Queen Mary University of London

Semi-structured data and advanced data modelling

Project Report

Database Management System for Taxi Company

Group 1 Members

Xenofon Georgitsaros id(200922308)

Konstantinos Merkouriadis student id(200916938)

Supervisor

Anthony Stockman

This project is about data management requirement for a taxi company. Data is stored and managed using Oracle relational database management system. Data is stored and managed for different subject areas including drivers, cars, operators, booking details, payment information, and revenue, client/customers, and driver shifts. Under this project, first logical database design has been created using entity relationship model depicting different entities for a taxi company and associations between these entities. Furthermore, physical database has been created using database table objects; data integrity has been managed using triggers. Few examples of query performance tuning and optimization have also been added towards the end of project work. Some implementation assumptions have been made while designing this database system.

Design assumptions:

Following assumptions have been made for database design

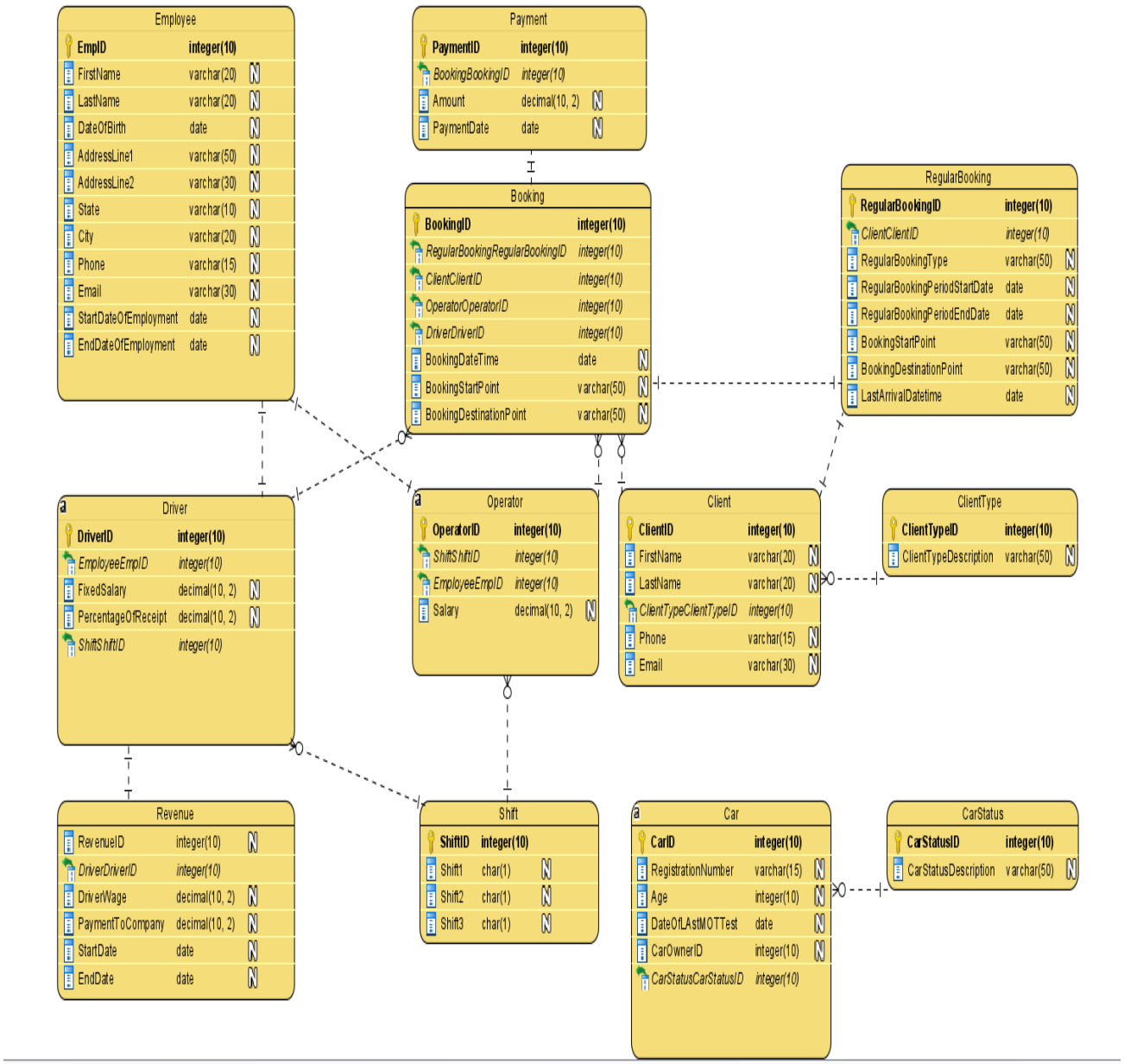
- System generated column has been added to most of tables for identification. These columns do not have any business significance.
- Identification column has been taken as integer for better performance
- Client data is normalized using client and client type entities
- Drivers and operators have been considered as employee of Taxi Company
- There are three shifts of eight hours each in a single day to provide 24x7 service
- Different table is used for Carstatus in order to normalize the cat table.
- Reference table will be populated first before their child tables eg CarStatus will be populated before Car table; similarly, ClientType table will be populated before Client table.

Database subject areas and tables:

Below is the list of tables used for storing information for Taxi Company

SNO	Table name	Description
1	Employee	Contains employee personal information and demographic
2	Driver	Contains information about driver shift, rate
3	Operator	Contains information about operator salary and shift
4	Shift	Contains information about different shift timings
5	Car	Contains information about car like registration number, owner of car, age of car, status of car
6	CarStatus	Contain information about car status as per requirement given.
6	Client	Contains information about client/customer
7	ClientType	Reference table containing different client types
8	Booking	Contains booking information
9	RegularBooking	Contains information of regular booking by client
10	Payment	Contains information about payment for booking
11	Revenue	Contains information about revenue earned by driver and company

Relational database design: First step in database design is to identify business entities and their association/relationship. A logical model entity-relationship diagram has been created for taxi company database using data modelling tool. It represents different entities and their relationships as follow. All primary key and foreign keys relationships have been depicted in this diagram. Degree of relationship (one-to-one, one-to-many etc.) is also depicted.


Employee
EmplID integer(10)
FirstName varchar(20) N
LastName varchar(20) N
DateOfBirth date N
AddressLine1 varchar(50) N
AddressLine2 varchar(30) N
State varchar(10) N
City varchar(20) N
Phone varchar(15) N
Email varchar(30) N
StartDateOfEmployment date N
EndDateOfEmployment date N
Payment
PaymentID integer(10)
BookingBookingID integer(10)
Amount decimal(10, 2) N
PaymentDate date N
Booking
BookingID integer(10)
RegularBookingRegularBookingID integer(10)
ClientClientID integer(10)
OperatorOperatorID integer(10)
DriverDriverID integer(10)
BookingDateTime date N
BookingStartPoint varchar(50) N
BookingDestinationPoint varchar(50) N
RegularBooking
RegularBookingID integer(10)
ClientClientID integer(10)
RegularBookingType varchar(50) N
RegularBookingPeriodStartDate date N
RegularBookingPeriodEndDate date N
BookingStartPoint varchar(50) N
BookingDestinationPoint varchar(50) N
LastArrivalDatetime date N
Driver
DriverID integer(10)
EmployeeEmplID integer(10)
FixedSalary decimal(10, 2) N
PercentageOfReceipt decimal(10, 2) N
ShiftShiftID integer(10)
Operator
OperatorID integer(10)
ShiftShiftID integer(10)
EmployeeEmplID integer(10)
Salary decimal(10, 2) N
Client
ClientID integer(10)
FirstName varchar(20) N
LastName varchar(20) N
ClientTypeClientTypeID integer(10)
Phone varchar(15) N
Email varchar(30) N
ClientType
ClientTypeID integer(10)
ClientTypeDescription varchar(50) N
Revenue
RevenueID integer(10) N
DriverDriverID integer(10)
DriverWage decimal(10, 2) N
PaymentToCompany decimal(10, 2) N
StartDate date N
EndDate date N
Shift
ShiftID integer(10)
Shift1 char(1) N
Shift2 char(1) N
Shift3 char(1) N
Car
CarID integer(10)
RegistrationNumber varchar(15) N
Age integer(10) N
DateOfLastMOTTest date N
CarOwnerID integer(10) N
CarStatusCarStatusID integer(10)
CarStatus
CarStatusID integer(10)
CarStatusDescription varchar(50) N

Following is description about each entity involved in data model

Employee Table: It holds personal and demographic information about taxi company employee. Operators and drivers are employee to taxi company. Following is table definition for employee table

ColumnName	DataType	Column Description
EmpID	Integer	Unique identified for employee
FirstName	Varchar	Employee first name
LastName	Varchar	Employee last name
DateOfBirth	Date	Employee date of birth
AddressLine1	Varchar	Employee address line 1
AddressLine2	Varchar	Employee address line 2
State	Varchar	Employee state
City	Varchar	Employee City
Phone	Varchar	Employee phone
Email	Varchar	Employee email address
StartDateOfEmployment	Date	Employement start date
EndDateOfEmployment	Date	Employement end date

Driver Table: It holds information about car driver shifts and salary. Following is definition of car driver table.

