

Praktikum

“Datenbanksystementwicklung”

– System J –

Knut Stolze

stolze@informatik.uni-jena.de

Agenda

- Einführung
- Systemarchitektur
- Prozessmodell
- Komponenten im Detail
- Organisation
- Erfahrungen

Einführung – System J

- Vollständiges Datenbankmanagementsystem (DBMS)
- Entwickelt im WS 2004/2005
 - 13 Studenten (7./8. Fachsemester)
- Entwicklungsprozess stark praxisorientiert
 - Angelehnt an DB2-Prozess
- Implementierung in C/C++
- SQL Syntax stark vereinfacht
 - Details beim Compiler mit erwähnt

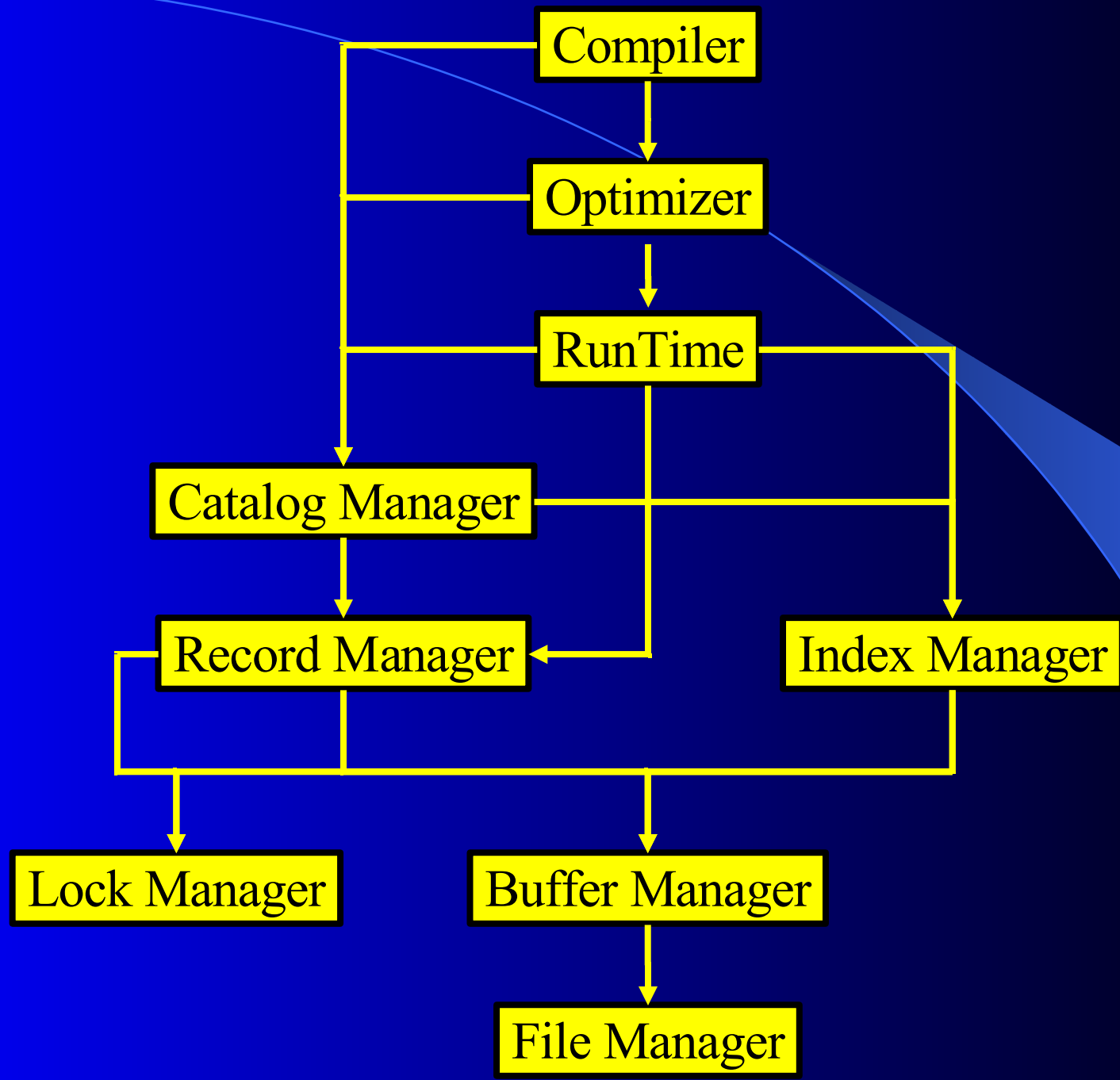
Motivation & Zielsetzung

- Ergänzung der Vorlesung DBS 2, insbesondere des Teils zur „Implementierung von Datenbanksystemen“
- Arbeiten in einem (kleinen) Team, und alle Teams sind zusammen an einem größeren Projekt beteiligt
 - Gemeinsam ein Ziel (lauffähiges System) zu erreichen
 - Kommunikation mit anderen Teams um Schnittstellen zu definieren und Probleme zu diskutieren/lösen
 - Organisation innerhalb eines Teams ist nicht festgelegt

Motivation & Zielsetzung (2)

- Selbständiges Einarbeiten in neue Konzepte und Programmiersprachen
- Einschränkungen und Abhängigkeiten von heutigen Datenbanksystemen von der internen Programmierung zu verstehen
- Umgang mit und Implementierung für verschiedenen Betriebssysteme
- Anwendung von „version control“ Systemen zur Verwaltung von Quelltexten

