CABKARO

Arjun Temura-	2020497
Aman Kumar-	2020492
Mohd Zaid-	2020525
Debijit Pramanik-	2020504

Scope of project:

The project implements a database management system for an online taxi booking web application, **CABKARO.**

- This app will help users to book cabs once they login into their account and select a pickup and drop location. Based on this information and the availability of drivers, we will map customers to drivers and register a ride.
- Customers(who book a cab) and employees(cab drivers) are the stakeholders
- The Customer can also select different payment options along with choice of taxi(eg Prime SUV, mini, auto etc)
- Once a ride is over, the driver is made availability is updated and the total fare is calculated.
- This fare depends on factors like time of booking, demand for taxis on that day etc.

Updated Schema

Entities:

1. customer-(user_id, name,email,mobile, password)
2. booking-(user_id, booking_id, current_location, pickup, destination, otp)
3. employees-(first name ,last name ,E_id ,Phone_no ,salary, availability)
4. cars-(car_type, car_model, registration_no, mileage ,seating)
5. fares-(car_type,start, end, cost/km)
6. luxury_cars-(Movies-tv,wifi)
7. distance-(total_price,distance, booking_id)
8. payment-(payment_id, payment_type)

Relationships:

1. Registers: Customer to booking-one to many

2. Total_journey: Booking to distance-one to one

3. Cost: Cars to fares-one to one

4. Luxury_cars is a child relation of cars relation (inheritance)

5. Rides relation: Ternary relationship between booking , cars and employees (one to one between any to relations)

6. PaymentType:Payment to Booking-one to many

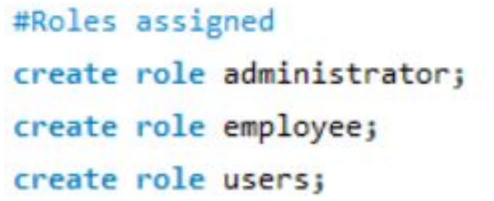
Changes after mid evaluation

1. Boolean attribute in Employee table (availability) to keep track Availability of Driver at a particular time.
2. Updated the ‘fares’ table, to store fares based on time of booking(eg. Fares are higher during office hours). New attribute **start, end** to keep track of the interval of cab booking.
3. New table “Payment” , with attributes payment_id(int) and payment_type (varchar)to specify the payment method for a ride.(UPI/Credit Card etc) . Therefore a new attribute, payment_id was added to the booking table

Roles assigned:

1. **Administrator**
2. **Employee**
3. **Users**

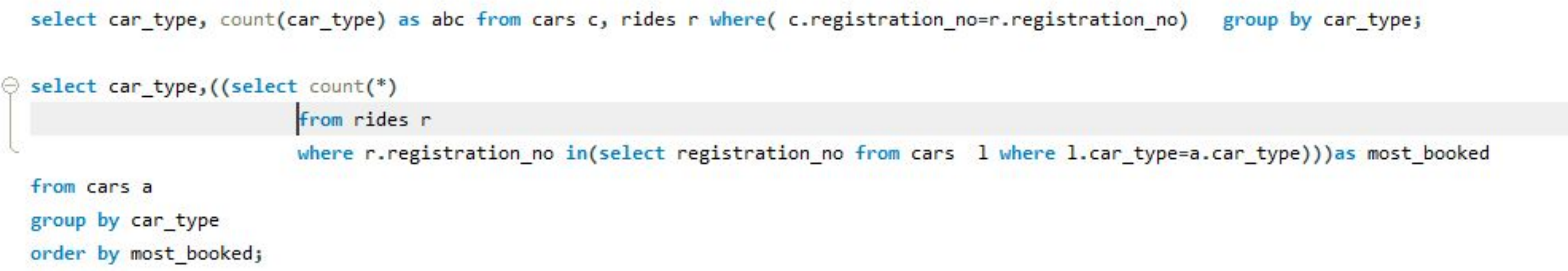
Grants and Views for all roles are implemented in MySQL.



```
#Roles assigned
create role administrator;
create role employee;
create role users;
```

SQL Query Optimisation:

1. **Counting how many cars of a particular car type has been booked in a day**.

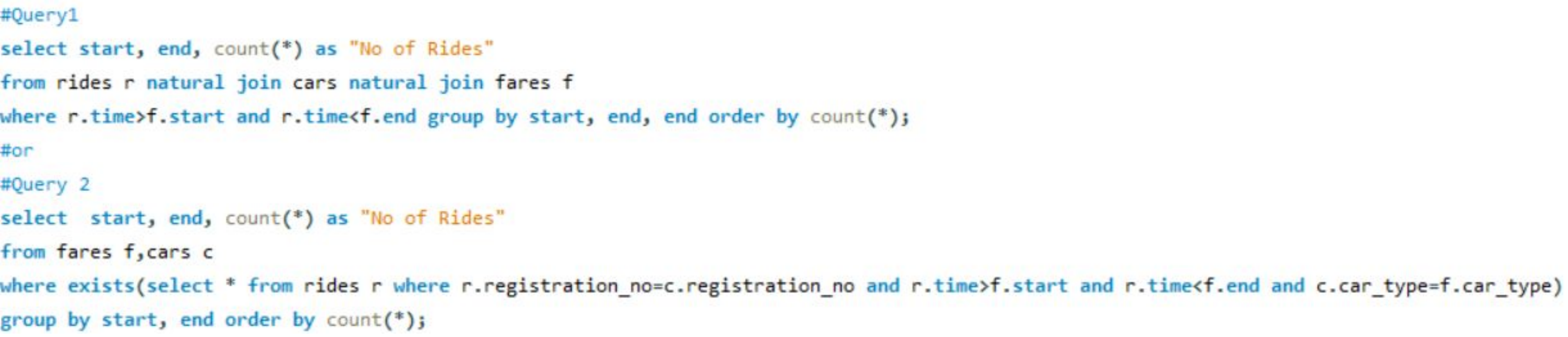


```sql
select car_type, count(car_type) as abc from cars c, rides r where( c.registration_no=r.registration_no)   group by car_type;

select car_type,((select count(*)
                    from rides r
                    where r.registration_no in(select registration_no from cars  l where l.car_type=a.car_type))as most_booked
from cars a
group by car_type
order by most_booked;
```

These two queries yield the same result.

However, the one with embedded query (**second quer**y) could take more time than the simple join one(**first query**). So here we could use the **join one (first query)** as the optimal query.

