5/7/2017

Final Term Project

Instructor: Periklis A. Papakonstantinou (periklis.research@gmail.com)
TA: Hafiz Asif, Jithesh

Anupam Tiwari (RUID – 161006352) and Jinil Amin (RUID – 176007904)

RUTGERS BUSINESS SCHOOL, NEWARK

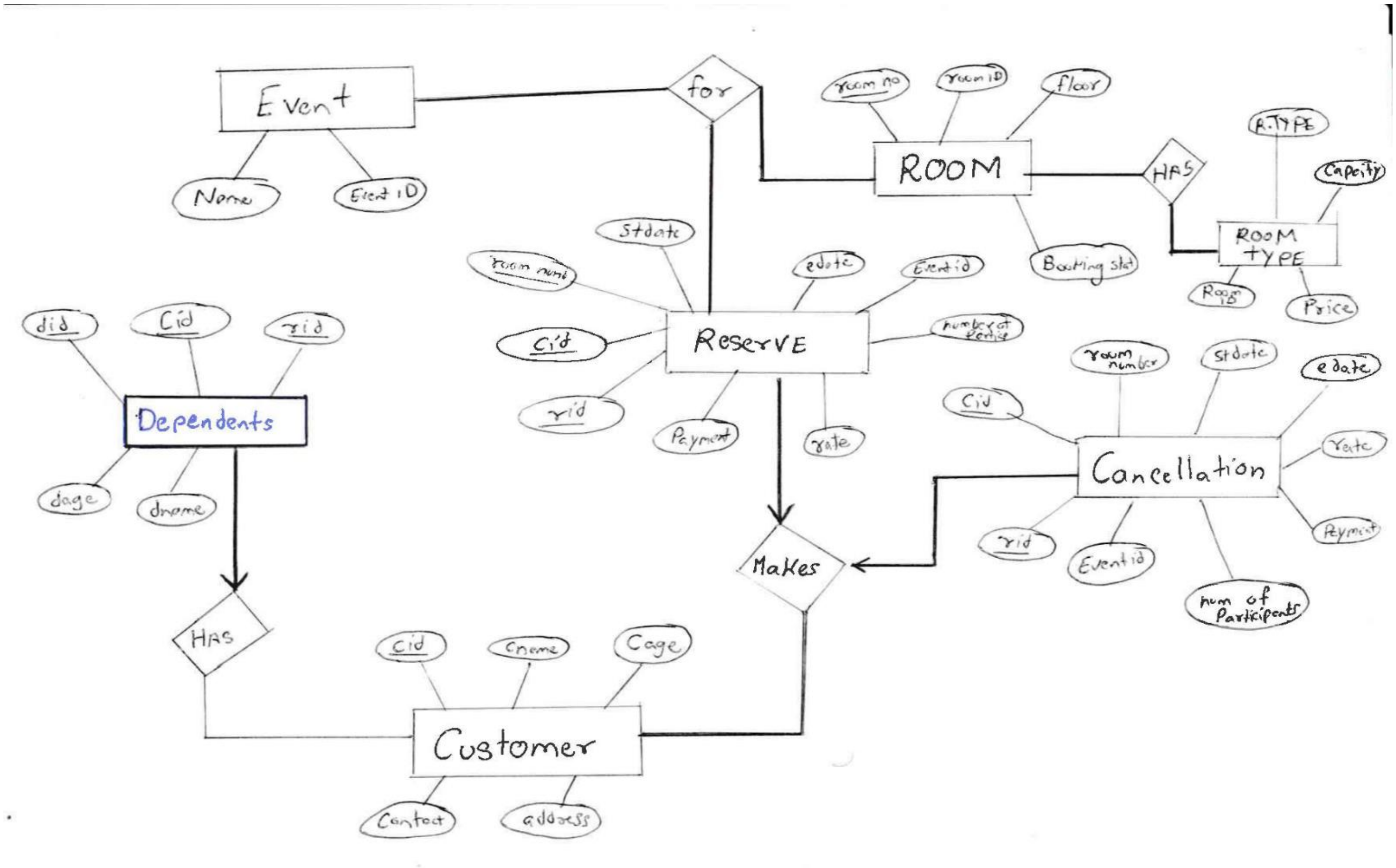
Table of Contents

1. Part 1 – ER Diagram and Constraints

1. General Description and Key Constraints Applied to E-R diagram.

- Two Primary Customer cannot book Reservation for the same date for the same room/ event
- A primary customer must be Above 18 years to make a Reservation.
- Dependents are **weak Entity,** must have a total Participation with Customer. Each customer is unique but can have many dependents shown in E-R.
- The Dependents are linked to Customer, through "Reserve ID, Customer ID."
- If the customer checks out & comes again then each time he will be assigned new reservation ID and even though the customer may have same or different dependents each time, they will be linked with reservation ID given to Customer. So, we can quickly reference back how many times customer visited with different Dependents.
- Customer makes Reservation for Events / Room; and is solely billed for it.
- Each Reservation belongs to only one Customer, but Customer can have many Reservation.
- There exists total Participation between customer and Reservation.
- Customer checks with Reservation for Room Availability.
- Customer Makes a reservation for an Event and is Sole.
- There are Table Value Constraints on the Event Type about how many participants are allowed for that event type
- The customer can cancel Reservation.
- We have assumed "0" for room booking and value of non-zero for Event booking.
- A customer must be enrolled as customer first to book reservation

2. E-R Diagram



2. Part 2 - Relational Model

Relation Scheme

1. Customer(cid:INT, cname:VARCHAR(45), age:INT(3), contact:BIGINT(10), address:VARCHAR(45))

   SQL :-

```
CREATE TABLE customer (
cid int(11) NOT NULL AUTO_INCREMENT,
cname varchar(45) NOT NULL,
age int(3) NOT NULL,
contact bigint(10) NOT NULL,
address varchar(45) NOT NULL,
PRIMARY KEY (cid),
UNIQUE KEY contact_UNIQUE (contact))
ENGINE=InnoDB AUTO_INCREMENT=10038 DEFAULT CHARSET=utf8;
```

2. Dependent(did:INT, cid:INT, rid:INT, dname:VARCHAR(45), dage:INT)

SQL:-

```
CREATE TABLE dependent (
did int(11) NOT NULL AUTO_INCREMENT,
cid int(11) NOT NULL,
rid int(11) NOT NULL,
dname varchar(45) NOT NULL,
dage int(3) NOT NULL,
PRIMARY KEY (did,cid,rid),
KEY drid_idx (rid),
KEY dcid_idx (cid),
CONSTRAINT drid FOREIGN KEY (rid) REFERENCES reserves (rid)
ON DELETE CASCADE ON UPDATE NO ACTION)
ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=utf8;
```

3. Event(eventid:INT, eventname:VARCHAR)

SQL:-

```
CREATE TABLE event (
eventid int(2) NOT NULL,
eventname varchar(45) NOT NULL,
 PRIMARY KEY (eventid)) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4. Payment(pid:INT, rid:INT, cid:INT, amount:DOUBLE, pdate:DATE, Duration:INT)

SQL:-

```
CREATE TABLE payment (
  pid int(11) NOT NULL AUTO_INCREMENT,
  rid int(11) NOT NULL,
  cid int(11) NOT NULL,
  amount double NOT NULL,
  pdate date NOT NULL,
  duration int(11) NOT NULL,
  PRIMARY KEY (pid))
 ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

5. Reserve(rid:INT, cid:INT, stdate:DATE, roomnum:INT, edate:DATE, eventid:INT, participants:INT, Payment:DOUBLE, rate:DOUBLE)

SQL:-

```
CREATE TABLE reserves (
  rid int(11) NOT NULL AUTO_INCREMENT,
  cid int(11) NOT NULL,
  stdate date NOT NULL,
  roomnum int(5) NOT NULL,
  edate date NOT NULL,
  eventid int(2) NOT NULL DEFAULT '0',
  numofparticipants int(3) NOT NULL,
  payment double NOT NULL DEFAULT '0',
  rate double NOT NULL DEFAULT '0',
  PRIMARY KEY (rid,cid,roomnum),
  KEY eveid_idx (eventid),
  KEY rcid_idx (cid),
  KEY rroomnum_idx (roomnum),
  CONSTRAINT eveid FOREIGN KEY (eventid) REFERENCES event (eventid)
  ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT rcid FOREIGN KEY (cid) REFERENCES customer (cid)
  ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT rroomnum FOREIGN KEY (roomnum) REFERENCES rooms (roomnum)
  ON DELETE NO ACTION ON UPDATE NO ACTION)
  ENGINE=InnoDB AUTO_INCREMENT=141 DEFAULT CHARSET=utf8;
```

6. RoomType(roomid:INT, roomtype:VARCHAR, rcapcity:INT, rprice:DOUBLE)

SQL:-

```
CREATE TABLE roomtype (
  roomid int(2) NOT NULL,
  roomtype varchar(20) NOT NULL,
  rcapacity int(3) NOT NULL,
  rprice double NOT NULL,
  PRIMARY KEY (roomid))
```

7. Rooms(roomnum:INT, roomid:INT, floor:VARCHAR, bookingstatus:CHAR)

SQL:-

```
CREATE TABLE rooms (
roomnum int(5) NOT NULL,
 roomid int(2) NOT NULL,
 floor varchar(10) NOT NULL,
 bookingstatus varchar(2) NOT NULL DEFAULT 'NB',
  PRIMARY KEY (roomnum),
  KEY roomid_idx (roomid),
  CONSTRAINT roomid FOREIGN KEY (roomid) REFERENCES roomtype (roomid)
  ON DELETE NO ACTION ON UPDATE CASCADE)
```

8. Cancellations(rid:INT, cid:INT, stdate:DATE, roomnum:INT, edate:DATE, Participants:INT, Payment:DOUBLE, rate:DOUBLE)

SQL:-

```
CREATE TABLE cancellations (
  rid int(11) NOT NULL,
  cid int(11) NOT NULL,
  stdate date NOT NULL,
  roomnum int(5) NOT NULL,
  edate date NOT NULL,
  eventid int(2) NOT NULL,
  numofparticipants int(3) NOT NULL,
  payment double NOT NULL,
  rate double NOT NULL,
  PRIMARY KEY (rid,cid,roomnum))
```

9. Checkout(rid:INT, checkoutstatus:VARCHAR)

SQL:-

```
    CREATE TABLE checkout (
     rid int(11) NOT NULL,
     checkoutstatus varchar(45) NOT NULL DEFAULT 'No',
      PRIMARY KEY (rid),
      CONSTRAINT ckrid FOREIGN KEY (rid) REFERENCES reserves (rid)
ON DELETE CASCADE ON UPDATE NO ACTION
```

Relational Algebra

1. Find the customer (or customers) who paid the highest room rate in 2017 and is also related to at least one more non-primary customer.

Sub Query-1

$$\pi_{rate}\left[\sigma_{\substack{eventid=0\\ \wedge\\ number\ of\ Participant=1\\ \wedge\\ date\geq 1/1/2017\\ \wedge\\ date\leq 12/31/2017}}(RESERVE)\right] - \pi_{rate}\left\{\sigma_{\substack{eventid=0\\ \wedge\\ number=1\\ \wedge\\ date\geq 1/1/2017\\ \wedge\\ date\leq 12/31/2017}}\left[RESERVE \times \rho_{\substack{RID=RID1\\ CID=CID1\\ Stdate=Stdate1\\ roomNo=roomNo1\\ Eventid=Eventid1\\ number\ of\ Participant=number\ of\ Participant1\\ Payment=Payment1\\ rate=rate1}}(RESERVE)\right]\right\}$$

$$\pi_{Customer\ Name}\left\{\sigma_{Sub\ Cid=C.cid}\left[Customer \times \underset{P(Cid=Subcid)}{Subquery\text{-}1}\right]\right\}$$

2. Find the room (or rooms) that was booked for the most consecutive number of days in a single booking by a customer in 2017.

$$\text{Sub Query 1} \longrightarrow \sigma_{\substack{\text{Eventid} = 0 \\ \text{A.Eventid} = \text{B.Enid} \\ \text{date} \geq 1/1/2017 \\ \wedge \\ \text{date} \leq 12/31/2017 \\ \text{a.duration} = \text{date}_2 - \text{date1} \\ \text{b.duration} = \text{date2} - \text{date1} \\ \text{a.duration} < \text{b.duration}}} \left[\text{RESERVE} \times \rho_{\substack{\text{rid} = \text{b.rid} \\ \text{Cid} = \text{b.cid} \\ \text{Stdate} = \text{b.Stdate} \\ \text{Endate} = \text{b.Endate} \\ \text{numofParticipant} = \text{Participants} \\ \text{Payment} = \text{b.Payment} \\ \text{Rate} = \text{b.Rate}}} (\text{RESERVE}) \right]$$

$$\pi_{\text{room}} \left\{ \left[\sigma_{\substack{\text{eventid} = 0 \\ \text{date1} \geq 1/1/2017 \vee \text{date2} \leq 12/31/2017 \\ \text{duration} = \text{date2} - \text{date1}}} (\text{RESERVE}) \right] - \text{Sub Query 1} \right\}$$

SQL

Triggers applied in Database

Triggers in Customer Table:

1. **Before INSERT:**

   - Validate age of a customer. Age should be greater than 18 and less than 110
   - Validate customer's contact number, if the number is already present then no need to update customer's detail again as he/she is already a customer of Hotel.

