1222·2022 800 ANNI

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Foundations of Databases A.Y. 2021-2022
Homework 3 – Physical Design

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

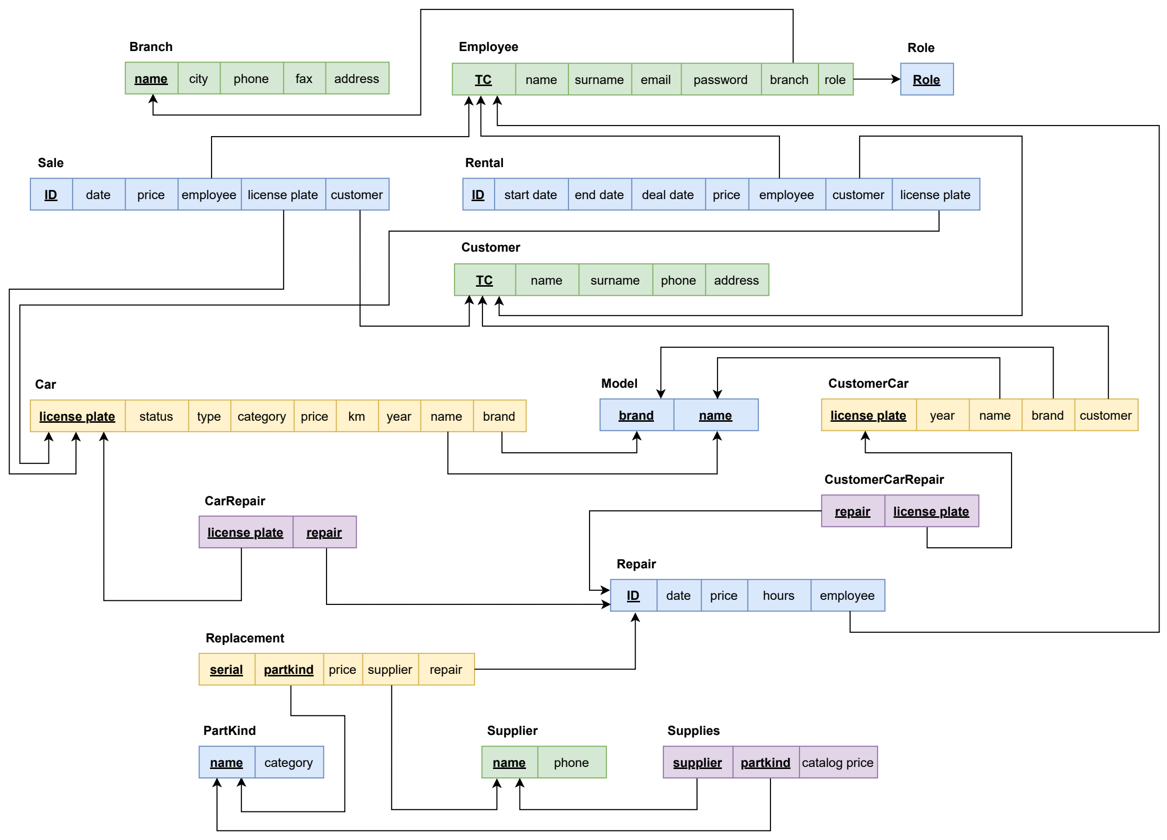
Deadline: December 17, 2021

GROUP Don'tBlameMySchema	**PROJECT** Car Dealer	
Last name Abbaspour	First Name Anahita	Student number 2005495
Last name Algun	First Name Mustafa	Student number 2049537
Last name Arslan	First Name Selen	Student number 2004968
Last name Baron	First Name Alex	Student number 2021133
Last name Cattai	First Name Christian	Student number 2021234
Last name Freskina	First Name Fatjon	Student number 2056098
Last name Leka	First Name Enrico	Student number 2019178
Last name Milanese	First Name Marco	Student number 2010656
Last name Scattolin	First Name Giulio	Student number 2055035

Variations to the Relational Schema

The relational schema was modified as follows:

- The entities NewCar and UsedCar were merged in a new Car entity.
- The schema now models some extra date and price attributes in Rental and Repair.
- The Replacement now refers to an instance of a part and PartKind models the catalog of parts.
- A new entity Model was added to represent the exact model of a Car.
- Some more minor tweaks were applied.