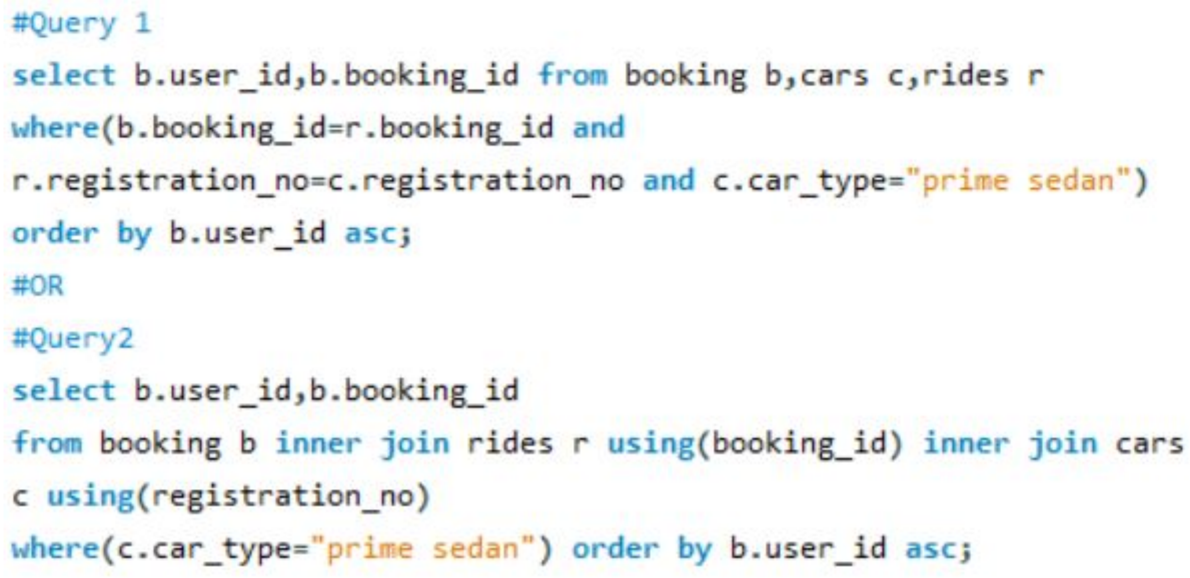
2. To show no. of rides booked in different time intervals for the day



```
#Query1
select start, end, count(*) as "No of Rides"
from rides r natural join cars natural join fares f
where r.time>f.start and r.time<f.end group by start, end, end order by count(*);
#or
#Query 2
select  start, end, count(*) as "No of Rides"
from fares f,cars c
where exists(select * from rides r where r.registration_no=c.registration_no and r.time>f.start and r.time<f.end and c.car_type=f.car_type)
group by start, end order by count(*);
```

Among the two queries, Query 1 uses natural join operation on 3 tables and Query 2 uses nested queries with nested queries. As nested query will take more time to give the same result, **Query 1 is more optimal**

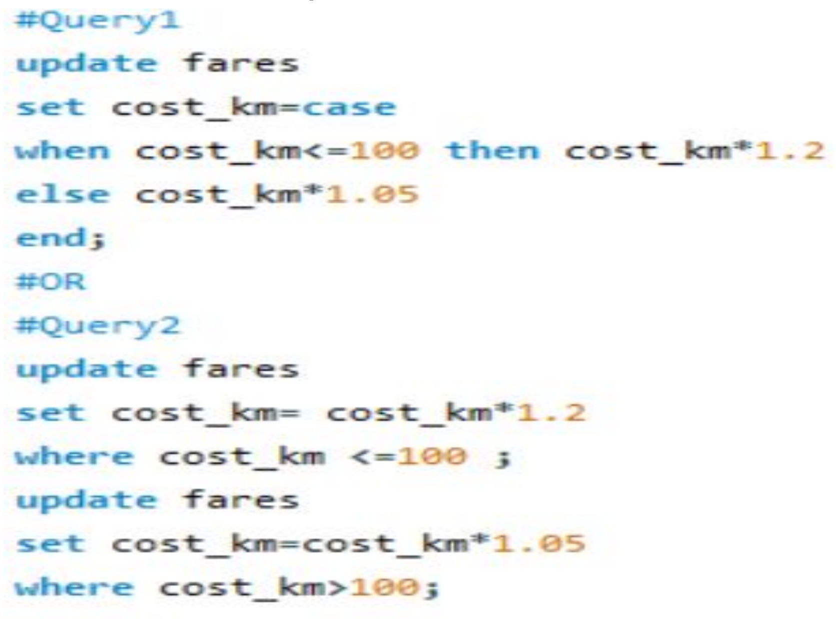
3. To find customers who traveled in prime sedan?



```
#Query 1
select b.user_id,b.booking_id from booking b,cars c,rides r
where(b.booking_id=r.booking_id and
r.registration_no=c.registration_no and c.car_type="prime sedan")
order by b.user_id asc;
#OR
#Query2
select b.user_id,b.booking_id
from booking b inner join rides r using(booking_id) inner join cars
c using(registration_no)
where(c.car_type="prime sedan") order by b.user_id asc;
```

In query 2, we have used inner join on booking, rides and cars whereas in query 1 we have used nested queries with join. Clearly **Query 2 is optimal**

4. Update the cost/km depending on the present fares(in response to fuel price hike)



```
#Query1
update fares
set cost_km=case
when cost_km<=100 then cost_km*1.2
else cost_km*1.05
end;
#OR
#Query2
update fares
set cost_km= cost_km*1.2
where cost_km <=100 ;
update fares
set cost_km=cost_km*1.05
where cost_km>100;
```

Query 1 uses switch case to complete the task in one query. However, Query 2 does the same task using two queries separately using where clause. Hence, **Query 1 is more optimal.**

Query optimization

5. Indexing

The use of proper indexing is first step to optimise the query. There are some important steps to use or create index .

