Oracle9i für den DBA
ISBN 3-8273-1559-X

3 Datenbankdesign

Beim Aufbau jeder Datenbank und jedes Datenbankschemas sind eine Fülle von Designentscheidungen zu treffen. Diese Entscheidungen stehen auf der einen Seite im Zusammenhang mit konkreten Anforderungen an die Funktionalität und Performance der betreffenden Installation und sind auf der anderen Seite abhängig von den gegebenen technischen Möglichkeiten des eingesetzten Datenbanksystems – in diesem Falle Oracle9i. Oftmals stehen mehrere Alternativen für eine Implementierung zur Verfügung. Das nachfolgende Kapitel möchte an dieser Stelle behilflich sein, um im „Wald“ der technischen Merkmale optimale Entscheidungen für die Implementierung von Schema- und Datenbankobjekten treffen zu können.

3.1 Namenskonventionen

Beim Umgang mit einer Datenbank werden auf den verschiedensten Ebenen Dinge benannt: Datenbanken, Instanzen, Dateien, Tablespaces, Tabellen, Views, Indizes usw. Neben den allgemeingültigen Regeln für die Benennung von Datenbankobjekten, die z.B. die maximale Länge und die erlaubten Zeichen festlegen und vom Datenbanksystem vorgegeben werden, ist es empfehlenswert, eigene Richtlinien für die Namensbildung zu schaffen.

Warum denken wir an dieser Stelle über diese Namenskonventionen nach?

Einige der in Frage kommenden Beweggründe sind:

1. Wenn wir darüber nachdenken, nachdem die Datenbank konfiguriert und die Daten geladen sind, ist es zu spät.
2. Eine Oracle-Datenbank wird auf verschiedenen Ebenen mit diversen Namen identifiziert (u.a. Datenbankname, Instanzname, Netzwerkdienstname sowie Dienstname). Sie können alle verschieden sein, müssen aber nicht.
3. Es gibt immer mehr Situationen, bei denen die im Unternehmen vorhandenen Datenbanken global betrachtet werden – der Enterprise Manager oder ein Storage-Subsystem bzw. SAN sind nur zwei Beispiele. Wenn dann fünf von acht Datenbanken `ORACLE` heißen, ist die Übersichtlichkeit dahin.
4. Konventionen können die Erschliessung des Kontextes der betreffenden Objekte erleichtern: `pk$tabellenname` kann dem Betrachter signalisieren, dass es sich um den Primaärschlüssel der Tabelle `tabellenname` handelt.

Im Folgenden wird zunächst über Datenbanken und deren Namensgebung diskutiert; im Anschluss dann über die Namensgebung bei Datenbankobjekten.

3.1.1 Datenbanken

Bei der Bennung von Datenbanken ist zu unterscheiden zwischen dem per Parameter vergebenen Datenbanknamen (db_name), dem Domänennamen der Datenbank (db_domain) und dem *globalen* Datenbanknamen, der aus den beiden genannten Parameter zusammengesetzt wird, jedoch auch – unabhängig davon – explizit gesetzt werden kann. Die folgenden Abschnitte erläutern die Zusammenhänge.

Domänen

Bei der Vergabe von Namen für eine Oracle-Datenbank gibt es zwei Stellen, an denen eine Domäne auftauchen kann:

- Die db_domain wird zusammen mit db_name (beides Initialisierungsparameter) interner Bestandteil des Datenbanknamens; zusammen ergeben diese Namen auch den Standardwert für den Initialisierungsparameter service_names, der ab Oracle8i zur Adressauflösung bei den Netzwerkdiensten genutzt werden kann.[1]
- Netzwerkdienstnamen, früher auch mit SQL*Net Alias oder Net8-Name bezeichnet, können ebenfalls mit einer Domäne versehen werden. Einen hohen Bekanntheitsgrad unter langjährigen Oracle-DBAs dürfte die Standarddomäne früherer Releases, world, haben.

