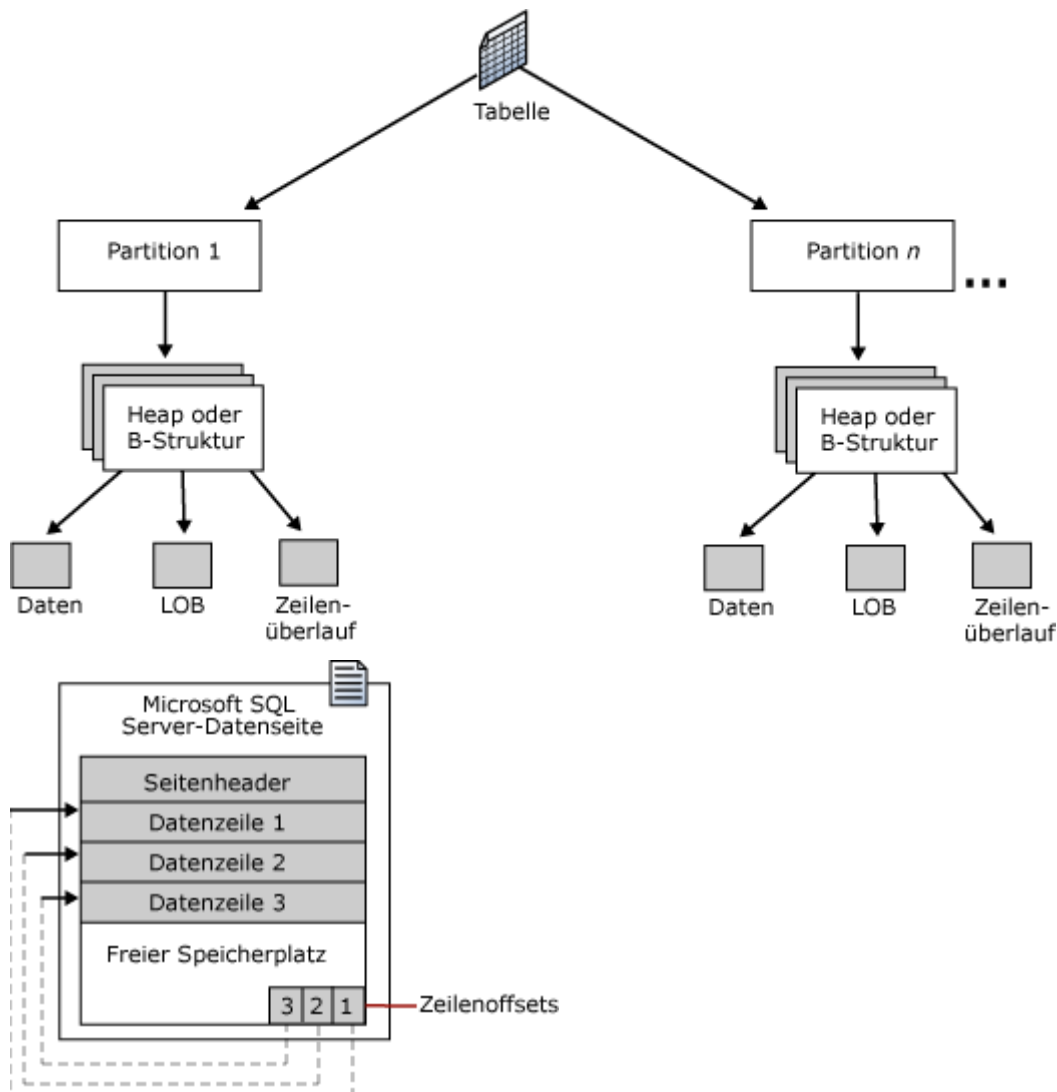
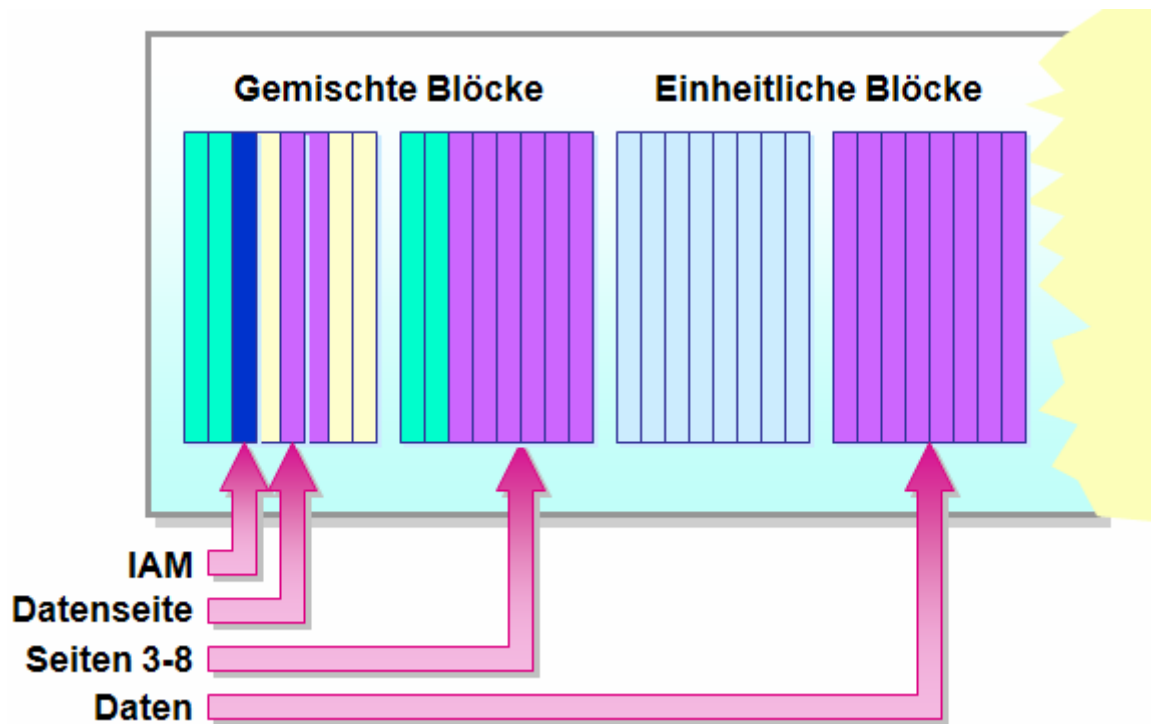
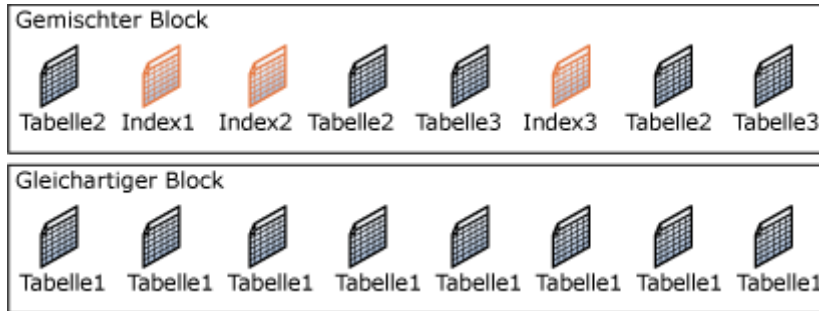