Indexing helps in faster retrieval of records

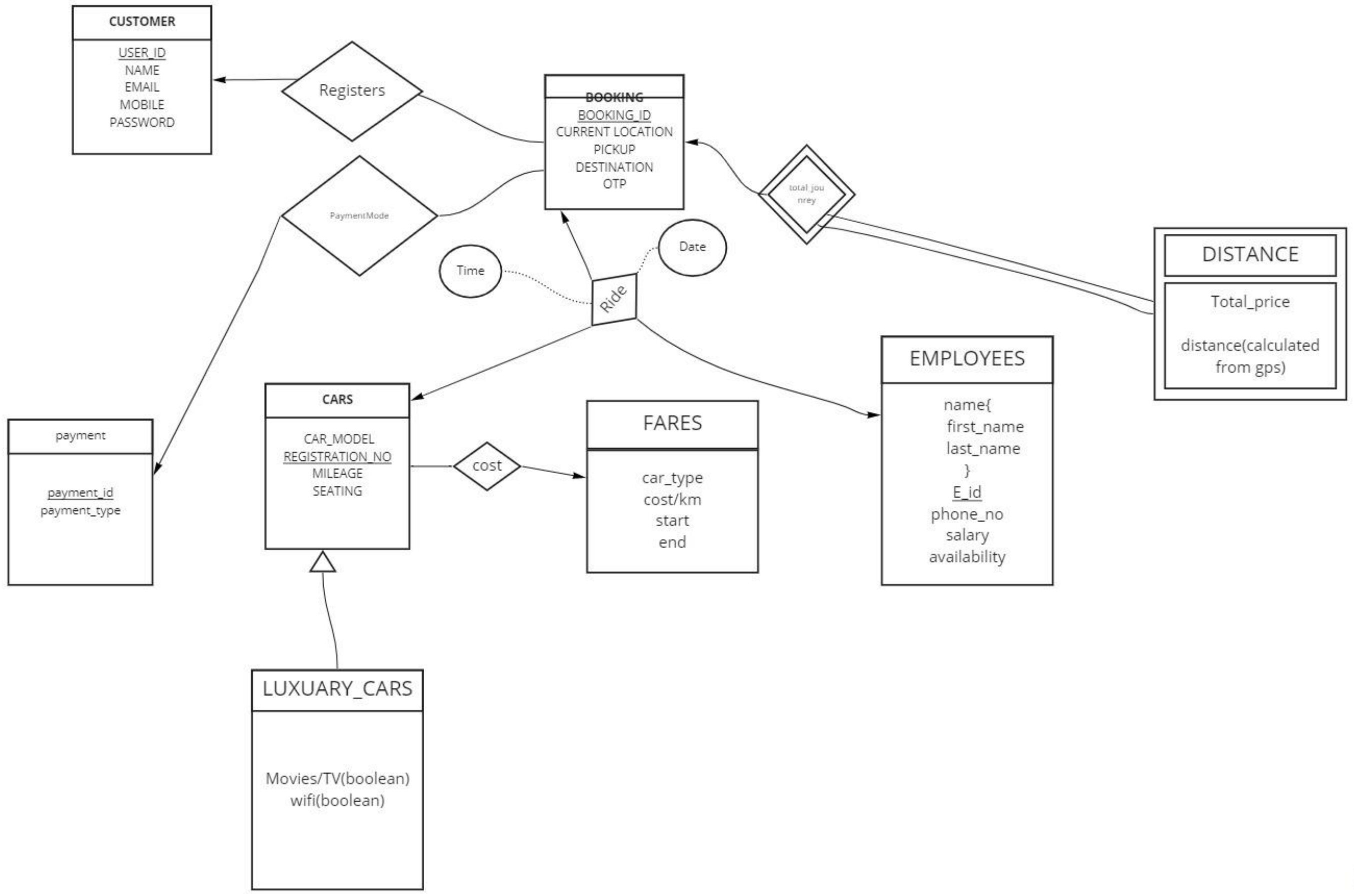
For example if we want to search tuples based on some other attributes except the   primary key

We would end up with much tuples with that same attribute

In such case we might require a unique index not which is not just the primary key but combination fo that attribute which is not

