UNIVERSITY OF ALBERTA

DEPARTMENT OF COMPUTING SCIENCE

An Introduction to Berkeley DB

CMPUT 291
File and Database Management Systems

Objectives

- ➢ Levels of abstraction in a typical DBMS.
- ➢ Introduction to Berkeley DB.
  - What is Berkeley DB.
  - File Organizations supported by Berkeley DB.
  - Access method operations, cursors.
- ➢ Take a look at the Berkeley DB API using simple examples in Python.

Levels of abstraction in a typical DBMS

- Data is stored in tables in conceptual schema.
- High level SQL commands (DDL or DML statements) are used to manipulate data in database.
- Data is actually organized in files and indexes in physical schema.
- A DBMS uses suitable storage structure, index files, and access paths to evaluate queries and return results.

What Is Berkeley DB?

- It is an open source embedded database library that provides a simple function-call API for data access and management.
- It supports a number of file structures and operations.
- You can use it to:
  (1) Develop programs working with storage structures and indexes.
  (2) Develop applications that don't need the full functionality of a DBMS and require high performance.

File organizations supported in Berkeley DB

- **Hash:**
  - ✓ Data is stored in an extended linear hash table
  - ✓ Good for applications that need quick random-access.

- **Btree:**
  - ✓ Data is stored in a sorted, balanced tree structure
  - ✓ Good for range-based searches

For more information see BerkeleyDB Access Methods

File organizations supported in Berkeley DB (Cont.)

Record-number-based:

The logical record number is the primary key of records (compared to Hash and Btree organizations when one or a combination of attributes form a key for records)

- **Queues:**
  - ✓ Data is stored in a queue as fixed-length records
  - ✓ Good for fast inserts at the tail of the queue
  - ✓ It also supports read/delete operation from the head of the queue.
  - ✓ provides record-level locking (as opposed to the page-level locking that the other access methods use)
- **Recnos**:
  - ✓ Data is stored in either fixed or variable-length records.

For more information see [BerkeleyDB Access Methods]

A summary of methods in Database class

Method	Description
open	Create/Open a database
close	Close a database
get	Get items from a database
put	Store items into a database
delete	Delete items from a database
stat	Return database statistics
truncate	Emptying a database of all records
sync	Flushes all modified records from the DB cache to disk
associate	Declares one DB as secondary index of a primary DB

For more information see BerkeleyDB API

A summary of methods in Cursor class

- A database cursor supports traversing the database.
- It is the only way to access individual duplicate records.

Method	Description
dup	Duplicates a cursor
close	Discards the cursor
count	Returns the number of duplicate data items for the key to which the cursor refers
delete	Deletes the key/value pair to which the cursor refers
get	Retrieves key/value pairs from the database
set	Moves the cursor to the specified key in the database and return the associated key/value pair
put	Stores key/value pairs into the database using a cursor

For more information see BerkeleyDB API

Note: Connecting to the lab machine using ssh

https://eclass.srv.ualberta.ca/mod/page/view.php?id=3105765

Berkeley DB API (Python)

1. Create the database:

```
from bsddb3 import db

database = db.DB() #handle for Berkeley DB database
DB_File = "fruit.db"
database.open(DB_File ,None, db.DB_HASH, db.DB_CREATE)
# database.open(DB_File ,None, db.DB_BTREE, db.DB_CREATE)
# database.open(DB_File ,None, db.DB_QUEUE, db.DB_CREATE)
# database.open(DB_File ,None, db.DB_RECNO, db.DB_CREATE)
# The arguments correspond to (fileName, database name within the file for multiple
databases, database type, flag to create database)
```

Berkeley DB API (Python)

2. Declare the cursor for the database:

```
curs = database.cursor()
```

3. Insert records into the database:

```
# insertion of (key, value) pair using the cursor:
#The arguments corresponds to (key, value, flag to insert key-value pair )
curs.put(b"apple", "red ", db.DB_KEYFIRST)

# insertion using the database object’s put method
database.put(b"pear", "green")
```

Berkeley DB API (Python)

4. Iterate through the database to display all key-value pairs:

```
# use the cursor
iter = curs.first()
while iter:
        print(iter)
        iter = curs.next()
```

Berkeley DB API (Python)

5. Retrieve a specific key-value or index-value pair

```
# using the database object’s get method: only retrieves the value
result = database.get(b'pear')
print(result)
# b'green'

# using the cursor object’s set method:
result = curs.set(b'pear')
#Moves the cursor to the specified key in the database and return the associated key/value pair.
print(result)
```

Berkeley DB API (Python)