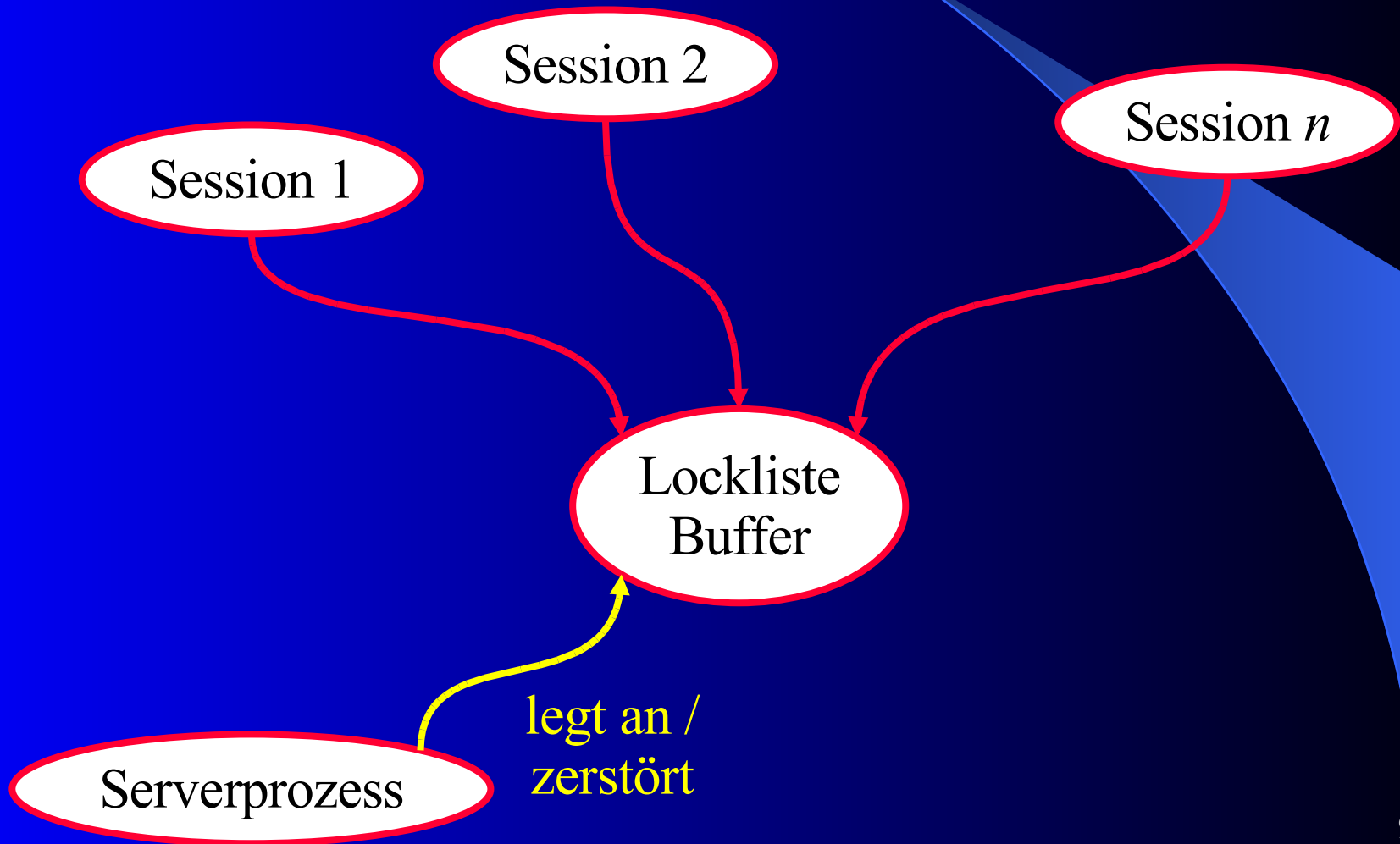
Systemarchitektur



Betriebssystemabstraktion (OSS)
Fehlerbehandlung (Error)

Prozessmodell

Überblick



Internas von Sessions

Session i

Je 1 Instanz von:

- Compiler
- Optimizer
- Catalog Manager
- RunTime
- Record Manager
- Index Manager
- Lock Manager
- Buffer Manager
- Memory Manager
- File Manager

Erläuterungen

- Jede Transaktion/Session wird in genau einem Prozess abgearbeitet
 - Kein IPC (inter-process-communication)
 - Jeder Prozess hat eigene Instanzen von den verschiedenen Komponenten (Managern)
- Buffer & Lockliste liegen im Shared Memory
 - Zugriff von mehreren Transaktionen muss mittels Latches synchronisiert werden
 - Serverprozess ist nur für das Anlegen der Shared Memory Segmente zuständig und beendet sich anschließend wieder

– Compiler –

Aufgaben des Compilers

- Parsen der SQL-Anweisung
- Aufbau des internen Zugriffsplan (Baum)
- Ermitteln der internen Spalten-, Tabellen- und Index-IDs
- Zuordnung der Spalten zu den Tabellen (SELECT)

Compiler

- Klasse: DbjCompiler
- Datenstrukturen
