**6 11 B-Trees**

**11.2.2 Feed Özellik Türü Tablosu**

1. ...
2. 10’P’ ’Phosphorus as % of DM’ 15
3. 11’Ca’ ’Calcium as % of DM’ 16
4. 12’RFV’ ’Relative Feed Value (calculated)’ 17
5. 13’S’ ’Sulfur as % of DM’ 18
6. 14’K’ ’Potassium as % of DM’ 19

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | 15’Mg’ | ’Magnesium as % of DM’ | 20 |
| 8 | 16’Fat’ | ’Fat as % of DM’ | 14 |
| 9 | ... |  |  |

Bölüm 11.2.2'deki tablo, FeedAttribType tablosundaki kayıtların bir alt kümesini içerir ve metin web sitesinde FeedAttribType.tbl olarak mevcuttur. Tam tablo, her biri 7 alandan oluşan 57 farklı satır içermektedir. Feed tablosunda olduğu gibi, FeedAttribType tablosu da satırlar ve sütunlar halinde düzenlenmiştir.

FeedAttribute" tablosunun bir alt kümesi 11.2.3. bölümde sunulmuştur. Her feed attribute, ilgili FeedID, FeedAttribTypeID ve verilen feed için bu nutrient'in miktarını içeren ve tablodaki Value sütunu olarak adlandırılan kısmından oluşur.

|  |  |
| --- | --- |
| **11.2.3 FeedAttribute Tablosu** | |
| 1 ... |  |
| 2 1316 | 10 0.250000 |
| 3 1316 | 11 0.210000 |
| 4 1316 | 12 128.000000 |
| 5 1316 | 13 0.150000 |
| 6 1316 | 14 1.200000 |
| 7 1316 | 15 0.200000 |
| 8 1316 9 ... | 16 3.000000 |
| 10 1317 | 10 0.110000 |
| 11 1317 | 11 0.220000 |
| 12 1317 | 12 129.000000 |
| 13 1317 | 13 0.110000 |
| 14 1317 | 14 0.530000 |
| 15 1317 | 15 0.130000 |
| 16 1317 17 ... | 16 2.000000 |
|  |  |

Feed verilerini bu şekilde saklamak oldukça esnektir. Yeni besin maddeleri kolayca eklenebilir. Feed'ler de eklenebilir. Feed öznitelikleri mevcutsa saklanabilir veya atlanabilir. Bazen, ilişkisel veri tabanlarını kullanan programların birden fazla tablodan veri alması gerekebilir ve tablolar arasındaki verileri ilişkilendirmesi gerekebilir. Örneğin, geçici olarak feed numarası, feed adı, besin adı ve ilgili feed için o besinin değerini içeren bir tablo oluşturmak uygun olabilir. Bu durum, 11.2.4 Bölümü'ndeki tablo gibi bir tablo oluşturmanın yararlı olabileceği durumlardan biridir.

Tüm feed'lerin ortalama fosfor içeriğini hesaplamak isteyebiliriz. Aslında, veri tabanındaki her besin türü için ortalama içeriği hesaplamak isteyebiliriz. Bu durumda, 11.2.4 Bölümü'ndeki tablo gibi bir tablo çok kullanışlı olacaktır.

**7 11 B-Trees**

**11.2.4 ATemporary (Geçici Bir) Tablo**

1. 10 Corn Silag P 0.25
2. 10 Corn Silag Ca 0.21
3. 10 Corn Silag RFV 128.0
4. 10 Corn Silag S 0.15
5. 10 Corn Silag K 1.2
6. 10 Corn Silag Mg 0.2 8 10 Corn Silag Fat 3.0
7. ...
8. 11 Almond Hul P 0.11
9. 11 Almond Hul Ca 0.22
10. 11 Almond Hul RFV 129.0
11. 11 Almond Hul S 0.11
12. 11 Almond Hul K 0.53
13. 11 Almond Hul Mg 0.13
14. 11 Almond Hul Fat 2.0
15. ...

İlişkisel veri tabanlarına genellikle SQL veri tabanları denir. SQL, Sistem Sorgu Dili anlamına gelir. SQL, ilişkisel veri tabanlarını sorgulamak için kullanılan bir dildir. SQL, Bölüm 11.2.4 'teki gibi geçici tablolar oluşturmak için kullanılabilir. Bu tabloyu oluşturmak için kullanılan SQL ifadesi şu şekilde yazılır:

Feed.FeedNum, Feed.Name, FeedAttribType.Name, FeedAttribute.Value WHERE

Feed.FeedID = FeedAttribute.FeedID AND

FeedAttribute.FeedAttribTypeID = FeedAttribType.FeedAttribTypeID

Bu SQL ifadesi, üç tablonun birleşimi olarak bilinir çünkü sonucu oluşturmak için üç tablo bir araya getirilecektir. Bu sorguyu üç tabloyu okuyan komutlara dönüştürmek ve birleştirmenin sonucu olarak verimli bir şekilde yeni bir geçici tablo oluşturmak ilişkisel veritabanına kalmıştır.

Kendi ilişkisel veri tabanımızı uygulayacak olsaydık, bu üç tablonun birleştirme işlemi Bölüm 11.2.5'te görünen koda benzer şekilde programlanabilirdi. Yanıltılmayın. İlişkisel veri tabanları bunun gibi belirli birleştirmeleri programlamaz, ancak üç tablonun birleştirilmesi işlevsel olarak bu koda eşdeğer olabilir. Programın tamamı, metnin web sitesinde joinquery.py olarak mevcuttur. Buradaki readField fonksiyonu, alan kısıtlaması nedeniyle kısaltılmıştır, ancak bir tablo dosyasından herhangi bir alanı okur. Birleştirme algoritması tablolardan birini seçer ve onu baştan sona okur. Bu durumda, FeedAttribute tablosu baştan sona okunur. Her bir feed özelliği için,feed tablosundaki eşleşen feed ıd bulunması gerekir. Bölüm 11.2.5 'teki kodda bu, her sorgu satırı için feed numarasını ve feed adını sağlamak üzere ortalama olarak feed tablosunun yarısını okumayı içerir. Benzer şekilde, feed öznitelik adını sağlamak için, her sorgu çıktısı satırı için feed öznitelik adını sağlamak amacıyla ortalama olarak FeedAttribType tablosunun yarısı okunur.

Bu işlemin karmaşıklığı, n FeedAttribute.tbl'deki kayıt sayısı ve m FeedAttribType.tbl ve Feed.tbl'deki kayıt sayılarının maksimumu olmak üzere O(n\*m)'dir. n ve m yaklaşık olarak eşitse bu, O(n²) performansıdır. İki değer yaklaşık olarak eşit olup olmadığına bakılmaksızın, bu sorgunun performansı, küçük örnek tablomuzda bile, pek iyi değildir. Sorguyu, 2.66GHz Intel Core i7 işlemci, 8GB RAM ve katı hal sürücüsü olan bir bilgisayarda çalıştırmak yaklaşık 4.993 saniye sürmektedir.

**8 11 B-Trees**

**11.2.5 Tabloların Birleştirilmesinin Programlanması**

**import datetime**

**def** readField(record,colTypes,fieldNum):

1. #fieldNumiszerobased
2. #recordisastringcontainingtherecord
3. #colTypesisthetypesforeachofthecolumnsintherecord
4. offset = 0
5. **for** i **in** range(fieldNum): 8 colType = colTypes[i]

9

1. **if** colType == "int":
2. offset+=10
3. **elif** colType[:4] == "char": 13 size = int(colType[4:])
4. offset += size
5. **elif** colType == "float":
6. offset+=20
7. ...
8. **return** val
9. **def** main():
10. #SELECTFeed.FeedNum,Feed.Name,FeedAttribType.Name,FeedAttribute.ValueWHERE
11. #Feed.FeedID=FeedAttribute.FeedIDAND
12. #FeedAttribute.FeedAttribTypeID=FeedAttribType.FeedAttribTypeID
13. attribTypeCols = ["int","char20","char60","int","int","int","int"]
14. feedCols = ["int","int","int","char50","datetime","float","float","int","char50","int"]
15. feedAttributeCols = ["int","int","float"]
16. before = datetime.datetime.now()
17. feedAttributeTable = open("FeedAttribute.tbl","r") 28 **for** record **in** feedAttributeTable:
18. feedID = readField(record,feedAttributeCols,0)
19. feedAttribTypeID = readField(record,feedAttributeCols,1)
20. value = readField(record,feedAttributeCols,2)
21. feedTable = open("Feed.tbl","r")
22. feedFeedID = -1
23. **while** feedFeedID != feedID:
24. feedRecord = feedTable.readline()
25. feedFeedID = readField(feedRecord,feedCols,0)
26. feedNum = readField(feedRecord,feedCols,2)
27. feedName = readField(feedRecord,feedCols,3)
28. feedAttribTypeTable = open("FeedAttribType.tbl", "r")
29. feedAttribTypeIDID = -1
30. **while** feedAttribTypeIDID != feedAttribTypeID:
31. feedAttribTypeRecord = feedAttribTypeTable.readline()
32. feedAttribTypeIDID = readField(feedAttribTypeRecord,attribTypeCols,0)
33. feedAttribTypeName = readField(feedAttribTypeRecord,attribTypeCols,1)
34. **print**(feedNum,feedName,feedAttribTypeName,value)
35. after = datetime.datetime.now()
36. deltaT = after - before
37. milliseconds = deltaT.total\_seconds() \* 1000
38. **print**("Time for the query without indexing was",milliseconds,"milliseconds.")

50

1. **if** \_\_name\_\_ == "\_\_main\_\_":
2. main()

Bölüm 11.2.5'teki kod, Feed.tbl ve FeedAttribType.tbl tablolarının, eşleşen feed ve feed attribute türünü bulmak için dış döngüde her seferinde ardışık olarak okunması nedeniyle verimsizdir. Disk sürücülerinin rastgele erişimli aygıtlar olduğunu fark edersek, bu sorgunun verimliliğini artırabiliriz. Bu, disk sürücüsünün okuma kafasını bir dosya içinde herhangi bir yere konumlandırabileceğimiz anlamına gelir. Eşleşen feed veya feed attribute türünü bulmak için tablonun başından başlamak zorunda değiliz. Eşleşen kaydı bulmak için tablo içinde gezinebiliriz.

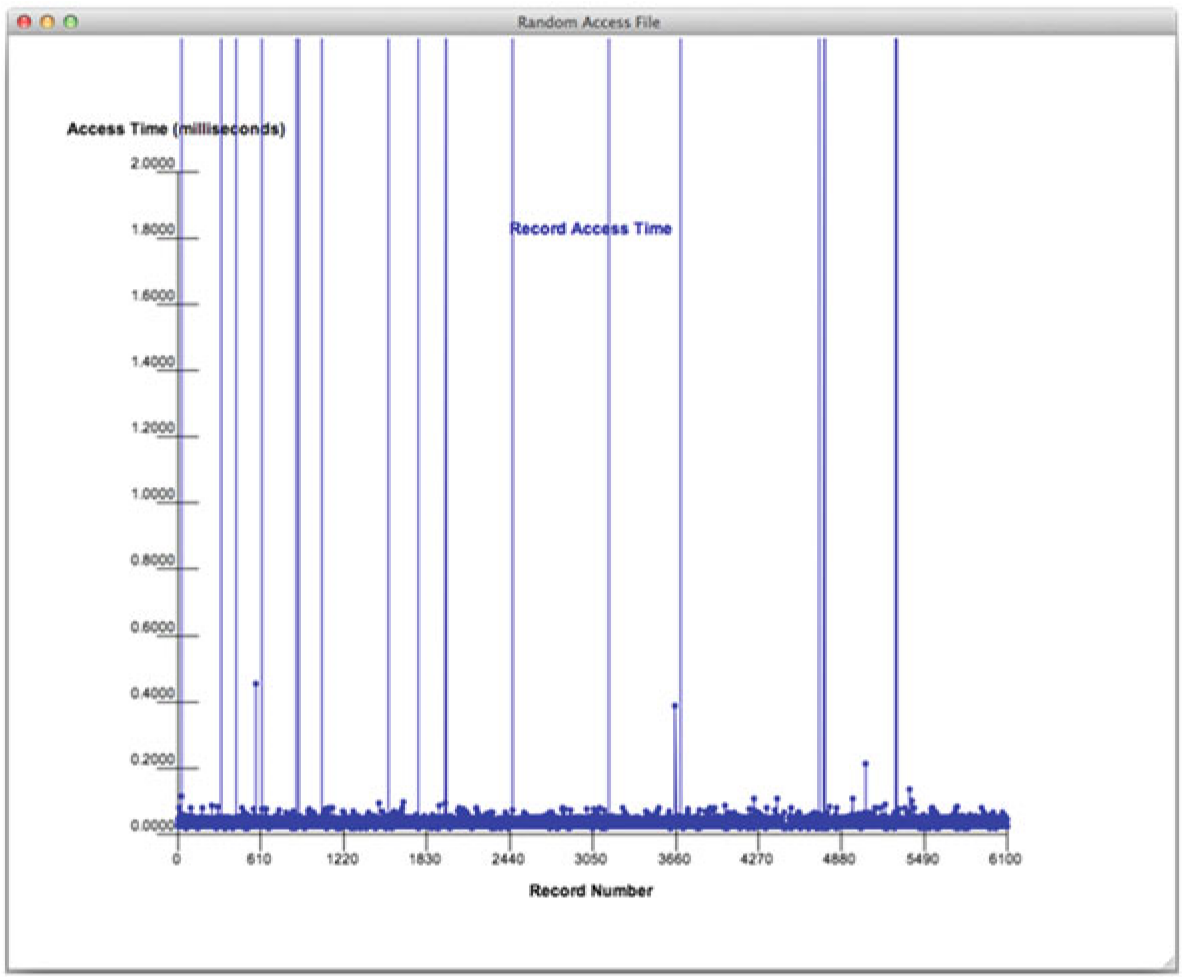
**9 11 B-Trees**

**11.2.6 The readRecord Fonksiyonu**

1. **def** readRecord(file,recNum,recSize):
2. file.seek(recNum\*recSize)
3. record = file.read(recSize)
4. **return** record

Python, dosyalarda okuma kafasının bir dosya içinde bir bayt ofsetine konumlandırılması için bir seek yöntemi içerir. Dosyalardaki read yöntemi, belirli bir sayıda bayt okur ve bunları bir dize olarak döndürür. Bu readRecord fonksiyonunu ve seek yönteminin işlevselliğini test etmek için, FeedAttribute.tbl dosyasındaki kayıtları rastgele erişmek amacıyla bir program yazılmıştır. Bu deneyin sonuçları Şekil 11.4'te gösterilmiştir. Veriler, dosya içindeki konumundan bağımsız olarak herhangi bir kayıta erişmenin yaklaşık aynı süreyi aldığını göstermektedir. Her deneyde olduğu gibi, birkaç anomali vardı. Ancak, kayıtların büyük çoğunluğuna aynı sürede veya neredeyse aynı sürede erişildi.

Diyelim ki Feed.tbl ve FeedAttribType.tbl dosyalarını anahtarlarına göre artan sırayla sıraladık. Feed.tbl dosyası FeedID'ye göre ve FeedAttribType.tbl dosyası FeedAttribTypeID'ye göre sıralanacaktır. Ardından, Bölüm 11.2.5'teki kodda her feed özniteliği için eşleşen kayıtları bulmak üzere bu iki dosyada ikili arama kullanabiliriz. Tablolar rastgele erişilebilir olduğundan, sorgu süresi O(n\*m)'den O(n log m)'ye düşürülebilir. Ancak,

,

**Fig.11.4** Bir Dosyadaki Rastgele Okunan Kayıtlara Erişim Süresi

**10 11 B-Trees**

Bir veri tabanı tablosunun her zaman veya hiç olmasa bile, bir alana göre sıralanacağını varsayamayız. Veri tabanlarına her zaman yeni kayıtlar eklenebilir ve eski kayıtlar silinebilir.

B-Trees ihtiyacı buradan geliyor. B-Trees, veri tabanı tablosu içindeki herhangi bir kayda O(log n) arama süresi sağlamak için, tabiri caizse, bir verit abanı tablosunun üst kısmına inşa edilen bir ağaç yapısıdır. Kayıtların kendileri herhangi bir sırada olabilse de, B-Trees tabloya O(log n) arama karmaşıklığını sağlar.

Bir B-Trees, ağaca kayıtlar veya öğeler eklenerek oluşturulur. Dizin oluşturulduktan sonra, B-Trees'de depolanan anahtar değerine dayalı olarak herhangi bir kaydın etkili bir şekilde aranmasını sağlar. Bölüm 11.2.7'deki kodu göz önünde bulundurun. 18-23. satırlar Feed.tbl indeksini, 39-44. satırlar ise FeedAttribType.tbl indekisni oluşturur. İndeksler oluşturulduktan sonra sorgu programlanırken kullanılır. 55. satırdan başlayan döngü artık iki tablodaki karşılık gelen kayıtları aramak için iki while döngüsü içermiyor. Bunun yerine, iki tablodaki karşılık gelen kayıtları bulmak için B-Trees'e başvurulur. Bu şekilde programlandığında, Bölüm 11.2.7'deki sorgu yaklaşık 1,628 saniyede çalışır; bu, orijinal, indekslenmemiş sorgudan üç kat daha hızlıdır. Buradaki örnek sorgu nispeten küçük tablolar kullanıyor. Feed.tbl veya Feed AttribType.tbl tablolarının milyonlarca kayıt içerdiği durumlarda hız artışını hayal edin. Bu durumda, özgün sorgu kabul edilebilir bir sürede tamamlanmazken, burada verilen indekslenmiş sorgu hemen hemen aynı sürede veya en kötü ihtimalle bir saniye daha uzun sürede tamamlanırdı.

**11.2.7 Efficient Join (Verimli Birleştirme)**

**def** main():

1. #SelectFeed.FeedNum,Feed.Name,FeedAttribType.Name,FeedAttribute.Valuewhere
2. #Feed.FeedID=FeedAttribute.FeedIDandFeedAttribute.FeedAtribTypeID=FeedAttribType.ID
3. attribTypeCols = ["int","char20","char60","int","int","int","int"]
4. feedCols = ["int","int","int","char50","datetime","float","float","int","char50","int"] 6 feedAttributeCols = ["int","int","float"]

7

8 feedAttributeTable = open("FeedAttribute.tbl","r")

9

1. **if** os.path.isfile("Feed.idx"):
2. indexFile = open("Feed.idx","r")
3. feedTableRecLength = int(indexFile.readline())
4. feedIndex = eval(indexFile.readline())
5. **else**:
6. feedIndex = BTree(3)
7. feedTable = open("Feed.tbl","r")
8. offset = 0
9. **for** record **in** feedTable:
10. feedID = readField(record,feedCols,0)
11. anItem = Item(feedID,offset)
12. feedIndex.insert(anItem)
13. offset+=1
14. feedTableRecLength = len(record)

24

1. **print**("Feed Table Index Created")
2. indexFile = open("Feed.idx","w")
3. indexFile.write(str(feedTableRecLength)+"**\n**")
4. indexFile.write(repr(feedIndex)+"**\n**")
5. indexFile.close()

30

**if** os.path.isfile("FeedAttribType.idx"):