Student ID: H00353899

Q1 (a)

(i) With reference to the ANSI/SPARC three tier architecture, what are the different schemas, and explain the effect to the schemas of adding a new index? (3 marks)

Answer:

There are three different types of schema corresponding to the three levels in the ANSI-SPARC architecture:

- The external schemas describe the different external views of the data and there may be many external schemas for a given database.
- The conceptual schema describes all the data items and relationships between them, together with integrity constraints (later). There is only one conceptual schema per database.
- The internal schema at the lowest level contains definitions of the stored records, the methods of representation, the data fields, and indexes. There is only one internal schema per database.

The overall description of a database is called the database schema.
[reference] https://en.wikipedia.org/wiki/ANSI-SPARC_Architecture

explain the effect to the schemas of adding a new index:

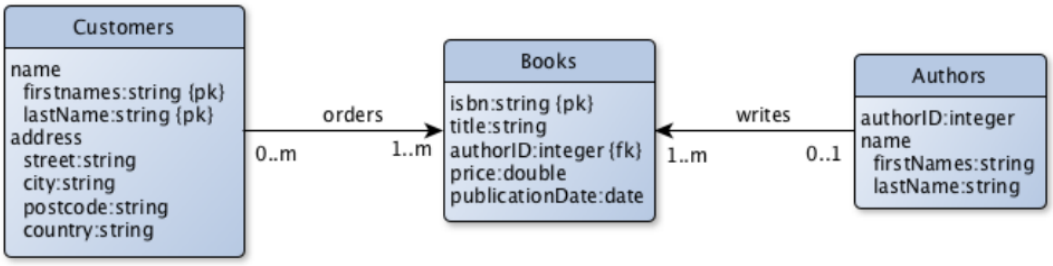
- Physical/internal schema can be changed without changing application: e.g. we may add or remove an index
- DBMS must change mapping from conceptual to physical.
- Referred to as physical data independence.

[reference] http://jcsites.juniata.edu/faculty/rhodes/dbms/dbarch.htm

(ii) With respect to the following data requirements, **identify and explain** the errors or problems in the ER diagram below. *(Marks will only be given when there is an explanation)*

- Customers are only stored in the system once they have ordered a book
- Customers should be able to order many books, the date of the order should be captured
- A book can be bought by many customers, but does not need to be bought by any
- A book must have one author but can have more
- An author can write many books
- Authors are only stored if they have written at least one book

(4 marks)



Answer:

Error #1: PKs fields of “Customers” table.
Explanation #1: In the table “Customers”, there are two PKs, and the PK is set to names, which is not unique for each customer, since in this world, many persons share the same names. The correct PK of the customers should be an unique, not null, customer number.

Error #2: The authorID field of “Books” table.
Explanation #2: A book must have one author but can have more. But in the table Books, there is only one author field, which limits the number author for each book to only one. The correct way is to leave this authorID field to an additional table “writes” as follows,

writes
authorID: integer {fk}
isbn: integer {fk}

Error #3: The “Authors” table does not have a PK.
Explanation #3: The field authorID of “Authors” should be set to PK since each author is a unique existing.

Error #4: Cardinality Constraint of “writes” is wrong.
Explanation #4: One author can write more than one books, and one book can have more than one authors. So the Cardinality Constraint on both sides are “1..m”. However, in the ER-diagram, the constraint over the authors side is only “0..1”

Error #5: Cardinality Constraint of “orders” is wrong.
Explanation #5: Customers are only stored in the system once they have ordered a book. This puts the constraint over Customers as “1...m” not the “0..m” as shown in the ER-diagram.

Error #6: The date of the order is missing from the orders table.
Explanation #6: In current ER-diagram, the order table does not have the field of the order date.

Q1. (b) Considering the following schemas and sample data.
(underlined attributes in the schema indicate the primary key)

`Customers(cid:int, name:string)`
`Boats(bid:int, bname:string, colour:enum[red,blue]`
`Reserves(cid:int, bid:int, day:date)`

Customers	
cid	**name**
1	Anne
2	Bob
3	Carol

Boats		
bid	**name**	**colour**
9001	Edinburgh	red
9002	Dubai	blue

Reserves		
cid	**bid**	**day**
1	9002	2020-12-01
3	9002	2020-12-11

(i) State what happens with each of the SQL statements and **explain** why the DBMS does this.

```
A: INSERT INTO Reserves
VALUES (4, 9001, 2020-12-12)
B: INSERT INTO Boats
VALUES (9003, 'Malaysia', 'yellow')
C: INSERT INTO Reserves
VALUES (NULL, 9001, 2020-12-07)
```

(3 marks)
Answer:
A. Insert the record to the table successfully
B. Insert fails. **Explain**: The 'yellow' is not one of the `enum[red,blue]`
C. Insert fails. **Explain**: The primary key `cid` is NULL.

(ii) Write an SQL query to find the names of customers who have hired a blue boat. (4 marks)
Answer:

In [24]:
```
%%sql
select Customers.name
from Reserves, Customers, Boats
where Reserves.cid = Customers.cid
and Reserves.bid = Boats.bid
and Boats.colour = "blue";
```

```
 * mysql+pymysql://jessica:***@local
2 rows affected.
```

Out[24]:

name
Anne
Carol

(iii) Write an SQL query to find the names of customers who have not reserved a boat. (3 marks)

Answer:

In [63]:
```
%%sql
select Customers.name
from Customers
left join Reserves
on Reserves.cid = Customers.cid
where Reserves.cid is null;
```

```
 * mysql+pymysql://jessica:***@localhost:3306/jes
1 rows affected.
```

Out[63]:

name
Bob

(iv) Write an SQL query to find the number of reservations per boat for each day.

Answer: To show the function of this query I have made some changes to the Reserves table by adding five more rows and the new table is

cid	bid	day
1	9002	2020-12-01
3	9002	2020-12-11
1	9001	2020-12-03
3	9001	2020-12-03
3	9001	2020-12-03
1	9001	2020-12-03
3	9001	2020-12-03

The query SQL and results.

In [62]:
```
%%sql
select bid, day, count(*) as reservation_number
from Reserves
group by bid, day;
```

```
 * mysql+pymysql://jessica:***@localhost:3306/jessica
3 rows affected.
```

Out[62]:

bid	day	reservation_number
9001	2020-12-03	5
9002	2020-12-01	1
9002	2020-12-11	1

For more than 4 hires in a day, return the name of the boat, together the date in a EUROPEAN style, and the number of times the boat was hired. (5 marks)
Answer:

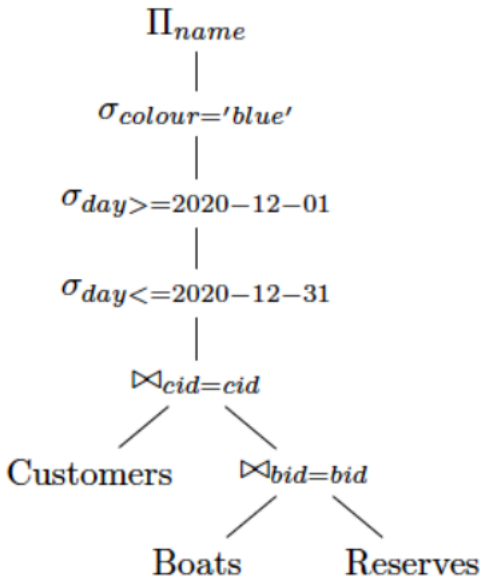
In [64]:
```
%%sql
select bname as boat_name,
concat(substr(day, 9,2), '-',
    substr(day, 6,2), '-',
    substr(day, 1,4)) as date_EUROPEAN_style,
reservation_number as boat_hired_time_number
from (
    select Reserves.bid, cast(day as varchar(100)) as day,
    Boats.bname, count(*) as reservation_number
    from Reserves, Boats
    where Boats.bid = Reserves.bid
    group by bid, day
) a
where reservation_number > 4;
```

```
 * mysql+pymysql://jessica:***@localhost:3306/jessica
1 rows affected.
```

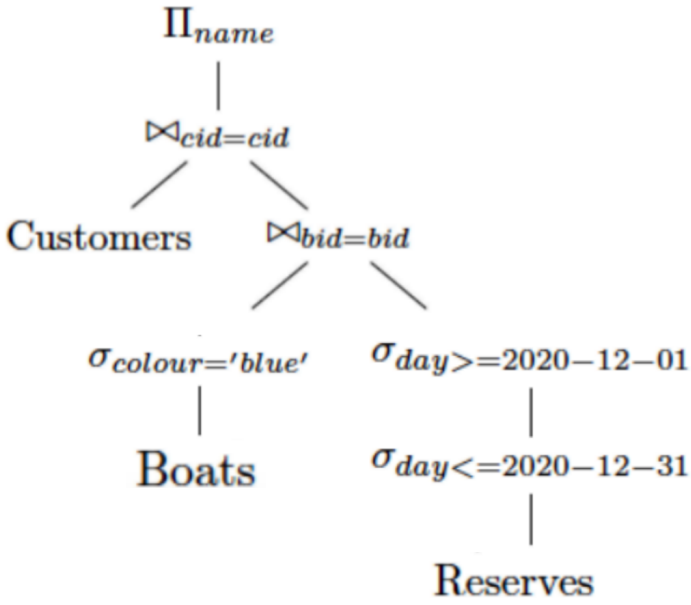
Out[64]:

boat_name	date_EUROPEAN_style	boat_hired_time_number
Edinburgh	03-12-2020	5

(v) Consider the following query plan for a query that returns the name of customers who have hired blue boats in December 2020. Restructure it into a more efficient plan for pipeline query processing. State the steps that you have taken and **explain** why they improve the performance. (4 marks)



Answer:
The reconstructed query plan tree is



Steps:
1. Moving the selections of day>2020-12-01 and day<2020-12-31 below the join bid=bid
2. Moving the selections of colour='blue' below the join bid=bid
3. Move the join cid=cid bellow projection name

Explain: the selections of day>2020-12-01, day<2020-12-31 and colour='blue' is independent of the joints. Moving them to the joints will not effect the joints output. However, doing the selection can firstly reduce the data size significantly, and the inputs for the joining is much smaller. So the most time-consuming steps of joining can be more efficient.

Q1 (c) Consider the following set of operations from two transactions T1 and T2 where the transaction number indicates the time point that the transaction started, the number at the start of the line is to provide a point of reference in writing your answers.

State and explain what happens at each step of processing when using Timestamp Concurrency Control. Ensure that you state any assumptions that you make while answering the question.

1: T1 Reads value X
2: T2 Reads value Y
3: T1 Reads value Y
4: T2 Writes value X
(4 Marks)

Answer:

Step 1 the user read the X at time1.

Step 2 the user read the value Y at time 2.

Step 3, at the same time 1, the user read value Y

Step 3, at the same time 2, the user read value X.

Q2 (a)

(i) One of the 5 Vs of Big Data is *Velocity*. Explain what it means and what this has to do with NoSQL (3 marks)

Answer:

- **what it means**: the speed of data processing/how fast the data is coming in.

[reference] https://www.bbva.com/en/five-vs-big-data/
https://whatis.techtarget.com/definition/3Vs

- **what this has to do with NoSQL**: The NoSQL is more suitable than SQL database to fit the Velocity. The reason is the the NoSQL database stores the data in a form of flat collections. The single piece of data is usually not partitioned and stored in a form of entity. Read and writing a single entity is faster and thus much more easier than SQL. This nature makes NoSQL a better choice to handle the read-time processing of the fast-coming data.

[reference]
https://www.dezyre.com/article/nosql-vs-sql-4-reasons-why-nosql-is-better-for-big-data-applications/86

(ii) Identify 3 things that are incorrect in the following XML (3 marks)

```
<data>
  <person>
    <firstname>Tim</firstname>
    <surname>Smith</surname>
    <tel>01318234479
        <email>tim@example.com </email>
    </tel>
  <person>
    <firstname>Lisa</firstname>
    <surname>Jones</surname>
  </person>
<data>
```

Answer:

Errror 3: The value of <tel> 01318234479 is not quotated. It should be "01318234479".

Errror 1: Elements are not closed: the first <person> should be coupled with a </person> at this location, but it is missing

Errror 4: <tel> and <email> are not nested propoerly. The <email> should not be in <tel> since <tel> has its own body. The correct nesting is <tel>"01318234479"</tel> <email>tim@example.com </email>

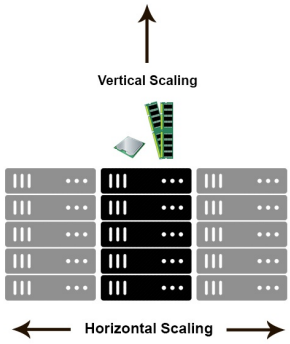
```
<data>
  <person>
    <firstname>Tim</firstname>
    <surname>Smith</surname>
    <tel>01318234479
        <email>tim@example.com </email>
    </tel>
  <person>
    <firstname>Lisa</firstname>
    <surname>Jones</surname>
  </person>
<data>
```

Errror 2: Elements are not closed: <date> should be chagned to </data>

(iii) Explain the difference between *vertical* and *horizontal scaling* (2 marks)

Answer:

- **Horizontal scaling** means that you scale by adding more **machines** into your pool of resources whereas
- **Vertical scaling** means that you scale by adding more **power (CPU, RAM)** to an existing machine.



[reference]
https://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases

Detailed Differences:

	Horizontal Scaling (scaling out)	Vertical Scaling (scaling up)
Databases	In a database world, horizontal scaling is usually based on the partitioning of data (each node only contains part of the data).	In vertical scaling, the data lives on a single node and scaling is done through multi-core, e.g. spreading the load between the CPU and RAM resources of the machine.
Downtime	In theory, adding more machines to the existing pool means you are not limited to the capacity of a single unit, making it possible to **scale with less downtime.**	Vertical scaling is limited to the capacity of one machine, scaling beyond that capacity can involve downtime and has an upper hard limit, i.e. the scale of the hardware on which you are currently running.
Concurrency	Also described as distributed programming, as it involves distributing jobs across machines over the network. Several patterns associated with this model: Master/Worker*, Tuple Spaces, Blackboard, MapReduce.	Actor model: concurrent programming on multi-core machines is often performed via multi-threading and in-process message passing.
Message passing	In distributed computing, the lack of a shared address space makes data	In a multi-threaded scenario, you can assume the existence of a

	sharing more complex. It also makes the process of sharing, passing or updating data more costly since you have to pass copies of the data.	shared address space, so data sharing and message passing can be done by passing a reference.
Examples	Cassandra, **MongoDB**, Google Cloud Spanner	MySQL, Amazon RDS

[reference] https://www.section.io/blog/scaling-horizontally-vs-vertically/

(iv) Consider the following property information (i.e. building) and write in JSON format (2 marks)

```
Property Type: House
Bedrooms: 3
Location: Morningside
Price: £600,000
```

Answer:

```
{
  "Property Type": "House",
  "Bedrooms": 3,
  "Location": "Morningside",
  "Price": "£600,000"
}
```

Q2 (b) Load the following data into your PostgreSQL server and answer the questions below showing your workings (i.e. SQL).

(i) How long is *route1* in metres (1 decimal place will do) ? (remember to show your SQL too) (1 mark)
Answer: 1227.7 metres

Query Editor | Query History

```
1  select ST_LENGTH(geom) as route1_length
2  from route1
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	route1_length double precision
1	1227.70000320404

(ii) How long are all the roads in this dataset in metres (to 1 dp)? (1 mark)
Answer: 117467.3 metres

Query Editor | Query History

```
1  select sum(ST_LENGTH(geom)) as all_road_length
2  from roads
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	all_road_length double precision
1	117467.303938496

(iii) How many *letter boxes* (pointxclass: 06340457) are within 200 metres of route1? (1 mark)
Answer: 4

Query Editor | Query History

```
1  select count(distinct poi.id) as letter_box_number
2  from poi
3  join route1
4  on ST_INTERSECTS(poi.geom, ST_BUFFER(route1.geom,200 ))
5  where poi.pointxclass = '06340457'
6
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	letter_box_number bigint
1	4

(iv) To the nearest metre what are the closest and furthest Euclidean distances (i.e. straight line) between a cash machine (pointxclass: 02090141) and a Bus Stop (pointxclass: 10590732) in the points of interest dataset. (2 marks)
Answer: closest distances: 8.1 metres
furthest distances: 4722.1 metres

Query Editor | Query History

```
1  drop table if exists cash_matchine;
2  create table cash_matchine as
3  select * from poi
4  where poi.pointxclass = '02090141';
5
6  drop table if exists bus_stop;
7  create table bus_stop as
8  select * from poi
9  where poi.pointxclass = '10590732';
10
11 drop table if exists cash_matchine_bus_stop_dist;
12 create table cash_matchine_bus_stop_dist as
13 SELECT ST_DISTANCE(cash_matchine.geom, bus_stop.geom) as dst
14 FROM cash_matchine
15 CROSS JOIN bus_stop;
16
17 select min(dst) as closest_distance,
18 max(dst) as furthest_distance
19 from cash_matchine_bus_stop_dist;
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	closest_distance double precision	furthest_distance double precision
1	8.06225774829855	4722.09616378154

(v) Which zone has the greatest density of points? Include the point density value (2 marks)

Answer: Zone 1 has the greatest density of point. The point density value is 2752.

Query Editor | Query History

```
1  select zones.zoneid, count(distinct poi.id) as point_density
2  from poi
3  join zones
4  on st_within(poi.geom, zones.geom)
5  group by zones.zoneid
6  order by count(distinct poi.id) desc
7  limit 1;
8
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	zoneid bigint	point_density bigint
1	1	2752

(vi) What are the total road lengths (in metres) per zone? (3 marks)

Answer: The total lengths of each zone are as follows:

Zone 1: 27494.4 metres,

Zone 2: 24819.5 metres,

Zone 3: 34284.0 metres,

Zone 4: 16921.5 metres.

Query Editor | Query History

```
1  select zones.zoneid,
2  sum(ST_LENGTH(roads.geom)) as total_road_length
3  from roads
4  join zones
5  on st_within(roads.geom, zones.geom)
6  group by zones.zoneid;
```

Data Output | Explain | Messages | Notifications | Geometry Viewer

	zoneid bigint	total_road_length double precision
1	1	27494.4321340259
2	2	24819.5029806827
3	3	34283.9987262696
4	4	16921.5480940755

Q2 (c) Load the following data into MongoDB and answer the questions showing your workings.

(i) How many tracks are listed? (1 mark)
Answer: 3503

```
> db.tracks.count()
3503
```

(ii) How many unique artist names begin with “The” or “the” ? (2 marks)
Answer: 12

```
>
> db.tracks.distinct("ArtistName", {$or: [{"ArtistName":/^The/i},
 {"ArtistName":/^the/i}]}).length;
12
>
```

The 12 names begins with “The” or “the” are given by the following code:

```
>
> db.tracks.distinct("ArtistName", {$or: [{"ArtistName":/^The/i}, {"ArtistNam
e":/^the/i}]})
[
        "The Black Crowes",
        "The Clash",
        "The Cult",
        "The Doors",
        "The Police",
        "The Rolling Stones",
        "The Tea Party",
        "The Who",
        "The King's Singers",
        "The 12 Cellists of The Berlin Philharmonic",
        "The Office",
        "The Posies"
]
>
```

(iii) Which genre is the most popular based on track counts? (2 marks)
Answer: “Rock” is the most poplar genre based on track counts. Its track count is 1297.

```
> db.tracks.aggregate([ {$sortByCount: "$Genre"},{$limit:1}])
{ "_id" : "Rock", "count" : 1297 }
>
```

(iv) What is the average track length in milliseconds for each genre (done as 1 query)? (2 marks)
Answer: The average track length in milliseconds for each genre is given as follows by the following query:

```
> db.tracks.aggregate([{$group: {_id:"$Genre",average_track_length: {"$avg":
'$Milliseconds'}}} ])
{ "_id" : "Drama", "average_track_length" : 2575283.78125 }
{ "_id" : "Comedy", "average_track_length" : 1585263.705882353 }
{ "_id" : "Soundtrack", "average_track_length" : 244370.88372093023 }
{ "_id" : "Blues", "average_track_length" : 270359.77777777775 }
{ "_id" : "Metal", "average_track_length" : 309749.4438502674 }
{ "_id" : "World", "average_track_length" : 224923.82142857142 }
{ "_id" : "Classical", "average_track_length" : 293867.5675675676 }
{ "_id" : "TV Shows", "average_track_length" : 2145041.0215053763 }
{ "_id" : "Rock And Roll", "average_track_length" : 134643.5 }
{ "_id" : "Easy Listening", "average_track_length" : 189164.20833333334 }
{ "_id" : "Opera", "average_track_length" : 174813 }
{ "_id" : "Sci Fi & Fantasy", "average_track_length" : 2911783.0384615385 }
{ "_id" : "Rock", "average_track_length" : 283910.0431765613 }
{ "_id" : "Pop", "average_track_length" : 229034.10416666666 }
{ "_id" : "Alternative & Punk", "average_track_length" : 234353.84939759035 }
{ "_id" : "Jazz", "average_track_length" : 291755.3769230769 }
{ "_id" : "Latin", "average_track_length" : 232859.26252158894 }
{ "_id" : "Bossa Nova", "average_track_length" : 219590 }
{ "_id" : "Heavy Metal", "average_track_length" : 297452.9285714286 }
{ "_id" : "R&B/Soul", "average_track_length" : 220066.8524590164 }
Type "it" for more
>
```

(v) What is the most number of times a track appears in the playlist collection? (1 mark)
Answer: 5.

```
> db.playlists.aggregate([ {$sortByCount: "$TrackId"},{$limit:1}])
{ "_id" : 3498, "count" : 5 }
>
```

(vi) Which track(s) do not appear in the playlist collection at all? (2 marks)
Answer:The track with Track 3 and track name “Fast As a Shark” does not appear in the palylist collection at all. The code to query this track is give as follows:

```
> db.tracks.aggregate([
... {
...     $lookup:
...     {
...         from: "playlists",
...         localField: "TrackId",
...         foreignField: "TrackId",
...         as: "tracks_playlists"
...     }
... },
... { "$match": { "tracks_playlists.0": { "$exists": false } } }
... ]).pretty()
{
        "_id" : ObjectId("5fcfba11cbacecefde9d3aa8"),
        "TrackId" : 3,
        "trackName" : "Fast As a Shark",
        "Composer" : "F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman",
        "Milliseconds" : 230619,
        "Bytes" : 3990994,
        "UnitPrice" : 0.99,
        "AlbumTitle" : "Restless and Wild",
        "ArtistName" : "Accept",
        "MediaType" : "Protected AAC audio file",
        "Genre" : "Rock",
        "tracks_playlists" : [ ]
}
>
```