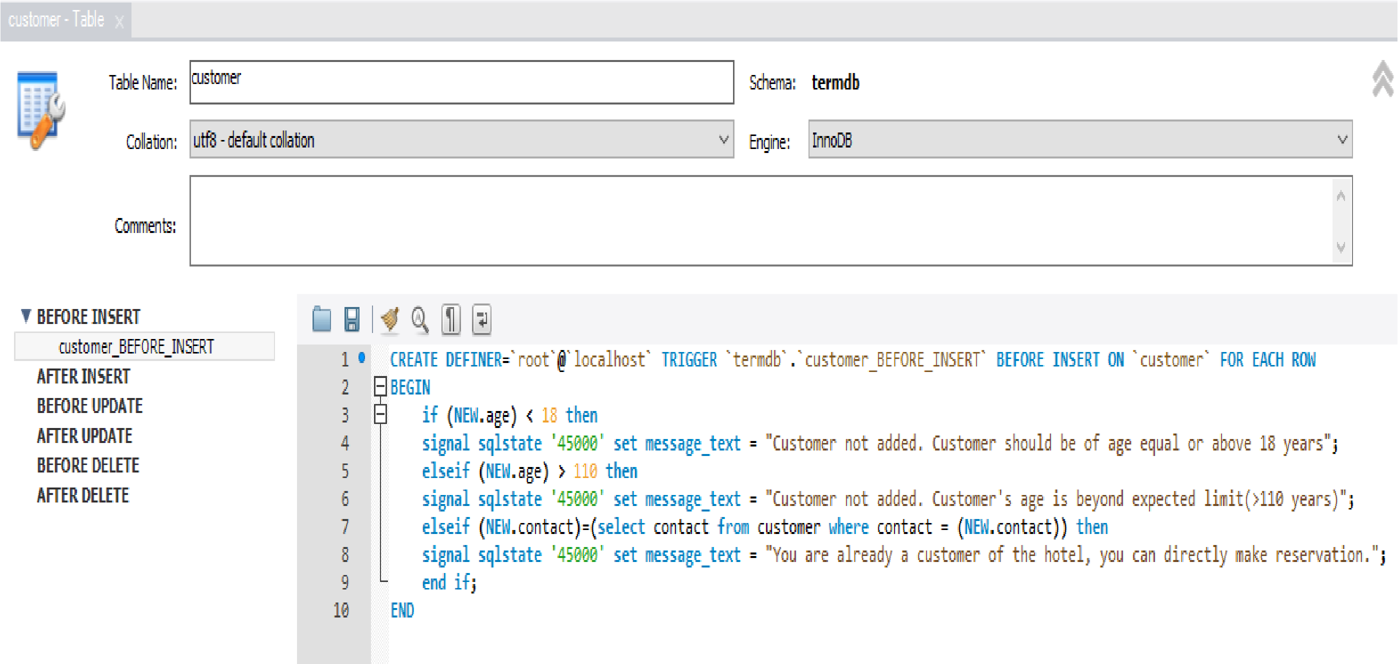


Table Name: customer

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`customer_BEFORE_INSERT` BEFORE INSERT ON `customer` FOR EACH ROW
BEGIN
    if (NEW.age) < 18 then
    signal sqlstate '45000' set message_text = "Customer not added. Customer should be of age equal or above 18 years";
    elseif (NEW.age) > 110 then
    signal sqlstate '45000' set message_text = "Customer not added. Customer's age is beyond expected limit(>110 years)";
    elseif (NEW.contact)=(select contact from customer where contact = (NEW.contact)) then
    signal sqlstate '45000' set message_text = "You are already a customer of the hotel, you can directly make reservation.";
    end if;
END
```

Triggers in Reserves Table:

1. Before INSERT:

- Room booking validation. Same room cannot be booked by two customers.
- Validate Check-in and Check-out date. Check-In date can be current date or future date however Check-out date can be any future date after Check-In date but not same as Check-In date.
- Check if Room rate correctly entered.

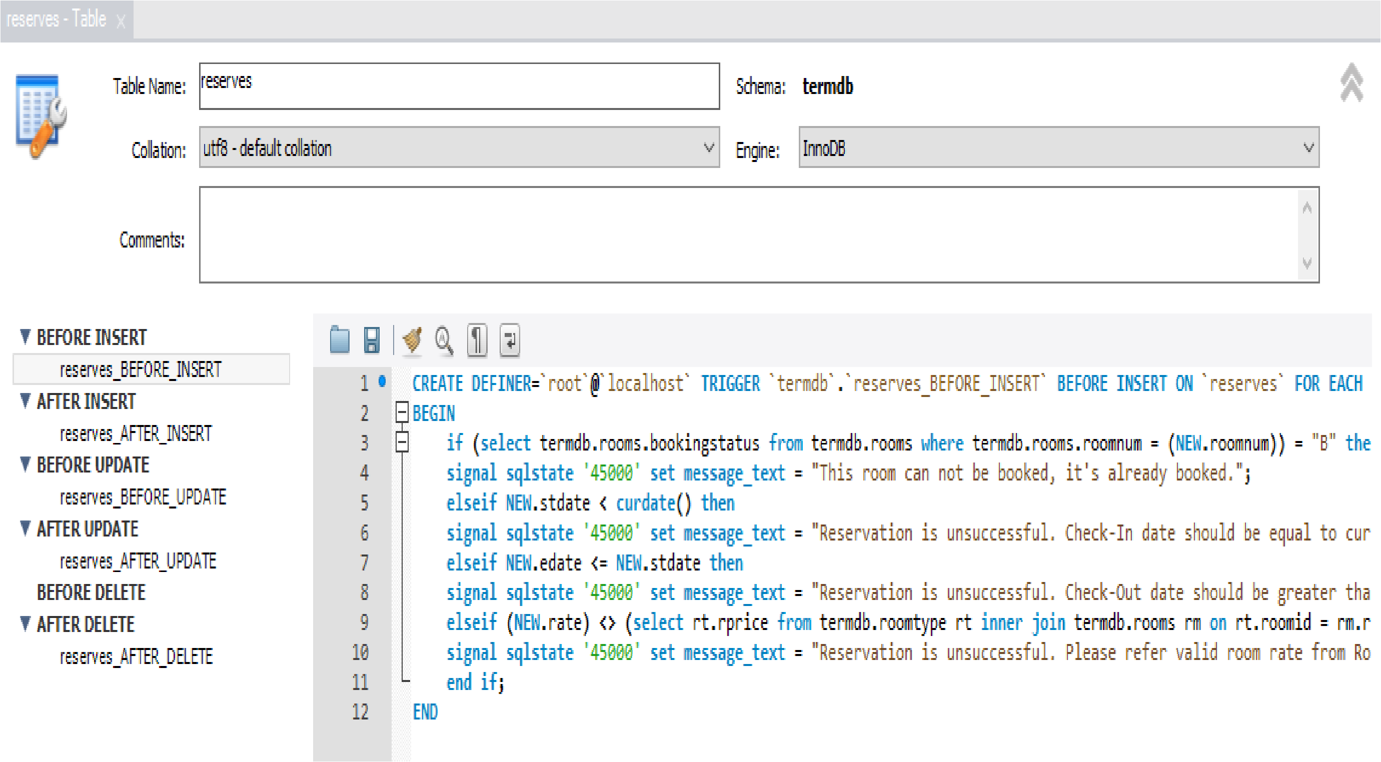


Table Name: reserves

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

- BEFORE INSERT
  - reserves_BEFORE_INSERT
- AFTER INSERT
  - reserves_AFTER_INSERT
- BEFORE UPDATE
  - reserves_BEFORE_UPDATE
- AFTER UPDATE
  - reserves_AFTER_UPDATE
- BEFORE DELETE
- AFTER DELETE
  - reserves_AFTER_DELETE

```
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`reserves_BEFORE_INSERT` BEFORE INSERT ON `reserves` FOR EACH
BEGIN
    if (select termdb.rooms.bookingstatus from termdb.rooms where termdb.rooms.roomnum = (NEW.roomnum)) = "B" the
    signal sqlstate '45000' set message_text = "This room can not be booked, it's already booked.";
    elseif NEW.stdate < curdate() then
    signal sqlstate '45000' set message_text = "Reservation is unsuccessful. Check-In date should be equal to cur
    elseif NEW.edate <= NEW.stdate then
    signal sqlstate '45000' set message_text = "Reservation is unsuccessful. Check-Out date should be greater tha
    elseif (NEW.rate) <> (select rt.rprice from termdb.roomtype rt inner join termdb.rooms rm on rt.roomid = rm.r
    signal sqlstate '45000' set message_text = "Reservation is unsuccessful. Please refer valid room rate from Ro
    end if;
END
```

2. After INSERT:

- After INSERT of reservation entry, update the "rooms" table and mark the booking status as booked ("B") against the reserved room.
- INSERT the reservation ID in Checkout table for maintain check out status for reservation.

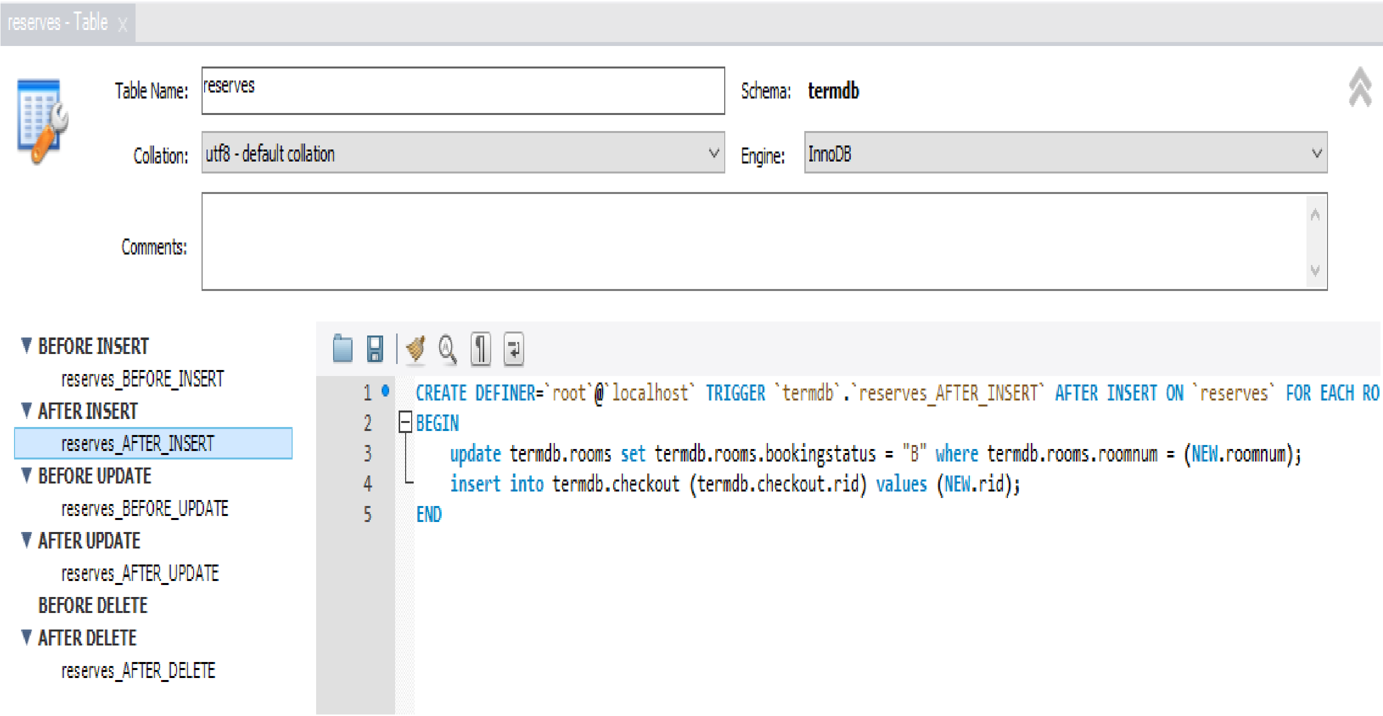


Table Name: reserves

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`reserves_AFTER_INSERT` AFTER INSERT ON `reserves` FOR EACH RO
BEGIN
    update termdb.rooms set termdb.rooms.bookingstatus = "B" where termdb.rooms.roomnum = (NEW.roomnum);
    insert into termdb.checkout (termdb.checkout.rid) values (NEW.rid);
END
```

3. Before UPDATE:

- Before modifying the room number, validate if the new room number being allotted is available or not.

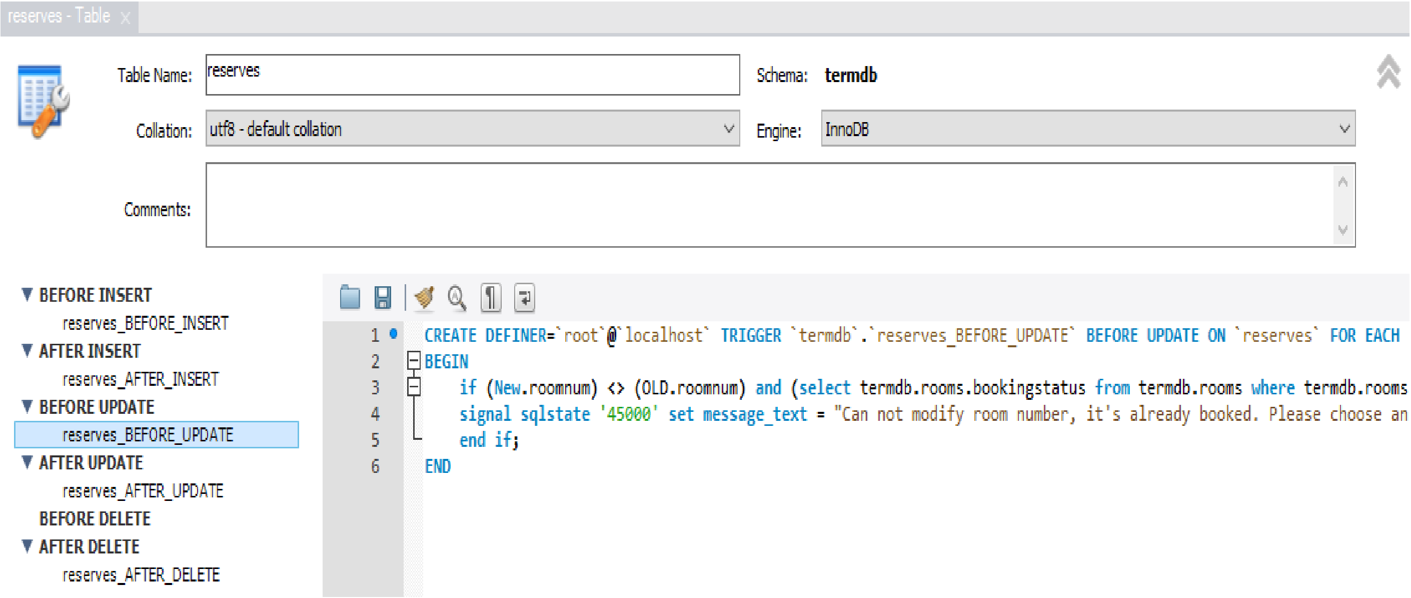


Table Name: reserves

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`reserves_BEFORE_UPDATE` BEFORE UPDATE ON `reserves` FOR EACH
BEGIN
    if (New.roomnum) <> (OLD.roomnum) and (select termdb.rooms.bookingstatus from termdb.rooms where termdb.rooms
    signal sqlstate '45000' set message_text = "Can not modify room number, it's already booked. Please choose an
    end if;
END
```

4. After UPDATE:

- If new room being assigned is available then assign that room to customer and update it as booked in rooms table and mark the previously booked room as not booked.

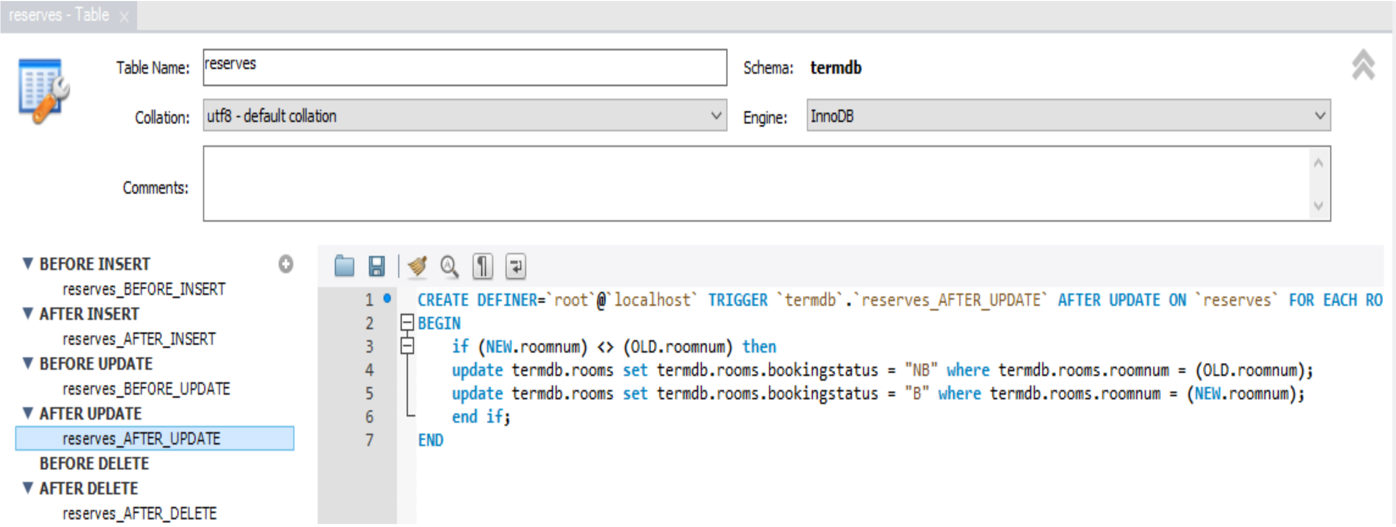


Table Name: reserves

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`reserves_AFTER_UPDATE` AFTER UPDATE ON `reserves` FOR EACH RO
BEGIN
    if (NEW.roomnum) <> (OLD.roomnum) then
    update termdb.rooms set termdb.rooms.bookingstatus = "NB" where termdb.rooms.roomnum = (OLD.roomnum);
    update termdb.rooms set termdb.rooms.bookingstatus = "B" where termdb.rooms.roomnum = (NEW.roomnum);
    end if;
END
```

5. After DELETE:

- If reservation cancelled then update room booking status as not booked in rooms table against that specific room and insert the cancelled entry into cancellation table for reference.