ColumnName	DataType	Column Description
DriverID	Integer	Unique identifier for Driver
EmpId	Integer	Unique identifier for employee. Referencing EmpId from Employee table
ShiftID	Integer	Unique identifier for Shift. Referencing ShiftID from Shift table
FixedSalary	Decimal	Fixed salary for driver
PercentageOfReceipt	Decimal	Driver earning as Percentage of receipt

Operator table: It holds information about operators who take car booking order and assign them to driver. Following is definition of operator table.

ColumnName	DataType	Column Description
OperatorID	Integer	Unique identifier for operator
EmpId	Integer	Unique identifier for employee . Referencing EmpId from Employee table
ShiftID	Integer	Unique identifier for Shift. Referencing ShiftID from Shift table
Salary	Decimal	Salary of Operator

Shift Table: It holds information about different shift schedules. There are three shift timings of eight hours each.

ColumnName	DataType	Column Description
ShiftID	Integer	Unique identifier for shift
Shift1	Boolean	Fist Shift 8:0:0 hours - 16:0:0 hours
Shift2	Boolean	Second shift 16:0:0 hours - 0:0:0 hours
Shift3	Boolean	Third Shift 0:0:0 hours - 8:0:0 hours

Car Table: It holds information for a car including its registration number, age of car, car status , car owner and date of last MOT Test.

ColumnName	DataType	Column Description
CarID	Interger	Unique identifier for car
RegistrationNumber	Varchar	Car registration number
Age	Interger	Age of car
DateOfLAstMOTTest	Date	Date for last MOT test
CarStatusID	Interger	Unique indetifer of car status linked to CarStatus table
CarOwnerID	Interger	Car owner id referencing Driverid from driver table

CarStatus Table: It holds information on car status. It is a kind of extension table/reference table to car table.

ColumnName	DataType	Column Description
CarStatusID	Integer	Unique identifier for Car Status
CarStatusDescription	Varchar	Valid values are roadworthy,inforservice,awaiting repair, written off

Client Table: It holds information about car booking client. It contains information like First Name, Last Name, Phone, Email and client type ID depicting whether a client is private client or corporate client.

ColumnName	DataType	Column Description
ClientID	Integer	Unique identifier for client
FirstName	Varchar	Client first name
LastName	Varchar	Client last name
ClientTypeID	Integer	Unique identifier for clientType. Referencing ClientType table
Phone	Varchar	Client Phone number
Email	Varchar	Client Email Address

ClientType Table: It holds information about differnet type of clients. It is kind of extension/reference table to main client table. Currently, it holds information about two different clientType like Private Client and Corporate client.

ColumnName	DataType	Column Description
ClientTypeID	Integer	Unique identifier for Client Type
ClientTypeDescription	Varchar	Client type description ; Valid values are private and corporate

Booking Table: It holds information about booking made by clients. It include relationship to client , driver, oprator, booking start date , booking period end data and reference to RegularBooking Table.

ColumnName	DataType	Column Description
BookingID	Integer	Unique identifier for booking
ClientID	Integer	Unique identifier for client. Referencing ClientID from Client table
OperatorID	Integer	Unique identifier for operator. Referencing OperatorID from Operator table
DriverID	Integer	Unique identifier for Driver. Referencing DriverID from Driver table
BookingDateTime	Date	Taxi booking date and time
BookingStartPoint	Varchar	Taxi booking start point
BookingDestinationPoint	Varchar	Taxi booking destination
RegularBookingID	Integer	Unique RegularBookingID referencing Regularbooking Table

Regular Booking Table: It holds information about regular booking made by client. It is kind of extension/reference table to Booking table and holds information about client, type of regular booking, regular booking start period, regular booking end period, reguarl booking starting point and regular booking destination point.

ColumnName	DataType	Column Description
RegularBookingID	Integer	Unique identifier for regular booking
ClientID	Integer	Unique identifier for client. Referencing ClientID from Client table
RegularBookingType	Varchar	Valid values are daily bookings, once weekly bookings
RegularBookingPeriodStartDate	Date	Regular booking tenure start date

RegularBookingPeriodEndDate	Date	Regular booking tenure end date
BookingStartPoint	Varchar	Regular booking start point
BookingDestinationPoint	Varchar	Regular booking termination point
LastArrivalDatetime	Date	Datetime for last arrival of car

Payment table: It holds information about payment amount and payment date.

ColumnName	DataType	Column Description
PaymentID	Integer	Unique Identifier for payment
BookingID	Integer	Unique identifier for booking. Referencing Booking id from booking table.
Amount	Decimal	Payment amount for booking
PaymentDate	Date	Date of payment

Revenue Table: It holds information about Revenue made by driver and Taxi Company. It has association with Driver table using DriverID.

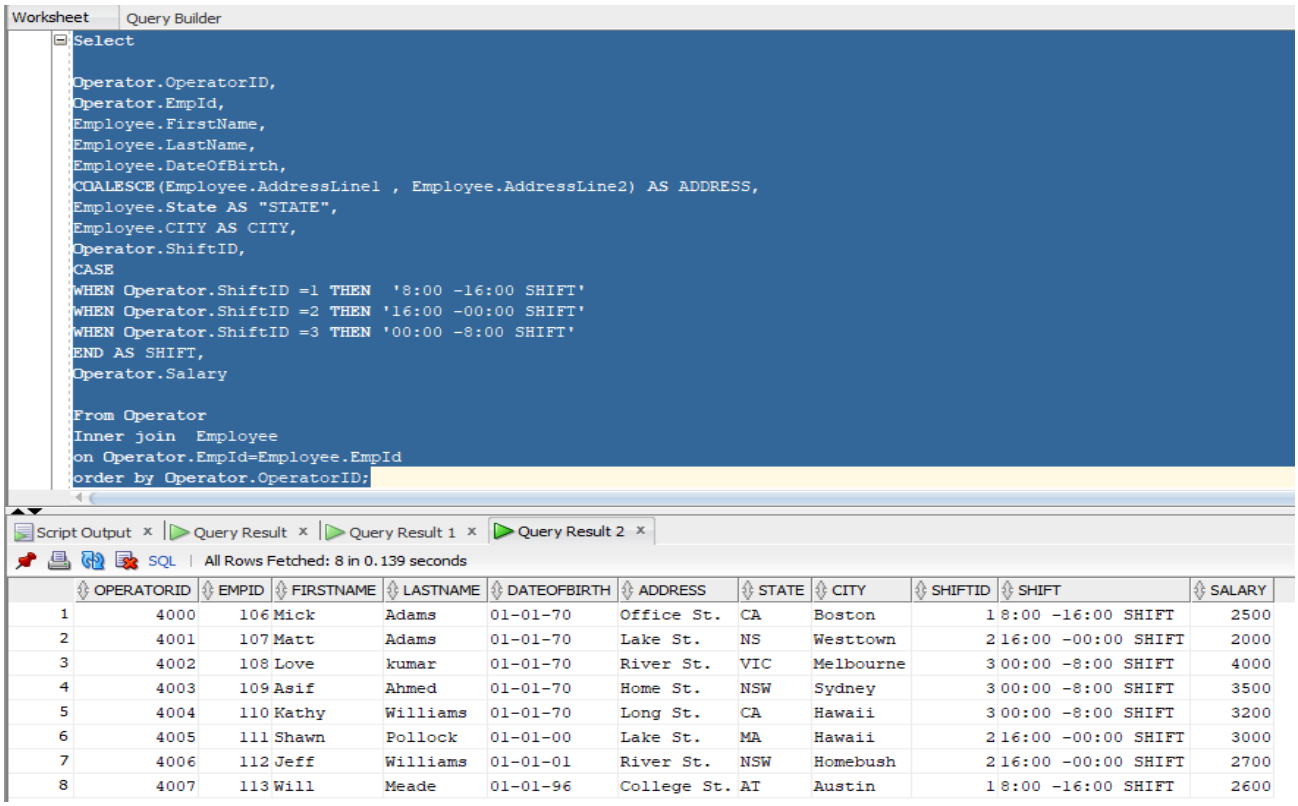
ColumnName	DataType	Column Description
RevenueID	Integer	Unique identifier for Revenue
DriverId	Integer	Unique identifier for Driver. Referencing Driver table
DriverWage	Decimal	Amount earned by Driver
PaymentToCompany	Decimal	Amount Paid to Company
StartDate	Date	Start Date for revenue calculation
EndDate	Date	End Date for revenue Calculation

