GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Übung Datenbanksysteme 2 (M-DB2)

Todor Ivanov

Timo Eichhorn

Update Folien – Übung 1

Übung:
Freitag 10:00 – 12:00 Uhr, Hörsaaltrakt Bockenheim – H I

Voraussetzung:

Erfolgreicher Abschluss des Moduls B-PRG oder des Moduls B-HW oder der beiden Module B-MOD und B-DS.

Nützliche Vorkenntnisse: Weiterführende Kenntnisse in Betriebssystemen, Programmiersprachen und Mathematik. Inhalte des Moduls B-DB1.

Übersicht (wird im Laufe der Veranstaltung angepasst!):

Tag	Datum	Inhalt	Material
MI	17/04/2019	Einführung	Organisatorisches Einfuehrung
FR	19/04/2019	Karfreitag	
MI	24/04/2019	Hashorganisation	Hashorganization
FR	26/04/2019	Übung 1	Üb1-Hash / Üb1_folien_V2
MI	01/05/2019	Feiertag 01. Mai	
FR	03/05/2019	Übung 2	Üb2-Hash

Update der Folien!

Link: http://www.bigdata.uni-frankfurt.de/database-systems-2-ss-2019/

Hash(funktion) allgemein

- Hashfunktion $h:K \rightarrow S$ ist eine Funktion, wobei die Mächtigkeit von S kleiner oder gleich von K ist ($|K| \geq |S|$)
- Anwendungen allgemein
  - Passwörter / Kryptographie
  - Prüfsummen
  - …
  - Datenbanken

Hash in Datenbanken

Verwendung als:

- Indexstruktur
- Join Implementierung (später in der Vorlesung)
- Partitionierung

Join-Implementierungen

- In der Regel MSSQL:
  - Unsortiert und größere Anzahl → **Hash Join**
  - Eine Tabelle klein, andere groß und unsortiert → Nested Loops Join
  - Join Attribute liegen sortiert vor → Merge Join

ACHTUNG: Sehr vereinfachte Darstellung der MS SQL Umsetzung, weitere Details im Laufe der Vorlesung!

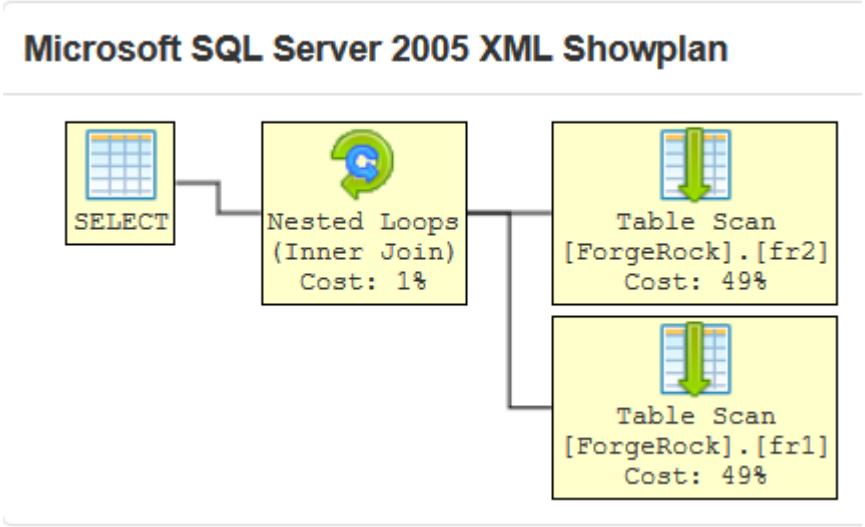
Links:

https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins?view=sql-server-2017

http://www.sqlfiddle.com/#!18

Nested Loops

```sql
-- Nested Loop Join
SELECT
  *
FROM
  ForgeRock fr1, ForgeRock fr2
;
```