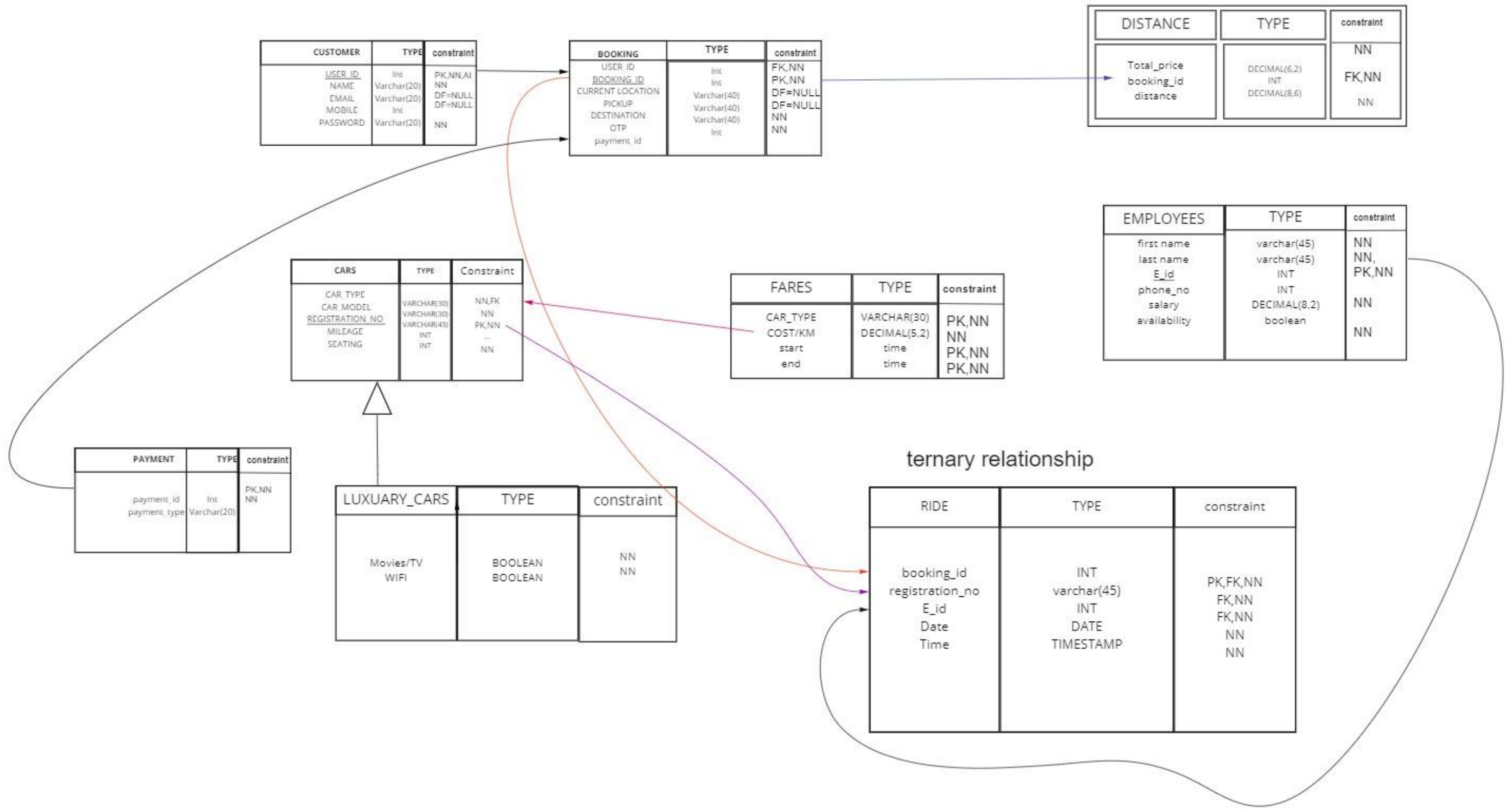
Primary key and of the primary key

ER


CUSTOMER
USER_ID
NAME
EMAIL
MOBILE
PASSWORD
Registers
BOOKING
BOOKING_ID
CURRENT LOCATION
PICKUP
DESTINATION
OTP
total jou nrey
DISTANCE
Total_price
distance(calculated from gps)
PaymentMode
Time
Date
Ride
payment
payment_id
payment_type
CARS
CAR_MODEL
REGISTRATION_NO
MILEAGE
SEATING
cost
FARES
car_type
cost/km
start
end
EMPLOYEES
name{
first_name
last_name
}
E_id
phone_no
salary
availability
LUXUARY_CARS
Movies/TV(boolean)
wifi(boolean)

miro

logical database design



CUSTOMER	TYPE	constraint
USER ID	Int	PK,NN,AI
NAME	Varchar(20)	NN
EMAIL	Varchar(20)	DF=NULL
MOBILE	Int	DF=NULL
PASSWORD	Varchar(20)	NN

BOOKING	TYPE	constraint
USER ID	Int	FK,NN
BOOKING ID	Int	PK,NN
CURRENT LOCATION	Varchar(40)	DF=NULL
PICKUP	Varchar(40)	DF=NULL
DESTINATION	Varchar(40)	NN
OTP	Int	NN
payment id		

DISTANCE	TYPE	constraint
Total_price	DECIMAL(6,2)	NN
booking_id	INT	FK,NN
distance	DECIMAL(8,6)	NN

EMPLOYEES	TYPE	constraint
first name	varchar(45)	NN
last name	varchar(45)	NN,
E_id	INT	PK,NN
phone_no	INT	
salary	DECIMAL(8,2)	NN
availability	boolean	NN

CARS	TYPE	Constraint
CAR TYPE	VARCHAR(30)	NN,FK
CAR MODEL	VARCHAR(30)	NN
REGISTRATION NO	VARCHAR(45)	PK,NN
MILEAGE	INT	–
SEATING	INT	NN

FARES	TYPE	constraint
CAR_TYPE	VARCHAR(30)	PK,NN
COST/KM	DECIMAL(5,2)	NN
start	time	PK,NN
end	time	PK,NN

PAYMENT	TYPE	constraint
payment id	Int	PK,NN
payment type	Varchar(20)	NN

LUXUARY_CARS	TYPE	constraint
Movies/TV	BOOLEAN	NN
WIFI	BOOLEAN	NN

RIDE	TYPE	constraint
booking_id	INT	PK,FK,NN
registration_no	varchar(45)	FK,NN
E_id	INT	FK,NN
Date	DATE	NN
Time	TIMESTAMP	NN

TRIGGERS

1. In rides table we had created a before insert trigger, this trigger insert or set date and time of record to current time.

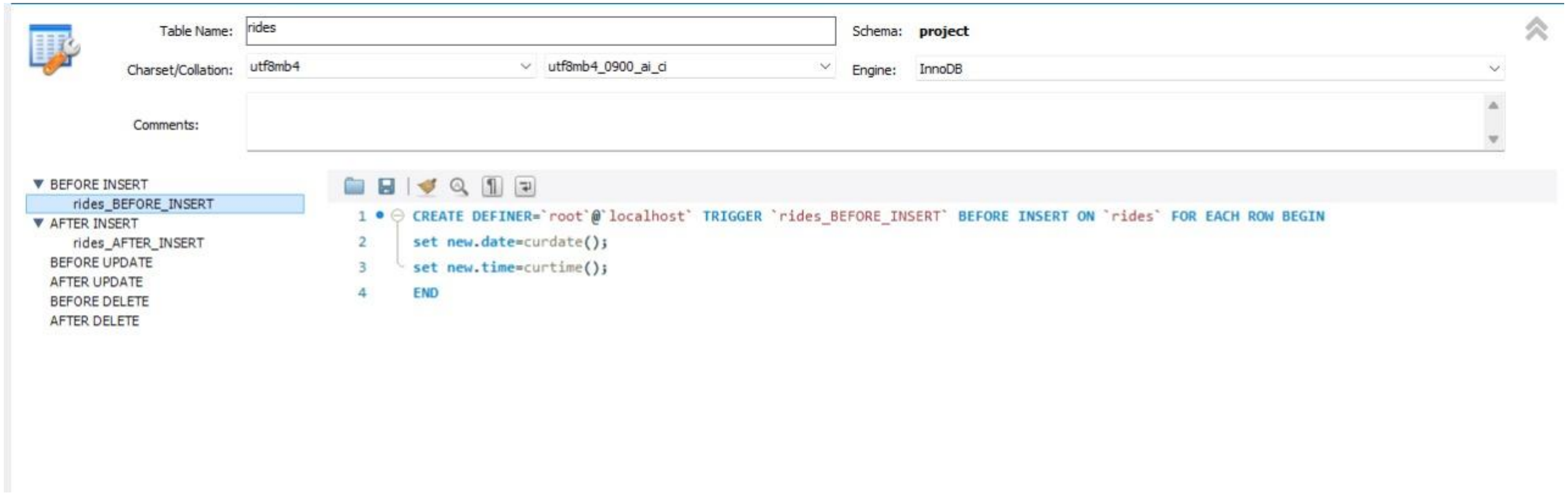


Table Name: rides
Schema: **project**
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Engine: InnoDB
Comments:

- ▼ BEFORE INSERT
  - rides_BEFORE_INSERT
- ▼ AFTER INSERT
  - rides_AFTER_INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

```
CREATE DEFINER=`root`@`localhost` TRIGGER `rides_BEFORE_INSERT` BEFORE INSERT ON `rides` FOR EACH ROW BEGIN
set new.date=curdate();
set new.time=curtime();
END
```

2. In rides table we had created after insert trigger, this trigger makes changes to employees table it sets availability to false or 0.