Physical Schema

```
-- Database Creation
CREATE DATABASE dbmsdb OWNER postgres ENCODING = 'UTF8';

-- Connect to the new db
\c dbmsdb

-- CREATE NEW DOMAINS

CREATE DOMAIN emailaddress AS VARCHAR (254)
    CHECK(VALUE ~ '[A-Za-z0-9._%]+@[A-Za-z0 -9.]+[.][A-Za-z]+$');

CREATE DOMAIN licenseplate AS CHAR (7)
    CHECK(VALUE ~ '[A-Z]{2}[0-9]{3}[A-Z]{2}');

CREATE DOMAIN replacementserial AS CHAR (10)
    CHECK(VALUE ~ '[A-Z]{2}[0-9]{5}');

CREATE DOMAIN taxcode AS CHAR (16)
    CHECK(VALUE ~ '[A-Z]{6}[0-9]{2}[A-Z]{1}[0-9]{2}[A-Z]{1}[0-9]{3}[A-Z]{1}');

CREATE DOMAIN phonenumber AS CHAR (10)
    CHECK(VALUE ~ '[0-9]{10}');

CREATE DOMAIN faxnumber AS CHAR (10)
    CHECK(VALUE ~ '[0-9]{10}');

CREATE DOMAIN yearnumber as DECIMAL(4)
    CHECK(VALUE > 1900 AND VALUE <= 9999);

-- TYPES

CREATE TYPE status AS ENUM (
    'Available',
    'UnderMaintenance',
    'Rented'
);

CREATE TYPE partcategory AS ENUM(
    'Tree',
    'Shock absorber',
    'Axle',
    'Bar',
    'Biella',
    'Barbell',
    'Exchange',
    'Carter',
    'Chain',
    'Countershaft',
    'Converter',
    'Belt',
    'Differential',
    'Fillet steak',
    'Fork',
    'Brake',
    'Clutch',
    'Ring',
    'Joint',
    'Gasket',
    'Nut',
    'Crank',
    'Muffler',
    'Spring',
    'Hub',
    'Oil seal',
    'Pads',
    'Plier',
    'Piston',
    'Pump',
    'Bridge',
    'Adapter',
    'Wheel',
    'Exhaust',
    'Selector',
    'Brake booster',
    'Power steering',
    'Door lock',
    'Drive shaft',
    'Suspension',
    'Probe',
    'Sensor',
    'Transmission',
    'Turbocharger',
    'Mirror'
);

CREATE TYPE cartypes AS ENUM(
    'New',
    'Used'
);

CREATE TYPE carcategory AS ENUM(
    'ForSale',
    'ForRental',
    'ForSaleRental'
);


-- TABLE CREATION

CREATE TABLE Branch(
    name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL,
    phone PHONENUMBER NOT NULL,
    fax FAXNUMBER,
    address TEXT NOT NULL
);
```

```sql
CREATE TABLE Role(
    role VARCHAR(20) PRIMARY KEY
);

CREATE TABLE Supplier(
    name VARCHAR(50) PRIMARY KEY,
    phone PHONENUMBER NOT NULL
);

CREATE TABLE Employee(
    tc TAXCODE PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    email EMAILADDRESS NOT NULL,
    password CHAR(40) NOT NULL, -- Hash of the password.
    roleid VARCHAR(20) NOT NULL,
    branchid VARCHAR(50) NOT NULL,
    FOREIGN KEY(roleid) REFERENCES Role(role),
    FOREIGN KEY(branchid) REFERENCES Branch(name)
);

CREATE TABLE PartKind(
    name VARCHAR(50) PRIMARY KEY,
    category partcategory NOT NULL
);

CREATE TABLE Customer(
    tc TAXCODE PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    surname VARCHAR(20) NOT NULL,
    phone PHONENUMBER NOT NULL,
    address TEXT NOT NULL
);

CREATE TABLE Model(
    brand VARCHAR(20) NOT NULL,
    name VARCHAR(20) NOT NULL,
    PRIMARY KEY (brand, name)
);

CREATE TABLE Car(
    lp LICENSEPLATE PRIMARY KEY,
    status STATUS NOT NULL,
    type CARTYPES NOT NULL,
    category CARCATEGORY NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    km DECIMAL(10) NOT NULL,
    year YEARNUMBER NOT NULL,
    name VARCHAR(20) NOT NULL,
    brand VARCHAR(20) NOT NULL,
    FOREIGN KEY(name,brand) REFERENCES Model(name,brand)
);
```

```sql
CREATE TABLE CustomerCar(
    lp LICENSEPLATE PRIMARY KEY,
    year YEARNUMBER NOT NULL,
    name VARCHAR(20) NOT NULL,
    brand VARCHAR(20) NOT NULL,
    customer TAXCODE NOT NULL,
    FOREIGN KEY(name,brand) REFERENCES Model(name,brand),
    FOREIGN KEY(customer) REFERENCES Customer(tc)
);

CREATE TABLE Sale(
    id SERIAL PRIMARY KEY,
    date DATE NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    employee TAXCODE NOT NULL,
    lp LICENSEPLATE NOT NULL,
    customer TAXCODE NOT NULL,
    FOREIGN KEY(employee) REFERENCES Employee(tc),
    FOREIGN KEY(lp) REFERENCES Car(lp),
    FOREIGN KEY(customer) REFERENCES Customer(tc)
);

CREATE TABLE Rental(
    id SERIAL PRIMARY KEY,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    deal_date DATE NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    employee TAXCODE NOT NULL,
    customer TAXCODE NOT NULL,
    lp LICENSEPLATE NOT NULL,
    CHECK(start_date < end_date),
    CHECK(deal_date <= start_date),
    FOREIGN KEY(employee) REFERENCES Employee(tc),
    FOREIGN KEY(customer) REFERENCES Customer(tc),
    FOREIGN KEY(lp) REFERENCES Car(lp)
);

CREATE TABLE Repair(
    id SERIAL PRIMARY KEY,
    date DATE NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    hours DECIMAL(2) NOT NULL,
    employee TAXCODE NOT NULL,
    FOREIGN KEY(employee) REFERENCES Employee(tc)
);

CREATE TABLE CarRepair(
    lp LICENSEPLATE NOT NULL,
    repair SERIAL NOT NULL,
    FOREIGN KEY(lp) REFERENCES Car(lp),
    FOREIGN KEY(repair) REFERENCES Repair(id),
    PRIMARY KEY(lp, repair)
);
```

```
CREATE TABLE CustomerCarRepair(
    repair SERIAL NOT NULL,
    lp LICENSEPLATE NOT NULL,
    FOREIGN KEY(repair) REFERENCES Repair(id),
    FOREIGN KEY(lp) REFERENCES CustomerCar(lp),
    PRIMARY KEY(repair, lp)
);

CREATE TABLE Replacement(
    serial SERIAL NOT NULL,
    partkind VARCHAR(50) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    supplier VARCHAR(50) NOT NULL,
    repair SERIAL NOT NULL,
    FOREIGN KEY(partkind) REFERENCES PartKind(name),
    FOREIGN KEY(supplier) REFERENCES Supplier(name),
    FOREIGN KEY(repair) REFERENCES Repair(id),
    PRIMARY KEY(serial, partkind)
);

CREATE TABLE Supplies(
    supplier VARCHAR(50) NOT NULL,
    partkind VARCHAR(50) NOT NULL,
    catalog_price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY(supplier) REFERENCES Supplier(name),
    FOREIGN KEY(partkind) REFERENCES PartKind(name),
    PRIMARY KEY(supplier, partkind)
);
```