Die beiden genannten Domänen haben eigentlich nichts miteinander zu tun. Sobald man allerdings mit dem Enterprise Manager arbeitet, entsteht plötzlich doch eine Verknüpfung: Bei der automatischen Erkennung von Netzwerkdiensten liefert der Oracle Agent dem Management-Server alle vorhandenen Instanzen und trägt sie sowohl in den Navigationsbaum als auch optional in die tnsnames.ora-Datei ein – natürlich in der Form db_name.db_domain.

Die Empfehlung an dieser Stelle lautet also, die Domänen in beiden Fällen gezielt zu nutzen und identisch zu halten.

Mittels eines weiteren Initialisierungsparameters können die Domänen zusammen mit Datenbanknamen auf Datenbank-Links übertragen werden: Sobald

```
global_names = true
```

steht, dürfen Datenbank-Links nur so benannt werden wie die Zieldatenbank. Diese Forderung ist gerade in *gewachsenen* Datenbanken schwer einzuhalten, da der funktionale Verlust, der durch das Deaktivieren eines Datenbank-Links entsteht, oft schwer zu überblicken ist. Fängt man jedoch mit einem neuen System *bei Null* an, so ist diese Einstellung unbedingt zu empfehlen.

Einfache Empfehlungen für Standarddomänen sind firma.de, firma.com oder einfach firma. Da die im Oracle-Namensmodell verwendeten Domänen nicht identisch zu den (meist jedoch ähnlichen) Netzwerkdomänen sein müssen, können mit

1. In der CONNECT_DATA-Klausel wird hierzu statt dem Parameter (SID=XXX) der Parameter (SERVICE_NAME=XXX[.DOM]) angegeben. Eine Verbindung wird hergestellt, wenn der angegebene SERVICE_NAME einem der in service_names angegebenen Namen entspricht.

Hilfe der Domänen zusätzliche Informationen hinterlegt werden; z.B. können alle produktiven Datenbanken in die Domäne `prod.firma.de` gelegt werden.

Datenbanknamen

Ausgehend davon, dass Domänen verwendet werden, kann sich der Datenbankname auf die Funktion der Datenbank fokussieren. Datenbanknamen wie `oradb` sind wenig sinvoll. Es sollten eher Namen wie `dwh` für Data Warehouse, `pps` für die Datenbank eines PPS oder `xyz` für die Datenbank unter der Anwendung XYZ ausgewählt werden.

Der Datenbankname kann insgesamt acht Zeichen lang sein. Im Hinblick auf die nächste Migration kann es sinnvoll sein, dem Datenbanknamen eine Nummer hinzuzufügen, also z.B. `pps01`.

Der Datenbankname wird als Initialisierungsparameter `db_name` hinterlegt. Er kann beim `CREATE DATABASE`-Kommando angegeben werden. Es wird grundsätzlich empfohlen, dies zu tun – damit hat man die Sicherheit, mit den richtigen Initialisierungsparametern zu arbeiten. Arbeitet man z.B. mit der falschen Initialisierungsparameterdatei, so geht das `CREATE DATABASE`-Kommando schief, und das ist gut so.

Der Datenbankname, zusammengesetzt aus `db_name` und `db_domain`, wird beim Anlegen der Datenbank u.a. dazu verwendet, den *globalen* Datenbanknamen abzuleiten. Dieser kann über die View `global_name` abgefragt werden. Der globale Datenbankname wird dazu verwendet, die Namen von Datenbank-Links zu erzwingen. Mit gesetztem Initialisierungsparameter

```
globale_names = true
```

muss – wie bereits erwähnt – ein Datenbank-Link immer genauso heißen wie der globale Datenbankname der Zieldatenbank. Unabhängig vom eigentlichen Datenbanknamen kann der globale Datenbankname mit dem Kommando

```
ALTER DATABASE RENAME GLOBAL_NAME TO globdb.firma.de;
```

geändert werden.

Der Datenbankname selbst – Parameter `db_name` – kann ausschließlich über ein `CREATE CONTROLFILE`-Kommando geändert werden.

Instanznamen

In der Oracle-Architektur sind Datenbank und Instanz getrennt, und somit können auch verschiedene Namen vergeben werden. Der Instanzname verfügt nicht über eine Domäne; er muss allerdings auf jedem Knoten eindeutig sein.

