Assignment 2   
CPS610 – Database 2  
Wednesday Section – Group 13  
Member #1: Andy Lee (500163559)  
Member #2: Sohrab Soltani (500801172)

# Introduction

In this assignment, we will simulate and deploy a distributed database system and some other functionality such as serial transaction and locking.

The database scheme is a trimmed down version of Sohrab’s CPS510 project of an ecommerce store selling music albums. Instead of having three virtual machines running one instance on each virtual machine, we run only one virtual machine with 3 independent users. Since each user is isolated in its own environment (as an independent instance/table space), this equates and simulates running independent databases on each site. Each user (site) connects to one another through a database link. The three instances are: DB\_HQ for the central headquarters site, DB\_CAN for the Canadian operations satellite site, and DB\_US for the United States operations satellite site. This fulfills the bonus requirement of creating a distributed database other than the university example.

DB\_HQ stores the album information along with Canadian and US price in its database. DB\_CAN and DB\_US will store any purchase transactions from their own region respectively, pulling album data from HQ by reconstructing vertical fragmentation. DB\_HQ utilizes that CPS610\_Purchases\_HQ view; aggregating all the purchases from CAN and US; reconstructing a horizontal fragmentation.

The following pages will lay out steps to implement this distributed database.

Contents

[Introduction 1](#_Toc66472545)

[Step 1: Create Users (01 Create Users.sql) 3](#_Toc66472546)

[Step 2: Database for Headquarters (02 DB\_HQ.sql) 4](#_Toc66472547)

[Step 3: Database for Canada (03 DB\_CAN.sql) 5](#_Toc66472548)

[Step 4: Database for United States (04 DB\_US.sql) 6](#_Toc66472549)

[Step 5: Create View and Procedure for DB\_HQ and DB\_CAD 6](#_Toc66472550)

[Step 6: Populate dummy data to HQ (05 DB\_HQ – Populate.sql) 8](#_Toc66472551)

[Step 7: Populate dummy data to DB\_CAD (06 DB\_CAN Populate.sql) 9](#_Toc66472552)

[Step 8: Populate dummy data to DB\_US (07 DB\_US – Populate.sql) 10](#_Toc66472553)

[Step 9: Run procedure to achieve old orders (08 DB\_HQ – Serial Transaction.sql) 11](#_Toc66472554)

[Horizontal and Vertical Fragmentation Reconstruction 12](#_Toc66472555)

[Relational Algebra: CPS610\_Users\_HQ (View) 12](#_Toc66472556)

[Relational Algebra: CPS610\_ALBUMS\_CAN (View) 13](#_Toc66472557)

[Relational Algebra: CPS610\_ALBUMS\_US (View) 13](#_Toc66472558)

Table of Figures

[Figure 1 Creating instances 3](#_Toc66391323)

[Figure 2 Three instances with the three new users 3](#_Toc66391324)

[Figure 3 Creating database link and tables in DB\_HQ 4](#_Toc66391325)

[Figure 4 Creating database link and tables in DB\_CAN 5](#_Toc66391326)

[Figure 5 Code output from DB\_US. 6](#_Toc66391327)

[Figure 6 Code output from DB\_HQ 6](#_Toc66391328)

[Figure 7 Code output from DB\_CAN 7](#_Toc66391329)

[Figure 8 Locking table 7](#_Toc66391330)

[Figure 9 Album data inserted to DB\_HQ 8](#_Toc66391331)

[Figure 10 Insert users, purchase records, and modifying purchase on DB\_CAN. 9](#_Toc66391332)

[Figure 11 Insert users, purchase records, and modifying purchase on DB\_US. 10](#_Toc66391333)

[Figure 12 Procedure to archive old purchase records 11](#_Toc66391334)

# Step 1: Create Users (01 Create Users.sql)

This script creates 3 users (server1, server2, and server3) to simulate the three different sites. It should be run on a third user platform, such as system. Each of the users will be granted sufficient permission. Next in SQL Developer, create 3 instances as DB\_HQ, DB\_CAN, DB\_US, with these credentials (See Figure 1). There should be 3 instances after this is done (See Figure 2).