SQL queries:

Query Statement 1:

SQL query to find operator, their name, their demographic, salary and shift timings

```
Select

Operator.OperatorID,
Operator.EmpId,
Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
COALESCE(Employee.AddressLine1 , Employee.AddressLine2) AS ADDRESS,
Employee.State AS "STATE",
Employee.CITY AS CITY,
Operator.ShiftID,
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END AS SHIFT,
Operator.Salary

From Operator
Inner join  Employee
on Operator.EmpId=Employee.EmpId
order by Operator.OperatorID;
```



Worksheet | Query Builder

```
Select

Operator.OperatorID,
Operator.EmpId,
Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
COALESCE(Employee.AddressLine1 , Employee.AddressLine2) AS ADDRESS,
Employee.State AS "STATE",
Employee.CITY AS CITY,
Operator.ShiftID,
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END AS SHIFT,
Operator.Salary

From Operator
Inner join  Employee
on Operator.EmpId=Employee.EmpId
order by Operator.OperatorID;
```

Script Output | Query Result | Query Result 1 | Query Result 2

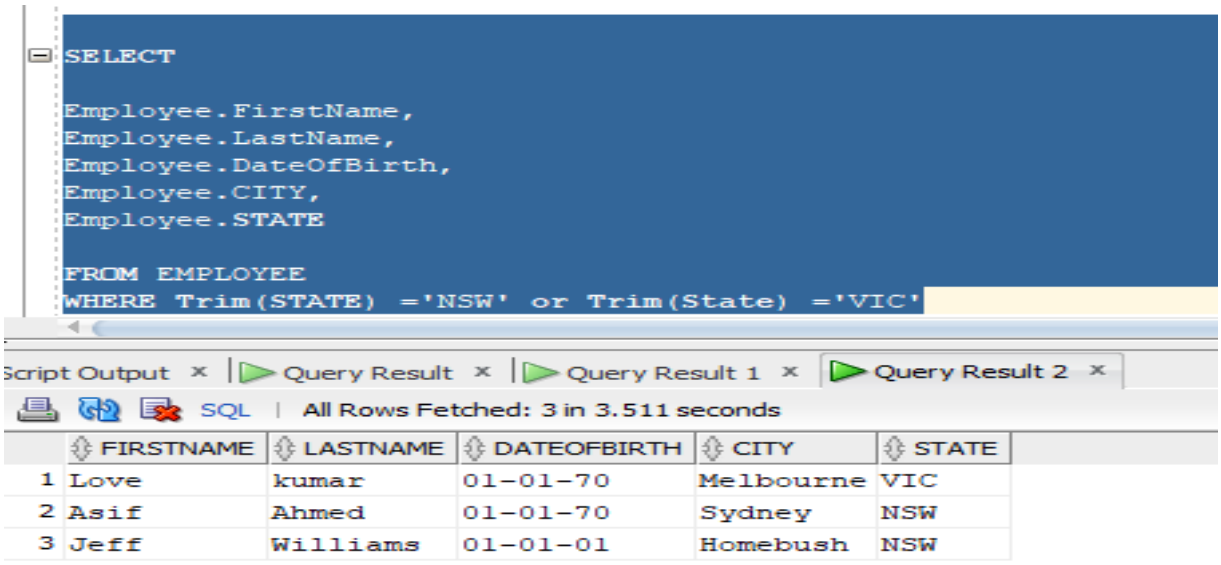
SQL | All Rows Fetched: 8 in 0.139 seconds

	OPERATORID	EMPID	FIRSTNAME	LASTNAME	DATEOFBIRTH	ADDRESS	STATE	CITY	SHIFTID	SHIFT	SALARY
1	4000	106	Mick	Adams	01-01-70	Office St.	CA	Boston	1	8:00 -16:00 SHIFT	2500
2	4001	107	Matt	Adams	01-01-70	Lake St.	NS	Westtown	2	16:00 -00:00 SHIFT	2000
3	4002	108	Love	kumar	01-01-70	River St.	VIC	Melbourne	3	00:00 -8:00 SHIFT	4000
4	4003	109	Asif	Ahmed	01-01-70	Home St.	NSW	Sydney	3	00:00 -8:00 SHIFT	3500
5	4004	110	Kathy	Williams	01-01-70	Long St.	CA	Hawaii	3	00:00 -8:00 SHIFT	3200
6	4005	111	Shawn	Pollock	01-01-00	Lake St.	MA	Hawaii	2	16:00 -00:00 SHIFT	3000
7	4006	112	Jeff	Williams	01-01-01	River St.	NSW	Homebush	2	16:00 -00:00 SHIFT	2700
8	4007	113	Will	Meade	01-01-96	College St.	AT	Austin	1	8:00 -16:00 SHIFT	2600

Query Statement 2:

SQL query to find employee who are from NSW OR VIC state.

```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE

FROM EMPLOYEE
WHERE Trim(STATE) ='NSW' or Trim(State) ='VIC';
```



```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE

FROM EMPLOYEE
WHERE Trim(STATE) ='NSW' or Trim(State) ='VIC'
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 3.511 seconds

	FIRSTNAME	LASTNAME	DATEOFBIRTH	CITY	STATE
1	Love	kumar	01-01-70	Melbourne	VIC
2	Asif	Ahmed	01-01-70	Sydney	NSW
3	Jeff	Williams	01-01-01	Homebush	NSW

Query Statement 3:

SQL query to find employee who are driver and have age >30

```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE,
EMPLOYEE.DateOfBirth,
ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0) AS AGE

FROM EMPLOYEE
INNER JOIN DRIVER
ON EMPLOYEE.EMPID=DRIVER.EMPID

WHERE ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0)  > 30;
```

```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE,
EMPLOYEE.DateOfBirth,
ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0) AS AGE



FROM EMPLOYEE
INNER JOIN DRIVER
ON EMPLOYEE.EMPID=DRIVER.EMPID

WHERE ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0)   > 30;
```

ipt Output × | Query Result × | Query Result 1 × | Query Result 2 ×

SQL | All Rows Fetched: 5 in 7.855 seconds

	FIRSTNAME	LASTNAME	DATEOFBIRTH	CITY	STATE	DATEOFBIRTH_1	AGE
1	James	Adams	01-01-70	Boston	MA	01-01-70	51
2	July	Vargees	01-05-76	Qunicy	NY	01-05-76	45
3	John	Adams	01-01-70	Jersey	AZ	01-01-70	51
4	Nanveen	Sharma	01-01-70	Macquire	CA	01-01-70	51
5	Ami	Adams	01-01-70	Hometown	MN	01-01-70	51

Query Statement 4:

SQL query to find employee who are driver and firstname starting with letter ‘J’

```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE,
EMPLOYEE.DateOfBirth,
ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0) AS AGE

FROM EMPLOYEE
INNER JOIN DRIVER
ON EMPLOYEE.EMPID=DRIVER.EMPID

where Employee.FirstName like 'J%';
```

```
SELECT

Employee.FirstName,
Employee.LastName,
Employee.DateOfBirth,
Employee.CITY,
Employee.STATE,
EMPLOYEE.DateOfBirth,
ROUND((SYSDATE - EMPLOYEE.DateOfBirth)/365.25,0) AS AGE



FROM EMPLOYEE
INNER JOIN DRIVER
ON EMPLOYEE.EMPID=DRIVER.EMPID

where Employee.FirstName like 'J%'
```

ript Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 2.889 seconds

	FIRSTNAME	LASTNAME	DATEOFBIRTH	CITY	STATE	DATEOFBIRTH_1	AGE
1	James	Adams	01-01-70	Boston	MA	01-01-70	51
2	July	Vargees	01-05-76	Qunicy	NY	01-05-76	45
3	John	Adams	01-01-70	Jersey	AZ	01-01-70	51

Query Statement 5:

SQL query to find car records where CarStatus is 'awaiting repair'

