Die Oracle-Datenbank wird stetig weiterentwickelt. Mittlerweile liegt sie in Version 11*g* R2 für alle gängigen Betriebssysteme vor. Mit jeder neuen Version nimmt auch der Funktionsumfang der Datenbank zu, der mit kostenpflichtigen Zusatzoptionen sogar noch erweiterbar ist.

Data-Warehouse-Features der Datenbank 11*g* R2

Timo Bergenthal, OPITZ CONSULTING GmbH

Viele der Features haben ein sehr spezielles Anwendungsgebiet und werden daher nur von einem kleinen Anwenderkreis benötigt. Andere Features gehören zum Allgemeingut und dürfen in keinem Data Warehouse (DWH) fehlen. In diesem Artikel erhalten DWH-Entwickler einen groben Überblick über einige für das Data Warehousing relevante Features der Oracle-Datenbank. Der Nutzen und die möglichen Anwendungsfälle der Features stehen hierbei im Vordergrund.

Beladung eines Data Warehouse

Ein DWH ist nach der Definition von William H. Inmon eine themenorientierte, integrierte, zeitorientierte und nicht flüchtige Sammlung von Daten [1]. Zu deren Aufbau wird meist ein relationales Datenbank-Managementsystem (RDBMS) verwendet. Darin werden ausgewählte Daten aus verschiedenen Quellen gesammelt, integriert und über einen längeren Zeitraum gespeichert, damit zeitliche Analysen möglich werden.

Mit welchen Mitteln können Daten aus anderen Systemen in eine Oracle-Datenbank geladen werden? Es wird davon ausgegangen, dass im ersten Schritt der reine Abzug von Quelldaten in eine Stage ohne nennenswerte Transformationen durchgeführt wird.

Die Funktion „Change Data Capture" ist nützlich für die Erkennung von Änderungen in definierten Quelltabellen. Diese Änderungen sind in Change Tables protokolliert. Change Tables werden geschachtelt in Change Set und unter Umständen in Change Source. Dieser Part obliegt in jedem Fall einem Datenbank-Administrator in der Rolle des Publishers. Hinzu kommt die Vergabe von Zugriffsrechten für sogenannte „Subscriber" auf dem Niveau einzelner Spalten. Für jeden Typ von Change Data Capture werden die entsprechenden Änderungen an den Quelltabellen mit geringer Latenzzeit in die Change Tables überführt.

Die Subscriber melden mit ihrer Subscription Bedarf an den Change-Daten an. Daraufhin erfolgt über Views der Zugriff auf die Change-Daten. Initial weisen diese jedoch eine leere Ergebnismenge auf. Erst durch Aufruf der Prozedur „EXTEND_WINDOW" aus dem Package „DBMS_CDC_SUBSCRIBEQ" verschaffen sich die Subscriber Zugriff auf die aktuellen Change-Daten.

