CPS 510 Final Report
Retail Store Chain Management Database

Derek Lee
Ryan Soliven
George Saade

Proposal:
Retail clothing store database management system

Database that stores information about inventory, employees, working hours, significant dates, work timetables, and sale info.

We chose a clothing store because we found that what may seem a simple system with not much going on other than a couple of cashiers and clothes on some shelves, does indeed have more going on to it. A store chain management system would have an interesting and sophisticated system setup and database that brings the whole thing together. It starts from keeping track of inventory, sales, and making sure employees are paid on time. Further below will try to cover such requirements and describe them in a bit more detail.

This database for a clothing store will include entities such as (but not limited to):

1) Inventory. Name, type of clothes, brands, colours, genders, sizes, placing records of inventory orders.

2) Money flow. Store balance, cashier registers, transactions and transaction history/refunds.

3) Employees records. Log in info, emails, address, phone number, employee ID, schedules, hourly rate, commission, holiday hours.

4) Access levels. Who and how will access the database:

- Owner/admin (see everything)
- Manager
    - in store access
        - see employee schedules, create new hours to schedule, employee data, add/remove employees, create discounts, make inventory orders
    - extended employee privileges
- Employee
    - in store access only
        - make transactions, pull discount data for items, “clock in/out” to work/leave
    - out of store access
        - see hours, sign up for hours, own employee data

5) Customer records. Log in info, emails, address, phone number, payment info, past orders, cart, reward points, etc.

6) Order Records. Order id, the price it was sold, time and date of order, description of sale (online or in store), incomplete / complete orders, quantity, discounts applied.

7) Employee’s schedule. Set up a weekly schedule for workers based on their different available work hours.

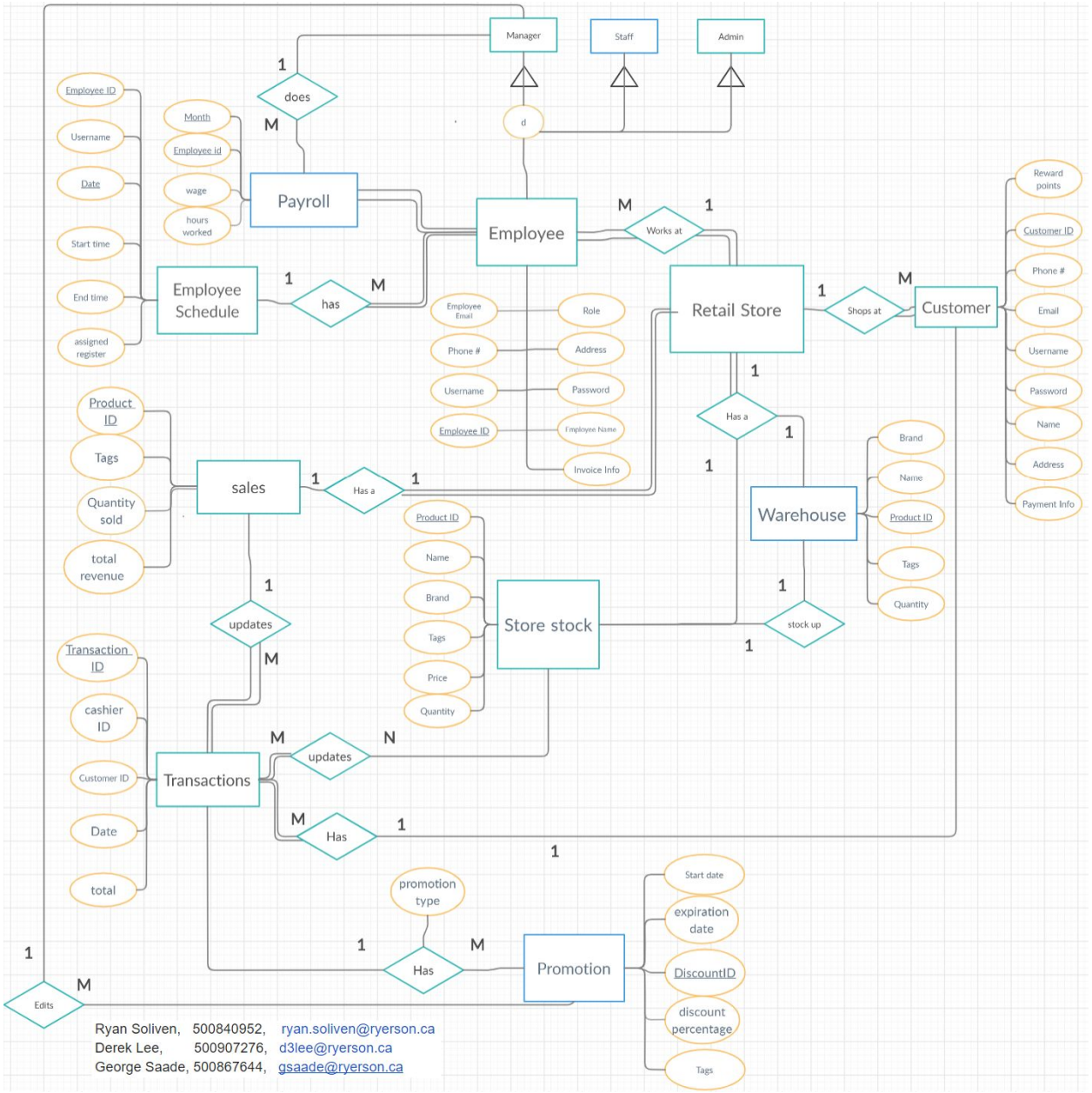
8) Warehouse inventory. Keep information about availability of warehouse stock and count, etc.

9) Promotions. Keep information about promotion keys and start and expiration dates.