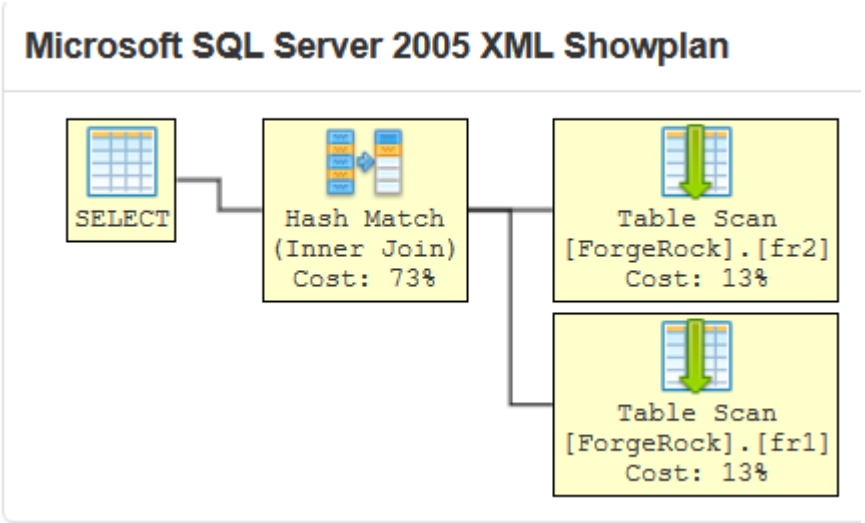


Links:

http://www.sqlfiddle.com/#!18

Hash Join

```
 8  -- Hash Join
 9  SELECT
10    *
11  FROM
12    ForgeRock fr1, ForgeRock fr2
13  WHERE
14    fr1.productName = fr2.productName
15  ;
```



Links:

http://www.sqlfiddle.com/#!18

Hash als Partitionierung

HASH Partitions

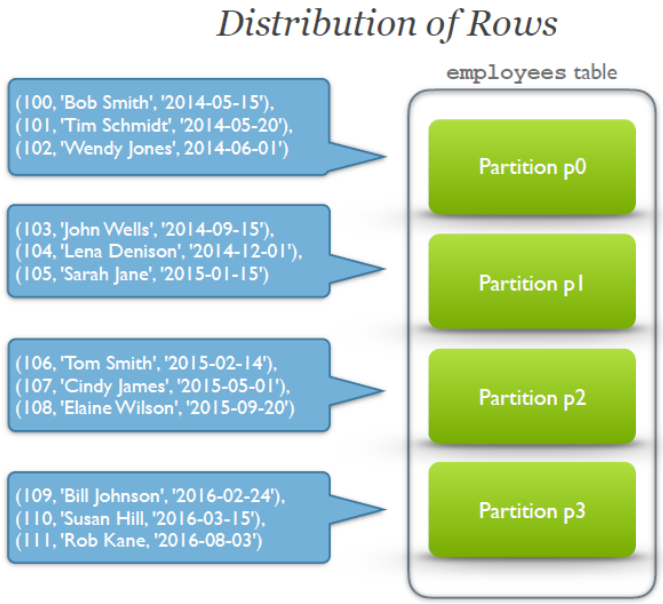
Partition Determined by a Hash from a Given Expression

```
CREATE TABLE employees (
  id INT AUTO_INCREMENT KEY,
  name VARCHAR(20), dept_id INT,
  join_date DATE)
ENGINE = MyISAM
PARTITION BY HASH(id) PARTITIONS 4;
```