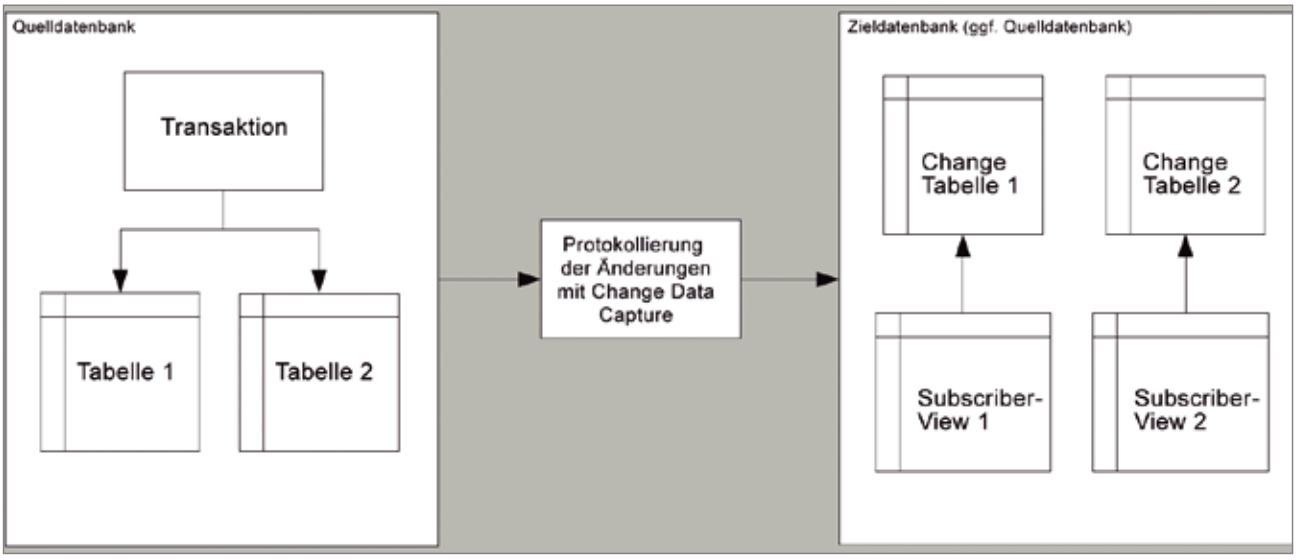

Abbildung 1: Change Data Capture

Ruft der Subscriber nach der Verarbeitung der Change-Daten die Prozedur „PURGE_WINDOW“ aus dem gleichen Package auf, sind diese im Anschluss daran für ihn nicht mehr sichtbar. Physisch werden sie aber erst gelöscht, wenn alle Subscriber mit Aufruf der Prozedur „PURGE_WINDOW“ angezeigt haben, dass sie die Change-Daten nicht mehr benötigen. Change Data Capture kann in verschiedenen Modi betrieben werden:

- Synchronous
- Asynchronous HotLog
- Asynchronous Distributed HotLog
- Asynchronous AutoLog

Eine weitere Möglichkeit für Change Data Capture stellt Oracle über sein Produkt „GoldenGate“ zur Verfügung. Dieses ist ebenfalls kostenpflichtig, bietet aber auch in heterogenen Systemlandschaften die Möglichkeit, Änderungen in Realzeit von einer Quelldatenbank in eine Staging-Datenbank zu kopieren [2].

Export Data Pump (expdp) und Import Data Pump (impdp) stellen vier unterschiedliche Methoden für den Ex- und Import zur Verfügung, wobei automatisch die performanteste Methode gewählt wird. Die Methoden – sortiert nach absteigender Performance – sind:

- Direkte Kopie eines Datafiles
- Direct-Path-Operationen
- External Tables
- Import über einen Netzwerklink

Der Job des Ex- beziehungsweise Imports wird in einer Mastertabelle überwacht. Damit sind sie gewöhnlich nach einem Abbruch resümierbar. Der Import direkt über das Netzwerk entspricht dem Absetzen eines „INSERT INTO … SELECT …“-Statements. Beim Start von expdp und impdp kann über den Parameter „CONTENT“ selektiert werden, ob nur Metadaten (Tabellenstruktur), nur Daten (Tabelleninhalte) oder beide zusammen geladen werden sollen. Seit Version 11*g* der Datenbank lässt sich zudem mithilfe des Parameters „COMPRESSION“ steuern, ob Metadaten und/oder Dateninhalte komprimiert werden sollen. Dies reduziert den Speicherbedarf von Dump-Dateien deutlich.

Beim Aufruf über die Kommandozeile greifen Datapump-Export und -Import auf die Funktionalitäten der PL/SQL-Packages „DBMS_DATAPUMP“ und „DBMS_METADATA“ zurück. Es ist daher nicht überraschend, dass Export und Import bei Bedarf auch ausschließlich mithilfe von Prozeduraufrufen in „DBMS_DATAPUMP“ umgesetzt werden können.

