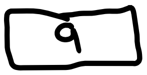
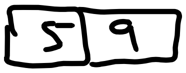
Question 1:

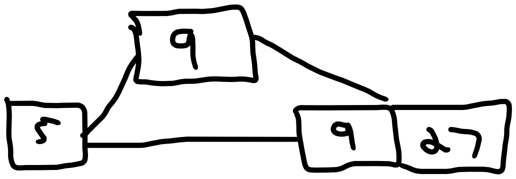
Insert 9:



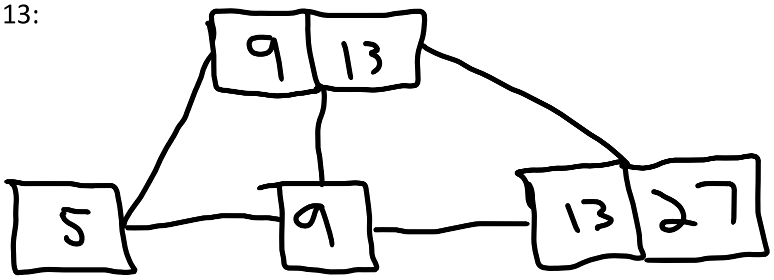
Insert 5:



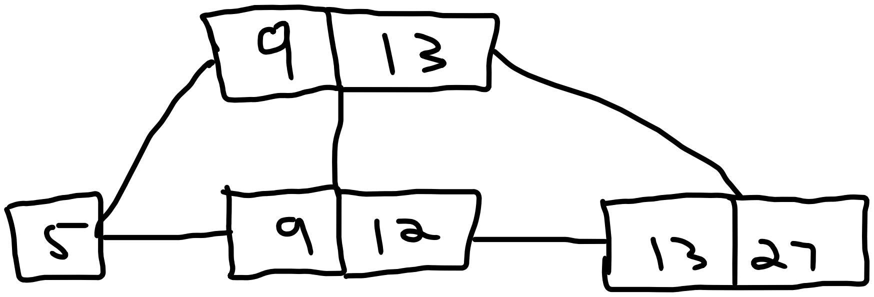
Insert 27:



Insert 13:

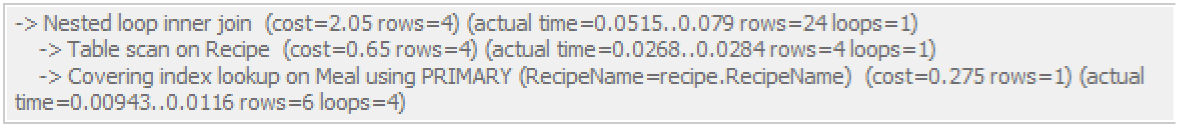


Insert 12:



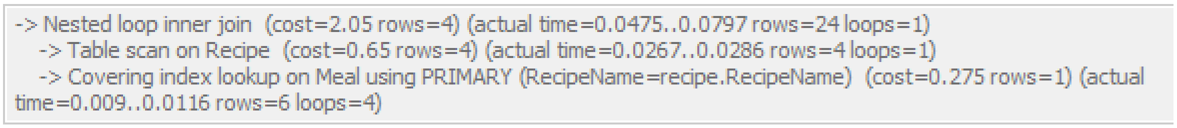
Question 2:

a)



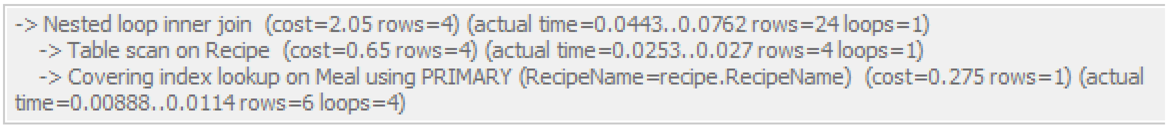
```
-> Nested loop inner join  (cost=2.05 rows=4) (actual time=0.0515..0.079 rows=24 loops=1)
   -> Table scan on Recipe  (cost=0.65 rows=4) (actual time=0.0268..0.0284 rows=4 loops=1)
   -> Covering index lookup on Meal using PRIMARY (RecipeName=recipe.RecipeName)  (cost=0.275 rows=1) (actual
time=0.00943..0.0116 rows=6 loops=4)
```

b)



```
-> Nested loop inner join  (cost=2.05 rows=4) (actual time=0.0475..0.0797 rows=24 loops=1)
   -> Table scan on Recipe  (cost=0.65 rows=4) (actual time=0.0267..0.0286 rows=4 loops=1)
   -> Covering index lookup on Meal using PRIMARY (RecipeName=recipe.RecipeName)  (cost=0.275 rows=1) (actual
time=0.009..0.0116 rows=6 loops=4)
```

c)



```
-> Nested loop inner join  (cost=2.05 rows=4) (actual time=0.0443..0.0762 rows=24 loops=1)
   -> Table scan on Recipe  (cost=0.65 rows=4) (actual time=0.0253..0.027 rows=4 loops=1)
   -> Covering index lookup on Meal using PRIMARY (RecipeName=recipe.RecipeName)  (cost=0.275 rows=1) (actual
time=0.00888..0.0114 rows=6 loops=4)
```

Based on my knowledge of SQL, it is always better to use a join over cartesian product due to the filtering of results instead of returning all. In the case of these 3 explain analyze results, it seems like they were either optimized or the database is too small to show a significant difference in performance. The best performing was the inner join with a where clause, though I would suspect through multiple runs and a larger database it would be the second query with just the join on the two tables.

