für alle Nutzer ausschließlich lesend zugreifbar und unterliegen dem Transaktionskonzept (siehe [Abschnitt 3.1, Transaktions-Management](#)). Deshalb muss man ggf. eine neue Transaktion öffnen, um die aktuellsten Daten zu sehen!



Um Transaktionskonflikte sowohl für den Nutzer als auch für das DBMS zu minimieren, sollte in einer Transaktion entweder ausschließlich auf Statistik-Objekte oder auf normale Datenbankobjekten zugegriffen werden.

EXA_DBA_AUDIT_SESSIONS

Diese Systemtabelle enthält Informationen zu allen Sessions in die Datenbank, sofern das Auditing in EXAoperation eingeschaltet ist.

Die Inhalte dieser Tabelle können mittels dem Befehl **TRUNCATE AUDIT LOGS** geleert werden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Id der Session
LOGIN_TIME	Zeitpunkt des Login
LOGOUT_TIME	Zeitpunkt des Logout
USER_NAME	Nutzername
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist
HOST	Rechnername oder IP-Adresse, von dem aus sich der Benutzer angemeldet hat.
OS_USER	Benutzername, unter dem sich der Benutzer am Betriebssystem des Rechners, von dem der Login kommt, angemeldet hat
OS_NAME	Betriebssystem des Client-Rechners
SUCCESS	TRUE Login-Versuch war erfolgreich FALSE Login-Versuch schlug fehl (z.B. aufgrund eines falschen Passworts)
ERROR_CODE	Fehler-Code bei Fehlschlag des Logins
ERROR_TEXT	Fehler-Text bei Fehlschlag des Logins

EXA_DBA_AUDIT_SQL

Diese Systemtabelle listet alle ausgeführten SQL-Statements auf, sofern das Auditing in EXAoperation eingeschaltet ist.

Die Inhalte dieser Tabelle können mittels dem Befehl [TRUNCATE AUDIT LOGS](#) geleert werden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung	
SESSION_ID	Eindeutige Session-ID (siehe auch EXA_DBA_AUDIT_SESSIONS)	
STMT_ID	Fortlaufende ID des Statements innerhalb der Session	
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)	
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)	
DURATION	Ausführungszeitdauer des Statements in Sekunden	
START_TIME	Startzeitpunkt des Statements	
STOP_TIME	Endzeitpunkt des Statements	
CPU	CPU Nutzung in Prozent	
TEMP_DB_RAM_PEAK	Maximaler Verbrauch an temporären Datenbankdaten der Anfrage in MiB (clusterweit)	
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)	
	 Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.	
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)	
	 Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.	
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt über die Ausführungszeit)	
SUCCESS	TRUE Statement wurde erfolgreich ausgeführt FALSE Statement ist fehlgeschlagen (z.B. mit einer DataException)	
ERROR_CODE	Fehler-Code bei Fehlschlag des Statements	
ERROR_TEXT	Fehler-Text bei Fehlschlag des Statements	
SCOPE_SCHEMA	Schema, in dem sich der Nutzer befand	
PRIORITY	Prioritätsgruppe	
NICE	NICE-Attribut	
RESOURCES	Zugeteilte Ressourcen in Prozent	
ROW_COUNT	Ergebniszahlen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	
EXECUTION_MODE	EXECUTE Normale Ausführung eines Statements PREPARE Prepare Schritte eines Prepared Statements CACHED Abfrage, die auf den Query Cache zugreift	

Spalte	Bedeutung
	PREPROCESS Ausführung des Präprozessor Skripts
SQL_TEXT	Ausgeführter SQL-Text (begrenzt auf maximal 2.000.000 Zeichen)

EXA_DBA_PROFILE_LAST_DAY

Diese Systemtabelle listet Profiling-Informationen für alle Sessions mit aktiviertem Profiling auf. Details zu diesem Thema finden Sie unter [Abschnitt 3.9, Profiling](#).

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
STMT_ID	Fortlaufende ID des Statements innerhalb der Session
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)
PART_ID	Fortlaufende ID des Ausführungsschritts innerhalb des Statements
PART_NAME	Name des Ausführungsschritts (siehe auch Abschnitt 3.9, Profiling)
PART_INFO	Erweiterte Information zum Ausführungsschritt: GLOBAL Globale Aktion (z.B. globaler Join) EXPRESSION INDEX Nichtpersistenter Join-Index, der auf einem Ausdruck erzeugt wird NL JOIN Nested Loop Join (Kreuzprodukt) REPLICATED Repliziertes Objekt (z.B. kleine Tabellen) TEMPORARY Temporäres Objekt (für Zwischenergebnisse)
OBJECT_SCHEMA	Schema, in dem das zu verarbeitende Objekt liegt
OBJECT_NAME	Name des zu verarbeitenden Objekts
OBJECT_ROWS	Anzahl an Zeilen des zu verarbeitenden Objekts
OUT_ROWS	Anzahl an Ergebnis-Zeilen des Ausführungsschritts
DURATION	Benötigte Dauer des Ausführungsschritts in Sekunden
CPU	CPU Nutzung des Ausführungsschritts in Prozent (gemittelt über die Ausführungszeit)
TEMP_DB_RAM_PEAK	Verbrauch an temporären Datenbankdaten des Ausführungsschritts in MiB (clusterweit, Maximum über die Ausführungszeit)
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit) <p> Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.</p>
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit) <p> Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.</p>
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt über die Ausführungszeit)

Spalte	Bedeutung
REMARKS	Zusätzliche Informationen
SQL_TEXT	Entsprechender SQL-Text

EXA_DBA_PROFILE_RUNNING

Diese Systemtabelle listet Profiling-Informationen für alle laufenden Anfragen. Details zu diesem Thema finden Sie unter [Abschnitt 3.9, Profiling](#).

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
STMT_ID	Fortlaufende ID des Statements innerhalb der Session
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)
PART_ID	Fortlaufende ID des Ausführungsschritts innerhalb des Statements
PART_NAME	Name des Ausführungsschritts (siehe auch Abschnitt 3.9, Profiling)
PART_INFO	Erweiterte Information zum Ausführungsschritt: GLOBAL Globale Aktion (z.B. globaler Join) EXPRESSION INDEX Nichtpersistenter Join-Index, der auf einem Ausdruck erzeugt wird NL JOIN Nested Loop Join (Kreuzprodukt) REPLICATED Repliziertes Objekt (z.B. kleine Tabellen) TEMPORARY Temporäres Objekt (für Zwischenergebnisse)
PART_FINISHED	Gibt an, ob dieser Ausführungsschritt bereits fertig berechnet ist
OBJECT_SCHEMA	Schema, in dem das zu verarbeitende Objekt liegt
OBJECT_NAME	Name des zu verarbeitenden Objekts
OBJECT_ROWS	Anzahl an Zeilen des zu verarbeitenden Objekts
OUT_ROWS	Anzahl an Ergebnis-Zeilen des Ausführungsschritts
DURATION	Benötigte Dauer des Ausführungsschritts in Sekunden
CPU	CPU Nutzung des Ausführungsschritts in Prozent (gemittelt über die Ausführungszeit)
TEMP_DB_RAM_PEAK	Verbrauch an temporären Datenbankdaten des Ausführungsschritts in MiB (clusterweit, Maximum über die Ausführungszeit)
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)  Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)  Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.

Spalte	Bedeutung
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt über die Ausführungszeit)
REMARKS	Zusätzliche Informationen
SQL_TEXT	Entsprechender SQL-Text

EXA_DBA_SESSIONS_LAST_DAY

Diese Systemtabelle enthält Informationen über Benutzersessions des letzten Tages.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
LOGIN_TIME	Zeitpunkt des Logins
LOGOUT_TIME	Zeitpunkt des Logouts
USER_NAME	Angemeldeter Benutzer
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist
HOST	Rechnername oder IP-Adresse, von dem aus sich der Benutzer angemeldet hat.
OS_USER	Benutzername, unter dem sich der Benutzer am Betriebssystem des Rechners, von dem der Login kommt, angemeldet hat
OS_NAME	Betriebssystem des Client-Rechners
SUCCESS	Information, ob der Loginversuch erfolgreich war
ERROR_CODE	Fehler-Code beim Login
ERROR_TEXT	Fehler-Text beim Login

EXA_DBA_TRANSACTION_CONFLICTS

Diese Systemtabelle enthält Informationen über Transaktionskonflikte.

Die Inhalte dieser Tabelle können mittels dem Befehl [TRUNCATE AUDIT LOGS](#) geleert werden.

Zugriff haben nur Benutzer mit Systemprivileg "SELECT ANY DICTIONARY".

Spalte	Bedeutung
SESSION_ID	Session-ID
CONFLICT_SESSION_ID	Session, die den Konflikt verursacht
START_TIME	Anfangszeitpunkt des Konflikts
STOP_TIME	Endzeitpunkt des Konflikts bzw. NULL bei offenen Konflikten
CONFLICT_TYPE	Typ des Konflikts:
	WAIT FOR COMMIT Eine Session muss auf das Commit einer anderen warten
	TRANSACTION ROLL-BACK Eine Session musste aufgrund einer anderen zurückgerollt werden

Spalte	Bedeutung
CONFLICT_OBJECTS	Name der betroffenen Objekte
CONFLICT_INFO	Zusätzliche Informationen zum Konflikt

EXA_DB_SIZE_LAST_DAY

Diese Systemtabelle zeigt die Datenbankgrößen der letzten 24 Stunden. Die Angaben sind jeweils als Summe über alle Knoten des Systems zu verstehen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
MEASURE_TIME	Zeitpunkt der Messung
RAW_OBJECT_SIZE	Unkomprimierte Datenmenge in GiB
MEM_OBJECT_SIZE	Komprimierte Datenmenge in GiB
AUXILIARY_SIZE	Größe der Hilfsstrukturen wie z.B. Indices in GiB
STATISTICS_SIZE	Größe der statischen Systemtabellen in GiB
RECOMMENDED_DB_RAM_SIZE	Empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
STORAGE_SIZE	Größe des persistenten Volumes in GiB
USE	Tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
OBJECT_COUNT	Anzahl an Schemaobjekten in der Datenbank.

EXA_DB_SIZE_HOURLY

Diese Systemtabelle zeigt die stündlich aggregierten Datenbankgrößen aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
RAW_OBJECT_SIZE_AVG	Durchschnittliche unkomprimierte Datenmenge in GiB
RAW_OBJECT_SIZE_MAX	Maximale unkomprimierte Datenmenge in GiB
MEM_OBJECT_SIZE_AVG	Durchschnittliche komprimierte Datenmenge in GiB
MEM_OBJECT_SIZE_MAX	Maximale komprimierte Datenmenge in GiB
AUXILIARY_SIZE_AVG	Durchschnittliche Größe der Hilfsstrukturen wie z.B. Indices in GiB
AUXILIARY_SIZE_MAX	Maximale Größe der Hilfsstrukturen wie z.B. Indices in GiB
STATISTICS_SIZE_AVG	Durchschnittliche Größe der statischen Systemtabellen in GiB
STATISTICS_SIZE_MAX	Maximale Größe der statischen Systemtabellen in GiB
RECOMMENDED_DB_RAM_SIZE_AVG	Durchschnittliche empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
RECOMMENDED_DB_RAM_SIZE_MAX	Maximale empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
STORAGE_SIZE_AVG	Durchschnittliche Größe des persistenten Volumes in GiB
STORAGE_SIZE_MAX	Maximale Größe des persistenten Volumes in GiB

Spalte	Bedeutung
USE_AVG	Durchschnittlicher tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
USE_MAX	Maximaler tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
OBJECT_COUNT_AVG	Durchschnittliche Anzahl an Schemaobjekten in der Datenbank.
OBJECT_COUNT_MAX	Maximale Anzahl an Schemaobjekten in der Datenbank.

EXA_DB_SIZE_DAILY

Diese Systemtabelle zeigt die täglich aggregierten Datenbankgrößen aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
RAW_OBJECT_SIZE_AVG	Durchschnittliche unkomprimierte Datenmenge in GiB
RAW_OBJECT_SIZE_MAX	Maximale unkomprimierte Datenmenge in GiB
MEM_OBJECT_SIZE_AVG	Durchschnittliche komprimierte Datenmenge in GiB
MEM_OBJECT_SIZE_MAX	Maximale komprimierte Datenmenge in GiB
AUXILIARY_SIZE_AVG	Durchschnittliche Größe der Hilfsstrukturen wie z.B. Indices in GiB
AUXILIARY_SIZE_MAX	Maximale Größe der Hilfsstrukturen wie z.B. Indices in GiB
STATISTICS_SIZE_AVG	Durchschnittliche Größe der statischen Systemtabellen in GiB
STATISTICS_SIZE_MAX	Maximale Größe der statischen Systemtabellen in GiB
RECOMMENDED_DB_RAM_SIZE_AVG	Durchschnittliche empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
RECOMMENDED_DB_RAM_SIZE_MAX	Maximale empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
STORAGE_SIZE_AVG	Durchschnittliche Größe des persistenten Volumes in GiB
STORAGE_SIZE_MAX	Maximale Größe des persistenten Volumes in GiB
USE_AVG	Durchschnittlicher tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
USE_MAX	Maximaler tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
OBJECT_COUNT_AVG	Durchschnittliche Anzahl an Schemaobjekten in der Datenbank.
OBJECT_COUNT_MAX	Maximale Anzahl an Schemaobjekten in der Datenbank.

EXA_DB_SIZE_MONTHLY

Diese Systemtabelle zeigt die monatlich aggregierten Datenbankgrößen aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
RAW_OBJECT_SIZE_AVG	Durchschnittliche unkomprimierte Datenmenge in GiB
RAW_OBJECT_SIZE_MAX	Maximale unkomprimierte Datenmenge in GiB

Spalte	Bedeutung
MEM_OBJECT_SIZE_AVG	Durchschnittliche komprimierte Datenmenge in GiB
MEM_OBJECT_SIZE_MAX	Maximale komprimierte Datenmenge in GiB
AUXILIARY_SIZE_AVG	Durchschnittliche Größe der Hilfsstrukturen wie z.B. Indices in GiB
AUXILIARY_SIZE_MAX	Maximale Größe der Hilfsstrukturen wie z.B. Indices in GiB
STATISTICS_SIZE_AVG	Durchschnittliche Größe der statischen Systemtabellen in GiB
STATISTICS_SIZE_MAX	Maximale Größe der statischen Systemtabellen in GiB
RECOMMENDED_DB_RAM_SIZE_AVG	Durchschnittliche empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
RECOMMENDED_DB_RAM_SIZE_MAX	Maximale empfohlene DB-RAM-Größe in GiB, um die höchste Performance des Systems ausschöpfen zu können
STORAGE_SIZE_AVG	Durchschnittliche Größe des persistenten Volumes in GiB
STORAGE_SIZE_MAX	Maximale Größe des persistenten Volumes in GiB
USE_AVG	Durchschnittlicher tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
USE_MAX	Maximaler tatsächlich benutzter Festplattenplatz im Verhältnis zur Größe des persistenten Volumes in Prozent
OBJECT_COUNT_AVG	Durchschnittliche Anzahl an Schemaobjekten in der Datenbank.
OBJECT_COUNT_MAX	Maximale Anzahl an Schemaobjekten in der Datenbank.

EXA_MONITOR_LAST_DAY

Diese Systemtabelle zeigt diverse Monitoring-Informationen des Systems an (jeweils Maximalwerte über die Clusterknoten).



Die angegebenen Daten-Raten sind kein Hinweis auf die Hardware Performance, sondern wurden lediglich zur besseren Vergleichbarkeit bei Abweichungen der Messintervalle eingesetzt. Aus dem Produkt der Rate und der Länge des letzten Messintervalls lässt sich die eigentliche Datenmenge ablesen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
MEASURE_TIME	Zeitpunkt der Messung
LOAD	Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
CPU	CPU Nutzung in Prozent (der Datenbankinstanz, gemittelt auf das letzte Messintervall)
TEMP_DB_RAM	Temporäre Datenbankdaten in MiB (der Datenbankinstanz, gemittelt auf das letzte Messintervall)
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt auf das letzte Messintervall)
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt auf das letzte Messintervall)
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt auf das letzte Messintervall)
SWAP	Swap-Rate in MiB pro Sekunde (pro Knoten, gemittelt auf das letzte Messintervall). Falls dieser Wert über 0 ist, kann dies auf ein Problem in der System-Konfiguration hinweisen.