Populate the Database: Example

The following code inserts some fake data to the database.

```
-- Connect to the database.
\c dbmsdb

-- Insert roles.
INSERT INTO role VALUES
('Seller'),
('Mechanic'),
('Financial');

-- Insert the car dealer branches.
INSERT INTO branch (name, city, phone, fax, address) VALUES
('Motorsport', 'Padova', '0497867801', '0492727225', 'Viale dei Giardini 2'),
('Unicar', 'Milano', '0259162579', '0252856087', 'Via Zane 5'),
('Automania', 'Verona', '0453183789', '0457631961', 'Via Garibaldi 15');

-- Insert the employees.
INSERT INTO employee (tc, name, surname, email, password, roleid, branchid) VALUES
('RSSMRA74E23G224P', 'Mario', 'Rossi', 'mario.rossi@gmail.com', '
    e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e', 'Seller', 'Motorsport'),
```

```sql
('PRIRSS85P29L900N', 'Piero', 'Russo', 'piero.russo@msn.com', '7448
    d8798a4380162d4b56f9b452e2f6f9e24e7a', 'Mechanic', 'Motorsport'),
('CLMGLI90E41G224E', 'Giulia', 'Colombo', 'giulia.colombo@gmail.com', '
    a3db5c13ff90a36963278c6a39e4ee3c22e2a436', 'Seller', 'Motorsport'),
('SNTRRT70B69L736R', 'Roberta', 'Santoro', 'roberta.santoro.3@gmail.com', '9
    c6b057a2b9d96a4067a749ee3b3b0158d390cf1', 'Financial', 'Motorsport'),
('BNVZZE18P16C377F', 'Zete', 'Beneventano', 'zete.beneventano@gmail.com', '5
    d9474c0309b7ca09a182d888f73b37a8fe1362c', 'Mechanic', 'Motorsport'),
('SYLDNM04C47B947T', 'Adelina Maria', 'Sylla', 'aelina.sylla@gmail.com', '
    ccf271b7830882da1791852baeca1737fcbe4b90', 'Seller', 'Motorsport'),
('PLNRMR33A62M271C', 'Rosmeri', 'Plantone', 'rosmeri.plantone@hotmail.com', '
    d3964f9dad9f60363c81b688324d95b4ec7c8038', 'Financial', 'Motorsport'),
('PRRMTH10E53D682J', 'Martha Jazmina', 'Parrinello', 'martha.parrinello@gmail.com', '136571
    b41aa14adc10c5f3c987d43c02c8f5d498', 'Seller', 'Unicar'),
('MRTMST90A46F980M', 'Mariastella Cristina', 'Martari', 'mariastella.martari@gmail.com', '
    b6abd567fa79cbe0196d093a067271361dc6ca8b', 'Financial', 'Unicar'),
('CRRLDE88M13A735I', 'Ledio', 'Carrozzo', 'ledio.carrozzo.4@gmail.com', '4143
    d3a341877154d6e95211464e1df1015b74bd', 'Mechanic', 'Unicar'),
('GHTMLT08H54D548U', 'Maria Altomare', 'Ghita', 'maria.ghita@alice.it', '29581
    e412c0981bffd7d0f4a9cdd9b114fb80947', 'Seller', 'Unicar'),
('BSCLRT83C10G997N', 'Alberto', 'Boschetti', 'albero.boschetti.2@gmail.com', '
    c6112fc247276d82a57cc9b56a41999dbba4b169', 'Mechanic', 'Unicar'),
('BNCLNZ95R13I181C', 'Lorenzo', 'Bianchi', 'bianchi.lorenzo@gmail.com', '97
    ea7ec8a6bb8ab9049d86bc39b5be2b0800b14b', 'Mechanic', 'Unicar'),
('BGLGLI09B14E415X', 'Giulio', 'Boglione', 'giulio.boglione.10@gmail.com', '22980
    f6cd0807e719d7f1b7cf25edc9df659ba1f', 'Seller', 'Unicar'),
('PCTLGU93P15F748V', 'Luigi', 'Pacotto', 'luigi.pacotto@gmail.com', '69340613
    d824dc2a0f66fef2b97214550b6ae248', 'Mechanic', 'Automania'),
('CSCGTA94T45I290X', 'Agata', 'Ceschini', 'agata.ceschini@hotmail.com', '37
    e3ecf5f468d8af8698ba15797184523a4b401c', 'Seller', 'Automania'),
('CZZLLN10C71F401D', 'Liliana', 'Cozzolino', 'liliana.cozzolino@yahoo.com', '
    eeec021898eec14f9c7c888c5899a81ad4dbf1ae', 'Seller', 'Automania'),
('SCCNNA13B57M332A', 'Anna', 'Siccardi', 'anna.siccardi.2@gmail.com', '45
    c20f2147e030b02a9a42080890a9183c054eba', 'Financial', 'Automania'),
('GVZLNE62D67H544S', 'Elena', 'Gavazza', 'gavazza.elena@gmail.com', '5
    bf6e5357bd42a6b1d2a3a040e16a91490064d26', 'Financial', 'Automania'),
('VCCMRA81D09C537Q', 'Mario', 'Vecchiatti', 'mario.vecchiatti.6@gmail.com', '02
    ede1eec131f84e860e69c181f01c93b2ff6d76', 'Mechanic', 'Automania');

-- Insert the customers.
INSERT INTO customer (tc, name, surname, phone, address) VALUES
('ZDCSBT84D96V439Z', 'Mario', 'Fumagalli', '3938065232', 'Via Garibaldi 2, Brescia'),
('XFKVFA20N06I270H', 'Leonardo', 'Urietti', '3348132175', 'Via Mazzini 160, Padova'),
('FROHVS03Y31Z372K', 'Maria', 'Compagni', '3471069829', 'Via Fermi 31, Verona'),
('FPOMKO55E27M241C', 'Paola', 'Feno', '3375151678', 'Via Dante 1, Vigonza'),
('ITQFWR28S88Y708C', 'Fioretta', 'Giovanetti', '3904089468', 'Via Svezia 15, Milano'),
('MJTIVC50U54C200B', 'Alberto', 'Bagarini', '3595872628', 'Viale dei Giardini, Roma'),
('TJVHSI68C02P347G', 'Franco', 'Borcoi', '3646455934', 'Piazza del Duomo, Milano'),
('YSOOUM39V36Q049N', 'Rosa', 'Alfieri', '3724483031', 'Via Piave, Mestre'),
('ZOHWSS89B32H213P', 'Tizio', 'Cheneso', '3466786545', 'Via Padova, Padova');

-- Insert the models.
INSERT INTO model (brand, name) VALUES
('Fiat', 'Panda'),
(’Fiat’, ’500’),
(’Ford’, ’Focus Wagon’),
(’Alfa Romeo’, ’Mito’),
(’Lancia’, ’Ypsilon’),
(’Mini’, ’Cooper’),
(’Opel’, ’Corsa’),
(’Toyota’, ’Yaris’),
(’Peugeot’, ’Expert’),
(’RangeRover’, ’Velar’),
(’Porsche’, ’Panamera’),
(’Citroen’, ’Picasso’),
(’Mercedes’, ’C220’);

-- Insert the cars.
INSERT INTO car (lp, status, type, category, price, km, year, name, brand) VALUES
(’FZ286JY’, ’Available’, ’New’, ’ForSale’, 16200.00, 0, 2021, ’Ypsilon’, ’Lancia’),
(’FW680RN’, ’Available’, ’New’, ’ForSale’, 22000.00, 0, 2021, ’Yaris’, ’Toyota’),
(’FN885VO’, ’Rented’, ’Used’, ’ForSaleRental’, 7800.00, 30000, 2019, ’Corsa’, ’Opel’),
(’DI229PJ’, ’Rented’, ’Used’, ’ForSaleRental’, 1500.00, 180000, 2009, ’Corsa’, ’Opel’),
(’DG080OB’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 2000.00, 180000, 2009, ’Yaris’, ’
    Toyota’),
(’FX020NR’, ’UnderMaintenance’, ’New’, ’ForSale’, 17000.00, 0, 2021, ’500’, ’Fiat’),
(’DL654NV’, ’Rented’, ’Used’, ’ForSaleRental’, 3000.00, 180000, 2009, ’Cooper’, ’Mini’),
(’DO360GM’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 4500.00, 165000, 2010, ’Focus Wagon’
    , ’Ford’),