 - Eingabe: String mit SQL Anweisung, DbjTable, DbjIndex
 - Ausgabe: DbjAccessPlan
- Verwendet
 - Katalog Manager zur Validierung des Plans

Unterstützte SQL-Syntax (1)

```
>>--CREATE TABLE--table-name-- (----->
      .--,-----.
      √                                     |
>-----column-name--| data-type |--+-----+--+----->
                                     '---NOT NULL---'
>-----+-----+-----)-----<<
      '---,---PRIMARY KEY--(column-name--)'---
```

```
>>--DROP TABLE--table-name-----<<
```

Unterstützte SQL-Syntax (2)

data-type:

```
| --+--INTEGER-----+-----|  
  +--INT-----+  
  ' --VARCHAR-- (--length--) -- '
```

```
  .--WORK--.
```

```
>>--COMMIT--+-----+-----<<
```

```
  .--WORK--.
```

```
>>--ROLLBACK--+-----+-----<<
```

Unterstützte SQL-Syntax (3)

```
>>--CREATE--+-----+--INDEX--index-name--ON----->
      '--UNIQUE--'
                                     .--OF TYPE BTREE--.
>--table-name--(--column-name--)--+-----+--<<
                                     '--OF TYPE HASH---'
```



```
>>--DROP INDEX--index-name-----<<
```