EXA_MONITOR_HOURLY

Diese Systemtabelle zeigt die stündlich aggregierten Monitoring-Informationen (der Werte aus [EXA_MONITOR_LAST_DAY](#)) aufsteigend sortiert nach dem Intervallbeginn.

-  Die angegebenen Daten-Raten sind kein Hinweis auf die Hardware Performance, sondern wurden lediglich zur besseren Vergleichbarkeit bei Abweichungen der Messintervalle eingesetzt. Aus dem Produkt der Rate und der Länge des letzten Messintervalls lässt sich die eigentliche Datenmenge ablesen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
LOAD_AVG	Durchschnittliche Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
LOAD_MAX	Maximale Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent (der Datenbankinstanz)
CPU_MAX	Maximale CPU Nutzung in Prozent (der Datenbankinstanz)
TEMP_DB_RAM_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
TEMP_DB_RAM_MAX	Maximaler Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde
SWAP_AVG	Durchschnittliche Swap-Rate in MiB pro Sekunde
SWAP_MAX	Maximale Swap-Rate in MiB pro Sekunde. Falls dieser Wert über 0 ist, kann dies auf ein Problem in der System-Konfiguration hinweisen.

EXA_MONITOR_DAILY

Diese Systemtabelle zeigt die täglich aggregierten Monitoring-Informationen (der Werte aus [EXA_MONITOR_LAST_DAY](#)) aufsteigend sortiert nach dem Intervallbeginn.

-  Die angegebenen Daten-Raten sind kein Hinweis auf die Hardware Performance, sondern wurden lediglich zur besseren Vergleichbarkeit bei Abweichungen der Messintervalle eingesetzt. Aus dem Produkt der Rate und der Länge des letzten Messintervalls lässt sich die eigentliche Datenmenge ablesen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
LOAD_AVG	Durchschnittliche Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)

Spalte	Bedeutung
LOAD_MAX	Maximale Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent (der Datenbankinstanz)
CPU_MAX	Maximale CPU Nutzung in Prozent (der Datenbankinstanz)
TEMP_DB_RAM_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
TEMP_DB_RAM_MAX	Maximaler Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde
SWAP_AVG	Durchschnittliche Swap-Rate in MiB pro Sekunde
SWAP_MAX	Maximale Swap-Rate in MiB pro Sekunde. Falls dieser Wert über 0 ist, kann dies auf ein Problem in der System-Konfiguration hinweisen.

EXA_MONITOR_MONTHLY

Diese Systemtabelle zeigt die monatlich aggregierten Monitoring-Informationen (der Werte aus [EXA_MONITOR_LAST_DAY](#)) aufsteigend sortiert nach dem Intervallbeginn.



Die angegebenen Daten-Raten sind kein Hinweis auf die Hardware Performance, sondern wurden lediglich zur besseren Vergleichbarkeit bei Abweichungen der Messintervalle eingesetzt. Aus dem Produkt der Rate und der Länge des letzten Messintervalls lässt sich die eigentliche Datenmenge ablesen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
LOAD_AVG	Durchschnittliche Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
LOAD_MAX	Maximale Systemlast (äquivalent zum Load-Wert der letzten Minute des Programms <code>uptime</code>)
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent (der Datenbankinstanz)
CPU_MAX	Maximale CPU Nutzung in Prozent (der Datenbankinstanz)
TEMP_DB_RAM_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
TEMP_DB_RAM_MAX	Maximaler Verbrauch an temporären Datenbankdaten in MiB (der Datenbankinstanz)
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde

Spalte	Bedeutung
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde
SWAP_AVG	Durchschnittliche Swap-Rate in MiB pro Sekunde. Falls dieser Wert über 0 ist, kann dies auf ein Problem in der System-Konfiguration hinweisen.
SWAP_MAX	Maximale Swap-Rate in MiB pro Sekunde. Falls dieser Wert über 0 ist, kann dies auf ein Problem in der System-Konfiguration hinweisen.

EXA_SQL_LAST_DAY

Diese Systemtabelle zeigt alle ausgeführten SQL-Statements ohne Bezug zum ausführenden Nutzer oder konkreten SQL-Texten. Es werden nur Statements erfasst, die erfolgreich kompiliert wurden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
STMT_ID	Fortlaufende ID des Nutzers innerhalb der Session
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)
DURATION	Ausführungsdauer des Statements in Sekunden
START_TIME	Startzeitpunkt des Statements
STOP_TIME	Endzeitpunkt des Statements
CPU	CPU Nutzung in Prozent
TEMP_DB_RAM_PEAK	Maximaler Verbrauch an temporären Datenbankdaten der Anfrage in MiB (clusterweit)
HDD_READ	Maximale Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt auf das letzte Messintervall)
	 Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.
HDD_WRITE	Maximale Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt auf das letzte Messintervall)
	 Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.
NET	Maximale Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt auf das letzte Messintervall)
SUCCESS	Ergebnis des Statements
	TRUE Statement wurde erfolgreich ausgeführt
	FALSE Statement ist fehlgeschlagen, z.B. mit einer DataException
ERROR_CODE	Fehler-Code bei Fehlschlag des Statements
ERROR_TEXT	Fehler-Text bei Fehlschlag des Statements
PRIORITY	Prioritätsgruppe
NICE	NICE-Attribut
RESOURCES	Zugeteilte Ressourcen in Prozent

Spalte	Bedeutung	
ROW_COUNT	Ergebniszeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	
EXECUTION_MODE	EXECUTE	Normale Ausführung eines Statements
	PREPARE	Prepare Schritte eines Prepared Statements
	CACHED	Abfrage, die auf den Query Cache zugreift
	PREPROCESS	Ausführung des Präprozessor Skripts

EXA_SQL_HOURLY

Diese Systemtabelle zeigt die stündlich aggregierte Anzahl an ausgeführten SQL-Statements aufsteigend sortiert nach dem Intervallbeginn. Je Intervall werden für die verschiedenen Statement-Typen (z.B. SELECT) mehrere Einträge erzeugt.

Zugriff haben alle Benutzer.

Spalte	Bedeutung	
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls	
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)	
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)	
SUCCESS	Ergebnis des Statements	
	TRUE	Statement wurde erfolgreich ausgeführt
	FALSE	Statement ist fehlgeschlagen, z.B. mit einer DataException
COUNT	Anzahl der Ausführungen	
DURATION_AVG	Durchschnittliche Ausführungsduer der Statements	
DURATION_MAX	Maximale Ausführungsduer der Statements	
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent	
CPU_MAX	Maximale CPU Nutzung in Prozent	
TEMP_DB_RAM_PEAK_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
TEMP_DB_RAM_PEAK_MAX	Maximaler Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde (pro Knoten)	
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde (pro Knoten)	
ROW_COUNT_AVG	Durchschnittliche Ergebniszeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	

Spalte	Bedeutung	
ROW_COUNT_MAX	Maximale Ergebniszeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	
EXECUTION_MODE	EXECUTE	Normale Ausführung eines Statements
	PREPARE	Prepare Schritte eines Prepared Statements
	CACHED	Abfrage, die auf den Query Cache zugreift
	PREPROCESS	Ausführung des Präprozessor Skripts

EXA_SQL_DAILY

Diese Systemtabelle zeigt die täglich aggregierte Anzahl an ausgeführten SQL-Statements aufsteigend sortiert nach dem Intervallbeginn. Je Intervall werden für die verschiedenen Statement-Typen (z.B. SELECT) mehrere Einträge erzeugt.

Zugriff haben alle Benutzer.

Spalte	Bedeutung	
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls	
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)	
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)	
SUCCESS	Ergebnis des Statements	
	TRUE	Statement wurde erfolgreich ausgeführt
	FALSE	Statement ist fehlgeschlagen, z.B. mit einer DataException
COUNT	Anzahl der Ausführungen	
DURATION_AVG	Durchschnittliche Ausführungsduer der Statements	
DURATION_MAX	Maximale Ausführungsduer der Statements	
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent	
CPU_MAX	Maximale CPU Nutzung in Prozent	
TEMP_DB_RAM_PEAK_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
TEMP_DB_RAM_PEAK_MAX	Maximaler Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde (pro Knoten)	
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde (pro Knoten)	
ROW_COUNT_AVG	Durchschnittliche Ergebniszeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	

Spalte	Bedeutung	
ROW_COUNT_MAX	Maximale Ergebnissezeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	
EXECUTION_MODE	EXECUTE	Normale Ausführung eines Statements
	PREPARE	Prepare Schritte eines Prepared Statements
	CACHED	Abfrage, die auf den Query Cache zugreift
	PREPROCESS	Ausführung des Präprozessor Skripts

EXA_SQL_MONTHLY

Diese Systemtabelle zeigt die monatlich aggregierte Anzahl an ausgeführten SQL-Statements aufsteigend sortiert nach dem Intervallbeginn. Je Intervall werden für die verschiedenen Statement-Typen (z.B. SELECT) mehrere Einträge erzeugt.

Zugriff haben alle Benutzer.

Spalte	Bedeutung	
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls	
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)	
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)	
SUCCESS	Ergebnis des Statements	
	TRUE	Statement wurde erfolgreich ausgeführt
	FALSE	Statement ist fehlgeschlagen, z.B. mit einer DataException
COUNT	Anzahl der Ausführungen	
DURATION_AVG	Durchschnittliche Ausführungsduer der Statements	
DURATION_MAX	Maximale Ausführungsduer der Statements	
CPU_AVG	Durchschnittliche CPU Nutzung in Prozent	
CPU_MAX	Maximale CPU Nutzung in Prozent	
TEMP_DB_RAM_PEAK_AVG	Durchschnittlicher Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
TEMP_DB_RAM_PEAK_MAX	Maximaler Verbrauch an temporären Datenbankdaten der Anfragen in MiB (clusterweit)	
HDD_READ_AVG	Durchschnittliche Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_READ_MAX	Maximale Festplatten-Leserate in MiB pro Sekunde (pro Knoten)	
HDD_WRITE_AVG	Durchschnittliche Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
HDD_WRITE_MAX	Maximale Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, nur COMMIT)	
NET_AVG	Durchschnittliche Netzwerkrate in MiB pro Sekunde (pro Knoten)	
NET_MAX	Maximale Netzwerkrate in MiB pro Sekunde (pro Knoten)	
ROW_COUNT_AVG	Durchschnittliche Ergebnissezeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	

Spalte	Bedeutung	
ROW_COUNT_MAX	Maximale Ergebniszeilen bei Abfragen bzw. geänderte Zeilen bei DML und DDL Statements	
EXECUTION_MODE	EXECUTE	Normale Ausführung eines Statements
	PREPARE	Prepare Schritte eines Prepared Statements
	CACHED	Abfrage, die auf den Query Cache zugreift
	PREPROCESS	Ausführung des Präprozessor Skripts

EXA_SYSTEM_EVENTS

Diese Systemtabelle zeigt Systemereignisse wie Startup oder Shutdown.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
MEASURE_TIME	Zeitpunkt des Ereignisses
EVENT_TYPE	Typ des Ereignisses
	STARTUP DBMS wurde gestartet und Logins sind ab diesem Zeitpunkt möglich
	SHUTDOWN DBMS wurde gestoppt
	BACKUP_START Start eines Backup-Erzeugungsprozesses
	BACKUP_END Ende eines Backup-Erzeugungsprozesses
	RESTORE_START Start des Einspielens eines Backups
	RESTORE_END Ende des Einspielens eines Backups
	RESTART Neustart des DBMS aufgrund eines Fehlers
	FAILSAFETY Knotenausfall mit ggf. automatischem DBMS-Neustart mit Reserveknoten
	RECOVERY_START Start des Datenwiederherstellungsprozesses
	RECOVERY_END Ende des Datenwiederherstellungsprozesses (zu diesem Zeitpunkt ist die Redundanz im Cluster vollständig wiederhergestellt)
	LICENSE_EXCEEDED Lizenz überschritten (Datenbanken weisen Einfügeoperationen zurück)
	LICENSE_OK Lizenz wieder O.K. (Datenbanken erlauben Einfügeoperationen)
SIZE_LIMIT_EXCEEDED	Locales Größenlimit der DB überschritten (Einfügeoperationen werden zurückgewiesen)
	SIZE_LIMIT_OK Lokales Größenlimit der DB wieder O.K. (Einfügeoperationen erlaubt)
DBMS_VERSION	Version des DBMS
NODES	Anzahl an Knoten im Cluster
DB_RAM_SIZE	Verwendete Lizenzgröße in GiB
PARAMETERS	Parameter für das DBMS

EXA_USAGE_LAST_DAY

Diese Systemtabelle zeigt die Nutzungsdaten des DBMS der letzten 24 Stunden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
MEASURE_TIME	Zeitpunkt der Messung
USERS	Anzahl eingeloggter Nutzer
QUERIES	Anzahl rechnender Verbindungen zum DBMS

EXA_USAGE_HOURLY

Diese Systemtabelle zeigt die stündlich aggregierten Nutzungsdaten des DBMS aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
USERS_AVG	Durchschnittliche Anzahl eingeloggter Nutzer
USERS_MAX	Maximale Anzahl eingeloggter Nutzer
QUERIES_AVG	Durchschnittliche Anzahl rechnender Verbindungen
QUERIES_MAX	Maximale Anzahl rechnender Verbindungen
IDLE	Prozentsatz, wie viel Zeit des Intervalls keine einzige Anfrage lief

EXA_USAGE_DAILY

Diese Systemtabelle zeigt die täglich aggregierten Nutzungsdaten des DBMS aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
USERS_AVG	Durchschnittliche Anzahl eingeloggter Nutzer
USERS_MAX	Maximale Anzahl eingeloggter Nutzer
QUERIES_AVG	Durchschnittliche Anzahl rechnender Verbindungen
QUERIES_MAX	Maximale Anzahl rechnender Verbindungen
IDLE	Prozentsatz, wie viel Zeit des Intervalls keine einzige Anfrage lief

EXA_USAGE_MONTHLY

Diese Systemtabelle zeigt die monatlich aggregierten Nutzungsdaten des DBMS aufsteigend sortiert nach dem Intervallbeginn.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
INTERVAL_START	Referenzzeitpunkt am Beginn des Intervalls
USERS_AVG	Durchschnittliche Anzahl eingeloggter Nutzer
USERS_MAX	Maximale Anzahl eingeloggter Nutzer
QUERIES_AVG	Durchschnittliche Anzahl rechnender Verbindungen

Spalte	Bedeutung
QUERIES_MAX	Maximale Anzahl rechnender Verbindungen
IDLE	Prozentsatz, wie viel Zeit des Intervalls keine einzige Anfrage lief

EXA_USER_PROFILE_LAST_DAY

Diese Systemtabelle listet Profiling-Informationen für alle *eigenen* Sessions mit aktiviertem Profiling auf. Details zu diesem Thema finden Sie unter [Abschnitt 3.9, Profiling](#).

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
STMT_ID	Fortlaufende ID des Statements innerhalb der Session
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)
PART_ID	Fortlaufende ID des Ausführungsschritts innerhalb des Statements
PART_NAME	Name des Ausführungsschritts (siehe auch Abschnitt 3.9, Profiling)
PART_INFO	Erweiterte Information zum Ausführungsschritt: GLOBAL Globale Aktion (z.B. globaler Join) EXPRESSION INDEX Nichtpersistenter Join-Index, der auf einem Ausdruck erzeugt wird NL JOIN Nested Loop Join (Kreuzprodukt) REPLICATED Repliziertes Objekt (z.B. kleine Tabellen) TEMPORARY Temporäres Objekt (für Zwischenergebnisse)
OBJECT_SCHEMA	Schema, in dem das zu verarbeitende Objekt liegt
OBJECT_NAME	Name des zu verarbeitenden Objekts
OBJECT_ROWS	Anzahl an Zeilen des zu verarbeitenden Objekts
OUT_ROWS	Anzahl an Ergebnis-Zeilen des Ausführungsschritts
DURATION	Benötigte Dauer des Ausführungsschritts in Sekunden
CPU	CPU Nutzung des Ausführungsschritts in Prozent (gemittelt über die Ausführungszeit)
TEMP_DB_RAM_PEAK	Verbrauch an temporären Datenbankdaten des Ausführungsschritts in MiB (clusterweit, Maximum über die Ausführungszeit)
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit) <p style="text-align: right;"> Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.</p>
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit) <p style="text-align: right;"> Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.</p>
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt über die Ausführungszeit)

Spalte	Bedeutung
REMARKS	Zusätzliche Informationen
SQL_TEXT	Entsprechender SQL-Text

EXA_USER_PROFILE_RUNNING

Diese Systemtabelle listet Profiling-Informationen für alle *eigenen* laufenden Anfragen. Details zu diesem Thema finden Sie unter [Abschnitt 3.9, Profiling](#).

