Adam Cunningham
CSC460 homework 4
NetID Laser
Dr. McCann
Due 11/12

Note: I received a 24 medical extension to complete this assignment.

5.4)
Commonalities:
All join operations operate on a pair of relations, with an underlying simple (or complex) restriction on the attribute sets operated on.
The resulting relation will, if possible given the join type and its parameters, contain tuples built from the information in the original two sets

All join operations can be written as a selection on the cartesian product of the two sets. Some additionally involve projections on either set or both

Differences:
Theta-join is as stated above, simply a selection on the cartesian product of two sets, with the restriction being that it *compares* two attributes from the sets using an equality operation such as <,>,=...ect.

Equi-join is a theta-join restricted to the = operation.

Natural-join is an equi-join with a disjunctive selection, where duplicate attributes titles are omitted.
for example some relation R(a,b) joined with some relation S(a,b,c) would be the selection on the cartesian product of R x S, where the selection statement is [R.a=S.a OR R.b=S.b] (because those are the attribute names they have in common), and the resulting relation would have the attribute names (a,b,c) (which is what I meant about duplicate attribute titles being omitted).

Outer-join is a Natural-join, but no tuple on the left side of the join operation is omitted, even if there is not a matching attribute value on the right side of the join. In place of the attribute that did not match, a null marker is instead placed(there is also a right side and both side version of this)

5.6)
DRC:
A well formed formula is an idea taken from First Order Logic, used in Predicate Logic aka Predicate Calculus, and finally transferred in this class to Domain Relational Calculus. It basically means it follows syntax, and can be understood and evaluated, and that it is made of

valid building blocks. It is typically recursively defined. I have paraphrased the formal definition below:
In Domain relational calculus, a formula is well-formed iff it is made up of Atoms. Atoms are of one of the forms:

- Relation(d1,d2 … dn) where R is of degree n, and ds are enumerated relation attributes
- d <comparator> d
  [where ds are enumerated relation attributes (domain variables),
  and the comparator is valid for the pair of attributes]
- d <comparator> constant
  [where d is a domain variable, and the comparator is valid for d and the constant]

a formula is then built of atoms via these three rules

- all atoms are formulas
- the conjunction, disjunction, and negation of a formula is a formula
- if a formula contains a free variable, then it is still a formula if the free variable has been universally or existentially quantified

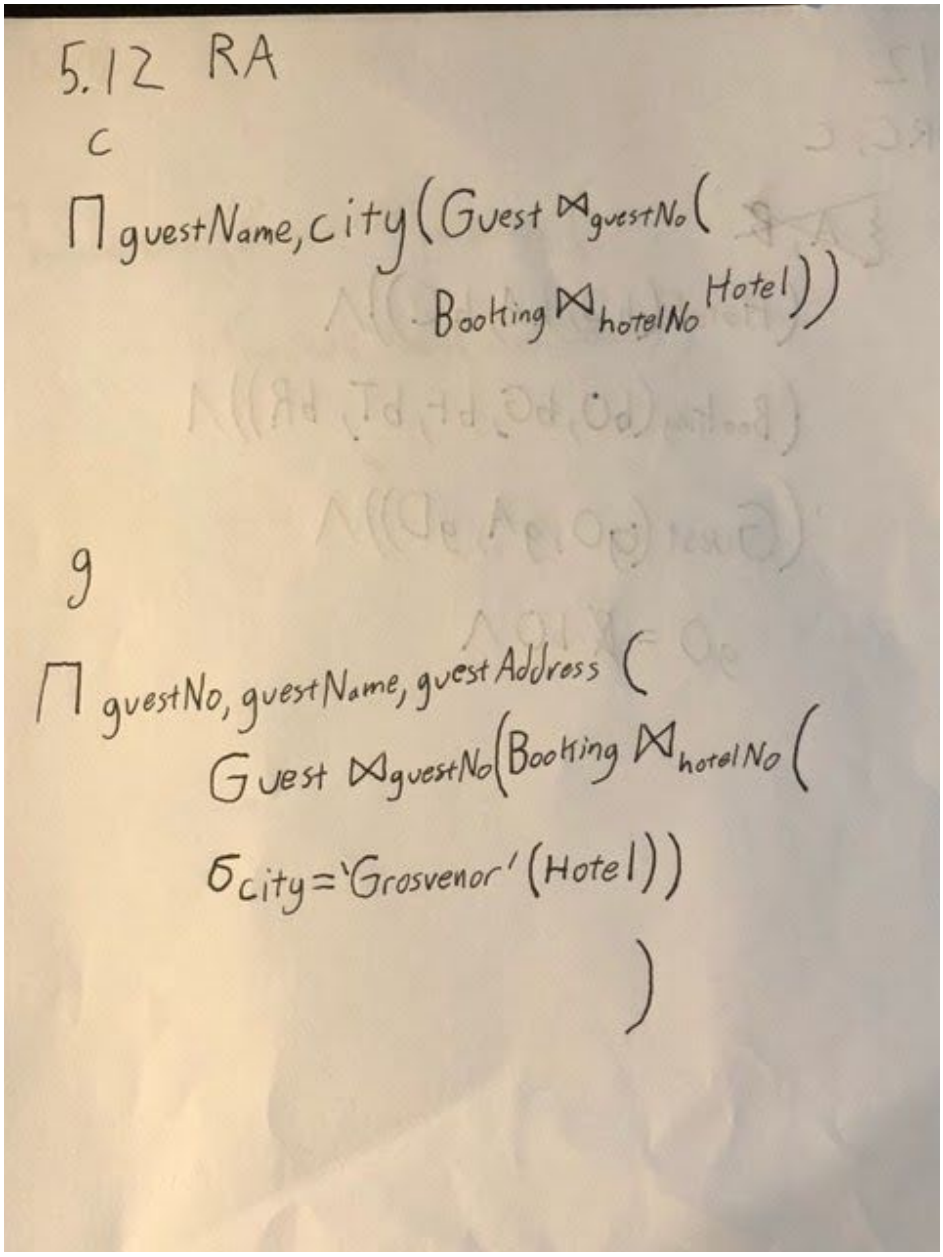
5.12) DRC:

list the names and cities of all guests
DRC: c
{guestName, city | (∃ hotelNo, ∃ guestNo)
(Hotel (hotelNo, hotelName, city) ^
Booking (hotelNo, guestNo, dateFrom, dateTo, roomNo) ^
Guest (guestNo, guestName, guestAddress)) }

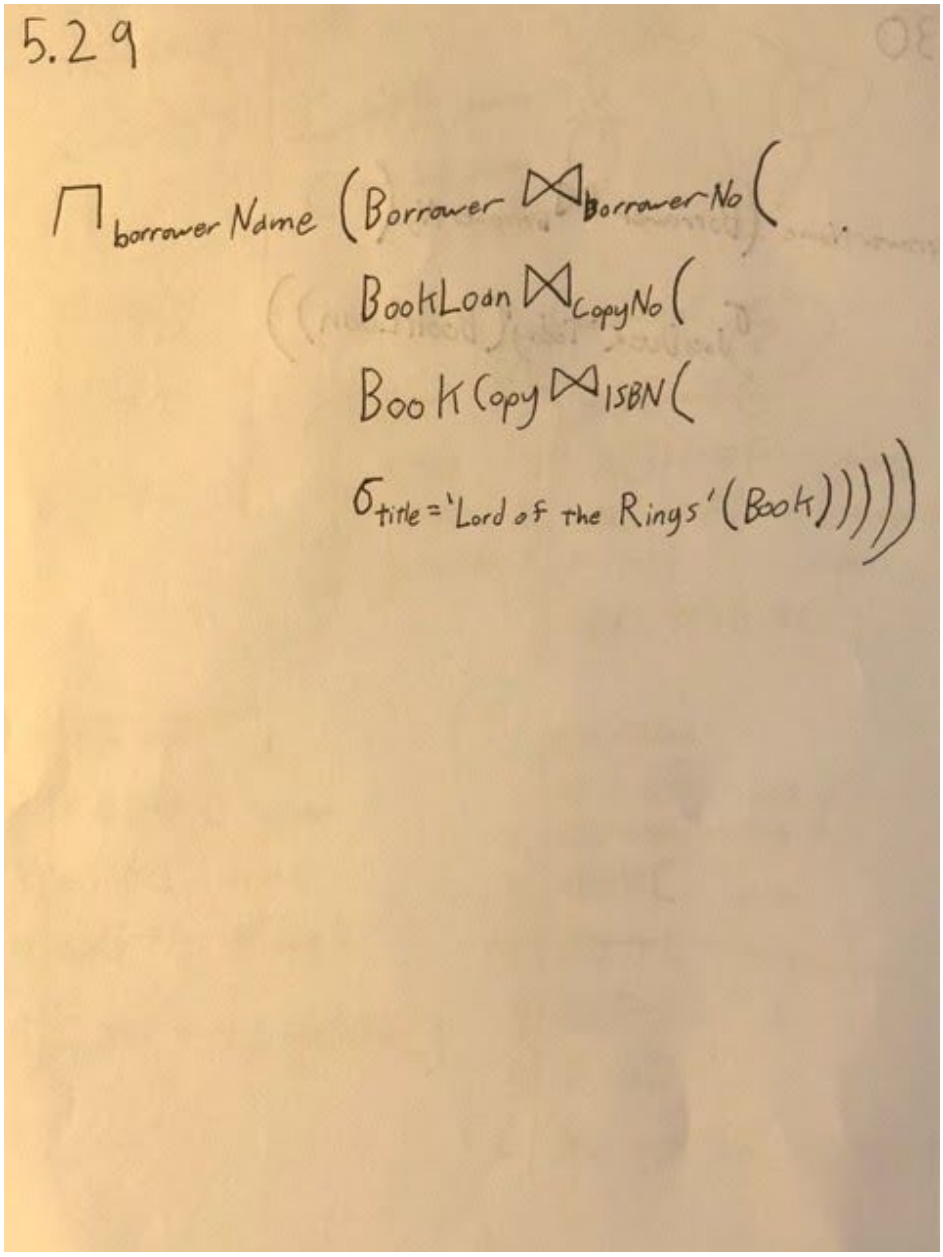
g
list the guest details (gno, gname, gaddr) of all guests staying at the grosvenor hotel