Unterstützte SQL-Syntax (4)

```
>>--INSERT--INTO--table-name--VALUES----->
```

```
.-- ,-----.  
|      .-- ,----- |  
V      V      |      |
```

```
>-----(value+-)-----<<
```

```
.--AS--corr-name--.
```

```
>>--DELETE--FROM--table-name----->
```

```
>-----<<
```

```
'--| where-clause |--'
```

Unterstützte SQL-Syntax (5)

```

      .-- , ----- .
      |
      V          .--AS--new-name--. |
>>--SELECT--+---column-name--+-----+--+>
      '---*-----'
      .-- , ----- .
      V          .--AS--corr-name--. |
>---FROM-----table-name--+-----+--+>
>--+-----+-----<<
      '--| where-clause |--'
```

Unterstützte SQL-Syntax (6)

where-clause:

```
|--WHERE--| predicate |-----|
```

predicate:

```
|--+--| expression |--| operation |--| expression |-----+--|  
+--| expression |--IS--+-----+--NULL-----+  
|           |--NOT--|  
+--(--| predicate |--)-----+  
+--NOT--(--| predicate |--)-----+  
+--| expression |--+-----+--LIKE--REGEX--value-----+  
|           |--NOT--|  
+--| expr |--+-----+--BETWEEN--| expr |--AND--| expr |--+  
|           |--NOT--|  
'--| predicate |--+--AND--+--| predicate |-----|  
|           |--OR---|
```

Unterstützte SQL-Syntax (7)

operation:

```
| --+-- "="-----+-----|  
  +-- "<"----+  
  +-- "<="--+  
  +-- ">"----+  
  +-- ">="--+  
  ' -- "<>"-- '
```

expression:

```
  .-- correlation-name--"."--.  
| --+--+-----+-----+-- column-name-----+-----|  
  ' -- value-----+-----'
```

Interner Zugriffsplan – Datenstruktur –

- DbjAccessPlan
 - DbjAccessPlanTable (mit Korrelationsname)
 - DbjAccessPlanColumn (zusätzlich mit neuem Spaltenname bei umbenannten Spalten)
- Aufgebaut von Parser
- Parser wurde mittels Bison generiert
 - Schlüsselworte dürfen nicht als Spalten-, Tabellen- oder Indexnamen verwendet werden
 - Ungültig ist also: `SELECT SELECT SELECT, FROM FROM FROM FROM WHERE WHERE = WHERE`
- Lexer per Hand geschrieben

– Optimizer –

Übersicht

- Erhält vom Compiler geparsten und validierten Ausführungsplan
- Regelbasierte Optimierung
 - Keine kostenbasierte Optimierung!
- Optimierungen nur für SELECT- und DELETE – Anweisungen
 - alle anderen Anweisungen werden unverändert durchgereicht

Optimierungen

1. Negationen (NOT) wird entfernt; Klammerungen werden aufgelöst
2. Evaluieren konstanter Prädikate
3. Selektionen vor Joins wenn Prädikat sich ausschließlich auf *eine* Tabelle bezieht
4. Nutze Indexscans wenn entsprechendes Prädikat auf Tabelle existiert
 - Erstbester Index wird verwendet
5. Sortiere Tabellen aufsteigend nach Tupel-Anzahl (Tabelle mit geringster Anzahl ist innere Tabelle im NLJOIN)

Internas

- Klasse: DbjOptimizer
- Datenstruktur: DbjAccessPlan, DbjIndex, DbjTable
- Verwendet
 - Katalog Manager um Indexe und Tupel-Anzahl der Tabellen zu ermitteln
 - Eingebaute Regeln zur Optimierung (keep it simple!)