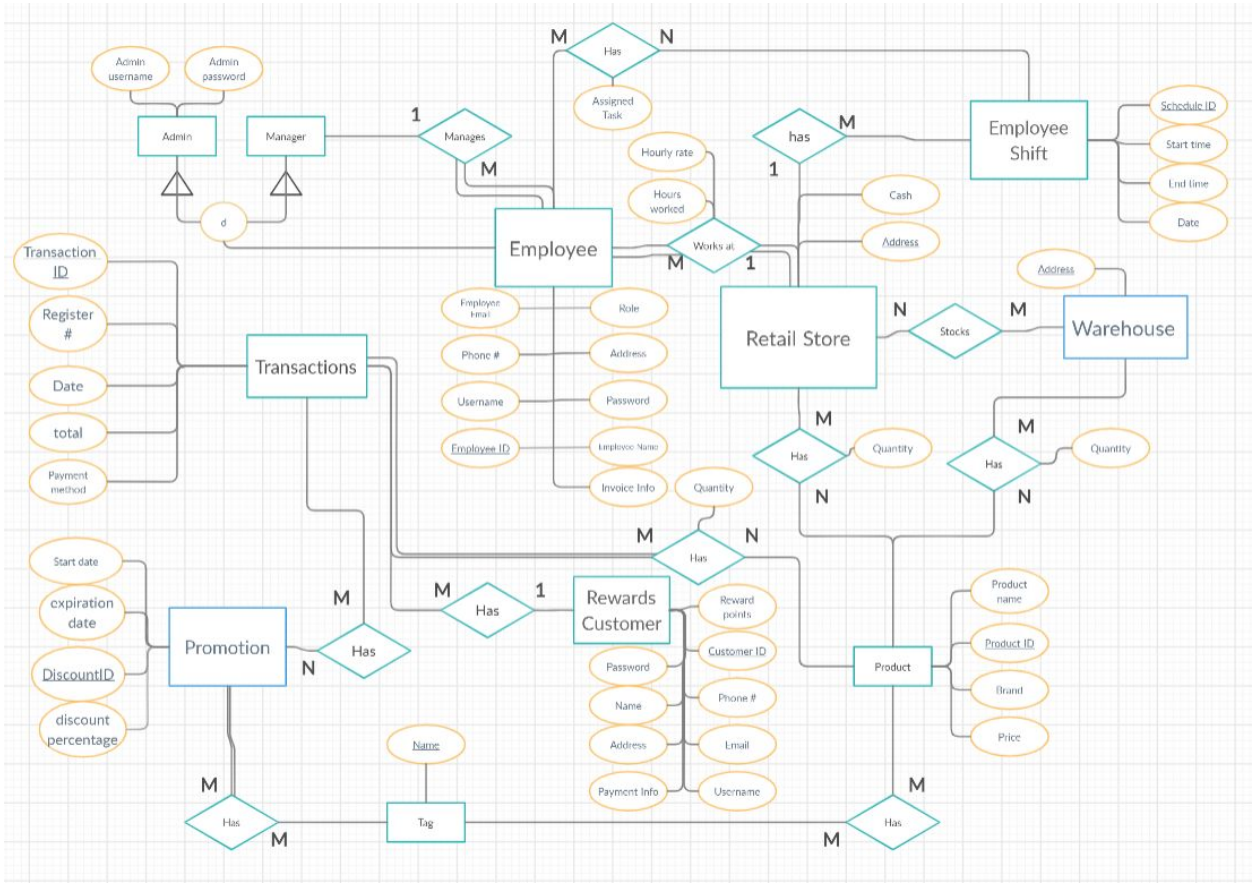
ER Diagrams:

The ER diagram has had multiple iterations, so the latest version will be most relevant to later portions of the report, though the changes are minor and versions are the same in spirit.

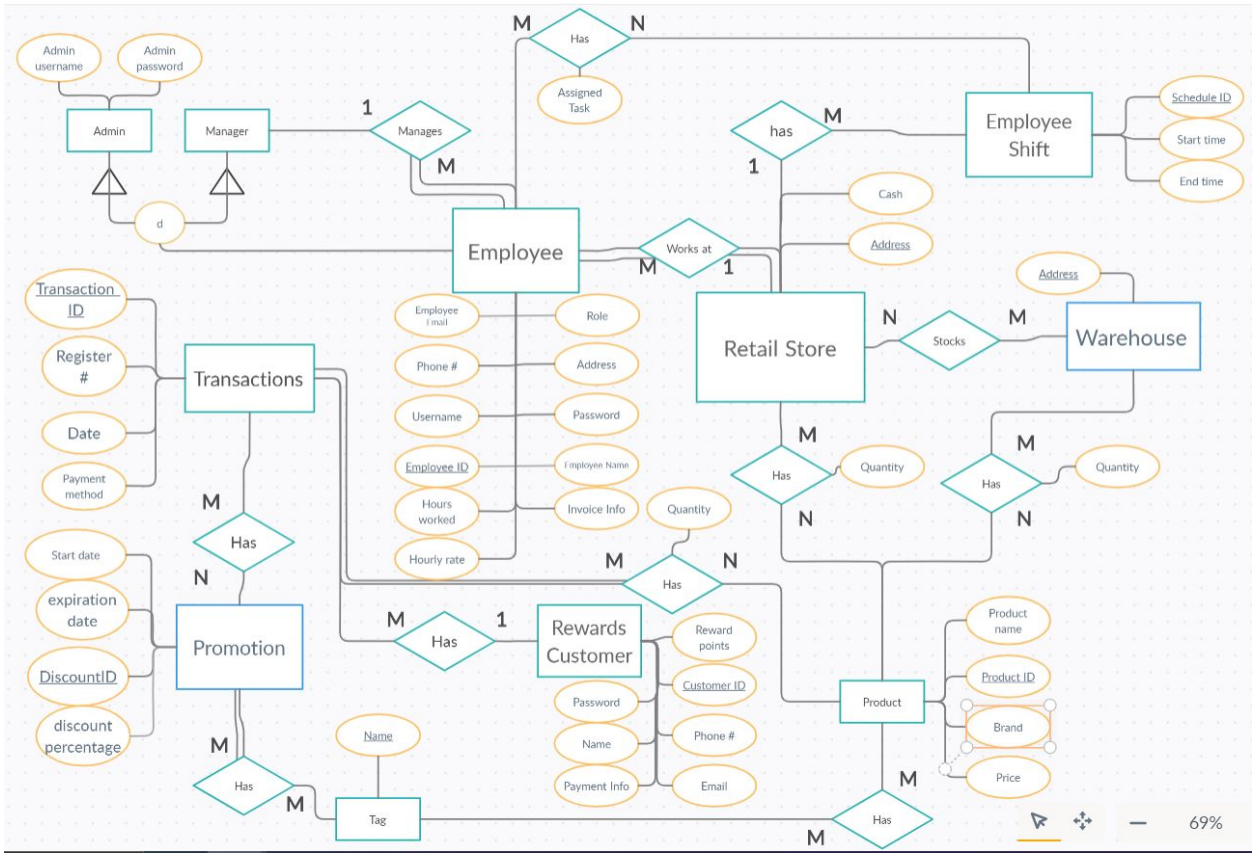
Original ER diagram:



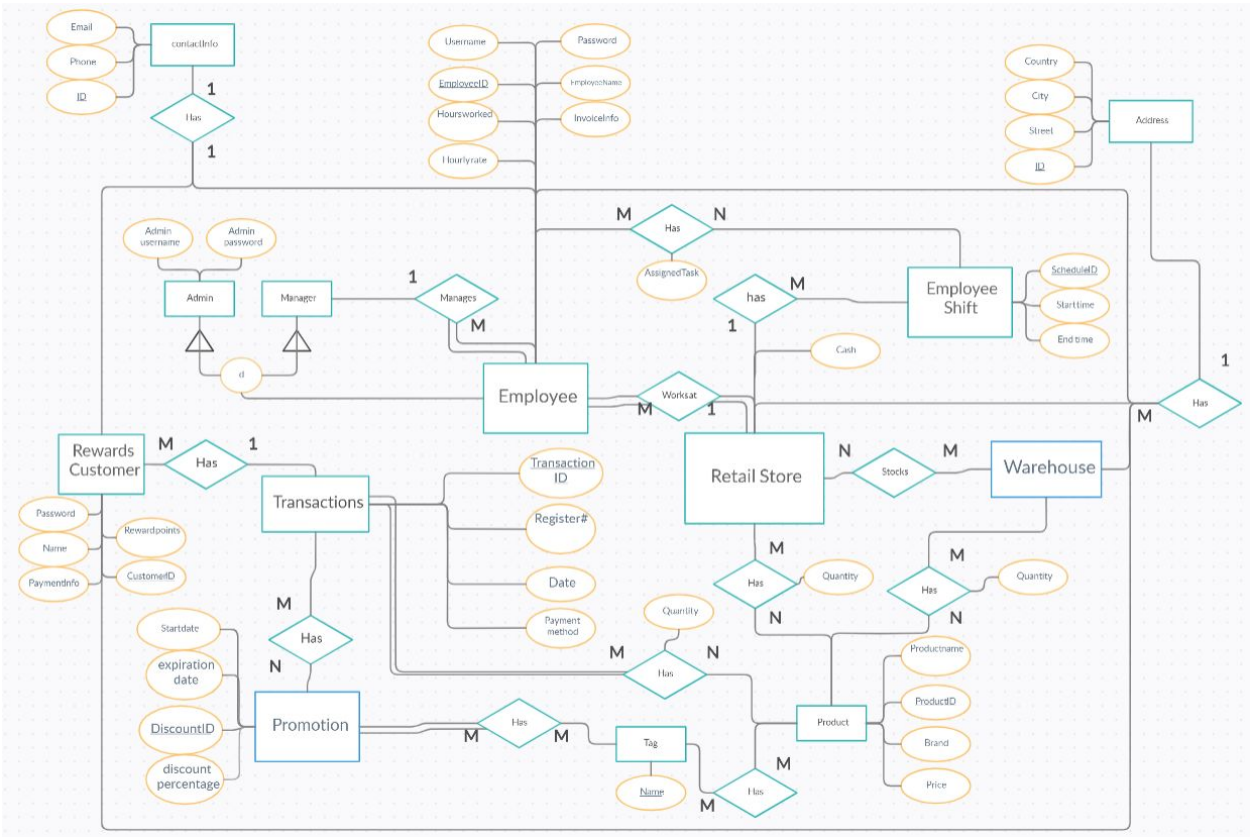
First revision to ER diagram (used since 3rd assignment):



Second revision to ER diagram (used since assignment 6):



Up to date ER diagram (used since 9th assignment):



SQL code

The original version of the source code can be found in the zip this report came in, in the file retailStoreOld.sql
The most recent version of the source code can be found in the zip this report came in, in the file retailStore.sql
Both sql files construct the database correctly if the whole file is run at once.

Basic queries

These queries were designed with the sql code in retailStoreOld.sql, so some may not function correctly if used on the most recent version.

TABLE	QUERY

retail_store	SELECT Address AS "ADDRESS", Cash AS "CASH ON SITE" FROM retail_store ORDER BY Cash DESC;
	`SQL> SELECT Address AS "ADDRESS", Cash AS "CASH ON SITE" FROM retail_store ORDER BY Cash DESC;`
	`ADDRESS CASH ON SITE`
	`55 Side Street 10872.22`
	`67 Hill Street 3321`
	`43 Forest Road 2320.5`
	`104 Main Street 1000`
	`290 Major Macaque Street 377.05`

manager	SELECT ID , FullName, StoreID AS "Store" FROM manager ORDER BY HourlyRate DESC;
	`SQL> SELECT ID , FullName, StoreID AS "Store" FROM manager ORDER BY HourlyRate DESC;`
	`ID FULLNAME Store`
	`10283756 Becca Li 104 Main Street`
	`34567123 Ibrahim Khan 43 Forest Road`

employee	SELECT ID AS "ID", FullName AS "Full Name", ManagerID AS "Manager", StoreID AS "Store" FROM employee;
	`SQL> SELECT ID AS "ID", FullName AS "Full Name", ManagerID AS "Manager", StoreID AS "Store" FROM employee;`
	`ID Full Name Manager Store`
	`472631 Marny Soo 34567123 43 Forest Road`
	`223456 Jimmy Ozar 34567123 43 Forest Road`
	`548273 Mani Yano 10283756 104 Main Street`
	`837463 Ernesto Wright 10283756 104 Main Street`
	SELECT COUNT(ID), StoreID FROM employee GROUP BY StoreID;
	`SQL> SELECT COUNT(ID), StoreID FROM employee GROUP BY StoreID;`
	`COUNT(ID) STOREID`
	`2 43 Forest Road`
	`2 104 Main Street`

admin	SELECT ADMINUSERNAME AS "Admin", ADMINPASSWORD AS "Password" FROM admin;
	`SQL> SELECT ADMINUSERNAME AS "Admin", ADMINPASSWORD AS "Password" FROM admin;`
	`Admin Password`
	`jsnFU87s aodjw85D`

transaction	SELECT STOREID, TransactionTime, PaymentMethod FROM transaction WHERE PaymentMethod = 'credit' ORDER BY TransactionTime DESC;
	`STOREID TRANSACTIONTIME PAYMENTMETHOD`
	`104 Main Street 06-APR-18 10.24.50.000000 AM credit`
	`43 Forest Road 12-SEP-17 12.55.21.000000 PM credit`
	`43 Forest Road 27-JUN-17 11.12.58.000000 AM credit`

promotion	SELECT StartTime, EndTime, DiscountPercentage FROM promotion ORDER BY DiscountPercentage DESC;

	SQL> SELECT StartTime, EndTime, DiscountPercentage FROM promotion ORDER BY DiscountPercentage DESC; STARTTIME ENDTIME DISCOUNTPERCENTAGE 20-NOV-19 09.00.00.000000 AM 11-FEB-20 05.00.00.000000 PM 75 01-JAN-00 09.00.00.000000 AM 01-JAN-00 05.00.00.000000 PM 75
tag	SELECT * FROM tag; SQL> SELECT * FROM tag; NAME --- Clearance Fall Spring Summer Winter
rewards_customer	SELECT * FROM rewards_customer WHERE PHONE IS NULL; SQL> SELECT * FROM rewards_customer WHERE PHONE IS NULL; FULLNAME ID PASSWORD PHONE EMAIL PAYMENTINFO REWARDPOINTS Antony Smit 293851 ufhwg88 ASmitt55@yahoo.ca bank ref #55908 0
warehouse	SELECT * FROM warehouse; SQL> SELECT * FROM warehouse; ADDRESS --- 28 Portside Street 43 Farside Road 90 Penny Boulevard
product	SELECT * FROM product WHERE price >= 15 ORDER BY price DESC; SQL> SELECT * FROM product WHERE price >= 15 ORDER BY price DESC; NAME ID PRICE --- black coat 02227356 45.99 red sweater 09822837 22.55 blue jeans 09827356 20.99 SELECT * FROM product ORDER BY price DESC; SQL> SELECT * FROM product ORDER BY price DESC; NAME ID PRICE --- black coat 02227356 45.99 red sweater 09822837 22.55 blue jeans 09827356 20.99 white tank top 09827399 12.99 thermal socks 09827669 7.99 pink raincoat 00027669 5.99
employee_shift	SELECT * FROM employee_shift WHERE StoreID = '104 Main Street' ORDER BY StartTime