3 Indizes

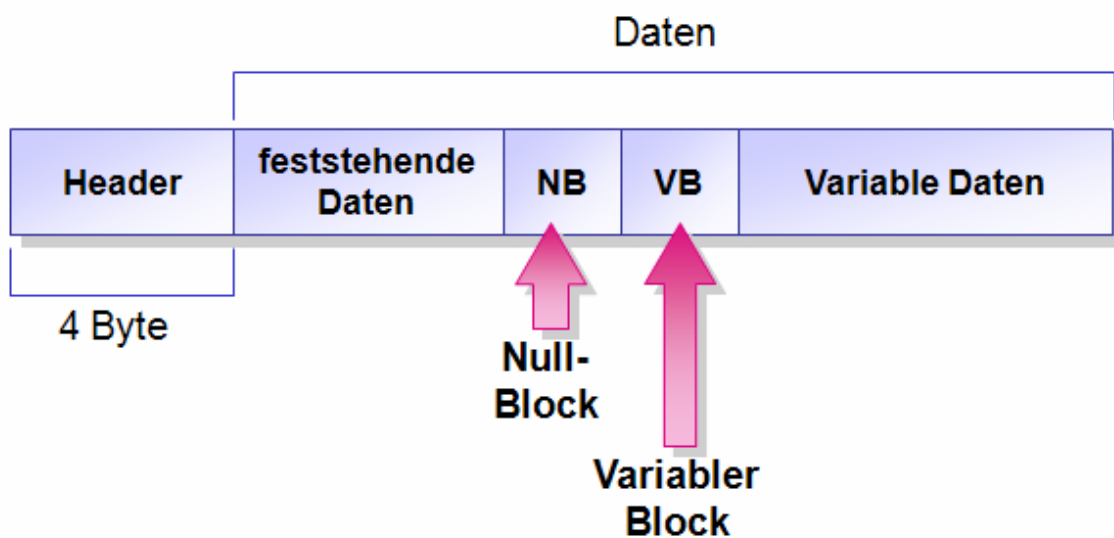
3.1 Indexarchitektur von SQL Server

Die folgende Abbildung zeigt die Organisationsstruktur einer Tabelle. Eine Tabelle befindet sich in einer oder mehreren Partitionen, und jede Partition enthält Datenzeilen entweder in einer Heap- oder in einer gruppierten Indexstruktur. Die Seiten des Heaps oder des gruppierten Indexes werden je nach den Spaltentypen in den Datenzeilen in einer oder mehreren Zuordnungseinheiten verwaltet.





