Oracle Berkeley DB Concurrent Data Store

A Use-Case Based Tutorial

Part II: Concurrent Data Store

- Overview
  - What is CDS?
  - When is CDS appropriate?
- Case Studies
  - Managing music
  - Storing XML

Concurrent Data Store

- Concurrency with reads and writes.
- Deadlock-free
- API-level locking
- What CDS does **not** do:
  - Recovery
  - Transactions
  - Replication

When to use CDS

- Read-mostly workload.
- Single-writer at a time is OK.
- Transient data (e.g., caching).
- No recovery after unclean shutdown.

CDS Programmatically

- Setting up your environment
- Write cursors
- Locking for complex applications

Setting up your Environment

- Environment **required** for CDS.
- Specify CDS on DB_ENV->open call.

```
/* Create handle. */
ret = db_env_create(&dbenv, 0);
    … error_handling …

/* Create and open the environment. */
flags = DB_CREATE |DB_INIT_CDB;

ret = dbenv->open(dbenv, HOME, flags, 0);
 … error_handling …
```

CDS Locking

- CDS acquires locks at the API level.
- By default, CDS locking is per-database.
  - Multiple readers in a single-database, OR
  - Single writer in a database.
- Applications can both read and write via cursors, so they need to be handled specially.

Write Cursors

- CDS deadlock free.
- Must avoid upgrading readlocks to writelocks.
- No upgrades on DBP operations.
- Cursors perform both reads and writes.
- Not all cursors write.
- Cursors that **may** write must be created with the DB_WRITECURSOR flag.

Creating a write cursor (code)

```
DBC *dbc = NULL;
ret = dbp->cursor(dbp, NULL, &dbc, DB_WRITECURSOR);
if (ret != 0)
    … error_handling …
```

Multiple Databases

- Normally CDS locks per-database.
- If applications maintain open cursors on one database while operating on another database, concurrent operations can deadlock.
- DB_CDB_ALLDB locks per-environment.

DB_CDB_ALLDB

- Big hammer solution.
- Performs locking on an environment-wide basis.
  - Only one write cursor in the entire environment.
  - No open read cursors in environment while writing.
- Simple, but …
- Low concurrency in presence of writes.
- Appropriate for applications whose operations touch multiple databases.

Configuring DB_CDB_ALLDB

```
/* Create handle. */
ret = db_env_create(&dbenv, 0);
    … error_handling …

/* Turn on environment-wide locking. */
ret = dbenv->set_flags(dbenv, DB_CDB_ALLDB, 1);

/* Create and open the environment. */
flags = DB_CREATE |DB_INIT_CDB;

ret = dbenv->open(dbenv, HOME, flags, 0);
 … error_handling …
```

Multi-threaded CDS

- Normally a locker-id is allocated per database.
- In a multi-threaded process, locker IDs are allocated per cursor.
- Multiple threads of control (in the same process) accessing multiple databases can deadlock.
- `cdsgroup_begin` allocates a locker-id for a thread of control.

cdsgroup_begin

- Assigns a locker to a set of accesses called a **group** (like a transaction).
- Allows multiple write-cursors in a group.
- Allows open read cursor in a group during a write in the same group.
- Prevents deadlocks between groups.
- Requires `DB_CDB_ALLDB`.
- Appropriate for multi-threaded applications that use multiple databases simultaneously.