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
STMT_ID	Fortlaufende ID des Statements innerhalb der Session
COMMAND_NAME	Name des Statements (z.B. SELECT, COMMIT, MERGE usw.)
COMMAND_CLASS	Klasse des Statements (z.B. DQL, TRANSACTION, DML usw.)
PART_ID	Fortlaufende ID des Ausführungsschritts innerhalb des Statements
PART_NAME	Name des Ausführungsschritts (siehe auch Abschnitt 3.9, Profiling)
PART_INFO	Erweiterte Information zum Ausführungsschritt: GLOBAL Globale Aktion (z.B. globaler Join) EXPRESSION INDEX Nichtpersistenter Join-Index, der auf einem Ausdruck erzeugt wird NL JOIN Nested Loop Join (Kreuzprodukt) REPLICATED Repliziertes Objekt (z.B. kleine Tabellen) TEMPORARY Temporäres Objekt (für Zwischenergebnisse)
PART_FINISHED	Gibt an, ob dieser Ausführungsschritt bereits fertig berechnet ist
OBJECT_SCHEMA	Schema, in dem das zu verarbeitende Objekt liegt
OBJECT_NAME	Name des zu verarbeitenden Objekts
OBJECT_ROWS	Anzahl an Zeilen des zu verarbeitenden Objekts
OUT_ROWS	Anzahl an Ergebnis-Zeilen des Ausführungsschritts
DURATION	Benötigte Dauer des Ausführungsschritts in Sekunden
CPU	CPU Nutzung des Ausführungsschritts in Prozent (gemittelt über die Ausführungszeit)
TEMP_DB_RAM_PEAK	Verbrauch an temporären Datenbankdaten des Ausführungsschritts in MiB (clusterweit, Maximum über die Ausführungszeit)
HDD_READ	Festplatten-Leserate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)  Falls dieser Wert größer als 0 ist, so mussten Daten in den Hauptspeicher geladen werden.
HDD_WRITE	Festplatten-Schreibrate in MiB pro Sekunde (pro Knoten, gemittelt über die Ausführungszeit)  Dieser Wert gibt nur die im COMMIT geschriebenen Daten. Für andere Befehle ist der Wert stets NULL.

Spalte	Bedeutung
NET	Netzwerkrate in MiB pro Sekunde (Summe aus Senden/Empfangen, pro Knoten, gemittelt über die Ausführungszeit)
REMARKS	Zusätzliche Informationen
SQL_TEXT	Entsprechender SQL-Text

EXA_USER_SESSIONS_LAST_DAY

Diese Systemtabelle enthält Informationen über die *eigenen* Benutzersessions des letzten Tages.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Eindeutige Session-ID
LOGIN_TIME	Zeitpunkt des Logins
LOGOUT_TIME	Zeitpunkt des Logouts
USER_NAME	Angemeldeter Benutzer
CLIENT	Client-Anwendung, die der Benutzer verwendet
DRIVER	Verwendeter Treiber
ENCRYPTED	Gibt an, ob die Verbindung verschlüsselt ist
HOST	Rechnername oder IP-Adresse, von dem aus sich der Benutzer angemeldet hat.
OS_USER	Benutzername, unter dem sich der Benutzer am Betriebssystem des Rechners, von dem der Login kommt, angemeldet hat
OS_NAME	Betriebssystem des Client-Rechners
SUCCESS	Information, ob der Loginversuch erfolgreich war
ERROR_CODE	Fehler-Code beim Login
ERROR_TEXT	Fehler-Text beim Login

EXA_USER_TRANSACTION_CONFLICTS_LAST_DAY

Diese Systemtabelle enthält Informationen über Transaktionskonflikte zu Benutzersessions.

Die Inhalte dieser Tabelle können mittels dem Befehl [TRUNCATE AUDIT LOGS](#) geleert werden.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
SESSION_ID	Session-ID
CONFLICT_SESSION_ID	Session, die den Konflikt verursacht
START_TIME	Anfangszeitpunkt des Konflikts
STOP_TIME	Endzeitpunkt des Konflikts bzw. NULL bei offenen Konflikten
CONFLICT_TYPE	Typ des Konflikts:
	WAIT FOR COMMIT Eine Session muss auf das Commit einer anderen warten
	TRANSACTION ROLL-BACK Eine Session musste aufgrund einer anderen zurückgerollt werden

Spalte	Bedeutung
CONFLICT_OBJECTS	Name der betroffenen Objekte
CONFLICT_INFO	Zusätzliche Informationen zum Konflikt

A.2.4. Oracle-kompatible Systemtabellen

Die folgenden Systemtabellen wurden eingeführt, um aus Oracle gewohnte Funktionalität bereitzustellen.

CAT

Diese Systemtabelle listet alle Tabellen und Views im aktuellen Schema auf. CAT ist somit nicht vollständig kompatibel zu Oracle, da dort alle Objekte sichtbar sind, die dem Nutzer gehören. Da in Exasol mehrere Schemas einem Nutzer gehören können, wurde die Entscheidung für diese Definition getroffen.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
TABLE_NAME	Name der Tabelle bzw. des Views
TABLE_TYPE	Art des Objekts: TABLE oder VIEW

DUAL

Diese Systemtabelle kann nach dem Vorbild von Oracles gleichnamiger Systemtabelle zur Ausgabe von statischer Information verwendet werden (z.B. "SELECT CURRENT_USER FROM DUAL"). Sie enthält eine Zeile mit einer einzigen Spalte.

Zugriff haben alle Benutzer.

Spalte	Bedeutung
DUMMY	Enthält einen Nullwert

Anhang B. Details zur Rechteverwaltung

In diesem Kapitel werden alle Details erläutert, die für die Rechteverwaltung wichtig sind. Hierzu gehört u.a. eine Auflistung aller möglichen System- bzw. Objektprivilegien sowie die Liste der hierfür zur Verfügung stehenden Systemtabellen.

Einen Einstieg in die grundlegenden Konzepte zum Thema Rechteverwaltung finden Sie in [Abschnitt 3.2, Rechteverwaltung](#). Nähere Details zu den einzelnen SQL-Befehlen befinden sich in [Abschnitt 2.2.3, Zugriffssteuerung mittels SQL \(DCL\)](#).

B.1. Liste der System- bzw. Objektprivilegien

Die folgenden zwei Tabellen enthalten alle in Exasol definierten System- und Objektprivilegien.

Tabelle B.1. Systemprivilegien in Exasol

Systemprivileg	Erlaubte Aktionen
MISC	
GRANT ANY OBJECT PRIVILEGE	Beliebige Objekt-Rechte vergeben oder entziehen
GRANT ANY PRIVILEGE	Beliebige System-Rechte vergeben oder entziehen (Dies ist ein mächtiges Privileg und sollte nur an wenige Nutzer vergeben werden)
GRANT ANY PRIORITY	Prioritäten setzen oder entziehen
CREATE SESSION	Zur Datenbank verbinden
KILL ANY SESSION	Session oder Query abbrechen
ALTER SYSTEM	Systemweite Einstellungen ändern (z.B. NLS_DATE_FORMAT)
USERS	
CREATE USER	Benutzer anlegen
ALTER USER	Das Passwort eines beliebigen Nutzers ändern (Dies ist ein mächtiges Privileg und sollte nur an wenige Nutzer vergeben werden)
DROP USER	Benutzer löschen
ROLES	
CREATE ROLE	Rollen erzeugen
DROP ANY ROLE	Rollen löschen
GRANT ANY ROLE	Beliebige Rollen zuweisen (Dies ist ein mächtiges Privileg und sollte nur an wenige Nutzer vergeben werden)
CONNECTIONS	
CREATE CONNECTION	Externe Verbindung anlegen
ALTER ANY CONNECTION	Verbindungsdaten einer Verbindung ändern
DROP ANY CONNECTION	Verbindung löschen
GRANT ANY CONNECTION	Verbindungen anderen Nutzern oder Rollen gewähren
USE ANY CONNECTION	Beliebige Verbindung in den Befehlen IMPORT und EXPORT benutzen
ACCESS ANY CONNECTION	Details von beliebigen Verbindungen in Skripten abfragen
SCHEMAS	
CREATE SCHEMA	Anlegen von Schemas
ALTER ANY SCHEMA	Ein Schema einem anderen User oder einer Rolle zuordnen
DROP ANY SCHEMA	Löschen von beliebigen Schemas
CREATE VIRTUAL SCHEMA	Anlegen von virtuellen Schemas
ALTER ANY VIRTUAL SCHEMA	Parameter von virtuellen Schemas verändern
ALTER ANY VIRTUAL SCHEMA REFRESH	Aktualisieren der Metadaten von virtuellen Schemas
DROP ANY VIRTUAL SCHEMA	Löschen von virtuellen Schemas
TABLES	
CREATE TABLE	Tabellen im eigenen Schema oder dem einer zugewiesenen Rolle erzeugen (DROP ist implizit, da der Besitzer einer Tabelle diese immer löschen kann)
CREATE ANY TABLE	Tabellen in einem beliebigen Schema erzeugen
ALTER ANY TABLE	Tabellen aus einem beliebigen Schema ändern
DELETE ANY TABLE	Zeilen einer Tabelle aus einem beliebigen Schema löschen

Systemprivileg	Erlaubte Aktionen
DROP ANY TABLE	Eine Tabelle aus einem beliebigen Schema löschen
INSERT ANY TABLE	In eine Tabelle aus einem beliebigen Schema Daten einfügen
SELECT ANY TABLE	Auf die Inhalte einer Tabelle oder View aus einem beliebigen Schema zugreifen (betrifft keine Systemtabellen)
SELECT ANY DICTIONARY	Auf die Inhalte einer beliebigen Systemtabelle zugreifen
UPDATE ANY TABLE	Zeilen einer Tabelle aus einem beliebigen Schema verändern
VIEWS	
CREATE VIEW	Views im eigenen Schema oder dem einer zugewiesenen Rolle anlegen (DROP ist implizit, da der Besitzer einer View diese immer löschen kann)
CREATE ANY VIEW	Views in einem beliebigen Schema erzeugen
DROP ANY VIEW	Views aus einem beliebigen Schema löschen
FUNCTIONS	
CREATE FUNCTION	Funktionen im eigenen Schema oder dem einer zugewiesenen Rolle anlegen (DROP ist implizit, da der Besitzer einer Funktion diese immer löschen kann)
CREATE ANY FUNCTION	Funktionen in beliebigen Schema anlegen
DROP ANY FUNCTION	Funktionen aus beliebigen Schema löschen
EXECUTE ANY FUNCTION	Funktionen aus beliebigen Schema ausführen
SCRIPTS	
CREATE SCRIPT	Skripte im eigenen Schema oder dem einer zugewiesenen Rolle anlegen (DROP ist implizit, da der Besitzer eines Skripts dieses immer löschen kann)
CREATE ANY SCRIPT	Skripte in beliebigen Schema anlegen
DROP ANY SCRIPT	Skripte aus beliebigen Schema löschen

Systemprivileg	Erlaubte Aktionen
EXECUTE ANY SCRIPT	Skripte aus beliebigen Schema ausführen

Tabelle B.2. Objektprivilegien in Exasol

Objektprivileg	Schemaobjekte	Erlaubte Aktionen
ALTER	Schema, Table, Virtual Schema	Ausführen des ALTER TABLE – Statements
SELECT	Schema, Table, View, Virtual Schema, Virtual Table	Zugriff auf die Tabelleninhalte
INSERT	Schema, Table	Zeilen in eine Tabelle hinzufügen
UPDATE	Schema, Table	Zeileninhalte einer Tabelle ändern
DELETE	Schema, Table	Löschen von Zeilen
REFERENCES	Table	Erzeugen von Foreign Keys, die diese Tabelle referenzieren
EXECUTE	Schema, Function, Script	Ausführen von Funktionen und Skripten
ACCESS	Connection	Zugriff auf Connection-Details aus einem Skript heraus
REFRESH	Virtual Schema	Aktualisieren der Metadaten eines virtuellen Schemas

B.2. Benötigte Privilegien für SQL-Befehle

In der folgenden Tabelle sind sämtliche von Exasol unterstützten SQL-Befehle sowie die dazu benötigten Privilegien aufgelistet.

Tabelle B.3. Benötigte Privilegien zur Ausführung von SQL-Befehlen

SQL-Befehl	Benötigte Privilegien
USERS	
CREATE USER u ...	System-Privileg CREATE USER
ALTER USER u ...	Man kann das eigene Passwort immer ändern. Für andere Benutzer wird das System-Privileg ALTER USER benötigt.
DROP USER u	System-Privileg DROP USER
ROLES	
CREATE ROLE r ...	System-Privileg CREATE ROLE
DROP ROLE r	System-Privileg DROP ROLE oder man muss diese Rolle mit der ADMIN OPTION erhalten haben.
CONNECTIONS	
CREATE CONNECTION c ...	System-Privileg CREATE CONNECTION
ALTER CONNECTION c ...	System-Privileg ALTER ANY CONNECTION oder man muss die Verbindung mit der ADMIN OPTION erhalten haben.
DROP CONNECTION c	System-Privileg DROP ANY CONNECTION oder man muss die Verbindung mit der ADMIN OPTION erhalten haben.
IMPORT ... FROM c ...	System-Privileg USE ANY CONNECTION oder man muss die Verbindung erhalten haben.
SCHEMAS	
CREATE SCHEMA s	System-Privileg CREATE SCHEMA
OPEN SCHEMA s	Keine Beschränkung. Anmerkung: die Schemaobjekte innerhalb des Schemas sind aber nicht automatisch lesbar!
DROP SCHEMA s	System-Privileg DROP ANY SCHEMA oder das Schema gehört dem aktuellen Benutzer. Wird CASCADE angegeben, so werden alle im Schema enthaltenen Schemaobjekte mitgelöscht!
ALTER SCHEMA s CHANGE OWNER u	System-Privileg ALTER ANY SCHEMA
CREATE VIRTUAL SCHEMA s	System-Privileg CREATE VIRTUAL SCHEMA
DROP VIRTUAL SCHEMA s	System-Privileg DROP ANY VIRTUAL SCHEMA oder das Schema gehört dem aktuellen Benutzer.
ALTER VIRTUAL SCHEMA s SET ...	System-Privileg ALTER ANY VIRTUAL SCHEMA, Objekt-Privileg ALTER auf s oder das Schema gehört dem aktuellen Benutzer. Zusätzlich werden Zugriffsrechte auf das entsprechenden Adapter-Skript benötigt.
ALTER VIRTUAL SCHEMA s REFRESH	System-Privilegien ALTER ANY VIRTUAL SCHEMA oder ALTER ANY VIRTUAL SCHEMA REFRESH, Objekt-Privilegien ALTER oder REFRESH auf s, oder das Schema gehört dem aktuellen Benutzer. Zusätzlich werden Zugriffsrechte auf das entsprechenden Adapter-Skript benötigt.
ALTER VIRTUAL SCHEMA s CHANGE OWNER u	System-Privileg ALTER ANY VIRTUAL SCHEMA oder das Schema gehört dem aktuellen Benutzer. Anmerkung: das Objektprivileg ALTER ist hierfür nicht ausreichend.
OPEN SCHEMA s	Keine Beschränkung.
TABLES	
CREATE TABLE t (<col_defs>)	System-Privileg CREATE TABLE, falls die Tabelle im eigenen Schema oder dem einer zugewiesenen Rolle liegt. Andernfalls System-Privileg CREATE ANY TABLE.