– Catalog Manager –

Aufgaben

- Verwaltet Katalogtabellen und den Zugriff darauf
- Wird bei DDL modifiziert
- Kennt Struktur aller Tabellen
 - Definiert internen Aufbau von Tupeln
 - Kann Records (Byte-Stream) in Tupel umwandeln

Catalog

- **SYSTABLES** (
tableName VARCHAR(128),
tableId INTEGER,
columnCount INTEGER,
createTime VARCHAR(26),
tupleCount INTEGER)

- Index auf tableName
und tableId

- **SYSCOLUMNS** (
tableId INTEGER,
columnName VARCHAR(128),
columnId INTEGER,
dataType VARCHAR(128),
maxLength INTEGER,
nullable VARCHAR(1))

- Index auf columnName und tableId

- **SYSINDEXES** (
tableId INTEGER,
indexName VARCHAR(128),
indexId INTEGER,
type VARCHAR(5),
columnId INTEGER,
unique VARCHAR(1),
createTime VARCHAR(26))

- Index auf tableId,
indexName und indexId

Alle Index-Informationen der Systemtabellen liegen ebenfalls „fest verdrahtet“ vor!

Aufbau eines Records

Beispieltabelle:

Angest(PNr Integer not Null,
Name Varchar(128) not Null,
AbtNr Integer not Null
Anzahl_Kinder Integer)

4711	14	Weihnachtsmann	815	N	50
------	----	----------------	-----	---	----



4712	10	Osterhase	815	Y
------	----	-----------	-----	---



Internas

- Klasse: DbjCatalogManager
- Datenstrukturen:
 - Eingabe: DbjRecord
 - Ausgabe: TableId/IndexId/..., DbjTable, DbjIndex
- Verwendet
 - Record Manager um Kataloginformationen von den entsprechenden Tabellen zu erhalten
 - Eingebaute Definition der Katalogtabellen

– RunTime –

RunTime

- Aufgabe
 - Abarbeitung des Optimierten Zugriffsplanes
- Verwendet Iterator-Konzept

Iteratoren

- Record-Iterator & Index-Iterator
 - Geliefert von Record bzw. Index Mgr
- Tuple-Iteratoren
 - RecordTuple, IndexTuple, CrossProduct, Projection, Selection
- Einfache Berechnung der Ergebnismenge durch Verschachtelung der Iteratoren → Bäume
Iteratorenbäume
 - Recht einfach aus Zugriffsplan aufgebaut
 - Können leicht erweitert werden (Sort, HashJoin, ...)

Internas

- Klasse: DbjRunTime
- Datenstrukturen: DbjAccessPlan, DbjTuple, DbjRecord, DbjIndexKey
- Verwendet
 - DbjTable (von Katalog Manager) zur Interpretation der Records in Tupel
 - Record Manager um die Records zu erhalten
 - Index Manager um Indexzugriff durchzuführen
- Anmerkungen
 - Physische Daten werden *nicht* kopiert (zu teuer) → Referenzen