Eine Datenseite ist wie folgt aufgebaut:



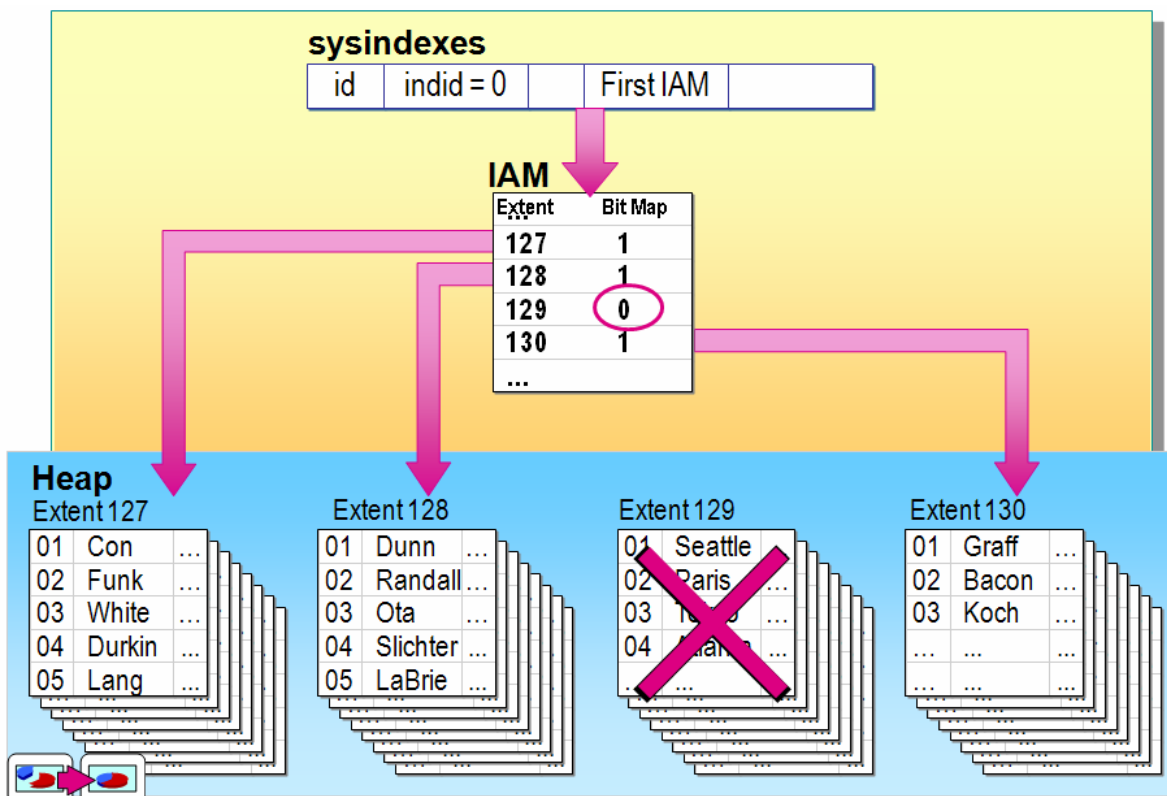
3.2 Suchen in einem Heap ("Table Scan")

Ein Heap ist eine Tabelle ohne gruppierten Index. Heaps haben eine Zeile in **sys.partitions**, mit **index_id = 0** für jede vom Heap verwendete Partition. Standardmäßig verfügt ein Heap über eine einzelne Partition. Wenn ein Heap über mehrere Partitionen verfügt, hat jede Partition eine Heapstruktur, in der die Daten für die jeweilige Partition enthalten sind. Wenn ein Heap z. B. über vier Partitionen verfügt, gibt es vier Heapstrukturen – jeweils eine in jeder Partition.

Je nach den im Heap enthaltenen Datentypen weist jede Heapstruktur eine oder mehrere Zuordnungseinheiten auf, um die Daten für eine bestimmte Partition zu speichern und zu verwalten. Zumindest verfügt jeder Heap über eine **IN_ROW_DATA**-Zuordnungseinheit pro Partition. Der Heap hat außerdem eine **LOB_DATA**-Zuordnungseinheit pro Partition, wenn diese LOB-Spalten (Large Object) enthält. Darüber hinaus verfügt er über eine **ROW_OVERFLOW_DATA**-Zuordnungseinheit pro Partition, wenn diese Spalten mit variabler Länge enthält, die das Zeilengrößenlimit von 8.060 Byte überschreiten.