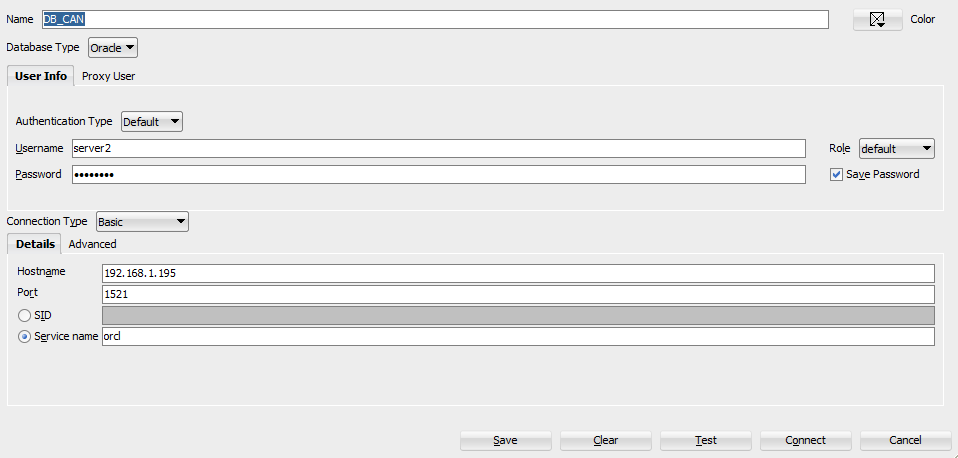


Figure Creating instances

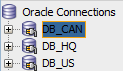


Figure Three instances with the three new users

# Step 2: Database for Headquarters (02 DB\_HQ.sql)

Inside the DB\_HQ instance, run only the script that creates database link, and tables. Do not run any “create view” code yet as the tables have not yet been created and will return an error, and also not the “create procedure”. See Figure 3.

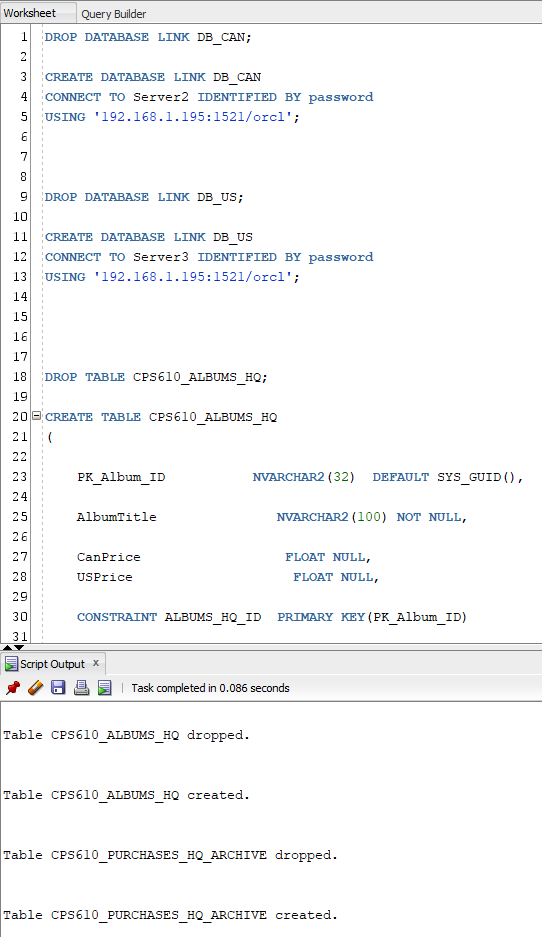


Figure Creating database link and tables in DB\_HQ

# Step 3: Database for Canada (03 DB\_CAN.sql)

Just like step 2, run only the portion of the code that creates database link, and new tables; ignoring the “create view”, and “create procedure” portion of the code. See Figure 4 for code output.