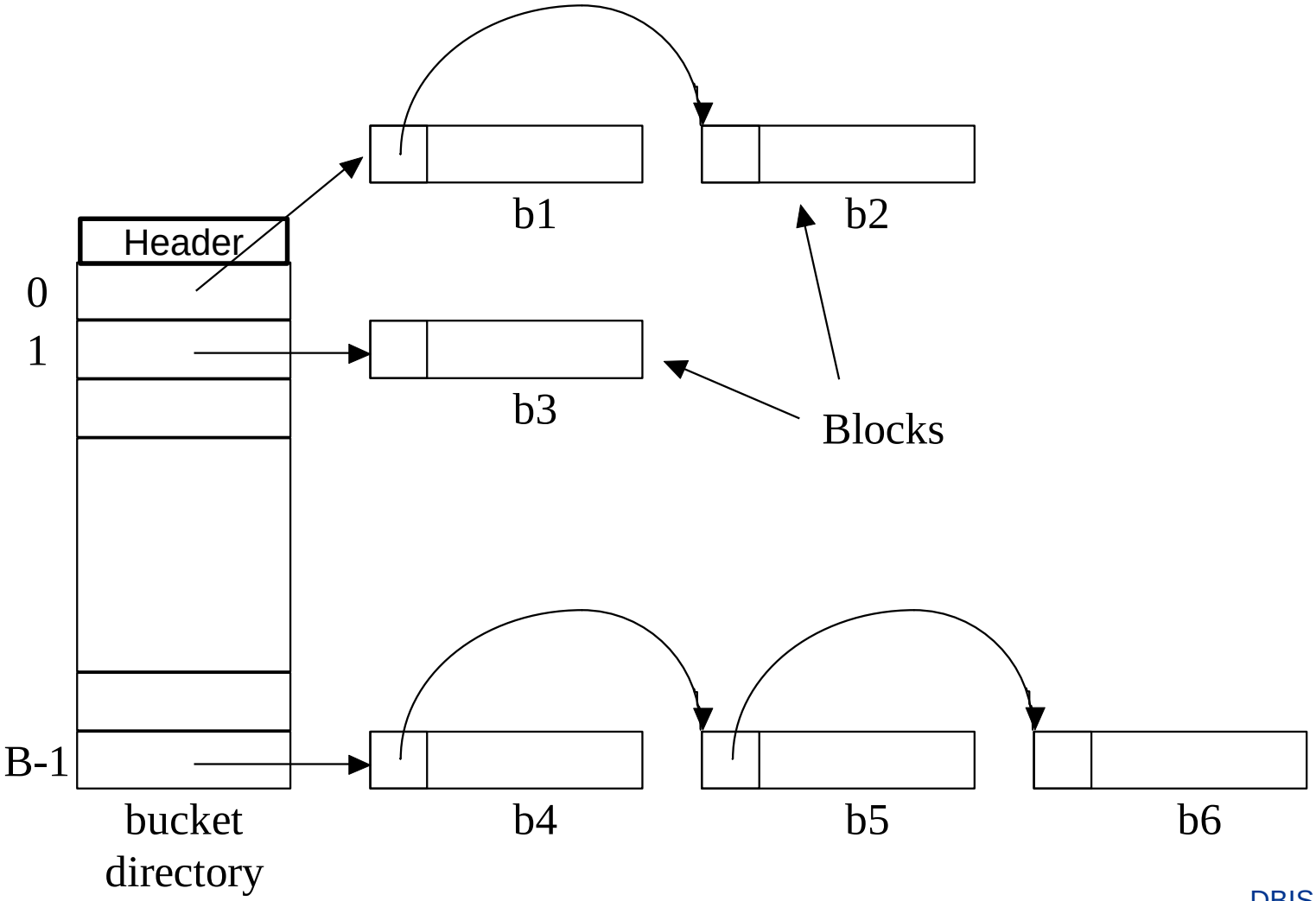
Partitions contain Same Number of Rows

Use **COALESCE** to Reduce Number of Partitions

```
ALTER TABLE employees
COALESCE PARTITION 1;
```