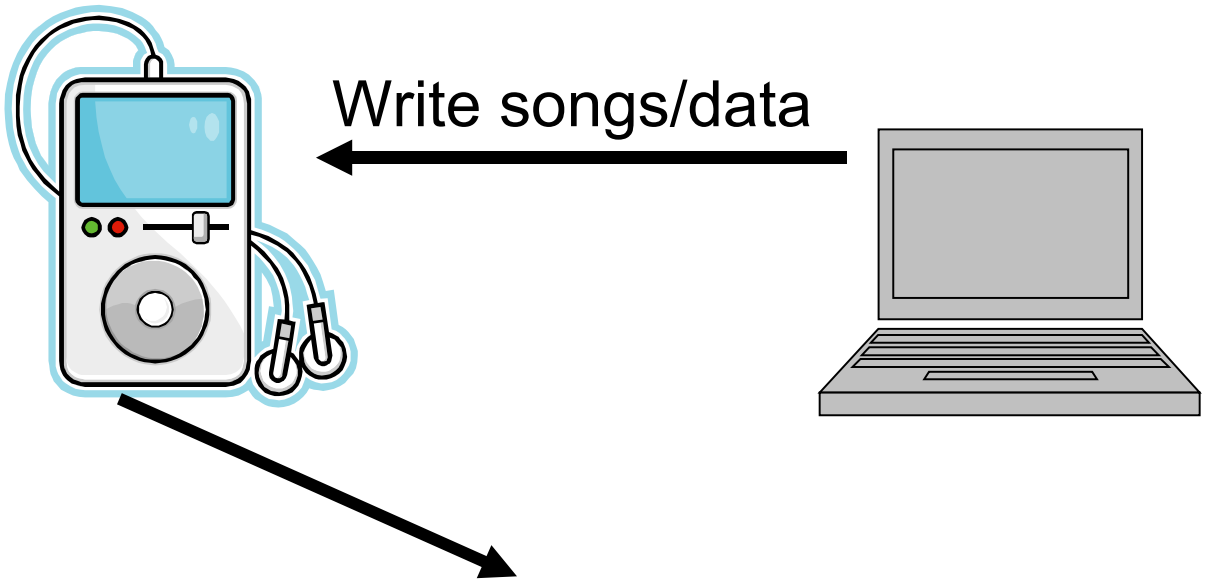
Using cdsgroup_begin

```
/*
 * Assume open dbenv (with DB_CDB_ALLDB) and two
 * open DBPs.
 */

DB_TXN *gid;
DBC *dbc1, *dbc2;

ret = dbenv->cdsgroup_begin(dbenv, &gid);
if (ret != 0)
    … error handling …
dbp1->cursor(dbp1, gid, dbc1, DB_WRITECURSOR);
dbp2->cursor(dbp2, gid, dbc2, DB_WRITECURSOR);

/* Perform operations on dbp1, dbp2, dbc1 & dbc2. */

/* End the group of operations. */
gid->commit(gid, 0);
```

Use Cases

- Managing a music database.
- Storing and Indexing XML.