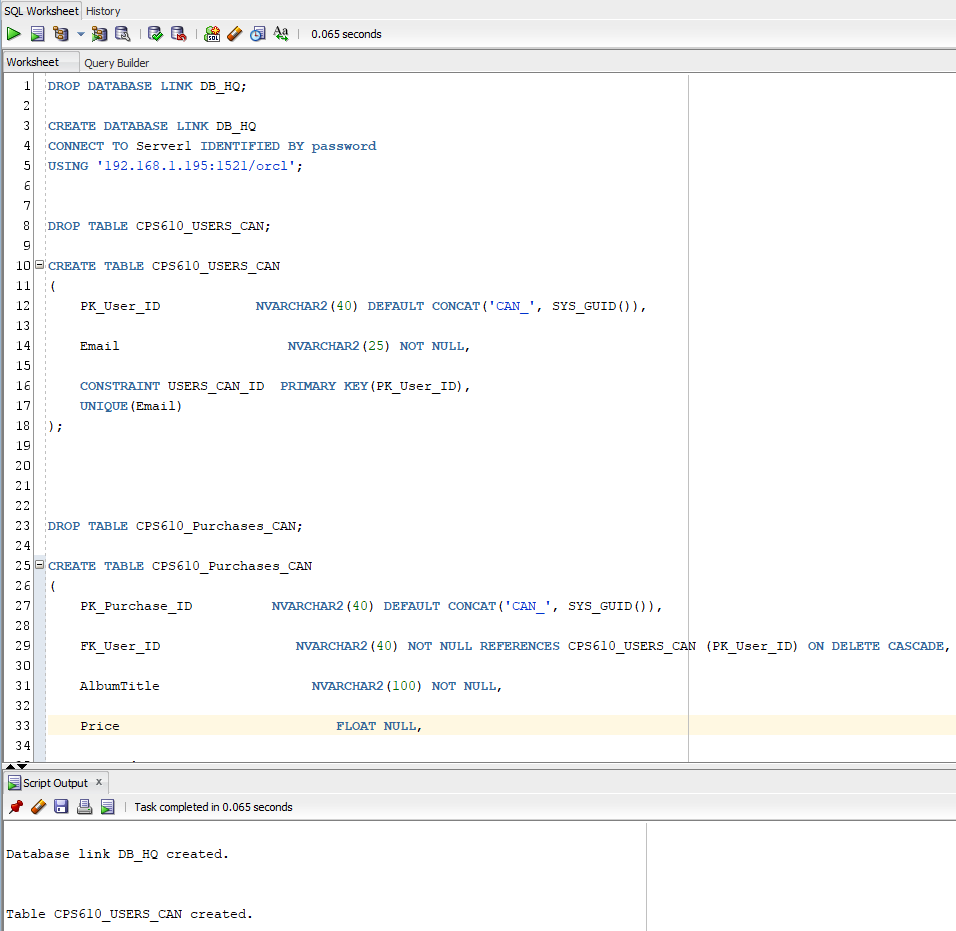


Figure Creating database link and tables in DB\_CAN

# Step 4: Database for United States (04 DB\_US.sql)

Just like steps 2 and 3, first run the portion of the code that creates database link, and new tables. But this time since all the tables needed have been created, run the “create view” and “create procedure”. See Figure 5 for code output.

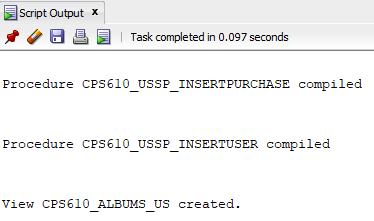


Figure Code output from DB\_US.

# Step 5: Create View and Procedure for DB\_HQ and DB\_CAD

Go back to DB\_HQ and run the “create view” and “create procedure”. Then go to DB\_CAN and run the “create view” and “create procedure”. The various “views” in these files fulfil the fragmentation reconstruction requirement for lab 5. See Figure 6 and Figure 7 for code output. The procedures of locking fulfil the requirement for lab 7; see Figure 8.

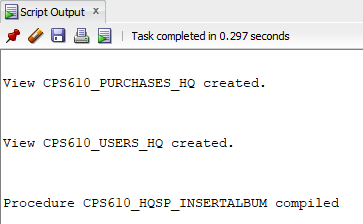


Figure Code output from DB\_HQ

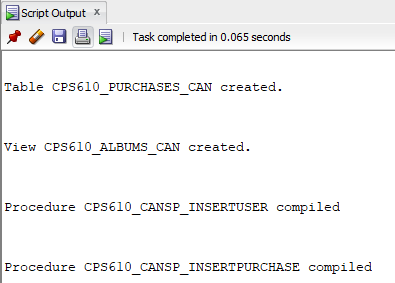


Figure Code output from DB\_CAN

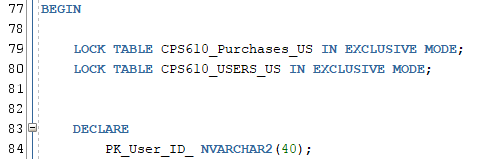


Figure Locking table

# Step 6: Populate dummy data to HQ (05 DB\_HQ – Populate.sql)

This step is to create some dummy data to work on. This inserts some albums with album name, Canadian price, and US price. This fulfills the population requirement for lab 5. See Figure 8 for code output.