Auch wenn es nicht zwingend erforderlich ist – der Instanzname sollte gleich oder zumindest ähnlich dem Datenbanknamen sein. Auf Grund der fehlenden Domänen kann er aus Übersichtlichkeitsgründen leicht abgewandelt werden, z.B. kann die Instanz zur Datenbank `dwh05.prod.firma.de` den Namen `pdwh05` tragen, um die Eigenschaft Produktivsystem auch im Instanznamen zu verdeutlichen.

Die maximale Länge des Instanznamens, hinterlegt im Umgebungsparameter `oracle_sid`, ist plattformspezifisch. Acht Stellen werden bei den getesteten Plattformen problemlos akzeptiert; mehr als acht Stellen werden generell nicht empfohlen.

Datenbanken im Netzwerk

Sobald man auf eine Datenbank im Netzwerk zugreifen will, kommt zusätzlich zu den bereits diskutierten Namen ein Netzwerkdienstname ins Spiel. Hiermit wird die Datenbank im Netzwerk identifiziert.

Netzwerkdienstnamen können mit Domänen versehen werden. Sie sehen daher den globalen Datenbanknamen sehr ähnlich und werden oft mit diesen verwechselt.

Hat man sich bei den Datenbanknamen Gedanken gemacht, so kann man natürlich die Netzwerkdienstnamen identisch zu den Datenbanknamen inklusive Domäne wählen. Diese Wahl trifft z.B. auch der Enterprise Manager, wenn er eine über den Agenten entdeckte Datenbank in sein Repository einträgt.

Oft gibt es für eine Datenbank mehrere Netzwerkdienstnamen. Oft wird für eine Anwendung ein spezieller Netzwerkdienstname verwendet, so dass die Anwendungsdaten bei Bedarf zwischen verschiedenen Datenbanken verschoben werden können.

Dieses Konzept findet sich auch bei den Dienstnamen als Instanzparameter wieder: Hiermit kann jeder Instanz eine Liste von Diensten (`service_names`) mitgegeben werden. Der Parameter wird standardmäßig auf den Wert `db_name.db_domain` gesetzt. Beim Verbindungsaufbau über die Oracle Net-Schnittstelle wird ein `service_name` mitgegeben, der mindestens einem Wert aus der Liste der `service_names` der Instanz entsprechen muss.

3.1.2 Objektnamen

Grundregeln

Folgende Regeln existieren für die Namensgebung von Objekten in der Datenbank:

1. Namen sind bis zu 30 Byte lang (Ausnahme: Datenbankname bis zu 8 Byte; Datenbank-Links bis zu 128 Byte).
2. Namen dürfen keine Anführungsstriche enthalten.
3. Groß-/Kleinschreibung wird bei Namen nicht unterschieden.
4. Namen müssen mit einem alphabetischen Zeichen des Datenbankzeichensatzes anfangen.
5. Namen können ausschließlich alphanumerische Zeichen, _, $ und # enthalten; Namen von Datenbank-Links zusätzlich . und @.
6. Reservierte Worte dürfen nicht verwendet werden.
7. `dual` darf nicht verwendet werden.
8. Auch nicht reservierte Worte, die aber eine Bedeutung im Oracle-Umfeld haben (z.B. `DIMENSION` oder `SEGMENT`), dürfen nicht verwendet werden.

9. Zwei Objekte im gleichen Namensraum (d.h. Schema) dürfen nicht den gleichen Namen haben.
10. Die Spalten einer Tabelle bzw. einer View müssen alle unterschiedliche Namen haben. Spalten unterschiedlicher Tabellen oder Views können gleich lautende Namen haben.
11. Prozeduren oder Funktionen innerhalb eines Packages können gleich lautende Namen haben, falls sie eine unterschiedliche Anzahl von Argumenten oder Argumente verschiedenen Typs haben.
12. Namen können in doppelten Anführungszeichen geschrieben werden. In diesem Fall treffen die Regeln 3 bis 7 nicht zu.

Allgemeine Empfehlungen

Neben den eigentlichen Regeln sind aber folgende Überlegungen wichtig, um Namen für Datenbankobjekte zu finden:

- Namen sollten lesbar und ausführlich sein. Die 30 Zeichen können ruhigen Gewissens ausgenutzt werden. Aussagekräftige Namen erhöhen die Lesbarkeit von SQL-Befehlen und Programmen erheblich.
- Passend zu den Voreinstellungen vieler Design Werkzeuge sollten Tabellennamen im Plural definiert werden. (Entitäten werden meist im Singular definiert.)
- Spaltennamen für Spalten mit identischer Bedeutung sollten im gesamten Schema konsistent benannt werden. Somit können sowohl Beziehungen zwischen Tabellen als auch Sachverhalte klar umschrieben werden.

Eine Spalte, die den Preis eines Produktes in Euro angibt, sollte ruhig `preis_in_euro` genannt werden. Eine solche Spalte lediglich `preis` zu nennen, kann verwirrend sein (man denke nur an die Euro-Umstellung). Sollte die Währung in einer eigenen Spalte `waehrung_id` gekennzeichnet sein (wobei dies natürlich einen Fremdschlüssel in einer Tabelle `waehrungen` repräsentiert), so kann der Spaltenname z.B. `preis_in_waehrung` sein.

An dieser Stelle empfehlen wir, für deutsche Namen mit Umlauten `ae`, `oe` und `ue` einzusetzen, obwohl auch Namen wie `währungen` möglich sind (dies auch ohne doppelte Anführungszeichen). Solche Namen können aber Probleme bereiten, z.B. wenn Sie bei der Programmierung in Oracle-fremden Produkten verwendet werden.

3.1.3 Systemobjekte

Auch für Systemobjekte wie Tablespaces, Datendateien, Rollback-Segmente usw. müssen Namen gefunden werden. Hierbei ist auf jeden Fall Kreativität gefragt, denn Systemobjekte sind für die Anwendung nicht unmittelbar erforderlich und spielen somit auch keine Rolle im System, die a priori benannt werden kann.

Daher ist es wichtig, sich einige Grundregeln zu überlegen, um den Überblick im Datenbankdschungel nicht zu verlieren.

Tablespaces für Systemobjekte wie Rollback- und Temporärsegmente sollten immer gleich genannt werden, z.B. `rbs` und `temp`. Bei Tablespaces für Anwendungsdaten

bringt die Anwendung oft einen Vorschlag mit. Ansonsten nimmt man für eine Anwendung X gerne `x_data` für Tabellen, `x_index` für Indizes usw.

Bei den Datendateien sollte gewährleistet sein, dass man am vollqualifizierten Dateinamen folgende Eigenschaften erkennen kann:

- Diese Datei gehört zu einer Oracle-Datenbank.
- Den Datenbanknamen (ohne Domäne)
- Den Tablespace-Namen
- Eine laufende Nummer für die n-te Datei des Tablespaces

Bei Rollback-Segmenten ist die Standardstrategie eine Nummerierung, z.B. `rbs01`, `rbs02` usw. Lediglich Rollback-Segmente für spezielle Aufgaben sollten auch entsprechend genannt werden, z.B. `rbs_batch`.

3.2 Physische Implementierung von Entitäten

Die Analyse der Anforderungen und die daran anschließende Entwicklung eines konzeptionellen Datenmodells ist bekanntlich systemneutral, d.h. unabhängig von dem eingesetzten Datenbanksystem. Anders verhält es sich dagegen bei der Umsetzung des konzeptionellen Modells in das physische Datenmodell. Hierbei sind die technischen Möglichkeiten des Zielsystems – in unserem Fall des Datenbanksystems Oracle8i oder Oracle9i – mit den funktionalen und operationalen Anforderungen der betreffenden Anwendung in Einklang zu bringen.

Die folgenden Abschnitte behandeln vor diesem Hintergrund die verschiedenen technischen Möglichkeiten, Entitäten in einer Oracle-Datenbank zu implementieren, und die mit diesen Möglichkeiten verbundenen Auswirkungen im Hinblick auf die Zugriffe und die Verwaltbarkeit der betreffenden Objekte. Ziel ist es dabei, die zur Verfügung stehenden Organisationsformen von Tabellen geschickt für die Optimierung von Zugriffen und Verwaltungsoperationen zu nutzen.

3.2.1 Überblick

Die Zeiten, in denen Entitäten einzig in Form von Heap-Tabellen oder Clustern implementiert werden konnten, sind lange vorbei. Mittlerweile existieren weitergehende Möglichkeiten der physischen Organisation, die – bei sinnvoller Nutzung – wesentlich zur Optimierung von Anwendungen und zur Vereinfachung der Administration beitragen können. Diese Organisationsformen sind im Einzelnen:

- die „klassischen“ in *Heap*-Form organisierten Tabellen, die Daten unsortiert speichern,
- im Cluster abgelegte Tabellen, deren Daten in Abhängigkeit von Cluster-Schlüsseln physisch gespeichert werden,
- Index-organisierte Tabellen, deren Daten über B*Baumstrukturen organisiert sind,

- partitionierte Tabellen, die physisch unabhängige Partitionen zu einer „logischen“ Tabelle zusammenfassen,
- externe Tabellen, die auf externe, in Dateiform abgelegte Daten zugreifen,
- temporäre Tabellen, die Daten für die Dauer einer Transaktion oder Session speichern,
- Objekttabellen, deren Datensätze über Objektbezeichner referenzierbar sind, und die die Einbettung von geschachtelten Tabellen (*nested tables*) erlauben.

Diese Organisationsformen werden mit ihren Besonderheiten in den folgenden Abschnitten im Detail besprochen.

3.2.2 Heap-organisierte Tabellen

Heap-organisierte Tabellen sind die „klassischen“ Datenbanktabellen von Oracle-Systemen. Sie speichern ihnen zugewiesene Datensätze unsortiert in einem beliebigen, für Insert-Operationen zur Verfügung stehenden Oracle-Block. Bei einer neu angelegten, leeren Tabelle bedeutet dies, dass die Daten in der Reihenfolge der Insert-Operationen nacheinander – im Anschluss an den Header-Block – in die Blöcke des betreffenden Datensegmentes geschrieben werden. Im fortgeschrittenen Zustand der Tabelle, nachdem auch DELETE- und UPDATE-Operationen die Tabelle verändert haben, werden Datenblöcke, die für INSERT-Operationen zur Verfügung stehen, über so genannte Freilisten verwaltet. Aufeinander folgende INSERT-Operationen werden dadurch nicht zwangsläufig in physisch aufeinander folgende Blöcke geschrieben, sondern können beliebig verstreut werden. Der jeweilige „Höchststand“ der Füllung wird durch die so genannte Hochwassermarke (*high water mark*) gekennzeichnet. Blöcke, die sich jenseits der Hochwassermarke befinden, haben demnach noch nie Datensätze für die betreffende Tabelle enthalten.

Der Zugriff auf Heap-organisierte Tabellen erfolgt entweder durch das vollständige Lesen der Tabelle (*full table scan*) oder über eine vorgegebene Satzadresse. Beim vollständigen Lesen werden alle Blöcke vom Beginn des Segmentes bis zur Hochwassermarke gelesen. Bei dynamischen Tabellen mit einem hohen Anteil an DELETE-Operationen kann dies bedeuten, dass entsprechend viele Leerblöcke gelesen werden müssen – ein typischer Fall für eine Reorganisation der betreffenden Tabelle. Der Zugriff über Satzadressen (*rowids*) erfolgt dagegen gezielt auf bestimmte Blöcke in bestimmten, über Dateinummern identifizierten Dateien. Die Hochwassermarke und die „Datenlöcher“ spielen in diesen Fällen keine Rolle. Die Satzadressen selbst werden in der Regel über entsprechende Indexzugriffe vom System ermittelt. Die Indizes werden in separaten Segmenten aufgebaut. Für aufwändig indizierte Tabellen kann der für Indizes benötigte Speicherplatz dem der Tabelle nahe kommen oder ihn gar überschreiten.

Heap-organisierte Tabellen sind gut geeignet für Entitäten, die keine besonderen Anforderungen an die Menge der Daten, deren physische Speichersequenz oder deren Indizierung stellen. Heap-Tabellen sind darüber hinaus eine gute Standardlösung für den Fall, dass die genauen Anforderungen an die Speicherung nicht bekannt sind.

Im folgenden Beispiel wird die Heap-Tabelle `kunde` mit den Attributen `kdnr` (Schlüssel), `kname` und `kadresse` erstellt. Die `organization`-Klausel ist dabei optional:

```
CREATE TABLE kunde (kdnr NUMBER CONSTRAINT kunde_pk PRIMARY KEY,
                    kname VARCHAR2(100), kadresse VARCHAR2(500))
ORGANIZATION HEAP;
```

3.2.3 Cluster

In Clustern abgelegte Tabellen speichern Daten in Abhängigkeit von Cluster-Schlüsseln in für diese Cluster-Schlüssel ausgewiesenen Datenblöcken. Teilen mehrere Tabellen einen Cluster-Schlüssel, können sie gemeinsam in einem Cluster angelegt werden. Entsprechend werden Datensätze mit gleichen Cluster-Schlüsseln gemeinschaftlich gespeichert.

Die Zuordnung von Cluster-Schlüsseln zu Datenblöcken und entsprechend der Zugriff auf diese Sätze wird über zwei unterschiedliche Verfahren – durch Indizierung und Hash-Algorithmen – geregelt. Dementsprechend sprechen wir von Index-Clustern oder Hash-Clustern.

Index-Cluster

Bei Index-Clustern übernimmt ein B*-Index, der so genannte Cluster-Index, die Reservierung einzelner Blöcke für bestimmte Cluster-Schlüssel. Da der Index im Kontext der Insert-Operationen gefüllt wird, erfolgt die Reservierung der Blöcke in der Reihenfolge, wie die Insert-Operationen der Cluster-Schlüssel von den Programmen durchgeführt werden. Die Hochwassermarke wird bei leeren Index-Clustern – wie auch bei Heap-Tabellen – synchron zu den Insert-Operationen verschoben.

Im folgenden Beispiel wird der Index-Cluster `testcl` mit den Tabellen `kunde` und `auftrag` angelegt:

```
CREATE CLUSTER testcl (kdnr NUMBER) INDEX;

CREATE INDEX i$testcl ON CLUSTER testcl;

CREATE TABLE kunde (kdnr NUMBER, kname VARCHAR2(100),
                    kadresse VARCHAR2(500))CLUSTER testcl(kdnr);

CREATE TABLE auftrag (aufnr NUMBER, kdnr NUMBER, aufdat DATE)
  CLUSTER testcl(kdnr);
```

Listing 3.1: Anlegen Index-Cluster

Hash-Cluster

Beim Hash-Cluster ist die Reihenfolge der Insert-Operationen nicht relevant für die Zuordnung der Datenblöcke zu den Cluster-Schlüsseln. Hier bestimmt eine Hash-Funktion den Block, in welchem ein Cluster-Schlüssel zu speichern ist. Die Hash-Funktion gibt in der Regel den Blockversatz relativ zum Beginn des betreffenden

Segmentes an. Damit nicht mehrere Cluster-Schlüssel einen Datenblock teilen, muss beim Anlegen eines Hash-Clusters die erwartete maximale Anzahl von Cluster-Schlüsseln möglichst präzise angegeben werden. Diese Schlüsselanzahl bestimmt direkt die Anzahl der Blöcke, die unterhalb der Hochwassermarke bereitgehalten werden müssen – unabhängig davon, ob Daten eingefügt wurden oder nicht. Insert-Operationen bewegen daher nur dann die Hochwassermarke, wenn ein Blocküberlauf stattfindet.