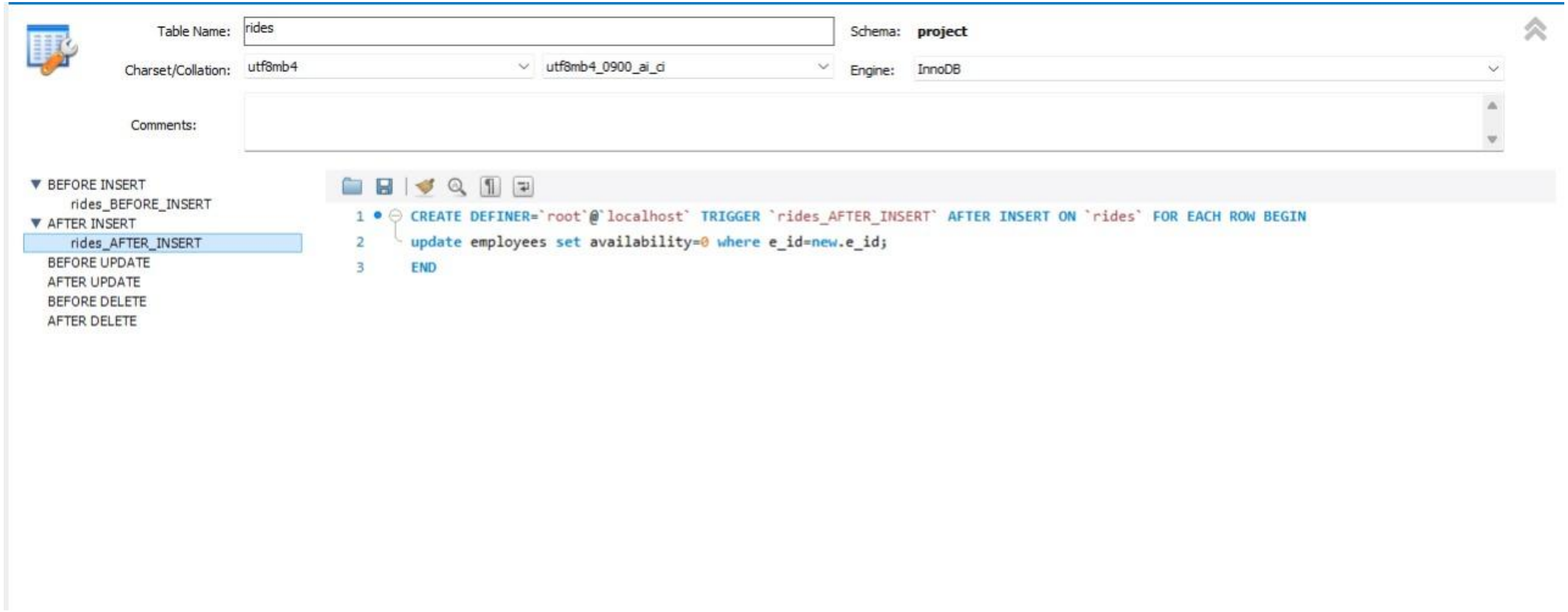


Table Name: rides
Schema: project
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Engine: InnoDB
Comments:

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `rides_AFTER_INSERT` AFTER INSERT ON `rides` FOR EACH ROW BEGIN
update employees set availability=0 where e_id=new.e_id;
END
```

3. In customer table we created before insert trigger, this trigger gets triggered when u try to entry password less than 8 characters it will give you an error.

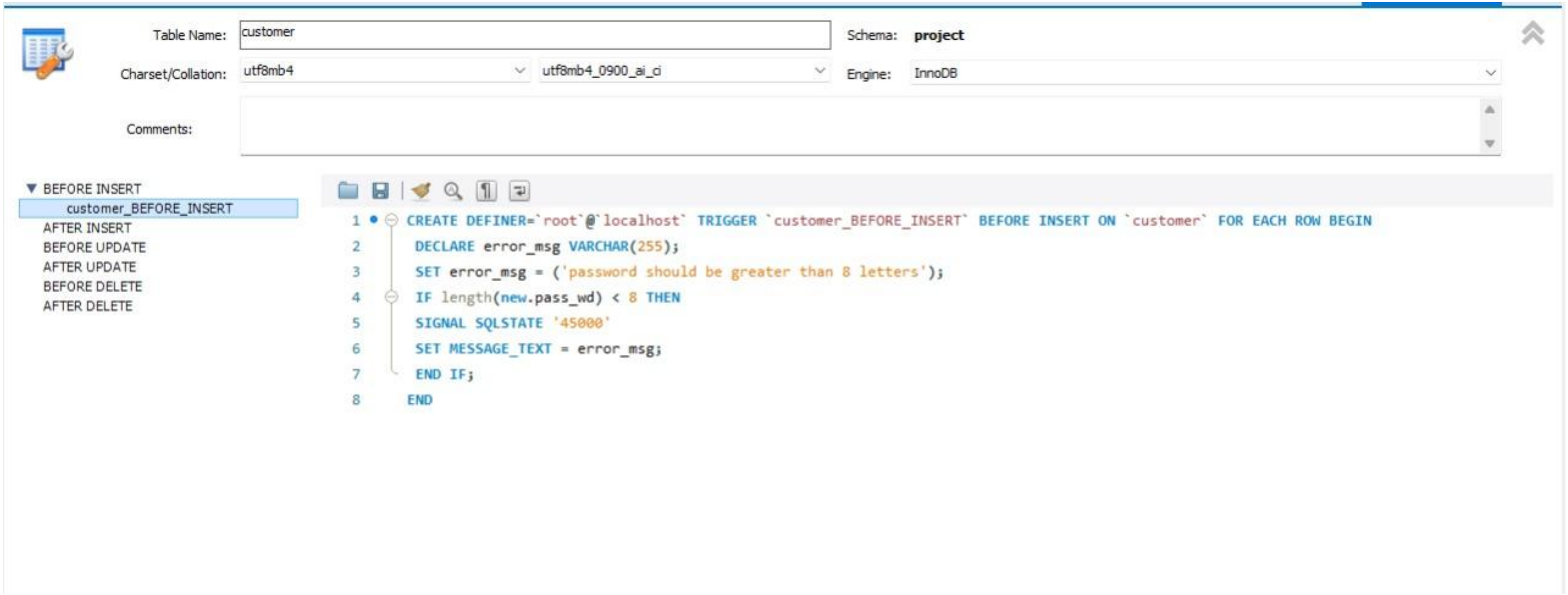


Table Name: customer
Schema: **project**
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Engine: InnoDB
Comments:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `customer_BEFORE_INSERT` BEFORE INSERT ON `customer` FOR EACH ROW BEGIN
 DECLARE error_msg VARCHAR(255);
 SET error_msg = ('password should be greater than 8 letters');
 IF length(new.pass_wd) < 8 THEN
 SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = error_msg;
 END IF;
END
```