```
select  *

from Car
inner join CarStatus

on car.CarStatusID=CarStatus.CarStatusID
and CarStatus.CarStatusDescription='awaiting repair'
order by CarID;
```

```
select  *

from Car
inner join CarStatus

on car.CarStatusID=CarStatus.CarStatusID
and CarStatus.CarStatusDescription='awaiting repair'
order by CarID;
```

Script Output | Query Result | Query Result 1 | Query Result 2

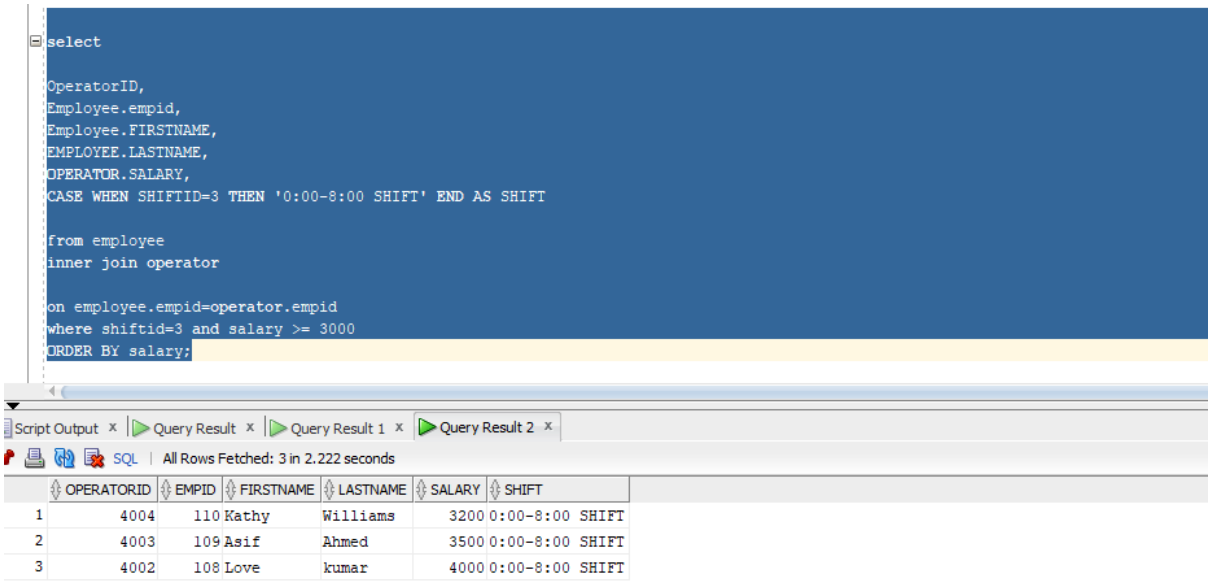
SQL | All Rows Fetched: 2 in 0.807 seconds

	CARID	REGISTRATIONNUMBER	AGE	DATEOFLASTMOTTEST	CARSTATUSID	CAROWNERID	CARSTATUSID_1	CARSTATUSDESCRIPTION
1	50004	uvw-777	1	20-08-19	203	3001	203	awaiting repair
2	50005	CEF-555	1	01-04-19	203	3004	203	awaiting repair

Query Statement 6:

SQL query to find operator who work in 00:00-8:00 shift and have salary not less than 3000.

```
select

OperatorID,
Employee.empid,
Employee.FIRSTNAME,
EMPLOYEE.LASTNAME,
OPERATOR.SALARY,
CASE WHEN SHIFTID=3 THEN '0:00-8:00 SHIFT' END AS SHIFT

from employee
inner join operator

on employee.empid=operator.empid
where shiftid=3 and salary >= 3000
ORDER BY salary;
```



```
select

OperatorID,
Employee.empid,
Employee.FIRSTNAME,
EMPLOYEE.LASTNAME,
OPERATOR.SALARY,
CASE WHEN SHIFTID=3 THEN '0:00-8:00 SHIFT' END AS SHIFT

from employee
inner join operator

on employee.empid=operator.empid
where shiftid=3 and salary >= 3000
ORDER BY salary;
```

Script Output | Query Result | Query Result 1 | Query Result 2

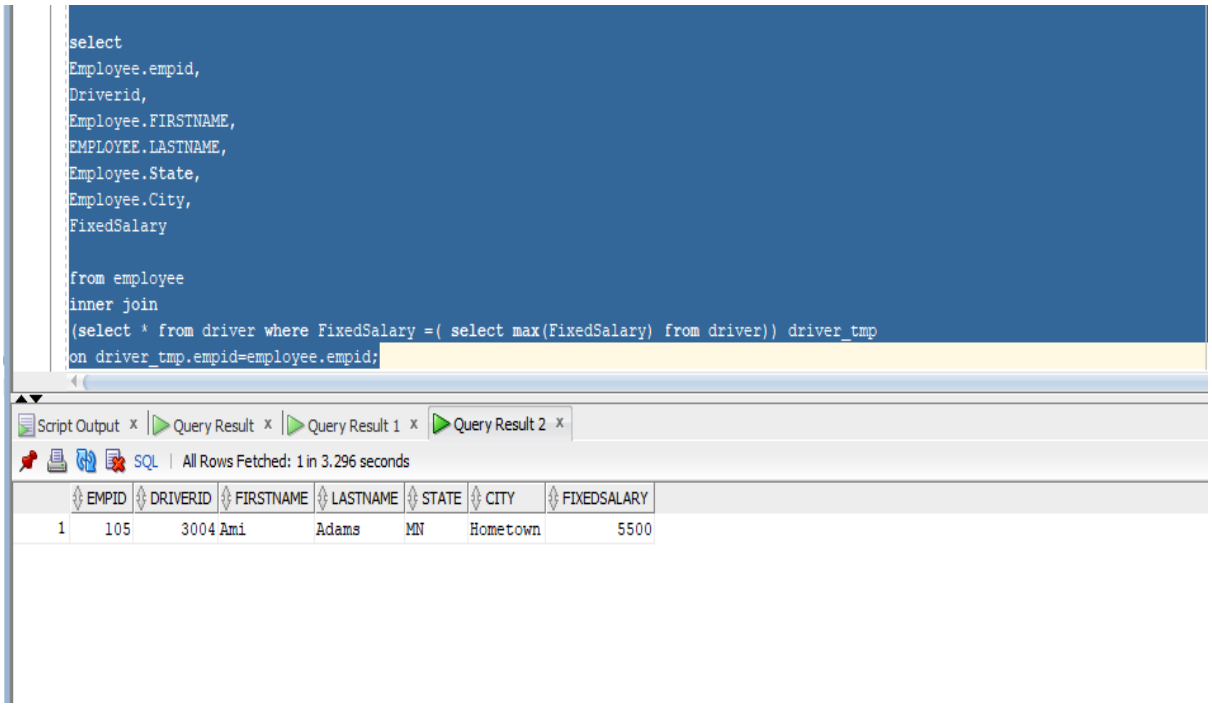
SQL | All Rows Fetched: 3 in 2.222 seconds

	OPERATORID	EMPID	FIRSTNAME	LASTNAME	SALARY	SHIFT
1	4004	110	Kathy	Williams	3200	0:00-8:00 SHIFT
2	4003	109	Asif	Ahmed	3500	0:00-8:00 SHIFT
3	4002	108	Love	kumar	4000	0:00-8:00 SHIFT

Query Statement 7:

SQL query to find driver first name , last name , state , city who has maximum salary.

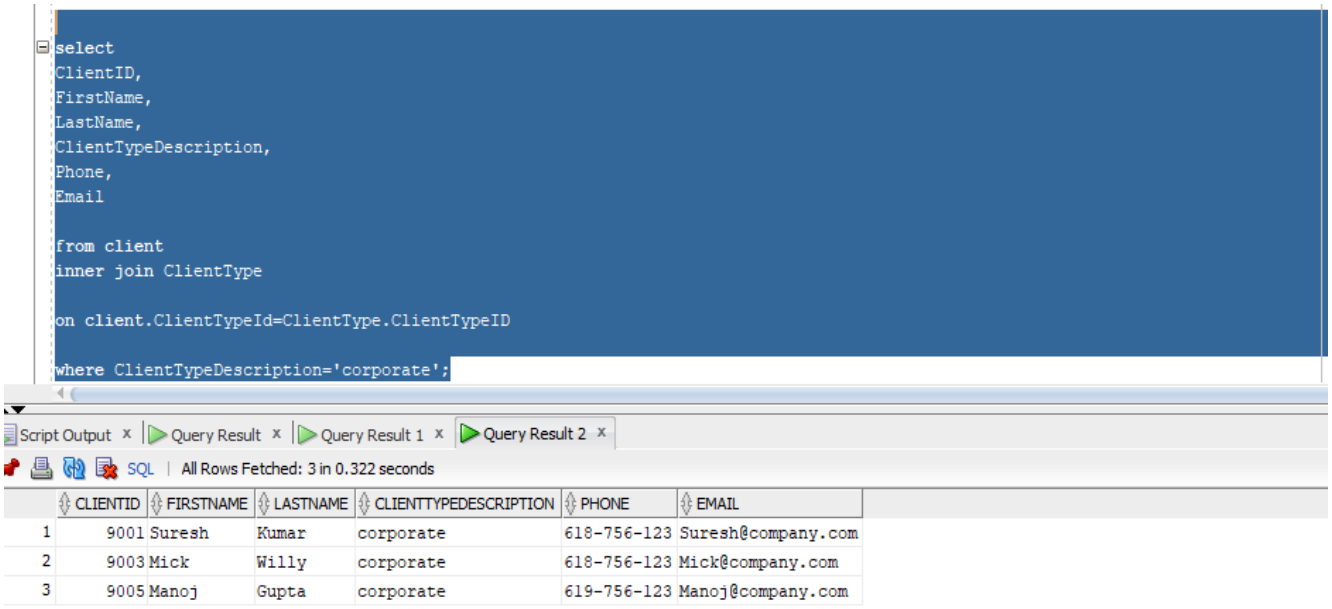
```
select
Employee.empid,
Driverid,
Employee.FIRSTNAME,
EMPLOYEE.LASTNAME,
Employee.State,
Employee.City,
FixedSalary