SQL-Befehl	Benötigte Privilegien
CREATE TABLE AS <subquery>	Wie bei CREATE TABLE t (<col_defs>), aber der Nutzer muss auch SELECT-Privilegien auf die Tabellen des Subquery besitzen.
CREATE OR REPLACE TABLE t ...	Wie bei CREATE TABLE t (<col_defs>), aber falls die Tabelle ersetzt wird, so muss der Nutzer ebenso die nötigen Privilegien besitzen wie für DROP TABLE t.
ALTER TABLE t ...	System-Privileg ALTER ANY TABLE, Objekt-Privileg ALTER auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
SELECT * FROM t	System-Privileg SELECT ANY TABLE oder Objekt-Privileg SELECT auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen. Bei Zugriff auf Objekte aus virtuellen Schemas werden noch die entsprechenden Zugriffsrechte auf das Adapter-Skript und ggfs. benutzte Connections benötigt.
INSERT INTO t ...	System-Privileg INSERT ANY TABLE oder Objekt-Privileg INSERT auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
UPDATE t SET ...	System-Privileg UPDATE ANY TABLE oder Objekt-Privileg UPDATE auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
MERGE INTO t USING u ...	Entsprechende INSERT- und UPDATE-Privilegien auf t bzw. dessen Schema sowie SELECT-Privilegien auf u.
DELETE FROM t	System-Privileg DELETE ANY TABLE oder Objekt-Privileg DELETE auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
TRUNCATE TABLE t	System-Privileg DELETE ANY TABLE oder Objekt-Privileg DELETE auf t bzw. dessen Schema oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
DROP TABLE t	System-Privileg DROP ANY TABLE oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
RECOMPRESS TABLE t	Ein modifizierendes ANY TABLE-Systemprivileg bzw. Objektprivileg oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
REORGANIZE TABLE t	Ein modifizierendes ANY TABLE-Systemprivileg bzw. Objektprivileg oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
PRELOAD TABLE t ;	Lese- oder Schreibzugriff mittels beliebigem System- bzw. Objektprivileg oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
<i>Foreign Key auf t erzeugen</i>	Objektprivileg REFERENCES auf t oder t gehört dem aktuellen Benutzer oder einer seiner Rollen.
VIEWS	

SQL-Befehl	Benötigte Privilegien
CREATE VIEW v AS ...	System-Privileg CREATE VIEW, falls die View im eigenen Schema oder dem einer zugewiesenen Rolle liegt. Andernfalls System-Privileg CREATE ANY VIEW. Außerdem muss der Besitzer der View (entspricht nicht automatisch dem CREATOR) entsprechende SELECT-Privilegien auf alle referenzierten Basistabellen besitzen. Bei Zugriff auf Objekte aus virtuellen Schemas benötigt der das SELECT ausführende Nutzer zudem noch die entsprechenden Zugriffsrechte auf das Adapter-Skript und ggfs. benutzte Connections.
CREATE OR REPLACE VIEW v ...	Wie bei CREATE VIEW, aber falls die View ersetzt wird, so muss der Nutzer ebenso die nötigen Privilegien besitzen wie für DROP VIEW v.
SELECT * FROM v	System-Privileg SELECT ANY TABLE oder Objekt-Privileg SELECT auf v bzw. deren Schema. Außerdem muss der Besitzer von v entsprechende SELECT-Privilegien auf die referenzierten Basistabellen der View besitzen.
DROP VIEW v	System-Privileg DROP ANY VIEW oder v gehört dem aktuellen Benutzer oder einer seiner Rollen.
FUNCTIONS	
CREATE FUNCTION f ...	System-Privileg CREATE FUNCTION, falls die Funktion im eigenen Schema oder dem einer zugewiesenen Rolle liegt. Andernfalls das System-Privileg CREATE ANY FUNCTION.
CREATE OR REPLACE FUNCTION f ...	Wie bei CREATE FUNCTION, aber falls die Funktion ersetzt wird, so muss der Nutzer ebenso die nötigen Privilegien besitzen wie für DROP FUNCTION f.
SELECT f(...) FROM t	System-Privileg EXECUTE ANY FUNCTION oder Objekt-Privileg EXECUTE auf die Funktion bzw. deren Schema.
DROP FUNCTION f	System-Privileg DROP ANY FUNCTION, falls die Funktion f nicht im eigenen Schema oder dem einer zugewiesenen Rolle liegt.
SCRIPTS	
CREATE SCRIPT s ...	System-Privileg CREATE SCRIPT, falls das Skript im eigenen Schema oder dem einer zugewiesenen Rolle liegt. Andernfalls das System-Privileg CREATE ANY SCRIPT.
CREATE OR REPLACE SCRIPT s ...	Wie bei CREATE SCRIPT, aber falls das Skript ersetzt wird, so muss der Nutzer ebenso die nötigen Privilegien besitzen wie für DROP SCRIPT s.
EXECUTE SCRIPT s	System-Privileg EXECUTE ANY SCRIPT oder Objekt-Privileg EXECUTE auf das Skript bzw. dessen Schema.
DROP SCRIPT s	System-Privileg DROP ANY SCRIPT, falls das Script s nicht im eigenen Schema oder dem einer zugewiesenen Rolle liegt.
RENAME	
RENAME o TO x	Falls o ein Schema ist, so muss das Schema dem Nutzer oder einer seiner Rollen gehören. Falls o ein Schemaobjekt ist, so muss das Objekt dem Benutzer oder einer seiner Rollen gehören (also in einem eigenen Schema oder dem einer zugewiesenen Rolle liegen). Falls o ein Nutzer, eine Rolle bzw. eine Connection ist, so benötigt der Nutzer entsprechende Systemprivilegien.
GRANT	

SQL-Befehl	Benötigte Privilegien
GRANT <sys_priv> TO u	System-Privileg GRANT ANY PRIVILEGE oder der Nutzer muss dieses Systemprivileg mit der WITH ADMIN OPTION erhalten haben.
GRANT <ob_priv> ON o TO u	System-Privileg GRANT ANY OBJECT PRIVILEGE oder der Nutzer oder eine seiner Rollen muss Besitzer des Schemaobjekts o sein.
GRANT r TO u	System-Privileg GRANT ANY ROLE oder der Nutzer muss diese Rolle mit der WITH ADMIN OPTION erhalten haben.
GRANT PRIORITY c TO u	System-Privileg GRANT ANY PRIORITY.
GRANT CONNECTION c TO u	System-Privileg GRANT ANY CONNECTION oder der Nutzer muss diese Verbindung mit der WITH ADMIN OPTION erhalten haben.
REVOKE	
REVOKE <sys_priv> FROM u	System-Privileg GRANT ANY PRIVILEGE oder der Nutzer muss dieses Systemprivileg mit der WITH ADMIN OPTION erhalten haben.
REVOKE <ob_priv> ON o FROM u	System-Privileg GRANT ANY OBJECT PRIVILEGE oder der Nutzer muss dieses Objekt-Privileg selbst gewährt haben.
REVOKE r FROM u	System-Privileg GRANT ANY ROLE oder der Nutzer muss diese Rolle mit der WITH ADMIN OPTION erhalten haben.
REVOKE PRIORITY FROM u	System-Privileg GRANT ANY PRIORITY.
REVOKE CONNECTION c FROM u	System-Privileg GRANT ANY CONNECTION oder der Nutzer muss diese Verbindung mit der WITH ADMIN OPTION erhalten haben.
MISC	
ALTER SESSION ...	Für diesen Befehl werden keinerlei Privilegien benötigt.
KILL SESSION ...	System-Privileg KILL ANY SESSION, sofern es nicht die eigene Session betrifft.
ALTER SYSTEM ...	System-Privileg ALTER SYSTEM.
DESCRIBE o	Schemaobjekte dürfen mit dem Befehl DESCRIBE beschrieben werden, falls der Nutzer oder eine seiner Rollen der Besitzer ist oder Zugriff auf das Objekt hat (Objekt- oder System-Privilegien).

B.3. Systemtabellen zur Rechteverwaltung

Die nachfolgenden Systemtabellen stehen als Information über den Zustand der Rechteverwaltung in Exasol zur Verfügung. Eine detaillierte Beschreibung aller Systemtabellen ist in [Anhang A, Systemtabellen](#) zu finden.

Benutzer

- EXA_DBAL_USERS** Alle Benutzer der Datenbank
- EXA_ALL_USERS** Alle Benutzer der Datenbank, eingeschränkte Information
- EXA_USER_USERS** Aktueller Benutzer

Rollen

- EXA_DBAL_ROLES** bzw. Alle Rollen der Datenbank
- EXA_ALL_ROLES**

EXA_DBROLE_PRIVS	Gewährte Rollen
EXA_USER_ROLE_PRIVS	Rollen, die dem aktuellen Benutzer direkt gewährt wurden
EXA_ROLE_ROLE_PRIVS	Rollen, die der aktuelle Benutzer indirekt über weitere Rollen inne hat
EXA_SESSION_ROLES	Rollen, die der aktuelle Benutzer inne hat

Verbindungen

EXA_DBA_CONNECTIONS	Alle Verbindungen der Datenbank
EXA_ALL_CONNECTIONS	Alle Verbindungen der Datenbank, eingeschränkte Information
EXA_DBA_CONNECTION_PRIVS	Alle gewährten Verbindungen
EXA_USER_CONNECTION_PRIVS	Verbindungen, die dem aktuellen Benutzer direkt gewährt wurden
EXA_ROLE_CONNECTION_PRIVS	Verbindungen, auf die der aktuelle Benutzer indirekt über seine Rollen Zugriff hat
EXA_SESSION_CONNECTIONS	Verbindungen, auf die der aktuelle Benutzer Zugriff hat

Systemprivilegien

EXA_DBA_SYS_PRIVS	Gewährte Systemprivilegien
EXA_USER_SYS_PRIVS	Systemprivilegien, die direkt an den aktuellen Benutzer vergeben wurden
EXA_ROLE_SYS_PRIVS	Systemprivilegien, die an Rollen des aktuellen Benutzers vergeben wurden
EXA_SESSION_PRIVS	Systemprivilegien, die dem Benutzer aktuell zur Verfügung stehen

Objektprivilegien

EXA_DBA_OBJ_PRIVS und EXA_DBA_RESTRICTED_OBJ_PRIVS	Objektprivilegien, die auf Objekte der Datenbank vergeben worden sind
EXA_ALL_OBJ_PRIVS	Wie EXA_DBA_OBJ_PRIVS , aber nur für zugreifbare Objekte der Datenbank
EXA_USER_OBJ_PRIVS und EXA_USER_RESTRICTED_OBJ_PRIVS	Objektprivilegien auf die Objekte, auf die der aktuelle Nutzer außer durch die Rolle PUBLIC Zugriff hat
EXA_ROLE_OBJ_PRIVS und EXA_ROLE_RESTRICTED_OBJ_PRIVS	Objektprivilegien, die an Rollen des Benutzers vergeben wurden
EXA_ALL_OBJ_PRIVS_MADE	Objektprivilegien, die der aktuelle Nutzer selbst vergeben hat, oder die dessen Objekte betreffen
EXA_USER_OBJ_PRIVS_MADE	Objektprivilegien, die Objekte des aktuellen Benutzers betreffen
EXA_ALL_OBJ_PRIVS_REC'D	Objektprivilegien, die dem aktuellen Nutzer direkt oder PUBLIC gewährt wurden
EXA_USER_OBJ_PRIVS_REC'D	Objektprivilegien, die dem aktuellen Benutzer direkt gewährt wurden

Anhang C. Konformität zum SQL-Standard

In diesem Abschnitt wird aufgelistet, welche Teile des aktuellen [SQL](#)-Standards (ISO/IEC 9075:2008) von Exasol unterstützt werden. Der SQL-Standard unterscheidet hierbei zwischen obligatorischen (mandatory) und optionalen (optional) Features.

C.1. SQL 2008 Standard Mandatory Features

In nachfolgender Tabelle ist die vollständige Liste der obligatorischen Features aufgelistet.

Obwohl viele Hersteller mit der Standardkonformität werben, ist uns kein System bekannt, das tatsächlich alle obligatorischen Features des SQL-Standards unterstützt.

Symbolbedeutung:

- ✓ Vollständig unterstützt
- ✓ Partiell unterstützt (d.h. nicht alle Subfeatures werden unterstützt)

Tabelle C.1. SQL 2008 Mandatory Features

Feature	Exasol
E011 Numeric data types	✓
E011-01 INTEGER and SMALLINT data types	✓
E011-02 REAL, DOUBLE PRECISION and FLOAT data types	✓
E011-03 DECIMAL and NUMERIC data types	✓
E011-04 Arithmetic operators	✓
E011-05 Numeric comparison	✓
E011-06 Implicit casting among numeric data types	✓
E021 Character string types	✓
E021-01 CHARACTER data type	✓
E021-02 CHARACTER VARYING data type	✓
E021-03 Character literals	✓
E021-04 CHARACTER_LENGTH function	✓
E021-05 OCTET_LENGTH function	✓
E021-06 SUBSTRING function	✓
E021-07 Character concatenation	✓
E021-08 UPPER and LOWER function	✓
E021-09 TRIM function	✓
E021-10 Implicit casting among the fixed-length and variable-length character string types	✓
E021-11 POSITION function	✓
E021-12 Character comparison	✓
E031 Identifiers	✓
E031-01 Delimited identifiers	✓
E031-02 Lower case identifiers	✓
E031-03 Trailing underscore	✓
E051 Basic query specification	✓
E051-01 SELECT DISTINCT	✓
E051-02 GROUP BY clause	✓
E051-04 GROUP BY can contain columns not in <select list>	✓
E051-05 Select list items can be renamed	✓
E051-06 HAVING clause	✓
E051-07 Qualified * in select list	✓
E051-08 Correlation names in the FROM clause	✓
E051-09 Rename columns in the FROM clause	✓
E061 Basic predicates and search conditions	✓
E061-01 Comparison predicate	✓
E061-02 BETWEEN predicate	✓
E061-03 IN predicate with list of values	✓
E061-04 LIKE predicate	✓
E061-05 LIKE predicate: ESCAPE clause	✓
E061-06 NULL predicate	✓

Feature	Exasol
E061-07 Quantified comparison predicate	
E061-08 EXISTS predicate	✓
E061-09 Subqueries in comparison predicate	✓
E061-11 Subqueries in IN predicate	✓
E061-12 Subqueries in quantified comparison predicate	
E061-13 Correlated Subqueries	✓
E061-14 Search condition	✓
E071 Basic query expressions	✓
E071-01 UNION DISTINCT table operator	✓
E071-02 UNION ALL table operator	✓
E071-03 EXCEPT DISTINCT table operator	✓
E071-05 Columns combined via table operators need not have exactly the same data type.	✓
E071-06 Table operators in subqueries	✓
E081 Basic privileges	✓
E081-01 SELECT privilege at the table level	✓
E081-02 DELETE privilege	✓
E081-03 INSERT privilege at the table level	✓
E081-04 UPDATE privilege at the table level	✓
E081-05 UPDATE privilege at the column level	
E081-06 REFERENCES privilege at the table level	✓
E081-07 REFERENCES privilege at the column level	
E081-08 WITH GRANT OPTION	
E081-09 USAGE privilege	
E081-10 EXECUTE privilege	✓
E091 Set functions	✓
E091-01 AVG	✓
E091-02 COUNT	✓
E091-03 MAX	✓
E091-04 MIN	✓
E091-05 SUM	✓
E091-06 ALL quantifier	✓
E091-07 DISTINCT quantifier	✓
E101 Basic data manipulation	✓
E101-01 INSERT statement	✓
E101-03 Searched UPDATE statement	✓
E101-04 Searched DELETE statement	✓
E111 Single row SELECT statement	
E121 Basic cursor support	
E121-01 DECLARE CURSOR	
E121-02 ORDER BY columns need not be in select list	
E121-03 Value expressions in ORDER BY clause	
E121-04 OPEN statement	