(’EI946PC’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 6000.00, 105000, 2014, ’Mito’, ’Alfa
     Romeo’),
(’DZ002TP’, ’Available’, ’Used’, ’ForSaleRental’, 4700.00, 135000, 2012, ’Mito’, ’Alfa Romeo’
    ),
(’DS025UY’, ’Available’, ’Used’, ’ForSaleRental’, 3200.00, 110000, 2011, ’Panda’, ’Fiat’),
(’DO362QR’, ’Available’, ’Used’, ’ForSaleRental’, 4300.00, 120000, 2010, ’Focus Wagon’, ’Ford
    ’),
(’DH185CM’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 1700.00, 145000, 2009, ’500’, ’Fiat’
    ),
(’EX524YC’, ’Rented’, ’Used’, ’ForSaleRental’, 8000.00, 75000, 2016, ’Cooper’, ’Mini’),
(’EM606QA’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 4200.00, 105000, 2014, ’Panda’, ’
    Fiat’),
(’DJ269JO’, ’UnderMaintenance’, ’Used’, ’ForSaleRental’, 1300.00, 150000, 2009, ’Yaris’, ’
    Toyota’),
(’FP495LS’, ’Available’, ’Used’, ’ForSaleRental’, 11000.00, 15000, 2020, ’Ypsilon’, ’Lancia’)
    ,
(’FE756EY’, ’Rented’, ’Used’, ’ForSaleRental’, 8800.00, 45000, 2018, ’Corsa’, ’Opel’),
(’DK237DR’, ’Rented’, ’Used’, ’ForSaleRental’, 1800.00, 150000, 2009, ’Panda’, ’Fiat’),
(’FY168KZ’, ’Available’, ’New’, ’ForSale’, 15000.00, 0, 2021, ’Corsa’, ’Opel’);

-- Insert CustomerCar
INSERT INTO CustomerCar (lp, year , name, brand, customer) VALUES
(’OX512LY’ , 2008 , ’Expert’, ’Peugeot’ , ’ZDCSBT84D96V439Z’),
(’SI028FY’ , 2014 , ’Focus Wagon’, ’Ford’ , ’XFKVFA20N06I270H’),
(’MV871FX’ , 2018 , ’Corsa’, ’Opel’ , ’FROHVS03Y31Z372K’),
(’XF505QK’ , 2017 , ’Corsa’, ’Opel’ , ’FPOMKO55E27M241C’),
(’YV798RF’ , 2016 , ’Velar’, ’RangeRover’ , ’ITQFWR28S88Y708C’),
(’FJ549TY’ , 2008 , ’Panamera’, ’Porsche’ , ’ZOHWSS89B32H213P’),
(’HD842VB’ , 2016 , ’Picasso’, ’Citroen’ , ’MJTIVC50U54C200B’),
('PA092PA' , 2012 , 'Focus Wagon', 'Ford' , 'TJVHSI68C02P347G'),
('VB408OO' , 2002 , 'Corsa', 'Opel' , 'YSOOUM39V36Q049N'),
('PI685YC' , 2008 , 'Panda', 'Fiat' , 'ZDCSBT84D96V439Z'),
('QS475SU' , 2018 , 'Expert', 'Peugeot', 'FROHVS03Y31Z372K'),
('AJ588VX' , 2019 , 'C220', 'Mercedes' , 'FPOMK055E27M241C'),
('RC495ZK' , 2016 , 'C220', 'Mercedes' , 'ITQFWR28S88Y708C'),
('TX671BX' , 2000 , 'Yaris', 'Toyota' , 'ZDCSBT84D96V439Z'),
('CT378MY' , 2009 , 'Cooper', 'Mini' , 'ITQFWR28S88Y708C');

-- Insert Sale
INSERT INTO Sale (date, price, employee, lp, customer) VALUES
('2020-09-17', 7000.00, 'RSSMRA74E23G224P', 'FZ286JY', 'ZDCSBT84D96V439Z'),
('2017-06-19', 15000.00, 'CLMGLI90E41G224E', 'FW680RN', 'XFKVFA20N06I270H'),
('2018-04-24', 10000.00, 'PRRMTH10E53D682J', 'DZ002TP', 'FROHVS03Y31Z372K'),
('2021-09-28', 20000.00, 'SYLDNM04C47B947T', 'DO362QR', 'FPOMK055E27M241C');

-- Insert Rental
INSERT INTO rental (start_date, end_date, deal_date, price, employee, customer, lp) VALUES
('2020-08-11', '2022-07-11', '2020-08-11', 900, 'RSSMRA74E23G224P', 'FROHVS03Y31Z372K', '
    FN885VO'),
('2020-06-04', '2023-07-04', '2020-06-04', 630, 'BGLGLI09B14E415X', 'FROHVS03Y31Z372K', '
    DI229PJ'),
('2020-06-08', '2022-08-08', '2020-06-08', 530, 'CZZLLN10C71F401D', 'FPOMK055E27M241C', '
    DL654NV'),
('2020-06-06', '2022-08-06', '2020-06-06', 200, 'RSSMRA74E23G224P', 'MJTIVC50U54C200B', '
    EX524YC'),
('2019-07-07', '2025-07-07', '2019-07-07', 820, 'BGLGLI09B14E415X', 'FROHVS03Y31Z372K', '
    FE756EY'),
('2019-11-18', '2023-04-18', '2019-11-18', 330, 'CZZLLN10C71F401D', 'ITQFWR28S88Y708C', '
    DK237DR'),
('2020-09-11', '2025-09-11', '2020-09-11', 600, 'BGLGLI09B14E415X', 'XFKVFA20N06I270H', '
    FN885VO'),
('2019-09-17', '2027-09-17', '2019-09-17', 120, 'CSCGTA94T45I290X', 'XFKVFA20N06I270H', '
    DI229PJ'),
('2020-05-30', '2022-11-30', '2020-05-28', 610, 'RSSMRA74E23G224P', 'FROHVS03Y31Z372K', '
    DL654NV'),
('2019-09-01', '2023-04-01', '2019-09-01', 660, 'CSCGTA94T45I290X', 'XFKVFA20N06I270H', '
    EX524YC');

-- Insert Repair
INSERT INTO Repair (date, price, hours, employee) VALUES
('2021-11-06', 2549.14, 7, 'PRIRSS85P29L900N'),
('2020-12-17', 6850.61, 30, 'VCCMRA81D09C537Q'),
('2021-06-24', 5065.35, 33, 'BNVZZE18P16C377F'),
('2021-10-17', 4106.15, 98, 'PRIRSS85P29L900N'),
('2021-11-09', 2799.37, 46, 'BSCLRT83C10G997N'),
('2021-04-23', 7616.73, 72, 'PRIRSS85P29L900N'),
('2021-09-06', 2812.5, 76, 'BNVZZE18P16C377F'),
('2021-05-22', 8042.78, 58, 'BNVZZE18P16C377F'),
('2021-04-02', 2218.69, 31, 'BSCLRT83C10G997N'),
('2021-10-14', 7968.18, 62, 'CRRLDE88M13A735I');

-- Insert CustomerCarRepair
INSERT INTO CustomerCarRepair (lp, repair ) VALUES
```

```
('OX512LY' , 1 ),
('SI028FY' , 2 ),
('MV871FX' , 3 ),
('XF505QK' , 4 ),
('YV798RF' , 5 );

-- Insert CarRepair
INSERT INTO CarRepair (lp, repair ) VALUES
('DS025UY', 6 ),
('DO362QR', 7 ),
('DH185CM', 8 ),
('EX524YC', 9 );

-- Insert Supplier
INSERT INTO Supplier (name, phone) VALUES
('AllstarAuto', 3376890987),
('MotorHigh', 3376890987),
('EngineShop', 3376890987);

-- Insert PartKind
INSERT INTO PartKind (name, category ) VALUES
('E9X Rear Suspension', 'Suspension'),
('Primary Tree 4085089', 'Tree'),
('JE Piston 244871-1', 'Piston'),
('ASUPERMALL 300mm', 'Mirror');

-- Insert Supplies
INSERT INTO Supplies (supplier, partkind, catalog_price ) VALUES
('AllstarAuto', 'E9X Rear Suspension', 200.00),
('AllstarAuto', 'JE Piston 244871-1', 1256.40),
('AllstarAuto', 'Primary Tree 4085089', 390.00),
('MotorHigh', 'E9X Rear Suspension', 235.00),
('MotorHigh', 'JE Piston 244871-1', 194.00),
('MotorHigh', 'Primary Tree 4085089', 378.00),
('EngineShop', 'JE Piston 244871-1', 789.9),
('EngineShop', 'ASUPERMALL 300mm', 74.90),
('EngineShop', 'E9X Rear Suspension', 289.99);

-- Insert Replacement
INSERT INTO Replacement ( partkind, price, supplier, repair ) VALUES
('ASUPERMALL 300mm', 70.00, 'EngineShop', 1),
('JE Piston 244871-1', 1200.00, 'AllstarAuto', 2),
('JE Piston 244871-1', 1300.00, 'AllstarAuto', 6);
```