Hash als Indexstruktur

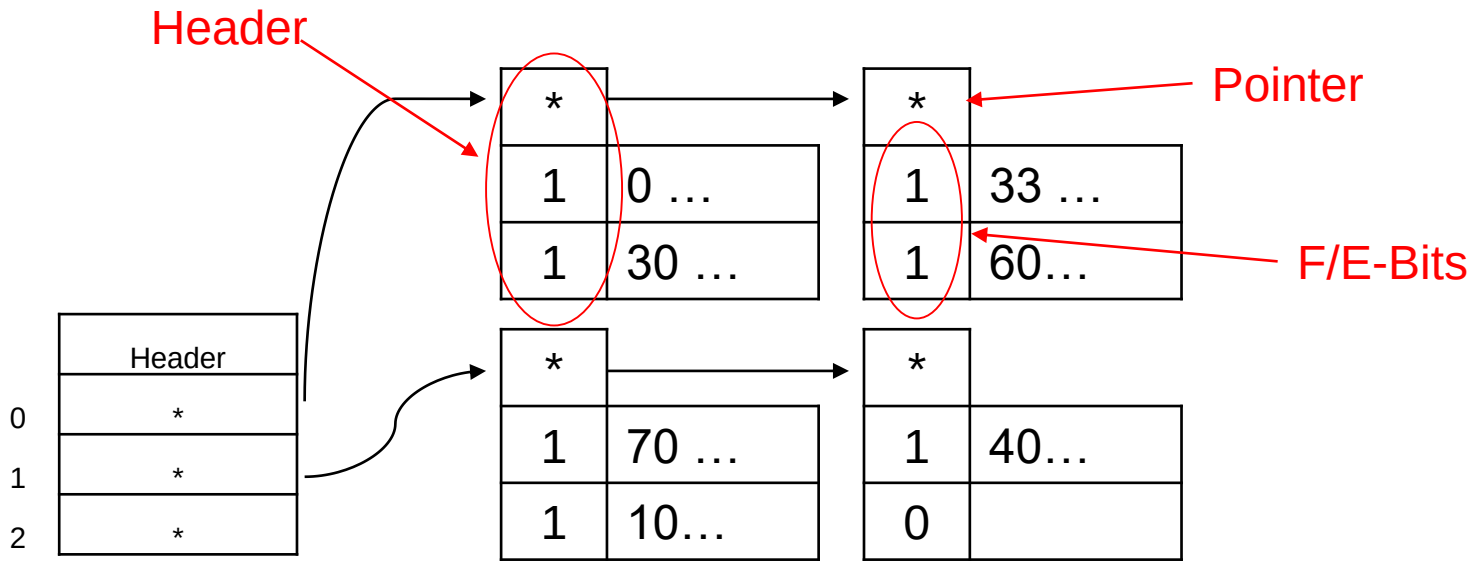


Vorgehensweise Aufgabe 1

- Wie viele Blöcke werden für das Bucket Directory benötigt?
- Wie viel Speicherplatz benötigt ein Datensatz?
- Wie viele Datensätze können in einem Block hinterlegt werden?
- Datensätze der Reihenfolge entsprechend einfügen!
- Header in den Blöcken beachten!

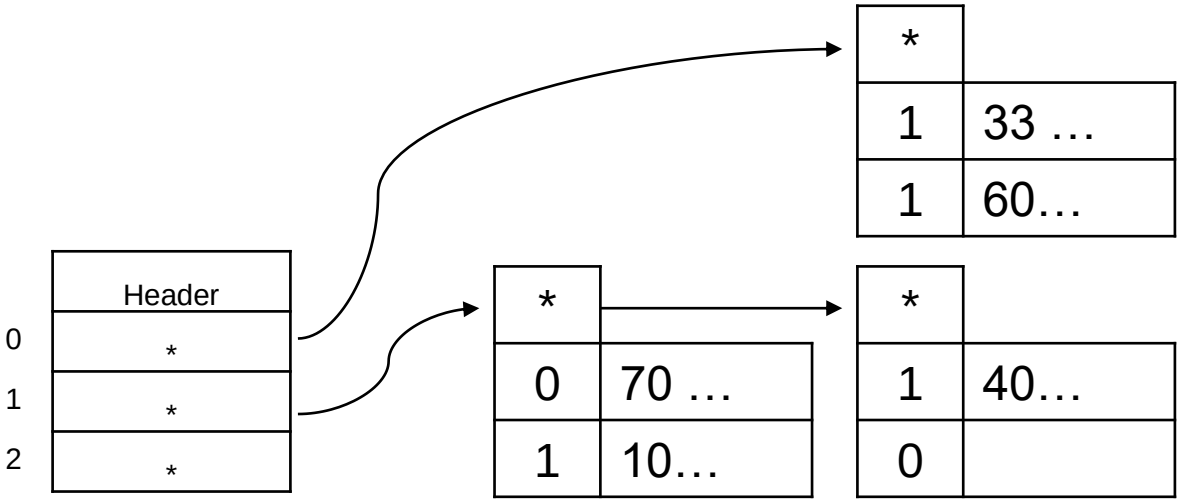
Aufgabe 1 - Lösungsidee

a) Erstellen und zeichnen Sie die Hashstruktur mit h(v) = v mod 3 über das Schlüsselattribut ID, wenn die Daten von oben nach unten eingegeben werden (inkl. Headerinformationen).