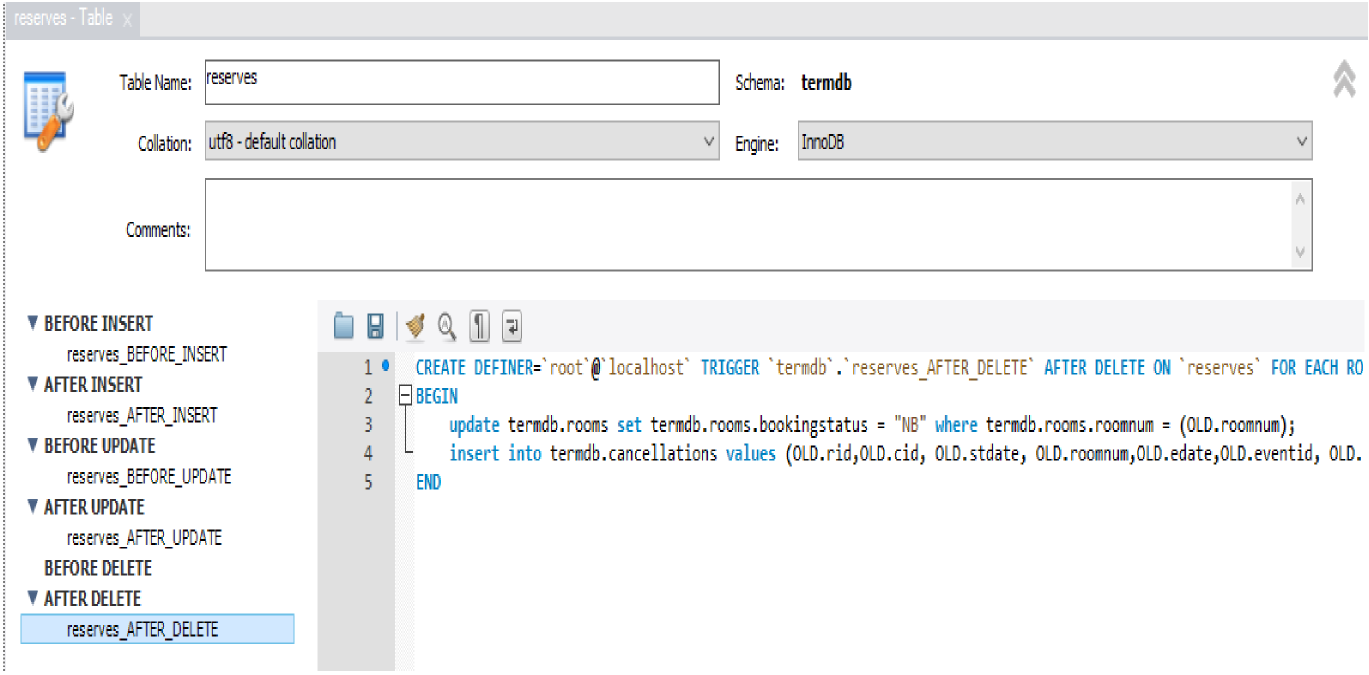


Table Name: reserves

Schema: termdb

Collation: utf8 - default collation

Engine: InnoDB

Comments:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`reserves_AFTER_DELETE` AFTER DELETE ON `reserves` FOR EACH RO
BEGIN
    update termdb.rooms set termdb.rooms.bookingstatus = "NB" where termdb.rooms.roomnum = (OLD.roomnum);
    insert into termdb.cancellations values (OLD.rid,OLD.cid, OLD.stdate, OLD.roomnum,OLD.edate,OLD.eventid, OLD.
END
```

Triggers in Checkout Table:

1. Before UPDATE:

- Before giving checkout to customer, validate if bill is fully paid or not. If not then customer cannot checkout without clearing dues.

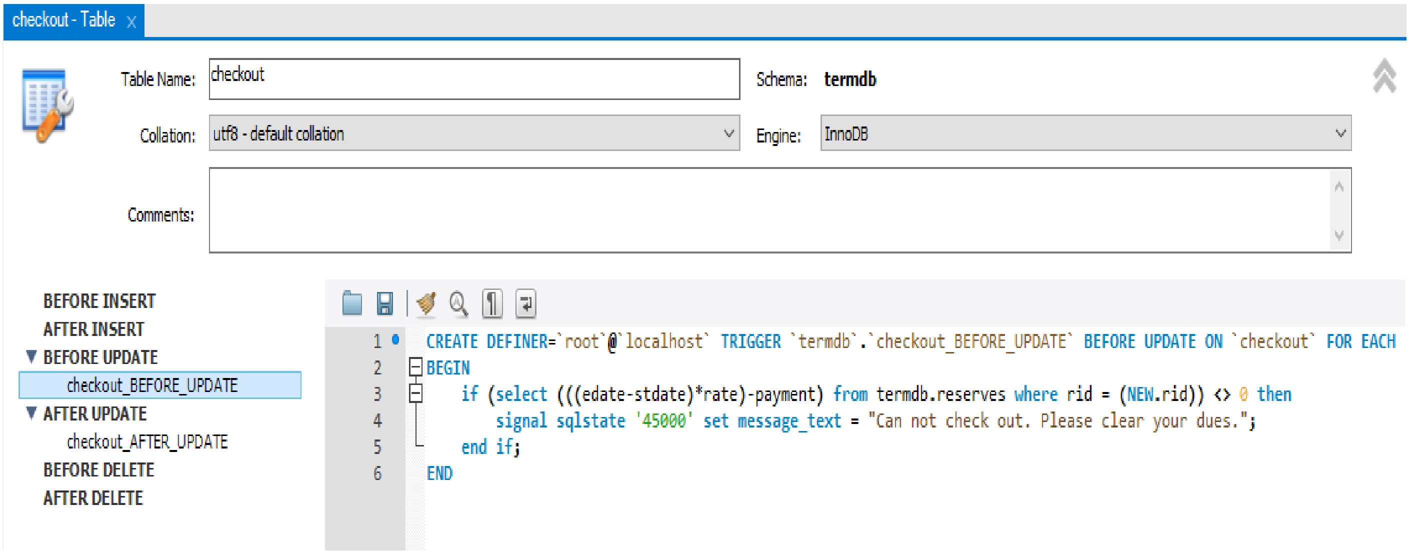


Table Name: checkout
Schema: termdb
Collation: utf8 - default collation
Engine: InnoDB
Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`checkout_BEFORE_UPDATE` BEFORE UPDATE ON `checkout` FOR EACH
BEGIN
    if (select (((edate-stdate)*rate)-payment) from termdb.reserves where rid = (NEW.rid)) <> 0 then
        signal sqlstate '45000' set message_text = "Can not check out. Please clear your dues.";
    end if;
END
```

2. After UPDATE:

- If checkout is done successfully then update that specific room number as not booked or available in rooms table.

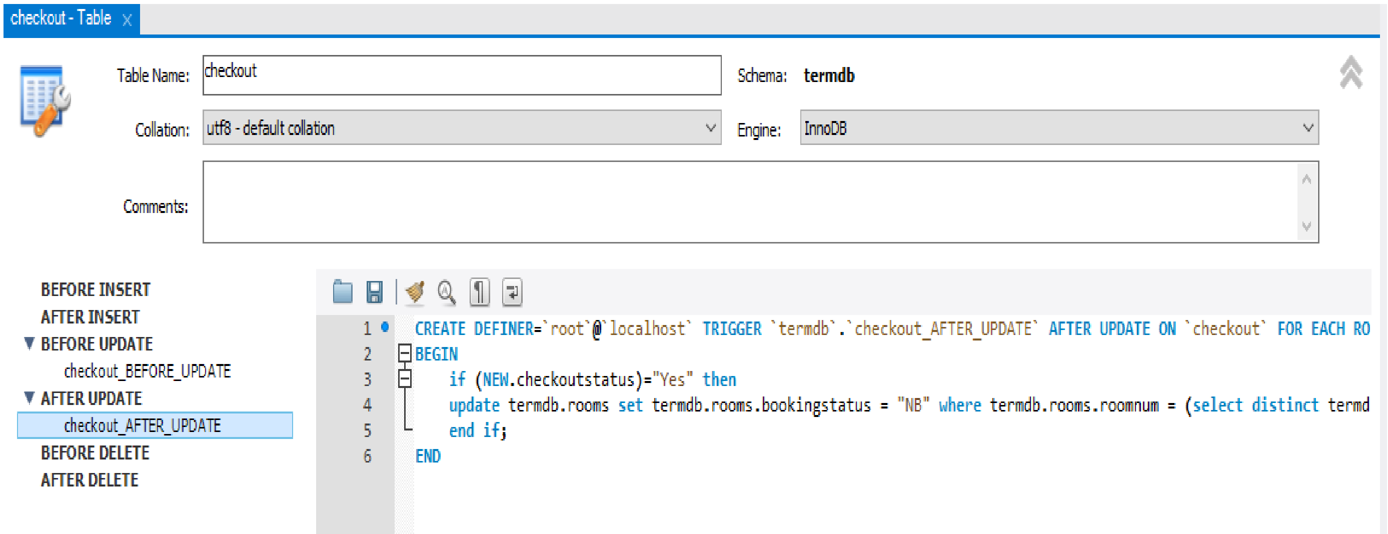


Table Name: checkout
Schema: termdb
Collation: utf8 - default collation
Engine: InnoDB
Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `termdb`.`checkout_AFTER_UPDATE` AFTER UPDATE ON `checkout` FOR EACH RO
BEGIN
    if (NEW.checkoutstatus)="Yes" then
    update termdb.rooms set termdb.rooms.bookingstatus = "NB" where termdb.rooms.roomnum = (select distinct termd
    end if;
END
```

Queries

1. Find the customer (or customers) who paid the highest room rate in 2017 and is also related to at least one more non-primary customer.

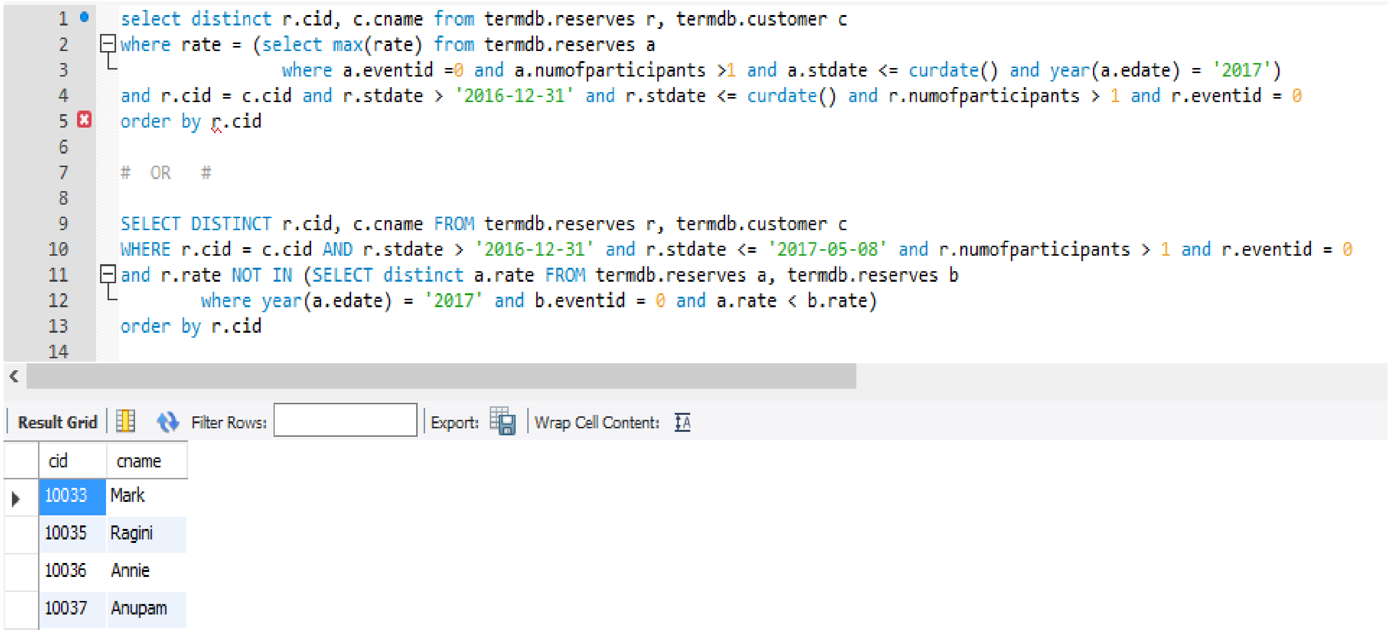
Query:

select distinct r.cid, c.cname from termdb.reserves r, termdb.customer c
where rate = (select max(rate) from termdb.reserves a
where a.eventid =0 and a.numofparticipants >1 and a.stdate <= curdate() and year(a.edate) = '2017')
and r.cid = c.cid and r.stdate > '2016-12-31' and r.stdate <= curdate() and r.numofparticipants > 1 and r.eventid = 0
order by r.cid

Alternate Query:

SELECT DISTINCT r.cid, c.cname FROM termdb.reserves r, termdb.customer c
WHERE r.cid = c.cid AND r.stdate > '2016-12-31' and r.stdate <= '2017-05-08' and r.numofparticipants > 1 and r.eventid = 0
and r.rate NOT IN (SELECT distinct a.rate FROM termdb.reserves a, termdb.reserves b
where year(a.edate) = '2017' and b.eventid = 0 and a.rate < b.rate)
order by r.cid

Output:



```
select distinct r.cid, c.cname from termdb.reserves r, termdb.customer c
where rate = (select max(rate) from termdb.reserves a
                    where a.eventid =0 and a.numofparticipants >1 and a.stdate <= curdate() and year(a.edate) = '2017')
and r.cid = c.cid and r.stdate > '2016-12-31' and r.stdate <= curdate() and r.numofparticipants > 1 and r.eventid = 0
order by r.cid