Question 3:

The decision to use an index in a database will give it hints on how to store the data. This will give it fast look up times in volatile memory. This comes at the cost of storage in your volatile memory which could be better partitioned to other data. Creates problems when there are too many indexes in volatiles memory. This should be saved for non-primary/foreign keys that are constantly being used. Ensuring that the cost benefits outweigh the negatives. An example could be with the summervacation database. We added the index to State which would make sense in a large database where many of the queries depend on that piece of data. In the case of our database, it does not seem necessary or to carry enough benefit.

Question 4:

```sql
Select p.UserName, g.GameName, min(s.Score) as LowScore, max(s.Score)as HighScore, avg(s.Score) as AverageScore

from Score as s, Player as p, Game as g

Where g.Id = p.FavoriteGame and s.PlayerId = p.Id

group by p.UserName, g.GameName;
```

```sql
Select p.UserName, g.GameName, min(s.Score) as LowScore, max(s.Score)as HighScore, avg(s.Score) as AverageScore

from Score as s

Join Player as p on s.PlayerId = p.Id

Join Game as g on g.Id = p.FavoriteGame

group by p.UserName, g.GameName;
```

```sql
Select p.UserName, g.GameName, min(s.Score) as LowScore, max(s.Score)as HighScore, avg(s.Score) as AverageScore

from Game as g

Join Player as p on p.FavoriteGame = g.Id

Join Score as s on s.PlayerId = p.Id

group by p.UserName, g.GameName;
```

Question 5:

Volatile:

- Volatile storage only works when the power is on
- Subject to data loss if the power switches off
- Must faster to access
- Primary storage

Non-volatile storage:

- Data will not be lost when power is off
- Secondary and tertiary storage
- Slower access time

Question 6:

Primary:

- Volatile Storage
- Fastest access time
- Cache
- Data in use

Secondary:

- Non-Volatile Storage
- Moderate access time
- Flash memory, magnetic disks
- Data not in use

Tertiary:

- Non-Volatile Storage
- Slowest access time
- Magnetic tape, optical storage
- Used for archival storage

Question 7:

a)

Physical Level – Describes how a record is stored on a disk.

- Unlikely to be known by the programmer and may be decided by a hardware engineer or other discipline that finds the most optimal place for it to reside.

Logical Level – Describes data stored in a database, and the relationships among it. (Database Schema)

- This level will be where the database programmers are designing, querying, and creating new relationships between data.

View Level – Application programs hide details of data types. Views can hide information for security purposes.

- This would be users of the application or programmers who do not know the underlying systems that make it work.

b)

Presentation:

- Model View Controller
- How data is presented to a user such as a graphical user interface
- Dependent on the user device
- Receives events, executes actions, and returns the view to the user

Application:

- Business Logic Layer
- A high level view on the data and actions that can be performed on it
- Hides details of data storage
- Often uses object data model

Database Layer

- Data Access Layer
- Interface between business logic layer and the database
- Provides object model mapping from business layer to relational model of database

Question 8:

A cartesian product will return all possible combinations, making it far less efficient. On the other hand, joins can be used and match tuples with the same value before a where clause is executed. Therefore, far less data needs to be filtered that match the conditions.

Question 9:

Mirroring:

- Storing extra information that can be used during a failure
- Duplicates all disks and all writes take place on both disks
- Prevents data loss unless both disks fail which is highly unlikely

Stripping:

Bit level:

- Splits bits across multiple disks
- Each access can read the data at eight times the rate
- Seek time becomes worse

Block level:

- Requests can run in parallel if the blocks are on different disks.
- Requests for a long sequence of blocks can utilize disks in parallel

Question 10:

Function:

```
DROP FUNCTION IF EXISTS DoesParkExist;

delimiter $$

Create Function DoesParkExist(ParkToCheck varchar(200))

Returns BOOl deterministic

BEGIN

            declare NumPark int;

   select count(*) into NumPark

   from Park

   Where ParkName = ParkToCheck;

return NumPark > 0;

END $$

delimiter ;
```

Procedure:

```
delimiter $$

DROP PROCEDURE IF EXISTS addActivityToPark $$

Create Procedure addActivityToPark(newActivityName varchar(100), newNumPlayers int, newParkName
varchar(200))

BEGIN

If not exists(

Select *

From SummerPlan

Where ActivityName = newActivityName and ParkName = newParkName

)

Then

        Insert Into Activity (ActivityName, NumPlayers) Values (newActivityName, newNumPlayers);

        Insert Into SummerPlan (ActivityName, ParkName) Values (newActivityName, newParkName);

End If;

END $$

Delimiter ;
```

Question 11:

I really enjoyed working in the smaller databases throughout this class since I had taken a few SQL courses on my own to prepare. It is nice to not be overwhelmed and be able to play around with them outside of class. For improvement it may not be possible due to the shortened semester but some of the slides are very text dense which can make it hard to visualize a concept or go back to review. I feel throughout this course I obtained an understanding of how to work with data and design a database. Combining this with my programming experience it gives a better understanding about the back end and how systems come together as a whole. Thank you for the semester, enjoy the holidays!