Die Spalte **first_iam_page** in der **sys.system_internals_allocation_units**-Systemsicht verweist auf die erste IAM-Seite (Index Allocation Map) in der Kette der IAM-Seiten, die zur Verwaltung des Speicherplatzes verwendet werden, der dem Heap zugeordnet ist. SQL Server 2005 verwendet die IAM-Seiten für Bewegungen innerhalb des Heaps. Die Datenseiten und die Zeilen innerhalb eines Heaps weisen keine bestimmte Reihenfolge auf und sind nicht verknüpft. Die einzige logische Verbindung zwischen den Datenseiten sind die Informationen, die auf den IAM-Seiten aufgezeichnet sind.

Tabellenscans oder serielle Lesevorgänge in einem Heap können durchgeführt werden, indem die IAM-Seiten gescannt werden, um auf diese Weise die Blöcke zu ermitteln, die Seiten des Heaps enthalten. Da die IAM die Blöcke in derselben Reihenfolge darstellt, in der sie in der Datendatei vorliegen, werden serielle Heapskans immer sequenziell durch jede Datei ausgeführt. Das Verwenden der IAM-Seiten zum Festlegen der Scanfolge bedeutet weiterhin, dass Zeilen aus dem Heap nicht notwendigerweise in der Reihenfolge zurückgegeben werden, in der sie eingefügt wurden.



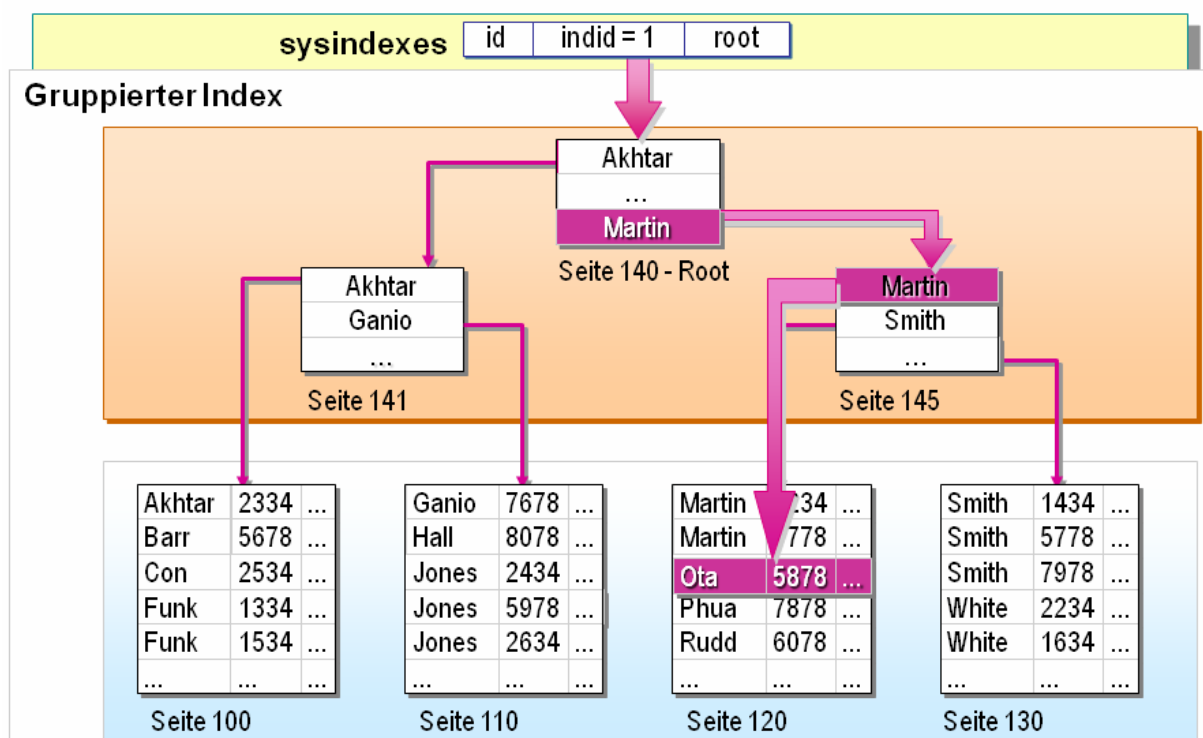
3.3 Suchen mit gruppiertem Index

In SQL Server sind Indizes in Form von B-Strukturen (**balanced tree**) aufgebaut. Jede Seite in der B-Struktur eines Indexes wird als Indexknoten bezeichnet. Der oberste Knoten der B-Struktur wird als Stammknoten bezeichnet. Die Knoten auf der untersten Ebene des Indexes werden als Endknoten bezeichnet. Alle anderen Indexebenen zwischen dem Stamm- und den Endknoten werden zusammenfassend als Zwischenebenen bezeichnet. In einem gruppierten Index enthalten die Endknoten die Datenseiten der zugrunde liegenden Tabelle. Die Stamm- und Zwischenebenenknoten enthalten Indexseiten, in denen Indexzeilen enthalten sind. Jede Indexzeile enthält einen Schlüsselwert und einen Zeiger auf eine Seite einer Zwischenebene in der B-Struktur oder auf eine Datenzeile in der Blattebene des Indexes. Die Seiten auf jeder Ebene des Indexes sind durch eine doppelt verknüpfte Liste miteinander verknüpft.

Gruppierte Indizes besitzen eine Zeile in **sys.partitions**, wobei **index_id = 1** für jede Partition ist, die vom Index verwendet wird. Standardmäßig besitzt ein gruppiertes Index eine Partition. Wenn ein gruppiertes Index über mehrere Partitionen verfügt, besitzt jede Partition eine B-Struktur, die die Daten für diese bestimmte Partition enthält. Wenn ein gruppiertes Index z. B. vier Partitionen besitzt, sind vier B-Strukturen vorhanden, eine in jeder Partition.

Abhängig von den Datentypen im gruppierten Index weist jede gruppierte Indexstruktur eine oder mehrere Zuordnungseinheiten auf, in denen die Daten für eine bestimmte Partition gespeichert und verwaltet werden. Jeder gruppierte Index weist mindestens eine **IN_ROW_DATA**-Zuordnungseinheit pro Partition auf. Der gruppierte Index besitzt außerdem eine **LOB_DATA**-Zuordnungseinheit pro Partition, wenn LOB-Spalten (Large Object) vorhanden sind. Außerdem ist eine **ROW_OVERFLOW_DATA**-Zuordnungseinheit pro Partition vorhanden, wenn der Index Spalten variabler Länge aufweist, die die Zeilengrößenbegrenzung von 8.060 Byte übersteigen.

Für einen gruppierten Index verweist die **root_page**-Spalte in **sys.system_internals_allocation_units** auf die oberste Ebene des gruppierten Indexes für eine bestimmte Partition. SQL Server durchsucht den Index in absteigender Reihenfolge nach der Zeile, die dem Schlüssel eines gruppierten Indexes entspricht. Um einen Bereich von Schlüsselwerten zu finden, bewegt sich SQL Server durch den Index, um den Anfangsschlüsselwert des Bereichs zu finden. Anschließend werden die Datenseiten mithilfe der Zeiger auf vorherige und folgende Seiten gescannt. Um die erste Seite in der Kette aus Datenseiten zu finden, beginnt SQL Server beim Stammknoten und folgt den Zeigern ganz links im Index.



3.4 Suchen mit nicht gruppierten Indizes

Nicht gruppierte Indizes weisen dieselbe B-Baumstruktur auf wie gruppierte Indizes, mit Ausnahme der beiden folgenden signifikanten Unterschiede:

- Die Datenzeilen der zugrunde liegenden Tabelle werden nicht auf der Grundlage ihrer nicht gruppierten Schlüssel sortiert und gespeichert.
- Die Blattebene eines nicht gruppierten Indexes besteht aus Indexseiten, nicht aus Datenseiten.

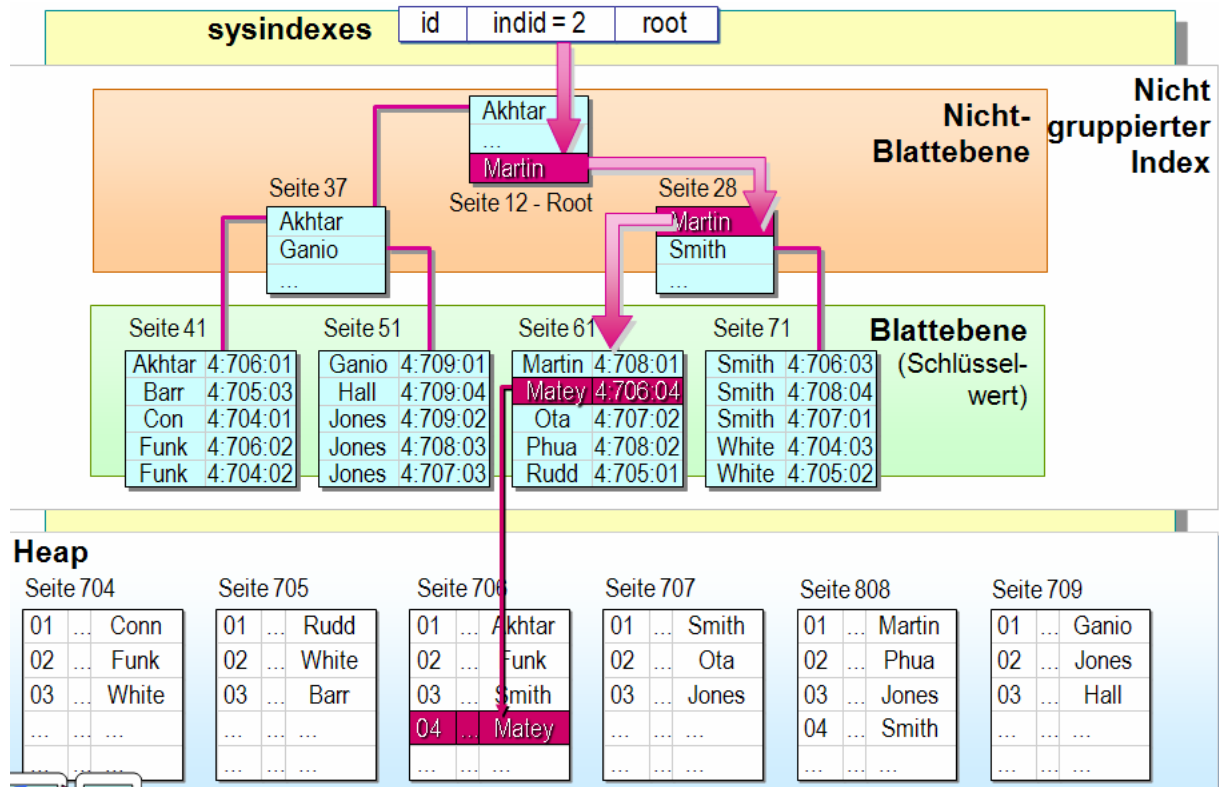
Nicht gruppierte Indizes können für eine Tabelle oder eine Sicht mit einem gruppierten Index oder einem Heap definiert werden. Jede Zeile eines nicht gruppierten Indexes enthält einen nicht gruppierten Schlüsselwert und einen Zeilenlokator. Dieser Zeilenlokator zeigt auf die Datenzeile des gruppierten Indexes oder des Heaps mit dem Schlüsselwert.

Zeilenlokatoren in nicht gruppierten Indexzeilen bestehen entweder aus Zeigern, die auf jeweils eine Zeile zeigen, oder aus einem Schlüssel eines gruppierten Indexes für eine Zeile, wie im Folgenden erläutert:

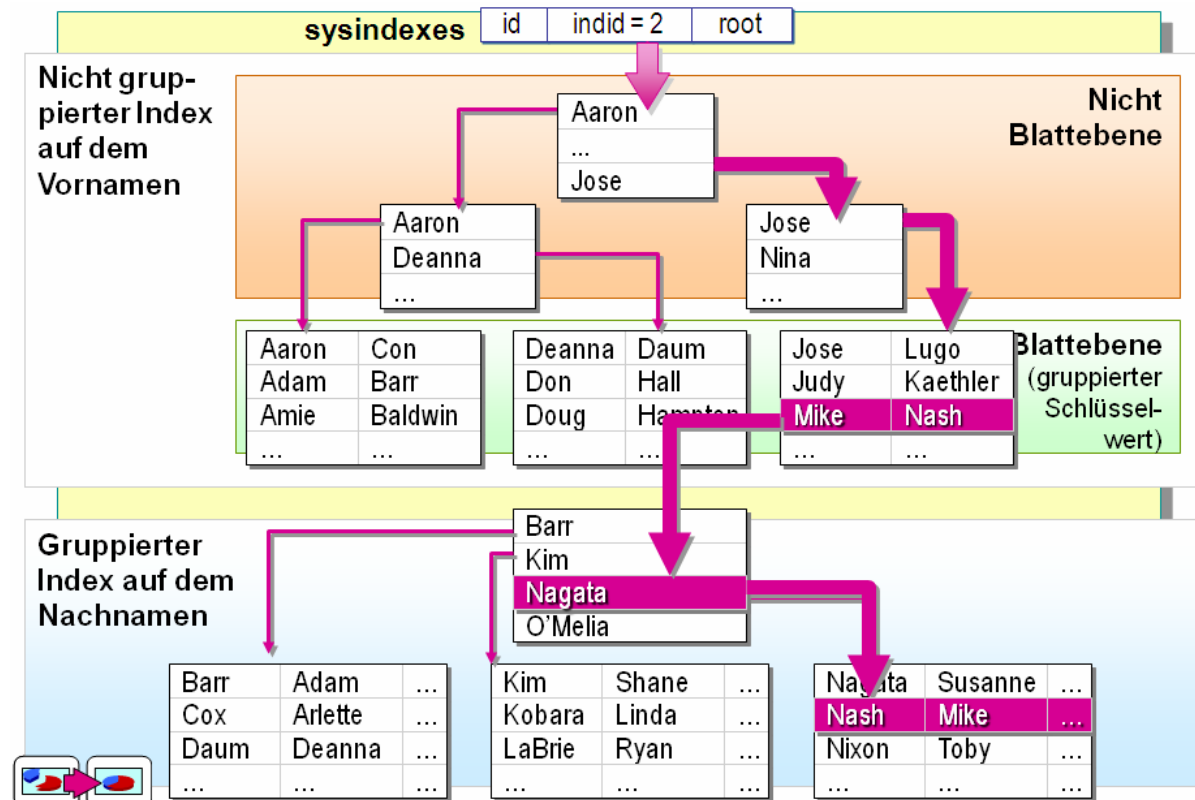
- Wenn es sich bei der Tabelle um einen Heap handelt (d. h. sie hat keinen gruppierten Index), entspricht der Zeilenlokator einem Zeiger auf die Zeile. Der Zeiger setzt sich aus dem Dateibezeichner (ID), der Seitennummer und der Nummer der Zeile auf der Seite zusammen. Der ganze Zeiger wird als Zeilen-ID (RID) bezeichnet.
- Wenn die Tabelle einen gruppierten Index besitzt oder der Index für eine indizierte Sicht erstellt wurde, ist der Zeilenlokator der Schlüssel des gruppierten Indexes für die Zeile. Wenn der gruppierte Index kein eindeutiger Index ist, fügt SQL Server 2005 zu den Schlüsseln, die mehrfach vorkommen, einen intern generierten Wert, `uniqueifier`, hinzu, damit die Schlüssel eindeutig werden. Dieser aus vier Bytes bestehende Wert ist für Benutzer nicht sichtbar. Er wird nur hinzugefügt, wenn der gruppierte Schlüssel eindeutig sein muss, um in nicht gruppierten Indizes verwendet werden zu können. SQL Server ruft die Datenzeilen ab, indem der gruppierte Index mithilfe des Schlüssels des gruppierten Indexes durchsucht wird, der in der Blattzeile des nicht gruppierten Indexes gespeichert ist.