Compression

Persistent gespeicherte Daten belegen immer physischen Speicherplatz. Der Umfang des belegten Speichers sowie der erzielte Datendurchsatz sind hingegen von vielerlei Faktoren abhängig. Für die Auseinandersetzung mit vielen dieser Faktoren ist in den meisten Fällen ein Administrator zuständig, daher wird dieses Thema in diesem Artikel nur am Rande betrachtet. Dagegen ragen Fragestellungen wie das Sizing von Blöcken und Tablespaces und insbesondere die Verwendung von Compression schon wesentlich weiter in den Zuständigkeitsbereich eines Entwicklers oder Architekten.

Compression wird zur Komprimierung von Tabellen und Indizes genutzt. Dadurch kann eine Reduzierung der Hardware-Anforderungen für Storage, Netzwerkbandbreite und RAM erzielt werden, was in einer Kostenersparnis münden und zugleich unter Umständen die Abfrage-Performance steigern kann.

Die Komprimierung erfolgt bei Tabellen auf Basis der Datenblöcke. Dabei sammelt man in einem Header die auftretenden Inhalte. In den eigentlichen Datenblöcken wird daraufhin nur noch eine Referenz auf den Eintrag im Header gespeichert. Folglich kann eine besonders hohe Komprimierungsrate erreicht werden, wenn sich in einem Block Werte häufig wiederholen. Um diesen Effekt noch zu verstärken, bietet sich die Wahl einer großen Blockgröße an.

In der Praxis lässt sich oft ein Komprimierungsfaktor zwischen zwei und drei erzielen.

Bei Compression ist zwischen zwei Varianten zu unterscheiden: Basic Compression und Advanced Compression, die weitere Möglichkeiten im Umfeld von OLTP und im Umgang mit Dateien (LOB) bieten und als eigene Option zu lizenzieren sind.

Performance-Steigerung

Theoretisch könnte ein DWH nur aus Tabellen und der Verarbeitungslogik zur Bewirtschaftung dieser Tabellen bestehen. Ohne weitere Objekte wie beispielsweise Indizes würden aber die Laufzeiten der Bewirtschaftungen und gerade auch der Abfragen durch Endanwender schnell Ausmaße annehmen, die einen akzeptablen Rahmen übersteigen. Daher hat Oracle eine ganze Palette von Features zur Steigerung der Performance in seine Datenbank integriert. Einige davon sind für OLTP- und Data-Warehouse-Systeme gleichermaßen von Bedeutung.

In einem DWH stellen sich zum Teil aber auch andere Herausforderungen, da dort meist große Datenmengen in einem einzigen Batchlauf verarbeitet werden. Dieser Batchlauf muss häufig in einem definierten Zeitfenster abgeschlossen werden.

OLTP-Systeme müssen im Gegensatz dazu in der Lage sein, viele kleinere Datenänderungen in kurzer Zeit zu verarbeiten. Die zunehmend in Mode kommenden Begriffe des Near- oder Realtime-DWH weichen diese Trennung allerdings auf, da in solchen Systemen binnen kurzer Zeit Änderungen in den Vorsystemen verarbeitet werden müssen. Diese Änderungen werden zum Beispiel mit Change Data Capture protokolliert und im DWH als Delta umgesetzt.

Partitionierung

Partitionierung ist eine kostenpflichtige Datenbankoption, die im DWH-Umfeld sehr weit verbreitet ist. Sie bezeichnet die Speicherung einer logischen Tabelle in vielen physikalischen Tabellen, sogenannten „Partitionen“. Diese können gezielt ausgelesen oder mit DML- oder DDL-Statements verarbeitet werden. Das erleichtert die

Verwaltung der Daten und sorgt bei größeren Datenmengen durch die Reduktion des I/O-Transfers für eine deutliche Steigerung der Performance.