#  OR  #

SELECT DISTINCT r.cid, c.cname FROM termdb.reserves r, termdb.customer c
WHERE r.cid = c.cid AND r.stdate > '2016-12-31' and r.stdate <= '2017-05-08' and r.numofparticipants > 1 and r.eventid = 0
and r.rate NOT IN (SELECT distinct a.rate FROM termdb.reserves a, termdb.reserves b
          where year(a.edate) = '2017' and b.eventid = 0 and a.rate < b.rate)
order by r.cid
```

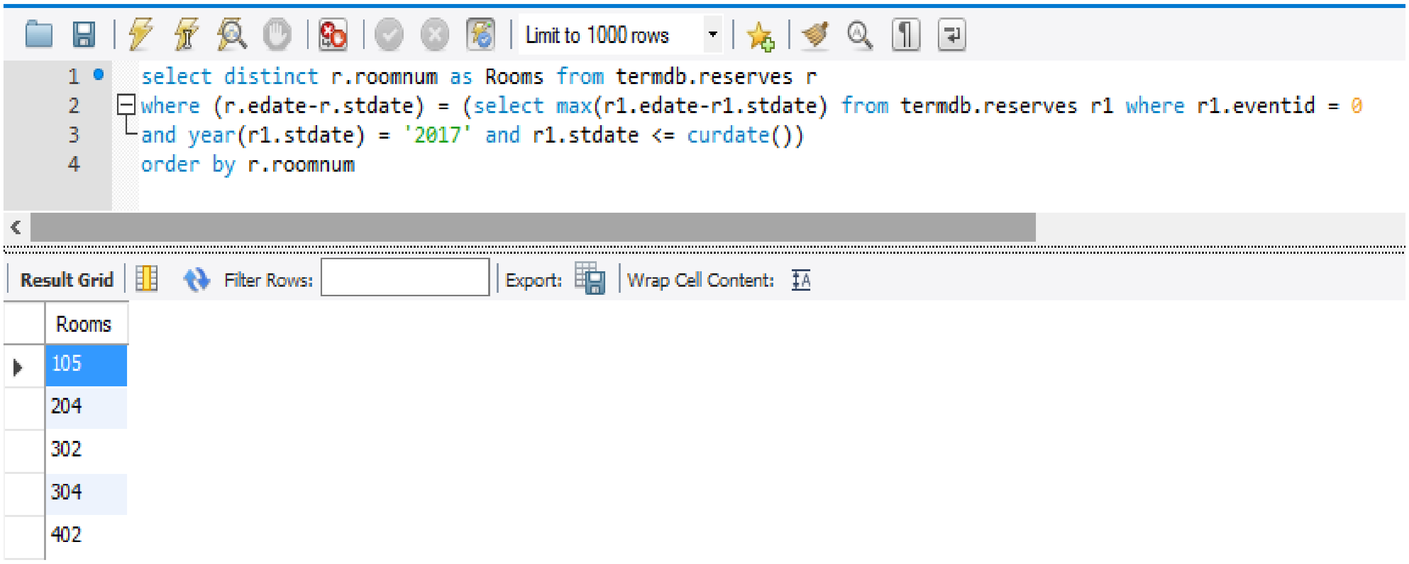
cid	cname
10033	Mark
10035	Ragini
10036	Annie
10037	Anupam

2.Find the room (or rooms) that was booked for the most consecutive number of days in a single booking by a customer in 2017.

Query:

```
select distinct r.roomnum as Rooms from termdb.reserves r
where (r.edate-r.stdate) = (select max(r1.edate-r1.stdate) from termdb.reserves r1 where r1.eventid =
0
and year(r1.stdate) = '2017' and r1.stdate <= curdate())
order by r.roomnum
```

Output:



```
select distinct r.roomnum as Rooms from termdb.reserves r
where (r.edate-r.stdate) = (select max(r1.edate-r1.stdate) from termdb.reserves r1 where r1.eventid = 0
and year(r1.stdate) = '2017' and r1.stdate <= curdate())
order by r.roomnum
```

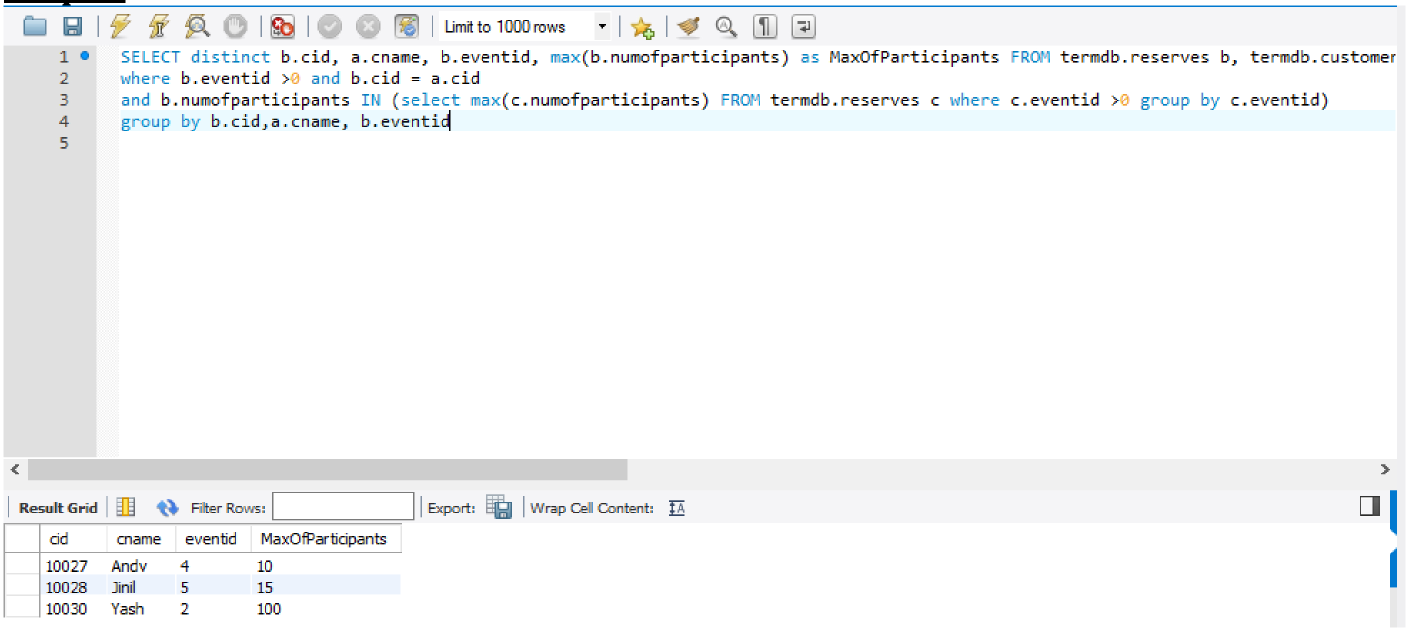
```
Rooms
105
204
302
304
402
```

3. For every event type find the name of the group that had the biggest number of Participants

Query:

SELECT distinct b.cid, a.cname, b.eventid, max(b.numofparticipants) as MaxOfParticipants
FROM termdb.reserves b, termdb.customer a
where b.eventid >0 and b.cid = a.cid
and b.numofparticipants IN (select max(c.numofparticipants) FROM termdb.reserves c where c.eventid >0 group by c.eventid)
group by b.cid,a.cname, b.eventid

Output:

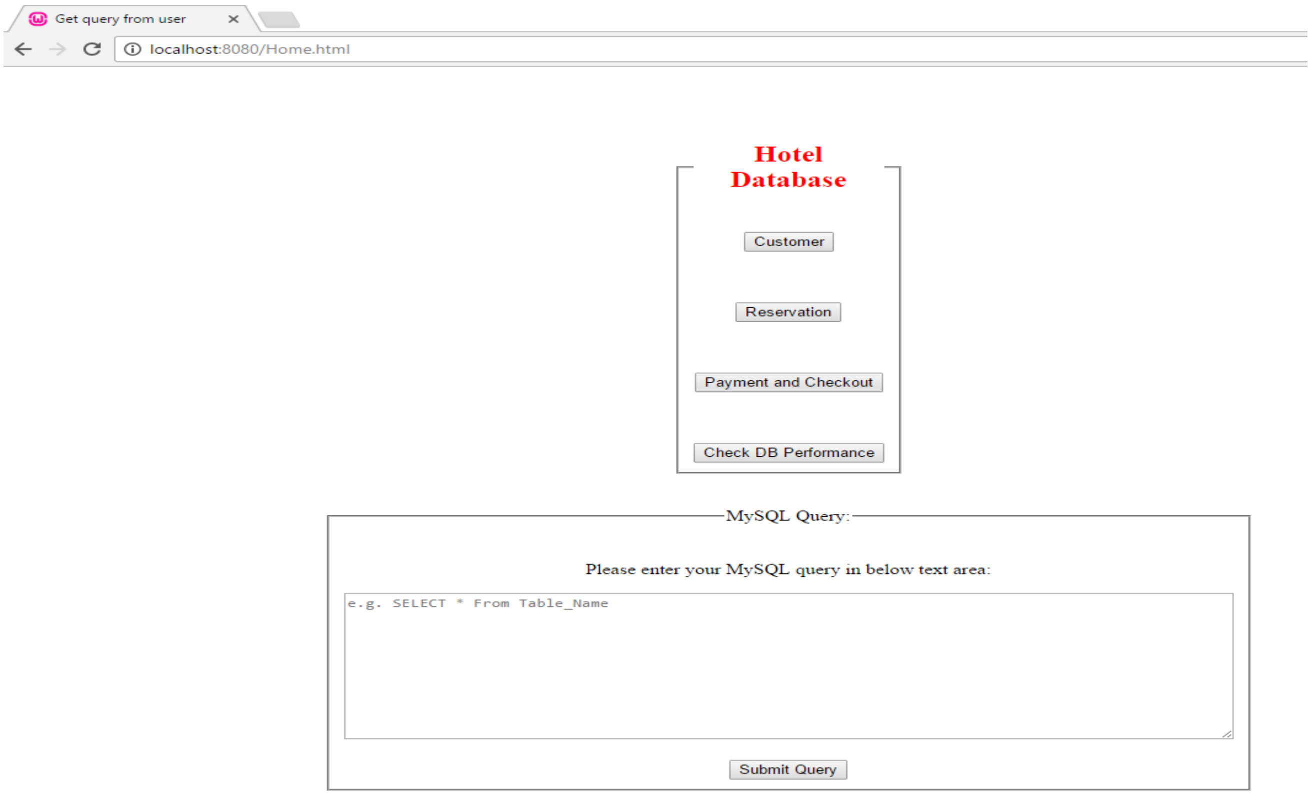