Nicht gruppierte Indizes besitzen eine Zeile `index_id > 0` in `sys.partitions` für jede von dem Index verwendete Partition. Standardmäßig besitzen nicht gruppierte Indizes nur eine Partition. Wenn ein nicht gruppierter Index mehrere Partitionen besitzt, weist jede Partition eine B-Baumstruktur auf, die die Indexzeilen der entsprechenden Partition enthält. Wenn ein nicht gruppierter Index beispielsweise vier Partitionen besitzt, gibt es vier B-Baumstrukturen – jeweils eine in jeder Partition.

Abhängig von den Datentypen des nicht gruppierten Indexes erhält jede Struktur mindestens eine Zuordnungseinheit, in der die Daten einer bestimmten Partition gespeichert und verwaltet werden. Jeder nicht gruppierte Index besitzt also eine `IN_ROW_DATA`-Zuordnungseinheit pro Partition, die die B-Baumstrukturseiten des Indexes speichert. Außerdem besitzt der nicht gruppierte Index eine `LOB_DATA`-Zuordnungseinheit, wenn er LOB-Spalten (Large Object) enthält. Endlich besitzt er eine `ROW_OVERFLOW_DATA`-Zuordnungseinheit pro Partition, wenn er Spalten variabler Länge enthält, die das Zeilengrößenlimit von 8.060 Byte überschreiten. Weitere Informationen zu Zuordnungseinheiten finden Sie unter Organisationsstruktur von Tabellen und Indizes. Die Seitenaufstellungen für den B-Baum sind durch `root_page`-Zeiger in der `sys.system_internals_allocation_units`-Systemansicht verankert.



Suche mit nicht gruppiertem Index auf der Basis eines gruppierten Indexes:



Ab SQL Server 2005 besteht die Möglichkeit, während der Indexerstellung die Tabellen online zu halten.

```
CREATE INDEX ix_Employee_ManagerID on HumanResources.Employee (ManagerID)
WITH (ONLINE=ON,MAXDOP=1)
```

„Covered Query“: Abfrage, bei der alle Spalten Teil eines Index sind.

Mit der INCLUDE-Funktion können nun auch Spalten aufgenommen werden, die nicht Teil des Indexes sind:

```
CREATE INDEX ix_AddressDetails on Contact.Address (AddressID)
INCLUDE (AddressLine1, AddressLine2)
```

3.5 Gefilterte Indizes

Haben sich auf Grund der "Sparse Columns" entwickelt.

Klassische nonclustered indizes decken ALLE Datenzeilen ab.

Gefilterte Indizes decken nur die Zeilen ab, die einem "Prädikat" entsprechen (NOT NULL), sind damit kleiner und schneller, gerade wenn "Sparse Columns" indiziert werden (aber auch für alle anderen Spalten verwendbar).

Man könnte einen Index definieren für Produkte, die als "nicht ausgelaufen" gekennzeichnet sind.

3.6 Volltextsuche

SQL Server 2008 stellt Anwendungen und Benutzern die nötige Funktionalität bereit, um Volltextabfragen in zeichenbasierten Daten in SQL Server-Tabellen ausführen zu können. Bevor Volltextabfragen für eine bestimmte Tabelle möglich sind, muss der Datenbankadministrator einen Volltextindex für die Tabelle erstellen. Der Volltextindex umfasst eine oder mehrere zeichenbasierte Spalten der Tabelle. Diese Spalten können jeden der folgenden Datentypen aufweisen: char, varchar, nchar, nvarchar, text, ntext, image, xml, varbinary oder varbinary(max). Jeder Volltextindex indiziert eine oder mehrere Spalten aus der Basistabelle. Für jede Spalte kann hierbei eine eigene Sprache festgelegt sein.