Principal Queries

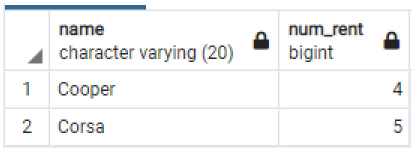
In this section we propose some queries to access our database:

1. Find the most rented models, i.e. the cars that were rented above the average.
2. Find the best performing branch in terms of total income from sales.

3. Find the best-selling car model.

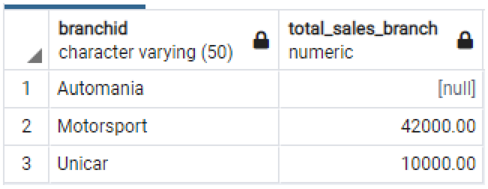
4. For each kind of replacement part, find the supplier which supplies it at lowest cost.

```
SELECT Model.name, COUNT(*) AS num_rent
FROM Model
JOIN Car ON Model.name = Car.name
JOIN Rental ON Car.lp = Rental.lp
GROUP BY Model.name
HAVING COUNT(*) > (SELECT AVG(num_rent)
    FROM (SELECT COUNT(*) AS num_rent
    FROM Model
    JOIN Car ON Model.name = Car.name
    JOIN Rental ON Car.lp = Rental.lp
    GROUP BY Model.name) AS foo);
```



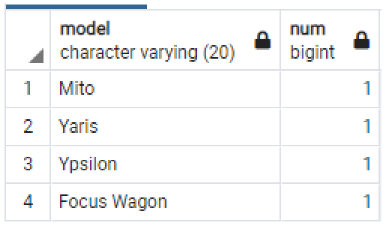
	name character varying (20)	num_rent bigint
1	Cooper	4
2	Corsa	5

```
SELECT branchid, SUM(total_sales) AS total_sales_branch FROM
    (SELECT employee, SUM(price) AS total_sales
    FROM Sale
    GROUP BY employee
    ORDER BY total_sales DESC) AS T FULL JOIN Employee AS E
    ON T.employee = E.tc
GROUP BY branchid
ORDER BY total_sales_branch DESC;
```