Die Partitionierung bezieht sich auf eine konkrete Tabellenspalte. Abhängig vom darin enthaltenen Wert entscheidet sich, in welcher Partition ein Datensatz abgelegt werden muss bzw. in welcher Partition ein Datensatz im Falle einer Abfrage zu suchen ist. Die Partitionierung kann auf drei unterschiedliche Weisen erfolgen:

- Range-Partitionierung
- List-Partitionierung
- Hash-Partitionierung

Partitionen können je nach Partitionsart selbst wieder partitioniert werden. Die dadurch entstehenden Partitionen heißen Sub-Partitionen. In diesem Fall spricht man von Composite-Partitionierung. Seit Version 11*g* ist eine nahezu beliebige Kombination von Partitionierungsmethoden möglich. Lediglich die Subpartitionierung einer Hash-partitionierten Tabelle ist nicht möglich.

„Partition Exchange Loading" ist ein weiteres Feature, das erst durch die Partitionierung von Tabellen ermöglicht wird. Dabei wird eine nicht partitionierte Tabelle als Partition in eine andere, partitionierte Tabelle eingehängt. Eventuell vorhandene Indizes können im gleichen Zuge übernommen werden.

Damit lassen sich Tabelleninhalte offline berechnen und anschließend ohne Rechenlast als Partition einhängen. Bei diesem Schritt findet intern nur eine Aktualisierung des Dictionaries statt. Der sich daraus ergebende Vorteil ist die uneingeschränkte Verfügbarkeit der partitionierten Zieltabelle während der Berechnung der Daten. Andernfalls kann es zu Sperren kommen, die auf diese Weise umgangen werden können. Das folgende Beispiel verdeutlicht die zugehörige Syntax:

```
ALTER TABLE partitionierte_ta-
belle
EXCHANGE PARTITION partition
WITH TABLE tabelle
INCLUDING INDEXES
WITHOUT VALIDATION;
```

Voraussetzung für Partition Exchange Loading ist die vollständige Übereinstimmung der Spalten und Datentypen inklusive ihrer Länge sowie der Constraints beider Tabellen.

Oft müssen Daten in unterschiedlicher Granularität aggregiert werden. So könnten beispielsweise die Verkäufe pro Monat auf der einen und pro Quartal oder Jahr auf der anderen Seite von Interesse sein. Beide Ergebnismengen könnten mithilfe zweier unterschiedlicher Abfragen unter Verwendung des „UNION-ALL"-Statements vereint werden.

Die Datenbank stellt aber Funktionen zur Verfügung, die helfen, die vereinigte Ergebnismenge komfortabel mit einer einfachen Syntax innerhalb eines Statements zu berechnen. Die Anwendung dieser Funktionen ist im Bedarfsfall auch aus Performance-Gesichtspunkten zu empfehlen, zumal sich mit der Kombination der Funktion „Rollup", die im Folgenden dargestellt wird, eine Fülle von Möglichkeiten eröffnet. Weitere Features in diesem Zusammenhang sind „Cube", „Grouping Sets" und die Gruppierungsfunktionen.

Der Begriff „Rollup" ist als konträres Vorgehen zum Drilldown durch Dimensions-Hierarchien hinlänglich bekannt. Dabei werden Detail-Ergebnisse weiter aggregiert und auf übergeordneter Ebene dargestellt. Diese wird in die „GROUPBY"-Klausel integriert. Das folgende Beispiel verdeutlicht die Anwendung:

```
SELECT t.jahr, t.quartal,
t.monat, SUM(s.revenue)
FROM sales s
JOIN d_zeit t ON (s.tag_id =
t.tag_id)
GROUP BY ROLLUP(t.jahr,
t.quartal, t.monat);
```

In diesem Beispiel wäre die Spalte „Monat" identisch dem Wert „NULL", sobald die Aggregationsebene (Jahr, Quartal, Monat) überschritten wird. Die anderen Attribute verhalten sich analog. Die ROLLUP-Funktion ist zwar häufig entlang von dimensionalen Hierarchie-Pfaden zu finden, grundsätzlich aber unabhängig davon einsetzbar.

Materialized Views enthalten häufig voraggregierte Daten und greifen dabei auch auf Funktionen wie „ROLLUP" oder „GROUPING SETS" zurück. Zur Kennzeichnung des Aggregationslevels wird in diesem Fall häufig die „GROUPING_ID"-Funktion in die Abfrage der Materialized View aufgenommen. Dies erweitert die Möglichkeiten der Datenbank bei der Aktualisierung der Materialized View beziehungsweise bei deren Verwendung in Zusammenhang mit Query Rewrite.