Managing a Music Database



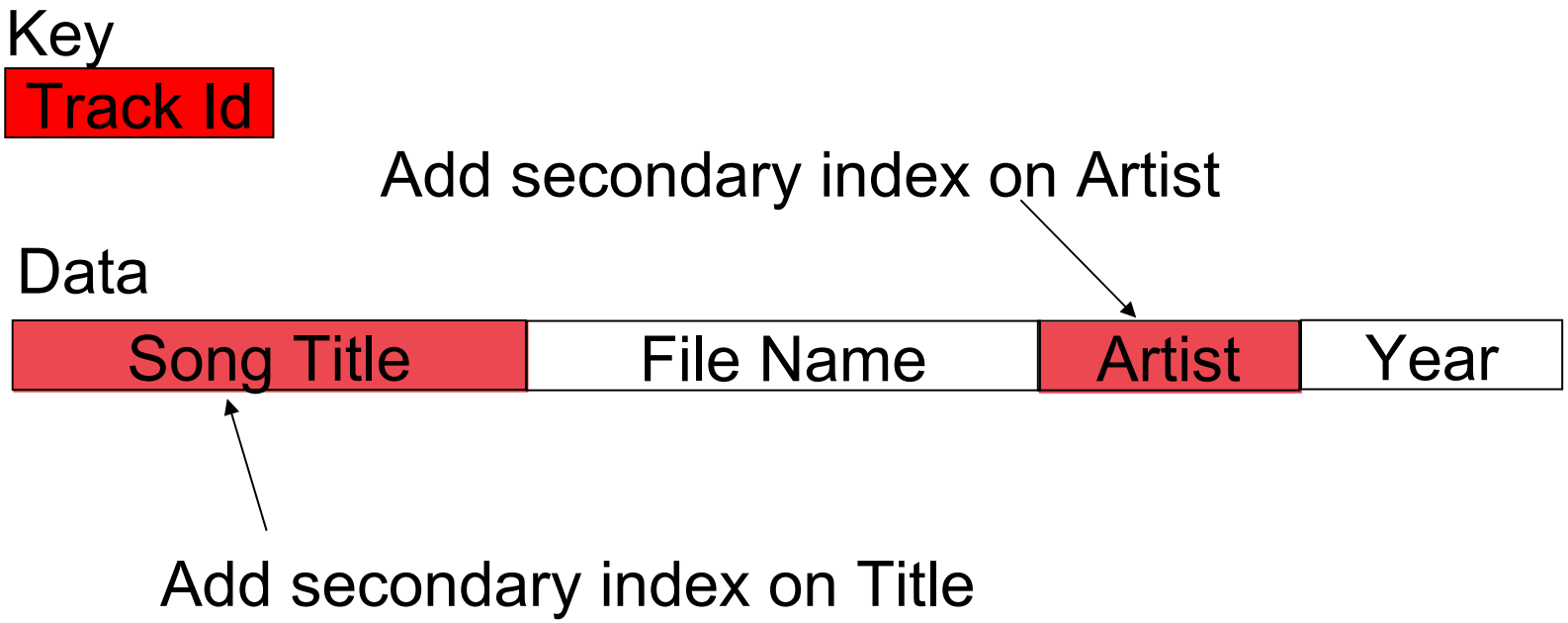
Music Data

- Songs have:
  - A unique ID
  - A (possibly non-unique) title
  - An artist
  - A year in which it was published
  - One or more albums on which it appeared
  - An MP3 containing the actual song

Song Queries

- Play song by title
- Find songs by artist
- Find songs by album
- Find song appearing on the most albums
- Updates
  - Add/remove song
  - Add/remove album

Primary Data & Indices



Processing Queries

- Play song by title
  - Secondary access by title, extract file name, play file.
- Find songs by artist
  - Secondary access by artist, iterate over all duplicates
- Find songs by album
  - Create album database; iterate using secondary with multiple
- Find the song that appears on the greatest number of albums
  - Create secondary on album; iterate, count, and max
- Updates
  - Add songs
  - Remove songs
  - Add album
  - Update album/artist information

Play Song by Title

```
1.   /* Assume title secondary database open as sdbp. */
2.   char *filename;
3.   int ret;
4.   DBT datadbt, keydbt;

6.   memset(&datadbt, 0, sizeof(datadbt));
7.   memset(&keydbt, 0, sizeof(keydbt));
8.   key->data = “Let it be”;
9.   key->size = strlen((char *)key->data) + 1;
10.  if ((ret = sdbp->get(sdbp, NULL, &keydbt, &datadbt, 0)) != 0)
11.      return (ret);
12.  filename = extract_file_from_record(datadbt->data);
13.  play_song(filename);
```

Retrieve by Artist

```
1.   /* Assume artist secondary database open as sdbp. */
2.   char *filename;
3.   int ret;
4.   DBT datadbt, keydbt;
5.   DBC *dbc;

7.   memset(&datadbt, 0, sizeof(datadbt));
8.   memset(&keydbt, 0, sizeof(keydbt));
9.   key->data = “Beatles”;
10.  key->size = strlen((char *)key->data) + 1;
11.  If ((ret = sdbp->cursor(sdbp, NULL, &dbc, 0)) != 0)
12.       goto err;
13.  for (ret = dbc->get(dbc, &keydbt, &datadbt, DB_SET)/
14.         ret == 0;
15.         ret = dbc->get(dbc, &keydbt, &datadbt, DB_NEXT_DUP))
16.                 filename = extract_file_from_record(datadbt->data);
```

The Album Database

- An album identifies a collection of songs.
- Representation choices:
  - Single key/data pair per album
    - Album names are unique keys
    - Data item is a collection of track IDs
  - Multiple key/data pairs per album
    - Album is a non-unique key
    - Each data item represents a single track
  - Key-only representation
    - Key is Album/Song
    - Data is empty
- How do you select the representation?

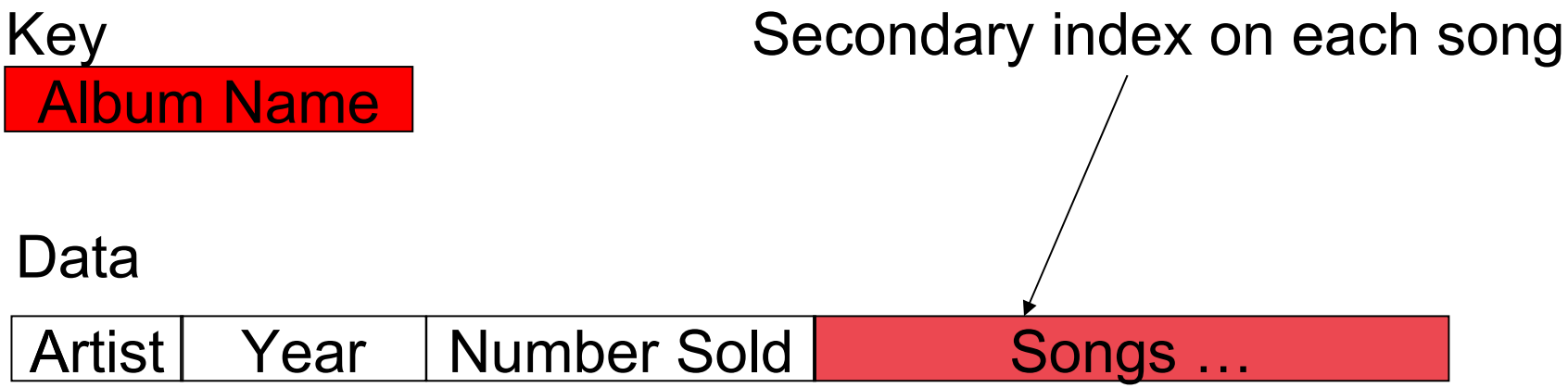
Selecting Album Representation (1)

