8 DML - Експлицитни спојувања и останати наредби

Заеднички изрази на табели (Common Table Expressions – CTE), погледи и привремени табели (Temporary Tables)

Кога имаме потреба од решавање на посложени прашања и/или сакаме да чуваме некаков меѓурезултат можеме да користиме погледи, заеднички изрази на табели или привремени табели. Откако било кој од овие елементи на базата на податоци ќе биде креиран може да се користи како било која друга табела од таа база.

CTE – особини и сценарија за користење:

- Се чува само дефиницијата, додека резултатот се пресметува во тек на извршување
- Во рамки на едно прашање можат да се искористат повеќе CTE, секое следно може да ги користи сите претходни
- Најмалку ја оптеретуваат базата на податоци во споредба со останатите елементи
- Се користи како замена за погледи кога истите не се неопходни т.е. кога нема потреба да се чува дефиницијата во метаподатоците на базата на податоци
- Овозможува групирање по атрибут кој се добива преку аритметичка операција или функција која е недетерминистичка
- Овозможува референцирање на резултантната табела повеќе пати во истиот израз
- Може да се искористи единствено во SQL SELECT изразот кој следува непосредно по дефиницијата

Синтакса:

```
WITH expression_name [ ( column_name [,...n] ) ] AS
( CTE_query_definition )
```

Привремени табели – особини и сценарија за користење:

- Се чуваат податоците, а не дефиницијата
- Може да се дефинираат индекси и надворешни клучеви
- Областа на важење им е во тековната сесија на тековниот корисник (ако како префикс на името имаат #) или се зачувуваат трајно во tempdb ако како префикс на името имаат ##
- Треба да се избришат (drop) после користењето. Локалните (со префикс #) се бришат автоматски после затворањето на сесијата
- Креирањето на привремените табели е идентично како креирањето на регуларните табели, со единствена разлика во именувањето т.е. имињата на привремените табели мора да започнуваат со #.

Синтакса:

```
CREATE TABLE #table_name
( table definition )
```

Погледи – особини и сценарија за користење:

- Се чува само дефиницијата, додека резултатот се пресметува во тек на извршување
- Се креира траен (материјализиран) објект во базата на податоци (може да се дефинираат различни привилегии за различни корисници)
- Може да се користи во било кој SQL израз
- Се користи да го фокусира, поедностави и прилагоди начинот на кој различните корисници ја гледаат базата на податоци
- Може да се искористи како безбедносен механизам така што ќе им се дозволи на корисниците да пристапуваат до податоците единствено преку погледи, без да им се дозволи пристап до табелите врз кои погледот е изграден.
- Може да се искористи како интерфејс за да се "симулира" табела која може да ја смени својата дефиниција
- Ако повеќе не се користат треба да се избришат за да не ја оптоваруваат базата на податоци

Синтакса:

```
CREATE VIEW <view_name> [(column1, column2...)] AS
( VIEW_query_definition )
```

ЗАДАЧА 1. Нека ја имаме следната база на податоци составена од следните табели:

COMPANY (Cname, address, city, country)
ACCOUNT (AccNum, type, balance, bankName)
PAYMENT (Cname*, AccNum*, amount, late)

а) Креирајте CTE, привремена табела и поглед кој ќе ги содржи сите информации од табелата PAYMENT.

```
-- CTE
WITH AUX_PAYMENT AS
(SELECT * FROM PAYMENT)

-- Ако сакаме да го искористиме CTE за да изброиме колку плаќања има во базата
SELECT COUNT(*) FROM AUX_PAYMENT;

-- Temp table
SELECT *
INTO #AUX_PAYMENT
FROM PAYMENT;

-- Привремената табела може да се искористи за да изброиме колку плаќања има
во базата
SELECT COUNT(*) FROM #AUX_PAYMENT;
```

```
-- VIEW
CREATE VIEW AUX_PAYMENT AS
SELECT *
FROM PAYMENT;

-- Ако сакаме да го искористиме погледот за да изброиме колку плаќања има во
базата
SELECT COUNT(*) FROM AUX_PAYMENT;
```

b) Креирајте CTE кое ќе ги содржи сите плаќања за сметката со број 4.

```
WITH PAYMENT_4 AS
(  SELECT *
   FROM PAYMENT
   WHERE AccNum = 4   )
```

c) Креирајте поглед кој ќе ги содржи името на секоја компанија и нејзините поштенски информации (во единствен атрибут).

```
CREATE VIEW ENVELOPE (name, postalInfo) AS
SELECT Cname, address + " " + city + ", " + country
FROM COMPANY
```

d) Креирајте CTE, привремена табела и поглед кој ќе ги содржи вкупниот број на плаќања извршени по држави, заедно со вкупната сума уплатена со тие плаќања.

```
-- CTE
WITH COUNTRY_PAYMENT AS (
SELECT country, COUNT(P.Cname) paymentNum, SUM(amount) total
FROM PAYMENT P, COMPANY C
WHERE P.Cname = C.Cname
GROUP BY country)


-- TEMP table
SELECT country, COUNT(P.Cname) paymentNum, SUM(amount) total
INTO #COUNTRY_PAYMENT
FROM PAYMENT P, COMPANY C
WHERE P.Cname = C.Cname
GROUP BY country


-- VIEW
CREATE VIEW COUNTRY_PAYMENT (country, paymentNum, total) AS
SELECT country, COUNT(P.Cname), SUM(amount)
FROM PAYMENT P, COMPANY C
WHERE P.Cname = C.Cname
GROUP BY country;
```

ЕКСПЛИЦИТНИ СПОЈУВАЊА

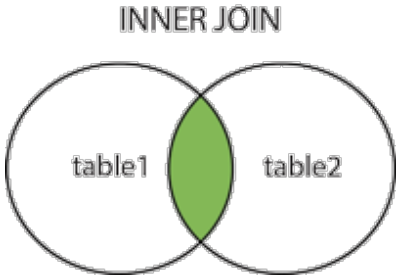
КРЕИРАЊЕ НА ТАБЕЛИ ЗА ТЕСТИРАЊЕ

```
CREATE TABLE table1
(ID INT, Value VARCHAR(10))

INSERT INTO Table1 (ID, Value)
VALUES (1,'First'),
       (2,'Second'),
       (3,'Third'),
       (4,'Forth'),
       (5,'Fifth')

CREATE TABLE table2
(ID INT, Value VARCHAR(10))

INSERT INTO Table2 (ID, Value)
VALUES (1,'First'),
       (2,'Second'),
       (3,'Third'),
       (6,'Sixth'),
       (7,'Seventh'),
       (8,'Eighth')
GO

SELECT *
FROM Table1

SELECT *
FROM Table2
```

INNER JOIN

Со овој тип на спојување ги спојува торките од двете табели за кои важи дека го исполнуваат условот за спојување. Може да се спојат повеќе од две табели и во тој случај се враќаат торките кои ги задоволуваат сите наведени услови.

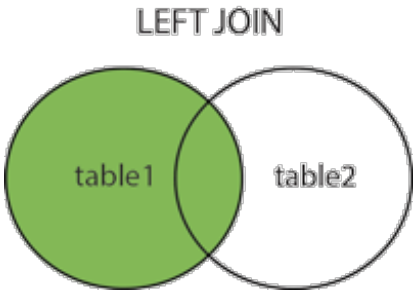
```
SELECT t1.*,t2.*
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.ID = t2.ID
```



LEFT OUTER JOIN

Со ова спојување се враќаат **сите торки од левата табела** и само оние торки од десната табела кои го задоволуваат условот за спојување. Торките од левата табела за кои не постои соодветна торка од десната табела (не е задоволен условот за спојување) се комбинираат со торки кај кои сите атрибути имаат NULL вредност. Лева табела е онаа која е наведена пред наредбата, а десна онаа после наредбата.

```
SELECT t1.*,t2.*
FROM Table1 t1
LEFT JOIN Table2 t2
ON t1.ID = t2.ID
```



Овој тип на спојување е корисен во таканаречените anti-join услови, односно кога треба да најдеме торки од една табела (лева) за кои не постојат соодветни торки во друга табела (десна). На пример:

```
SELECT t1.*,t2.*
FROM Table1 t1
LEFT JOIN Table2 t2
ON t1.ID = t2.ID
WHERE t2.ID IS NULL
```

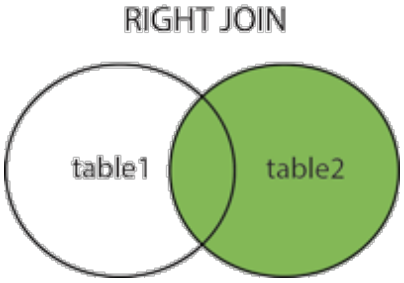
Истото може да се постигне и на друг, помалку ефикасен начин (поради оптимизациите за индексите на табелите, разбирливоста на прашањето и ограничената можност за користење кога постојат повеќе услови за спојување):

```
SELECT t1.*
FROM Table1 t1
WHERE t1.ID NOT IN (SELECT t2.ID FROM Table2 t2)
```

RIGHT OUTER JOIN

Ова спојување е многу слично на претходното. Единствената разлика е во тоа што тука во резултатот ги имаме **сите торки од десната табела** и само оние торки од левата табела кои го задоволуваат условот за спојување (повторно имаме NULL вредности таму каде што спојувањето не е возможно).

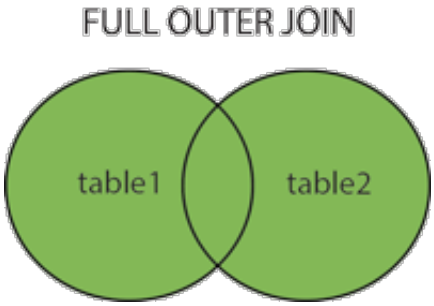
```
SELECT t1.*,t2.*
FROM Table1 t1
RIGHT JOIN Table2 t2
ON t1.ID = t2.ID
```



FULL OUTER JOIN

Овој тип на спојување ги враќа сите торки од двете табели без разлика дали ги задоволуваат условите за спојување. За оние торки од левата табела за кои не постои соодветна торка од десната табела (нема торка со која може да се спои според наведениот услов) сите атрибути од десната табела добиваат вредности NULL. Важи и обратното.

```
SELECT t1.*,t2.*
FROM Table1 t1
FULL OUTER JOIN Table2 t2
ON t1.ID = t2.ID
```



CROSS JOIN

Овој тип на соединување одговара на декартов производ на табелите. Тука не се дефинира никаков услов за соединување. Еквивалентен резултат би се добил со просто наведување на имињата на табелите одвоени помеѓу себе со запирка.

```
SELECT t1.*,t2.*
FROM Table1 t1
CROSS JOIN Table2 t2
```

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	4	Fourth
5	5	Fifth

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	6	Sixth
5	7	Seventh
6	8	Eighth

	ID	Value	ID	Value
1	1	First	1	First
2	1	First	2	Second
3	1	First	3	Third
4	1	First	6	Sixth
5	1	First	7	Seventh
6	1	First	8	Eighth
7	2	Second	1	First
8	2	Second	2	Second
9	2	Second	3	Third
10	2	Second	6	Sixth
11	2	Second	7	Seventh
12	2	Second	8	Eighth

ЗАДАЧА 2. Дадена е база на податоци за една компанија со следните релации:

Model (Id, Name, Description) /инфо за модели
Size (Id, Name, Description) /инфо за големини
Color (Id, Name) /инфо за бои
StoreProduct (Id, ModelId*, SizeId*, ColorId*, RetailPrice, ItemsOnStock) /инфо за производи продавани во продавница
FactoryProduct (Id, ModelId*, SizeId*, ColorId*, FactoryPrice) /инфо за производи кои се произведуваат во фабриката

Напишете ги SQL изразите кои ќе ги решат следните барања:

1. Табелата FactoryProduct е празна и треба да се пополни со сите можни комбинации на модели, големини и бои за кои компанијата води информација.
2. Вратете ги сите информации за производи кои се произведени во фабриката, но не се продаваат во продавница.
3. Најдете колкав процент од црвените производи од фабриката се продаваат во продавница.
4. Компанијата решава да го прекине производството на жолти производи поради промена во модни трендови. Треба да се отстранат сите вакви производи од табелата FactoryProduct.
5. Најдете ги производите кои сеуште се продаваат во продавница, но фабриката повеќе не ги произведува.
6. Вратете ги сите производи (од фабриката и продавницата) кои компанијата ги има на располагање.
7. Вратете ја разликата помеѓу цената на производот во фабриката и продавницата за производите кои ги има на двете места, или цената на производот во фабриката ако тој производ го има само во фабриката, или цената на производот во продавницата ако тој производ го има само во продавницата.
8. Зголемете ја двојно количината на залихата на сини производи.

***РЕШЕНИЕ*:**

Најпрво ќе ги креираме табелите на базата и ќе ги пополниме. Вредностите кои ги внесуваме овде се произволни и имаат единствено показна цел:

```
create table Model(
Id int identity primary key,
Name nvarchar(100) not null,
Description nvarchar(200) null)
```

Ограничувањето `identity` за даден атрибут значи дека истиот ќе се зголемува автоматски (MSSQL синтакса). Во комбинација со `primary key` значи дека атрибутот е примарен клуч и дека истиот ќе се зголемува за еден за секоја нова торка која се внесува во табелата.

```
CREATE TABLE Size(
Id INT IDENTITY PRIMARY KEY,
Name NVARCHAR(100) NOT NULL,
Description NVARCHAR(200) NULL)


CREATE TABLE Color(
Id INT IDENTITY PRIMARY KEY,
Name NVARCHAR(100) NOT NULL)


CREATE TABLE StoreProduct(
Id INT IDENTITY PRIMARY KEY,
ModelId INT NULL REFERENCES Model(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
SizeId INT NULL REFERENCES Size(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
ColorId INT NULL REFERENCES Color(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
RetailPrice FLOAT NOT NULL DEFAULT 0,
ItemsOnStock INT NOT NULL DEFAULT 0)


CREATE TABLE FactoryProduct(
Id INT IDENTITY PRIMARY KEY,
ModelId INT NULL REFERENCES Model(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
SizeId INT NULL REFERENCES Size(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
ColorId INT NULL REFERENCES Color(Id) ON DELETE SET NULL ON
UPDATE CASCADE,
FactoryPrice FLOAT NOT NULL DEFAULT 0)


INSERT INTO Model(Name)
VALUES ('Jeans'), ('Shirt')

INSERT INTO Color(Name)
VALUES ('Blue'), ('Yellow'), ('Red')

INSERT INTO Size(Name)
VALUES ('S'), ('M'), ('L')


SELECT * FROM Model
SELECT * FROM Color
SELECT * FROM Size
SELECT * FROM FactoryProduct
SELECT * FROM StoreProduct

-- со последните наредби ќе ја добиеме моменталната состојба на базата
```

```
-- 1. Табелата FactoryProduct е празна и треба да се пополни со сите можни
комбинации на модели, големини и бои за кои компанијата води информација.

     insert into FactoryProduct(ModelId,SizeId,ColorId)
     select m.Id,s.Id,c.Id
     from Model m cross join Size s cross join Color c

-- Атрибутот за цената на производот ќе го пополниме со произволна
формула (после последната наредба сите цени се 0)

     update FactoryProduct
     set FactoryPrice=50*ModelId*SizeId*ColorId

     select * from FactoryProduct

-- Дополнително заради показни причини ќе ја пополниме и табелата
StoreProduct со парцијално реплицирање на торките од табелата
FactoryProduct.

     insert into
     StoreProduct(ModelId,SizeId,ColorId,RetailPrice,ItemsOnStock)
     select ModelId,SizeId,ColorId, 100*ModelId*SizeId*ColorId
     ,ModelId+SizeId+ColorId
     from FactoryProduct fp
     where (ModelId+SizeId+ColorId) % 3 = 0

     select * from StoreProduct


-- 2. Вратете ги сите информации за производи кои се произведени во
фабриката, но не се продаваат во продавница.

     create view FactoryProductsNotInStore as
     select fp.*
     from FactoryProduct fp left outer join StoreProduct sp on
          fp.ModelId=sp.ModelId and fp.SizeId=sp.SizeId and
          fp.ColorId=sp.ColorId
     where sp.Id is null

     select * from FactoryProductsNotInStore


--3. Најдете колкав процент од црвените производи од фабриката се
продаваат во продавница.

     create view RedProducts as
     select Id
     from Color
     where Color.Name='Red'
```

```
select (select count(sp.*) from StoreProduct sp inner join
RedProducts rp on sp.ColorId=rp.Id)*1.0/( select count(fp.*)
from FactoryProduct fp inner join RedProducts rp on
fp.ColorId=rp.Id)
```

```
--4. Компанијата решава да го прекине производството на жолти производи
поради промена во модни трендови. Треба да се отстранат сите вакви
производи од табелата FactoryProduct.

delete from FactoryProduct
where ColorId in (select Id from Color c where
c.Name='yellow')
```

```
--5. Најдете ги производите кои сеуште се продаваат во продавница, но
фабриката повеќе не ги произведува.

select sp.*
from FactoryProduct fp right outer join StoreProduct sp on
     fp.ModelId=sp.ModelId and fp.SizeId=sp.SizeId and
     fp.ColorId=sp.ColorId
where fp.Id is null
```

```
--6. Вратете ги сите производи (од фабриката и продавницата) кои
компанијата ги има на располагање.

-- без дупликати
select fp.ModelId,fp.ColorId,fp.SizeId
from FactoryProduct fp
union
select sp.ModelId,sp.ColorId,sp.SizeId
from StoreProduct sp

-- со дупликати
select fp.ModelId,fp.ColorId,fp.SizeId
from FactoryProduct fp
union all
select sp.ModelId,sp.ColorId,sp.SizeId
from StoreProduct sp
```

Наредбата `union all` за разлика од `union` не ги отстранува дупликатите од резултатот (можете да ја гледате како аналогија на конкатенација на резултатите од две прашања).

```
--7. Вратете ја разликата помеѓу цената на производот во фабриката и
продавницата за производите кои ги има на двете места, или цената на
производот во фабриката ако тој производ го има само во фабриката, или
цената на производот во продавницата ако тој производ го има само во
продавницата.
```

```
select ISNULL(fp.ModelId,sp.ModelId)
ModelId,ISNULL(fp.SizeId,sp.SizeId)
SizeId,ISNULL(fp.SizeId,sp.SizeId) SizeId,
     case when fp.Id is null then sp.RetailPrice
          when sp.Id is null then fp.FactoryPrice
          else sp.RetailPrice-fp.FactoryPrice end 'Price or
Difference',
     case when fp.Id is null then 'Prodazna cena'
          when sp.Id is null then 'Fabricka cena'
          else 'Razlika' end Tip
from FactoryProduct fp full outer join StoreProduct sp on
     fp.ModelId=sp.ModelId and fp.SizeId=sp.SizeId and
fp.ColorId=sp.ColorId
```

Th

Функцијата CASE ја има следната синтакса:

```
CASE expression
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    WHEN conditionN THEN resultN
    ELSE result
END
```

и се однесува идентично како и SWITCH функцијата во C т.е. претставува избор од повеќе можности, при што наведените услови се испитуваат последователно како што се наведени и секој следен услов ги исклучува сите претходни. