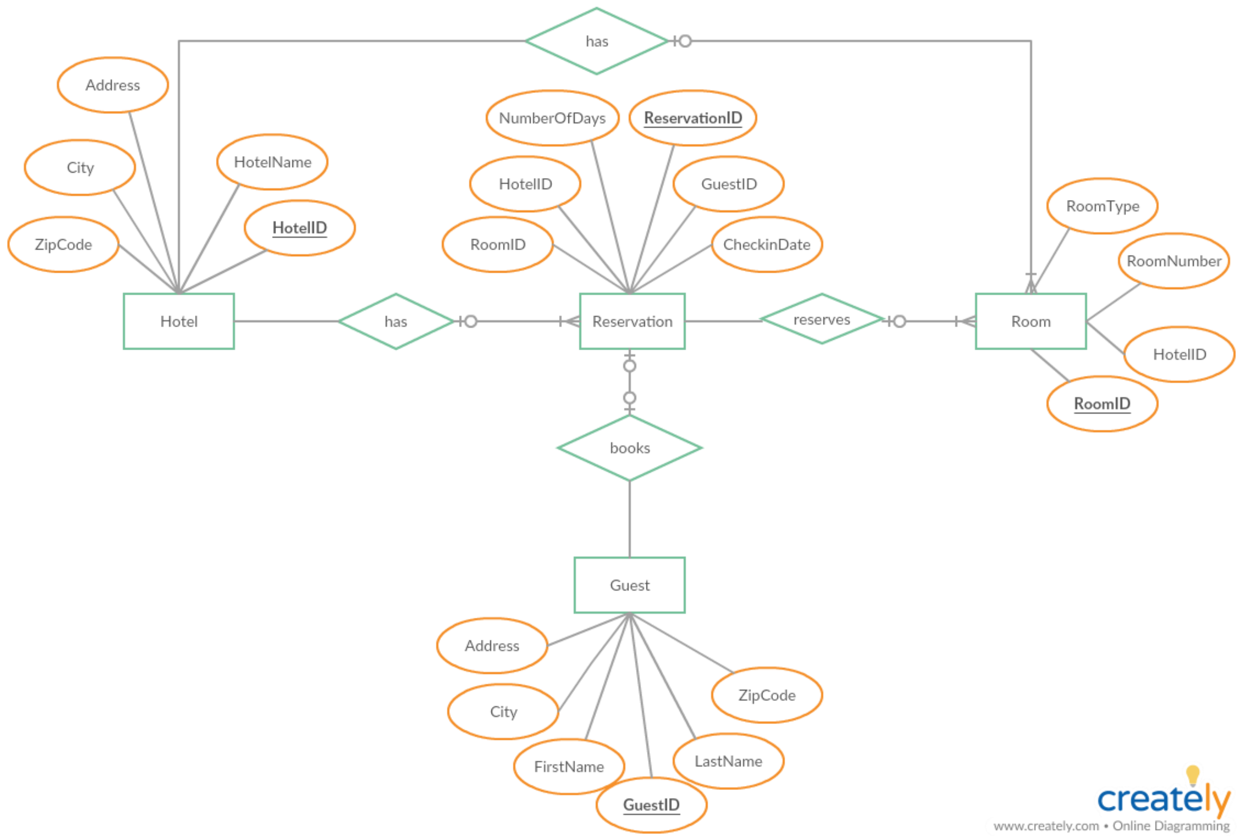
Project Milestone: Logical Design and Data Load

IMT 563, Autumn 2018

Team 07: Aditya Chatterjee, Sahil Aggarwal, Shreya Sabharwal, Akshay Khanna

1. Provide an ER Diagram for your finalized schema.



2. **Create three views that you anticipate will be commonly accessed.**
a. **At least one view should include a join**
b. **At least one view should involve a group by.**
c. **Create a view of your choice**

master_view: This view is a master view that has important details of all the tables of the schema. It captures the total reservations, hotel name, hotel address, guest details, room details of all guests in all hotels in all rooms.

```
-- total reservations with hotel, room and guest data

create view master_view as
select res.reservationid, res.hotelid, res.checkindate, res.guestid,
guest.firstname, guest.lastname, hotel.hotelname, hotel.address as
Hotel_Address,
hotel.city as Hotel_City, hotel.zipcode as Hotel_Zipcode,
res.roomid, room.roomnumber, room.roomtype, guest.address as
Guest_Address,
guest.city as Guest_City, guest.zipcode as Guest_Zipcode
from reservation res
left join guest on res.guestid = guest.guestid
left join room on res.roomid = room.roomid
left join hotel on res.hotelid = hotel.hotelid
```

v_hotel_guest: This view is used to aggregate the number of guests that have stayed in each of the hotels till date.

```
-- how many guests made reservations for each hotel

create view v_hotel_guest as
select hotel.hotelname, count(res.guestid) as No_of_guests
from reservation res
left join guest on res.guestid = guest.guestid
left join hotel on res.hotelid = hotel.hotelid
group by hotel.hotelname
order by No_of_guests DESC
```

v_hotelroom: This view is used to aggregate the number of rooms of each room type in each hotel