from employee
inner join
(select * from driver where FixedSalary =( select max(FixedSalary) from driver))
driver_tmp
on driver_tmp.empid=employee.empid;
```



Query Statement 8:

SQL query to find all corporate clients.

```
select
ClientID,
FirstName,
LastName,
ClientTypeDescription,
Phone,
Email

from client
inner join ClientType

on client.ClientTypeId=ClientType.ClientTypeID

where ClientTypeDescription='corporate';
```



```
select
ClientID,
FirstName,
LastName,
ClientTypeDescription,
Phone,
Email

from client
inner join ClientType

on client.ClientTypeId=ClientType.ClientTypeID

where ClientTypeDescription='corporate';
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.322 seconds

	CLIENTID	FIRSTNAME	LASTNAME	CLIENTTYPEDESCRIPTION	PHONE	EMAIL
1	9001	Suresh	Kumar	corporate	618-756-123	Suresh@company.com
2	9003	Mick	Willy	corporate	618-756-123	Mick@company.com
3	9005	Manoj	Gupta	corporate	619-756-123	Manoj@company.com

Query Statement 9:

SQL query to find all private clients and their bookings.

```
select
client.ClientID,
FirstName,
LastName,
ClientTypeDescription,
Phone,
Email,
BookingID,
BookingDateTime,
BookingStartPoint,
BookingDestinationPoint
```

```
from client
inner join ClientType

on client.ClientTypeId=ClientType.ClientTypeID

inner join booking
on booking.ClientID=client.ClientID

where ClientTypeDescription='private';
```

```
select
client.ClientID,
FirstName,
LastName,
ClientTypeDescription,
Phone,
Email,
BookingID,
BookingDateTime,
BookingStartPoint,
BookingDestinationPoint

from client
inner join ClientType

on client.ClientTypeId=ClientType.ClientTypeID

inner join booking
on booking.ClientID=client.ClientID

where ClientTypeDescription='private';
```

ipt Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 2 in 4.54 seconds

	CLIENTID	FIRSTNAME	LASTNAME	CLIENTTYPEDESCRIPTION	PHONE	EMAIL	BOOKINGID	BOOKINGDATETIME	BOOKINGSTARTPOINT	BOOKINGDESTINATIONPOINT
1	9002	Narsh	Kapoor	private	428-756-123	Narsh@xyz.com	502	01-10-20	Location19	Location21
2	9004	Rajat	Kumar	private	638-756-658	Rajat@gmail.com	504	05-07-20	Location1	Location3

Query Statement 10:

SQL query to find driver details whose salary is greater than average salary of all drivers.

```
select
employee.empid ,
firstname,
lastname,
Dateofbirth,
State,
city,
FixedSalary


from employee
inner join
(select * from driver where fixedsalary >= ( select avg(fixedsalary)as avg_salary from
driver) )driver_avg
on employee.empid=driver_avg.empid;
```

```
select
employee.empid ,
firstname,
lastname,
Dateofbirth,
State,
city,
FixedSalary


from employee
inner join
(select * from driver where fixedsalary >= ( select avg(fixedsalary)as avg_salary from driver) )driver_avg
on employee.empid=driver_avg.empid
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 2 in 2.648 seconds

	EMPID	FIRSTNAME	LASTNAME	DATEOFBIRTH	STATE	CITY	FIXEDSALARY
1	101	James	Adams	01-01-70	MA	Boston	5000
2	105	Ami	Adams	01-01-70	MN	Hometown	5500

Query Statement 11:

SQL query to find number of operators working in different shift timings

```
select
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END AS SHIFT, count(1) As "No_OF_Operators"

from Operator

Group by
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END
order by 1;
```

```
select
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END AS SHIFT, count(1) As "No_OF_Operators"

from Operator

Group by
CASE
WHEN Operator.ShiftID =1 THEN  '8:00 -16:00 SHIFT'
WHEN Operator.ShiftID =2 THEN '16:00 -00:00 SHIFT'
WHEN Operator.ShiftID =3 THEN '00:00 -8:00 SHIFT'
END
order by 1
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 5.07 seconds

	SHIFT	No_OF_Operators
1	00:00 -8:00 SHIFT	3
2	16:00 -00:00 SHIFT	3
3	8:00 -16:00 SHIFT	2

Query Statement 12:

SQL query to find all booking done by client with first name =' Mick' or lastname='kumar'

```
select
Booking.*,
FIRSTNAME,
LASTNAME

from booking
inner join client
on booking.clientid=client.clientid
where client.firstname='Mick' or ( upper(LastName)='KUMAR');
```

```
select
Booking.*,
FIRSTNAME,
LASTNAME

from booking
inner join client
on booking.clientid=client.clientid
where client.firstname='Mick' or ( upper(LastName)='KUMAR');
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 4.378 seconds

	BOOKINGID	CLIENTID	OPERATORID	DRIVERID	BOOKINGDATETIME	BOOKINGSTARTPOINT	BOOKINGDESTINATIONPOINT	REGULARBOOKINGID	FIRSTNAME	LASTNAME
1	501	9001	4000	3002	01-09-20	Location3	Location2	7001	Suresh	Kumar
2	503	9003	4003	3000	01-11-20	Location1	Location5	7003	Mick	Willy
3	504	9004	4004	3001	05-07-20	Location1	Location3	7004	Rajat	Kumar

Database Triggers:

Trigger 1: Following is before insert trigger on Driver table.