Die Funktion „GROUP_ID" ist eines von diversen Hilfsfeatures. Mit ihr können Dubletten auf Grund einer unglücklich gewählten „GROUP-BY"-Klausel durch einen Wert größer als Null in der Ergebnismenge kenntlich gemacht werden. Diese Hilfsfunktion kann sowohl in der Ergebnismenge dargestellt werden als auch in der „HAVING"- oder „ORDER-BY"-Klausel Verwendung finden.

Analytische Funktionen

Analytische Funktionen ermöglichen die Berechnung von Ergebnissen auf Basis mehrerer Zeilen, dem sogenannten „Fenster". Die Gesamtergebnismenge reduziert sich dabei aber nicht, anders als bei Aggregat-Funktionen. So kann für jede Zeile beispielsweise ein Ranking bezüglich eines frei wählbaren Kriteriums berechnet und in einer zusätzlichen Spalte dargestellt werden. Die „PARTITION-BY"-Klausel gibt an, für welche Spalten die jeweils gleiche Kombination von Werten als eine Gruppe von Zeilen interpretiert wird. Diese Angabe ist vergleichbar mit der „GROUP-BY"-Klausel bei einer gewöhnlichen Aggregation und nicht zu verwechseln mit Partitionen in Zusammenhang mit der Partitionierung von Tabellen.

Die „ORDER-BY"-Klausel definiert das Attribut, nach dem sich die Berechnung des Ranges richtet. In obigem Beispiel würde folglich für jede Abteilung der Rang der Mitarbeiter in Bezug auf ihr Gehalt bestimmt. Der Mitarbeiter mit dem größten Gehalt erhält Rang 1.

Für andere analytische Funktionen kann explizit das Fenster angegeben

Abbildung 2: Filtern der Tabellen-Inhalte

werden, auf dessen Basis die Ergebnisse berechnet werden sollen. Dieses Fenster wird für jede Gruppe – bestimmt durch die „PARTITION-BY"-Klausel – von Neuem initialisiert. Das angegebene Fenster orientiert sich immer an der aktuellen Zeile. Seine Größe kann über die physikalische Anzahl von Zeilen oder ein logisches Intervall, beispielsweise über die Inhalte einer Spalte vom Datentyp „DATE", definiert werden. Die „ORDER-BY"-Klausel schreibt die Reihenfolge innerhalb eines Fensters vor. Mithilfe von Fensterfunktionen lässt sich beispielsweise eine rollende Summe berechnen.

Andere Beispiele stellen die Fensterfunktionen „LAG" und „LEAD" dar. Diese gewähren Einsicht in Zeilen, die auf Basis einer gewählten Sortierreihenfolge eine definierte Anzahl von Zeilen vor (LAG) beziehungsweise nach (LEAD) der aktuellen Zeile liegen. Auf diese Weise ist ein direkter Vergleich unterschiedlicher Zeilen möglich, ohne zuvor einen Self-Join implementieren zu müssen. Dies spart nicht nur Implementierungsaufwand, sondern häufig auch Rechenlast.

Die Datenbank bietet darüber hinaus eine ganze Palette von analytischen Funktionen. Hinzu kommt, dass mit dem Oracle Data Cartridge Interface (ODCI) eigene Aggregat-Funktionen entwickelt werden können, die sich auch als analytische Funktion nutzen lassen.

Schutz von Daten

Datensicherheit ist im Zeitalter der elektronischen Datenverarbeitung in vielen Bereichen wichtig. Im DWH-Umfeld spielt sie eine besonders große Rolle, weil viele aussagekräftige und sensible Daten von einer Vielzahl von Anwendern mit unterschiedlichen Rechten und unterschiedlichen Zielen interpretiert werden sollen. Neben der normalen Vergabe von Rechten auf Objektebene stellt Oracle weitere Funktionen zur Verfügung, die je nach angemeldetem Benutzer eine unterschiedliche Sicht auf die Daten ermöglichen.