{guestNo, guestName, guestAddress |
(Hotel (hotelNo, hotelName, **'Grosvenor'**) ^
Booking (hotelNo, guestNo, dateFrom, dateTo, roomNo) ^
Guest (guestNo, guestName, guestAddress)) }

5.12 RA:



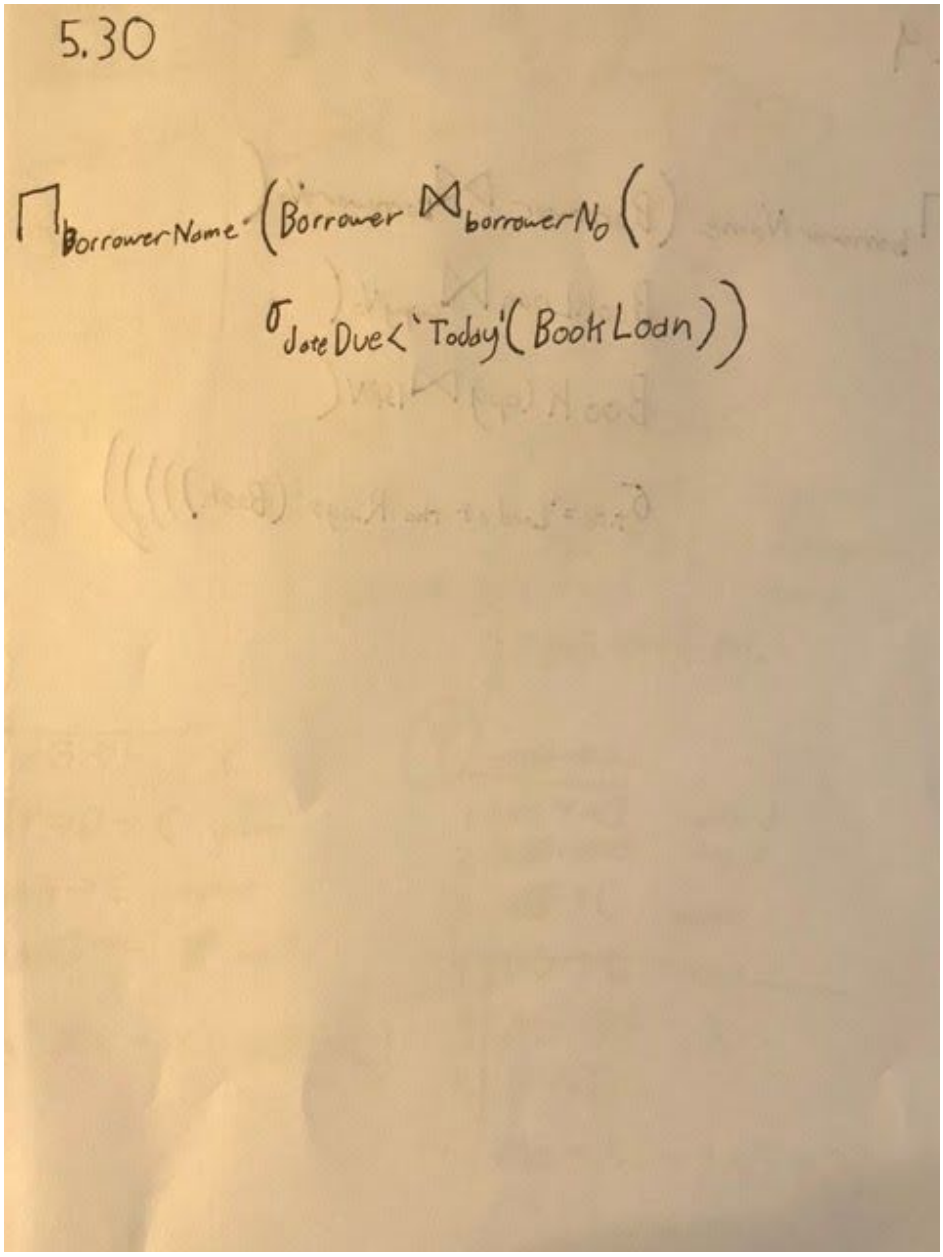
5.29
{R.borrowerName | Borrower(R) ^
(∃L)(∃K)(∃C) (BookLoan(L) ^ Book(K) ^ BookCopy(C)^
L.copyNo = C.copyNo ^
C.ISBN = K.ISBN ^
K.title = “Lord of the Rings” ^
L.borrowerNo = R.borrowerNo) }



