Indexierungsstrategie im Data Warehouse
Zwischen Albtraum und optimaler Performance

Dani Schnider
Trivadis AG
Zürich/Glattbrugg, Schweiz

Schlüsselworte:

Data Warehouse, Indexierung, Staging Area, Cleansing Area, Core, Data Mart, Star Schema, B-Tree Index, Bitmap Index, ETL, Partitionierung, Star Transformation, Bitmap Join Index

Einleitung

In Data Warehouses gelten andere Regeln als in klassischen OLTP-Systemen. Das gilt auch für die Indexierung der Datenbank. Wie indexiere ich meine Staging Area, mein Core Data Warehouse und meine Data Marts? Viele Data Warehouses haben falsche, zu viele oder gar keine Indizes, weil es leider oft an den Grundlagen und an der Zeit fehlt, sich Gedanken über eine geeignete Indexierungsstrategie zu machen. Ein ungeeigneter Index richtet mehr Schaden an, als dass er die Performance der Abfragen verbessert.

Widersprüchliche Indexierungsansätze

Es gibt viele verschiedene Meinungen, Tipps und Tricks, wie ein Data Warehouse zu indexieren sei. Teilweise sind die Aussagen zur Indexierungsstrategie widersprüchlich, wie folgende Beispiele zeigen: *«In einem Data Warehouse werden mehr Indizes pro Tabelle erstellt als in einem OLTP-System» – «Wenn die Performance schlecht ist, müssen zusätzliche Indizes kreiert werden» – «Full Table Scans sind langsam» – «Ein Data Warehouse sollte möglichst wenige Indizes haben» – «Jede Tabelle muss einen Primary Key haben» – ...*

Diese Aufzählung von Zitaten – die übrigens alle aus real existierenden DWH-Projekten stammen – ließe sich beliebig fortsetzen. Dass solche Falschaussagen oder ungenauen Regeln oft zu hören sind, liegt daran, dass viele DWH-Entwickler und Datenbankadministratoren versuchen, ihr Wissen über Performance Tuning aus OLTP-Systemen in Data Warehouses anzuwenden. Dabei übersehen sie, dass dort – wie bereits erwähnt – andere Gesetzmäßigkeiten gelten. Lassen wir für unsere nachfolgenden Erläuterungen einen (fiktiven) Kunden sprechen, der offenbar Performanceprobleme mit seinem Data Warehouse hat: *«Wir haben in unserem Data Warehouse viel zu lange Antwortzeiten. Obwohl wir schon viele Indizes erstellt haben, werden die Abfragen trotzdem nicht schneller. Viele der Indizes werden von Oracle gar nicht verwendet, sondern erst, wenn ich Optimizer-Hints in den Abfragen einfüge. Und dann dauert es sogar noch länger, genau wie bei den Ladeprozessen, die auch immer langsamer werden. Es ist der reinste Albtraum! Was soll ich tun?»*

Wie können wir diesem Kunden helfen? Welche Indexierungsstrategie ist in einem Data Warehouse sinnvoll, um Performanceprobleme bei den Abfragen und bei den Ladeprozessen zu vermeiden? Eine einfache und allgemeingültige Patentlösung, die für jedes Data Warehouse gilt, gibt es leider nicht. Aber es gibt ein paar wichtige Grundsätze, die zu beachten sind, und die sind je nach Teilbereich einer DWH-Architektur unterschiedlich.

Unterschiedliche Indexierung pro Teilbereich

Ein Data Warehouse besteht aus verschiedenen Komponenten oder Teilbereichen, die alle einen unterschiedlichen Zweck haben. Im Buch «Data Warehousing mit Oracle – Business Intelligence in der Praxis»[1] wird eine DWH-Architektur mit Staging Area, Cleansing Area, Core und Data Marts beschrieben:

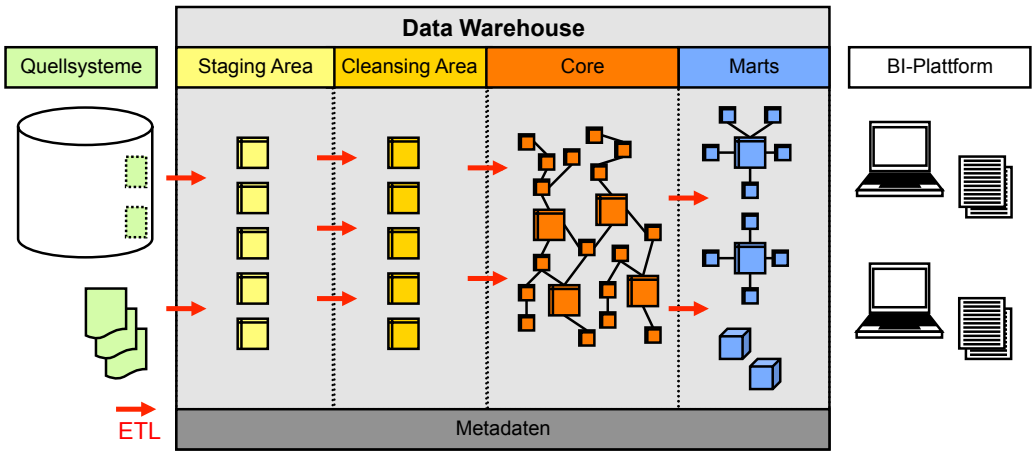

Abb. 1: Komponenten eines Data Warehouses

Wie sind nun die verschiedenen Teilbereiche in dieser typischen DWH-Architektur zu indexieren? Wo sind welche Arten von Indizes sinnvoll, und wo sind sie ganz zu vermeiden? Entscheidend ist, dass für jede Komponente eine unterschiedliche Indexierungsstrategie gilt, je nachdem, was sie innerhalb des DWH-Gesamtsystems für eine Aufgabe zu erfüllen hat. Deshalb schauen wir uns die einzelnen Teilbereiche im Detail an und versuchen, unserem fiktiven Kunden zu helfen, der sich zwischendurch mit Fragen melden wird.

Staging Area

Die Staging Area dient dazu, Daten aus den Quellsystemen möglichst effizient in die DWH-Datenbank zu laden. Transformationen finden hier typischerweise noch keine statt. Die Struktur der Stage-Tabellen ist meistens identisch zur Tabellenstruktur der Quellsysteme oder der Schnittstellen-Views.

«Was für Indizes sind nun auf den Stage-Tabellen zu erstellen?» Die Antwort ist einfach: gar keine! – *«Aber werden dann die ETL-Zugriffe auf die Staging Area nicht langsamer?»* – Im Gegenteil. Ein Index ist nur für selektive Abfragen zweckmäßig, also für SQL-Queries, die nur einen kleinen Teil der Datensätze einer Tabelle lesen. Die Staging Area enthält nur jeweils die Daten einer Lieferung. Für die weitere Verarbeitung durch die ETL-Prozesse werden alle diese Daten verwendet. Selektive Abfragen werden in der Staging Area nie ausgeführt. Jeder Index auf einer Stage-Tabelle würde somit nur den Ladevorgang verlangsamen, hätte aber keinen Nutzen für die Weiterverarbeitung der Daten aus der Staging Area.

---

[1] Claus Jordan, Dani Schnider, Joachim Wehner, Peter Welker: Data Warehousing mit Oracle – Business Intelligence in der Praxis, Hanser Verlag, 2011, ISBN 978-3-446-42562-0

Cleansing Area

In der Cleansing Area werden Datenbereinigungen, Integritätsprüfungen, Transformationen und Datenanreicherungen ausgeführt. Auch die Cleansing Area enthält nur die Daten der letzten Lieferung. *«Dann gelten hier also bezüglich Indexierung die gleichen Regeln wie in der Staging Area?»* Ja, allerdings mit ein paar Ausnahmen: Falls bereits in der Cleansing Area künstliche Schlüssel (Surrogate Keys) für das DWH erstellt werden, werden auf den Cleansing-Tabellen typischerweise Primärschlüssel definiert. Bei der Erstellung eines Primary Key Constraints kreiert Oracle automatisch einen Unique Index auf die betreffenden Attribute. Oft besteht zusätzlich das Bedürfnis, die fachlichen Schlüssel oder die technischen Identifikationsschlüssel aus dem Quellsystem auf Eindeutigkeit zu prüfen. Dies kann mit Unique Key Constraints implementiert werden. Auch dafür erstellt Oracle automatisch einen Unique Index.