Virtual Private Database ermöglicht eine sehr feingranulare Vergabe von Rechten, sodass unterschiedliche Benutzer jeweils eine andere Sicht auf das gleiche Tabellenobjekt erhalten können. Es können je nach Benutzer Filterkriterien, sogenannte „Policies", definiert werden, die dazu führen, dass der eine Benutzer andere Zeilen und Spalten sieht als der nächste. Diese Policies können ferner auf DML-Statements ausgeweitet werden, damit die Änderung von Datensätzen nur innerhalb eines definierten Datenbereichs erfolgen kann. Mittels einer Policy beziehungsweise der Kombination mehrerer Policies lassen sich nahezu beliebig komplexe Zugriffsbeschränkungen anhand unterschiedlichster Geschäftsregeln definieren.

In einem späteren Schritt werden Abfragen auf die jeweilige Tabelle durch die Datenbank intern und transparent umformuliert und um die Zeichenkette erweitert, die die Policy-Funktion zurückliefert. So werden die Tabelleninhalte gefiltert. Grafisch dargestellt könnte dies wie in Abbildung 2 aussehen.

Database Vault stellt eine Technologie dar, mit der es möglich wird, Daten vor internen Bedrohungen zu schützen. Diese internen Bedrohungen sind real und durch diverse repräsentative Studien belegt. Die Absicherung gegen diese Risiken ist daher gerechtfertigt und gründet im Allgemeinen nicht auf mangelndem Vertrauen in die eigenen Mitarbeiter.

Mit Database Vault lassen sich innerhalb einer Web-Schnittstelle oder mithilfe einer API-Schutzzone, sogenannte „Realms", definieren, die privilegierten Benutzern oder Administratoren den Zugriff auf sensible Daten verwehren. Zusätzlich können Kriterien festgesetzt werden, die für die Benutzerauthentifizierung erforderlich sind und die die gewöhnliche Authentifizierung erweitern. Dies minimiert die Gefahr, dass sich Benutzer unter Vorgabe einer falschen Identität unbefugten Zugriff auf Daten verschaffen. Abbildung 3 zeigt beispielhaft und

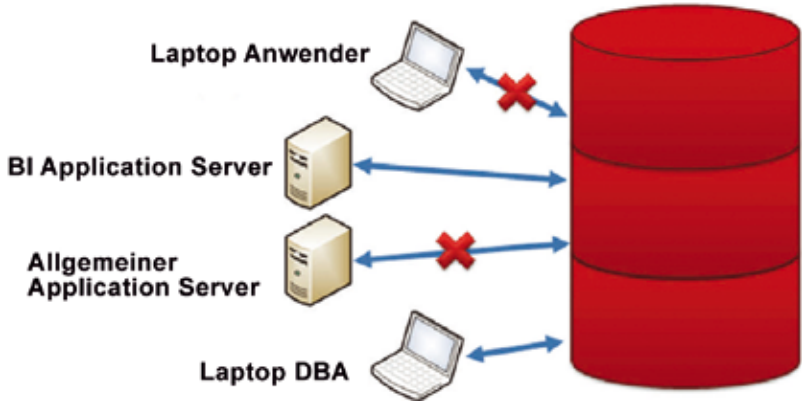

Abbildung 3: Mögliche Konfiguration für Database Vault

sehr vereinfacht eine mögliche Konfiguration.

Mittels der Definition von Command Rules können außerdem Regeln bestimmt werden, die den erlaubten Satz von Datenbank-Befehlen eines Benutzers beschränken. So kann einem Benutzer beispielsweise das Löschen von Tabellen in seinem eigenen Schema untersagt werden.