- Carefully review queries
  - Find songs by album -- easy either way.
  - Find song that appears on the greatest number of albums -- slow unless you have a secondary index.
- A primary database with secondaries, **must** have a unique primary key.
  - Single-item per album OR
  - Key-only representation

Selecting Album Representation (2)

- Single-item per album
  + Compact
  + Easy to store other album data (e.g., year).
  - Application must parse data
- Key-only representation
  - Application must figure out end of album
  - Not obvious where to store album-wide data (could encode is as a “special” track ID).
- We’ll pick single-item per album.

Album Data


Key
Album Name
Secondary index on each song
Data
Artist
Year
Number Sold
Songs …

Album Secondary

- Map song to album(s).
- Need a callback function

```
3.   song_callback(DB* dbp, DBT *key, DBT *data, DBT *result){
4.       char *songs;

6.       /* Extract songlist from data DBT. */
7.       songs = songs_from_data(data);
8.       result->size = num_songs(songs);
9.       result->data = malloc(result->size * sizeof(DBT));
10.      result->flags |= DB_DBT_MULTIPLE | DB_DBT_APPMALLOC;
11.      foreach song in songs
12.              place song in DBT in result array
13.  }
```

Updates

- Two queries:
  - Add song
  - Remove song
- Design questions:
  - What happens when you remove a song that belongs to an album?
  - How do you coordinate adding songs and albums?

Add a song

- If song doesn’t exist, no need to worry about its existence in an album.
- So, this is a simple add (BDB updates the secondaries automatically)

```
3.   DBT keydbt, datadbt;
4.   memset(&data, 0, sizeof(data));
5.   memset(&key, 0, sizeof(key));
6.   key->data = “123456”;
7.   key->size = strlen(key->data) + 1;
8.   data->data = data_to_bytestring(song_data);
9.   data->size = sizeof(data_to_bytestring(song_data));
10.  dbp->put(dbp, NULL, &keydbt, &datadbt, 0);
```

Remove a Song

- If the song belongs to an album, then we've got multiple databases to update.
- This is the kind of update that requires use of DB_CDB_ALL.

Removing a song

- Look up song by title (get song ID).
- Remove song from song database.
- Create cursor on album-by-song secondary
- Foreach album containing song
- Remove song from album (requires writing album)
- Close album cursor

Remove song (code)

```
1.  DB       *sbtdbp; /* song-by-title index handle */
2.  DB       *songs;  /* song database handle */
3.  DBT keydbt, pkeydbt, datadbt;

5.  /* 1. Look up primary key of song. */
6.  memset(&datadbt, 0, sizeof(datadbt));
7.  memset(&keydbt, 0, sizeof(keydbt));
8.  memset(&pkeydbt, 0, sizeof(pkeydbt));
9.  key->data = "Time in a bottle";
10. key->size = 17;
11. sbtdbp->pget(dbc, &keydbt, &pkeydbt, &datadbt, DB_SET);

13. /* 2. Now delete song. */
14. songs->del(songs, &pkeydbt, 0);
```

Remove Song (code)

```
DBP *absdbp;      /* Album-by-song secondary handle */
DBC *adbc = NULL;
DBT akeydbt;      /* Holds primary key of album. */

/* 3. Create cursor on album-by-song secondary. */
memset(&datadbt, 0, sizeof(datadbt));
memset(&akeydbt, 0, sizeof(keydbt));
absdbp->cursor(absdbp, NULL, &adbc, 0);

/*
 * Opening a cursor requires a READ lock on both the
 * secondary index AND the primary database (albums).
 */
```

Remove Song

```
/* 4. Foreach album containing song. */
while (ret =
    adbc->pget(adbc, &pkeydbt, &akeydbt, &datadbt, DB_SET;
    ret == 0; ret = adbc->get(adbc,
    &keydbt, &akeydbt, &datadbt, DB_NEXT)) {

        /* 5. Remove the song from this album. */
        … remove song ID from song array in datadbt->data …
        ret = album->put(album, &akeydbt, &datadbt, 0);
}

/* 6. Close cursor. */
adbc->close(adbc, 0);
```

Why we need DB_CDB_ALL

- The cursor that iterates over the album secondary will acquire read locks on both the secondary index and the primary database.
- When we update the album data, we need to write lock the primary database.
- Without DB_CDB_ALL, these calls will deadlock.
- With DB_CDB_ALL, we acquire a single lock that protects all the databases.