6. Remove a (key,value)

```
# remove using the cursor – deletes the key-value pair currently referenced by the cursor:
curs.current() #Returns the key/value pair currently referenced by the cursor
curs.delete()
# our database now has (apple,red) only
# remove by using database object
database.delete(b'apple')
# Now our database is empty.
```

7. Close database and cursor:

```
curs.close()
database.close()
```

Berkeley DB API (Python)

8. Show records with the same key (duplicates):

```
# Set duplicate flag before you create the database:
# db.DB_DUP allows duplicate data items in the tree
database.set_flags(db.DB_DUP)

#Duplicate key-value pairs could be traversed using cursor

# prints no. of k-v pairs that have the same key (for the key which the cursor is pointing to)
print(curs.count())

# prints the next k-v pair if it is a duplicate
print(curs.next_dup())
#returns ‘None’ if the next k-v pair of the database is not a duplicate data record for the
current key/value pair
```

Exercise 1: Iterate_All

- Write a code to iterate through all key-value pairs(including the duplicates) using next_dup() method for duplicate keys.
  - In order to test your code, add the following key-value pairs to see if they are retrieved:

```
database.put(b'key1', "value1")
database.put(b'key1', "value2")
database.put(b'key2', "value1")
database.put(b'key2', "value2")
```

Exercise 1: Iterate_All (Cont.)

- Iterating through k-v pair including duplicates:

```
curs = database.cursor()
iter = curs.first()

while (iter):

    print(curs.count())
    #prints no. of rows that have the same key for the current key-value pair referred by cursor.

    print(iter)

    #iterating through duplicates:
    dup = curs.next_dup()
    while(dup!=None):
        print(dup)
        dup = curs.next_dup()

    iter = curs.next()
```

- A solution is provided on e-class (named Iterate_All.py)

Exercise 2: Populate bdb

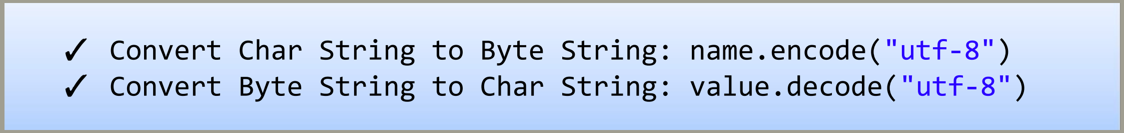
Write a program that:

- Gets a student name(as a key) from input
- Searches the database if there is any key(student) with the same name
- Prints any found key as well as its value
- Then, asks the user if the input key should be inserted, and if yes, the program asks for its value as well

  (let’s say the value is the student’s mark)
- Terminates if “q” is entered.

  ➢ Pay attention that duplicate keys should be supported (multiple marks for a student)

  ➢ Hint:



```
✓ Convert Char String to Byte String: name.encode("utf-8")
✓ Convert Byte String to Char String: value.decode("utf-8")
```

Exercise 2: Populate bdb (Cont.)

- In order to set the cursor on the first data item for the input name:

```
name = input("Enter a student Name to look up: ")
result = curs.set(name.encode("utf-8"))
```

- To get rid of the byte prefix (b' ') for printing students' names and marks:

```
print("Name: "+str(result[0].decode("utf-8"))+", Mark: "+str(result[1].decode("utf-8")))
```

- A solution is provided on e-class (named Populate_bdb.py)

Exercise 3: Range Search

Write a program that:

- Gets a name called Starting_Name
- Gets another name called Ending_Name
- Searches for all students' names from the previous exercise that come after or in the same position as Starting_Name (when sorted alphabetically) and come before Ending_Name.
- Prints the names and marks of the students found in the previous step.

➢ Hint:

✓ In Btree DB type, By default, the sort order is lexicographical, with shorter keys coming before longer keys.

✓ Curs.set_range(key): In Btree DB type, sets the cursor to key/value pair which is the smallest key greater than or equal to the specified key, and returns the pair.

Exercise 3: Range Search

- To get the record that has the smallest key greater than or equal to the Starting Name:

```
Starting_Name = input("Enter the Starting_Name: ")
result = curs.set_range(Starting_Name.encode("utf-8"))
```

- To check if the student's name comes before Ending_Name:

```
if (str(result[0].decode("utf-8")[0:len(Ending_Name)]) < Ending_Name):
   ...
```

- A solution is provided on e-class (named Range_Search.py)

- **Question:** What if the keys were **numbers** instead of names, and you were asked to output the k-v pairs with keys **higher** than **Starting_Number** and **Lower** than **Ending_Number**?

What's Next?

- Check out the page “introduction to Python bsddb3” for more information.
- Take a look at Berkeley DB API for C & Java in this tuto rial.