4. In booking table we created before insert trigger, this trigger gets triggered when u insert a row in table and trigger adds OTP for you.

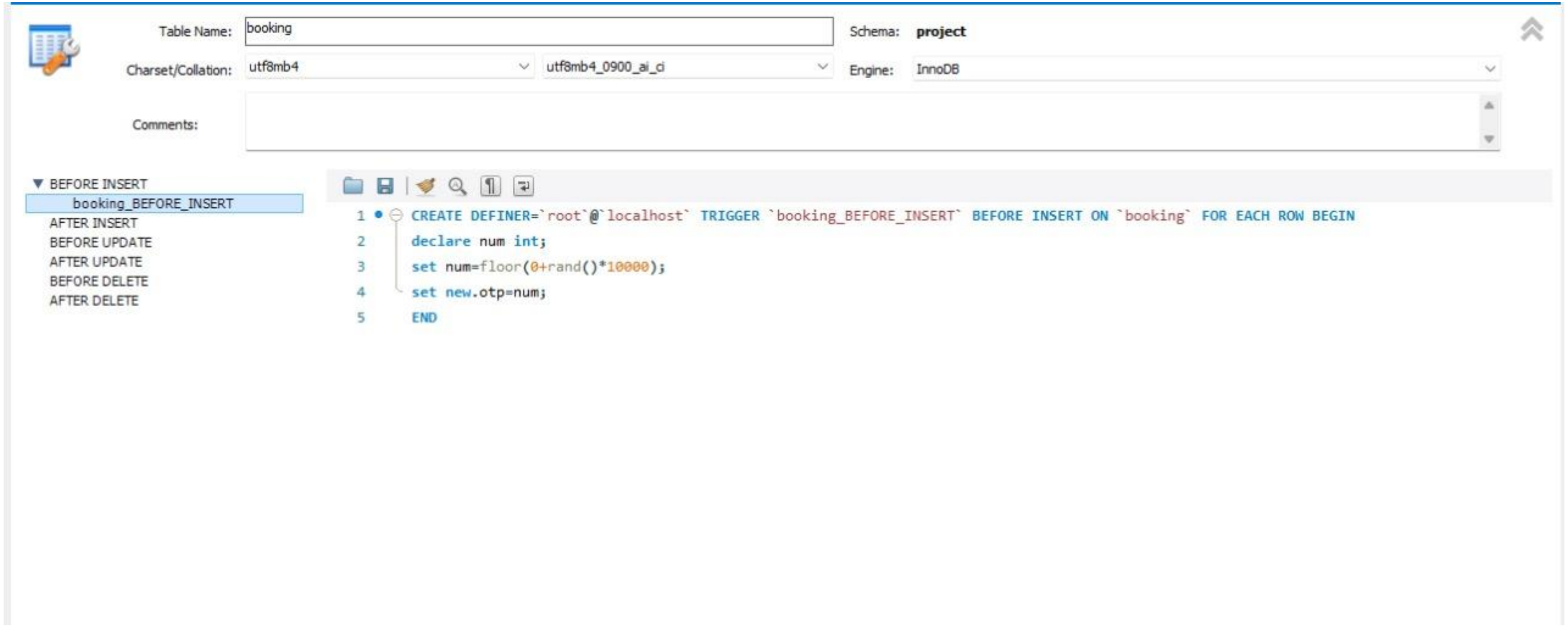


Table Name: booking
Schema: project
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Engine: InnoDB
Comments:

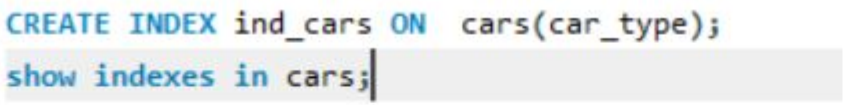
```
CREATE DEFINER=`root`@`localhost` TRIGGER `booking_BEFORE_INSERT` BEFORE INSERT ON `booking` FOR EACH ROW BEGIN
declare num int;
set num=floor(0+rand()*10000);
set new.otp=num;
END
```

INDEXING

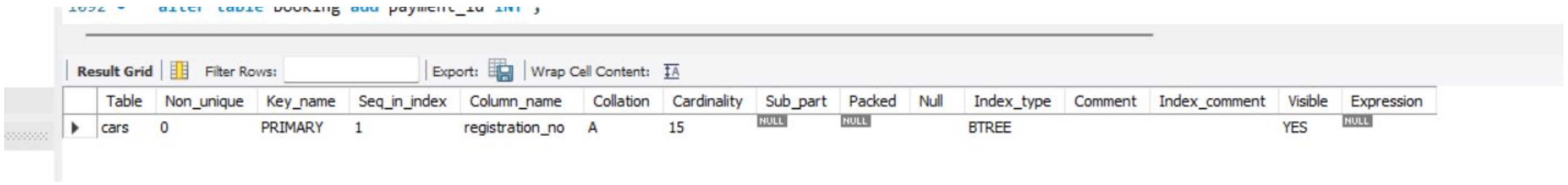
```
CREATE INDEX ind_ride ON rides (booking_id,time);
CREATE INDEX ind_customer ON customer (name,user_id);


CREATE INDEX ind_booking ON booking (user_id);
CREATE INDEX ind_emp ON employees(first_name,e_id);
CREATE INDEX ind_cars ON  cars(car_type);
CREATE Unique INDEX ind_fares ON fares (start,end,car_type);
```