Database Vault erlaubt dabei die Verwendung vordefinierter Standardrollen, die sogenannte „Separation of Duty". Beispiel einer solchen Rolle ist die Benutzer-Administration. Einem Datenbank-Administrator ohne diese Rolle kann damit das Recht entzogen werden, weitere Benutzer anzulegen. Neben der reinen Zugriffskontrolle bietet Database Vault ein zusätzliches Auditing, bei dem versuchte Zugriffsverletzungen in vordefinierten Standardberichten dargestellt werden können. Database Vault ist verfügbar für die Datenbank-Releases 9*i* R2, 10*g* R2 und 11*g*.

Label Security ist eine kostenpflichtige Datenbank-Option. Diese ermöglicht – ähnlich wie VPD – die Beschränkung des Zugriffs von Benutzern auf ausgewählte Datensätze einer Tabelle. Dank des Enterprise Managers ist es jedoch nicht erforderlich, eigene Policy-Funktionen zu entwickeln. Stattdessen können in der Programmoberfläche auf Datenebene sogenannte „Labels" vergeben werden, die sich in „Level", „Compartments" und „Groups" untergliedern. Dadurch ist eine feingranulare Abbildung der Sensibilität der Daten möglich. Jedes Level, jedes Compartment und jede Group wird vom Administrator mit einer numerischen Repräsentation versehen. Je größer der Wert, desto sensibler sind die Daten. Beim Zugriff auf die Daten geht die Datenbank daraufhin nach einem Schema vor (siehe Abbildung 4).

„Data Masking" bezeichnet die Maskierung von sensiblen Daten zur Erstellung reell wirkender Testdaten. Zu diesem Zweck bietet die Oracle-Datenbank ein Management-Pack, das über den Enterprise Manager genutzt werden kann. Es bietet eine Such-Funktionalität, mit der man sensible Daten ausfindig machen kann. Wurden

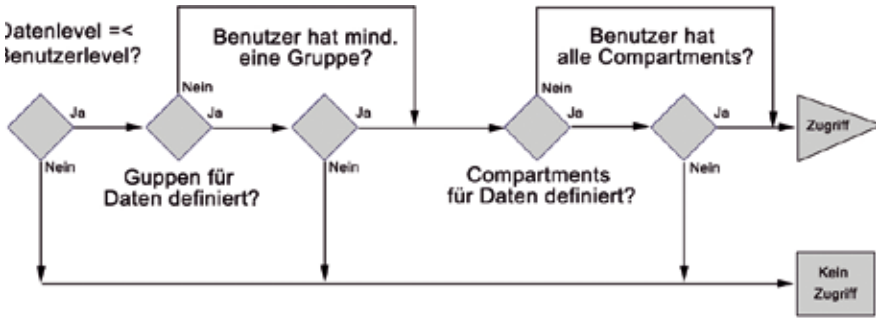

Abbildung 4: Zugriff der Datenbank auf die Daten

Abbildung 5: Ver- und Entschlüsselung der Daten

solche Daten in der Datenbank identifiziert, können sie mithilfe vordefinierter oder eigener Maskierungsfunktionen maskiert werden.

Die vordefinierten Funktionen bieten neben der Generierung von zufälligen Werten auch Formatmasken zur Generierung von Telefonnummern oder Kreditkartennummern. Darüber hinaus ermöglicht Data Masking das deterministische Vertauschen von Werten. Auf diese Weise erhalten vormals gleiche Werte auch nach der Maskierung einen identischen Wert. Weitere Möglichkeiten sind dadurch gegeben, dass abhängig von Bedingungen unterschiedliche Maskierungsfunktionen angewandt werden können und dass durch die Definition zusammengehöriger Spalten die in den Daten vorliegenden Zusammenhänge aufrechterhalten werden können.

Die Maskierung der Daten basiert auf PL/SQL-Routinen, die bei Bedarf durch einen Administrator angepasst werden können und sich in andere Prozesse einbinden lassen. So kann die Anwendung von Data Masking im Zuge des Duplizierens von Tabellen oder während des Datenexports mit Export Data Pump erfolgen.