5.30)

as per piazza instructions, let 'TODAY' be a constant, comparable with the tuple variable in the domain of BookLoan.dateDue, signifying the current date and time.

{R.borrowerName | Borrower(R) ^
(∃L) (BookLoan(L) ^
L.borrowerNo = R.borrowerNo ^
L.dateDue < 'TODAY') }



6.5)
The WHERE clause is the expression which restricts the rows returned by the query based on the conditional statement that follows it. It can be used to test for comparisons, set membership, can match strings against a regex, and check to see if an attribute is null.
WHERE can also be constructed using boolean operators, AND, OR, NOT to perform compound conditional checks.

GROUP BY divides tuples into sub-lists, where each sub list contains equivalent values for the column name following the group by statement.
when using aggregate functions, the column name which is not called by an aggregate function in the select statement, but does appear as a column name in the result because it was called by SELECT, must be in a GROUP BY.

aggregate functions (max, min, ect) can only be used in the select clause, and after a HAVING. HAVING is used with the GROUP BY clause in order to restrict things about the group, so after grouping by, the HAVING clause functions much like a WHERE, but WHERE cannot be used after GROUP BY. HAVING always includes an aggregate function.

6.9)
List the names and addresses of all guests living in London, alphabetically ordered by name.

```
SELECT guestName, guestAddress
FROM Guest
WHERE guestAddress LIKE %London%
ORDER BY guestName;
```

6.15)
***The way to check if a date is in august is unclear. Similar to the TODAY example above, I will define it myself.*
I will say a date is in August if the first digit in the date string is equal to 8.
Ex, '8-31-21' would be an august appointment for next year.
*(I'll assume the database is clean, so '8 thousand BC' will also be in august. This is SQL as a query not a management. I wont worry about edge cases here. I also understand a stay might contain august without having it in the to/from dates, piazza said checking both was fine.)***

```
SELECT COUNT(*) AS count
FROM Booking
WHERE (dateFrom LIKE '8%') OR (dateTo LIKE '8%');
```

6.18)
***Like the previous date, time questions in this case I will use 'TODAY'. If 'TODAY' is between a booking's start and end, it will be occupied.*
*'TODAY' would be replaced with a date/time representing the current date/time.***

```
SELECT Room.roomNo, Room.hotelNo, type, price, guestName
FROM Booking, Room, Guest
WHERE ('TODAY' BETWEEN dateFrom AND dateTo) AND
Room.hotelNo = Booking.hotelNo AND
Guest.hotelNo = Booking.hotelNo;
```

6.46)

```
SELECT Borrower.borrowerName
FROM BookLoan, Book, BookCopy, Borrower
WHERE BookLoan.copyNo = BookCopy.copyNo AND
BookCopy.ISBN = Book.ISBN AND
Book.title = “Lord of the Rings” AND
BookLoan.borrowerNo = Borrower.borrowerNo;
```