Die Volltextsuche basierte bis SQL Server 2005 auf MS Search-Dienst und ist nun Teil der DB Engine.

Backup und Restore gehen ohne Einschränkungen mit dieser Suche!!

Beispiel:

```
SELECT product_id FROM products WHERE CONTAINS(product_description, "Snap Happy
100EZ" OR FORMSOF(THESAURUS,'Snap Happy') OR '100EZ') AND product_cost<200
```

4 Einschränkungen (Constraints)

4.1 Spalten- und Tabelleneinschränkungen

Einschränkungen können Spalten- oder Tabelleneinschränkungen sein:

- Eine Spalteneinschränkung wird als Teil einer Spaltendefinition angegeben und gilt nur für diese Spalte.
- Eine Tabelleneinschränkung wird unabhängig von einer Spaltendefinition deklariert und kann für mehr als eine Spalte in einer Tabelle gelten. Tabelleneinschränkungen müssen verwendet werden, wenn mehr als eine Spalte in eine Einschränkung eingeschlossen werden muss.

4.2 Primary Key-Einschränkungen, Default-Einschränkungen

```
use Auftrag;
create table dbo.tArtikel
( ArtNr int identity(1,1),
  ArtBez nvarchar(50) NOT NULL,
  Einzelpreis money NOT NULL constraint DF_tArtikel_Einzelpreis default (0.0),
  constraint PK_tArtikel_ArtNr primary key nonclustered
  (ArtNr ASC) with (ignore_dup_key = off)
);
```

Wenn in einer Tabelle z. B. zwei oder mehr Spalten für den Primärschlüssel verwendet werden, müssen Sie eine Tabelleneinschränkung verwenden, um beide Spalten in den Primärschlüssel einzuschließen. Stellen Sie sich eine Tabelle vor, die Ereignisse aufzeichnet, die für einen Computer in einer Fabrik eintreten. Nehmen Sie weiterhin an, dass unterschiedliche Ereignistypen gleichzeitig eintreten können, dass jedoch nie zwei Ereignisse desselben Typs gleichzeitig eintreten. Dieser Sachverhalt kann in der Tabelle erzwungen werden, indem Sie die **type**- und die **time**-Spalte in einen Primärschlüssel einschließen, der zwei Spalten umfasst.

```
CREATE TABLE factory_process
( event_type int,
  event_time datetime,
  event_site char(50),
  event_desc char(1024),
  CONSTRAINT event_key PRIMARY KEY (event_type, event_time) )

CREATE TABLE tPLZ
(
  PLZ char(5) not NULL,
  Ort varchar(50) not NULL
  CONSTRAINT PK_tPLZ PRIMARY KEY (PLZ, Ort)
);
```

4.3 Foreign Key-Constraints

```
alter table dbo.tAuftrag
  add MitarbeiterNr int null;

alter table dbo.tAuftrag
  with check -- vorhandene Datensätze werden überprüft
  add constraint FK_MitarbeiterNr_tMitarbeiter
  foreign key (MitarbeiterNr)
```

```
references dbo.tMitarbeiter(MitarbeiterNr);

alter table dbo.tAuftrag -- beginnen Sie immer mit der Detailtabelle,
                        -- das ist die Tabelle, die den Fremdschlüssel enthält
with check             -- vorhandene Datensätze werden überprüft
add constraint FK_KdNr_tKunden
foreign key (KdNr)     -- Fremdschlüssel
references dbo.tKunden(KdNr) -- Bezug auf Primärschlüssel der anderen Tabelle
on update cascade;   -- Kaskadierungsoptionen
```

5 Sichten (Views)

Man sollte nie direkt mit den Tabellen, sondern immer mit Abfragen arbeiten.

```
use Auftrag
go

create view dbo.Auftragssicht
as
select
    tAuftragsdetails.AuftrNr,
    tAuftrag.Datum,
    tAuftrag.KdNr,
    tKunden.Vorname,
    tKunden.Nachname,
    tAuftragsdetails.ArtNr,
    tArtikel.ArtBez,
    tAuftragsdetails.Anzahl,
    tArtikel.Einzelpreis,
    Anzahl * Einzelpreis AS Zeilenpreis
from
    tKunden as k inner join tAuftrag as a
        on k.KdNr = a.KdNr
    inner join tAuftragsdetails as d
        on a.AuftrNr = d.AuftragsNr
    inner join tArtikel as art
        on d.ArtNr = art.ArtNr
where
    d.AuftrNr = 1;
```

WICHTIG!

Niemals verknüpfte Primärschlüsselfelder in der Abfrage verwenden!

Verknüpfte Felder in der Detailtabelle MÜSSEN in der Abfrage enthalten sein!