Datenbank-Implementierungstechniken
Implementierung des RecordManagers

Gruppe 36561

Daniel Ölschlegel, Christian Müller, Ralf Hoffmann

16. Mai 2010

Inhaltsverzeichnis

1 Zielstellung

Ziel war die Implementierung des RecordManagers für das ToyDB-System auf Grundlage der vorgegebenen Schnittstellen. Die implementierten Klassen sollen dabei die JUnit-Tests reibungslos bestehen.

2 Umsetzung

Als Entwicklungsumgebung wurde Eclipse gewählt. Die Schnittstellen-Definition sowie Test werden dabei jeweils vom Betreuer gestellt.

3 Designentscheidung

Es wurde ein RecordManager unter Umsetzung des PAX-Verfahren implementiert. Dabei wird ein PagedFile mit festen Seitengrößen angelegt. Auf der ersten Seite werden die Informationen über die Datenbank abgelegt. Diese sind der Relationsname sowie die Anzahl der Attribute, ihr Name, Typ und die jeweilige Länge.
Die weiteren Seiten bestehen aus einem Header in Form eines Belegungsindikators, welcher belegte Slots auf einer Seite kennzeichnet. Im Folgenden werden die Attribute - genauer gesagt deren Werte - abgelegt.
Eine Seite besteht also aus dem Header, der die freien Slots auf einer Seite kennzeichnet, sowie aus mehreren hintereinander gereihten Minipages, welche jeweils nur ein Attribut der Records speichern. Somit ist ein Record in mehreren Minipages auf einer Seite verteilt, aber gleiche Attribute liegen nah beieinander.

4 Bemerkungen

- Durch die Ablage der Verwaltungsdaten auf nur eine Seite wird die Dimension der Relation eingeschränkt.
- Datensätze werden auf maximal eine Seite abgelegt, also nicht über mehrere Seiten verteilt.
- Die Daten werden nicht sortiert im PagedFile abgelegt, wodurch eine Suche immer über ein komplettes PagedFile geschieht.
- In einem Vektor wird für alle Seiten des pagedFile hinterlegt, ob die jeweilige Seite noch freie Slots hat.

5 Implementation

5.1 RecordImpl

Es wurde der Konstruktor für `RecordImpl` sowie Getter für `RID` und `data` implementiert.

5.1.1 Attribute

- `private byte[] data;` - der Attributwert selbst
- `private RID rid;` - der Record Identifier, bestehend aus Seiten- und Slot-Nummer

5.1.2 Methoden

- `public RecordImpl(RID rid, byte[] data)`
  Konstruktor für die Klasse `RecordImpl`
  Die Attribute `data` und `RID` werden auf die übergebenen Werte gesetzt.
- `public byte[] getData()`
  Getter für `data`
- `public RID getRID()`
  Getter für `RID`

5.2 RecordFileImpl

Implementiert wurden die geforderten Methoden `close`, `destroy`, `getRecord`, `insertRecord`, `deleteRecord`, `updateRecord`, `getAttrInfo`, `getNumOfPages` sowie `forceAll`

5.2.1 Attribute

- `private AttrInfo[] attribs;` - Liste der Attribut-Informationen
- `private int recordRawSize = 0;` - Größe eines Records
- `private int usedPages = 0;` - Anzahl benutzter Seiten
- `private PagedFileImpl pFile;` - Instanz von `PagedFile`
- `private BitSet bs = new BitSet();` - Bitset über alle Seiten um zu hinterlegen, ob mindestens ein Slot auf den jeweiligen Seiten frei ist.
- `private int mpSize = 0;` - Größe einer Minipage
- `private int headerSize = 0;` - Größe des Seitenheaders
- `private byte lastByte = 0x00;` - Hilfsvariable für Header-Erzeugung
- `private int stepRange = 0;` - Abstand zwischen 2 Attributen auf den Seiten
- `private int[] mpBegin;` - Begin der Miniseiten auf einer Seite
- `private int[] attLen;` - Attributlängen
- `private static final byte[] calc = {-128, 64, 32, 16, 8, 4, 2, 1;}` - Array für Bit-Operationen