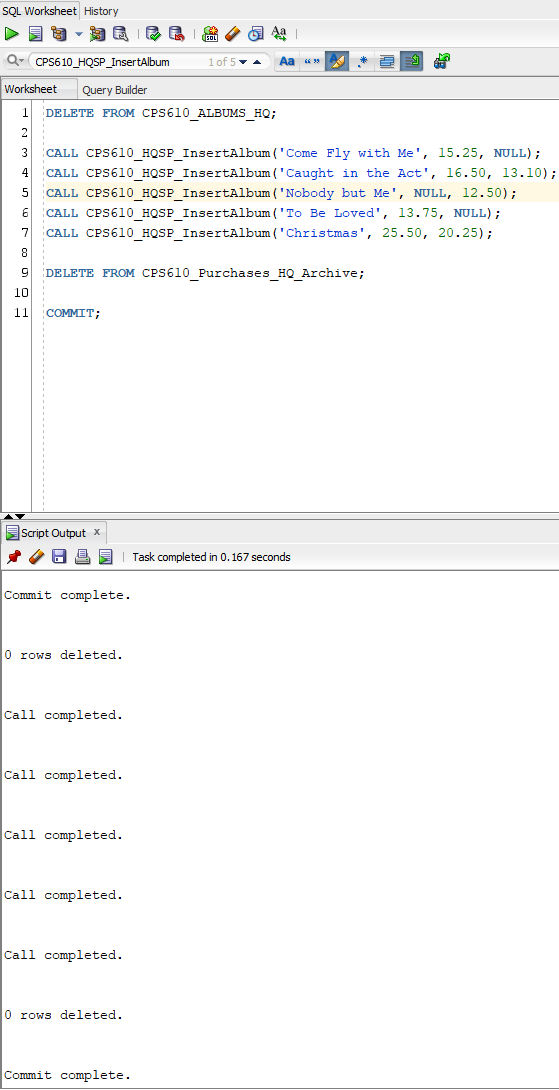


Figure Album data inserted to DB\_HQ

# Step 7: Populate dummy data to DB\_CAD (06 DB\_CAN Populate.sql)

This will first insert some users, based on user email, using the procedural code, then insert purchases based on the user’s email address and album name. It will also select [user3@example.ca](mailto:user3@example.ca) and change the purchase date of the order with “Christmas” to 370 days ago. This fulfills the population requirement for lab 5. See Figure 9 for code output.

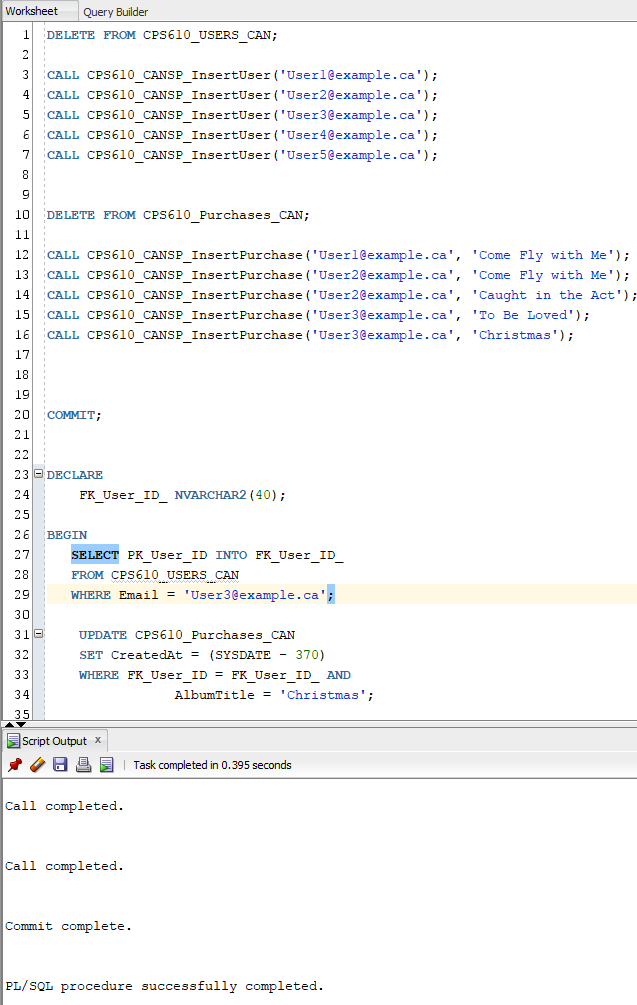


Figure Insert users, purchase records, and modifying purchase on DB\_CAN.