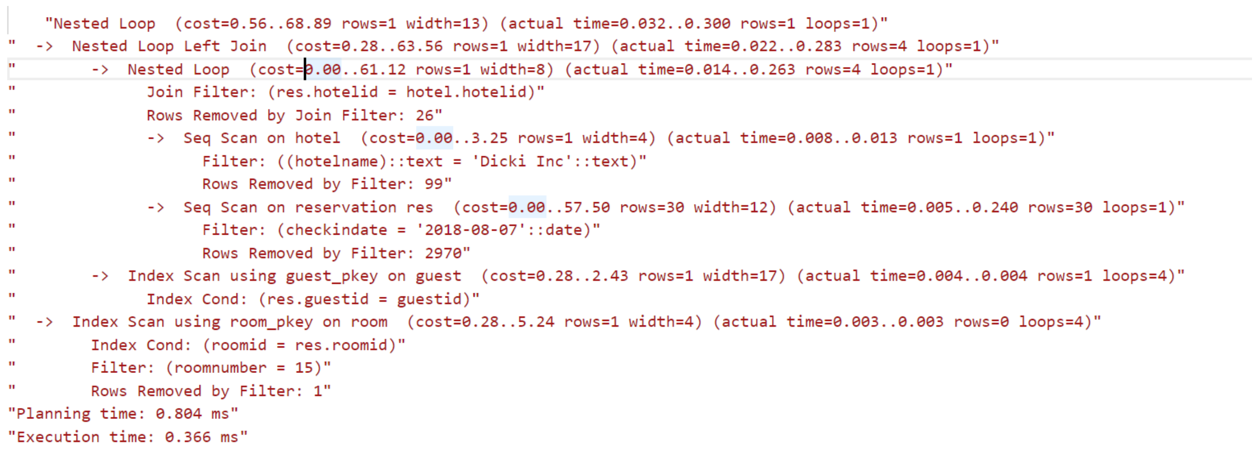
```
-- calculating number of rooms for all room types of all hotels

create view v_hotelroom as
select h.hotelid, count(r.roomid), h.hotelname,
r.roomtype
from hotel h
join room r
on r.hotelid = h.hotelid
group by h.hotelid, r.roomtype;
```

3. **For each of the three views, provide a representative query, generate a query plan using EXPLAIN ANALYZE, and discuss it.**

A. Extracting details of guests staying in room number 15 of hotel 'Dicki Inc'

```
explain analyze
select firstname, lastname from master_view
where roomnumber = '15' and hotelname = 'Dicki Inc'
and checkindate = '2018-08-07'
```

Query Plan:



```
   "Nested Loop  (cost=0.56..68.89 rows=1 width=13) (actual time=0.032..0.300 rows=1 loops=1)"
"  ->  Nested Loop Left Join  (cost=0.28..63.56 rows=1 width=17) (actual time=0.022..0.283 rows=4 loops=1)"
"        ->  Nested Loop  (cost=0.00..61.12 rows=1 width=8) (actual time=0.014..0.263 rows=4 loops=1)"
"              Join Filter: (res.hotelid = hotel.hotelid)"
"              Rows Removed by Join Filter: 26"
"              ->  Seq Scan on hotel  (cost=0.00..3.25 rows=1 width=4) (actual time=0.008..0.013 rows=1 loops=1)"
"                    Filter: ((hotelname)::text = 'Dicki Inc'::text)"
"                    Rows Removed by Filter: 99"
"              ->  Seq Scan on reservation res  (cost=0.00..57.50 rows=30 width=12) (actual time=0.005..0.240 rows=30 loops=1)"
"                    Filter: (checkindate = '2018-08-07'::date)"
"                    Rows Removed by Filter: 2970"
"        ->  Index Scan using guest_pkey on guest  (cost=0.28..2.43 rows=1 width=17) (actual time=0.004..0.004 rows=1 loops=4)"
"              Index Cond: (res.guestid = guestid)"
"  ->  Index Scan using room_pkey on room  (cost=0.28..5.24 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=4)"
"        Index Cond: (roomid = res.roomid)"
"        Filter: (roomnumber = 15)"
"        Rows Removed by Filter: 1"
"Planning time: 0.804 ms"
"Execution time: 0.366 ms"
```

Explain Analyze returns a tree structure of plan nodes representing the actions taken. The first operation is an *index scan* on *room* table with *roomid*. Since roomid is a primary key of room table, the optimizer performs an index scan instead of a sequential scan and a filter of *roomnumber* is applied. Again, an index scan is done on guest table with guestid where *guestid* is a primary key. A sequential scan will be done on *reservation* with checkindate as filter. (The optimizer selects sequential scan because index is not created on *checkindate*.)