```
CREATE OR REPLACE TRIGGER trg_before_driver_insr
BEFORE INSERT
  on driver
  FOR EACH ROW

DECLARE
rec_cnt number;

BEGIN

select count(1) into rec_cnt
from driver  inner join employee
on  :new.EmpID =employee.EmpID;

    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'Refertial integrity violation between driver and
employee table.');
    END IF;

END;
```

Usage: This trigger is to maintain referential integrity between driver and employee tables. This trigger will send error message if we try to insert data into Driver table but that person does not exist in employee table.

Trigger 2: Trigger to be execute before insertion into car table

```
CREATE OR REPLACE TRIGGER trg_before_car_insr
BEFORE INSERT
  on car
  FOR EACH ROW

DECLARE
rec_cnt number;

BEGIN

select count(1) into rec_cnt
from car  inner join CarStatus
on :new.CarStatusId =CarStatus.CarStatusID;

    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'Refertial integrity violation between Car and
CarStatus table.');
    END IF;

END;
```

Usage: This trigger is to maintain referential integrity between car and carstatus tables. This trigger will send error message if we try to insert data into car table but that carstatus on new record does not exist in car table.

Trigger 3: Before delete trigger

```
CREATE OR REPLACE TRIGGER trg_before_car_del
BEFORE delete
  on carstatus
  FOR EACH ROW

DECLARE
rec_cnt number;

BEGIN

select count(1) into rec_cnt
from car  inner join CarStatus
on :new.CarStatusId =CarStatus.CarStatusID;
```

```
    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'Delete record from car table before deleting
from carstatus table.');
    END IF;

END;
```

Usage: This trigger is to avoid accidental deletion of carstatus record from carstatus table which is getting referenced in car table.

Trigger 4: After insert trigger

```
CREATE OR REPLACE TRIGGER Employee_after_insert
AFTER INSERT
   ON employee
   FOR EACH ROW

DECLARE
rec_cnt number;

BEGIN

   -- Find username of person performing the INSERT into the table
select count(1) into rec_cnt
from employee where state =:new.state;

    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'This new employee belongs to new state.');
    END IF;


END;
```

Usage: This trigger is after insert trigger. It will display a message after inserting a new record in employee table if new employee belongs to a state which is not existing in employee table.

Trigger 5: Trigger to be execute before insertion into Client table

```
CREATE OR REPLACE TRIGGER trg_before_client_insr
BEFORE INSERT
  on client
  FOR EACH ROW

DECLARE
rec_cnt number;
```

```
BEGIN

select count(1) into rec_cnt
from client  inner join ClientType
on :new.ClientTypeID= ClientType. ClientTypeID;

    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'Refertial integrity violation between Client and
ClientTypetable.');
    END IF;

END;
```

Usage: This trigger is to maintain referential integrity between client and ClientType tables. This trigger will send error message if we try to insert data into client table but that ClientType on new record does not exist in ClientType table.

Trigger 6:

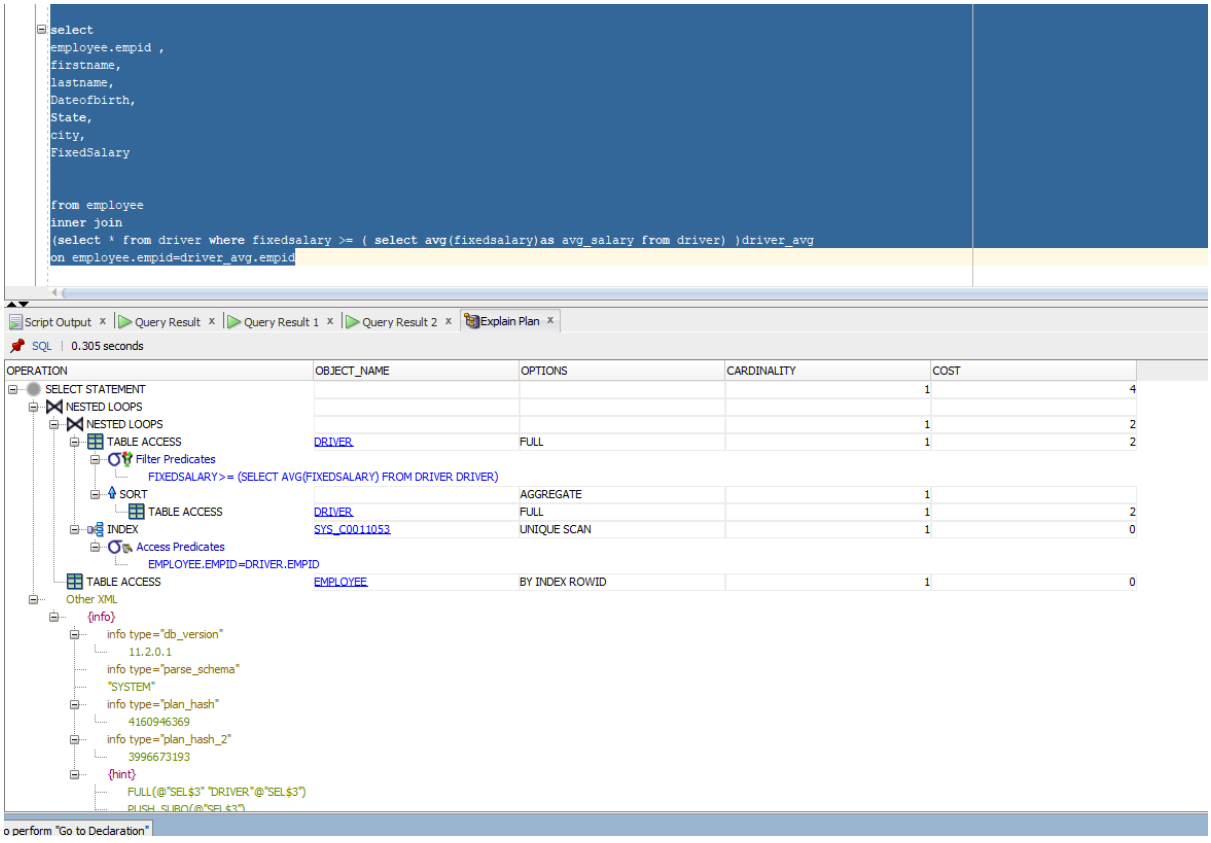
```
CREATE OR REPLACE TRIGGER trg_before_operator_insr
BEFORE INSERT
  on operator
  FOR EACH ROW

DECLARE
rec_cnt number;

BEGIN

select count(1) into rec_cnt
from operator  inner join shift
on :new.Shiftid= shift.shiftID;

    IF (rec_cnt =0) THEN
      RAISE_APPLICATION_ERROR(-20000,'Shift on operator record does not exist in shift
table.');
    END IF;

END;
```

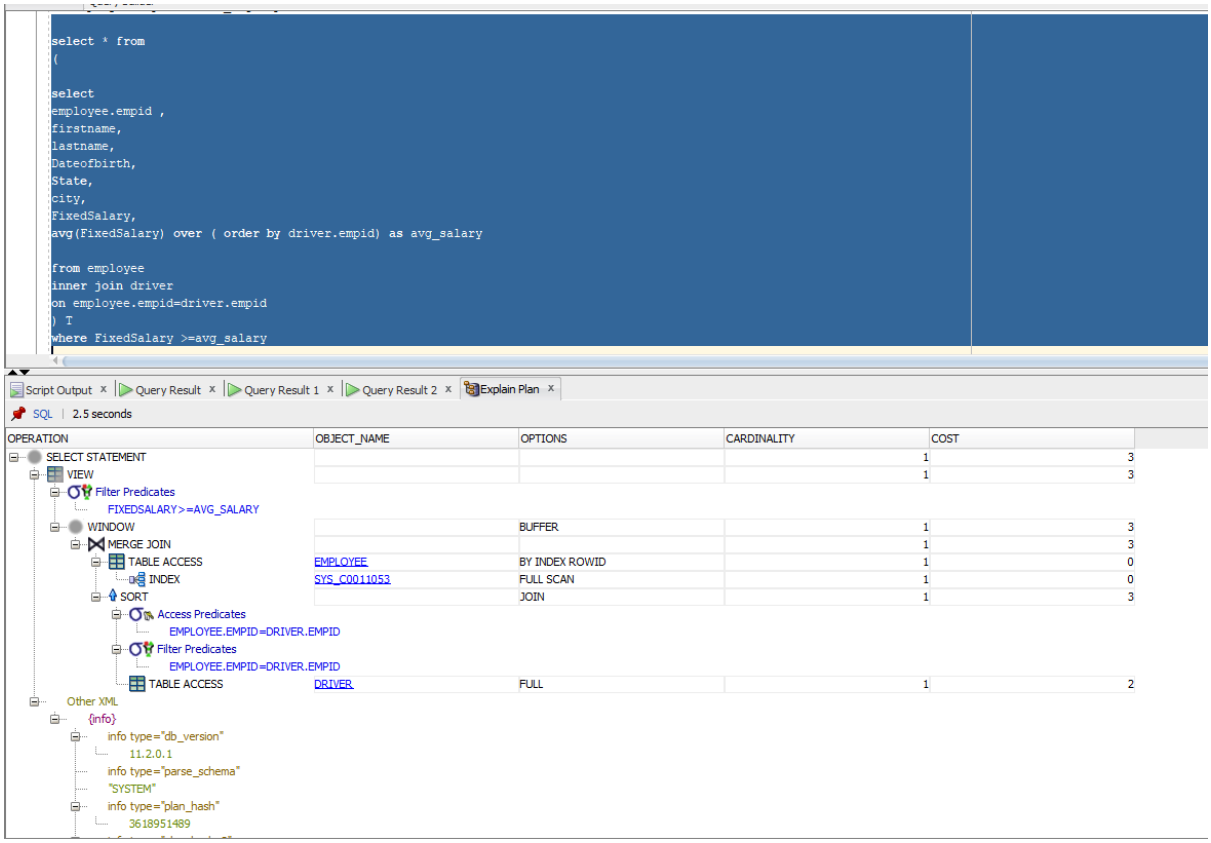
Usage: This trigger is to maintain referential integrity between Operator table and Shift table. This trigger will send error message if we try to insert data into Operator table with shiftid that does not exist in shift table.

Performance tuning:

Query performance can be optimized in multiple ways depending on requirement. It can be done by re-writing query and checking the explain plan of the query. It can be done by using appropriate join condition between tables in query. It can be done by using proper indexes so that table scan can be minimized to improve performance. Query performance can also be achieved by using appropriate filter condition as earlies as possible in query.

Experiment 1: In this experiment, query re-write has been used to improve the performance of the query. We can observe from explain plan of the query that execution time has been reduced.





Both queries are giving same result but the execution time in query1 is 6 times less than the execution time in query 2. Query 1 is using windows function to figure out average salary for all the driver , however query 2 is using simple group by function. We can observer better performance improvement with more records in driver table. Query 1 execution time .3 seconds and execution time for query 2 is 2.5 seconds

Experiment 2: In this experiment, tables are joined using inner and full outer joins with filter conditions.