```
SELECT distinct b.cid, a.cname, b.eventid, max(b.numofparticipants) as MaxOfParticipants FROM termdb.reserves b, termdb.customer
where b.eventid >0 and b.cid = a.cid
and b.numofparticipants IN (select max(c.numofparticipants) FROM termdb.reserves c where c.eventid >0 group by c.eventid)
group by b.cid,a.cname, b.eventid
```

cid	cname	eventid	MaxOfParticipants
10027	Andv	4	10
10028	Jinil	5	15
10030	Yash	2	100

3. Part 3 - HTML Forms and SQL Output

➢ Home Page



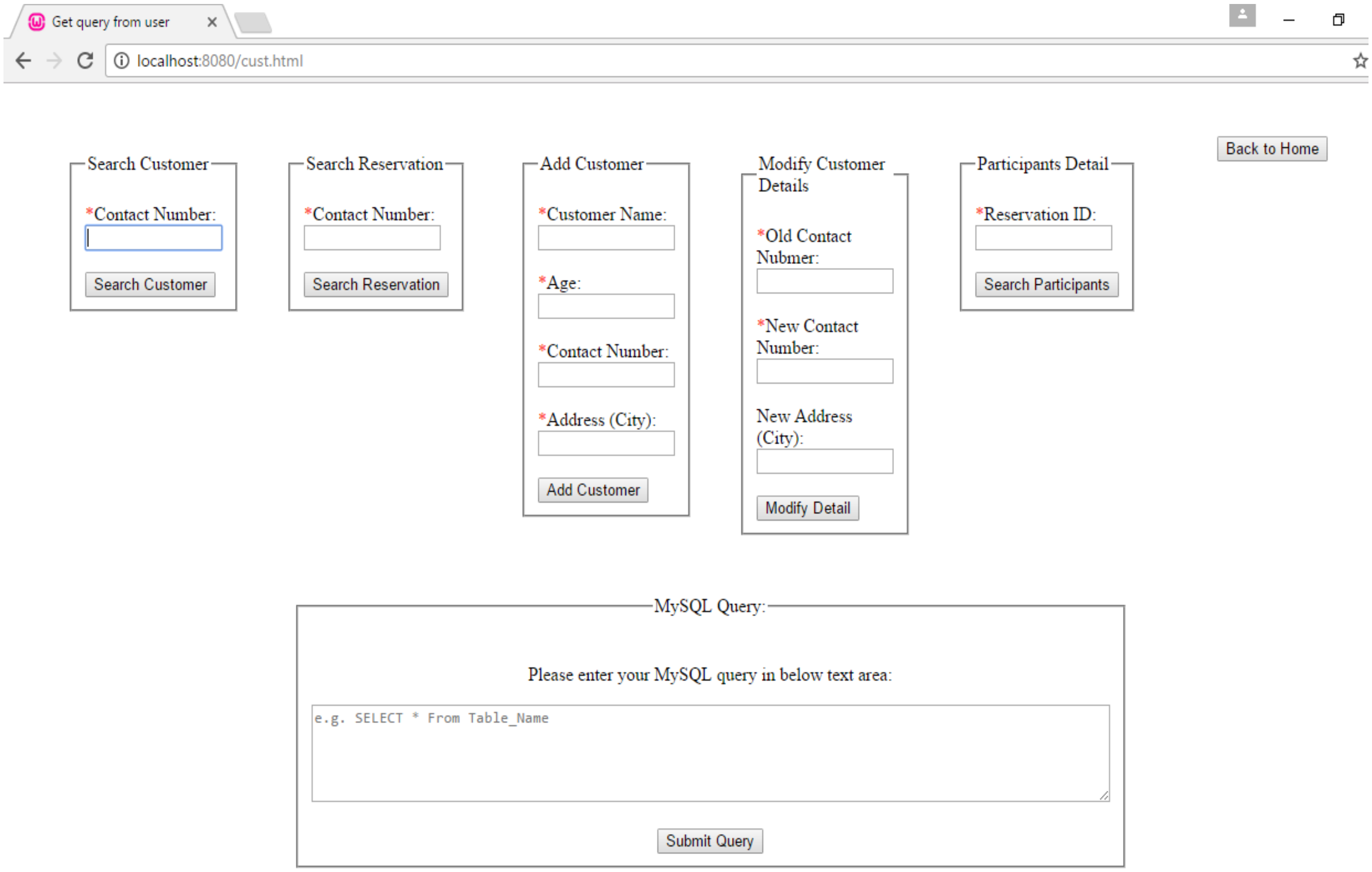
Hotel Database

MySQL Query:

Please enter your MySQL query in below text area:

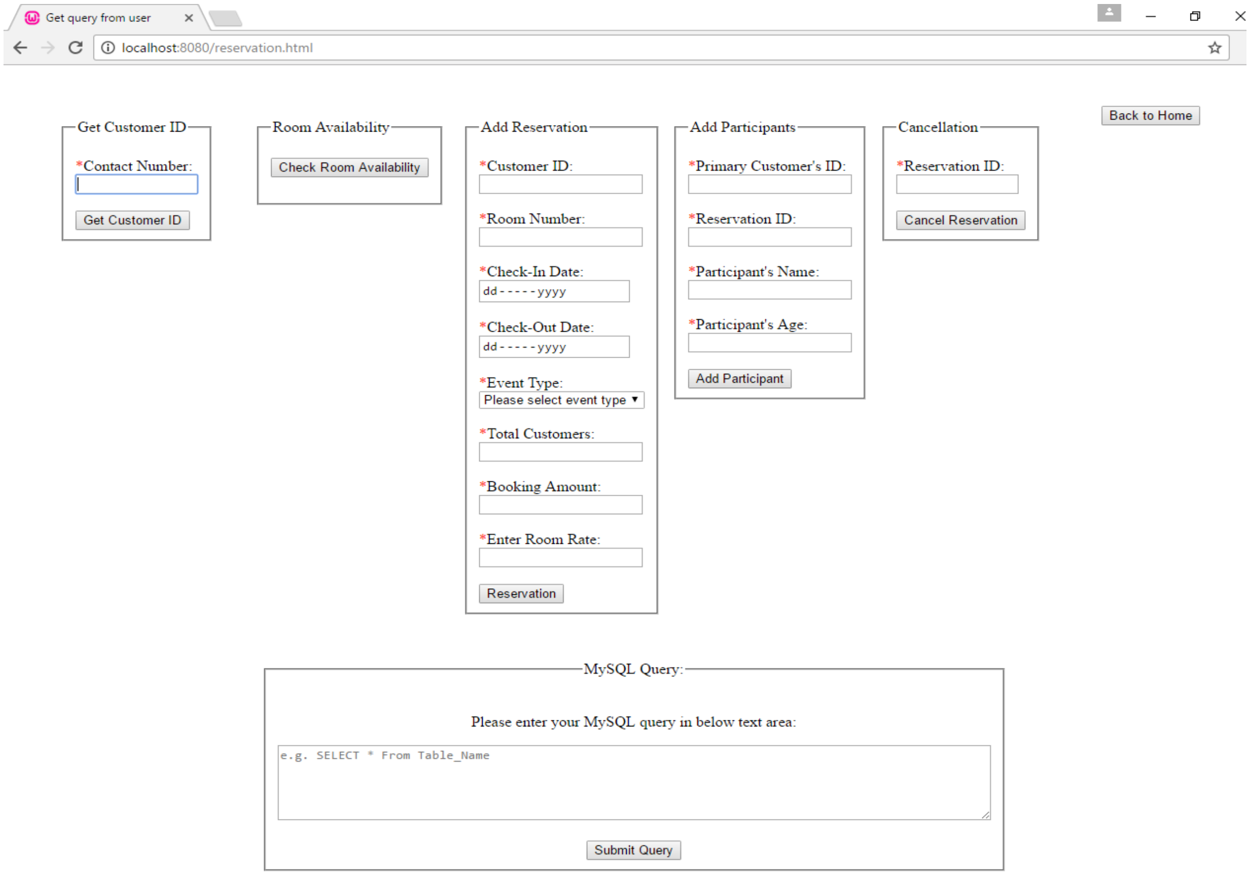
e.g. SELECT * From Table_Name

➢ Customer Page



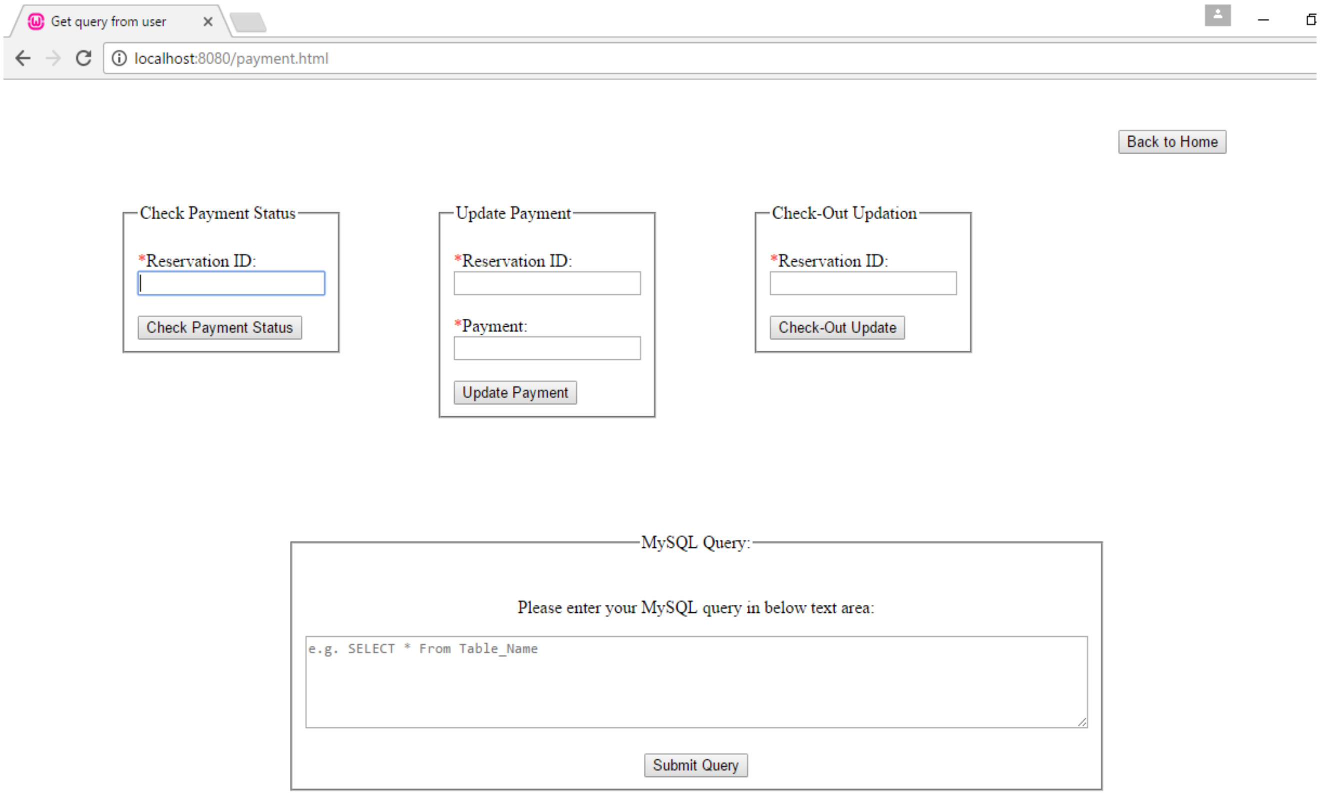
Please enter your MySQL query in below text area:

➢ Reservation Page



Please enter your MySQL query in below text area:

- ## Payment & check-out page

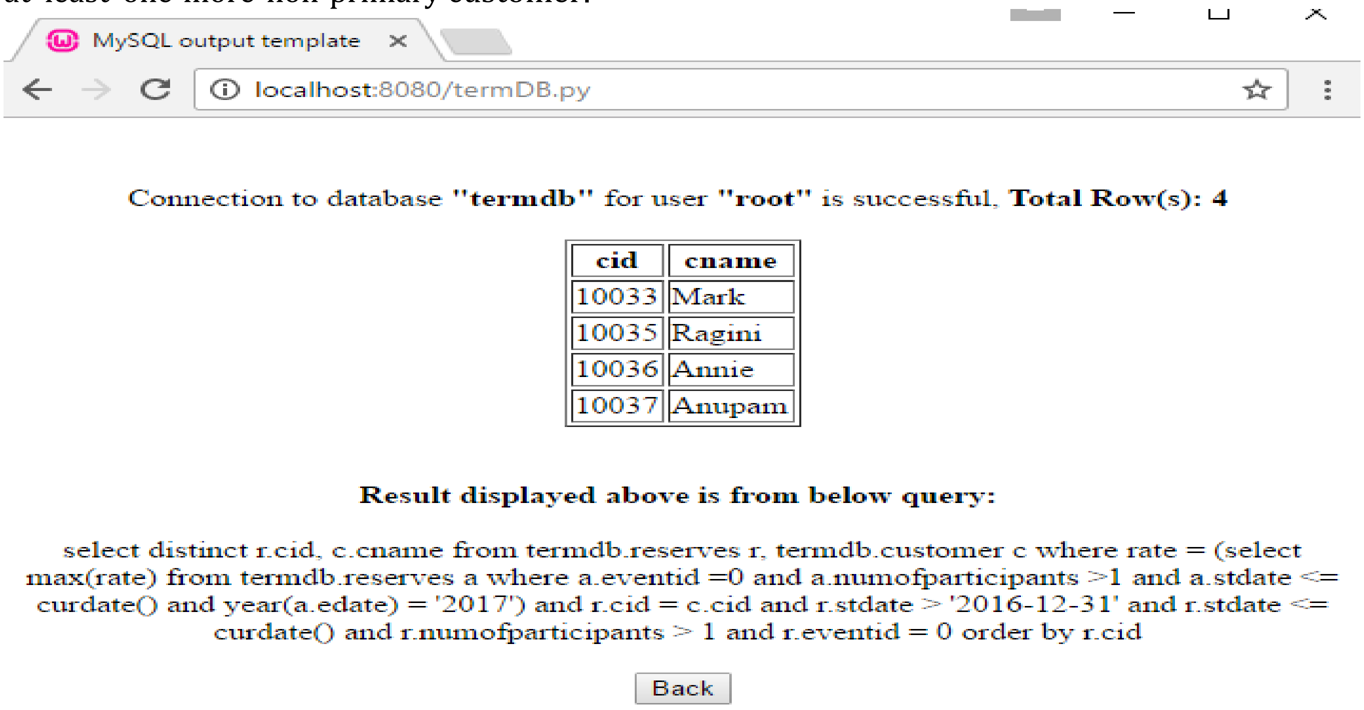


Please enter your MySQL query in below text area:

```
e.g. SELECT * From Table_Name
```

SQL Queries Output when running From HTML Form

1. Find the customer (or customers) who paid the highest room rate in 2017 and is also related to at least one more non-primary customer.



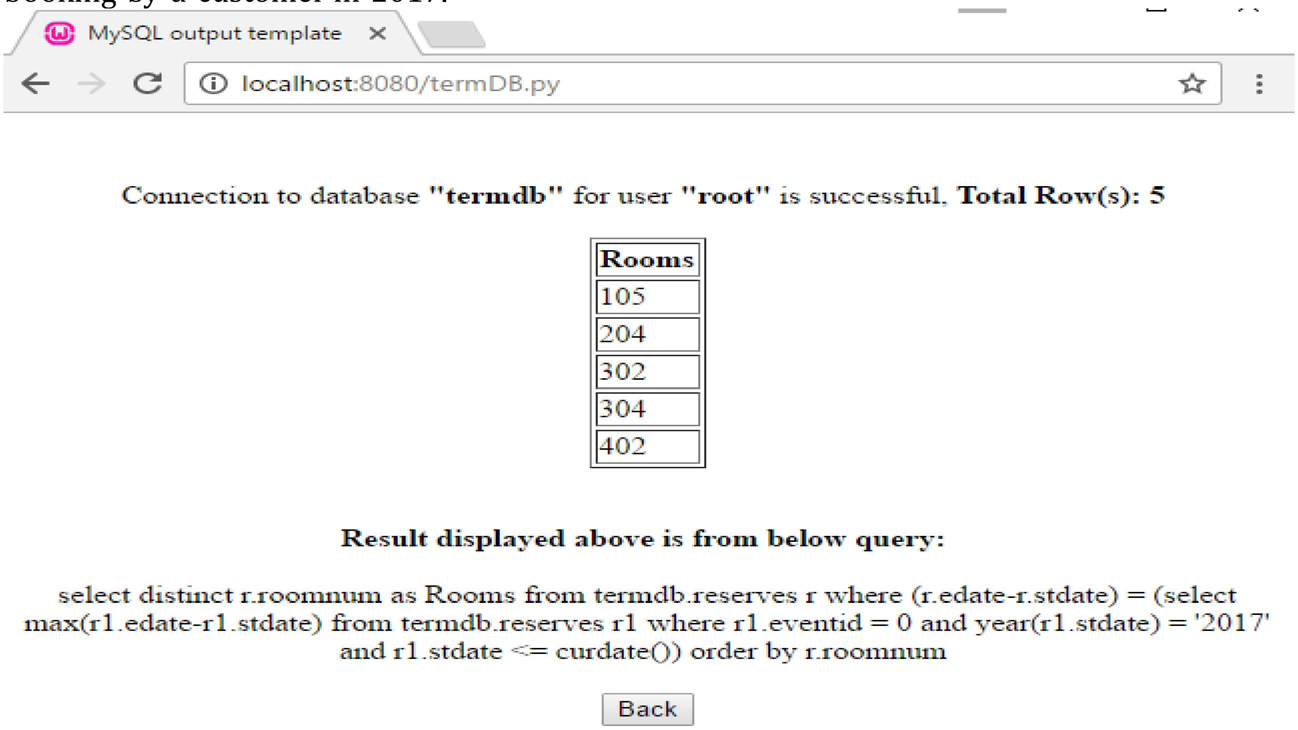
Connection to database **"termdb"** for user **"root"** is successful, **Total Row(s): 4**

cid	cname
10033	Mark
10035	Ragini
10036	Annie
10037	Anupam

Result displayed above is from below query:

select distinct r.cid, c.cname from termdb.reserves r, termdb.customer c where rate = (select max(rate) from termdb.reserves a where a.eventid =0 and a.numofparticipants >1 and a.stdate <= curdate() and year(a.edate) = '2017') and r.cid = c.cid and r.stdate > '2016-12-31' and r.stdate <= curdate() and r.numofparticipants > 1 and r.eventid = 0 order by r.cid

2.Find the room (or rooms) that was booked for the most consecutive number of days in a single booking by a customer in 2017.



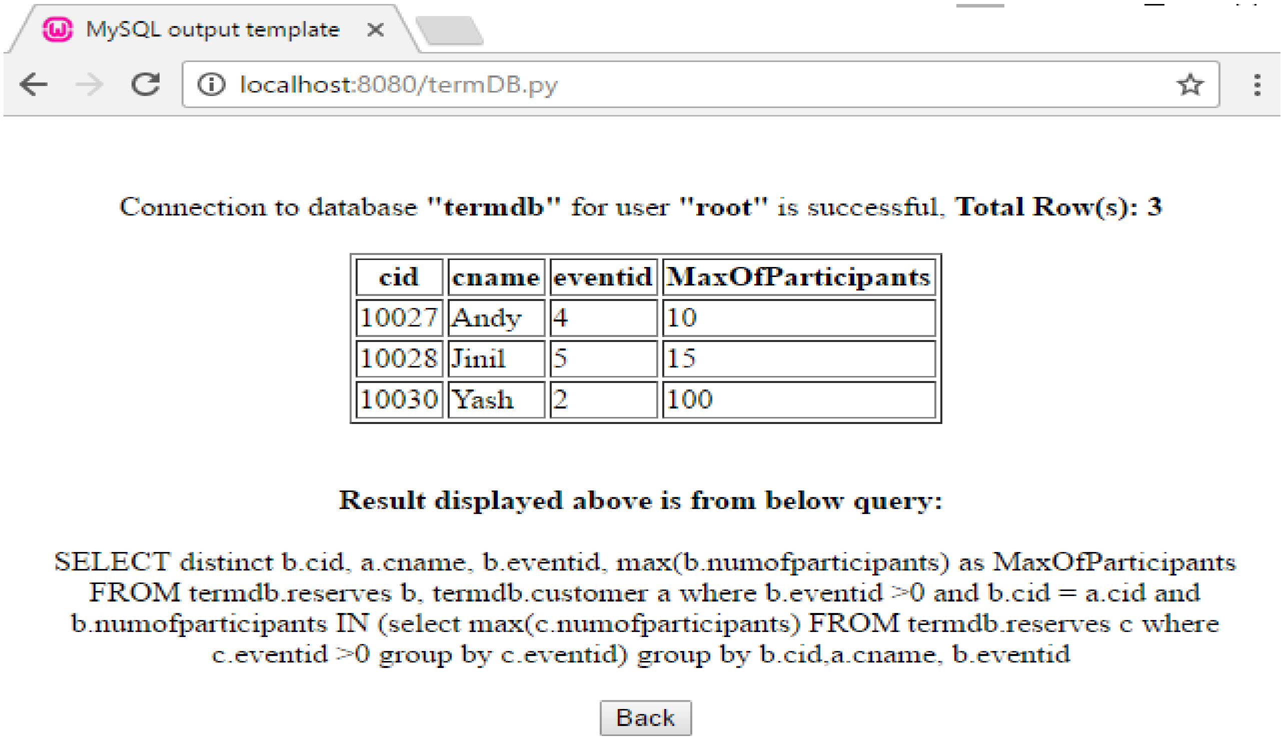
Connection to database **"termdb"** for user **"root"** is successful, **Total Row(s): 5**

Rooms
105
204
302
304
402

Result displayed above is from below query:

select distinct r.roomnum as Rooms from termdb.reserves r where (r.edate-r.stdate) = (select max(r1.edate-r1.stdate) from termdb.reserves r1 where r1.eventid = 0 and year(r1.stdate) = '2017' and r1.stdate <= curdate()) order by r.roomnum

3. For every event type find the name of the group that had the biggest number of Participants



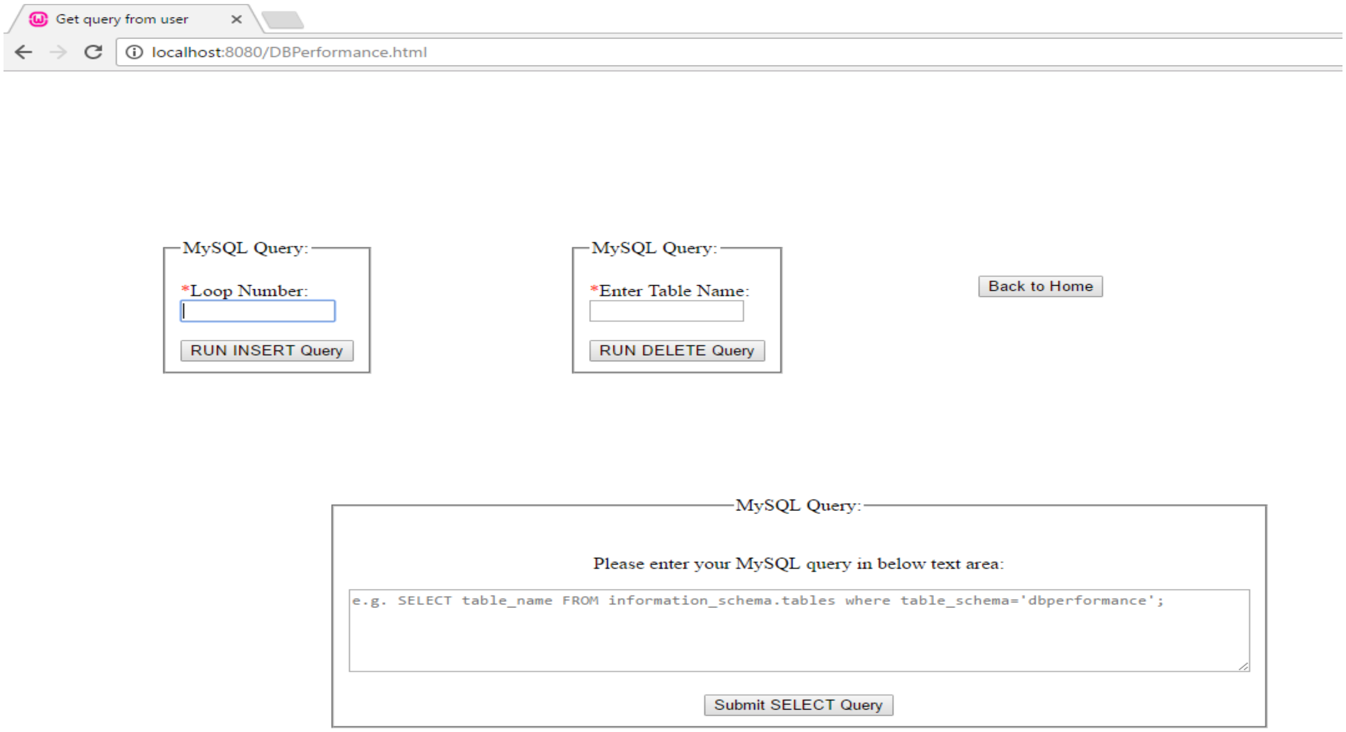
Connection to database **"termdb"** for user **"root"** is successful, **Total Row(s): 3**

cid	cname	eventid	MaxOfParticipants
10027	Andy	4	10
10028	Jinil	5	15
10030	Yash	2	100

Result displayed above is from below query:

```
SELECT distinct b.cid, a.cname, b.eventid, max(b.numofparticipants) as MaxOfParticipants
FROM termdb.reserves b, termdb.customer a where b.eventid >0 and b.cid = a.cid and
b.numofparticipants IN (select max(c.numofparticipants) FROM termdb.reserves c where
c.eventid >0 group by c.eventid) group by b.cid,a.cname, b.eventid
```

4. Part 4 – DBA (Performance Check)



MySQL Query:

*Loop Number:

RUN INSERT Query

MySQL Query:

*Enter Table Name:

RUN DELETE Query

Back to Home

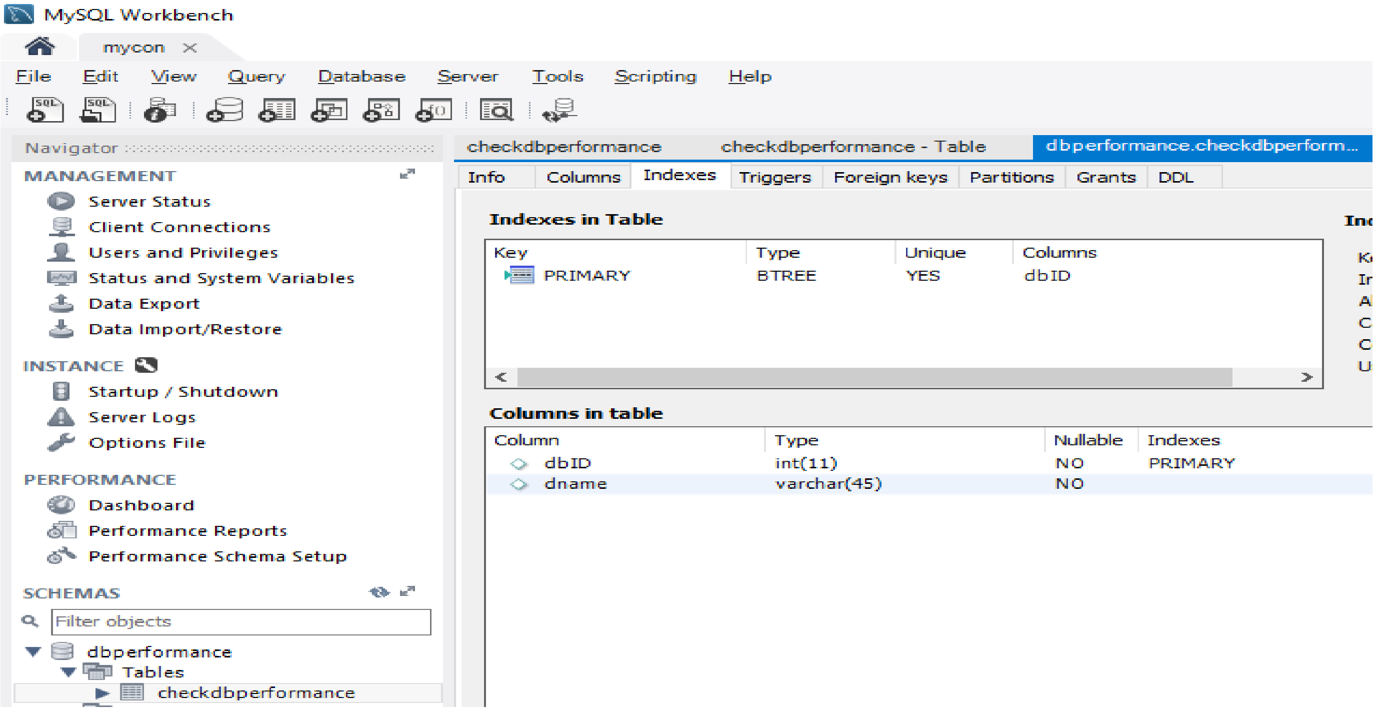
MySQL Query:

Please enter your MySQL query in below text area:

```
e.g. SELECT table_name FROM information_schema.tables where table_schema='dbperformance';
```

Submit SELECT Query

Without Index on Dname

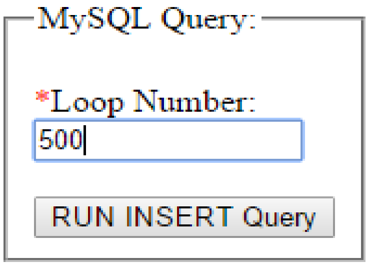


Indexes in Table

Key	Type	Unique	Columns
PRIMARY	BTREE	YES	dbID

Columns in table

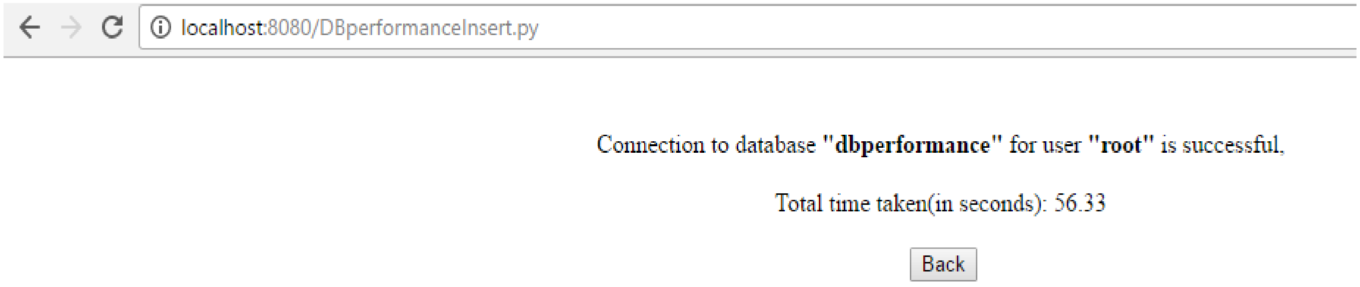
Column	Type	Nullable	Indexes
dbID	int(11)	NO	PRIMARY
dname	varchar(45)	NO	



MySQL Query:

*Loop Number: 500

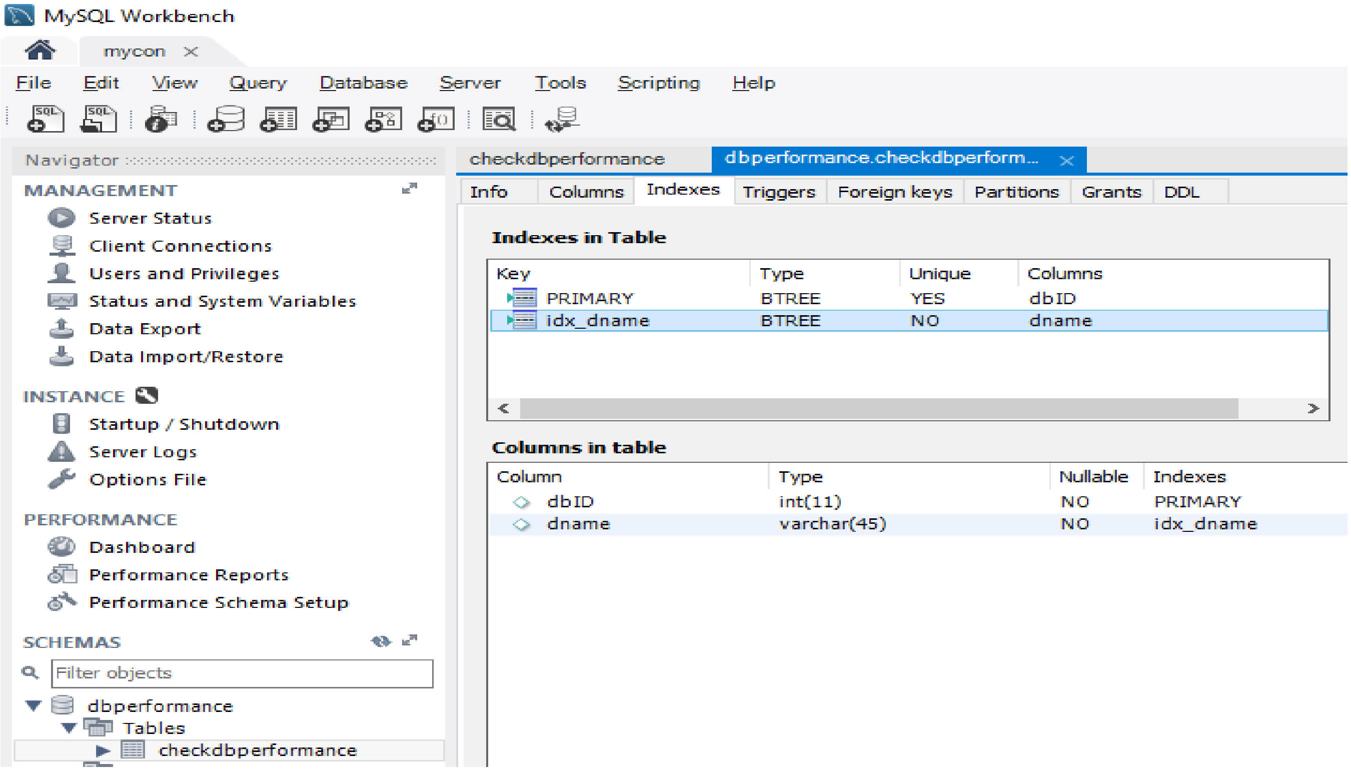
Time taken to insert 500 records before indexing: 56.33 Seconds



Connection to database "**dbperformance**" for user "**root**" is successful,

Total time taken(in seconds): 56.33

With Index on dname column

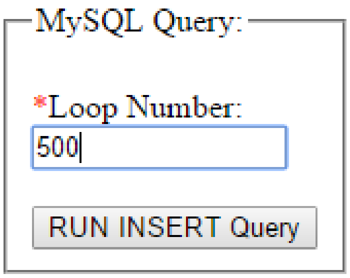


Indexes in Table

Key	Type	Unique	Columns
PRIMARY	BTREE	YES	dbID
idx_dname	BTREE	NO	dname

Columns in table

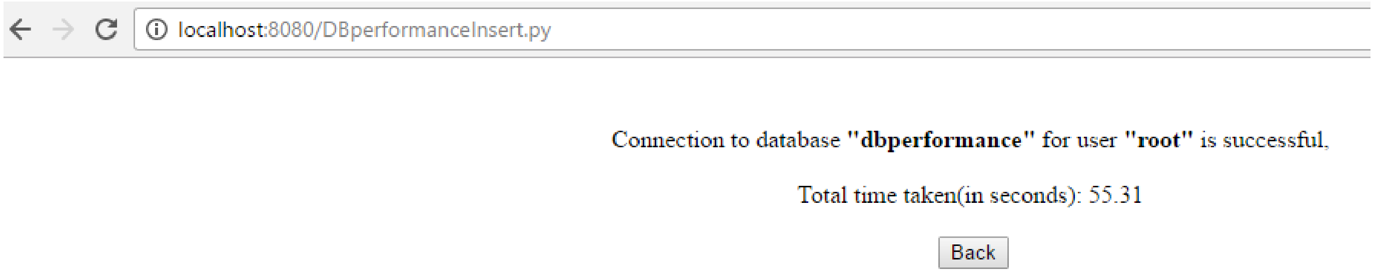
Column	Type	Nullable	Indexes
dbID	int(11)	NO	PRIMARY
dname	varchar(45)	NO	idx_dname



MySQL Query:

*Loop Number:
500

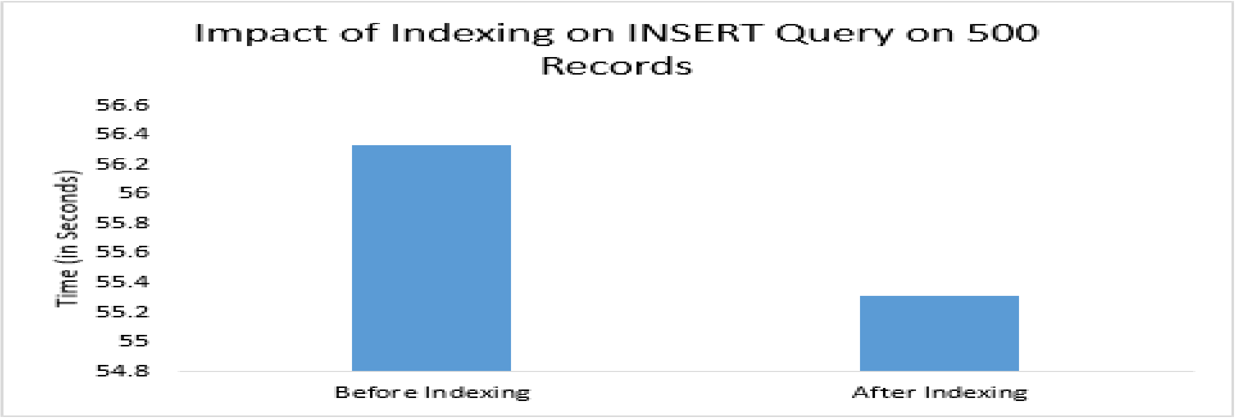
Time taken to insert 500 records after indexing: 55.31 Seconds



localhost:8080/DBperformanceInsert.py

Connection to database "**dbperformance**" for user "**root**" is successful.

Total time taken(in seconds): 55.31



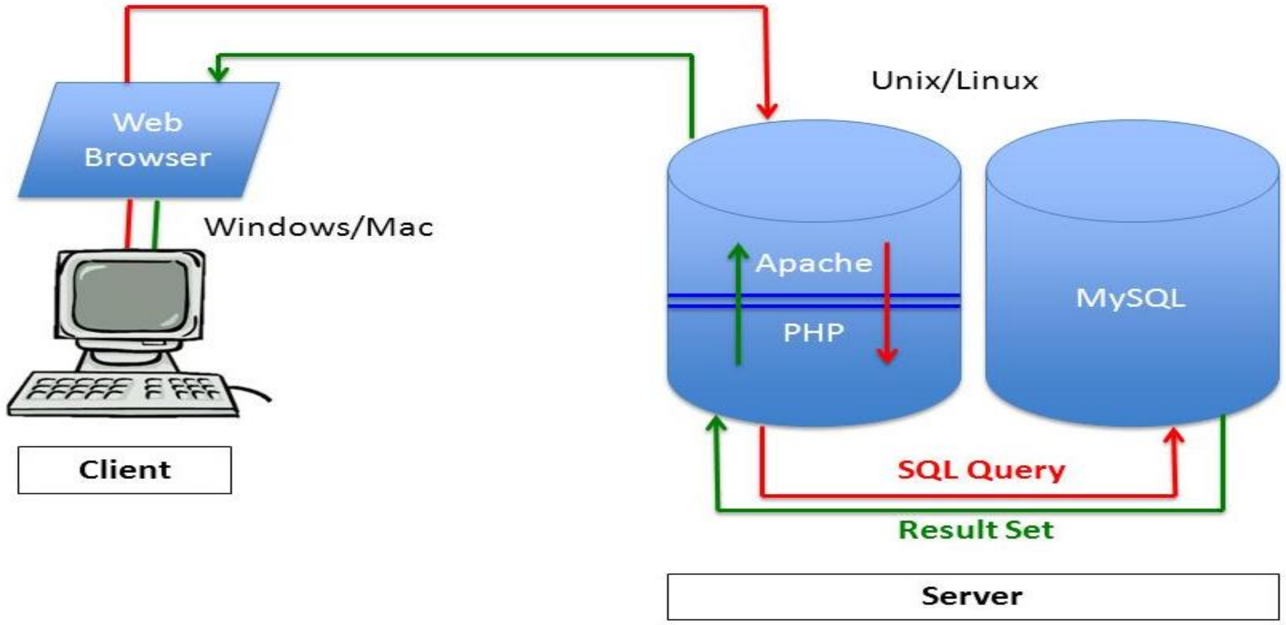
➢ Findings

- From the above analysis it is clear that before applying indexing on **dname** <-> **did** , query took a large amount of time generate the output.
- Here B+tree base indexing was used where **did** was used as a root node to reference to **dname**.
- Query insertion and updation becomes effective due to indexing method

5. Part 5 - RDBMS and Web Services

Integrating RDBMS and Web Services

RDBMS is a complex Database management system which makes use of relational models for implementing the database. RDBMS from the back-end of a three -tier Architecture. Now a day use of RDBMS is increasing rapidly due to boost in economic development. A complex data structure storing makes it favorable for high back-end use. Usually, integration of RDBMS can be done with a large variety of interface.

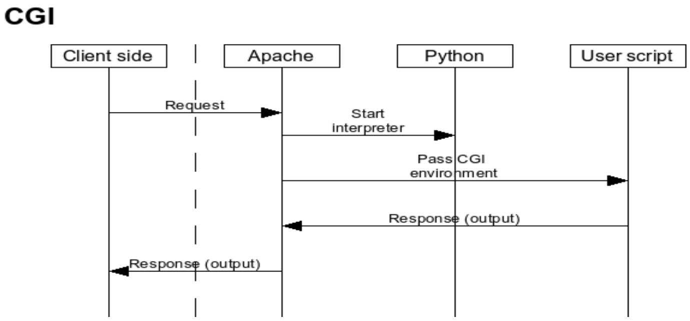


Client Side <-> Server <-> Database

- Issues that must be Address while integrating web services with RDBMS are: -
    - The method in which the Web-to-Database stores and extracts complex data objects such as documents, graphics, and streaming video through a Web server.
    - The storage and performance overhead of binary objects in a DBMS.
    - The ability of a DBMS to handle binary object transaction.
    - The capacity and limitations of extended or OLE data types.
    - The limitations of client browsers (a.k.a. whether the browser needs a plug-in).
    - The capability of a DBMS to support enterprise caliber databases.
    - The frequency of end user data access and update.
- In addition to that it must also Provide Security, Atomic Database Transaction and User Verification and Control.

Following are the commonly used Interface for Integration

- CGI (common gateway Interface)
  It’s standard protocol for web servers to execute programs that run like Console applications running on a server that generates web pages dynamically. Such programs are known as *CGI scripts* . The specifics of how the server executes the script are determined by the server. In the typical case, a CGI script runs at the time the user makes a request. Interface CGI is Simple that is why we can use a variety of different languages for CGI scripting. As the advancement are going on RDBMS, CGI faces some shortcomings like unable to interact with user & web browser directly, the time complexity for generating dynamic pages on request of the user. Primary languages like Java fail to connect with CGI at run time even though it allows generating scripts in Perl, Shell, Python. CGI is dealing with Static issues while interfacing presentation tier with Application Tier.

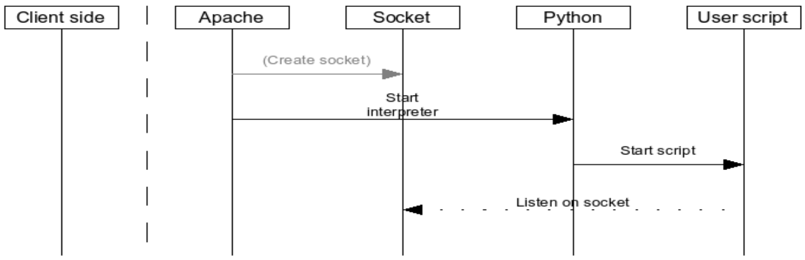


- **FastCGI** approach deals with reducing overhead associated with Interfacing web server and RDBMS. It Provides handling of multiple webpage requests at once.

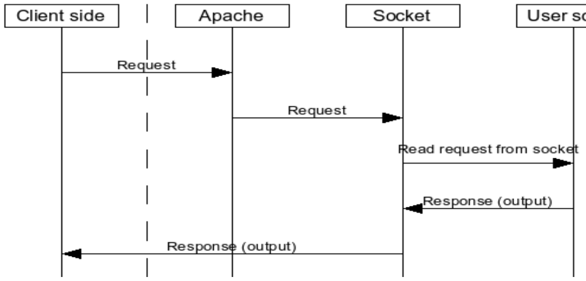
Technology Overview	**CGI**	**FastCGI**
Memory usage	Low	High
CPU Usage	High	Low
Security	Low	High
Run as file owner	No	Yes
Overall Performance	Slow	Fast

- The Above Table Provide a clear picture why use of Fast CGI over CGI is increasing while integrating with RDBMS.

FastCGI (Startup)



FastCGI (Request handling)



- The above technology was use for integrating application program with RDBMS, but now most of the Programming languages have their Interface while communicating with Apache. Because In the earlier days of the web, server-side scripting was exclusively performed by using a combination of C programs, Perl scripts, and shell scripts using the Common Gateway Interface (CGI). **Those scripts were executed by the operating system, and the results were served back by the web server.** Many modern web servers can directly perform **on-line scripting languages** such as **ASP, JSP, Perl, PHP, and Ruby** either by the web server itself or via extension modules (e.g. mod_perl or mod_php) to the web server. For example, WebDNA includes its embedded database system. This direct execution scripting can be used to build up complex multi-page sites, reducing overhead because of the lower number of calls to external interpreters thereby make it favorable for integrating with large RDBMS - (wikipedia)

- Java Servlets is a Program that extends the capability of a server. It is An object that receives a request and generates a response based on this application. Java servlets are used in the similar way CGI is used, it interprets user input from web pages and passes the request made by the user to the Java Program which performs Business Logic using SQL to access the Database. Java servlets are interpreter which uses JSP to Generate Dynamic Content as per request. Java servlets are used because they are easy to comprehend with Java Program & provide wide variety of Functionality of Java class.
- Major advantage: - **Multi-thread Request Computing by Java servlets over single-thread computing by CGI.**

- **Apache Tomcat**, often referred to as **Tomcat Server**, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run. Tomcat server is typically build to provide Java base environment compared to Apache

Http server as Java application logic has significant benefits like time complexity, space complexity, runtime execution, multithreading because of which developer mainly prefers it. Due to which tomcat server established to effectively Integrate Java Environment with RDBMS.

- **PHP** is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language. PHP code may be embedded into HTML or HTML5 markup, or it can be used in combination with various web template systems, web content management systems and web frameworks. The PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable. The web server software combined the results of the interpreted and executed PHP code, which may be any data, including images, with the generated web page. PHP code may also be executed with a command-line interface (CLI) and can be used to implement standalone graphical applications.
- **Mode_Php** is an interfacing technology makes PHP a part of Apache by having the Apache server interpret the PHP code.
- **Su_Php** runs PHP outside of the Apache script as CGI. Unlike CGI, however, it will run the scripts as a user other than the Apache user

Characteristic	**Mode_PHP**	**Su_PHP**
Memory Usage	low	low
CPU Usage	low	High
Security	low	High
Overall Performance	fast	slow

- PL Intermediary between RDBMS(Mysql) and Web Interface

  - R-studio
    R has certain limitation regarding the processing of enormous dataset and doesn't support concurrent access to data. A Relational Database Management System on the other hand can provide fast access to selected portions of the big data, and can provide simultaneous access from multiple users running on various hosts. Various R packages can be used to communicate with RDBMS, each with a different level of abstraction. Some of these packages have the functionality of copying entire data frames to and from databases. For example, RODBC, RMySQL, RJDBC

  - Perl via CGI
    Easily interpreted language, providing variety of Functionality used along with CGI for Interacting with RDBMS, portable environment platform.

  - Ruby provides fast development time with straightforward interpretation of code, cost-effective in fetch or updating data in Mysql database

- Thus it is clear that a wide variety of intermediates are available for Interfacing with RDBMS. Each technology has its advantage; most Programming Language are use depend on what kind of RDBMS we are interfacing with.

Ruby: Object-oriented Programming Language	**PHP: Web Scripting Language**
Open-sourced	Free software released under the PHP license
Works on multiple platforms	Easy to learn (Short learning curve)
Can be embedded into HTML	Large community of users and developers
A Very High-Level Language (VHLL)	Provides extensive database support
Offers encapsulation of data methods within objects	Provides significant number of available extensions and source codes
Pure OOP (Object-Oriented Programming)	Allows execution of code in restricted environments
Super advanced string and text manipulation techniques	Offers native session management and extension API