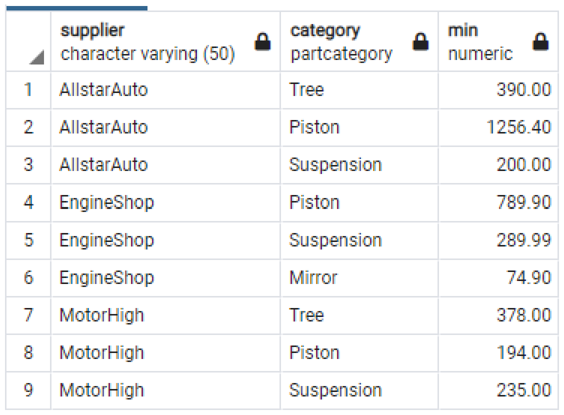
	branchid character varying (50)	total_sales_branch numeric
1	Automania	[null]
2	Motorsport	42000.00
3	Unicar	10000.00

```
SELECT model, num
FROM (  SELECT c.name AS model, count(*) AS num
        FROM car AS c INNER JOIN sale AS s ON c.lp = s.lp
        GROUP BY c.name ) AS counter1
WHERE num = (
        SELECT max(num)
        FROM (  SELECT c.name AS model, count(*) AS num
                FROM car AS c INNER JOIN sale AS s ON c.lp = s.lp
                GROUP BY c.name ) AS counter2 );
```



	model character varying (20)	num bigint
1	Mito	1
2	Yaris	1
3	Ypsilon	1
4	Focus Wagon	1

```
SELECT s.supplier, p.category, MIN(s.catalog_price)
FROM supplies AS s INNER JOIN partkind AS p ON p.name = s.partkind
GROUP BY s.supplier, p.category
ORDER BY s.supplier ASC;
```



	supplier character varying (50)	category partcategory	min numeric
1	AllstarAuto	Tree	390.00
2	AllstarAuto	Piston	1256.40
3	AllstarAuto	Suspension	200.00
4	EngineShop	Piston	789.90
5	EngineShop	Suspension	289.99
6	EngineShop	Mirror	74.90
7	MotorHigh	Tree	378.00
8	MotorHigh	Piston	194.00
9	MotorHigh	Suspension	235.00

JDBC Implementations of the Principal Queries and Visualization

Our implementation of the queries relies on two type of classes: the main classes implement the queries, while the helper classes implement some auxiliary functionality.

Principal Queries and Visualization implementations

FindAvailableBrandAndOrder.java

```
package dbms;

import java.sql.SQLException;

import static dbms.ExceptionalExecutor.execute;
import static dbms.SQLConfiguration.withConnection;
import static dbms.SQLStatement.executeQueryThen;
import static dbms.SQLStatement.withStatement;
import static java.lang.String.format;
```

```java
public class FindAvailableBrandAndOrder {
    final static String SQL_QUERY = "SELECT * FROM car WHERE brand = '%s' AND status = '
        Available' ORDER BY price ASC;";
    static String itsBrand;

    public static void main(String[] args) {
        readArguments(args);
        execute(FindAvailableBrandAndOrder::executeStatement);
    }

    private static void readArguments(String[] args) {
        if (args.length < 1) {
            System.err.println("Expecting the brand of the car, quitting..");
            System.exit(-1);
        }
        itsBrand = args[0];
    }

    private static void executeStatement() throws Exception {
        withConnection(() ->
            withStatement(() ->
                executeQueryThen(getQuery(), FindAvailableBrandAndOrder::print)));
    }

    private static String getQuery() {
        return format(SQL_QUERY, itsBrand);
    }

    private static void print() throws SQLException {
        printHeader();
        while (SQLStatement.next()) {
            String licensePlate = SQLStatement.getString("lp");
            String type = SQLStatement.getString("type");
            String category = SQLStatement.getString("category");
            double price = SQLStatement.getDouble("price");
            int km = SQLStatement.getInt("km");
            int year = SQLStatement.getInt("year");
            String name = SQLStatement.getString("name");
            String brand = SQLStatement.getString("brand");
            printLine(licensePlate, type, category, formatPrice(price), Integer.toString(km),
                 Integer.toString(year), name, brand);
        }
    }

    private static String formatPrice(double price) {
        char euro = '\u20AC';
        return String.format("%.2f%c", price, euro);
    }

    private static void printHeader() {
        printLine("L. P.", "Type", "Category", "Price", "Km", "Year", "Name", "Brand");
    }
    private static void printLine(String licensePlate, String type, String category, String
        price, String km, String year, String name, String brand) {
        System.out.format("%10s%14s%16s%8s%10s%14s%6s%16s%n", licensePlate, brand, name, year
            , km, price, type, category);
    }
}
```

FindPerformancesOfBranches.java

```
package dbms;

import java.sql.SQLException;

import static dbms.ExceptionalExecutor.execute;
import static dbms.SQLConfiguration.withConnection;
import static dbms.SQLStatement.executeQueryThen;
import static dbms.SQLStatement.withStatement;

public class FindPerformancesOfBranches {
    final static String SQL_QUERY = "SELECT branchid, SUM(total_sales) AS total_sales_branch
        FROM \n" +
        "    (SELECT employee, SUM(price) AS total_sales\n" +
        "    FROM Sale\n" +
        "    GROUP BY employee\n" +
        "    ORDER BY total_sales DESC) AS R INNER JOIN Employee AS P\n" +
        "    ON R.employee = P.tc\n" +
        "GROUP BY branchid \n" +
        "ORDER BY total_sales_branch DESC;";

    public static void main(String[] args) {
        execute(FindPerformancesOfBranches::executeStatement);
    }

    private static void executeStatement() throws Exception {
        withConnection(() ->
            withStatement(() ->
                executeQueryThen(SQL_QUERY, FindPerformancesOfBranches::print)));
    }

    private static void print() throws SQLException {
        printHeader();
        while (SQLStatement.next()) {
            String employee = SQLStatement.getString("branchid");
            String total_sales = SQLStatement.getString("total_sales_branch");
            printLine(employee, total_sales);
        }
    }

    private static void printHeader() {
        printLine("Branch Id", "    Total Sales of Branch");
    }

    private static void printLine(String branch_id, String total_sales_branch) {
        System.out.format("%12s%12s%n", branch_id, total_sales_branch);
    }
}
```