5.2.2 Methoden

- `public RecordFileImpl(AttrInfo[] attribs, PagedFileImpl pFile) throws DBException`
  Konstruktor der `RecordFileImpl`. Es werden diverse Überprüfungen und Initialisierungen durchgeführt.
  So wird geprüft ob das `pFile`-Objekt nicht Null ist, die Relation mindestens ein Attribut besitzt und die Länge der Attribute in der Relation bestimmt. Des weiteren wird ermittelt, wie viele Elemente auf eine Miniseite abgelegt werden können, die Headersize in Byte, die Größe einer Miniseite sowie deren Begin auf einer Seite.
  Anschließend wird der Belegungsindikator über dem PagedFile angelegt, welcher Auskunft darüber gibt, ob noch Slots auf den einzelnen Seiten frei sind und die Anzahl belegter Seiten ermittelt.
- `private boolean fullSlots(byte[] data)`
  Ermitteln, ob alle Slots belegt sind. Dazu wird der Header aus dem Rohdatenstrom durchlaufen und nach 0en gesucht. Ist mindestens eine 0 vorhanden, ist mindestens ein Slot frei und somit die Seite nicht voll.
  Rückgabe: `true` wenn voll, `false` wenn mindestens 1 Slot frei.

- `private boolean emptySlots(byte[] data)`
  Ermitteln, ob alle Slots einer Seite frei sind. Dazu wird der Header aus dem Rohdatenstrom durchlaufen und nach 1en gesucht. Ist mindestens eine 1 vorhanden, so ist mindestens ein Slot belegt und somit die Seite nicht leer.
  Rückgabe: `true` wenn leer, `false` wenn mindestens 1 Slot belegt.

- `public Record getFirstRecord() throws DBException`
  Laden der Seite im `pFile`. Solange kein Slot gefüllt ist wird die nächste Seite geladen und dort weiter gesucht. Ist ein Slot mit einem Eintrag gefunden, so wird der Record mittels `getRecord` zurück gegeben.

- `public Record getNextRecord(RID rid) throws DBException`
  Laden und auslesen der in `RID` angegebenen Seite. Befindet sich der nächste Record auf der Seite so wird dieser zurück gegeben, andernfalls befindet sich der Record auf der nächsten Seite und wird mittels `getRecord` zurück gegeben.

- `public void close() throws DBException`
  Schliessen des PagedFile mittels `close`;

- `public Record getRecord(RID rid) throws DBException`
  Die Daten der Seite angegebenen Seite werden ausgelesen und die Position des Records auf der Seite ermittelt. Anschließend werden die Daten aus dem Record in einen neuen übertragen und zurückgegeben.

- `private void writeRecord(int pageNum, int slotNum, byte[] raw) throws DBException`
  Es wird die angegebene Seite im PagedFile geöffnet und der Record an die angegebene Slot Position geschrieben sowie die Seite als dirty markiert, wodurch sie umgehend zurück geschrieben wird.

- `public RID insertRecord(byte[] data) throws DBException`
  Ermitteln der Seite mit einem freien Slot. Ist keine Seite frei, so wird eine neue angelegt: Dabei wird ein Header angelegt und als komplett frei (bis auf den ersten Slot) markiert sowie die Anzahl der benutzten Seiten um 1 erhöht und der Belegungsindikator für die Seite angelegt.
  Es wird ein freier Slot auf der Seite ermittelt, falls diese nicht gerade neu angelegt wurde. Sind nun alle Slots belegt, so wird die Seite als voll markiert. Der Record wird anschliessend auf die Seite geschrieben und der `RID` zurückgegeben.