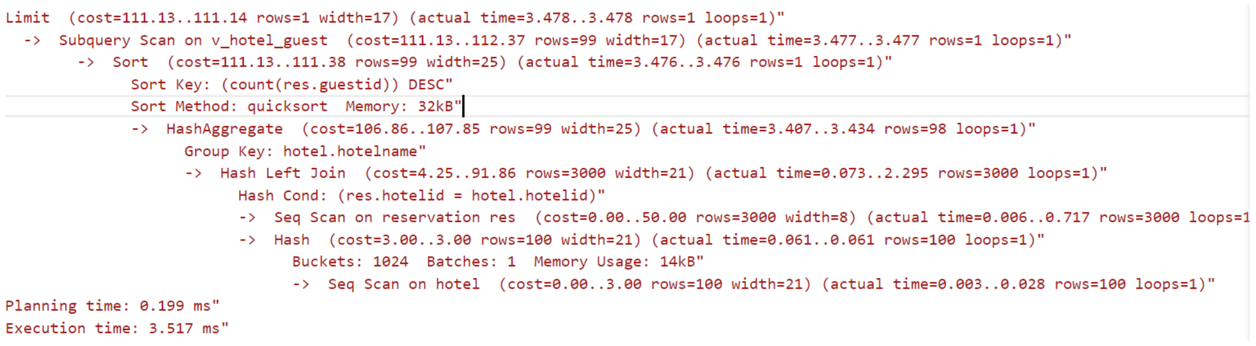
Since the reservation table captures all the transactions and is the largest table, a sequential scan on it is a costly operation. Next, a sequential scan on hotel is planned with *hotelname* as filter (no index on hotelname). A *nested loop* is then planned with join condition of hotelid followed by a nested loop with join condition of guestid. At the end, a nested loop is run to join room and reservation.

The last operation of nested loop is associated with maximum cost (68.89) due to huge size of reservation in nested loop. A nested loop joins two tables by fetching result from one table and querying the other table for each row from the first table. Since these loops are inefficient, these operations are very expensive.

B. Extracting hotel name with maximum guest count till date

```
explain analyze
select hotelname from v_hotel_guest
limit 1;
```

Query Plan:



```
Limit  (cost=111.13..111.14 rows=1 width=17) (actual time=3.478..3.478 rows=1 loops=1)"
  ->  Subquery Scan on v_hotel_guest  (cost=111.13..112.37 rows=99 width=17) (actual time=3.477..3.477 rows=1 loops=1)"
        ->  Sort  (cost=111.13..111.38 rows=99 width=25) (actual time=3.476..3.476 rows=1 loops=1)"
              Sort Key: (count(res.guestid)) DESC"
              Sort Method: quicksort  Memory: 32kB"
              ->  HashAggregate  (cost=106.86..107.85 rows=99 width=25) (actual time=3.407..3.434 rows=98 loops=1)"
                    Group Key: hotel.hotelname"
                    ->  Hash Left Join  (cost=4.25..91.86 rows=3000 width=21) (actual time=0.073..2.295 rows=3000 loops=1)"
                          Hash Cond: (res.hotelid = hotel.hotelid)"
                          ->  Seq Scan on reservation res  (cost=0.00..50.00 rows=3000 width=8) (actual time=0.006..0.717 rows=3000 loops=1
                          ->  Hash  (cost=3.00..3.00 rows=100 width=21) (actual time=0.061..0.061 rows=100 loops=1)"
                                Buckets: 1024  Batches: 1  Memory Usage: 14kB"
                                ->  Seq Scan on hotel  (cost=0.00..3.00 rows=100 width=21) (actual time=0.003..0.028 rows=100 loops=1)"
Planning time: 0.199 ms"
Execution time: 3.517 ms"
```

First, a *sequential scan* is planned on hotel. Next, hashing is done and a sequential scan is done on reservation. *Hash left join* is planned to join reservation with hotel on hotelid column.

