Guide to the Berkeley DB Example Programs

This document helps answer the question: “Which Berkeley DB sample programs use feature *X*?” Topics are grouped from the simplest operations on databases and environments to more involved applications, ending with demonstrations of the most specialized features. Most of these programs are included with every Berkeley DB distribution in the db-X.Y.Z/examples directory; the remaining ones are listed on the BDB Learn More page.

Guide to Features and Examples

Simple Database Access

Add lines of text to a database & display them – ex_access

This simple program uses the DB->put() and DB->get() API calls to store lines of text, entered from the standard input, into a btree database. Each key is the text as it was entered; the data is the reversed version of the key. At EOF (or “quit”), it opens a cursor and displays each record, ordered by the key. Since the records are DB->put() with the DB_NOOVERWRITE flag, the error DB_KEYEXIST is returned if you enter a key which has already been stored in the database.

Option:

-r Remove the database during startup (default: off: add new records to any existing ones).

Language	Example Source or Project File
C	c/ex_access.c
C++	cxx/AccessExample.cpp
C++ STL	stl/AccessExample.cpp
Java	java/src/db/AccessExample.java
C#	csharp/excs_access/excs_access.csproj

Database Types

Using a btree with both keys and record numbers – ex_btrec

This program shows how to store automatically numbered records in a btree database, by setting its DB_RECNUM flag. This variety of database can locate a record in two ways: either by specifying the key, or by specifying the dynamic logical "record number" – the relative location of the record in the database, as in the line numbers of a text file.

After populating the database with 1000 key-value pairs, it then switches to a query phase, where you select records not by the key, but by the automatically assigned, dynamically changing record number.

Options: none

Language	Example Source or Project File
C	c/ex_btrec.c
C++	cxx/BtRecExample.cpp
C#	csharp/excs_btrec/excs_btrec.csproj
Java	java/src/db/BtRecExample.java

Comparing the characteristics of btrees and heaps – ex_heap

This program demonstrates some differences between the heap and btree database types.

It starts by populating the database, and then switches into a phase of adding and removing data while keeping a fairly constant amount of data in the database. A DB_HEAP maintains a constant database size if the heap size is configured properly, while the btree database may continue to grow.

Options:

-b Create a btree database in addition to a heap database.

-c <int> Override the default cache size for the environment.

-d Test on variable-length data (default: fixed-length).

-h <dir> Specify the home directory for the environment (required).

-n <int> Specify the number of records per repetition (default: 10,000).

-p <int> Set the pagesize for the database (default: filesystem block size).

-r <int> Set the repeat count: the number of insert/delete pairs per record (default: 1).

-S <int> Set the gigabyte portion of the maximum heap database size (default: no limit).

-s <int> Set the byte portion of the maximum heap database size (default: no limit).

Language	Example Source or Project File
C	c/ex_heap.c

Using sequences to automatically generate item identifiers – ex_sequence

This shows how a DB_SEQUENCE provides a stream of increasing or decreasing 64-bit integers.

Options:

-r Remove the database, if it already exists (default: off, do not remove).

[<filename>] Set the database name to <filename> (default: sequence.db).

Language	Example Source or Project File
C	c/ex_sequence.c
C++	cxx/SequenceExample.cpp
C#	csharp/excs_btrec/excs_sequence.csproj
Java	java/src/db/SequenceExample.java

Larger Examples

Setting up a fully transactional environment – ex_env

This shows how to set up a Transactional Data Store environment. It shows how to customize several parameters before opening the environment.

Options:

-h <dir> Use <dir> as the home directory for the environment (default: TESTDIR).

-d <dir> Use <dir> as the database directory for the environment (default: DATA).

-l Lock the environment region files in memory, where supported by the operating system.

Language	Example Source or Project File
C	c/ex_env.c
C++	cxx/EnvExample.cpp
C#	csharp/excs_btrec/excs_env.csproj
Java	java/src/db/EnvExample.java

Multithreaded reads and writes – ex_thread

The ex_thread example demonstrates multithreaded access. It shows how to prepare and open the environment and database handles so that they can be safely shared by freely running threads. The example also demonstrates deadlock handling, which is nearly always needed by such applications.