# Step 8: Populate dummy data to DB\_US (07 DB\_US – Populate.sql)

Same as step 7, inserting some fake users into the US database, and also some purchases made by these users. The modify [user5@example.us](mailto:user5@example.us) Christmas purchase to 370 days ago. This fulfills the population requirement for lab 5. See Figure 10 for code output.

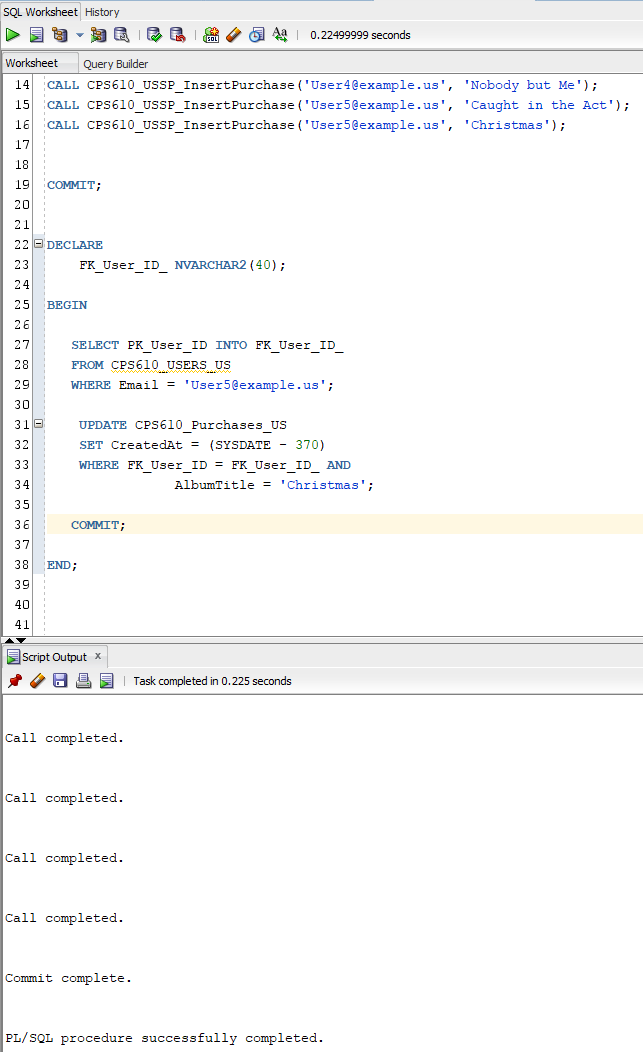


Figure Insert users, purchase records, and modifying purchase on DB\_US.

# Step 9: Run procedure to achieve old orders (08 DB\_HQ – Serial Transaction.sql)

This procedure first runs on the DB\_CAN; finding all purchases that are at least a year old and move them to the archive table in DB\_HQ. After moving the records, it will be removed from the purchase table in DB\_CAN. The same procedure will then run on the DB\_US. This fulfils the serial transaction requirement for lab 6. See Figure 11 for code output.