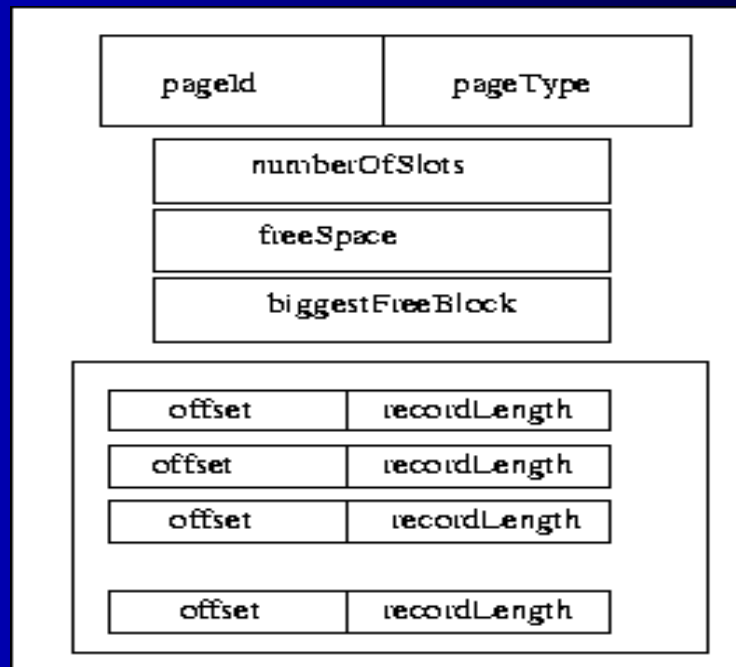
– Record Manager –

Aufgaben

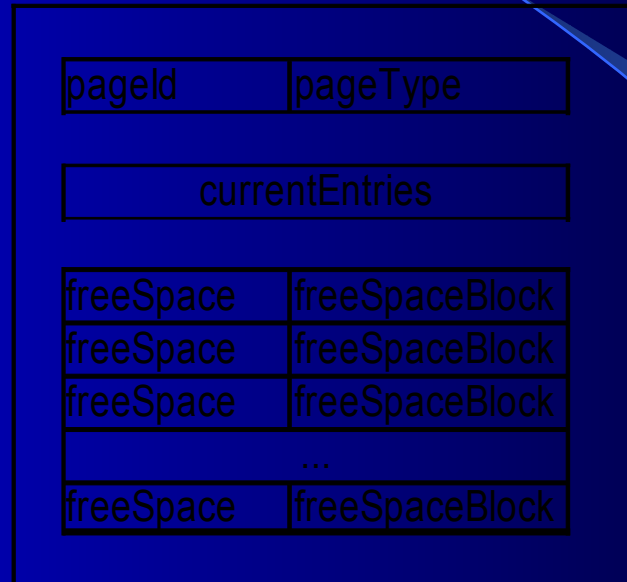
- Verwaltet Records auf Datenseiten
- Verwaltet Datenseiten im Segment
 - Freispeicherverwaltung
- Stellt Record-Iterator zur Verfügung

Datenseiten

- 255 Records pro Seite
- freeSpace, biggestFreeBlock jeweils genaue Angabe der verwendeten Bytes
- TID Konzept (tableId, pageId, slotId)



Freispeicherverzeichnis



- 2043 Einträge pro FSI-Seite bei 4kB Seitengröße
- freeSpace & freeSpaceBlock je 1 Byte – fassen 16 Byte auf Seite zusammen
- Seitennummern werden implizit (Array-Index) mitgeführt

Internas

- Klasse: DbjRecordManager
- Datenstrukturen: TupelId, DbjRecord, DbjPage
- Verwendet
 - Buffer Manager zum Zugriff auf die Seiten
 - Lock Manager um Datenseiten zu sperren
- Anmerkungen
 - Die Daten der Records werden aus dem Systempuffer in die DbjRecord Struktur kopiert
 - Seite kann schnell wieder freigegeben werden

– Index Manager –

Übersicht

- *Aufgabe*

*Anlegen und Verwalten von Indexten auf
Tabellenspalten*

Verwaltung der Index-Strukturen

Verwaltung der Seiten im Index-Segment

- *Arten von Indexten*

B Tree Index (B Baum)*

*Hash Index → aus Zeitmangel nicht
implementiert*

- *Datentypen*

Integer

Varchar mit 256 Zeichen Begrenzung

Physische Strukturen

- Knoten im B-Baum = Seite
 - Innere Knoten und Blätter
- Dynamisches Freispeicherverzeichnis (verkettet)

Innere Knoten

- Globaler Seitenheader (Seiten-ID, Seitentyp)
- Spezifischer Header
 - Erster linker Eintrag
 - Verweis auf Vater
 - Anzahl der Einträge im Knoten
- Array von Einträgen (Entry)
 - Schlüsselwert
 - Verweis auf Sohn dessen Schlüssel größer sind

Blattknoten

- Globaler Seitenheader (Seiten-ID, Seitentyp)
- Spezifischer Header
 - Verweis auf Vater
 - Verweis auf rechter und linker Bruder
 - Anzahl der Einträge im Knoten
- Array von Einträgen (Entry)
 - Schlüsselwert
 - Teil einer Tupel-ID
 - nur Seiten-ID und Slot-ID, keine Segment-ID

Internas

- Klasse: DbjIndexManager
- Datenstrukturen: DbjKey, TupelId, DbjPage
- Verwendet
 - Buffer Manager zum Zugriff auf die Seiten
 - Lock Manager um Indexseiten zu sperren
- Anmerkungen
 - Verwendet **nicht** Record Manager; Seiteninhalte werden selbst verwaltet!