Feature	Exasol
E121-06 Positioned UPDATE statement	
E121-07 Positioned DELETE statement	
E121-08 CLOSE statement	
E121-10 FETCH statement: implicit NEXT	
E121-17 WITH HOLD cursors	
E131 Null value support (nulls in lieu of values)	✓
E141 Basic integrity constraints	✓
E141-01 NOT NULL constraint	✓
E141-02 UNIQUE constraints of NOT NULL columns	
E141-03 PRIMARY KEY constraint	✓
E141-04 Basic FOREIGN KEY constraint with the NO ACTION default for both referential delete action and referential update action.	✓
E141-06 CHECK constraint	
E141-07 Column defaults	✓
E141-08 NOT NULL inferred on PRIMARY KEY	✓
E141-10 Names in a foreign key can be specified in any order	
E151 Transaction support	✓
E151-01 COMMIT statement	✓
E151-02 ROLLBACK statement	✓
E152 Basic SET TRANSACTION statement	
E152-01 SET TRANSACTION statement: ISOLATION LEVEL SERIALIZABLE clause	
E152-02 SET TRANSACTION statement: READ ONLY and READ WRITE clauses	
E153 Updatable queries with subqueries	
E161 SQL comments using leading double minus	✓
E171 SQLSTATE support	
E182 Module language	
F031 Basic schema manipulation	✓
F031-01 CREATE TABLE statement to create persistent base tables	✓
F031-02 CREATE VIEW statement	✓
F031-03 GRANT statement	✓
F031-04 ALTER TABLE statement: ADD COLUMN clause	✓
F031-13 DROP TABLE statement: RESTRICT clause	✓
F031-16 DROP VIEW statement: RESTRICT clause	✓
F031-19 REVOKE statement: RESTRICT clause	✓
F041 Basic joined table	✓
F041-01 Inner join (but not necessarily the INNER keyword)	✓
F041-02 INNER keyword	✓
F041-03 LEFT OUTER JOIN	✓
F041-04 RIGHT OUTER JOIN	✓
F041-05 Outer joins can be nested	✓
F041-07 The inner table in a left or right outer join can also be used in an inner join	✓
F041-08 All comparison operators are supported (rather than just =)	✓

Feature	Exasol
F051 Basic date and time	✓
F051-01 DATE data type (including support of DATE literal)	✓
F051-02 TIME data type (including the support of TIME literal) with fractional seconds precision of at least 0	
F051-03 TIMESTAMP data type (including the support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6	✓
F051-04 Comparison predicate on DATE, TIME and TIMESTAMP data types	✓
F051-05 Explicit CAST between datetime types and character string types	✓
F051-06 CURRENT_DATE	✓
F051-07 LOCALTIME	
F051-08 LOCALTIMESTAMP	✓
F081 UNION and EXCEPT in views	✓
F131 Grouped operations	✓
F131-01 WHERE, GROUP BY and HAVING clauses supported in queries with grouped views	✓
F131-02 Multiple tables supported in queries with grouped views	✓
F131-03 Set functions supported in queries with grouped views	✓
F131-04 Subqueries with GROUP BY and HAVING clauses and grouped views	✓
F131-05 Single row SELECT with GROUP BY and HAVING clauses and grouped views	
F181 Multiple module support	
F201 CAST function	✓
F221 Explicit defaults	✓
F261 CASE expression	✓
F261-01 Simple CASE	✓
F261-02 Searched CASE	✓
F261-03 NULLIF	✓
F261-04 COALESCE	✓
F311 Schema definition statement	✓
F311-01 CREATE SCHEMA	✓
F311-02 CREATE TABLE for persistent base tables (within CREATE SCHEMA)	
F311-03 CREATE VIEW (within CREATE SCHEMA)	
F311-04 CREATE VIEW: WITH CHECK OPTION (within CREATE SCHEMA)	
F311-05 GRANT STATEMENT (within CREATE SCHEMA)	
F471 Scalar subquery values	✓
F481 Expanded NULL predicate	✓
F812 Basic flagging	
S011 Distinct data types	
T321 Basic SQL-invoked routines	✓
T321-01 User-defined functions with no overloading	✓
T321-02 User-defined stored procedures with no overloading	
T321-03 Function invocation	✓
T321-04 CALL statement	
T321-05 RETURN statement	✓

Feature	Exasol
T631 IN predicate with one list element	✓

C.2. SQL 2008 Standard Optional Features

Tabelle C.2. Von Exasol unterstützte SQL 2008 Optional Features

Feature-ID	Feature
F033	ALTER TABLE statement: DROP COLUMN clause
F052	Intervals and datetime arithmetic
F171	Multiple schemas per user
F222	INSERT Statement: DEFAULT VALUES clause
F302-01	INTERSECT DISTINCT table operator
F312	MERGE statement
F321	User authorization
F381	Extended schema manipulation
F381-01	ALTER TABLE statement: ALTER COLUMN clause
F381-02	ALTER TABLE statement: ADD CONSTRAINT clause
F381-03	ALTER TABLE statement: DROP CONSTRAINT clause
F391	Long identifiers
F401-02	Extended joined table: FULL OUTER JOIN
F401-04	Extended joined table: CROSS JOIN
F641	Row and table constructors
T031	BOOLEAN data type
T121	WITH (excluding RECURSIVE) in query expression
T171	LIKE clause in table definition
T172	AS subquery clause in table definition
T173	Extended LIKE clause in table definition
T174	Identity columns
T331	Basic roles
T351	Bracketed SQL comments /* ... */ comments)
T431	Extended grouping capabilities
T432	Nested and concatenated GROUPING SETS
T433	Multiargument GROUPING function
T434	GROUP BY DISTINCT
T441	ABS and MOD functions
T461	Symmetric BETWEEN predicate
T551	Optional key words for default syntax
T621	Enhanced numeric functions

Anhang D. Unterstützte Zeichensätze

In diesem Abschnitt sind alle Zeichensätze aufgelistet, die beim Ladeprozess mittels **IMPORT** und **EXPORT** und in EXAplus (siehe auch [SET ENCODING](#)) unterstützt werden.

Zeichensatz	Aliase
ASCII	US-ASCII, US, ISO-IR-6, ANSI_X3.4-1968, ANSI_X3.4-1986, ISO_646.IRV:1991, ISO646-US, IBM367, IBM-367, CP367, CP-367, 367
ISO-8859-1	ISO8859-1, ISO88591, LATIN-1, LATIN1, L1, ISO-IR-100, ISO_8859-1:1987, ISO_8859-1, IBM819, IBM-819, CP819, CP-819, 819
ISO-8859-2	ISO8859-2, ISO88592, LATIN-2, LATIN2, L2, ISO-IR-101, ISO_8859-2:1987, ISO_8859-2
ISO-8859-3	ISO8859-3, ISO88593, LATIN-3, LATIN3, L3, ISO-IR-109, ISO_8859-3:1988, ISO_8859-3
ISO-8859-4	ISO8859-4, ISO88594, LATIN-4, LATIN4, L4, ISO-IR-110, ISO_8859-4:1988, ISO_8859-4
ISO-8859-5	ISO8859-5, ISO88595, CYRILLIC, ISO-IR-144, ISO_8859-5:1988, ISO_8859-5
ISO-8859-6	ISO8859-6, ISO88596, ARABIC, ISO-IR-127, ISO_8859-6:1987, ISO_8859-6, ECMA-114, ASMO-708
ISO-8859-7	ISO8859-7, ISO88597, GREEK, GREEK8, ISO-IR-126, ISO_8859-7:1987, ISO_8859-7, ELOT_928, ECMA-118
ISO-8859-8	ISO8859-8, ISO88598, HEBREW, ISO-IR-138, ISO_8859-8:1988, ISO_8859-8
ISO-8859-9	ISO8859-9, ISO88599, LATIN-5, LATIN5, L5, ISO-IR-148, ISO_8859-9:1989, ISO_8859-9
ISO-8859-11	ISO8859-11, ISO885911
ISO-8859-13	ISO8859-13, ISO885913, LATIN-7, LATIN7, L7, ISO-IR-179
ISO-8859-15	ISO8859-15, ISO885915, LATIN-9, LATIN9, L9
IBM850	IBM-850, CP850, CP-850, 850
IBM852	IBM-852, CP852, CP-852, 852
IBM855	IBM-855, CP855, CP-855, 855
IBM856	IBM-856, CP856, CP-856, 856
IBM857	IBM-857, CP857, CP-857, 857
IBM860	IBM-860, CP860, CP-860, 860
IBM861	IBM-861, CP861, CP-861, 861, CP-IS
IBM862	IBM-862, CP862, CP-862, 862
IBM863	IBM-863, CP863, CP-863, 863
IBM864	IBM-864, CP864, CP-864, 864
IBM865	IBM-865, CP865, CP-865, 865
IBM866	IBM-866, CP866, CP-866, 866
IBM868	IBM-868, CP868, CP-868, 868, CP-AR
IBM869	IBM-869, CP869, CP-869, 869, CP-GR
WINDOWS-1250	CP1250, CP-1250, 1250, MS-EE
WINDOWS-1251	CP1251, CP-1251, 1251, MS-CYRL
WINDOWS-1252	CP1252, CP-1252, 1252, MS-ANSI
WINDOWS-1253	CP1253, CP-1253, 1253, MS-GREEK

Zeichensatz	Aliase
WINDOWS-1254	CP1254, CP-1254, 1254, MS-TURK
WINDOWS-1255	CP1255, CP-1255, 1255, MS-HEBR
WINDOWS-1256	CP1256, CP-1256, 1256, MS-ARAB
WINDOWS-1257	CP1257, CP-1257, 1257, WINBALTRIM
WINDOWS-1258	CP1258, CP-1258, 1258
WINDOWS-874	CP874, CP-874, 874, IBM874, IBM-874
WINDOWS-31J	WINDOWS-932, CP932, CP-932, 932
WINDOWS-936	CP936, CP-936, 936, GBK, MS936, MS-936
CP949	WINDOWS-949, CP-949, 949
BIG5	WINDOWS-950, CP950, CP-950, 950, BIG, BIG5, BIG-5, BIG-FIVE, BIGFIVE, CN-BIG5, BIG5-CP950
SHIFT-JIS	SJIS
UTF8	UTF-8, ISO10646=UTF8

Anhang E. Customer Service

Der Customer Service der Exasol AG unterstützt seine Kunden bei allen Fragen in Bezug auf Installation, Inbetriebnahme, Betrieb und Nutzung von Exasol. Alle Anliegen unserer Kunden werden vom Customer Service gerne entgegengenommen.

Anfragen

Wir empfehlen grundsätzlich, alle Anfragen an die E-Mail Adresse <service@exasol.com> zu richten. Es genügt eine kurze Schilderung des Anliegens per E-Mail bzw. in dringenden Fällen auch per Telefon. Wir bitten Sie, Ihre Kontaktdaten stets mit anzugeben.

Interne Prozesse

Vom Customer Service wird jede Anfrage aufgenommen und wenn möglich sofort beantwortet. Ist keine sofortige Antwort möglich, wird die Anfrage kategorisiert und an die entsprechende Abteilung bei Exasol weitergeleitet. In diesem Fall erhält unser Kunde eine Rückmeldung, bis wann er mit einer abschließenden Antwort oder einem Zwischenergebnis rechnen kann.

Downloads

Dieses Handbuch, aktuelle Software wie z. B. Treiber und weitere Informationen finden Sie in unserem Kundenportal unter wwwexasol.com.

Kontaktdaten

Exasol AG
Customer Service
Telefon: 00800 EXASUPPORT (00800 3927 877 678)
Email: <service@exasol.com>

Abkürzungsverzeichnis

A

ADO.NET	Abstract Data Objects .NET
ANSI	American National Standards Institute
API	Application Programming Interface

B

BI	Business Intelligence
BOM	Byte Order Mark

C

CLI	Call Level Interface
CSV	Comma Separated Values

D

DB	Datenbank
DBA	Datenbankadministrator
DBMS	Datenbankmanagementsystem
DCL	Data Control Language
DDL	Data Definition Language
DiagRec	Diagnostic Record, specified in the ODBC standard
DML	Data Manipulation Language
DNS	Domain Name Service
DQL	Data Query Language
DSN	Data Source Name

E

ETL	Extract, Transform, Load
-----	--------------------------

F

FBV	Fix Block Values
-----	------------------

G

GB	$\text{Gigabyte} = 10^9 = 1.000.000.000 \text{ Bytes}$
GiB	$\text{Gibibyte} = 2^{30} = 1.073.741.824 \text{ Bytes}$

H

HPC	High Performance Computing
I	

ISO	International Standards Organization
J	

JDBC	Java DataBase Connectivity
JRE	Java Runtime Environment
JSON	JavaScript Object Notation - ein Standard-Format, dass menschenlesbaren Text benutzt um Datenobjekte mittels Attribut-Wert Paaren zu übermitteln
JVM	Java Virtual Machine

K

kB	$\text{Kilobyte} = 10^3 = 1.000 \text{ Bytes}$
kiB	$\text{Kibibyte} = 2^{10} = 1.024 \text{ Bytes}$

L

LDAP	Lightweight Directory Access Protocol (Dienst zur Authentifizierung)
M	

MB	$\text{Megabyte} = 10^6 = 1.000.000 \text{ Bytes}$
MiB	$\text{Mebibyte} = 2^{20} = 1.048.576 \text{ Bytes}$

O

ODBC	Open DataBase Connectivity
OLE DB	Object Linking and Embedding DataBase

P

PCRE	Perl Compatible Regular Expressions
POSIX	Portable Operating System Interface [for Unix]

POSIX BRE	POSIX Basic Regular Expressions
POSIX ERE	POSIX Extended Regular Expressions

S

SASL	Simple Authentication and Security Layer
SDK	Software Development Kit
SQL	Standard Query Language

T

TMS	Transaction Management System
-----	-------------------------------

U

UDF	User Defined Function (Benutzerdefinierte Funktion)
-----	---

W

WKB	Well Known Binary (Binäre Darstellung von Geodaten)
WKT	Well Known Text (Textuelle Darstellung von Geodaten)

Index

A

ABS Funktion, 151
 ACOS Funktion, 151
 ADD_DAYS Funktion, 152
 ADD_HOURS Funktion, 152
 ADD_MINUTES Funktion, 153
 ADD_MONTHS Funktion, 154
 ADD_SECONDS Funktion, 154
 ADD_WEEKS Funktion, 155
 ADD_YEARS Funktion, 155
 ADO.NET Data Destination, 417
 ADO.NET Data Processing Extension, 419
 ADO.NET Data Provider, 412
 Benutzung, 413
 Installation, 412
 Alias, 80
 ALTER ANY CONNECTION Systemprivileg, 35, 68
 ALTER ANY SCHEMA Systemprivileg, 14
 ALTER ANY TABLE Systemprivileg, 20, 23, 24, 36
 ALTER ANY VIRTUAL SCHEMA REFRESH Systemprivileg, 14
 ALTER ANY VIRTUAL SCHEMA Systemprivileg, 14
 ALTER CONNECTION Statement, 68
 ALTER Objektprivileg, 14, 20, 23, 24
 ALTER SCHEMA Statement, 14
 ALTER SESSION Statement, 95
 ALTER SYSTEM Statement, 98
 ALTER TABLE Statement
 ADD COLUMN, 19
 ADD CONSTRAINT, 24
 ALTER COLUMN DEFAULT, 19
 ALTER COLUMN IDENTITY, 19
 DISTRIBUTE BY, 23
 DROP COLUMN, 19
 DROP CONSTRAINT, 24
 DROP DISTRIBUTION KEYS, 23
 MODIFY COLUMN, 19
 MODIFY CONSTRAINT, 24
 RENAME COLUMN, 19
 RENAME CONSTRAINT, 24
 ALTER USER Statement, 64
 ALTER USER Systemprivileg, 36, 64
 AND Prädikat, 139
 APPROXIMATE_COUNT_DISTINCT Funktion, 156
 ASC, 82
 ASCII Funktion, 156
 ASIN Funktion, 157
 ATAN Funktion, 157
 ATAN2 Funktion, 158
 Auditing, 459, 460
 Authentifizierung
 LDAP, 64
 Passwort, 63

AVG Funktion, 158

B

Benutzer
 Benutzer anlegen, 63
 Benutzer löschen, 65
 Passwort ändern, 64
 Rechte entziehen, 72
 Rechte zuweisen, 69
 Benutzerdefinierte Funktionen
 FOR Schleife, 29
 Funktion erzeugen, 27
 Funktion löschen, 30
 IF Verzweigung, 28
 Syntax, 28
 Variablenzuweisung, 28
 WHILE Schleife, 29
 BETWEEN Prädikat, 140
 Bezeichner, 5
 begrenzt, 6
 regulär, 6
 reservierte Wörter, 7
 schemaqualifiziert, 7
 BIT_AND Funktion, 159
 BIT_CHECK Funktion, 159
 BIT_LENGTH Funktion, 160
 BIT_LROTATE Funktion, 160
 BIT_LSHIFT Funktion, 161
 BIT_NOT Funktion, 162
 BIT_OR Funktion, 162
 BIT_RROTATE Funktion, 163
 BIT_RSHIFT Funktion, 163
 BIT_SET Funktion, 164
 BIT_TO_NUM Funktion, 164
 BIT_XOR Funktion, 165
 BOOLEAN Datentyp, 109
 BucketFS, 67, 339