«Wie sieht es aus, wenn in der Cleansing Area auch Foreign Key Constraints definiert werden? Ich habe mal gelesen, dass Fremdschlüssel immer indexiert werden sollten». Das ist empfehlenswert für OLTP-Systeme, um selektive Master-Detail-Abfragen effizient durchführen zu können und um Lockingprobleme bei DELETE-Operationen auf der Mastertabelle zu vermeiden. In der Cleansing Area eines Data Warehouses werden aber weder selektive Abfragen noch Löschoperationen ausgeführt. Deshalb ist eine Indexierung der Fremdschlüssel überflüssig.

Core

Das Core dient zur Integration der Daten aus unterschiedlichen Quellsystemen sowie zur Historisierung aller Daten. Es enthält oft große Datenmengen, weil hier Informationen auf Detailebene über längere Zeit gespeichert werden. Das Core ist die zentrale Datenbasis für alle Data Marts. Benutzer greifen aber nicht direkt auf die Tabellen im Core zu, sondern nur auf die Data Marts, in welchen die Daten für die entsprechende Anwendergruppe bereitgestellt werden. *«Das heißt, dass im Core gar keine Indizes erstellt werden müssen?»* Es gibt tatsächlich Data Warehouses, in denen auf den Core-Tabellen kein einziger Index vorhanden ist. Da jedoch der Aufbau eines Core je nach Projektumfeld, Architektur sowie technischen und fachlichen Rahmenbedingungen sehr unterschiedlich sein kann, gibt es verschiedene Indexierungsstrategien für diese DWH-Komponente. Ein Kriterium ist zum Beispiel die Wahl des Datenmodells. Ob für das Core ein relationales Datenmodell in 3. Normalform oder ein dimensionales Datenmodell mit Fakten und Dimensionen verwendet wird, hat Auswirkungen auf die Indexierung. Ebenso muss zwischen Stammdaten und Bewegungsdaten unterschieden werden.

«Wo liegen die Unterschiede bei der Indexierung von Stammdaten und Bewegungsdaten?» Stammdatentabellen (bzw. Dimensionstabellen in einem Core mit dimensionalem Datenmodell) besitzen in der Regel einen Primary Key, bestehend aus einem künstlichen Schlüssel, der im DWH generiert wird. Zusätzlich existiert ein Unique Key auf einem fachlichen Schlüssel oder dem Identifikationsschlüssel des Quellsystems. Für jeden Primary Key und Unique Key Constraint wird implizit ein Unique Index angelegt. Diese Indizes dienen vor allem der Integritätsprüfung sowie eventuell der Optimierung von Lookups in den ETL-Prozessen. Dies kann bei der Versionierung von Daten (z.B. mittels Slowly Changing Dimensions Typ 2) eventuell nützlich sein, ist aber in vielen Fällen für die Laufzeit der ETL-Prozesse nicht relevant.

Bei Bewegungsdaten (bzw. Fakten im dimensionalen Modell) gilt der Grundsatz: Je weniger Indizes, desto besser. Weil hier oft große Mengen von Datensätzen ins Core eingefügt werden und auf der Bewegungstabelle keine Lookups notwendig sind, würde jeder zusätzliche Index nur die Performance der ETL-Prozesse beeinträchtigen. *«Aber um nachher die Daten in die Data Marts zu laden, ist doch*

ein Index auf das Ereignisdatum notwendig?» Nicht unbedingt. Wird ein Data Mart vollständig geladen (Initial Load), so müssen typischerweise alle Daten aus den Core gelesen, eventuell aggregiert und in den Data Mart geladen werden. Ein solcher ETL-Prozess kann durch Indizes nicht beschleunigt werden. Die schnellste Methode ist hier, die Core-Tabelle mittels Full Table Scan zu lesen.