Aufgabe 1 - Lösungsidee

b) Löschen Sie die Datensätze 70, 0 und 30 und zeichnen Sie die Hashstruktur erneut auf.



Aufgabe 1 - Lösungsidee

c) Beschreiben Sie was sich ändern würde, wenn statt char(20) varchar(20) genutzt würde (für beide Attribute)?

- Char(20) reserviert fest 20 Bytes im Block

Header	ID	Attribut1	Attribut2	ID	Attribut1	Attribut2	Unben. Rest

- Varchar(xx) nutzt nur die benötigten Bytes (hier: max. 20 für Daten)

Folgen?

Aufgabe 1 c) - Lösungsidee

Definition aus der Vorlesung:

Interpretation von Rohdaten (raw data)

Datensätze im Rohformat (keine weiterführende Speicherstrategien) unterliegen folgenden Bedingungen:

- Festes Format.
- Felder in derselben Reihenfolge.
- Jedes Feld hat eine feste Anzahl von Bytes.
- Jedes Feld hat einen festen Datentyp.

Aufgabe 1 c) - Lösungsidee

- Varchar(xx) nutzt nur die benötigten Bytes (hier: max. 20 für Daten)

Header	33	A4*$	B4*$	§	60	A7*$	B7*$	§	Unbenutzter Rest

Ein Lösungsansatz:

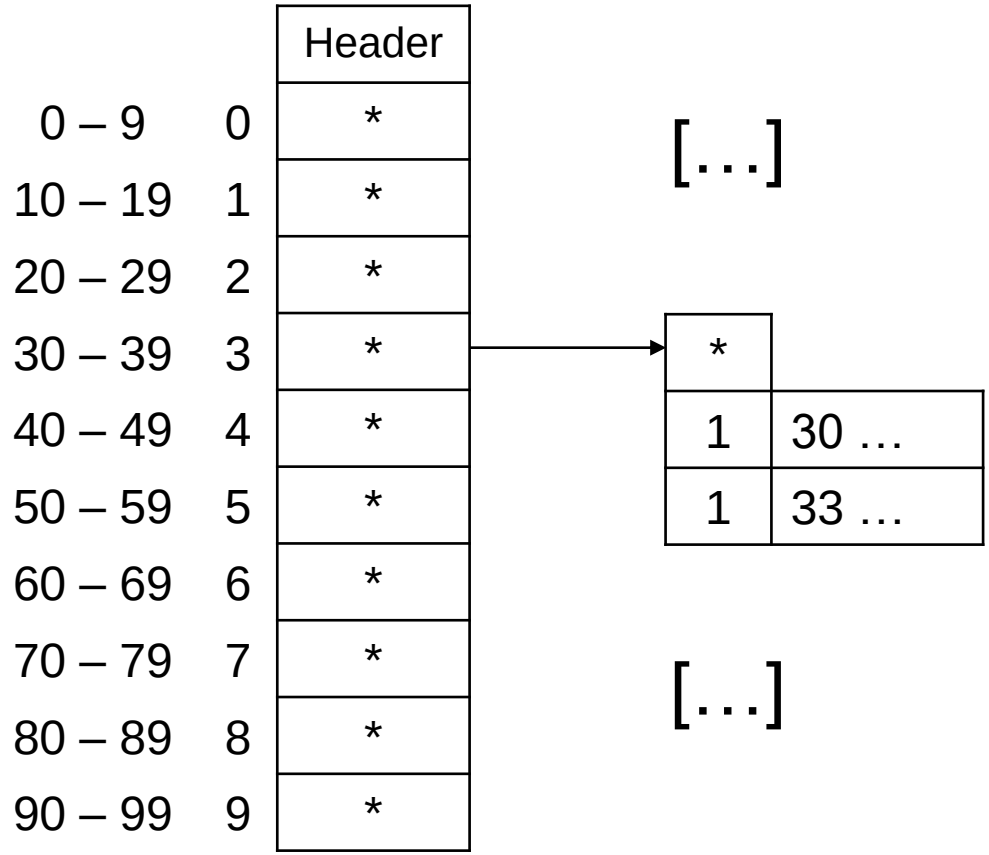
- Trennzeichen für Attribut notwendig (da keine feste Länge): $
- Pointer für jedes Attribut, falls dieses verlängert wird und kein zusätzlicher Speicher im Block vorhanden: *
- Trennzeichen für Datensatz, da keine feste Länge für DS definiert: §