```
select *

from employee
inner join operator
on employee.empid =operator.empid
inner join shift
on shift.shiftid = operator.shiftid
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Explain Plan

SQL | 0.032 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
NESTED LOOPS				
NESTED LOOPS			1	2
NESTED LOOPS			1	2
TABLE ACCESS	OPERATOR	FULL	1	2
TABLE ACCESS	SHIFT	BY INDEX ROWID	1	0
INDEX	SYS_C0011054	UNIQUE SCAN	1	0
Access Predicates				
SHIFT.SHIFTID=OPERATOR.SHIFTID				
INDEX	SYS_C0011053	UNIQUE SCAN	1	0
Access Predicates				
EMPLOYEE.EMPID=OPERATOR.EMPID				
TABLE ACCESS	EMPLOYEE	BY INDEX ROWID	1	0
Other XML				
{info}				
info type="db_version"				
11.2.0.1				
info type="parse_schema"				
"SYSTEM"				
info type="plan_hash"				
2152183562				
info type="plan_hash_2"				

perform "Go to Declaration"

```
select *

from employee
full outer join operator
on employee.empid =operator.empid
full outer join shift
on trim(cast(shift.shiftid as varchar(10))) = trim(cast(operator.shiftid as varchar(10)))

where operator.empid is not null and shift.shiftid is not null
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Explain Plan

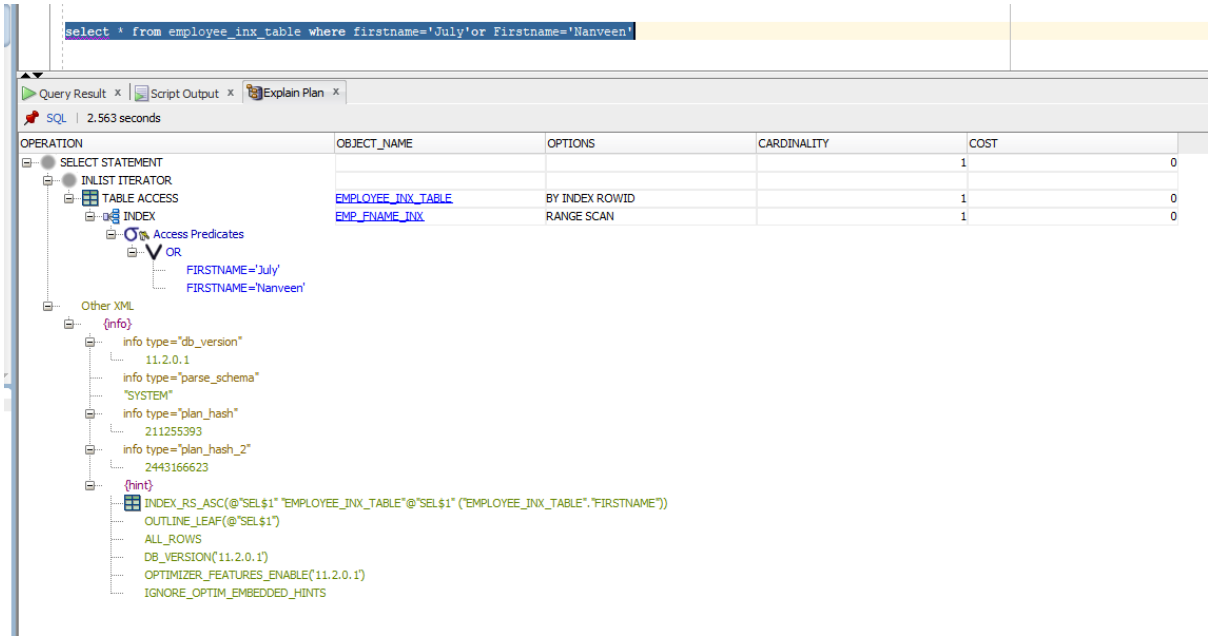
SQL | 6.348 seconds

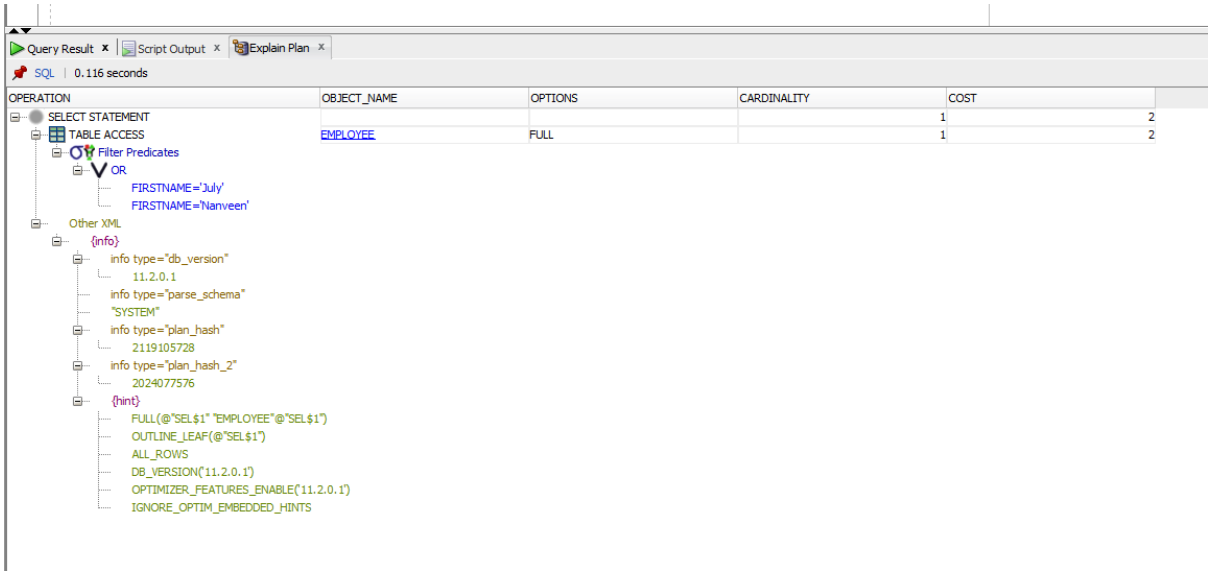
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
VIEW	SYS.VW_FOJ_0		1	7
Filter Predicates				
AND				
OPERATOR.EMPID IS NOT NULL				
SHIFT.SHIFTID IS NOT NULL				
HASH JOIN		FULL OUTER	1	7
Access Predicates				
TRIM(CAST(SHIFT.SHIFTID AS varchar(10)))=TRIM(CAST(OPERATOR.SHIFTID AS varchar(10)))				
TABLE ACCESS	SHIFT	FULL		2
VIEW	SYS.VW_FOJ_1		1	5
HASH JOIN		FULL OUTER	1	5
Access Predicates				
EMPLOYEE.EMPID=OPERATOR.EMPID				
TABLE ACCESS	EMPLOYEE	FULL		2
TABLE ACCESS	OPERATOR	FULL	1	2
Other XML				
{info}				
info type="db_version"				
11.2.0.1				
info type="parse_schema"				
"SYSTEM"				

to perform "Go to Declaration"

Both queries are giving same result set but the execution plan is expensive in case of full outer join with filter condition in comparison to inner join. As per explain plan for inner join execution time is just 0.032 second however it is much higher in case of second execution with full outer join with filter conditions. It is 6.34 seconds. This experiment proves that appropriate joins have impact on query performance and execution time.

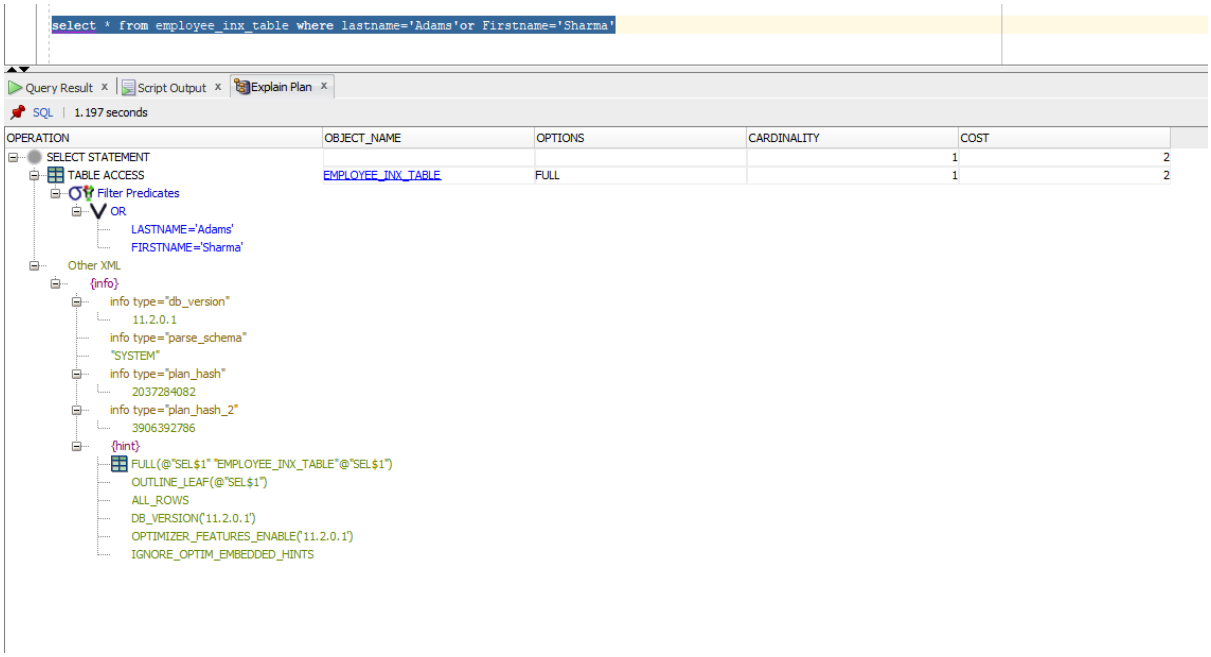
Experiment 3: In this experiment, same query is executed on table with index and without index.





It can be observed that in case of indexes table query performance is better than table without index. When we create index and use it , it helps reading operation by not performing full table scan , which is the case when there is no index on the table. With help of index, query optimizer will read the specific record instead of scanning the entire table for result. Hence having indexes on table provide better performance for read query.

Experiment 4: In this experiment, having indexes on table does not automatically improve the query performance. We have to utilize them in query.



This experiment shows that although we have index on the table but since query is not using indexed column in the search condition, index is not getting utilized and optimizer is performing full table scan instead of range scan or specific rows from table.

MongoDB design:

MongoDB is a general purpose NoSQL database, offering the best of both RDBMS and NoSQL worlds. In comparison to other NOSQL databases, which are designed for specific purpose, MongoDB is more generic in nature and can server various kind of loads and applications. MongoDB has flexible schema design. Data is stored in terms of database, collection, documents and fields. It is closely related to terms in relational database like Oracle. Collection in MongoDB is equivalent to table in RDBMS. Similarly, a row or an instance in RDBMS is equivalent to collection in MongoDB. Attribute or column in RDBMS is represented by fields in MongoDB. Furthermore, a join in RDBMS is represented by embedded document in MongoDB. Also, MongoDB schema is not rigid but it is flexible and dynamic. Data in MongoDB is stored as JSON object and in terms of Key value pair.