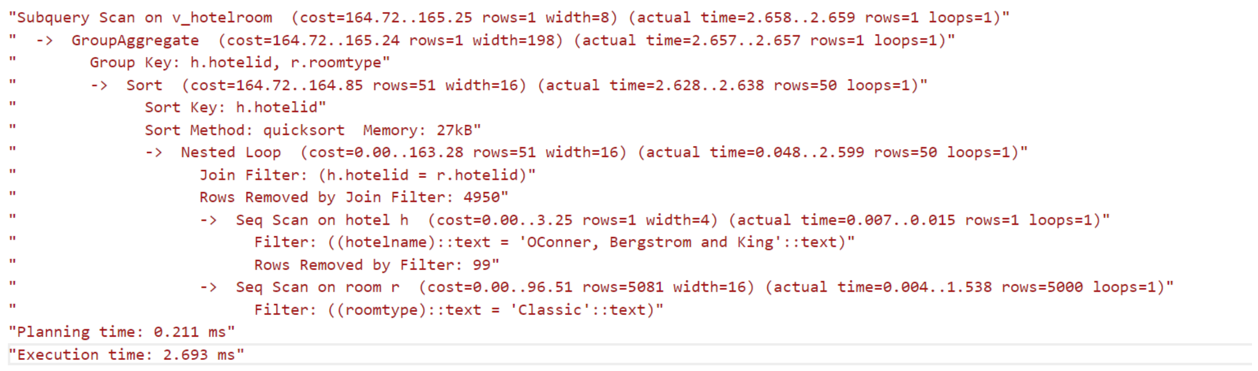
Hash join loads one side of the join in to a hash table which is then probed for each record from the other table on the join. *HashAggregate* is planned to perform group by operation on *hotelname* and has a high cost (107.85). HashAggregate scans the hash table and returns a row per groupby key and performs the aggregation (count in our case). Next, quick sort is planned to perform order by operation on number of guests. A *subquery scan* is then queued up to extract hotel names from the view which has the maximum cost (112.37). At the end, *limit* is planned to fetch the top most row.

Limit is the selection condition here and has a very high cost (111.14). It is not pushed to the leaf node, in fact this is the first node and is planned at the last.

C. Extracting number of classic rooms for hotel, 'OConner, Bergstrom and King'

```
explain analyze
select count from v_hotelroom
where roomtype='Classic'
and hotelname = 'OConner, Bergstrom and King'
```

Query Plan:



```
"Subquery Scan on v_hotelroom  (cost=164.72..165.25 rows=1 width=8) (actual time=2.658..2.659 rows=1 loops=1)"
"  ->  GroupAggregate  (cost=164.72..165.24 rows=1 width=198) (actual time=2.657..2.657 rows=1 loops=1)"
"        Group Key: h.hotelid, r.roomtype"
"        ->  Sort  (cost=164.72..164.85 rows=51 width=16) (actual time=2.628..2.638 rows=50 loops=1)"
"              Sort Key: h.hotelid"
"              Sort Method: quicksort  Memory: 27kB"
"              ->  Nested Loop  (cost=0.00..163.28 rows=51 width=16) (actual time=0.048..2.599 rows=50 loops=1)"
"                    Join Filter: (h.hotelid = r.hotelid)"
"                    Rows Removed by Join Filter: 4950"
"                    ->  Seq Scan on hotel h  (cost=0.00..3.25 rows=1 width=4) (actual time=0.007..0.015 rows=1 loops=1)"
"                          Filter: ((hotelname)::text = 'OConner, Bergstrom and King'::text)"
"                          Rows Removed by Filter: 99"
"                    ->  Seq Scan on room r  (cost=0.00..96.51 rows=5081 width=16) (actual time=0.004..1.538 rows=5000 loops=1)"
"                          Filter: ((roomtype)::text = 'Classic'::text)"
"Planning time: 0.211 ms"
"Execution time: 2.693 ms"
```

Sequential scans are first executed on room to add a filter of Classic room types and on hotel with the filter of specific hotelname (*sequential scan* because of missing index on filters, *hotelname* and *roomtype*). These select conditions are pushed to the leaf nodes.

The results are then passed to a nested loop operation to join hotel with room on hotelid. A quick sort is then planned to sort by hotelid and grouping is scheduled at the end using *GroupAggregate* action with maximum cost (165.24). GroupAggregate is used to perform aggregation on a presorted set based on groupby clause (this also explains the quick sort done before GroupAggregate). The *nested loop* is also expensive (163.28) but small size of tables don’t increase the cost much. At the last, subquery scan is planned to fetch the count column. The cost of this and GroupAggregate is the maximum.