Helper classes

SQLConfiguration.java

```
package dbms;

import java.io.FileReader;
import java.sql.*;
import java.util.Properties;
import java.util.regex.Pattern;

import static dbms.ExceptionalExecutor.withContext;
import static java.lang.Integer.parseInt;
import static java.lang.String.format;

/**
 * Abstracts SQL connection configuration.
 */
public class SQLConfiguration {
    private static final String PROPERTIES_FILE_NAME = "dbms.properties";
    private static String itsHost;
    private static String itsDatabase;
    private static int itsPort;
    private static String itsUsername;
    private static String itsPassword;
    private static Connection itsConnection;

    /**
     * Executes the given {@code action} after a connection has been successfully established
         .
     */
    public static void withConnection(ExceptionalRunnable action) throws Exception {
        try {
            withContext(
                "while establishing a connection to " + getDatabaseUrl(),
                () -> itsConnection = makeConnection());
            action.run();
        } finally {
            // If resource has been successfully instantiated...
            if (itsConnection != null) {
                // Once execution exists this scope, it will be eligible for garbage
                    collection
                Connection c = itsConnection;
                // Ensure its reference is null before closing it
                itsConnection = null;
                // Close the resource
                withContext("while closing current connection", c::close);
            }
        }
    }

    private static Connection makeConnection() throws Exception {
```

```
    return DriverManager.getConnection(getDatabaseUrl(), itsUsername, itsPassword);
}

private static String getDatabaseUrl() throws Exception {
    loadConfiguration();
    final String URL = "jdbc:postgresql://%s:%d/%s";
    return format(URL, itsHost, itsPort, itsDatabase);
}

private static void loadConfiguration() throws Exception {
    Properties properties = new Properties();
    loadPropertiesFile(properties);
    readConfiguration(properties);
}

private static void readConfiguration(Properties properties) throws Exception {
    readHost(properties);
    readPort(properties);
    readDatabase(properties);
    readUsername(properties);
    readPassword(properties);
}

private static void loadPropertiesFile(Properties properties) throws Exception {
    withContext(
        "while opening configuration file \"" + PROPERTIES_FILE_NAME + "\"",
        () -> properties.load(new FileReader(PROPERTIES_FILE_NAME)));
}

private static void readHost(Properties properties) throws Exception {
    withContext("while reading \"host\" parameter from the configuration file", () -> {
        itsHost = properties.getProperty("host");
        if (!isHostName() && !isHostIpAddress())
            throw new IllegalArgumentException("The host must be an IP address or a
                hostname.");
    });
}

private static boolean isHostIpAddress() {
    final String HOST_IP_REGEX = "^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.)
        {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";
    return Pattern.compile(HOST_IP_REGEX).matcher(itsHost).matches();
}

private static boolean isHostName() {
    final String HOST_NAME_REGEX = "^(([a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\\-]*[a-zA-Z0-9])
        \\.)*([A-Za-z0-9]|[A-Za-z0-9][A-Za-z0-9\\-]*[A-Za-z0-9])$";
    return Pattern.compile(HOST_NAME_REGEX).matcher(itsHost).matches();
}

private static void readPort(Properties properties) throws Exception {
    withContext("while reading \"port\" parameter from the configuration file", () ->
        itsPort = parseInt(properties.getProperty("port"))
    );
```

```java
    }

    private static void readDatabase(Properties properties) throws Exception {
        withContext("while reading \"database\" parameter from configuration file", () -> {
            itsDatabase = properties.getProperty("database");
            if (!isDatabaseValid())
                throw new IllegalArgumentException("The database name is not valid.");
        });
    }

    private static boolean isDatabaseValid() {
        final String DATABASE_NAME_REGEX = "^[0-9a-zA-Z$_]+$";
        return Pattern.compile(DATABASE_NAME_REGEX).matcher(itsDatabase).matches();
    }

    private static void readUsername(Properties properties) throws Exception {
        withContext("while reading \"user\" parameter from configuration file", () ->
            itsUsername = properties.getProperty("user")
        );
    }

    private static void readPassword(Properties properties) throws Exception {
        withContext("while reading \"pass\" parameter from configuration file", () ->
            itsPassword = properties.getProperty("pass")
        );
    }

    public static Statement createStatement() throws SQLException {
        return itsConnection.createStatement();
    }

    public static PreparedStatement prepareStatement(String sql) throws SQLException {
        return itsConnection.prepareStatement(sql);
    }
}
```

SQLStatement.java

```java
package dbms;

import java.sql.*;

import static dbms.ExceptionalExecutor.withContext;

/**
 * Abstracts SQL statements.
 */
public class SQLStatement {
    private static Statement itsStatement;
    private static ResultSet itsResultSet;
    private static PreparedStatement itsPreparedStatement;
    private static int itsExecutionCount;

    /**
 * Executes the given {@code action} after a {@link PreparedStatement} has been
     successfully built
 * by the SQL code in {@code sql}.
 */
public static void withPreparedStatement(String sql, ExceptionalRunnable action) throws
    Exception {
    try {
        itsPreparedStatement = SQLConfiguration.prepareStatement(sql);
        action.run();
    } finally {
        // If resource has been successfully instantiated...
        if (itsPreparedStatement != null) {
            // Once execution exists this scope, it will be eligible for garbage
                collection
            PreparedStatement ps = itsPreparedStatement;
            // Ensure its reference is null before closing it
            itsPreparedStatement = null;
            // Close the resource
            withContext("while closing prepared statement", ps::close);
        }
    }
}

/**
 * Executes the given {@code action} after a {@link Statement} has been successfully
     created.
 */
public static void withStatement(ExceptionalRunnable action) throws Exception {
    try {
        itsStatement = SQLConfiguration.createStatement();
        action.run();
    } finally {
        // If resource has been successfully instantiated...
        if (itsStatement != null) {
            // Once execution exists this scope, it will be eligible for garbage
                collection
            Statement s = itsStatement;
            // Ensure its reference is null before closing it
            itsStatement = null;
            // Close the resource
            withContext("while closing statement", s::close);
        }
    }
}

/**
 * Executes the given {@code action} after a {@link ResultSet} has been successfully
     returned after
 * the given {@code query} has been executed.
 */
public static void executeQueryThen(String query, ExceptionalRunnable action) throws
    Exception {
    try {
        withContext(
            "while executing query \"" + query + "\"",
            () -> itsResultSet = itsStatement.executeQuery(query));
        action.run();
    } finally {
        // If resource has been successfully instantiated...
        if (itsResultSet != null) {
            // Once execution exists this scope, it will be eligible for garbage
                collection
            ResultSet rs = itsResultSet;
            // Ensure its reference is null before closing it
            itsResultSet = null;
            // Close the resource
            withContext("while closing result set", rs::close);
        }
    }
}

public static boolean next() throws SQLException {
    return itsResultSet.next();
}

public static String getString(String columnLabel) throws SQLException {
    return itsResultSet.getString(columnLabel);
}

public static double getDouble(String columnLabel) throws SQLException {
    return itsResultSet.getDouble(columnLabel);
}

public static int getInt(String columnLabel) throws SQLException {
    return itsResultSet.getInt(columnLabel);
}

public static Date getDate(String columnLabel) throws SQLException {
    return itsResultSet.getDate(columnLabel);
}

public static void setString(int parameterIndex, String value) throws SQLException {
    itsPreparedStatement.setString(parameterIndex, value);
}

public static void setInt(int parameterIndex, int value) throws SQLException {
    itsPreparedStatement.setInt(parameterIndex, value);
}

public static void execute() throws SQLException {
    itsPreparedStatement.execute();
    itsExecutionCount++;
}

public static int getExecutionCount() {
    return itsExecutionCount;
}
}
```