– Lock Manager –

Übersicht

- Aufgaben

Verwaltung von logischen Sperren auf Seiten

- *Nur Schreib- und Lese-Sperren*

- *Deadlock-Erkennung*

Timeout

Datenstruktur

- Im Shared Memory
- Header
 - Statusinformationen
 - Hash-Tabelle → Einstieg in kürzere Listen mit Lock-Einträgen
- Lock-Einträge
 - Segment-ID + Page-ID
 - Sperrtyp
 - Transaktions-ID (= Prozess-ID)
 - Verkettung mit nächstem/vorherigen Lock-Eintrag

Internas

- Klasse: DbjLockManager
- Datenstrukturen: SegmentId, PageId

– Buffer Manager –

Aufgaben

- Stellt benötigte Daten- und Index-Seiten bereit
- Schreibt beim COMMIT alle geänderten Seiten auf Platte
- Verdrängen von Seiten bei Bedarf

Datenstruktur

- Im Shared Memory
- Verkettete LRU-Liste
 - Array mit 1 Eintrag pro Seite/Slot im Puffer
- Hash-Tabelle
 - Zum schnellen Auffinden einer Seite
 - Verkettete Liste von Seiten
 - Array mit 1 Eintrag pro Seite/Slot im Puffer
- Array von Slots für Seiten (Daten)

Internas

- Klasse: DbjBufferManager
- Datenstrukturen: DbjPage, “char *” (Blöcke)
- Verwendet
 - File Manager um die Blöcke bereitzustellen
 - Shared memory für den Puffer
- Anmerkungen
 - Implementiert Force/Steal;
ausser für geänderte Seiten – dort erfolgt No-Steal

– File Manager –

Überblick

- Aufgabe
 - Verwaltung der geöffneten Dateien
 - Speicherung der Blöcke in Dateien
 - Pro Operation 1 Block gelesen/geschrieben
- Konventionen
 - 1 Block = 1 Seite
 - 1 Datei = 1 Segment

Internas

- Klasse: DbjFileManager
- Datenstrukturen: SegmentId, Dateien, “char *” (Blöcke)
- Anmerkungen
 - Verwendet keine anderen Komponenten; basiert nur auf Betriebssystem

– Sonstige Dienste –

Fehlerbehandlung

- Zentrale Klasse zur Fehlerverwaltung
 - Würde leicht NLS unterstützen können
- Klasse: DbjError
- Datenstrukturen: DbjComponent, DbjErrorCode
- Anmerkungen
 - Verwendet keine anderen Komponente; wird aber von allen anderen genutzt!

Systemsteuerung

- Klasse: DbjSystem
- Datenstrukturen: keine
- Verwendet
 - Buffer Manager um Puffer einmalig anzulegen (bzw. freizugeben)
 - Lock Manager um Lockliste einmalig anzulegen (bzw. freizugeben)
 - Catalog Manager um Katalog zu initialisieren

Operating System Services (OSS)

- Abstraktion aller kritischen Betriebssystemfunktionen
 - File-I/O
 - Speicherverwaltung
 - Private & Shared Memory
 - Latching
 - Kurzzeitsperren; über Semaphoren realisiert
- Unterstützte Plattformen
 - AIX, Linux, Windows/Cygwin

– TODOs –

RunTime

- ORDER BY
- UPDATE
- Subselects

Index Manager

- Hash Index
- Performance des B-Tree Index

Log Manager

- Komplette neu implementieren
- Fokus auf High-Performance-Schreib-Operationen

Lock Manager

- Implementierung von hierarchischen Sperren
 - Tabellen/Index-Sperren
 - Seiten-Sperren
 - Tupel-Sperren

Buffer Manager

- Implementierung weiterer Strategien zur Seitenverdrängung