→ 13 Byte pro DS (aus den Beispieldaten)

→ Nun 7 DS im Block möglich

Aufgabe 1 - Lösungsidee

d) Vorausgesetzt der Wertebereich der ID-Werte ist beschränkte auf [0-99]. Geben Sie für diese Situation eine mögliche ordnungserhaltende Hashfunktion an und zeichnen Sie die Hashstruktur, wenn die Daten von oben nach unten eingegeben werden.



Aufgabe 2

a) Wie kann die Hashorganisation verwendet werden, wenn das Schlüsselattribut nicht als Integer definiert ist?

b) Mit welchen Nachteilen muss bei der Verwendung einer Hashorganisation gerechnet werden?

c) Welche Optimierungsverfahren gibt es, wenn die Hashorganisation nicht mehr in angemessener Zeit Ergebnisse zurückliefert?

Aufgabe 2 - Lösungsidee

a) Wie kann die Hashorganisation verwendet werden, wenn das Schlüsselattribut nicht als Integer definiert ist?

→ Umrechnen des Attributwerts in ein Integerwert:

z.B.: Vorgehen 1) aus der Vorlesung (Ullman):

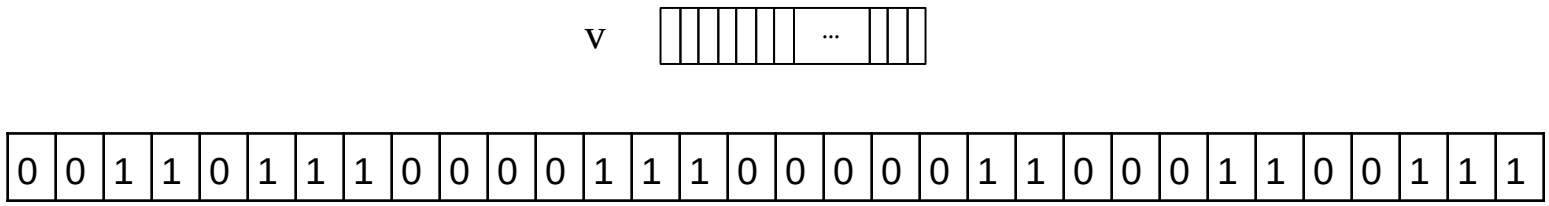
v(Elsamosaurus) = 12

→ Anzahl der verwendeten Zeichen!