Some of entities in Taxi data management project can be implemented using MongoDB. Below are examples of some of data elements on how they will be done using MongoDB NOSQL databse.

First we need to create a database in MongoDB. Say TaxiCompany

Command to create database in MongoDB is

use TaxiCompany

Then we need to create required collections in this database basis our data model requirement. As per requirement , we need to create collection for Employee, Shift, CarStatus, ClientType, Driver, Operator, Car, Client, Booking, regularBookings and revenue. For this exercise, we will be demonstrating collection creation for Driver and Operator.

Driver collection definition

Column Name	DataType	Description
firstName	String	
lastName	String	
dateOfBirth	Date	
address	Object	This Object field contains fields: country (String), city (String), street (String) and flat (Int).
contactDetails	Object	This Object field contains fields: telephone (String because of different possible formats) and email (String).
dateOfEmployement	Date	
dateOfemploymentEnding	Date	Optional field. When there is no dismiss date yet, this means that the driver is currently working in the company.
salary	Integer	One of these 2 fields is optional. This mean that driver use the certain scheme for salary and the existence of exact field reference to a certain type of salary: fixed monthly salary or percentage of receipt basis.
percentageOfReceipt	Integer	
shift	Object	This Object field shows which shift is used for a driver. Contains fields: startTime (String) and endTime (String). For correct comparison of those strings in queries the shift endTime should have two values for noon: 00:00 and 24:00. Therefore, the shift variants are the following: 00:00-08:00, 08:00-16:00, 16:00-24:00.
cars	Array	This Array might contain several Objects inside, because taxi driver can own several cars (assumption). Each object contains: registrationNumber (String) - unique field for each car, age (Int), dateOfLastMOTTest (Date) and carStatus (String , possible values: roadworthy, in for service, awaiting repair, written off).

MongoDB code example for insert data into Driver collection

```
db .drivers .insert({
firstName : 'Andrey' ,
lastName : 'Newman' ,
dateOfBirth : new Date ('1987-12-12'),
 address :  {
       country : 'UK' ,
       city : 'London' ,
       street : 'Oxford street' ,
       flat : 12
       },
    contactDetails :  {
    telephone : '+44 020 7033 3920' ,
    email : 'andrey.newman@gmail.com'
  },
dateOfEmployment : new Date ('2008-01-12'),
percentageOfReceipt : 40,

    shift :  {
       startTime : '08:00' ,
       endTime : '16:00'
       },
cars :  [
        {    registrationNumber  : '123AJOO22MLG' ,             age  :  3 ,
dateOfLastMOTTest : new Date ('2017-07-11'), carStatus  : 'roadworthy }
         ]
});
```

Operator collection definition

Column Name	DataType	Description
firstName	String	
lastName	String	
dateOfBirth	Date	
address	Object	This Object field contains fields: country (String), city (String), street (String) and flat (Int).
contactDetails	Object	This Object field contains fields: telephone (String because of different possible formats) and email (String).
dateOfEmployement	date	
dateOfemploymentEnding	Date	Optional field. When there is no dismiss date yet, this means that the driver is currently working in the company.
salary	Integer	

shift	Object	This Object field shows which shift is used for a driver. Contains fields: startTime (String) and endTime (String). For correct comparison of those strings in queries the shift endTime should have two values for noon: 00:00 and 24:00. Therefore, the shift variants are the following: 00:00-08:00, 08:00-16:00, 16:00-24:00.

```
db .operators .insert(

{

firstName : 'Anny' ,

LastName : 'Winehouse' ,

dateOfBirth : new Date ('1997-01-06'),

address :  {

country : 'UK' ,

city : 'London' ,

street : 'Regent street' ,

flat :  2     }

contactDetails :  {

telephone : '+44 020 8659 3794' ,

email : 'anny1997@gmail.com'

},

dateOfEmployment : new Date ('2016-05-22'),

salary : 20000 ,

shift :  {

startTime : '08:00' ,

endTime : '16:00'

}

}
```