Im folgenden Beispiel wird der Hash-Cluster `testclh` mit den Tabellen `kundeh` und `auftragh` angelegt. Es werden 1.000 Cluster-Schlüssel erwartet. Entsprechend werden 1.000 Blöcke initialisiert. Die Hash-Funktion arbeitet – spezifiziert durch die `hash is`-Klausel – auf Basis der numerischen Spalte `kdnr`:

```
CREATE CLUSTER testclh (kdnr NUMBER(10,0)) HASHKEYS 1000 HASH IS kdnr;

CREATE TABLE kundeh (kdnr NUMBER(10,0), kname VARCHAR2(100),
                     kadresse VARCHAR2(500))
 CLUSTER testclh(kdnr);

CREATE TABLE auftragh (aufnr NUMBER, kdnr NUMBER(10,0), aufdat DATE)
CLUSTER testclh(kdnr);
```

Listing 3.2: Anlegen Hash-Cluster

Cluster-Implementierungen sind keine Standardlösungen, sie sollten sorgfältig geplant werden. Generell können sie die folgenden *Vorteile* bieten:

- „physische Joins“ durch die Speicherung mehrerer Tabellen in einem Cluster, sofern der Cluster-Schlüssel identisch mit dem Join-Schlüssel ist
- Adressierung der Datensätze im Hash-Cluster ohne Vorhandensein eines Indexes
- „sortierte“ Speicherung der Datensätze

Diesen Vorteilen stehen jedoch eine Reihe von *Nachteilen* gegenüber:

- Für den Fall, dass nur eine der im Cluster abgelegten Tabellen gelesen wird, kommt es zu erhöhten Blockzugriffen.
- bei nur partiell gefüllten Hash-Clustern erhöhte Anzahl von Blockzugriffen
- Ungenügende Füllung von Cluster-Blöcken bei „geringem“ Datenvolumen pro Cluster-Schlüssel; umgekehrt Verkettung von Cluster-Blöcken (Überlauf) bei „großem“ Datenvolumen pro Cluster-Schlüssel; nicht einschätzbare Wirkungen bei „schwankendem“ Datenvolumen pro Cluster-Schlüssel
- Eindeutigkeit kann nur über zusätzliche Indizes sichergestellt werden, die zusätzlichen Speicherplatz kosten.
- ineffiziente Segmentgrößen bei Hash-Clustern, wenn die Anzahl der Cluster-Schlüssel nicht bekannt ist oder falsch eingeschätzt wurde
- Die Hash-Funktion kann nur für Gleichheitsoperatoren genutzt werden.

- Cluster können nicht repliziert und partitioniert werden.

Die vorangehenden Abschnitte machen deutlich, dass Cluster-Implementierungen nur selten zum Einsatz kommen dürften.

3.2.4 Index-organisierte Tabellen (IOT)

Index-organisierte Tabellen speichern ihre Daten – sortiert nach dem Primärschlüssel – in Form eines B*-Index-Baumes. Im Gegensatz zu normalen B*-Index-Bäumen enthalten die B*-Bäume von IOT-Tabellen jedoch auch Spalten, die nicht zu dem Tabellenschlüssel gehören. Es ist darüber hinaus möglich, bestimmte Spalten, die nicht zum Schlüssel gehören, „auszulagern“, d.h. außerhalb des B*-Baumes in einem separaten Segment – dem Überlaufbereich – zu speichern. Auf diese Weise können die Datensätze in den Blättern des Baumes kurz gehalten und dadurch der Baum kompakter aufgebaut werden. Beginnend mit Oracle8i lassen sich zusätzliche Sekundärindizes für IOT-Tabellen definieren, um Zugriffe, die nicht über die Primärschlüsselspalten erfolgen, zu optimieren. Diese Sekundärindizes enthalten logische Satzadressen für den Zugriff auf den Index-Baum. Da Einträge in Index-Bäumen durch INSERT- und UPDATE-Operationen und damit verbundenen Split-Operationen auf den Index-Blättern möglicherweise verschoben werden, können Zugriffe über logische Satzadressen fehlschlagen. In diesen Fällen ist dann eine zusätzliche Scan-Operation über den Primärschlüssel notwendig, um den betreffenden Datensatz zu lesen.

