Investigate Database Recovery

Q1. Choose a database recovery problem or scenario (perhaps from work) and then propose a solution using the techniques described in Chapter 11 in the textbook. Briefly describe the technique, when it is appropriate to use and what recovery problem it solves.

Briefly describe the technique—Shadow paging

Shadow paging is a technique that makes copy of the database whenever a change is made, and it's hard to be extended for the use of concurrent transactions. This technique uses two page tables named current page table and shadow page table: the current page table is used to point the most recent database pages on disk; while the shadow page table is used to store the old state of database. When the transaction starts, the shadow page table copies the data from the current page table and then saves them on the disk. Entries present in current page table may be changed during execution but shadow page table will never get changed during the execution.

When to use

Since the shadow paging is a simple and straightforward recovery technique and has disadvantages such as data fragmentation and garbage collection, it is suitable for simple database systems, read-mostly database systems or databases where updates are infrequent.

Scenario and how to solve the recovery problem using this technique

Suppose we have a small e-commerce website, and most of the transactions are related to reading rather than updating.

In this scenario, shadow paging technique could be used. When a user makes a purchase, a transaction is initiated to update the relevant tables in the database. During the transaction, a shadow page table is created to record the original state of the database. Once the transaction is successfully committed, the current page table is updated with the new state resulting from the transaction, while the shadow page table remains unchanged.

If there is a system failure during execution of transaction but before commit operation, the current page table is discarded and the disk blocks containing the modified disk pages are released. The original state of the database prior to the transaction execution can be recovered from the shadow page table.

In case no failure occurs and the transaction is committed, the shadow page table is discarded and the disk blocks with the old data are released.

Citation

ITL Education Solutions Limited. Introduction to Database Systems. Pearson India, November 2008. O'Reilly Online Learning, https://learning.oreilly.com/library/view/introduction-to-database/9788131731925/.

Q2. Using any of the SQLite database we have previously worked with, write an update that requires related modification of multiple tables and conduct those updates within a transaction. Test the code so that you show that the transaction works and write one test where the transaction fails and rolls back.

connect db

```
library(RSQLite)
library(DBI)
# connect to sqlite db
wd = getwd()
fpath = paste0(wd, "/")
dbfile = "Recovery.db"

DB_Con <- dbConnect(RSQLite::SQLite(), paste0(fpath, dbfile))
```

drop tables

```
DROP TABLE IF EXISTS accounts;
```

```
DROP TABLE IF EXISTS history;
```

create tables

```
CREATE TABLE IF NOT EXISTS accounts (
    a_id INTEGER NOT NULL PRIMARY KEY,
    balance DECIMAL NOT NULL DEFAULT 0,
    CHECK(balance >= 0)
);
```

```
CREATE TABLE IF NOT EXISTS history (
    h_id INTEGER PRIMARY KEY AUTOINCREMENT,
    a_id INTEGER NOT NULL,
    flag TEXT NOT NULL,
    amount DECIMAL NOT NULL,
    change_date DATE NOT NULL,
    FOREIGN KEY (a_id) REFERENCES accounts(a_id)
);
```

insert data

```
INSERT INTO accounts (a_id, balance) VALUES
  (1, 1000),
  (2, 2000);
```

create a function for executing transactions

```
transferMoney <- function (dbcon, src_id, dest_id, amount)
{
  transFailed = FALSE
  # can't transfer to same account
  if (src_id == dest_id)
    return (FALSE)

  # begin transaction
  dbExecute(dbcon, "BEGIN TRANSACTION")

  # update src account
  sql <- "UPDATE accounts
            SET balance = balance - ?
            WHERE a_id = ?"
  # Pass one set of values directly using the param argument:
  ps <- dbSendStatement(dbcon, sql, params = list(amount, src_id))

  # no such account
  if (dbGetRowsAffected(ps) < 1)
    transFailed = TRUE
  # clear up
  dbClearResult(ps)

  # update dest account
  sql <- "UPDATE accounts
            SET balance = balance + ?
            WHERE a_id = ?"
  ps <- dbSendStatement(dbcon, sql, params = list(amount, dest_id))
  if (dbGetRowsAffected(ps) < 1)
    transFailed = TRUE
  dbClearResult(ps)

  # add transaction to history
  sql <- "INSERT INTO history (a_id, flag, amount, change_date)
           VALUES (?, 'decrease', ?, date('now'))"
  ps <- dbSendStatement(dbcon, sql, params = list(src_id, amount))
  if (dbGetRowsAffected(ps) < 1)
    transFailed = TRUE
  dbClearResult(ps)

  sql <- "INSERT INTO history (a_id, flag, amount, change_date)
           VALUES (?, 'increase', ?, date('now'))"
  ps <- dbSendStatement(dbcon, sql, params = list(dest_id, amount))
  if (dbGetRowsAffected(ps) < 1)
    transFailed = TRUE
  dbClearResult(ps)


  # commit transaction if no failure, otherwise rollback
  if (transFailed == TRUE)
    dbExecute(dbcon, "ROLLBACK TRANSACTION")
```

```
  else
    dbExecute(dbcon, "COMMIT TRANSACTION")

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

test valid transaction: account 1 tranfer 100 to account 2

```
src_id <- 1
dest_id <- 2
amount <- 100
status <- transferMoney(DB_Con, src_id, dest_id, amount)
if (status) {
  cat('Transfer succeeded')
} else {
  cat('Transfer failed')
}
```

```
## Transfer succeeded
```

inspect two tables

```
sql1 <- "SELECT * FROM accounts"
rs1 <- dbGetQuery(DB_Con, sql1)
rs1
```

```
##   a_id balance
## 1    1     900
## 2    2    2100
```

```
sql2 <- "SELECT * FROM history"
rs2 <- dbGetQuery(DB_Con, sql2)
rs2
```

```
##   h_id a_id     flag amount change_date
## 1    1    1 decrease    100  2023-03-21
## 2    2    2 increase    100  2023-03-21
```

test invalid transaction: account 2 tranfer 50 to account 3, while account 3 doesn't exist

```
src_id <- 2
dest_id <- 3
amount <- 50
status <- transferMoney(DB_Con, src_id, dest_id, amount)
if (status) {
  cat('Transfer succeeded')
```

```
} else {
  cat('Transfer failed')
}
```

```
## Transfer failed
```

inspect two tables

```
sql1 <- "SELECT * FROM accounts"
rs1 <- dbGetQuery(DB_Con, sql1)
rs1
```

```
##   a_id balance
## 1    1     900
## 2    2    2100
```

```
sql2 <- "SELECT * FROM history"
rs2 <- dbGetQuery(DB_Con, sql2)
rs2
```

```
##   h_id a_id     flag amount change_date
## 1    1    1 decrease    100  2023-03-21
## 2    2    2 increase    100  2023-03-21
```

disconnect db

```
dbDisconnect(DB_Con)
```