Options:

-h <home> Specify the home directory for the environment (default: TESTDIR).

-n <int> Specify the number of records (default: 1,000).

-r <int> Specify the number of reading threads (default: 4).

-v Print verbose messages during processing (default: off).

-w <int> Specify the number of writing threads (default: 4).

Language	Example Source File
C	c/ex_thread.c

Using the bulk interface, secondary indexes, and subdatabases – ex_bulk

This demonstrates how to fetch and modify many records within a single call of DB->get(), DB->put(), and DB->del(). It also shows how to:

a) define a custom btree key comparison function, so that records are ordered as natural integers,

b) store two databases in a single Berkeley DB file, and

c) construct a secondary “index” database for quick record access via a component of the data field.

Options:

-c <int> Set the cachesize to <int> bytes (default:1000 * pagesize).

-d <int> Set the number of 'duplicates': additional data items per key (default: none).

-i <int> Set the number of read iterations (default: 1000000).

-n <int> Set the number of keys to insert (default: 1000000).

-p <int> Set the database pagesize (default: 65536).

-v Turn on verbose output (default: off).

-D Perform bulk deletes after inserts (default: off).

-I Just initialize an empty environment; do no inserts (default: off).

-R Perform bulk reads (default: off).

-S Perform bulk operation in secondary database (default: off).

Language	Example Source or Project File
C	c/ex_bulk.c
C++	cxx/BulkExample.cpp
C#	csharp/excs_btrec/excs_bulk.csproj
Java	java/src/db/BulkExample.java

Full Use Cases

Replicated stock quote server – ex_rep

Berkeley DB supports building highly available applications via replication groups, which contain a *master* environment and one or more read-only *clients.* Replicas may be on the same machine or connected by local or wide-area networks.

The replication example is a small stock quote server. There are two versions of the program: one uses Berkeley DB's Replication Manager support, and the other uses the underlying base replication API.

The file `examples/c/ex_rep/README` provides details about the replication examples.

Language	Replication Manager Source or Project File	Base Replication Source Code
C	c/ex_rep/mgr/rep_mgr.c	c/ex_rep/base/rep_base.c
C++	cxx/excxx_repquote/RepQuoteExample.cpp	none
C#	csharp/excs_repquote/excs_repquote.csproj	none
Java	java/src/db/repquote/RepQuoteExample.java	none

The TPC-B transaction processing benchmark – ex_tpcb

TPC-B is an early transaction processing benchmark that simulates bank transfers from one account to another. The program is first run in an initialization mode which loads the data. Subsequent runs in one or more processes perform a workload.

Database initialization (the -i flag) and running the benchmark (-n flag) must take place separately (i.e., first create the database, then run one or more copies of the benchmark). When running more than one TPC-B process, it is necessary to run the deadlock detector (db_deadlock), since it is possible for concurrent processes to deadlock.

Options:

- -a <int> Set the number of accounts per teller (default: 100,000).
- -b <int> Set the number of branches (default: 10).
- -c <int> Set the cache size in bytes (default: 4MB).
- -f Fast I/O mode; don't flush transactions to stable storage (default: off).
- -h <dir> Set the home directory. (default: TESTDIR).
- -i Initialize the environment and load databases (default: off, that is, run the workload).
- -n <int> Perform this many transactions (no default, it must be specified when not initializing).
- -S <int> Set the random number seed (default: the current time, in seconds since the epoch).
- -t <int> Set the number of bank tellers (default: 100).

Language	Example Source or Project File
C	c/ex_tpcb.c
C++	cxx/TpcbExample.cpp
C++ STL	stl/TpcbExample.cpp
C#	csharp/excs_btrec/excs_bulk.csproj
Java	java/src/db/TpcbExample.java

Using in-memory databases for high-speed message processing

Devices such as network routers and firewall appliances process large quantities of short-lived data. They need to quickly handle packets, yet do not need to keep them after processing. The message processsing examples show how to use in-memory instances of Berkeley DB queue databases in order to achieve high throughput message processing.

Example	Start exploring here:
Router with Firewall	BDB Learn More page: Firewall example.zip
Prioritized Message Processing	BDB Learn More page: Priority Message Processing.zip

Event processing using BDB SQL

These event processing examples show how to use the SQL API to Berkeley DB. Both examples concern physical events: vehicles passing through an a automated toll booth and potential customers passing near a business.