C

CASCADE
 in DROP SCHEMA Statement, 13
 in DROP VIEW Statement, 27
 CASCADE CONSTRAINTS
 in ALTER TABLE Statement, 21
 in DROP TABLE Statement, 19
 in REVOKE Statement, 74
 CASE Funktion, 165
 CAST Funktion, 167
 CEIL Funktion, 167
 CEILING Funktion, 167
 CHAR Datentyp, 111
 CHARACTER_LENGTH Funktion, 168
 CH[A]R Funktion, 168
 CLI
 Allgemeines, 421
 Beispiel, 423
 Best-Practice, 424

Linux/Unix-Version, 422
Windows-Version, 421
CLOSE SCHEMA Statement, 101
Clustervergrößerung, 104
COALESCE Funktion, 169
COLOGNE_PHONETIC Funktion, 169
COMMENT Statement, 36
COMMIT Statement, 92
CONCAT Funktion, 170
CONNECT BY, 80
 CONNECT_BY_ISCYCLE, 81
 CONNECT_BY_ISLEAF, 81
 CONNECT_BY_ROOT, 81
 LEVEL, 80
 NO_CYCLE, 80
 START WITH, 80
 SYS_CONNECT_BY_PATH, 80
CONNECT_BY_ISCYCLE, 81
CONNECT_BY_ISCYCLE Funktion, 170
CONNECT_BY_ISLEAF, 81
CONNECT_BY_ISLEAF Funktion, 171
CONNECT_BY_ROOT, 81
Constraints, 17
 FOREIGN KEY, 25
 NOT NULL, 25
 PRIMARY KEY, 25
 Status
 DISABLE, 25, 97, 99
 ENABLE, 25, 97, 99
CONSTRAINT_STATE_DEFAULT, 25, 97, 99
CONVERT Funktion, 171
CONVERT_TZ Funktion, 172
CORR Funktion, 173
COS Funktion, 174
COSH Funktion, 174
COT Funktion, 175
COUNT Funktion, 175
COVAR_POP Funktion, 176
COVAR_SAMP Funktion, 177
CREATE ANY FUNCTION Systemprivileg, 28
CREATE ANY SCRIPT Systemprivileg, 31
CREATE ANY TABLE Systemprivileg, 15, 18
CREATE ANY VIEW Systemprivileg, 26
CREATE CONNECTION Statement, 67
CREATE CONNECTION Systemprivileg, 36, 67
CREATE FUNCTION Statement, 27
CREATE FUNCTION Systemprivileg, 28
CREATE ROLE Statement, 65
CREATE ROLE Systemprivileg, 35, 36
CREATE SCHEMA Statement, 12
CREATE SCHEMA Systemprivileg, 12
CREATE SCRIPT Statement, 31
CREATE SCRIPT Systemprivileg, 31
CREATE TABLE Statement, 15
CREATE TABLE Systemprivileg, 15, 18
CREATE USER Statement, 63
CREATE USER Systemprivileg, 35, 36, 63
CREATE VIEW Statement, 26

CREATE VIEW Systemprivileg, 26
CREATE VIRTUAL SCHEMA Systemprivileg, 12
CROSS JOIN, 79
CSV Datenformat, 279
CUBE, 81
CURDATE Funktion, 177
CURRENT_DATE Funktion, 178
CURRENT_SCHEMA, 12
CURRENT_SCHEMA Funktion, 178
CURRENT_SESSION Funktion, 179
CURRENT_STATEMENT Funktion, 179
CURRENT_TIMESTAMP Funktion, 180
CURRENT_USER Funktion, 180

D

Data Destination, 412, 417
Data Processing Extension, 412, 419
Data types
 GEOMETRY, 111
DATE Datentyp, 109
Datenbank
 Datenbank reorganisieren, 104
Datentypen, 108
 Aliase, 112, 113
 Datum/Zeit
 DATE, 109
 INTERVAL DAY TO SECOND, 111
 INTERVAL YEAR TO MONTH, 111
 TIMESTAMP, 109
 TIMESTAMP WITH LOCAL TIME ZONE, 109
Details, 108
Numerische
 DECIMAL, 108
 DOUBLE PRECISION, 108
Typkonvertierungsregeln, 113
Zeichenketten
 CHAR, 111
 VARCHAR, 112
Überblick, 108
DATE_TRUNC Funktion, 181
DAY Funktion, 181
DAYS_BETWEEN Funktion, 182
DBTIMEZONE Funktion, 182
DCL Statements, 63
 ALTER CONNECTION, 68
 ALTER USER, 64
 CREATE CONNECTION, 67
 CREATE ROLE, 65
 CREATE USER, 63
 DROP CONNECTION, 69
 DROP ROLE, 66
 DROP USER, 65
 GRANT, 69
 REVOKE, 72
DDEX Provider, 412
DDL Statements, 12
 ALTER SCHEMA, 14

- ALTER TABLE
 ADD COLUMN, 19
 ADD CONSTRAINT, 24
 ALTER COLUMN DEFAULT, 19
 ALTER COLUMN IDENTITY, 19
 DISTRIBUTE BY, 23
 DROP COLUMN, 19
 DROP CONSTRAINT, 24
 DROP DISTRIBUTION KEYS, 23
 MODIFY COLUMN, 19
 MODIFY CONSTRAINT, 24
 RENAME COLUMN, 19
 RENAME CONSTRAINT, 24
COMMENT, 36
CREATE FUNCTION, 27
CREATE SCHEMA, 12
CREATE SCRIPT, 31
CREATE TABLE, 15
CREATE VIEW, 26
DROP FUNCTION, 30
DROP SCHEMA, 13
DROP SCRIPT, 35
DROP TABLE, 19
DROP VIEW, 27
RENAME, 35
SELECT INTO, 18
DECIMAL Datentyp, 108
DECODE Funktion, 183
Default-Werte für Spalten, 17, 19, 21, 22, 38, 40, 42, 115
 Anzeigen, 116
 Beispiel, 115, 116
 Defaultwert löschen, 22
 Erlaubte Werte, 116
 Mögliche Fehlerquellen, 117
DEFAULT_LIKE_ESCAPE_CHARACTER, 96, 99
DEGREES Funktion, 184
DELETE ANY TABLE Systemprivileg, 43
DELETE Objektprivileg, 43
DELETE Statement, 43
DENSE_RANK Funktion, 184
DESC, 82
DESCRIBE Statement, 101
DISABLE, 25
DISTINCT, 80
DIV Funktion, 185
DML Statement, 38
DML Statements
 DELETE, 43
 EXPORT, 52
 IMPORT, 44
 INSERT, 38
 MERGE, 40
 TRUNCATE, 43
 UPDATE, 39
DOUBLE PRECISION Datentyp, 108
DROP ANY CONNECTION Systemprivileg, 69
DROP ANY FUNCTION Systemprivileg, 30
DROP ANY SCHEMA Systemprivileg, 13
DROP ANY SCRIPT Systemprivileg, 35
DROP ANY TABLE Systemprivileg, 19
DROP ANY VIEW Systemprivileg, 27
DROP ANY VIRTUAL SCHEMA Systemprivileg, 13
DROP CONNECTION Statement, 69
DROP DEFAULT Klausel, 22
DROP FUNCTION Statement, 30
DROP ROLE Statement, 66
DROP SCHEMA Statement, 13
DROP SCRIPT Statement, 35
DROP TABLE Statement, 19
DROP USER Statement, 65
DROP USER Systemprivileg, 65
DROP VIEW Statement, 27
DUMP Funktion, 185
- E**
EDIT_DISTANCE Funktion, 186
emit(), 314, 320, 327, 333
EMITS, 306
ENABLE, 25
ETL, 275
 Benutzerdefinierter EXPORT mittels UDFs, 278
 Benutzerdefinierter IMPORT mittels UDFs, 276
 CSV Datenformat, 279
 Dateiformate, 279
 EXPORT-Befehl, 52
 FBV Datenformat, 281
 Hadoop Support, 278
 IMPORT-Befehl, 44
 Scripting, 276
 SQL-Befehle, 275
 Virtuelle Schemas, 279
EXAplus, 371
 Befehlsreferenz, 379
 Benutzeroberfläche, 372
 Installation, 371
 Konsolenmodus, 376
EXAplus-Befehle, 380
 @, 380
 @@, 381
 ACCEPT, 381
 BATCH, 381
 COLUMN, 382
 CONNECT, 384
 DEFINE, 385
 DISCONNECT, 385
 EXIT, 385
 HOST, 386
 PAUSE, 386
 PROMPT, 386
 QUIT, 385
 SET AUTOCOMMIT, 387
 SET AUTOCOMPLETION, 387
 SET COLSEPARATOR, 387
 SET DEFINE, 388

SET ENCODING, 388
SET ESCAPE, 389
SET FEEDBACK, 389
SET HEADING, 389
SET LINESIZE, 390
SET NULL, 390
SET NUMFORMAT, 391
SET PAGESIZE, 391
SET SPOOL ROW SEPARATOR, 392
SET TIME, 392
SET TIMING, 392
SET TRUNCATE HEADING, 393
SET VERBOSE, 393
SHOW, 394
SPOOL, 394
START, 380
TIMING, 395
UNDEFINE, 395
WHENEVER, 395
EXA_TIME_ZONES, 96, 98, 110, 172
EXCEPT, 84
(Siehe auch MINUS)
EXECUTE ANY FUNCTION Systemprivileg, 28
EXECUTE ANY SCRIPT Systemprivileg, 31, 94
EXECUTE Objektprivileg, 28, 31, 94
EXECUTE SCRIPT Statement, 93
EXISTS Prädikat, 140
EXP Funktion, 187
EXPLAIN VIRTUAL Statement, 102
EXPORT Statement, 52
EXTRACT Funktion, 187

F

FBV Datenformat, 281
FIRST_VALUE Funktion, 188
FLOOR Funktion, 189
FLUSH STATISTICS Statement, 106
FOR Schleife, 29
FOREIGN KEY, 25
Überprüfung der Eigenschaft, 89
Format-Modelle, 130
Datum/Zeit, 130
Numerische, 132
FROM, 80
FROM_POSIX_TIME Funktion, 189
FULL OUTER JOIN, 79
Funktion
Funktion umbenennen, 35
Funktionen, 144
Aggregationsfunktionen, 148
APPROXIMATE_COUNT_DISTINCT, 156
AVG, 158
CORR, 173
COUNT, 175
COVAR_POP, 176
COVAR_SAMP, 177
FIRST_VALUE, 188
GROUPING[_ID], 191
GROUP_CONCAT, 190
LAST_VALUE, 199
MAX, 208
MEDIAN, 209
MIN, 210
PERCENTILE_CONT, 218
PERCENTILE_DISC, 220
REGR_*, 228
REGR_AVGX, 229
REGR_AVGY, 229
REGR_COUNT, 229
REGR_INTERCEPT, 229
REGR_R2, 229
REGR_SLOPE, 228
REGR_SXX, 229
REGR_SXY, 229
REGR_SYY, 229
STDDEV, 243
STDDEV_POP, 244
STDDEV_SAMP, 245
ST_INTERSECTION, 122
ST_UNION, 123
SUM, 246
VARIANCE, 262
VAR_POP, 261
VAR_SAMP, 261
Analytische Funktionen, 148
AVG, 158
CORR, 173
COUNT, 175
COVAR_POP, 176
COVAR_SAMP, 177
DENSE_RANK, 184
FIRST_VALUE, 188
LAG, 198
LAST_VALUE, 199
LEAD, 201
MAX, 208
MEDIAN, 209
MIN, 210
PERCENTILE_CONT, 218
PERCENTILE_DISC, 220
RANK, 224
RATIO_TO_REPORT, 224
REGR_*, 228
REGR_AVGX, 229
REGR_AVGY, 229
REGR_COUNT, 229
REGR_INTERCEPT, 229
REGR_R2, 229
REGR_SLOPE, 228
REGR_SXX, 229
REGR_SXY, 229
REGR_SYY, 229
ROW_NUMBER, 234
STDDEV, 243
STDDEV_POP, 244

STDDEV_SAMP, 245
SUM, 246
VARIANCE, 262
VAR_POP, 261
VAR_SAMP, 261
Benutzerdefiniert
 CREATE FUNCTION, 27
 DROP FUNCTION, 30
Bitfunktionen, 147
 BIT_AND, 159
 BIT_CHECK, 159
 BIT_LROTATE, 160
 BIT_LSHIFT, 161
 BIT_NOT, 162
 BIT_OR, 162
 BIT_RROTATE, 163
 BIT_RSHIFT, 163
 BIT_SET, 164
 BIT_TO_NUM, 164
 BIT_XOR, 165
Datum/Zeit
 FROM_POSIX_TIME, 189
 POSIX_TIME, 222
Datum/Zeit Funktionen, 146
 ADD_DAYS, 152
 ADD_HOURS, 152
 ADD_MINUTES, 153
 ADD_MONTHS, 154
 ADD_SECONDS, 154
 ADD_WEEKS, 155
 ADD_YEARS, 155
 CONVERT_TZ, 172
 CURDATE, 177
 CURRENT_DATE, 178
 CURRENT_TIMESTAMP, 180
 DATE_TRUNC, 181
 DAY, 181
 DAYS_BETWEEN, 182
 DBTIMEZONE, 182
 EXTRACT, 187
 HOUR, 194
 HOURS_BETWEEN, 195
 LOCALTIMESTAMP, 204
 MINUTE, 211
 MINUTES_BETWEEN, 211
 MONTH, 212
 MONTHS_BETWEEN, 213
 NOW, 214
 NUMTODSINTERVAL, 216
 NUMTOYMINTEGER, 216
 ROUND (datetime), 232
 SECOND, 237
 SECONDS_BETWEEN, 237
 SESSIONTIMEZONE, 238
 SYSDATE, 248
 SYSTIMESTAMP, 248
 TO_CHAR (datetime), 250
 TO_DATE, 251
 TO_DSINTERVAL, 252
 TO_TIMESTAMP, 253
 TO_YMINTERVAL, 254
 TRUNC[ATE] (datetime), 256
 WEEK, 263
 YEAR, 263
 YEARS_BETWEEN, 264
Geodaten-Funktionen
 ST_*, 241
 ST_AREA, 121
 ST_BOUNDARY, 122
 ST_BUFFER, 122
 ST_CENTROID, 122
 ST_CONTAINS, 122
 ST_CONVEXHULL, 122
 ST_CROSSES, 122
 ST_DIFFERENCE, 122
 ST_DIMENSION, 122
 ST_DISJOINT, 122
 ST_DISTANCE, 122
 ST_ENDPOINT, 121
 ST_ENVELOPE, 122
 ST_EQUALS, 122
 ST_EXTERIORRING, 122
 ST_FORCE2D, 122
 ST_GEOOMETRYN, 122
 ST_GEOENTRYTYPE, 122
 ST_INTERIORRINGN, 122
 ST_INTERSECTION, 122
 ST_INTERSECTS, 122
 ST_ISCLOSED, 121
 ST_ISEMPTY, 122
 ST_ISRING, 121
 ST_ISSIMPLE, 122
 ST_LENGTH, 121
 ST_NUMGEOMETRIES, 122
 ST_NUMINTERIORRINGS, 122
 ST_NUMPOINTS, 121
 ST_OVERLAPS, 123
 ST_POINTN, 121
 ST_SETSRID, 123
 ST_STARTPOINT, 121
 ST_SYMDIFFERENCE, 123
 ST_TOUCHES, 123
 ST_TRANSFORM, 123
 ST_UNION, 123
 ST_WITHIN, 123
 ST_X, 121
 ST_Y, 121
Geodatenfunktionen, 146
Hierarchische Queries, 147
 CONNECT_BY_ISCYCLE, 170
 CONNECT_BY_ISLEAF, 171
 LEVEL, 203
 SYS_CONNECT_BY_PATH, 247
Konvertierungsfunktionen, 147
 CAST, 167
 CONVERT, 171