Use Cases

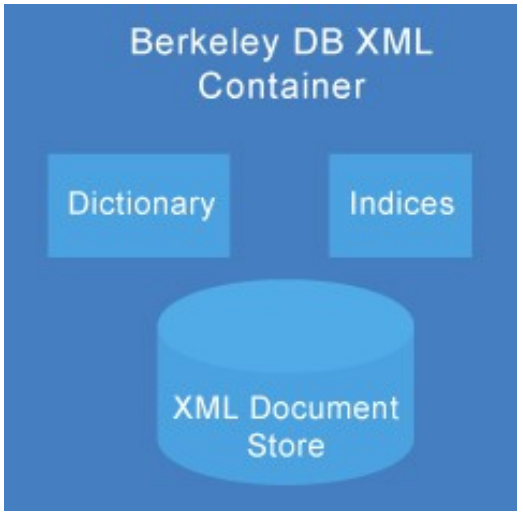
- Managing a music database.
- Storing and indexing XML.

BDB XML

- Berkeley DB XML is built atop Berkeley DB.
- Can configure DBXML to use CDS.
- Let’s look at how XML uses BDB and the implications of CDS locking.

XML Containers

- A container is a file containing multiple databases:
  - A document database
  - Index databases
  - Index statistic database
- Operations on containers:
  - Put/get documents
  - Update document
  - Add/remove indexes
  - Query using XQuery
  - Administration: create, remove, rename …



DbXML Internal Basics

- A default index provides keyed access to all tags in all documents.
- XML references all tags by ID number.
- Dictionary maps from tag-name to ID number.
- Most operations touch multiple databases.
- When using CDS, must use DB_CDB_ALL.
- Frequently used multi-threaded.

Add Default Index

- For each document in the container:
  - Create an event reader (cursor) for the document
  - Iterate over the document's nodes
    - If node tag, not in dictionary, add to dictionary
    - Collect index information into “keystash”
  - Close event reader (cursor)
  - Take all items in keystash and add to index.

Add Default Index with cdsgroup_begin

- For each document in the container:
  - Start cdsgroup.
  - Create an event reader (cursor) for the document.
  - Iterate over the document's nodes
    1. If node tag, not in dictionary, add to dictionary
    2. Collect index information into “keystash”
  - Close event reader (cursor)
  - End cdsgroup.
  - Take all items in keystash and add to index.

Coding with cdsgroup_begin

```
/*
 * Assume that we have the document DB (doc_dbp) and
 * dictionary DB (dict_dbp) open.
 */
DB_TXN *tid;
DBC *dbc;
DBT keydbt; datadbt;

memset(&keydbt, 0, sizeof(keydbt));
memset(&datadbt, 0, sizeof(datadbt));

/* 1. Start cds group. */
ret = dbenv->cdsgroup_begin(dbenv, &tid);

/* 2. Create cursor on document database.
ret = doc_dbp->cursor(doc_dbp, tid, &dbc, 0);
```

cdsgroup_begin (cont)

```
/* 3. Iterate over nodes in document.
while (ret = dbc->get(dbc, &keydbt, &datadbt, DB_FIRST);
    ret == 0;
    ret = dbc->get(dbc, &keydbt, &datadbt, DB_NEXT)) {
       /* 4. Enter node in dictionary. */
       data_to_node(&datadbt, nodename, nodeid);
       ret = dict_dbp->put(dict_dbp, tid, nodename, nodeid, 0);
       /* 5. Save index info into keystash. */
}

/* 6. Close cursor. */
dbc->close(adbc, 0);
/* 7.End group. */
tid->commit(tid, 0);
```

Concurrent Data Store: What we Learned

- What is concurrent data store (CDS)?
- When to use CDS.
- Updating data with CDS.
- Using DB_CDB_ALL and cdsgroup_begin.
- (More) Use of secondary indices.

End of Part II