- `public void deleteRecord(RID rid) throws DBException`
  Ermitteln der Seiten- und Slot-Nummer und Einlesen der Daten auf der Seite. Falls die Seite existiert, so wird das Belegt-Bit im Header gelöscht und die Seite als `dirty` markiert. Falls nun keine Slots auf der Seite mehr belegt sind, wird die Seite gelöscht und die Anzahl benutzter Seiten um 1 verringert. Sind noch Slots frei, so wird die Seite als belegbar markiert.

- `public void updateRecord(Record rec) throws DBException`
  Ermittlung der Seite und des Slots des übergebenen Records sowie Auslesens der ermittelten Seite im PagedFile mit abschliessendem Zurückschreiben der Daten im Record in den passenden Slot auf der Seite im `pFile`.

- `public AttrInfo[] getAttrInfo()`
  Rückgabe des Arrays `attribs`

- `public int getNumOfPages()`
  Falls nicht bereits die Anzahl benutzter Seiten in usedPages hinterlegt wurde, so wird beginnend mit der ersten Seite des `pFile`s mittels `nextPage` des `pFile`s geprüft ob die Seite existiert und gegebenenfalls aufaddiert.

- `public void forceAll() throws DBException`
  Aufruf `forceAllPages` des `pFile`s.

5.3 RecordFileFactoryImpl

Implementiert wurde der Konstruktor für `RecordFileFactoryImpl` sowie die Methoden `RecordFile createFile`, RecordFile `openFile`, `destroyFile`, RecordFileScan `createScan`.

5.3.1 Attribute

-

5.3.2 Methoden

- `public RecordFile createFile(String fileName, AttrInfo[] attribs) throws DBException`
  Eine neue Datei mit dem übergebenen Namen wird mittels der `createFile`-Methode der Klasse `PagedFile` erstellt, falls diese noch nicht existiert. Falls Attribute angegeben wurden, so wird eine neue Seite erstellt, welche die Informationen über die verwendete Relation aufnehmen soll. Zu beachten ist, dass derzeit eine Limitierung auf eine Index-Seite vorliegt und somit nur eine begrenzte Anzahl an Attributen möglich ist. Die erstellte Seite wird abschliessend un`pinned` sowie als `dirty` markiert, damit sie zurück geschrieben wird.

- `public RecordFile openFile(String fileName) throws DBException`
  Falls die angegebene Datei existiert, so wird die erste Seite der Datei mittels `openFile` der Klasse `PagedFileImpl` geladen. Alle hinterlegten Attribut-Deklarationen werden anschliessend in das Array `attribs` gelesen und die Seite wieder ungepinnt

- `public void destroyFile(String fileName) throws DBException`
  Überprüfung ob übergebene Datei existiert und gegebenenfalls Löschen der gewählten Datei.

- `public RecordFileScan createScan() throws DBException`
  Rückgabe einer neuen `RecordFileScanImpl` - Instanz

5.4 RecordFileScanImpl

5.4.1 Attribute

- `private RID CurrentRID;` - `CurrentRID` gegenwärtiger `RID`
- `private AttrInfo[] AttribList;` - `AttribList` Liste der gewünschten Attribute
- `private int AttribSize = 0;` - `AttribSize` Größe des Attributes
- `private RecordFileImpl rFile;` - Instanz von `rFile`
- `private int Offsets[];` `Offsets` zum Speichern der Position der Attribute im Record

5.4.2 Methoden

- `public void openScan(RecordFile rFile, AttrInfo[] attribs) throws DBException`
  Erstellen einer lokalen Kopie der Liste der Attribut sowie der RecordFile-Referenz. Einlesen aller Attribut-Informationen des RecordFiles sowie Berechnung der Positionen der Attribute im Record. Als Referenzpunkt für die Suche wird die `RID` gespeichert.
- `public Record getNextRecord() throws DBException`
  Falls der Scan initialisiert wurde, so wird der aktuelle Record an den Anfang der Relation gesetzt, andernfalls der nächste Record angefordert. Existiert der angegebene Record, so werden die Attributwerte in einen neuen Record kopiert und der generierte Record zurückgegeben.
- `public void closeScan() throws DBException`
  Keine Funktion