	DESC; SQL> SELECT * FROM employee_shift WHERE StoreID = '104 Main Street' ORDER BY StartTime DESC; ID StartTime EndTime StoreID 092873 23-APR-19 09.00.00.000000 AM 23-APR-19 05.00.00.000000 PM 104 Main Street 092873 21-FEB-18 09.00.00.000000 AM 21-FEB-18 05.00.00.000000 PM 104 Main Street 092273 16-MAY-17 09.00.00.000000 AM 16-MAY-17 05.00.00.000000 PM 104 Main Street
employee_shift_employee	SELECT * FROM employee_shift_employee WHERE EmployeeID = '472631'; SQL> SELECT * FROM employee_shift_employee WHERE EmployeeID = '472631'; SHIFTID EMPLOYEE ASSIGNEDTASK 092873 472631 Sales floor 092273 472631 Sales floor
warehouse_retail_store	SELECT * FROM warehouse_retail_store WHERE StoreLocation = '104 Main Street'; SQL> SELECT * FROM warehouse_retail_store WHERE StoreLocation = '104 Main Street'; STORELOCATION WAREHOUSELOCATION 104 Main Street 28 Portside Street 104 Main Street 43 Farside Road
retail_store_product	SELECT * FROM retail_store_product WHERE ProductID = '09827399'; SQL> SELECT * FROM retail_store_product WHERE ProductID = '09827399'; STORELOCATION PRODUCTI QUANTITY 104 Main Street 09827399 14 43 Forest Road 09827399 22
warehouse_product	SELECT * FROM warehouse_product WHERE ProductID = '49827669'; SQL> SELECT * FROM warehouse_product WHERE ProductID = '49827669'; no rows selected SQL> SELECT * FROM warehouse_product WHERE ProductID = '09827356'; WAREHOUSELOCATION PRODUCTI QUANTITY 28 Portside Street 09827356 252 43 Farside Road 09827356 212
transaction_product	SELECT * FROM transaction_product where ProductID = '00027669'; SQL> SELECT * FROM transaction_product where ProductID = '00027669'; TRANSACT PRODUCTI QUANTITY CUSTOMER 113822 00027669 1
transaction_promotion	SELECT * FROM transaction_promotion where TransactionID = '113822'; SQL> SELECT * FROM transaction_promotion where TransactionID = '113822'; TRANSACT PROMOTIO 113822 0001
promotion_tag	SELECT * FROM promotion_tag WHERE Tag = 'Clearance';

	SQL> SELECT * FROM promotion_tag WHERE Tag = 'Clearance'; PROMOTIO TAG -------- -------------------------------- 0001 Clearance
product_tag	SELECT * FROM product_tag WHERE Tag = 'Spring'; SELECT DISTINCT ProductID FROM product_tag; SQL> SELECT * FROM product_tag WHERE Tag = 'Spring'; PRODUCTI TAG -------- -------------------------------- 00027669 Spring 09827356 Spring SQL> SELECT DISTINCT ProductID FROM product_tag; PRODUCTI -------- 00027669 02227356 09822837 09827356 09827399 09827669

Advanced queries:

These queries were designed with the sql code in retailStoreOld.sql, so some may not function correctly if used on the most recent version. These queries are later updated for use in the 9th assignment code with the most recent SQL code and ER diagram.

Join Queries

description	query
List all warehouses that are authorized to exchange inventory with the store located at '104 Main Street' and has the product 'blue jeans' ('09827356')	SELECT w.Address AS Warehouse, wp.Quantity FROM warehouse w INNER JOIN warehouse_retail_store wr ON w.Address = wr.WarehouseLocation AND wr.StoreLocation = '104 Main Street' INNER JOIN warehouse_product wp ON wp.ProductID = '09827356' AND wp.WarehouseLocation = w.Address; WAREHOUSE QUANTITY 28 Portside Street 252 43 Farside Road 212
Given product 'Yellow sandals' ('49827669') and timestamp '2020-06-13 10:23:00', retrieve any relevant promotional discounts	SELECT product_tag.Tag AS Promotion, promotion.DiscountPercentage AS Percent FROM product_tag INNER JOIN promotion_tag ON promotion_tag.Tag = product_tag.Tag AND product_tag.ProductID = '49827669' INNER JOIN promotion ON promotion_tag.PromotionID = promotion.ID AND TO_TIMESTAMP('2020-06-13 10:23:00', 'YYYY-MM-DD HH24:MI:SS') > promotion.StartTime AND TO_TIMESTAMP('2020-06-13 10:23:00', 'YYYY-MM-DD HH24:MI:SS') < promotion.EndTime; PROMOTION PERCENT Clearance 75 Summer 50

VIEWs

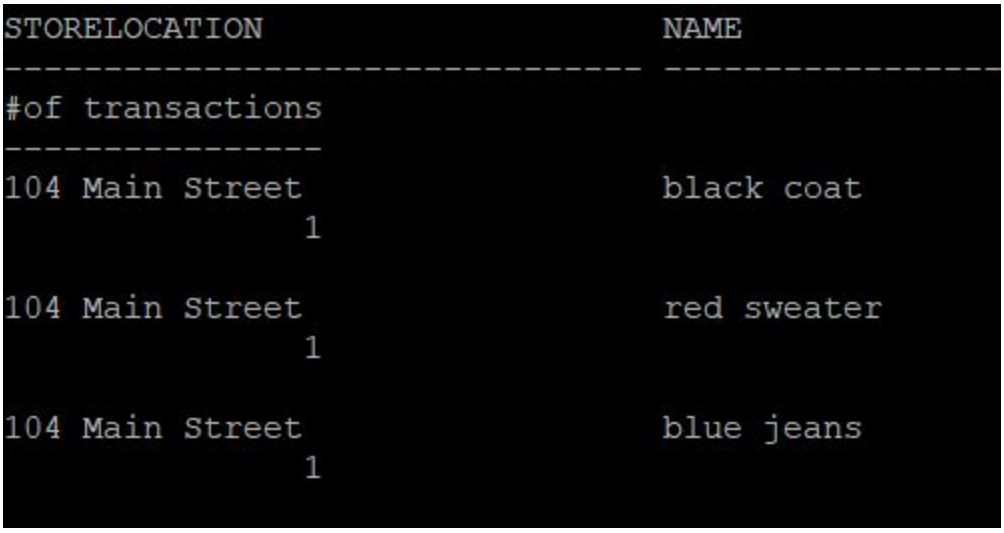
See all employees who have worked at least 200 hours at their store location	DROP VIEW experienced_employees; CREATE VIEW experienced_employees(ID, Name, Store, Hours) AS (SELECT ID, FullName, STOREID, HoursWorked FROM employee WHERE HoursWorked > 199); ID NAME STORE HOUR 472631 Marny Soo 43 Forest Road 20 223456 Jimmy Ozar 43 Forest Road 43 548273 Mani Yano 104 Main Street 34
Shows what stores has less than or equal to 50 quantity of a product	DROP VIEW need_restock; CREATE VIEW need_restock(StoreLocation, ProductID, Quantity) AS (SELECT StoreLocation, ProductID, Quantity FROM retail_store_product WHERE Quantity <= 50);

	STORELOCATION PRODUCTI QUANTITY 104 Main Street 09822837 24 104 Main Street 09827399 14 43 Forest Road 09827399 22 43 Forest Road 09822837 42
Shows what promotions are currently in effect for all stores	DROP VIEW active_promotions; CREATE VIEW active_promotions(PromotionID, Percentage, EndsOn) AS (SELECT p.ID, p.DiscountPercentage, p.EndTime FROM promotion p WHERE (SELECT CURRENT_TIMESTAMP FROM dual) > p.StartTime AND (SELECT CURRENT_TIMESTAMP FROM dual) < p.EndTime); PROMOTIO PERCENTAGE ENDSON 0001 75 01-JAN-00 05.00.00.000000 PM 0005 50 01-NOV-20 05.00.00.000000 PM

Advanced Queries

List of products being sold at a retail store and being held at Warehouse Location '90 Penny Boulevard' AND Warehouse Location '43 Farside Road'	SELECT ID, Name FROM product WHERE EXISTS (SELECT rsp.ProductID FROM retail_store_product rsp, warehouse_product w1, warehouse_product w2 WHERE ID = rsp.productid AND w1.WarehouseLocation = 'W2' AND w2.WarehouseLocation = 'W3' AND rsp.ProductID = w1.ProductID AND rsp.ProductID = w2.ProductID); ID NAME 09827669 thermal socks 02227356 black coat 00027669 pink raincoat
Number of warehouses per store and number of employees	select r.addressid , count(w.addressid) as "WareHouses#", (select count(*) from employee e where r.addressid = e.storeid group by r.addressid) as employees#

	from RETAIL_STORE r, WAREHOUSE_RETAIL_STORE l, WAREHOUSE w where r.addressid = l.storelocation and l.warehouselocation = w.addressid group by r.addressid; ADDRESS WareHouses# EMPLOYEES# 67 Hill Street 2 43 Forest Road 2 2 104 Main Street 2 2 290 Major Macaque Street 3 55 Side Street 1
Lists products that are exclusive to store locations. NOT EXISTS = MINUS	SELECT StoreLocation, rsp1.ProductID, Name FROM retail_store_product rsp1, product WHERE rsp1.ProductID = product.ID AND NOT EXISTS (SELECT * FROM retail_store_product rsp2 WHERE rsp1.ProductID = rsp2.ProductID AND rsp1.StoreLocation <> rsp2.StoreLocation); STORELOCATION PRODUCTI NAME 43 Forest Road 00027669 pink raincoat 104 Main Street 09827356 blue jeans
List of transactions that were made by credit OR debit	SELECT * FROM transaction WHERE PaymentMethod = 'credit' UNION SELECT * FROM transaction WHERE PaymentMethod = 'debit'; ID REGISTER TRANSACTIONTIME PAYMENTMETHOD STOREID credit 43 Forest Road
List of managers with fewer than 4 subordinates	SELECT m.FullName, count(e.ID) AS subordinates FROM manager m LEFT JOIN employee e ON m.ID = e.ManagerID GROUP BY m.FullName HAVING count(e.ID) < 4; FULLNAME SUBORDINATES Ibrahim Khan 2 Becca Li 2

How each item perform in each store(number of sales)	SELECT rp.STORELOCATION,p.name, (SELECT count(t.id) from TRANSACTION t, TRANSACTION_PRODUCT tp where t.id = tp.transactionid and tp.productid = p.id group by p.name) as "#of transactions" FROM RETAIL_STORE_PRODUCT rp, product p where rp.productid = p.id; 

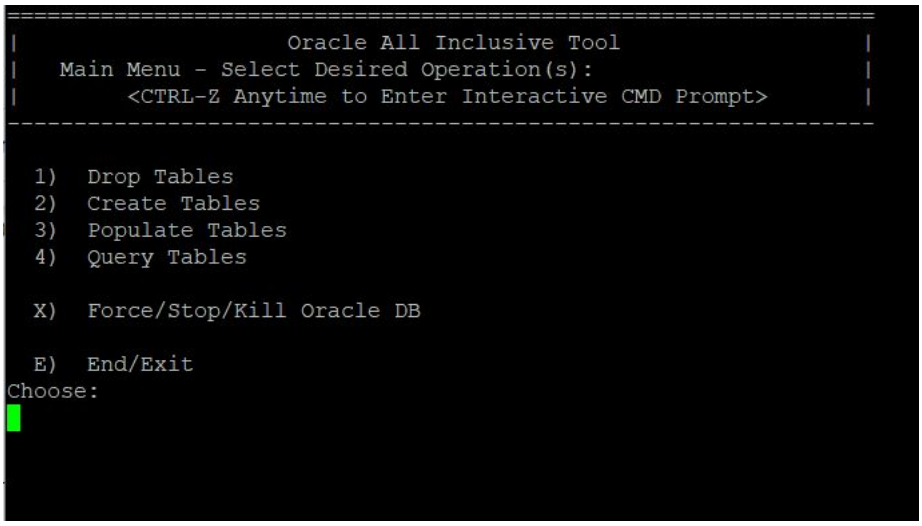
```
STORELOCATION                  NAME
------------------------------ ----------------
#of transactions
---------------
104 Main Street                black coat
              1

104 Main Street                red sweater
              1

104 Main Street                blue jeans
              1
```

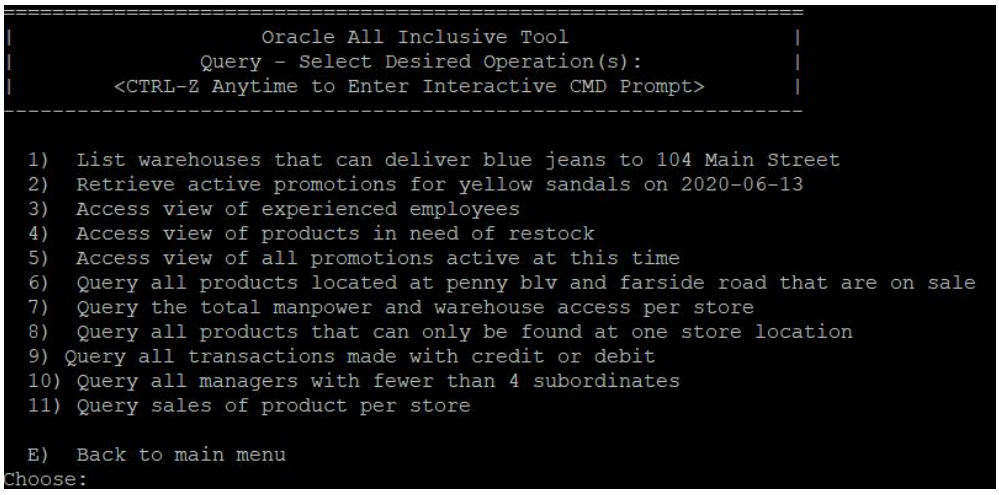
Shell scripts:

These scripts were made on the Ryerson University moon servers and are best run there.

To run you will need **menu.sh**, **drop_tables.sh**, **create_tables.sh**, **populate_tables.sh**, and **queries.sh**. These files can be found in the zip this report was submitted in. The **username/password** has been censored so you should find-replace all instances of "**username/password**" in each file with an accepted username and password.

The menu can be started by running **menu.sh**, with all other files in the same directory.





Functional dependencies

FD's were based on the original SQL code.

Table	Functional dependencies
retail_store	Address -> Cash
manager	ID -> Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID Username -> Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID
employee	ID -> Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID, manager_ID Username -> Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID, manager_ID managerID -> StoreID
admin	ID -> AdminUsername, AdminPassword, Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked Username -> AdminUsername, AdminPassword, Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked
transaction	ID -> RegisterNumber, TransactionTime, PaymentMethod, StoreID
promotion	ID -> StartTime, EndTime, DiscountPercentage
tag	N/A
product	ID -> Name, Price
rewards_customer	ID -> FullName, Password, Phone, Email, PaymentInfo, RewardPoints Email -> FullName, Password, Phone, ID, PaymentInfo, RewardPoints
warehouse	N/A
employee_shift	ID -> StartTime, EndTime, StoreID
employee_shift_employee	ShiftID, EmployeeID -> AssignedTask
warehouse_retail_store	N/A
retail_store_product	StoreLocation, ProductID -> Quantity
warehouse_product	WarehouseLocation, ProductID -> Quantity
transaction_product	TransactionID, ProductID -> Quantity
transaction_promotion	N/A
promotion_tag	N/A
product_tag	N/A

3NF analysis:

Table	Functional dependencies	3NF?
retail_store	Address -> Cash	No because address can be broken down into street, city, country; violates 1NF solution: move address into separate table to break into indivisible parts. AddressID does not create new FDs.
manager	ID -> Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID Username -> Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID	No because address can be broken down into street, city, country; violates 1NF solution: move address into separate table to break into indivisible parts. AddressID does not create new FDs. No because Email or Phone are can determine Address, Password, FullName, StoreID, HourlyRate, HoursWorked Solution: make a new table contactINFO with attributes contactID, Email, Phone. Original table holds foreign key. 1-1 relation between contact info and individuals means foreign key becomes candidate, which does not interfere with 3NF.
employee	ID -> Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID, manager_ID Username -> Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID, manager_ID managerID -> StoreID	No because address can be broken down into street, city, country; violates 1NF solution: move address into separate table to break into indivisible parts. AddressID does not create new FDs. No because Email or Phone are can determine Address, Password, FullName, StoreID, HourlyRate, HoursWorked Solution: use new table contactINFO with attributes contactID, Email, Phone. original table holds foreign key. Original table holds foreign key. 1-1 relation between contact info and individuals means foreign key becomes candidate, which does not interfere with 3NF. No because manager id determines storeid solution(- mid_strID(manager_ID,StoreID))
admin	ID -> AdminUsername, AdminPassword, Email, Phone, Address, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked Username -> AdminUsername, AdminPassword, Email, Phone, Address, ID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked	No because address can be broken down into street, city, country; violates 1NF solution: move address into separate table to break into indivisible parts. AddressID does not create new FDs. No because Email or Phone are can determine Address, Password, FullName, StoreID, HourlyRate, HoursWorked Solution: use new table contactINFO with attributes contactID, Email, Phone. original table holds foreign key. Original table holds foreign key. 1-1 relation between contact info and individuals means foreign key becomes candidate, which does not interfere with 3NF.
transaction	ID -> RegisterNumber, TransactionTime, PaymentMethod, StoreID	Yes
promotion	ID -> StartTime, EndTime, DiscountPercentage	Yes
tag	N/A	Yes
product	ID -> Name, Price	Yes
rewards_customer	ID -> FullName, Password, Phone, Email, PaymentInfo, RewardPoints Email -> FullName, Password, Phone, ID, PaymentInfo, RewardPoints	No, because there is transitivity in that ID -> Email -> everything else. Solution: use new table contactINFO with attributes contactID, Email, Phone. The original table holds foreign keys. Original table holds foreign key. 1-1 relation between contact info and individuals means foreign key becomes

		candidate, which does not interfere with 3NF.
warehouse	N/A	N/A
employee_shift	ID -> StartTime, EndTime, StoreID	Yes
employee_shift_employee	ShiftID, EmployeeID -> AssignedTask	Yes
warehouse_retail_store	N/A	Yes
retail_store_product	StoreLocation, ProductID -> Quantity	Yes
warehouse_product	WarehouseLocation, ProductID -> Quantity	Yes
transaction_product	TransactionID, ProductID -> Quantity	Yes
transaction_promotion	N/A	Yes
promotion_tag	N/A	Yes
product_tag	N/A	Yes

New tables

table	code	Functional dependencies (now 3nf)

retail_store	retail_store(AddressID, Balance)	AddressID -> Balance
manager	manager(ID, ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID)	ID -> ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID Username -> (ID, ContactID, AddressID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID ContactID -> ID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked
employee	employee(ID, ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID, storeID)	ID -> ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID Username -> ID, ContactID, AddressID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID ContactID -> ID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID
mID_strID	mID_strID(manager_ID,StoreID)	manager_ID -> StoreID
admin	admin (ID, AdminUsername, AdminPassword, AddressID, ContactID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked)	ID -> AdminUsername, AdminPassword, AddressID, ContactID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked Username -> ID, AdminUsername, AdminPassword, AddressID, ContactID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked
rewards_customer	rewards_customer (ID, FullName, Password, PaymentInfo, contactID, RewardPoints)	ID -> FullName, Password, PaymentInfo, ContactID, RewardPoints ContactID -> ID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked
contactINFO	contactINFO(ContactID, Email, Phone)	ContactID -> Email, Phone Email -> Email, Phone, ContactID Phone-> Email, Phone, ContactID
address	address(addressID, street, city, country)	AddressID -> street, city, country

BCNF analysis:

Table	Functional dependencies	BCNF?
retail_store	AddressID -> Balance	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
manager	ID -> ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID Username -> ID, ContactID, AddressID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, StoreID ContactID -> ID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked ID -> ID Username ->Username ContactID -> ContactID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
employee	ID -> ContactID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID Username -> ID, ContactID, AddressID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID ContactID -> ID, AddressID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked, manager_ID ID -> ID Username ->Username ContactID -> ContactID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
admin	ID -> AdminUsername, AdminPassword, AddressID, ContactID, Username, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked Username -> ID, AdminUsername, AdminPassword, AddressID, ContactID, Password, FullName, InvoiceInfo, HourlyRate, HoursWorked ID -> ID Username ->Username	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
transaction	ID -> RegisterNumber, TransactionTime, PaymentMethod, StoreID ID ->ID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
promotion	ID -> StartTime, EndTime, DiscountPercentage ID -> ID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.

tag	N/A	
product	ID -> Name, Price ID ->ID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
rewards_customer	ID -> FullName, Password, PaymentInfo, ContactID, RewardPoints ContactID -> ID, FullName, Password, PaymentInfo, RewardPoints ID ->ID ContactID -> ContactID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
warehouse	N/A	
employee_shift	ID -> StartTime, EndTime, StoreID ID ->ID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
employee_shift_employee	ShiftID, EmployeeID -> AssignedTask ShiftID ->ShiftID EmployeeID ->EmployeeID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
warehouse_retail_store	N/A	
retail_store_product	StoreLocation, ProductID -> Quantity ProductID -> ProductID StoreLocation ->StoreLocation	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
warehouse_product	WarehouseLocation, ProductID -> Quantity WarehouseLocation ->WarehouseLocation ProductID ->, ProductID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
transaction_product	TransactionID, ProductID -> Quantity TransactionID->TransactionID ProductID ->ProductID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
transaction_promotion	N/A	
promotion_tag	N/A	
product_tag	N/A	
contact_Info	ContactID -> Email, Phone Email -> Phone, ContactID Phone-> Email, ContactID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
address	AddressID -> street, city, country AddressID ->AddressID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.
mID_strID	manager_ID -> StoreID manager_ID ->manager_ID	Yes, because all non trivial l functional dependencies have a left side that is irreducible and is a candidate key as their determinant.

Bernstein's algorithm

This algorithm was performed on a hypothetical version of the transaction table.

table	process
transaction (ID, RegisterNumber, TransactionTime, PaymentMethod, Card#, CardHolder, EmployeeID) ID -> RegisterNumber, TransactionTime, PaymentMethod, Card#, CardHolder, EmployeeID. Card# -> PaymentMethod, CardHolder RegisterNumber, TransactionTime -> EmployeeID EmployeeID, TransactionTime -> RegisterNumber	1) Determine Functional dependencies ID -> RegisterNumber ID -> TransactionTime ID -> PaymentMethod ID -> Card# ID -> CardHolder ID -> EmployeeID Card# -> PaymentMethod Card# -> CardHolder RegisterNumber, TransactionTime -> EmployeeID EmployeeID, TransactionTime -> RegisterNumber 2a) Find redundancies ID -> RegisterNumber: ID+ = {ID} ID -> TransactionTime: ID+ = {ID} ID -> PaymentMethod: ID+ = {ID} ID -> Card#: ID+ = {ID, PaymentMethod, CardHolder} ID -> CardHolder: ID+ = {ID} ID -> EmployeeID: ID+ = {ID} Card# -> PaymentMethod: Card#+ = {Card#} Card# -> CardHolder: Card#+ = {Card#} RegisterNumber, TransactionTime -> EmployeeID: RegisterNumber, TransactionTime+ = {RegisterNumber, TransactionTime} EmployeeID, TransactionTime -> RegisterNumber: EmployeeID, TransactionTime+ = {EmployeeID, TransactionTime} 2b) remove partial dependencies RegisterNumber -> EmployeeID: RegisterNumber + = {RegisterNumber, TransactionTime} TransactionTime+ = {RegisterNumber, TransactionTime} EmployeeID -> RegisterNumber: EmployeeID = {EmployeeID} TransactionTime+ = {TransactionTime} No partial dependencies can be found. All FDs are fully dependent 3) find keys ID is only in LHS so it is part of key Register number, Card#, TransactionTime, EmployeeID are in both LHS and RHS so they may be part of key All other attributes appear in RHS only, thus they are not part of key ID + = {ID, RegisterNumber, TransactionTime, PaymentMethod, Card#, CardHolder, EmployeeID} For brevity, all potential keys containing ID (all of them) will be valid keys 4) make tables ID -> RegisterNumber, TransactionTime, PaymentMethod, Card#, CardHolder, EmployeeID : R1(ID, RegisterNumber, TransactionTime, PaymentMethod, Card#, CardHolder, EmployeeID) Card# -> PaymentMethod, CardHolder : R2(PaymentMethod, CardHolder) RegisterNumber, TransactionTime -> EmployeeID : R3(RegisterNumber, TransactionTime) EmployeeID, TransactionTime -> RegisterNumber : R4(EmployeeID, TransactionTime) Since R1 is the superset of all tables, R1 is the final table

BCNF Decomposition

table	process
rewards_customer(ID, Email, Password, PaymentInfo, RewardPoints ID -> Email, Password Email -> PaymentInfo, RewardPoints	R = ID, Email, Password, PaymentInfo, RewardPoints F = {ID -> Email, Password, Email -> PaymentInfo, RewardPoints} Attribute Closure: ID+ = {ID, Email, PaymentInfo, Password, RewardPoints} Email+ = {PaymentInfo, RewardPoints} Password+ = {Password} PaymentInfo+ = {PaymentInfo} RewardPoints+ = {RewardPoints} ID is a candidate key as ID gets us all other attributes. Looking at the first FD in F, ID -> Email, Password, ID is a key so this FD does not violate BCNF. Looking at the second FD in F, Email -> PaymentInfo, RewardPoints, Email is not a key so this FD violates BCNF. We then decompose R into two sub schemas: (Email, PaymentInfo, RewardPoints) and (ID, Email, Password). (Email, PaymentInfo, RewardPoints) is BCNF with F1 = {Email -> PaymentInfo, RewardPoints} where Email is the key. (ID, Email, PaymentInfo) is BCNF with F2 = {ID -> Email, Password} where ID is the key. Therefore final BCNF schema for R is R1 = (Email, PaymentInfo, RewardPoints) R2 = (ID, Email, Password)

Java Interface:

This implementation uses the most recent version of the SQL source code, which can be found in **retail_store.sql**.
This code was designed to run on the Ryerson University moon servers.
Compiled with the **ojdbc6.jar** file from oracle website.
compiled and run by running:

javac -cp ojdbc6.jar: JdbcOracleConnectionTemplate.java
java -cp ojdbc6.jar: JdbcOracleConnectionTemplate

requires the files **createTables.txt**, **populateTables.txt**, **dropTables.txt**, and **queries.txt** to be in the same directory as the main program **JdbcOracleConnectionTemplate.java**. Replace the "**username/password**" in **JdbcOracleConnectionTemplate.java** with an acceptable **username/password**. All these files can be found in the zip this report was submitted with.

```
d13lee@metis:~/cps510/java$ javac -cp ojdbc6.jar: JdbcOracleConnectionTemplate.java
d13lee@metis:~/cps510/java$ java -cp ojdbc6.jar: JdbcOracleConnectionTemplate
------------------------------------------------------------
Connected to database successfully.
------------------------------------------------------------
Select an option:
1) create tables
2) populate tables
3) drop tables
4) perform queries
q) quit out
4
------------------------------------------------------------
1) Relevant promotions for item 49827669 if transaction is made at time 2020-06-13 10:23:00
2) All transactions made at any store with credit or debit
3) All managers with their number of subordinates fewer than 4
4) All products found exclusively at one store location
5) List of products being sold at a retail store and being held at Warehouse Location '90 Penny Boulevard' AND Warehouse Location '43 Farside Road'
6) Count warehouses and employees working at each store
7) Count the number of sales of each product per store location
8) List all warehouses that are authorized to exchange inventory with the store located at '104 Main Street' and has the product 'blue jeans'
q) quit
3
Ibrahim Khan has 2 subordinates.
Becca Li has 2 subordinates.
------------------------------------------------------------
```

Relational algebra:

These were formulated with the most recent design of the SQL code and ER diagram.

	Query	RA form
1	SELECT product_tag.Tag AS Promotion, promotion.DiscountPercentage AS Percent FROM product_tag INNER JOIN promotion_tag ON promotion_tag.Tag = product_tag.Tag AND product_tag.ProductID = '49827669' INNER JOIN promotion ON promotion_tag.PromotionID = promotion.ID AND TO_TIMESTAMP('2020-06-13 10:23:00', 'YYYY-MM-DD HH24:MI:SS') > promotion.StartTime AND TO_TIMESTAMP('2020-06-13 10:23:00', 'YYYY-MM-DD HH24:MI:SS') < promotion.EndTime;	PROMOS_FOR_49827669_ON_DAY ← $\pi_{\text{Tag, DiscountPercentage}}$ (product_tag $\bowtie_{\text{promotion_tag.Tag = product_tag.Tag}}$ (promotion_tag $\bowtie_{\text{'2020-06-13 10:23:00' > promotion.StartTime, '2020-06-13 10:23:00' < promotion.EndTime}}$ (promotion)))
2	SELECT * FROM transaction WHERE PaymentMethod = 'credit' UNION SELECT * FROM transaction WHERE PaymentMethod = 'debit';	CREDIT ← $\sigma_{\text{PaymentMethod = 'credit'}}$(transaction) DEBIT ← $\sigma_{\text{PaymentMethod = 'debit'}}$(transaction) CREDIT_OR_DEBIT ← CREDIT ∪ DEBIT
3	SELECT m.FullName, count(e.ID) AS subordinates FROM manager m LEFT JOIN employee e ON m.ID = e.Manager_ID GROUP BY m.FullName HAVING count(e.ID) < 4;	UNDERSTAFFED ← $_{\text{manager.FullName}}\mathcal{F}_{\text{COUNT employee.ID}}$ (manager $⟕_{\text{manager.ID = employee.Manager_ID}}$ (employee))
4	SELECT StoreLocation, rsp1.ProductID, Name FROM retail_store_product rsp1, product WHERE rsp1.ProductID = product.ID AND NOT EXISTS (SELECT * FROM retail_store_product rsp2 WHERE rsp1.ProductID = rsp2.ProductID AND rsp1.StoreLocation <> rsp2.StoreLocation);	$\Pi_{\text{StoreLocation, rsp1.productID, Name}}$ ($\sigma_{\text{rsp1.ProductID = product.ID}}$ (retail_store_product rsp1□□ product) - ($\sigma_{\text{rsp1.ProductID = rsp2.ProductID AND rsp1.StoreLocation ≠rsp2.StoreLocation}}$ (retail_store_product rsp2))
5	SELECT ID, Name FROM product WHERE EXISTS (SELECT rsp.ProductID FROM retail_store_product rsp, warehouse_product w1, warehouse_product w2 WHERE ID = rsp.productid AND w1.WarehouseLocation = 'W2' AND w2.WarehouseLocation = 'W3' AND rsp.ProductID = w1.ProductID AND rsp.ProductID = w2.ProductID);	$\Pi_{\text{ID, Name}}$ (product) ∩ $\Pi_{\text{rsp.ProductID}}$($\sigma_{\text{ID = rsp.ProductID}}$ AND w1.WarehouseLocation = 'W2' AND w2.WarehouseLocation = 'W3' AND rsp.ProductID = w1.ProductID AND rsp.ProductID = w2.ProductID(retail_store_product rsp□□ warehouse_product w1□□ warehouse_product w2)
6	select r.addressid , count(w.addressid) as "WareHouses#", (select count(*) from employee e where r.addressid = e.storeid group by r.addressid) as employees#	EMPCOUNT← $_{\text{R.addressid}}\mathcal{F}_{\text{COUNT}}$ $\sigma_{\text{r.addressid = e.storeid}}$ (employee e) r.addressid F count w.addressid , EMPCOUNT $\sigma_{\text{r.addressid = l.storelocation}}$ AND l.warehouselocation = w.addressid (RETAIL_STORE r□□ WAREHOUSE_RETAIL_STORE l□□ WAREHOUSE w)

	from RETAIL_STORE r, WAREHOUSE_RETAIL_STORE l, WAREHOUSE w where r.addressid = l.storelocation and l.warehouselocation = w.addressid group by r.addressid;	
7	SELECT rp.STORELOCATION,p.name, (SELECT count(t.id) from TRANSACTION t, TRANSACTION_PRODUCT tp where t.id = tp.transactionid and tp.productid = p.id group by p.name) as "#of transactions" FROM RETAIL_STORE_PRODUCT rp, product p where rp.productid = p.id;	#of transactions $\leftarrow_{p.name} F_{count\ t.id}\ \sigma_{t.id = tp.transactionid\ AND\ tp.productid = p.id}$ (TRANSACTION t □□ TRANSACTION_PRODUCT tp) $\Pi_{rp.STORELOCATION,\ IDGRP,\ \#of\ transactions}\ \sigma_{rp.productid = p.id}$ (RETAIL_STORE_PRODUCT rp□□ product p)
8	SELECT w.AddressID AS warehouse, wp.Quantity FROM warehouse w INNER JOIN warehouse_retail_store wr ON w.AddressID = wr.WarehouseLocation AND wr.StoreLocation = '01' INNER JOIN warehouse_product wp ON wp.ProductID = '09827356' AND wp.WarehouseLocation = w.AddressID;	$\Pi_{w.AddressID,\ wp.Quantity}$ (warehouse w $\bowtie \sigma_{w.AddressID = wr.WarehouseLocation\ AND\ wr.StoreLocation = '01'}$ (warehouse_retail_store wr $\bowtie \sigma_{wp.ProductID = '09827356'\ AND\ wp.WarehouseLocation = w.AddressID}$ (warehouse_product wp)))