TO CHECK INDEXES ON TABLE WE USE SHOW INDEXES from table_name command

cars



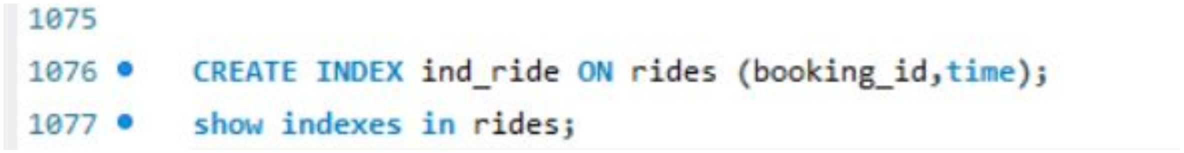
```sql
CREATE INDEX ind_cars ON  cars(car_type);
show indexes in cars;
```



```sql
alter table booking add payment_id INT ;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
cars	0	PRIMARY	1	registration_no	A	15	NULL	NULL		BTREE			YES	NULL

rides



```
CREATE INDEX ind_ride ON rides (booking_id,time);
show indexes in rides;
```

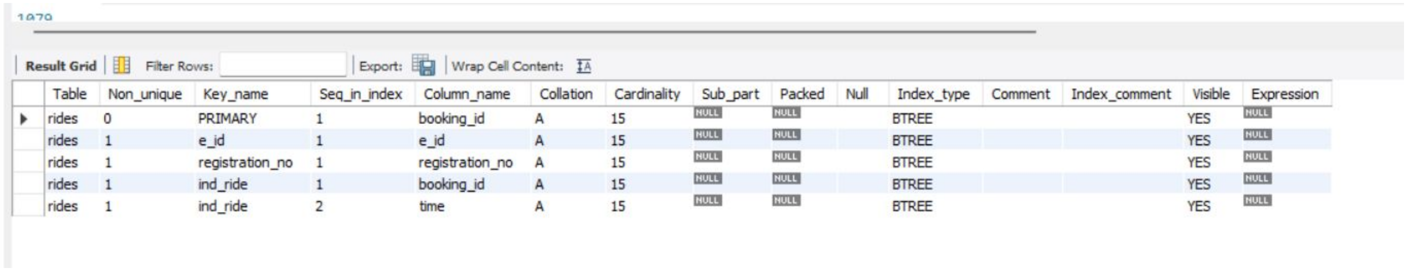


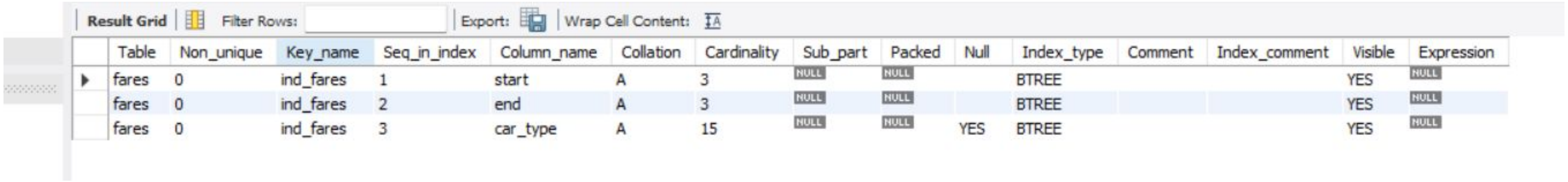
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
rides	0	PRIMARY	1	booking_id	A	15	NULL	NULL		BTREE			YES	NULL
rides	1	e_id	1	e_id	A	15	NULL	NULL		BTREE			YES	NULL
rides	1	registration_no	1	registration_no	A	15	NULL	NULL		BTREE			YES	NULL
rides	1	ind_ride	1	booking_id	A	15	NULL	NULL		BTREE			YES	NULL
rides	1	ind_ride	2	time	A	15	NULL	NULL		BTREE			YES	NULL

fares

```
CREATE Unique INDEX ind_fares ON fares (start,end,car_type);
explain select car_type from fares where start >'11:00:00';
```

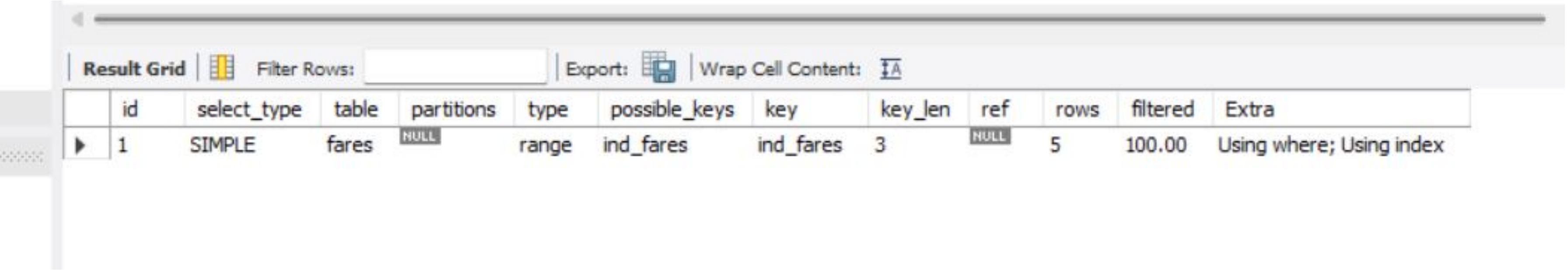
```
CREATE Unique INDEX ind_fares ON fares (start,end,car_type);
```

```
show indexes in fares;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

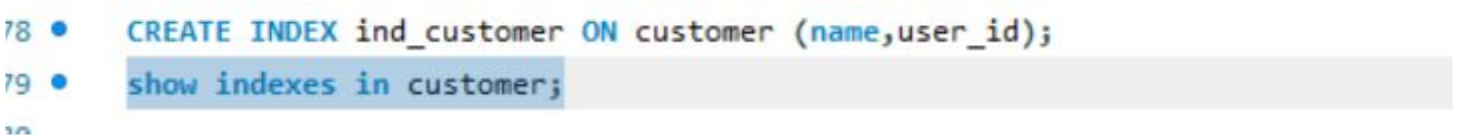
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
fares	0	ind_fares	1	start	A	3	NULL	NULL		BTREE			YES	NULL
fares	0	ind_fares	2	end	A	3	NULL	NULL		BTREE			YES	NULL
fares	0	ind_fares	3	car_type	A	15	NULL	NULL	YES	BTREE			YES	NULL



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	fares	NULL	range	ind_fares	ind_fares	3	NULL	5	100.00	Using where; Using index

customer



```sql
CREATE INDEX ind_customer ON customer (name,user_id);
show indexes in customer;
```

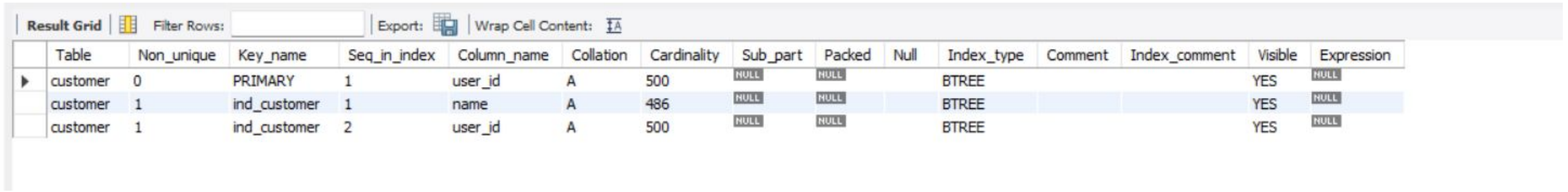
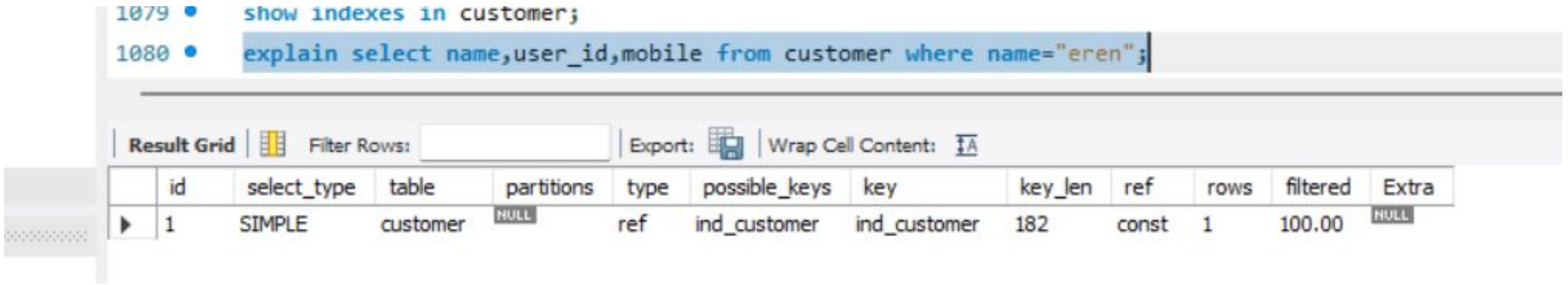


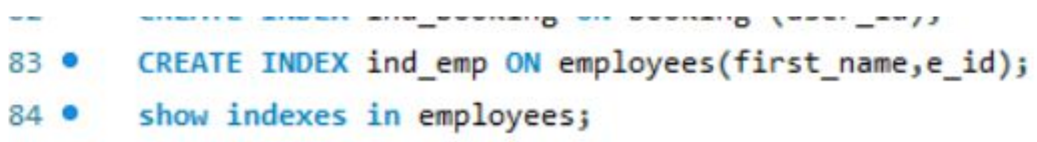
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
customer	0	PRIMARY	1	user_id	A	500	NULL	NULL		BTREE			YES	NULL
customer	1	ind_customer	1	name	A	486	NULL	NULL		BTREE			YES	NULL
customer	1	ind_customer	2	user_id	A	500	NULL	NULL		BTREE			YES	NULL



```sql
show indexes in customer;
explain select name,user_id,mobile from customer where name="eren";
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	customer	NULL	ref	ind_customer	ind_customer	182	const	1	100.00	NULL

employees



```sql
CREATE INDEX ind_emp ON employees(first_name,e_id);
show indexes in employees;
```

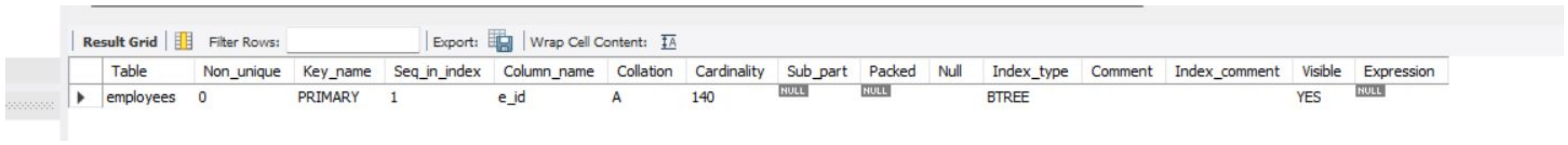
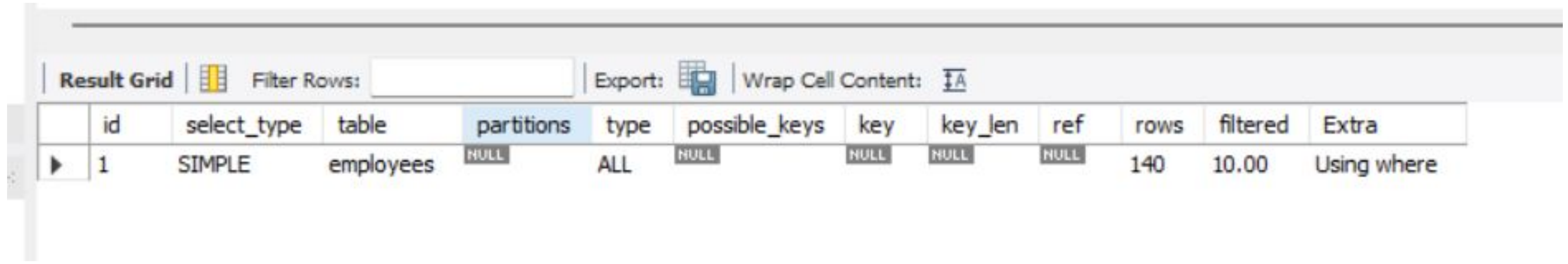


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
employees	0	PRIMARY	1	e_id	A	140	NULL	NULL		BTREE			YES	NULL



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	140	10.00	Using where