IS_BOOLEAN, 197	HASH_SHA[1], 193
IS_DATE, 197	HASH_TIGER, 193
IS_DSINTERVAL, 197	IPROC, 196
IS_NUMBER, 197	LEAST, 202
IS_TIMESTAMP, 197	NPROC, 214
IS_YMINTERVAL, 197	NULLIF, 215
NUMTODSINTERVAL, 216	NULLIFZERO, 215
NUMTOYMINTERVAL, 216	NVL, 217
TO_CHAR (datetime), 250	NVL2, 217
TO_CHAR (number), 251	ROWID, 234
TO_DATE, 251	SYS_GUID, 247
TO_DSINTERVAL, 252	USER, 259
TO_NUMBER, 253	VALUE2PROC, 260
TO_TIMESTAMP, 253	ZEROIFNULL, 264
TO_YMINTERVAL, 254	Zeichenkettenfunktionen, 145
Numerische Funktionen, 144	ASCII, 156
ABS, 151	BIT_LENGTH, 160
ACOS, 151	CHARACTER_LENGTH, 168
ASIN, 157	CH[A]R, 168
ATAN, 157	COLOGNE_PHONETIC, 169
ATAN2, 158	CONCAT, 170
CEIL, 167	DUMP, 185
CEILING, 167	EDIT_DISTANCE, 186
COS, 174	INSERT, 195
COSH, 174	INSTR, 196
COT, 175	LCASE, 200
DEGREES, 184	LEFT, 202
DIV, 185	LENGTH, 203
EXP, 187	LOCATE, 205
FLOOR, 189	LOWER, 207
LN, 204	LPAD, 207
LOG, 205	LTRIM, 208
LOG10, 206	MID, 210
LOG2, 206	OCTET_LENGTH, 218
MOD, 212	POSITION, 221
PI, 221	REGEXP_INSTR, 225
POWER, 222	REGEXP_REPLACE, 226
RADIANS, 223	REGEXP_SUBSTR, 227
RAND[OM], 223	REPEAT, 230
ROUND (number), 233	REPLACE, 230
SIGN, 238	REVERSE, 231
SIN, 239	RIGHT, 232
SINH, 239	RPAD, 236
SQRT, 241	RTRIM, 236
TAN, 249	SOUNDEX, 240
TANH, 249	SPACE, 240
TO_CHAR (number), 251	SUBSTR, 245
TRUNC[ATE] (number), 257	SUBSTRING, 245
Sonstige skalare Funktionen, 147	TRANSLATE, 255
CASE, 165	TRIM, 255
COALESCE, 169	UCASE, 257
CURRENT_SCHEMA, 178	UNICODE, 258
CURRENT_SESSION, 179	UNICODECHR, 258
CURRENT_STATEMENT, 179	UPPER, 259
CURRENT_USER, 180	
DECODE, 183	
GREATEST, 190	
HASH_MD5, 192	

G

Geodaten, 120

- Funktionen, 121
 ST_AREA, 121
 ST_BOUNDARY, 122
 ST_BUFFER, 122
 ST_CENTROID, 122
 ST_CONTAINS, 122
 ST_CONVEXHULL, 122
 ST_CROSSES, 122
 ST_DIFFERENCE, 122
 ST_DIMENSION, 122
 ST_DISJOINT, 122
 ST_DISTANCE, 122
 ST_ENDPOINT, 121
 ST_ENVELOPE, 122
 ST_EQUALS, 122
 ST_EXTERIORRING, 122
 ST_FORCE2D, 122
 ST_GEOMETRYN, 122
 ST_GEOMETRYTYPE, 122
 ST_INTERIORRINGN, 122
 ST_INTERSECTION, 122
 ST_INTERSECTS, 122
 ST_ISCLOSED, 121
 ST_ISEMPTY, 122
 ST_ISRING, 121
 ST_ISSIMPLE, 122
 ST_LENGTH, 121
 ST_NUMGEOMETRIES, 122
 ST_NUMINTERIORRINGS, 122
 ST_NUMPOINTS, 121
 ST_OVERLAPS, 123
 ST_POINTN, 121
 ST_SETSRID, 123
 ST_STARTPOINT, 121
 ST_SYMDIFFERENCE, 123
 ST_TOUCHES, 123
 ST_TRANSFORM, 123
 ST_UNION, 123
 ST_WITHIN, 123
 ST_X, 121
 ST_Y, 121
 Leere Menge, 120
 Objekte, 120
 GEOMETRY, 120
 GEOMETRYCOLLECTION, 120
 Leere Menge, 120
 LINEARRING, 120
 LINESTRING, 120
 MULTILINESTRING, 120
 MULTIPOINT, 120
 MULTIPOLYGON, 120
 POINT, 120
 POLYGON, 120
 GEOMETRY data type, 111
 GEOMETRY Objekt, 120
 GEOMETRYCOLLECTION Objekt, 120
 getBigDecimal(), 320
 getBoolean(), 320
 getDate(), 320
 getDouble(), 320
 getInteger(), 320
 getLong(), 320
 getString(), 320
 getTimestamp(), 320
 GRANT ANY CONNECTION Systemprivileg, 70, 72
 GRANT ANY OBJECT PRIVILEGE Systemprivileg, 69, 72
 GRANT ANY PRIORITY Systemprivileg, 69, 72
 GRANT ANY PRIVILEGE Systemprivileg, 69, 71, 72
 GRANT ANY ROLE Systemprivileg, 69, 72
 GRANT Statement, 69
 Graph Analyse, 80
 Graph Suche, 80
 GREATEST Funktion, 190
 GROUPING SETS, 82
 GROUPING[_ID] Funktion, 191
 GROUP_CONCAT Funktion, 190
- ## H
- Hadoop, 276, 279
 HASH_MD5 Funktion, 192
 HASH_SHA[1] Funktion, 193
 HASH_TIGER Funktion, 193
 HAVING, 82
 HCatalog, 279
 HOUR Funktion, 194
 HOURS_BETWEEN Funktion, 195
- ## I
- Identifier, 5
 Identity-Spalten, 19, 21, 38, 40, 42, 117
 Anzeigen, 118
 Beispiel, 117
 Setzen und Ändern, 118
 IF EXISTS
 in DROP COLUMN Statement, 22
 in DROP CONNECTION Statement, 69
 in DROP FUNCTION Statement, 31
 in DROP ROLE Statement, 66
 in DROP SCHEMA Statement, 13
 in DROP SCRIPT Statement, 35
 in DROP TABLE Statement, 19
 in DROP USER Statement, 65
 in DROP VIEW Statement, 27
 IF Verzweigung, 28
 IMPORT Statement, 44, 82
 IN Prädikat, 141
 INNER JOIN, 79
 INSERT ANY TABLE Systemprivileg, 38
 INSERT Funktion, 195
 INSERT Objektprivileg, 38
 INSERT Statement, 38
 INSTR Funktion, 196
 INTERSECT, 84
 INTERVAL DAY TO SECOND Datentyp, 111

INTERVAL YEAR TO MONTH Datentyp, 111
IPROC Funktion, 196
IS NULL Prädikat, 142
IS_BOOLEAN Funktion, 197
IS_DATE Funktion, 197
IS_DSINTERVAL Funktion, 197
IS_NUMBER Funktion, 197
IS_TIMESTAMP Funktion, 197
IS_YMINTERVAL Funktion, 197

J

Java, 319
JDBC-Treiber, 406
 Benutzung, 407
 Best-Practice, 411
 Standards, 406
 Systemvoraussetzungen, 406
Join, 79, 80
 CROSS JOIN, 79
 FULL OUTER JOIN, 79
 INNER JOIN, 79
 LEFT JOIN, 79
 OUTER JOIN, 79
 RIGHT JOIN, 79

K

Kerberos, 63, 403, 410
KILL Statement, 95
Kommentar, 5

L

LAG Funktion, 198
LAST_VALUE Funktion, 199
LCASE Funktion, 200
LDAP, 64
LEAD Funktion, 201
LEAST Funktion, 202
LEFT Funktion, 202
LEFT JOIN, 79
LENGTH Funktion, 203
LEVEL, 80
LEVEL Funktion, 203
LIKE Prädikat, 143
LIMIT, 82
LINEARRING Objekt, 120
LINESTRING Objekt, 120
Literale, 125
 Beispiele, 125
 Boolesche, 126
 Datum/Zeit, 126
 Intervalle, 126
 NULL, 128
 Numerische, 125
 Zeichenketten, 128
LN Funktion, 204
LOCAL, 80
LOCALTIMESTAMP Funktion, 204

LOCATE Funktion, 205
LOG Funktion, 205
LOG10 Funktion, 206
LOG2 Funktion, 206
LOWER Funktion, 207
LPAD Funktion, 207
LTRIM Funktion, 208
Lua, 313

M

MapReduce, 311
MAX Funktion, 208
MEDIAN Funktion, 209
MERGE Statement, 40
MID Funktion, 210
MIN Funktion, 210
MINUS, 84
MINUTE Funktion, 211
MINUTES_BETWEEN Funktion, 211
MOD Funktion, 212
MONTH Funktion, 212
MONTHS_BETWEEN Funktion, 213
MULTILINESTRING Objekt, 120
MULTIPOINT Objekt, 120
MULTIPOLYGON Objekt, 120

N

next(), 314, 320, 327
next_row(), 333
NICE, 97, 274, 431, 440, 455, 460, 469
NLS_DATE_FORMAT, 96, 99
NLS_DATE_LANGUAGE, 96, 99
NLS_FIRST_DAY_OF_WEEK, 96, 99, 181, 233, 256
NLS_NUMERIC_CHARACTERS, 96, 99
NLS_TIMESTAMP_FORMAT, 96, 99
NOCYCLE, 80
NOT NULL, 25
NOT Prädikat, 140
NOW Funktion, 214
NPROC Funktion, 214
NULL
 Literal, 128
NULLIF Funktion, 215
NULLIFZERO Funktion, 215
NULLS FIRST, 82
NULLS LAST, 82
NUMTODSINTERVAL Funktion, 216
NUMTOYMINTEGER Funktion, 216
NVL Funktion, 217
NVL2 Funktion, 217

O

Objektprivileg
 ACCESS, 484
 ALTER, 14, 20, 484
 DELETE, 43, 484
 EXECUTE, 28, 31, 94, 484

INSERT, 38, 484
REFERENCES, 484
REFRESH, 14, 484
SELECT, 76, 484
UPDATE, 39, 484
Zusammenfassung, 481
OCTET_LENGTH Funktion, 218
ODBC-Treiber, 397
 Best-Practice, 405
 Connection-String Parameter, 402
 Linux/Unix-Version, 400
 Standards, 397
 Windows-Version, 397
 Bekannte Probleme, 399
 Connection Pooling, 399
 Installation, 397
 Konfiguration, 398
 Systemvoraussetzungen, 397
Zeichensätze, 405
OPEN SCHEMA Statement, 100
Operatoren, 135
 Arithmetische, 135
 CONNECT BY Operatoren, 137
 Konkatenationsoperator, 136
OR Prädikat, 140
OR REPLACE Option
 in CREATE CONNECTION, 67
 in CREATE FUNCTION, 28
 in CREATE SCRIPT, 31
 in CREATE TABLE, 15
 in CREATE VIEW, 26
ORDER BY, 82
OUTER JOIN, 79

P

Pareto, 367
Passwort, 63
PERCENTILE_CONT Funktion, 218
PERCENTILE_DISC Funktion, 220
PI Funktion, 221
POINT Objekt, 120
POLYGON Objekt, 120
POSITION Funktion, 221
POSIX_TIME Funktion, 222
POWER Funktion, 222
PREFERRING, 40, 43, 81, 367, 369
PRELOAD Statement, 107
PRIMARY KEY, 25
 Überprüfung der Eigenschaft, 87
PRIOR, 80
Priorität, 273, 430, 431, 432, 439, 440, 441, 446, 455, 456, 460, 469
 Beispiel, 274
 Einleitung, 273
 NICE, 97, 274
 Priorität entziehen, 72
 Priorität zuweisen, 69

Prioritäten in Exasol, 273
PROFILE, 97, 99
Profiling, 364, 461, 462, 475, 476
 Aktivierung und Auswertung, 364
 Beispiel, 365
 Einleitung, 364
 System-Parameter, 97, 99
Protocol Buffers, 344
Prädikate, 138
 BETWEEN, 140
 EXISTS, 140
 IN, 141
 IS NULL, 142
 LIKE, 143
 Logische Verknüpfung, 139
 AND, 139
 NOT, 140
 OR, 140
 Präzedenz der Prädikate, 138
 REGEXP_LIKE, 142
 Vergleiche, 138
Präprozessor, 353
Python, 326

Q

Query Cache, 96, 99
Query Cache für SQL-Anfragen, 96, 99
Query Timeout, 97, 99
QUERY_CACHE, 96, 99
QUERY_TIMEOUT, 97, 99

R

R (Programmiersprache), 331
RADIAN Funktion, 223
RAND[OM] Funktion, 223
RANK Funktion, 224
RATIO_TO_REPORT Funktion, 224
Rechteverwaltung, 269
 Beispiel, 271
 Benutzer, 269
 Metainformationen, 271
 Privilegien, 270
 Rollen, 269
 Zugriffskontrolle bei SQL-Befehlen, 270
RECOMPRESS Statement, 103
REFRESH Objektprivileg, 14
REGEXP_INSTR Funktion, 225
REGEXP_LIKE Prädikat, 142
REGEXP_REPLACE Funktion, 226
REGEXP_SUBSTR Funktion, 227
REGR_* Funktionen, 228
REGR_AVGX Funktion, 229
REGR_AVGY Funktion, 229
REGR_COUNT Funktion, 229
REGR_INTERCEPT Funktion, 229
REGR_R2 Funktion, 229
REGR_SLOPE Funktion, 228

REGR_SXX Funktion, 229
REGR_SXY Funktion, 229
REGR_SYY Funktion, 229
Reguläre Ausdrücke, 8
 Beispiele, 8
 Sprachelemente, 8
RENAME Statement, 35
REORGANIZE Statement, 104
REPEAT Funktion, 230
REPLACE Funktion, 230
Reservierte Wörter, 7
reset(), 314, 320, 327, 333
RESTRICT
 in DROP SCHEMA Statement, 13
 in DROP VIEW Statement, 27
RETURNS, 306
REVERSE Funktion, 231
REVOKE Statement, 72
RIGHT Funktion, 232
RIGHT JOIN, 79
ROLLBACK Statement, 93
Rollen
 Rechte entziehen, 72
 Rechte zuweisen, 69
 Rollen anlegen, 65
 Rollen löschen, 66
ROLLUP, 81
ROUND (datetime) Funktion, 232
ROUND (number) Funktion, 233
ROWID, 234
ROW_NUMBER Funktion, 234
RPAD Funktion, 236
RTRIM Funktion, 236

S

SCALAR, 306
Schema
 anlegen, 12
 Besitzer ändern, 14
 löschen, 13
 Schema kommentieren, 36
 schließen, 101
 umbenennen, 35
 öffnen, 100
Schemaobjekte, 12
Schemaqualifizierte Bezeichner, 7
Schnittstellen
 ADO.NET Data Provider, 412
 EXAplus, 371
 JDBC-Treiber, 406
 ODBC-Treiber, 397
 SDK, 421
 WebSockets, 421
Scripting
 Arrays, 285
 Ausführungs-Blöcke, 286
 Beispiel, 282
 Debug-Output, 297
 Dictionary Tables, 286
 Einfache Variablen, 285
 Error Handling, 290
 Funktion
 pairs(), 286
 Funktionen, 289
 error(), 290
 exit(), 295
 import(), 294
 join(), 297
 output(), 297
 pcall(), 290
 pqquery(), 291
 query(), 291
 quote(), 297
 sqlparsing.find(), 355
 sqlparsing.getsqltext(), 356
 sqlparsing.isany(), 355
 sqlparsing.iscomment(), 354
 sqlparsing.isidentifier(), 354
 sqlparsing.iskeyword(), 354
 sqlparsing.isnumericalliteral(), 355
 sqlparsing.isstringliteral(), 355
 sqlparsing.iswhitespace(), 354
 sqlparsing.iswhitespaceorcomment(), 354
 sqlparsing.normalize(), 355
 sqlparsing.setsqltext(), 356
 sqlparsing.tokenize(), 354
 string.find(), 298
 string.format(formatstring, e1, e2, ...), 300
 string.gmatch(s, pattern), 299
 string.gsub(s, pattern, repl [, n]), 299
 string.len(s), 300
 string.lower(s), 300
 string.match(s, pattern [, init]), 299
 string.rep(s, n), 300
 string.reverse(s), 300
 string.sub(s, i [, j]), 299
 string.upper(s), 300
 table.concat(), 305
 table.insert(), 304
 table.maxn(), 305
 table.remove(), 304
 table.sort(), 305
Hilfsfunktionen für SQL Bezeichner, 297
Importieren externer Skripte, 294
Internet Zugriff, 303
Kommentare, 283
Kontrollstrukturen, 287
Lexikalische Konventionen, 283
Math-Bibliothek, 303
Metadaten, 296
nil, 284
Operatoren, 288
Parametrisierte SQL Befehle, 292
Rückgabewerte, 295
Skript-Parameter, 293

