Assignment / Investigate Database Recovery

Connor Clancy - clancy.co@northeastern.edu

Spring 2023

Question 1 - Recovery Scenario

For this scenario, lets say we are working at e-commerce company that sells furniture online. This company keeps track of their inventory in a relational database. Since this is an e-commerce company, and multiple customers can be completing purchase at the same time, concurrency is supported within the database. Additionally, since customers cannot effectively browse the website or complete purchases if the inventory database is offline, we must reduce the time take for recovery if the database fails for some reason.

To ensure the continuity of their business, and the stability of their systems, this company has chosen to use the **ARIES Recovery Algorithm** because of its simple and flexible implementation, support for concurrency controls, and because it reduces overhead and recovery time.

The algorithm works by using **log sequence numbers (LSN)** and **pageLSN** to organize log transactions. The most recent portion of the log is kept in main memory and is called the **log tail**. A log record is created whenever a a page is updated, a transaction is committed or aborted, an update is undone, or a transaction is ended. This algorithm also utilizes **checkpoints** (in fuzzy form to reduce overhead) that can be accessed and utilized to speed up recovery efforts.

By using the ARIES Recovery Algorithm, this company can ensure that they will have accurate, ACID compliant, database transactions, and a quick recovery time across their concurrent systems, allowing them to minimize downtime and loss of revenue in the event of a database failure resulting in the need for a database recovery.

Question 2 - Database Transactions

For this assignment, I have chosen to use the **Sakila** database from last weeks assignment.

R Code Setup

```
library(RSQLite)
library(sqldf)
```

Libary Imports

```
## Loading required package: gsubfn
```

```
## Loading required package: proto
```

```
options(sqldf.driver = "SQLite")
```

```
fpath = paste0(dirname(getwd()), "/CS5200.ExploreQueries.Clancy/")
dbfile = "sakila.db"
```

```r
# connect to the database if exists, else create a new database
lcon <- dbConnect(RSQLite::SQLite(), paste0(fpath, dbfile))
```

```r

dbGetQuery(lcon, "SELECT film_id, title FROM film LIMIT 5;")
```

Connect to SQLite Database

```
##   film_id            title
## 1       1 ACADEMY DINOSAUR
## 2       2   ACE GOLDFINGER
## 3       3 ADAPTATION HOLES
## 4       4 AFFAIR PREJUDICE
## 5       5      AFRICAN EGG
```

Write R Transaction Function This function allows us to add an exiting film to the inventory of a store. In order for the transaction to be successful, the `film_id` and the `store_id` have to exist in their respective tables. At the time of writing this notebook, there are only 1000 valid films and 2 stores in the database.

```r
#' Method to add inventory to the sakila database. This function will only add
#' new inventory to the database if a valid film_id and store_id are provided to
#' the function.
#'
#' @param film_id a valid film_id from the FILM table
#' @param store_id a valid store_id from the STORE table
#' @return TRUE if the the transaction is committed; FALSE if rollback
#'
addInv <- function(dbcon, film_id, store_id) {

  # create a default variable at the beginning of the function.  If this ever
  # gets changed to TRUE during execution, the transaction will fail.
  txnFailed = FALSE

  dbExecute(dbcon, "BEGIN TRANSACTION")

  # check film_id
  film_count <- dbGetQuery(lcon,
                           "SELECT
                            COUNT(film_id) AS films
                            FROM film WHERE film_id = ?",
                           params = list(film_id))

  if(film_count$films < 1) {
    txnFailed = TRUE
  }

  # check store_id
  store_count <- dbGetQuery(lcon,
                           "SELECT
                            COUNT(store_id) AS stores
                            FROM store WHERE store_id = ?",
                           params = list(store_id))

  if(store_count$stores < 1) {
    txnFailed = TRUE
  }

  # insert statement for new inventory data
  sql <- "INSERT INTO inventory (film_id, store_id, last_update)
          VALUES (?, ?, date('now'))"
  ps <- dbSendStatement(dbcon, sql, params=list(film_id, store_id))

  if(dbGetRowsAffected(ps) < 1) {
    txnFailed = TRUE
  }
  dbClearResult(ps)

  # commit the transaction if all checks have passed, else rollback
  if(txnFailed == TRUE) {
    dbExecute(dbcon, "ROLLBACK TRANSACTION")
  } else {
    dbExecute(dbcon, "COMMIT TRANSACTION")
  }

  # return status; TRUE if successful; FALSE if failed
  return(!txnFailed)
}
```

Part I - Sucessful Transaction

```r
status <- addInv(lcon, 1, 1)

if(status == TRUE) {
  cat('Success')
} else {
  cat('Failed')
}
```

```
## Success
```

Part II - Failed Transaction

```r
status <- addInv(lcon, 1005, 5)

if(status == TRUE) {
  cat('Success')
} else {
  cat('Failed')
}
```

```
## Failed
```