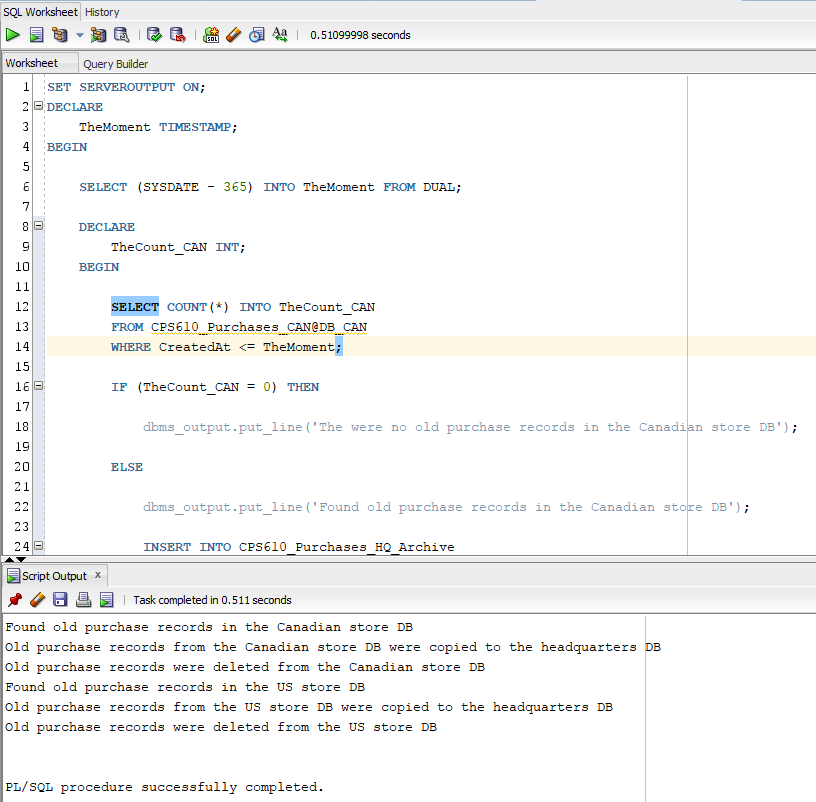


Figure Procedure to archive old purchase records

# Horizontal and Vertical Fragmentation Reconstruction

The CPS610\_Purchases\_HQ view showcases the reconstruction horizontal fragmentation and vertical fragmentation from the HQ based on data from 3 different sites. See below Figure 13.

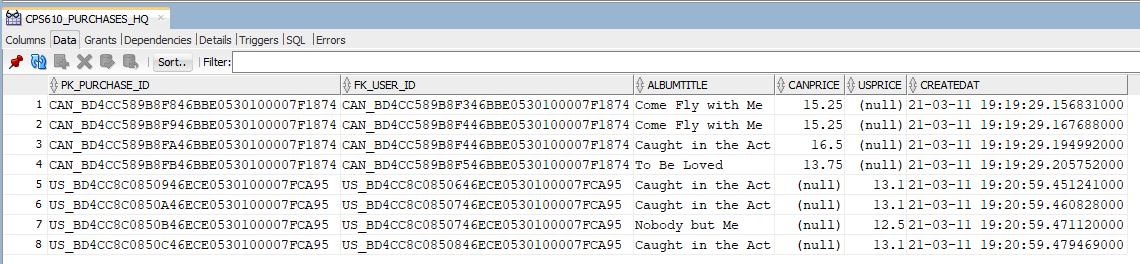
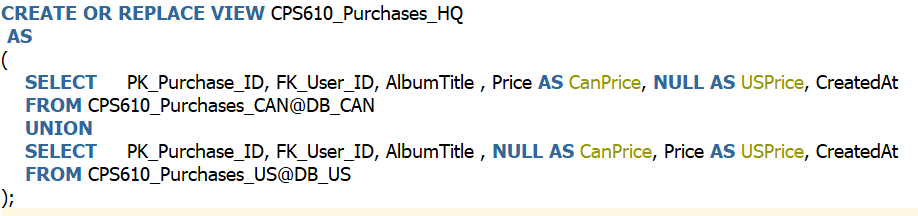
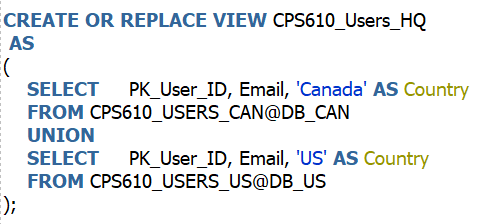


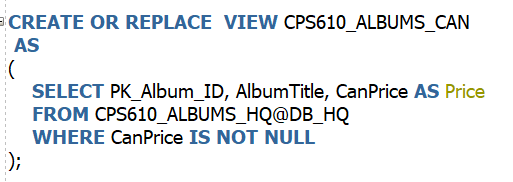
Figure 13 Horizontal and vertical fragmentation reconstruction  
  
Relational Algebra: CSP610\_Purchases\_HQ (View)



# Relational Algebra: CPS610\_Users\_HQ (View)



# Relational Algebra: CPS610\_ALBUMS\_CAN (View)



# Relational Algebra: CPS610\_ALBUMS\_US (View)