- Sonstige Anmerkungen, 298
Sprachumfang, 282
SQL Befehle ausführen, 291
SQL-Parsing, 303
String-Bibliothek, 298
Systemtabellen, 305
Table-Bibliothek, 304
Typen & Werte, 284
Unicode, 302
XML-Parsing, 302
Scripting-Programmierung, 282
SCRIPT_LANGUAGES , 97, 99
SDK, 421
 CLI, 421
SECOND Funktion, 237
SECONDS_BETWEEN Funktion, 237
SELECT, 76
SELECT ANY TABLE Systemprivileg, 76
SELECT INTO Statement, 18
SELECT Objektprivileg, 76
SESSIONTIMEZONE Funktion, 238
SET, 306
SIGN Funktion, 238
SIN Funktion, 239
SINH Funktion, 239
size(), 314, 320, 327, 333
Skript
 Skript ausführen, 93
 Skript erzeugen, 31
 Skript kommentieren, 36
 Skript löschen, 35
 Skript umbenennen, 35
Skyline, 367, 368
 Beispiel, 368
 Motivation, 367
 So funktioniert Skyline, 367
 Syntaxelemente, 369
SOUNDEX Funktion, 240
SPACE Funktion, 240
Spalte
 Datentyp ändern, 19
 Defaultwert setzen, 19
 Identity-Spalte setzen, 19
 Spalte hinzufügen, 19
 Spalte kommentieren, 36
 Spalte löschen, 19
 Spalte umbenennen, 19
Spalten-Alias, 80
SQL Befehl, 12
SQL-Präprozessor, 353
 Beispiele, 357
 Best Practice, 356
 Bibliothek sqlparsing, 353
 Einleitung, 353
Funktionen
 sqlparsing.find(), 355
 sqlparsing.getsqltext(), 356
 sqlparsing.isany(), 355
 sqlparsing.iscomment(), 354
 sqlparsing.isidentifier(), 354
 sqlparsing.iskeyword(), 354
 sqlparsing.isnumericaliteral(), 355
 sqlparsing.isstringliteral(), 355
 sqlparsing.iswhitespace(), 354
 sqlparsing.iswhitespaceorcomment(), 354
 sqlparsing.normalize(), 355
 sqlparsing.setsqltext(), 356
 sqlparsing.tokenize(), 354
SQL-Standard, 491
SQL_PREPROCESSOR_SCRIPT, 97, 100
SQRT Funktion, 241
START WITH, 80
STDDEV Funktion, 243
STDDEV_POP Funktion, 244
STDDEV_SAMP Funktion, 245
ST_* Funktionen, 241
ST_AREA Funktion, 121
ST_BOUNDARY Funktion, 122
ST_BUFFER Funktion, 122
ST_CENTROID Funktion, 122
ST_CONTAINS Funktion, 122
ST_CONVEXHULL Funktion, 122
ST_CROSSES Funktion, 122
ST_DIFFERENCE Funktion, 122
ST_DIMENSION Funktion, 122
ST_DISJOINT Funktion, 122
ST_DISTANCE Funktion, 122
ST_ENDPOINT Funktion, 121
ST_ENVELOPE Funktion, 122
ST_EQUALS Funktion, 122
ST_EXTERIORRING Funktion, 122
ST_FORCE2D Funktion, 122
ST_GEOENTRYN Funktion, 122
ST_GEOENTRYTYPE Funktion, 122
ST_INTERIORRINGN Funktion, 122
ST_INTERSECTION Funktion, 122
ST_INTERSECTS Funktion, 122
ST_ISCLOSED Funktion, 121
ST_ISEMPTY Funktion, 122
ST_ISRING Funktion, 121
ST_ISSIMPLE Funktion, 122
ST_LENGTH Funktion, 121
ST_NUMGEOMETRIES Funktion, 122
ST_NUMINTERIORRINGS Funktion, 122
ST_NUMPOINTS Funktion, 121
ST_OVERLAPS Funktion, 123
ST_POINTN Funktion, 121
ST_SETSRID Funktion, 123
ST_STARTPOINT Funktion, 121
ST_SYMDIFFERENCE Funktion, 123
ST_TOUCHES Funktion, 123
ST_TRANSFORM Funktion, 123
ST_UNION Funktion, 123
ST_WITHIN Funktion, 123
ST_X Funktion, 121
ST_Y Funktion, 121

SUBSTR Funktion, 245
SUBSTRING Funktion, 245
SUM Funktion, 246
SYSDATE Funktion, 248
System-Parameter
 CONSTRAINT_STATE_DEFAULT, 97, 99
 DEFAULT_LIKE_ESCAPE_CHARACTER, 96, 99
 NICE, 97
 NLS_DATE_FORMAT, 96, 99
 NLS_DATE_LANGUAGE, 96, 99
 NLS_FIRST_DAY_OF_WEEK, 96, 99
 NLS_NUMERIC_CHARACTERS, 96, 99
 NLS_TIMESTAMP_FORMAT, 96, 99
 PROFILE, 97, 99
 QUERY_CACHE, 96, 99
 QUERY_TIMEOUT, 97, 99
 SCRIPT_LANGUAGES , 97, 99
 SQL_PREPROCESSOR_SCRIPT, 97, 100
 TIMESTAMP_ARITHMETIC_BEHAVIOR, 96, 98
 TIME_ZONE, 96, 98
 TIME_ZONE_BEHAVIOR, 96, 98

Systemprivileg
 ACCESS ANY CONNECTION, 482
 ALTER ANY CONNECTION, 35, 68, 482
 ALTER ANY SCHEMA, 14, 482
 ALTER ANY TABLE, 20, 482
 ALTER ANY VIRTUAL SCHEMA, 14, 482
 ALTER ANY VIRTUAL SCHEMA REFRESH, 14, 482
 ALTER SYSTEM, 482
 ALTER USER, 64, 482
 CREATE ANY FUNCTION, 28, 483
 CREATE ANY SCRIPT, 31, 483
 CREATE ANY TABLE, 15, 18, 482
 CREATE ANY VIEW, 483
 CREATE CONNECTION, 67, 482
 CREATE FUNCTION, 28, 483
 CREATE ROLE, 35, 482
 CREATE SCHEMA, 12, 482
 CREATE SCRIPT, 31, 483
 CREATE SESSION, 63, 482
 CREATE TABLE, 15, 18, 482
 CREATE USER, 35, 63, 482
 CREATE VIEW, 483
 CREATE VIRTUAL SCHEMA, 12, 482
 DELETE ANY TABLE, 43, 482
 DROP ANY CONNECTION, 482
 DROP ANY FUNCTION, 483
 DROP ANY ROLE, 482
 DROP ANY SCHEMA, 13, 482
 DROP ANY SCRIPT, 483
 DROP ANY TABLE, 19, 483
 DROP ANY VIEW, 483
 DROP ANY VIRTUAL SCHEMA, 482
 DROP ANY VIRTUAL SCHEMA Systemprivileg, 13
 DROP USER, 65, 482
 EXECUTE ANY FUNCTION, 28, 483

EXECUTE ANY SCRIPT, 31, 94, 484
GRANT ANY CONNECTION, 70, 72, 482
GRANT ANY OBJECT PRIVILEGE, 69, 72, 482
GRANT ANY PRIORITY, 69, 72, 482
GRANT ANY PRIVILEGE, 69, 71, 72, 482
GRANT ANY ROLE, 69, 72, 482
INSERT ANY TABLE, 38, 483
KILL ANY SESSION, 482
SELECT ANY DICTIONARY, 483
SELECT ANY TABLE, 76, 483
UPDATE ANY TABLE, 39, 483
USE ANY CONNECTION, 44, 482
USE ANY TABLE, 53
Zusammenfassung, 481

Systemtabellen, 425
CAT, 479
DUAL, 479
EXA_ALL_COLUMNS, 425
EXA_ALL_CONNECTIONS, 426
EXA_ALL_CONSTRAINTS, 426
EXA_ALL_CONSTRAINT_COLUMNS, 427
EXA_ALL_DEPENDENCIES, 427
EXA_ALL_FUNCTIONS, 427
EXA_ALL_INDICES, 428
EXA_ALL_OBJECTS, 429
EXA_ALL_OBJECT_SIZES, 430
EXA_ALL_OBJ_PRIVS, 428
EXA_ALL_OBJ_PRIVS_MADE, 428
EXA_ALL_OBJ_PRIVS_REC'D, 429
EXA_ALL_ROLES, 430
EXA_ALL_SCRIPTS, 430
EXA_ALL_SESSIONS, 431
EXA_ALL_TABLES, 431
EXA_ALL_USERS, 432
EXA_ALL_VIEWS, 432
EXA_ALL_VIRTUAL_COLUMNS, 432
EXA_ALL_VIRTUAL_SCHEMA_PROPERTIES, 433
EXA_ALL_VIRTUAL_TABLES, 433
EXA_DBA_COLUMNS, 433
EXA_DBA_CONNECTIONS, 434
EXA_DBA_CONNECTION_PRIVS, 434
EXA_DBA_CONSTRAINTS, 435
EXA_DBA_CONSTRAINT_COLUMNS, 435
EXA_DBA_DEPENDENCIES, 436
EXA_DBA_DEPENDENCIES_RECURSIVE, 436
EXA_DBA_FUNCTIONS, 437
EXA_DBA_INDICES, 437
EXA_DBA_OBJECTS, 438
EXA_DBA_OBJECT_SIZES, 438
EXA_DBA_OBJ_PRIVS, 437
EXA_DBA_RESTRICTED_OBJ_PRIVS, 435
EXA_DBA_ROLES, 439
EXA_DBA_ROLE_PRIVS, 439
EXA_DBA_SCRIPTS, 439
EXA_DBA_SESSIONS, 440
EXA_DBA_SYS_PRIVS, 440
EXA_DBA_TABLES, 441

- EXA_DB_SIZE_DAILY, 465
- EXA_DB_SIZE_HOURLY, 464
- EXA_DB_SIZE_LAST_DAY, 464
- EXA_DB_SIZE_MONTHLY, 465
- EXA_MONITOR_DAILY, 467
- EXA_MONITOR_HOURLY, 467
- EXA_MONITOR_LAST_DAY, 466
- EXA_MONITOR_MONTHLY, 468
- EXA_SQL_DAILY, 471
- EXA_SQL_HOURLY, 470
- EXA_SQL_LAST_DAY, 469
- EXA_SQL_MONTHLY, 472
- EXA_SYSTEM_EVENTS, 473
- EXA_USAGE_DAILY, 474
- EXA_USAGE_HOURLY, 474
- EXA_USAGE_LAST_DAY, 473
- EXA_USAGE_MONTHLY, 474
- EXA_USER_PROFILE_LAST_DAY, 475
- EXA_USER_PROFILE_RUNNING, 476
- EXA_USER_SESSIONS_LAST_DAY, 477
- EXA_USER_TRANSACTION_CONFLICTS_LAST_DAY, 477
- SYSTIMESTAMP Funktion, 248
- SYS_CONNECT_BY_PATH, 80
- SYS_CONNECT_BY_PATH Funktion, 247
- SYS_GUID Funktion, 247
- T**
- Tabelle
 - Alle Zeilen löschen, 43
 - Exportieren, 52
 - Identity-Spalte setzen, 19
 - Importieren, 44
 - Laden, 44, 52
 - SELECT, 76
 - Spalte hinzufügen, 19
 - Spalte löschen, 19
 - Spalte umbenennen, 19
 - Spalten-Defaultwert setzen, 19
 - Spalten-Typ ändern, 19
 - Tabelle anlegen, 15, 18
 - Tabelle kommentieren, 36
 - Tabelle löschen, 19
 - Tabelle rekompriemieren, 103
 - Tabelle umbenennen, 35
 - Tabellenoperatoren, 84
 - Typen anzeigen, 101
 - Werte ändern, 39
 - Zeilen einfügen, 38
 - Zeilen löschen, 43
 - Änderungsdaten übernehmen, 40
- Tabellenoperatoren, 84
- TAN Funktion, 249
- TANH Funktion, 249
- Timeout für SQL-Anfragen, 97, 99
- TIMESTAMP Datentyp, 109
- Statistiken
 - EXA_DB_AUDIT_SESSIONS, 459
 - EXA_DB_AUDIT_SQL, 460
 - EXA_DB_PROFILE_LAST_DAY, 461
 - EXA_DB_PROFILE_RUNNING, 462
 - EXA_DB_SESSIONS_LAST_DAY, 463
 - EXA_DB_TRANSACTION_CONFLICTS, 463

-
- TIMESTAMP WITH LOCAL TIME ZONE Datentyp, 109
TIMESTAMP_ARITHMETIC_BEHAVIOR, 96, 98
TIME_ZONE, 96, 98
TIME_ZONE_BEHAVIOR, 96, 98, 109
TO_CHAR (datetime) Funktion, 250
TO_CHAR (number) Funktion, 251
TO_DATE Funktion, 251
TO_DSINTERVAL Funktion, 252
TO_NUMBER Funktion, 253
TO_TIMESTAMP Funktion, 253
TO_YMINTERVAL Funktion, 254
Transaction, 271
Transaktion, 92, 93, 106, 267, 459
Transaktions-Management, 267
Transaktionskonflikt, 448, 459, 463, 477
TRANSLATE Funktion, 255
TRIM Funktion, 255
TRUNCATE AUDIT LOGS Statement, 105
TRUNCATE Statement, 43
TRUNC[ATE] (datetime) Funktion, 256
TRUNC[ATE] (number) Funktion, 257
- U**
UCASE Funktion, 257
UDF Skripte, 306
Aggregats- und Analytische Funktionen, 308
Benutzerdefiniertes ETL mittels UDFs, 312
BucketFS, 339
cleanup(), 313, 320, 326, 331
Dynamische Eingabe- und Ausgabe-Parameter, 309
Dynamische Parameterliste, 313, 321, 326, 332
Einführende Beispiele, 307
Einleitung, 306
emit(), 314, 320, 327, 333
EMITS, 306
init(), 320
Java, 319
Lua, 313
MapReduce Programme, 311
Metadaten, 321
next(), 314, 320, 327
next_row(), 333
ORDER BY, 307
Parameter, 313, 320, 326, 332
Performance, 307
Python, 326
R, 331
reset(), 314, 320, 327, 333
RETURNS, 306
run(), 313, 320, 326, 331
SCALAR, 306
SET, 306
size(), 314, 320, 327, 333
Skalare Funktionen, 307
Zugriff auf externe Dienste, 312
UNICODE Funktion, 258
UNICODECHR Funktion, 258
UNION [ALL], 84
UNIQUE
Überprüfung der Eigenschaft, 88
UPDATE ANY TABLE Systemprivileg, 39
UPDATE Objektprivileg, 39, 43
UPDATE Statement, 39
UPPER Funktion, 259
USER Funktion, 259
USING, 80
- V**
VALUE2PROC Funktion, 260
VARCHAR Datentyp, 112
Variablenzuweisung, 28
VARIANCE Funktion, 262
VAR_POP Funktion, 261
VAR_SAMP Funktion, 261
Verbindung
Verbindung entziehen, 72
Verbindung erzeugen, 67
Verbindung kommentieren, 36
Verbindung löschen, 69
Verbindung umbenennen, 35
Verbindung zuweisen, 69
Verbindungsdaten ändern, 68
View
INVALID, 27
Status, 27
View anlegen, 26
View kommentieren, 36
View löschen, 27
View umbenennen, 35
Virtuelle Schemas, 346
Adapter und Properties, 347
Details für Experten, 350
EXPLAIN VIRTUAL, 102
Meta-Daten, 350
Privilegien zur Administration, 349
Virtuelle Schemas und Tabellen, 346
Zugriffs-Konzept, 348
- W**
WebSockets, 421
WEEK Funktion, 263
WHERE, 80
WHILE Schleife, 29
WITH, 80
- Y**
YEAR Funktion, 263
YEARS_BETWEEN Funktion, 264
- Z**
Zeitzonen, 172
ZEROIFNULL Funktion, 264
ZeroMQ, 344