Anders sieht die Situation aus, wenn nur die Änderungen und neuen Daten in den Data Mart übertragen werden (Incremental Load). Dann wäre ein geeigneter Index eine Möglichkeit, um die Ladeprozesse zu beschleunigen. Allerdings auch dann nur, wenn pro Ladelauf nur ein kleiner Prozentsatz (< 1-2%) der Daten aus dem Core in den Data Mart geladen werden. Wird zum Beispiel ein Data Mart täglich inkrementell geladen aus einem Core mit mehreren Jahren historischen Daten, kann ein Index auf das entsprechende Datumattribut für den ETL-Prozess zweckmäßig sein. Bei einem monatlichen Load ist die entsprechende Abfrage auf das Core weniger selektiv, sodass ein Indexzugriff nicht mehr effizient ist. *«Gibt es in diesem Fall bessere Alternativen, um den ETL-Prozess zu beschleunigen?»* Ideal ist, wenn die Core-Tabellen mit den Bewegungsdaten nach dem Ereignisdatum partitioniert sind. Wenn für einen Incremental Load jeweils eine Monats- oder sogar Tagespartition der Core-Tabelle gelesen werden muss, ist dies der effizienteste Weg, um einen Data Mart inkrementell zu laden. Dann sind keine Indizes auf den Bewegungs- oder Fakttabellen notwendig.

«Apropos Partitionierung: Was ist bei der Indexierung von partitionierten Tabellen zu beachten?» Ein wichtiger Grund für den Einsatz von Partitionierung in Data Warehouses ist die Möglichkeit, eine «rollende Historisierung» zu implementieren. Vor dem Laden von neuen Daten werden jeweils neue Partitionen angelegt, und die ältesten Partitionen werden gelöscht. Auf solchen nach Datum partitionierten Tabellen sollten ausschließlich lokale Indizes – also Indizes, die gleich partitioniert sind wie die Tabelle – erstellt werden. Globale Indizes sind für Data Warehouses nur in Ausnahmefällen geeignet.

Zusammenfassend kann festgehalten werden, dass die Core-Tabellen möglichst sparsam indexiert werden sollten, da es – außer bei Incremental Loads von Data Marts, bei denen nur wenige Daten pro Ladevorgang aus dem Core selektiert werden – meistens nicht effizient ist, in den ETL-Prozessen über Indizes auf das Core zuzugreifen.

«Unsere Anwender führen zum Teil Adhoc-Queries direkt auf den Core-Tabellen aus. Ist es da nicht problematisch, wenn keine Indizes zur Verfügung stehen?» Ein wichtiger Grundsatz bei der Architektur eines Data Warehouses ist die Unterscheidung zwischen Core und Data Marts. Das Core ist nicht für Benutzerabfragen optimiert, sondern dient ausschließlich als Datenbasis für die Data Marts. Greifen die Anwender direkt auf das Core zu, haben wir einen Zielkonflikt zwischen guter Abfrageperformance und effizienten ETL-Prozessen. Bei den Stammdaten ist dies meistens kein Problem, da die Datenmengen pro ETL-Lauf meistens klein sind. Bei Bewegungstabellen bzw. Fakttabellen können aber viele zusätzliche Indizes dazu führen, dass die Laufzeiten der Ladeprozesse deutlich länger werden.

Dies kann vermieden werden, indem die ETL-Prozesse mengenbasiert ausgeführt werden. Beim Einfügen der Daten mit einem einzigen SQL-Befehl werden die Indizes am Ende des Statements nachgeführt. Bei zeilenweiser Verarbeitung (z.B. mit einem Cursor-Loop in PL/SQL) werden die Indizes für jeden Datensatz aktualisiert.

«Würde es etwas helfen, die Indizes vor dem Laden zu löschen und nachher neu zu erstellen?» Das nachträgliche Erstellen der Indizes ist tatsächlich eine gute Möglichkeit, um die Performance der ETL-Prozesse zu beschleunigen, sogar bei mengenbasierter Verarbeitung. Die Indizes müssen dabei nicht

gelöscht, sondern vor dem Ladevorgang auf UNUSABLE gesetzt werden. Nachdem die Daten vollständig in die Zieltabelle geladen sind, wird ein Index-Rebuild auf allen Indizes durchgeführt. Bei einem Initial Load eines Data Marts ist dies eine elegante und effiziente Lösung. Beim inkrementellen Laden des Core oder eines Data Marts muss aber in Betracht gezogen werden, dass ein Index-Rebuild immer für alle Daten gemacht werden muss, nicht nur für die neu hinzugeladenen. Außerdem besteht bei diesem Ansatz das Problem, dass während des Ladevorgangs die Indizes nicht zur Verfügung stehen, falls in dieser Zeit Adhoc-Abfragen gemacht werden. Zumindest für partitionierte Zieltabellen gibt es einen Ausweg aus diesem Dilemma. Hier kann das Prinzip des «Partition Exchange» verwendet werden.