Im folgenden Beispiel wird die Index-organisierte Tabelle `artikelpreis` mit den Schlüsselspalten `artnr`, `von_datum` und `bis_datum` angelegt. Die Spalten `kommentar` und `rabattliste` werden dabei – sofern der Schwellenwert von 10 % pro Block und Datensatz überschritten wird – in einen Überlaufbereich ausgelagert, der sich in der Tablespace `ts_over` befindet:

```
CREATE TABLE artikelpreis
 (artnr VARCHAR2(20), von_datum  DATE, bis_datum DATE,
  preis_euro number(6,2), kommentar VARCHAR2(2000),
  rabattliste number(4),
  CONSTRAINT pk_artikelpreis PRIMARY KEY(artnr, von_datum, bis_datum)
 )
 ORGANIZATION INDEX TABLESPACE ts_data
 PCTTHRESHOLD 10 INCLUDING preis_euro OVERFLOW TABLESPACE ts_over;
```

Listing 3.3: Index-organisierte Tabelle

Im nächsten Beispiel wird der Sekundärindex `i$artpreis_rab` für die Spalte `rabattliste` angelegt:

```
CREATE INDEX i$artpreis_rab ON artikelpreis(rabattliste);
```

Wie normale Indizes, so lassen sich auch Index-organisierte Tabellen zusätzlich komprimieren. Bei der Komprimierung werden die Spalten des Primärschlüssels in Präfix- und Suffixbereiche aufgeteilt und die wiederholt auftretenden Werte der Präfixe innerhalb *eines* Blockes unterdrückt. Durch die Komprimierung wird eine

Verkürzung der Datensätze in den Blättern des Index-Baumes erreicht, die den Index kompakter und damit für I/O-Operationen effizienter machen.

Im folgenden Beispiel wird die oben beschriebene Tabelle `artikelpreis` mit einer Komprimierung auf der ersten Schlüsselspalte – `artnr` – angelegt. Dementsprechend werden redundante Artikelnummern innerhalb eines Index-Blattes unterdrückt:

```
CREATE TABLE artikelpreis
 (artnr VARCHAR2(20), von_datum  DATE, bis_datum DATE,
  preis_euro NUMBER(6,2), kommentar VARCHAR2(2000),
  rabattliste NUMBER(4),
  CONSTRAINT pk_artikelpreis PRIMARY KEY(artnr, von_datum, bis_datum)
 )
 ORGANIZATION INDEX COMPRESS 1 TABLESPACE ts_data
 PCTTHRESHOLD 10 INCLUDING preis_euro OVERFLOW TABLESPACE ts_over;
```

Listing 3.4: Index-organisierte Tabelle (komprimiert)

Index-organisierte Tabellen sind sehr gut geeignet für die Umsetzung aller Entitäten, die einen großen Teil ihres Informationsgehaltes aus den Schlüsselspalten beziehen. Hierbei schlägt vor allem die Einsparung von Speicherplatz zu Buche, da keine separaten Segmente für Tabellen und Indizes angelegt werden müssen. Mögliche Kandidaten sind z.B.

- so genannte Intersection-Tabellen, die n:m-Relationen[2] des Datenmodells auflösen und
- große Referenztabellen, die nur wenige Spalten enthalten, die nicht zum Schlüssel gehören.

3.2.5 Partitionierte Tabellen

Partitionierte Tabellen bieten die Möglichkeit, Datensätze nach vorgegebenen Verteilungskriterien physikalisch unabhängig, d.h. in separaten Tablespaces mit unterschiedlich großen Extents, zu speichern und diese Partitionen separat zu verwalten. Aus Sicht der Anwendung sind alle Daten über den Namen der partitionierten Tabellen les- und schreibbar, der Optimizer dagegen kennt die physikalische Aufteilung der Tabelle und ist in der Lage, entsprechende Zugriffspfade zu generieren. Die zu einer partitionierten Tabelle gehörenden Indizes lassen sich ebenso partitionieren – nach eigenständigen oder identischen Kriterien wie die Tabelle selbst. Für die Festlegung der Verteilungskriterien stehen drei unterschiedliche Strategien zur Verfügung:

- Die *Range-Partitionierung* verteilt die Daten nach festen, in der Syntax vorgegebenen Wertebereichen.
- Die *Hash-Partitionierung* regelt die Verteilung über eine Hash-Funktion.

2. Jedes A bezieht sich auf ein oder mehrere B und umgekehrt kann sich jedes B auf ein oder mehrere A beziehen.