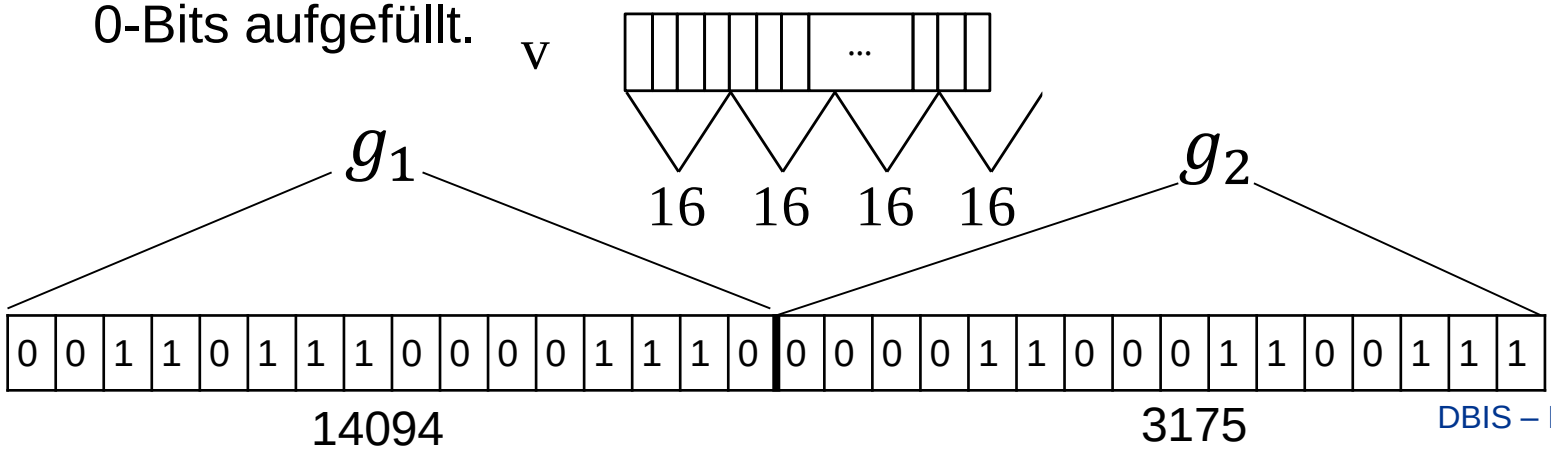
Aufgabe 2 a) - Lösungsidee

Vorgehen 2:

1. Wir nehmen an der Schlüssel v liegt in Form von n Bits vor.



2. Die Bits werden in Gruppen g_i zu jeweils 16 Bit zusammengefasst. Die letzte Gruppe wird, falls nötig, mit 0-Bits aufgefüllt.



Aufgabe 2 a) - Lösungsidee

Vorgehen 2:

3. Die Gruppen g_i werden jetzt als Integer-Zahlen behandelt und addiert.

$$S = \sum g_i$$

S = 14094 + 3175 = 17269

4. Die Summe wird durch die Anzahl der Buckets dividiert und der Rest wird als Bucket-Nummer des Satzes genommen.

r = S *mod* #Buckets

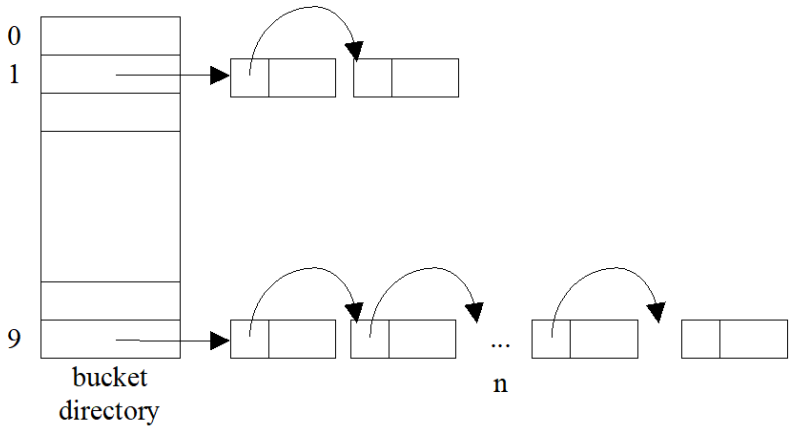
z.B.: Für eine Bucketzahl von 100 → r = 17269 mod 100 = **69**

Aufgabe 2 - Lösungsidee

b) Mit welchen Nachteilen muss bei der Verwendung einer Hashorganisation gerechnet werden?

- Schlechte Performance bei Range-Queries
- Schlechte Performance bei Abfragen auf Nicht-Schlüssel-Attribute
- Perfomance abhängig sowohl von Hashfunktion als auch Daten

→ Dies kann zur „Entartung“ einer Hashtabelle führen



Weitere: Anzahl der Buckets zu gering und deswegen sehr viele Blöcke innerhalb dieser!

Aufgabe 2 - Lösungsidee

c) Welche Optimierungsverfahren gibt es, wenn die Hashorganisation nicht mehr in angemessener Zeit Ergebnisse zurückliefert?

- Analyse der vorhandenen Daten und anpassen der Hashfunktion
  - Erhöhen der Bucketanzahl
  - Wahl der Hashfunktion, damit eine gleichmäßige Verteilung gewährleistet wird
- Mit welcher Art von Queries wird auf die Daten zugegriffen?

Folgen bei Pinned Records?