6.52)

```
Select Borrower.borrowerName
FROM (
SELECT ISBN, COUNT(available) as copyCount
FROM BookCopy
GROUP BY (ISBN)
), BookCopy, Borrower, BookLoan
WHERE BookLoan.copyNo = BookCopy.copyNo AND
BookCopy.ISBN = Book.ISBN AND
ISBN = bookCopy.ISBN AND
BookLoan.borrowerNo = Borrower.borrowerNo AND
copyCount > 3;
```

7.19) For our materialized view SupplierParts, which contains the distinct part numbers that are supplied by at least one supplier:
To maintain this as a materialized view, relevant information, that is, information about parts supplied by at least one supplier rows can be inserted or deleted when

1. A part that was not supplied becomes supplied by at least one supplier
2. A new part is added to the part table, and it is supplied by a new or previously existing supplier.
3. A part that used to be supplied becomes unsupplied by any supplier.

These events would require having access to the Part and Supplier tables.
However when

1. A part that was supplied by one supplier becomes supplied by more than one supplier, or
2. A supplier stops supplying a part, but it is still supplied by at least one other supplier
3. A part supplied by nobody is removed or inserted into the parts list
4. A supplier that did not offer any unique parts is removed from the supplier table

We would not need access to the Supplier or Part tables to maintain the view.
The reason for not needing access, is because the view would not change.

Any changes to the tables that DO NOT add a part which passes the equality condition following the WHERE, or change a part that used to into one that does not, do not need to be inserted/removed from the materialized view in order to maintain it. Those cases would therefore not require access to the underlying tables: Part and Supplier.

7.25)
Create a view consisting of the Employee and Department tables without the address, DOB, and sex attributes.

```
CREATE VIEW PublicInformation (empNo, fName, lName, position, deptNo, deptName, mgrEmpNo) AS
SELECT DISTINCT empNo, fName, lName, position, deptNo, deptName, mgrEmpNo
FROM Employee, Department
Where Employee.deptNo = Department.deptNo;
```

8.9)
The advantages of triggers are similar to the advantages of writing functions. Private, secure, deduplicated code that can be reused and run locally on the server, and supports good architecture design (among other upsides listed in the book). Triggers are one of the only control structures native to SQL, which is part of the reason our last program was written in Java. They allow us to check, safeguard, and update the database without running a program or putting it a query.

Some downsides however: as long as the database is running, so are they. They are unpredictable, they slow the data manipulation, and may have to be rewritten (as the book discusses). In my opinion, they are definitely worth it to include in SQL related database design, even though they seem clunky, limited, and non-intuitive. Nobody likes to manually clean up data, triggers save us the trouble.

8.11 a)

```
CREATE TRIGGER underpricedDouble
BEFORE UPDATE OF price ON Room
REFERENCING NEW ROW AS new
FOR EACH ROW WHEN (:new.type = ‘double’ AND NOT :new.price > 100)
DECLARE
invalid_price EXCEPTION;
BEGIN
RAISE invalid_price
EXCEPTION
END
```

14.15 a)
insertion of new patients and staff would HAVE to schedule an appointment (appointment time can't be null, its a PK).
deletion of a staff member would require deletion of all their appointments, and the data that is stored in those appointments, like patient name.
modification of any column would require modifying many other columns, for example if a patient switched their preferred surgeon.

***some additional thoughts, I don't think they count as anomalies though:*
A dentist might somehow get scheduled to perform their own surgery.
Two staff members cannot perform surgery at the same time.
an erroneous insertion into a table with so much duplicated data could easily cause patients, doctors, ect to be overbooked. Two patients could end up with the same IDs or worse, an important surgery could go overlooked.
The primary key seems to be the time. What if there are no surgeries scheduled for that time, all of the other keys would have to be null.
*and much more craziness***

b)
functional dependencies:
AppointmentDateTime is a PK

1. AppointmentDateTime->staffNo,dentistName,patNo,patName,surgeryNo
2. staffNo->dentistName
3. patNo->patName
4. staffNo,patNo,AppointmentDateTime->dentistName,patName,surgeryNo

names could technically be on the left hand side too, but I am intentionally leaving them out due to context and better judgement as discussed in class.

Non-book:
17) BDE+ has closure {ABCDEFG}
ABCD+ has closure {ABCD}
neither are CK

18)
AG->F
F->D
D->C
C->H
H->E