CSVFileReader.java

```java
package dbms;

import java.io.BufferedReader;
import java.io.FileReader;

import static dbms.ExceptionalExecutor.withContext;

/**
 * Provides abstractions for reading CSV files.
 */
public class CSVFileReader {
    /**
     * Feeds {@code rowConsumer} an instance of {@link CSVRow} for each eligible row in the
         file
     * whose name is given by the {@code fileName} parameter.
     */
    public static void forEachRowInCSVFile(String fileName, CSVRowSQLConsumer rowConsumer)
        throws Exception {
        withContext("while opening CSV file " + fileName, () -> {
            try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
                withContext("while reading " + fileName, () -> {
                    String line;
                    while ((line = br.readLine()) != null)
                        CSVRow.parseThen(line, rowConsumer);
                });
            }
        });
    }
}
```

CSVRow.java

```java
package dbms;

/**
 * Represents a row in a CSV file.
 */
public class CSVRow {
    private final String[] itsFields;

    public CSVRow(String line) {
        itsFields = line.split(",");
    }

    public static void parseThen(String line, CSVRowSQLConsumer rowConsumer) throws Exception
         {
        if (isNotComment(line))
            rowConsumer.accept(new CSVRow(line));
    }

    private static boolean isNotComment(String line) {
        return !line.startsWith("#");
    }

    public int readIntAt(int fieldIndex) {
        return Integer.parseInt(itsFields[fieldIndex]);
    }

    public String readStringAt(int fieldIndex) {
        return itsFields[fieldIndex];
    }
}
```

CSVRowSQLConsumer.java

```java
package dbms;

/**
 * Represents a consumer of {@link CSVRow} which executes SQL-related code.
 */
public interface CSVRowSQLConsumer {
    void accept(CSVRow csvRow) throws Exception;
}
```

ExceptionalExecutor.java

```java
package dbms;

import java.io.FileNotFoundException;
import java.sql.SQLException;
import java.util.Deque;
import java.util.LinkedList;

import static java.lang.String.valueOf;

/**
 * Handles exceptions gracefully, providing a context-aware way to report exceptional
     behavior.
 *
 * Use {@link #execute(ExceptionalRunnable)} as an entry-point, then {@link #withContext(
     String, ExceptionalRunnable)}
 * will add user-driven context that may include also value of variables, which are usually
     absent from
 * the stack trace.
 */
public class ExceptionalExecutor {
    private static final Deque<String> itsContextStack = new LinkedList<>();

    /**
     * Executes the given runnable, handling any exception it may throw.
     */
    public static void execute(ExceptionalRunnable exceptionalRunnable) {
        try {
            exceptionalRunnable.run();
        } catch (SQLException sqlException) {
            onSQLException(sqlException);
        } catch (FileNotFoundException fileNotFoundException) {
            onFileNotFoundException(fileNotFoundException);
        } catch (IllegalArgumentException illegalArgumentException) {
            onIllegalArgumentException(illegalArgumentException);
        } catch (Exception exception) {
            // If execution reaches this point, we may not want to catch this exception,
            // so we decorate it as an Error and let it go.
            throw new Error(exception);
        } finally {
            itsContextStack.clear();
        }
    }

    /**
     * Wraps the given runnable in a new context.
     */
    public static void withContext(String context, ExceptionalRunnable exceptionalRunnable)
        throws Exception {
        itsContextStack.push(context);
        exceptionalRunnable.run();
        itsContextStack.pop();
    }

    private static void onSQLException(SQLException sqlException) {
        printHeader(sqlException);
        printEntry("SQL status code", sqlException.getSQLState());
        printEntry("SQL error code", valueOf(sqlException.getErrorCode()));
    }

    private static void onFileNotFoundException(FileNotFoundException fileNotFoundException)
        {
        printHeader(fileNotFoundException);
    }

    private static void onIllegalArgumentException(IllegalArgumentException
        illegalArgumentException) {
        printHeader(illegalArgumentException);
    }

    private static void printHeader(Exception e) {
        System.err.println("Unrecoverable error" + getContext() + ":");
        printEntry("Message", e.getMessage());
    }

    private static void printEntry(String key, String value) {
        System.err.format("- %s: %s%n", key, value);
    }

    private static String getContext() {
        String s = itsContextStack.peek();
        if (s == null || "".equals(s))
            return "";
        else
            return " " + s;
    }
}
```

ExceptionalRunnable.java

```
package dbms;

/**
 * Represents an action that may throw an exception.
 */
public interface ExceptionalRunnable {
    void run() throws Exception;
}
```

Group Members Contribution

Name	**Surname**	**Contribution**
Anahita	Abbaspour	
Mustafa	Algun	JDBC implementation of some queries
Selen	Arslan	Populate the database
Alex	Baron	Physical schema Some queries Final review
Christian	Cattai	Database population Physical Schema Some fixes here and there Final review
Fatjon	Freskina	Part of queries.sql Small contribution insert.sql and cardealership.sql
Enrico	Leka	Physical Schema One query Final review
Marco	Milanese	Variations to the relational schema Part of database data population Final review
Giulio	Scattolin	JDBC utility and helper classes README.md contents InsertCustomerCars query