Compatible with DB2, MySQL, Oracle, and Sybase	Can be deployed on most web servers
Scalable & big programs written in Ruby are easily maintainable	Works on almost every operating system and platform
Ability to write multi-threaded applications	
Offers advanced array class	**Used by:**
Address external libraries in Ruby or C	Zend
Better security features	Yahoo
It has a debugger	Facebook
It has flexible syntax	Google
Powerful string handling	NASA
	W3C
Used by:	
Google Sketch up	
37signals	
GitHub	
Shopify	
Indiegogo	

Python: General Purpose Programming Language
Advantages
Easy and quick to learn
Runs on multiple systems and platforms
Readable and organized syntax
Offers rapid prototyping and dynamic semantics capabilities
Great community support
Easily construct applications by testing & importing crucial functions
Reusability through carefully implementing packages and modules
Object Oriented Programming-driven
Disadvantages
It does not really do multiprocessor/multi-core work very well
Smaller pool of Python developers compared to other languages
Database access limitation
Reputed to be slower than languages such as Java
Use By:
yahoo Map
Zope Corporation
Linux Weekly News
Shopville
Ultra seek

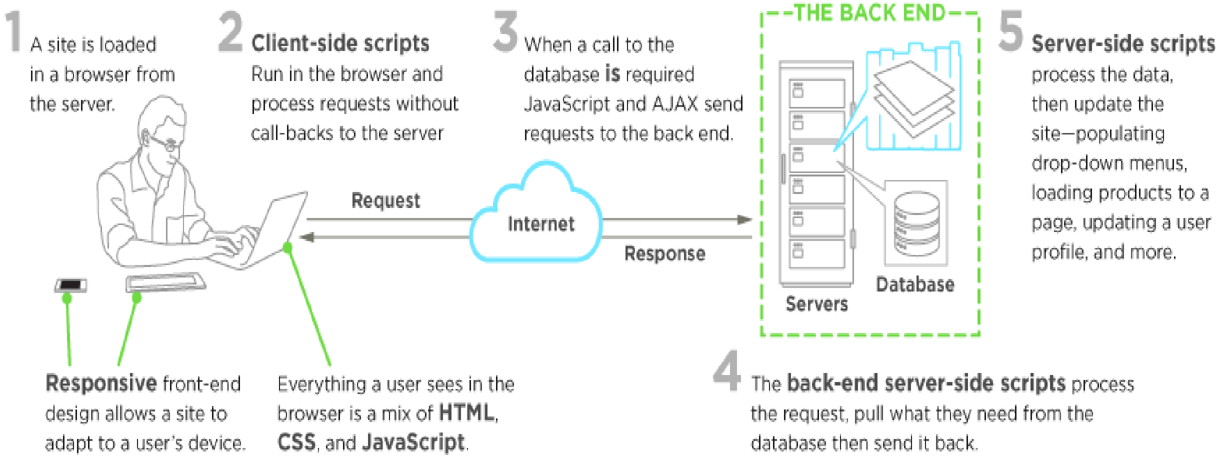
- **JavaScript** well known client-side scripting language used for preprocessing the user request before a call to the server is made to access the database. It works on the Front-end side of the 3-tier Application. JavaScript is used along with HTML/CSS at the web browser of the client.

- **JavaScript Frameworks**
  - **Angular JS –** for massive website
  - **Jquery** - for different platform like Android, iOS, Safari, Firefox
  - **Bootstrap** - for rapid application development
  - **Express**.js - A compile-to-JavaScript language that is a superset of JavaScript,
  - **AJAX –** combination of JavaScript and XML
  - **ASP.NET FRAMEWORK –** for running JScript & VBScript on the client browser.

- **Why should JavaScript not be Used?**
  - It focusses on an already created web page by server-side script but doesn't create one on dynamic request by the client, just perform alteration based on user input.
  - JavaScript cannot access the file system that resides on the web server and cannot be used to connect to the database on the web server.
  - **Security -** Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable JavaScript.
  - **Reliance on End User -** JavaScript is sometimes interpreted differently by different browsers. Whereas server-side scripts will always produce the same output, client-side

scripts can be a little unpredictable. Don't be overly concerned by this though - if you test your script in all the major browsers you should be safe.

- **Workflow**



1 A site is loaded in a browser from the server.

2 **Client-side scripts** Run in the browser and process requests without call-backs to the server

3 When a call to the database **is** required JavaScript and AJAX send requests to the back end.

4 The **back-end server-side scripts** process the request, pull what they need from the database then send it back.

5 **Server-side scripts** process the data, then update the site—populating drop-down menus, loading products to a page, updating a user profile, and more.

Responsive front-end design allows a site to adapt to a user's device.

Everything a user sees in the browser is a mix of **HTML**, **CSS**, and **JavaScript**.

- **Location**

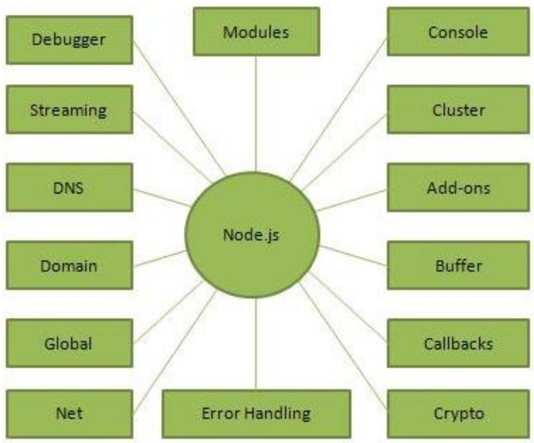
Client-side	Server-side
JavaScript	PHP
HTML	Python
CSS	Servlets(JSP)
XML	FastCGI
	CGI

- This is important to know which technology resides where because we can get a clear picture of the workflow from client -> server -> RDBMS.

- **Node JS** is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side . they are written in JavaScript and can run on OS X, Microsoft Windows & Linux at runtime. It contains a large library of JavaScript Modules which simplify web application development to a great extent.

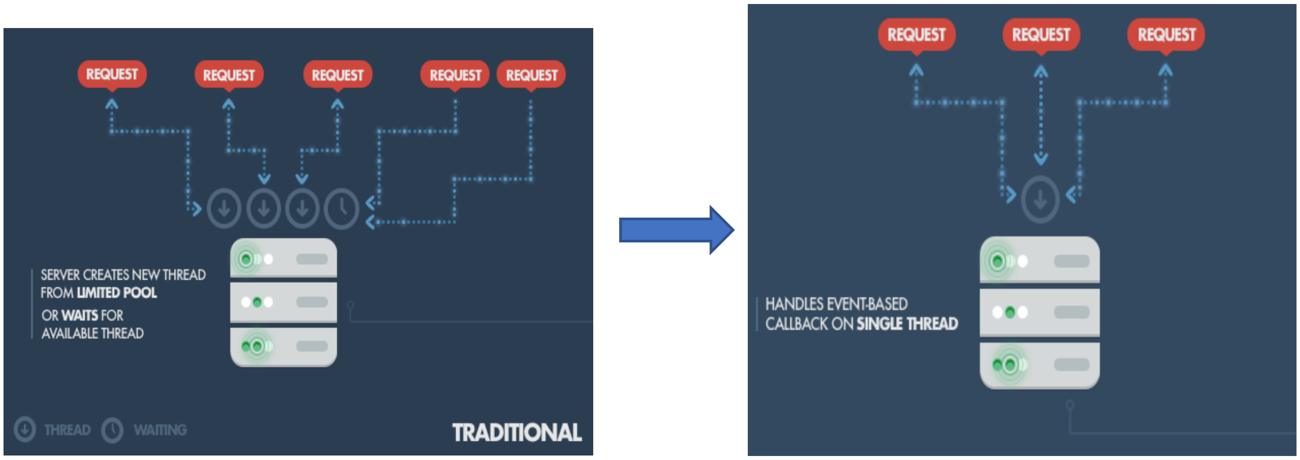
- Features
  - Asynchronous and Event Driven
  - Very Fast
  - Single thread but highly scalable
  - No buffering
  - License

It is used by many companies now a day by eBay, Microsoft, General Electric, Yahoo, and Google.



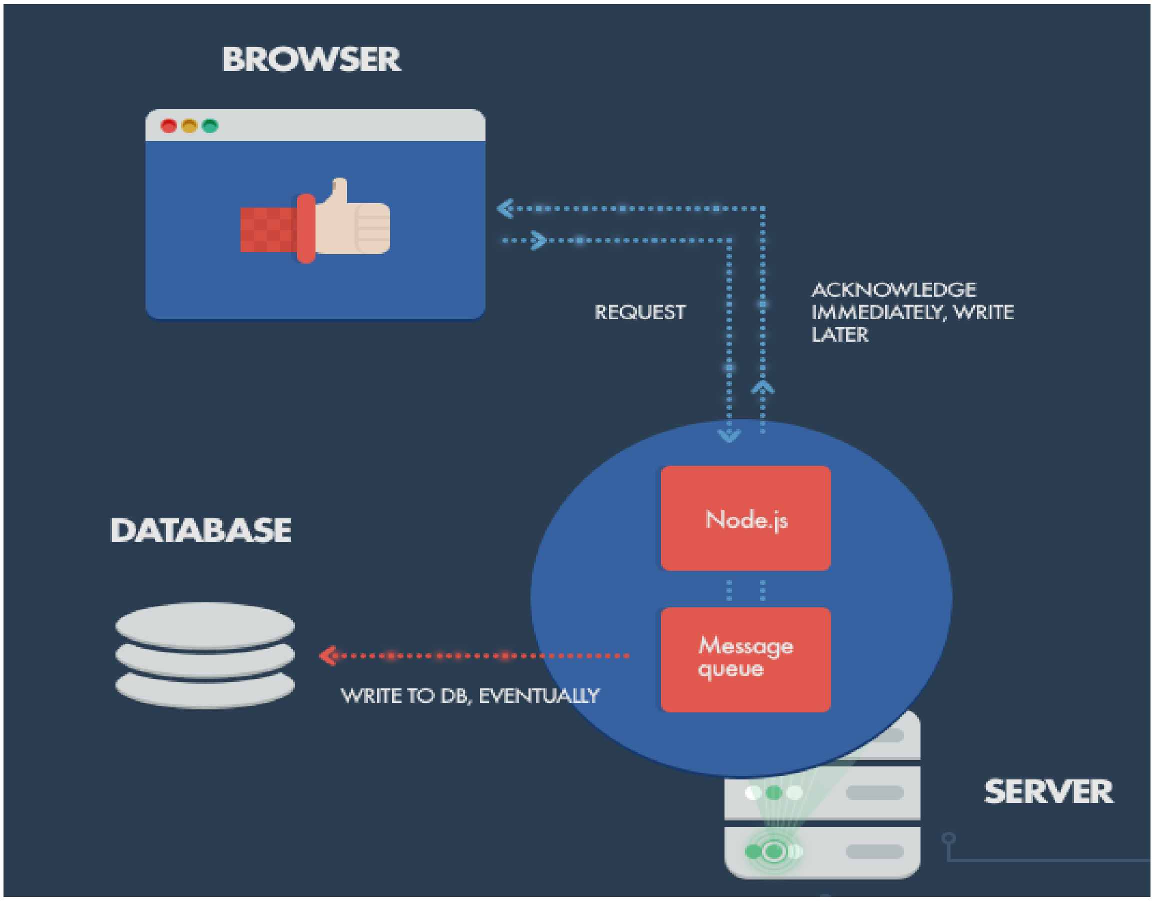
- Node.js is used for various application like
  - I/O bound application
  - Data streaming application
  - Data Intensive Real-time Application
  - JSON API

- **Node.js Integration with MySQL**
  We use the node-MySQL module for interfacing node.js with RDBMS. The main purpose of using this is because it allows a **connection pool** to RDBMS using single Thread instead of creating a new thread for each new request.


REQUEST
REQUEST
REQUEST
REQUEST
REQUEST
SERVER CREATES NEW THREAD FROM LIMITED POOL
OR WAITS FOR AVAILABLE THREAD
THREAD
WAITING
TRADITIONAL
REQUEST
REQUEST
REQUEST
HANDLES EVENT-BASED CALLBACK ON SINGLE THREAD

- NodeJS Interaction Workflow with RDBMS

  - Concurrent Request Execution made by the client through single thread pool connection to different users.



- Django

  It is a Python Web Framework that encourages rapid web development with clean, pragmatic design linked effeciently with back-end data. The primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components and the principle of don't repeat yourself. It also provides an optional administrative create, read, update & delete interface that is generated dynamically through introspection and configured via admin models

- Advantages
  - Ridiculously fast
  - Fully loaded with Libraries tools
  - Reassuringly Secure
  - Scalability
  - Versatile

- WSGI

  - The **Web Server Gateway Interface** (**WSGI**) Provides a high-level specification for simple and universal interface between web server and application logic (python-environment).
  - WSGI acts as a bridge between the server (apache or Nginx) and the python framework.
  - When WSGI request is Invoked , the server side executes the application and provides environment information and a callback function to the application side.

- The application logic’s programming environment reads the request, returning the response to the server-side using the callback function it was provided.
- For implementing Communication gap *WSGI middleware* is used . The client send request to server via this middleware. The application's response is delivered by the middleware to the server and ultimately to the client. Multiple middlewares forming a stack of WSGI-compliant applications.
- A "middleware" component is used to perform functions like Routing a request to different application objects based on Response URL, after changing the environment variables accordingly.Allowing multiple applications or frameworks to run side-by-side in the same processLoad balancing and remote processing, by forwarding requests and responses over a networkPerforming content post-processing, such as applying XSLT stylesheets

- **How WSGI is Connected to Python Application?**
  Apache uses port 80 or 8080 depending on the Availability to Http Request. once Http request is generated, WSGI parses it to find a way to respond to Client. It has number of ways available to response like use CGI as secondary to run a script or respond as file. In the first case where WSGI uses CGI, Apache prepares an environment and invoke the script through CGI protocol which is called as Unix Fork/exec Situation. Here CGI acts as a child process which includes sockets and stout against its Parent WSGI.

- The CGI subprocess writes a response, which goes back to Apache; Apache sends this response to the browser. CGI is primitive and annoying. Mostly because it forks a subprocess for every request, and subprocess must exit or close stdout and stderr to signify the end of the Process.

- WSGI is an interface that is based on the CGI design pattern

- It parses the HTTP Request Headers for you and adds these to the environment. It supplies any POST-oriented input as a file-like object in the environment. It also provides you a function that will formulate the response, saving you from many formatting details.

- Configuring WSGI with Python
  there are 2 main ways by which we can configure WSGI to read/write to Python.

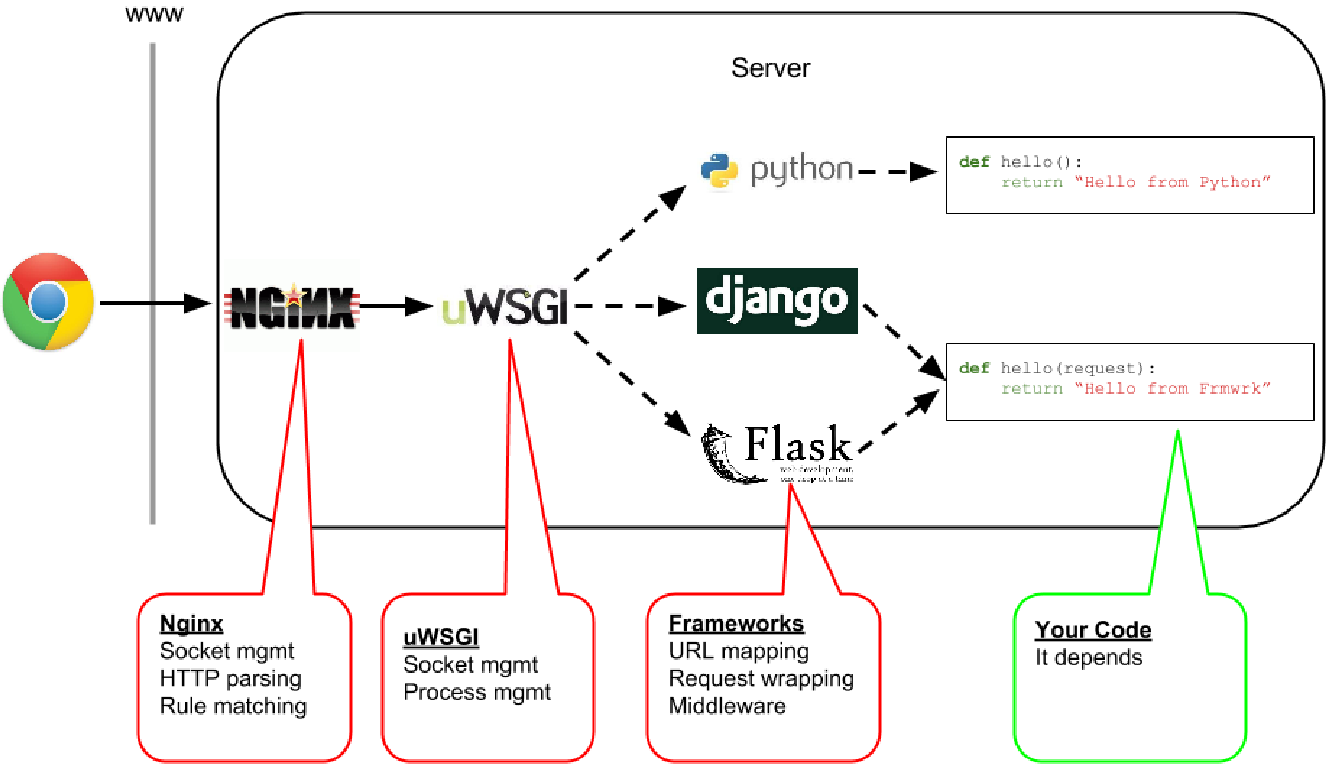
Embedded

- Open the Httpconif file in notepad of the Apache Software, look over the modules { mod_wsgi / mod_python } uncomment them !
- This changes makes python embeds inside Apache, as we know forking of 2 process is expensive, here by making changes no process is forked and apache runs the Django framework directly.

Daemon

- By configuring mod_wsgi / mod_fastcgi it allows Apache to Interact with a separate long running process using the WSGI protocol.
- Execute long-running Django process; then you configure Apache's mod_fastcgi to communicate with this process.

- Mod_wsgi can work in either mode: embedded or daemon.When mod_fastcgi is read , Django uses flup to create a WSGI-compatible interface from the information provided by mod_fastcgi.
- Pipeline Execution is Followed :
  Apache -> mod_fastcgi ->FLUP(using FastCGI) -> Django (using WSGI) -> program Request



```
def hello():
    return “Hello from Python”
```

```
def hello(request):
    return “Hello from Frmwrk”
```

References

1. All graphics and figures are taken from **Google images** internet.
2. We referred **Wikipedia** for most of the contents and we then manually written our essay.