The toll booth system looks up the transponder or license plate information of the vehicle and adds a charge to its billing account, as well as quickly alerting law enforcement when detecting a vehicle that is on a “watch list”.

The location-based advertising example looks up the shopping preferences of a smart-phone owner and available promotions offered by stores near the phone. A relevant match causes an electronic coupon to be sent to the phone while the potential customer is still nearby.

Example	Start exploring here:
An Automated Toll Booth	BDB Learn More page: Automated Toll Booth
Location-Based Advertising	BDB Learn More page: Personalized Advertising

C# application: inventory management

This example is a simple inventory management system which uses the C# interface to Berkeley DB. It supports adding, changing, and removing inventory items as well as managing and analyzing sales records.

Example	Start exploring here:
C#: Inventory Management	BDB Learn More page: “Inventory Management System”

Java application: a parking lot ticketing system

This Java example simulates an automated parking lot kiosk which issues time-stamped tickets upon entering, and, when exiting, uses those tickets to calculate the correct fee. At the end of the virtual “day”, it generates a report summarizing the kiosk's activity.

Example	Start exploring here:
Java: Automated Parking Lot	BDB Learn More page: “Parking Lot Fees”

Medical imaging between a mobile app and Oracle Server – BDB-DICOM

This example shows a Windows Mobile application to manage local copies of images obtained from an Oracle Server instance. DICOM (Digital Imaging and Communications in Medicine) is a standard suitable for any kind of medical image: X-ray, CT scan, MRI, etc. This example includes the sample image files, Java middleware, and a C++ Windows Mobile 5 application.

Example	Start exploring here:
Windows Mobile 5 application	BDB Learn More page: “Medical Imaging/DICOM”

Specialized Applications

Using a memory pool to read files – ex_mpool

This example fills a plain file (not a database) with data and performs random reads of it through the MPOOL file interface. It displays the read throughput in blocks and megabytes per second. It could be used as the basis for code which preloads an important database into the Berkeley DB cache.

Options:

-c <int> Set the cache size to <int> pages (default: 50).

-h <int> Specify how many reads to perform (default: 10,000 'hits')

-k Keep the existing environment (default: remove it)

-n <int> Set the number of pages in the file (default: 50)

-p <int> Set the pagesize (default: 1024)

Language	Example Source or Project File
C	c/ex_mpool.c
C++	cxx/MpoolExample .cpp

Comma separated values – csv

The c/csv directory contains application helpers for dealing with comma separated values.

The program `csv_code` compiles a text description of the CSV fields into the corresponding C structure definition, while also generating C functions to read, print, and search for records based on the named fields.

The program `csv_load` can then load a CSV file of that format into a database.

Once the data has been loaded, `csv_query` can interactively query the database.

The file `examples/c/csv/README` provides details about the CSV processing suite.

Language	Example Sources
C	c/csv/*

Using the lock manager for non-Berkeley DB data – ex_lock

The lock manager is very flexible, and can protect more than just Berkeley DB databases. If your application suite consistently makes read and write requests to the lock manager, it can manage access to items outside of the domain of Berkeley DB, such as URLs or read-world items.

This example uses read and write locks to control access to any kind of named object. To see how this works, run two or more instances of this interactive program, in separate terminal emulator windows. By giving (object-name, lock-mode) pairs to the prompts you can observe when the requests are granted immediately or when they are delayed until conflicting locks are released.

Options:

-h <dir> Set the home directory (default: TESTDIR).

-m <int> Set the number of locks to allocate (default: let BDB reserve the default allocation).

-u Unlink the environment if one already exists (default: do not remove it).

Language	Main Source Code or Project File
C	c/ex_lock.c
C++	cxx/LockExample.cpp
C#	csharp/excs_lock/excs_lock.csproj
Java	java/src/db/LockExample.java

Extending the transaction system for your application – ex_apprec

It is possible to add your own record types to the transaction log, and register functions to be called when those records are backed out during DB_TXN->abort() or processed during database recovery. This example adds support for a transactionally-protected “make directory” operation.

Language	Example Source File
C	c/ex_apprec/*