„Transparent Data Encryption" ist Teil der Advanced-Security-Option für die Enterprise Edition der Datenbank. Mit diesem Feature lassen sich Daten nach einer umfangreichen Auswahl von Algorithmen verschlüsseln (siehe Abbildung 5). Die Verschlüsselung ist transparent für die Applikationen, die die Daten weiterverarbeiten. Folglich ist eine Anpassung von Applikationslogik nicht erforderlich. Datenbankintern werden die Daten vor dem Schreiben auf die Festplatte automatisch verschlüsselt und beim Lesen wieder entschlüsselt, noch bevor sie an die Applikation übergeben werden. Wird eine Sicherung durchgeführt, bleiben die Daten hingegen verschlüsselt. Die Verwaltung der Schlüssel

zum Ver- und Entschlüsseln geschieht ebenfalls datenbankintern in einem sogenannten „Wallet“, das seinerseits auch verschlüsselt ist.

Für dieses Wallet muss initial ein Master Key gesetzt werden, der systemweite Gültigkeit besitzt. Das Wallet muss zur Verarbeitung der Daten geöffnet sein, was nach einem Neustart der Instanz nicht gewährleistet ist. Das Wallet kann durch den folgenden Befehl geöffnet werden, wobei „myPassword“ mit dem sogenannten „Master Key“ substituiert werden muss:

```
ALTER SYSTEM SET WALLET OPEN
IDENTIFIED BY
„myPassword“;
```

Wenn das Öffnen des Wallets vergessen wurde, schließt man das Wallet explizit mit dem Befehl:

```
ALTER SYSTEM SET WALLET CLOSE;
ORA-28365:
WALLET IS NOT OPEN
```

Die Verschlüsselung von Datenbank-Inhalten ist auf verschiedene Art und Weise möglich:

- Es können einzelne Spalten verschlüsselt werden. Ist die ausgewählte Spalte indiziert, werden das Löschen des Index und die anschließende Neuanlage desselben auf Basis der nunmehr verschlüsselten Spalte empfohlen.
- Seit Datenbankversion 11*g* können ganze Tabellen und Secure Files/LOBS verschlüsselt werden.
- Ebenfalls seit version 11*g* ist die Verschlüsselung ganzer Tablespaces möglich.

Neben der verschlüsselten Ablage von Daten im Dateisystem kann auch die Kommunikation mit der Datenbank verschlüsselt werden, um diese gegen das Auslesen von Inhalten über den Netzwerkverkehr abzusichern. In diesem Fall ist die Anpassung der Datei „sqlnet.ora“ auf Server und Clients vonnöten. Unterstützung für die Verschlüsselung des Netzwerkverkehrs bietet zum Beispiel der Oracle Net Manager.

Fazit

Die Oracle-Datenbank liefert eine zunehmende Fülle von Features, über die man schnell den Überblick verlieren kann. Viele Entwickler kennen diese Features gar nicht oder sie kommen nicht vollumfänglich zur Anwendung und ihre Vorteile bleiben ungenutzt. So entstehen oftmals eigene Lösungen, die schwer zu warten sind, bei der Entwicklung unnötige Personal-Ressourcen vereinnahmen und außerdem häufig unter schlechter Performance oder großer Fehleranfälligkeit leiden. Auf der anderen Seite geben Unternehmen viel Geld für Datenbank-Optionen aus, die letzten Endes gar nicht genutzt werden.

Für Data-Warehouse-Entwickler ist ein umfangreiches Wissen über die vorhandenen Datenbank-Features und die damit verbundenen Kosten und Einsparpotenziale also essentiell wichtig. Dazu gibt es viele hilfreiche Tipps und Bewertungen [3].

Quellenangaben

[1] Wikipedia, Definition eines Data-Warehouse: http://de.wikipedia.org/wiki/Data-Warehouse#Definition

[2] Oracle GoldenGate: http://www.oracle.com/us/products/middleware/dataintegration/059240.html

[3] Whitepaper des Autors, Features der Oracle-Datenbank aus der Data-Warehouse-Perspektive: http://www.opitz-consulting.com/veroeffentlichungen/whitepaper.php.

Timo Bergenthal
OPITZ CONSULTING GmbH
timo.bergenthal
@opitz-consulting.com