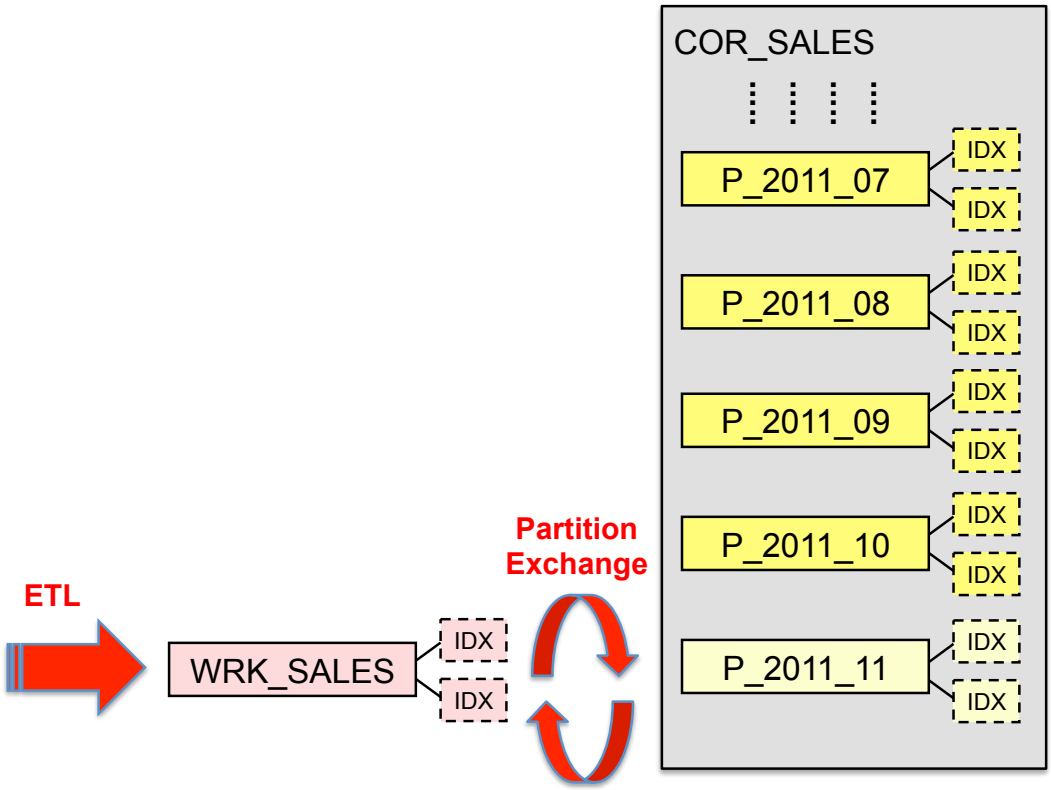

Abb. 2: Partition Exchange bietet die Möglichkeit, die Indizes nach dem Laden der Daten zu erstellen

Die Daten werden statt in die Core-Tabelle in eine Hilfstabelle mit der gleichen Struktur geladen. Nach dem Ladevorgang werden auf der Hilfstabelle die notwendigen Indizes erstellt. Während dieser Zeit können die Anwender weiterhin Abfragen auf die partitionierte Core-Tabelle durchführen. Schliesslich wird für die Core-Tabelle eine neue (leere) Partition erstellt und folgendes Statement ausgeführt:

```
ALTER TABLE cor_sales
EXCHANGE PARTITION p_2011_11 WITH TABLE wrk_sales
INCLUDING INDEXES WITHOUT VALIDATION;
```

In diesem Schritt wird die Hilfstabelle mit der leeren Partition der Zieltabelle ausgetauscht. Die Indizes auf der Hilfstabelle werden mit den entsprechenden Index-Partitionen der leeren Partition vertauscht. Da es sich dabei nur um wenige Änderungen im Data Dictionary handelt, geht dieser Vorgang sehr schnell. Nun stehen die zuvor geladenen Daten in der neuen Partition zur Verfügung. Die Hilfstabelle ist leer und kann gelöscht oder für den nächsten Ladelauf verwendet werden.

Voraussetzung für dieses Verfahren ist, dass die partitionierte Tabelle ausschließlich lokale Indizes enthält und dass pro Ladelauf jeweils eine ganze Partition (z.B. ein Monat oder ein Tag) geladen wird. Das gleiche Prinzip kann übrigens auch zum Laden der Fakttabellen in den Data Marts verwendet werden.

Data Marts

In den Data Marts werden die Daten so aufbereitet und zur Verfügung gestellt, dass die Anwender möglichst einfach und effizient darauf zugreifen können. Die Indexierungsstrategie ist somit auf die Benutzerabfragen ausgerichtet. Data Marts besitzen in der Regel ein dimensionales Datenmodell und werden in Oracle als Star Schema[2] implementiert. *«Richtig, unsere Data Marts haben wir als Star Schemas mit Dimensions- und Fakttabellen gebaut. Aber wie sollen wir diese Tabellen nun indexieren?»*

Jede Dimensionstabelle besitzt einen Primary Key, welcher von den Fakttabellen referenziert wird. Somit erstellt Oracle auch hier automatisch einen Unique Index auf das Primärschlüsselattribut. Viele Dimensionstabellen sind klein und enthalten nur wenige Dutzend oder ein paar Hundert Datensätze. Auch bei selektiven Abfragen bringen hier zusätzliche Indizes nur einen kleinen oder gar keinen Nutzen, da bei einem Full Table Scan auf die Dimensionstabelle nur wenige Blöcke gelesen werden müssen.

«Wir haben aber einzelne Dimensionen mit sehr vielen Daten. Unsere Kundendimension hat zum Beispiel drei Millionen Einträge. Wären da nicht weitere Indizes sinnvoll?» Doch, bei großen Dimensionstabellen mit Tausenden oder Millionen von Datensätzen kann es zweckmäßig sein, zusätzliche Indizes auf Attribute anzulegen, welche häufig als Filterkriterien verwendet werden. Um möglichst flexible Abfragen mit unterschiedlichen Kombinationen von WHERE-Bedingungen zu erlauben, ist es empfehlenswert, Bitmap Indizes zu erstellen. *«Aber ein Bitmap Index kann doch nur für Attribute mit wenigen verschiedenen Werten verwendet werden!»* Bitmap Indizes wurden zwar ursprünglich für die Indexierung von nicht-selektiven Attributen eingeführt, können aber problemlos auch für selektive Attribute mit vielen unterschiedlichen Werten verwendet werden. Der große Vorteil von Bitmap Indizes besteht darin, dass in einer SQL-Abfrage mehrere Indizes auf der gleichen Tabelle kombiniert werden können. Enthält eine Tabelle B-Tree Indizes, so entscheidet sich der Optimizer für den selektivsten Index und schränkt die weiteren Attribute erst beim Zugriff auf die Tabelle ein[3]. Es wird also pro Abfrage und Tabelle nur jeweils ein B-Tree Index verwendet. Bei Bitmap Indizes ist dies nicht der Fall.

«Ich habe auf das Attribut „Geschlecht" unserer Kundendimension einen Bitmap Index erstellt, aber der wird nie verwendet, wenn ich eine Abfrage auf alle männlichen oder alle weiblichen Kunden mache». Ein einzelner Bitmap Index auf ein nicht-selektives Attribut nützt meistens nichts, denn ein Zugriff über diesen Index wäre nicht effizient. Im konkreten Fall würden 50% der Daten, bei Ihrer Kundendimension also ca. 1.5 Millionen Einträge, über den Bitmap Index selektiert. Erst durch die Kombination mit Einschränkungen auf andere Attribute wird die Abfrage selektiv und liefert einen kleinen Prozentsatz von Daten zurück. Wenn diese Attribute ebenfalls indexiert sind, können die Bitmap Indizes kombiniert werden.

---

[2] Die nachfolgenden Ausführungen gelten sinngemäß auch für Snowflake Schemas.

[3] Eine Ausnahme sind die sogenannten „B-tree bitmap plans", bei denen für zwei oder mehr B-Tree Indizes eine Bitmap Conversion durchgeführt wird. Die so ermittelten Bitmaps werden danach miteinander kombiniert. Dies kann mit dem Hint `index_combine` erzwungen werden.

«Jetzt haben wir uns ausführlich über die Indexierung von Dimensionstabellen unterhalten. Wie sieht es nun bei den Fakttabellen meiner Data Marts aus?» Auf den Fakttabellen werden ebenfalls Bitmap Indizes angelegt, und zwar auf sämtlichen Dimensionsattributen, also den Fremdschlüsseln auf die Dimensionstabellen. Diese Indizes werden für eine der typischen Zugriffsarten auf ein Star Schema, die sogenannte «Star Transformation», verwendet.

«Wie funktioniert die Star Transformation?» Bei der Star Transformation werden zuerst die Filterkriterien auf jeder Dimensionstabelle angewendet, um die Datenmenge so früh wie möglich einzuschränken. Mit den daraus resultierenden Dimensionsschlüsseln wird auf den zugehörigen Bitmap Index der Fakttabelle zugegriffen. Dies wird für alle in der Abfrage beteiligten Dimensionen gemacht. Die Bitmap Indizes werden miteinander kombiniert, und erst ganz am Schluss wird auf die Fakttabelle zugegriffen.

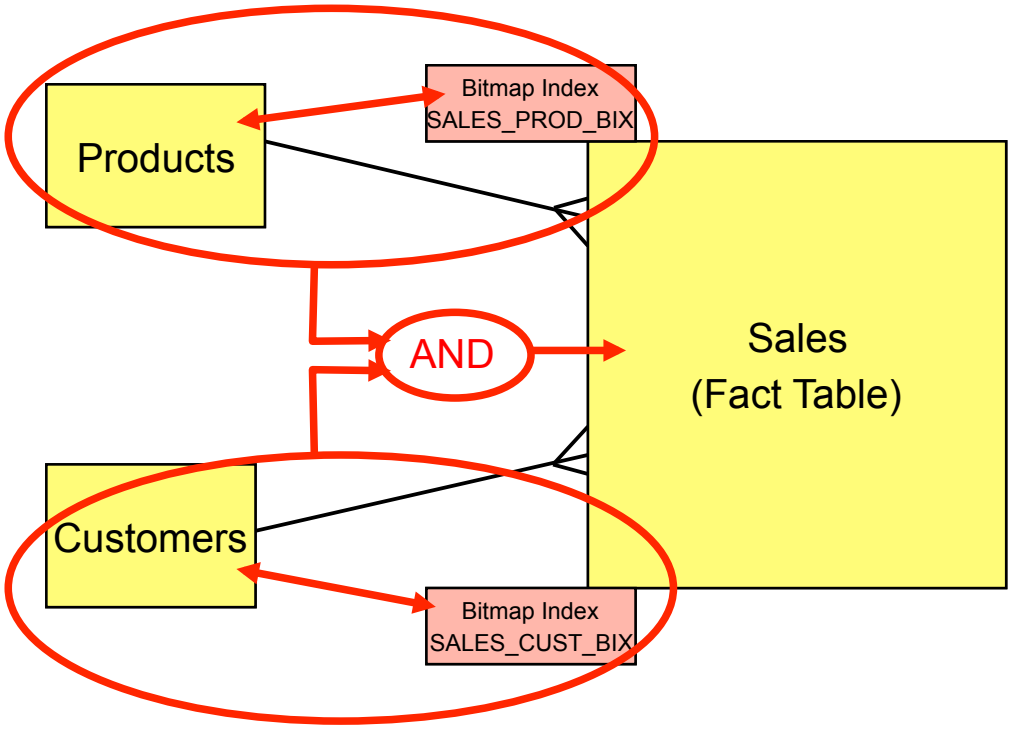

Abb. 3: Bei der Star Transformation werden zuerst alle Dimensionen ausgewertet

Dank diesem Prinzip kann ein wesentlicher Knackpunkt bei Abfragen auf ein Star Schema gelöst werden: Um die Datenmenge möglichst früh einzuschränken, ist es zweckmäßig, zuerst die Tabellen mit Filterkriterien – also die Dimensionstabellen – zu lesen. Aufgrund der Joinbedingungen müsste aber als zweites auf die Fakttabelle zugegriffen werden, und zwar unabhängig davon, welche Dimension zuerst gelesen wird. Da die Fakttabelle in der Regel die mit Abstand größte Tabelle in einem Data Mart ist, ist es effizienter, zuerst alle Filter einzuschränken und erst am Schluss auf die größte Tabelle zuzugreifen. Dies ist nur mit der Star Transformation möglich.

«Der Primärschlüssel auf unserer größten Fakttabelle bereitet uns ziemliche Probleme. Er umfasst die Kombination von allen Dimensionsschlüsseln. Weil wir ziemlich viele Dimensionen haben, ist der Primary Key Index grösser als die Fakttabelle». Dieser Index ist überflüssig, da er sicher nie für eine Abfrage verwendet wird. Sein einziger Zweck ist es, mehrfach vorhandene Fakten mit den gleichen Dimensionsreferenzen zu vermeiden. Falls die Eindeutigkeit der Schlüsselkombinationen geprüft werden soll – was übrigens nicht in allen Fakttabellen gewünscht oder möglich ist – ist es einfacher

und effizienter, dies durch die ETL-Prozesse sicherzustellen, anstatt einen zusammengesetzten Primary Key auf die Fakttabelle zu definieren. *«Dann wäre es also besser, einen künstlichen Schlüssel als Primary Key zu definieren, der über eine Sequence gefüllt wird?»* Wofür soll dieser Primärschlüssel nützlich sein? In einem korrekt modellierten Star Schema wird die Fakttabelle von keiner anderen Tabelle referenziert. Deshalb ist ein Primary Key auf der Fakttabelle nicht notwendig.

«Dann besitzt also die Fakttabelle nur je einen Bitmap Index für jedes Dimensionsattribut. Oder gibt es noch weitere Indizes auf der Fakttabelle?» Für Dimensionsattribute, die oft als Filterkriterium verwendet werden, gibt es in Oracle die Möglichkeit, einen Bitmap Join Index zu erstellen. Dies ist ein Index auf der Fakttabelle, der jedoch ein Attribut aus der Dimensionstabelle indexiert. Somit wird der Join zwischen Dimensions- und Fakttabelle nicht zum Abfragezeitpunkt durchgeführt, sondern beim Laden der Fakttabelle, also während des ETL-Prozesses. Dadurch können Abfragen auf das Star Schema zusätzlich beschleunigt werden. Werden beispielsweise oft Abfragen mit Einschränkungen auf den Produktnamen in der Produktdimension ausgeführt, kann ein entsprechender Bitmap Join Index definiert werden:

```
CREATE BITMAP INDEX sales_prodname_bji
    ON fact_sales (p.prod_name)
  FROM fact_sales s, dim_products p
 WHERE s.prod_id = p.prod_id
```

«Werden solche Bitmap Join Indizes oft verwendet?» Da die Abfragen dank der Star Transformation meistens schon schnell genug sind, werden Bitmap Join Indizes eher selten eingesetzt. Sie können höchstens noch als «Sahnehäubchen» angesehen werden, um die Abfrageperformance der Data Marts noch weiter zu optimieren.

Zusammenfassung

Wichtig bei der Indexierung eines Data Warehouses ist es, dass zwischen den einzelnen Teilbereichen der DWH-Architektur unterschieden wird. Auf Staging Area, Cleansing Area und Core greifen normalerweise nur die ETL-Prozesse zu. Deshalb sollten diese Bereiche nicht oder sehr sparsam indexiert werden. Data Marts sind so zu indexieren, dass Benutzerabfragen mit unterschiedlichsten Kombinationen von Filterkriterien effizient ausgeführt werden können. Dies kann in Oracle mit Hilfe von Bitmap Indizes sowie der Star Transformation optimal erreicht werden.

Kontaktadresse:

Dani Schnider
Trivadis AG
Europa-Strasse 5
CH-8152 Glattbrugg

Telefon: +41(0)44-808 70 20
Fax: +41(0)44-808 70 21
E-Mail dani.schnider@trivadis.com
Internet: www.trivadis.com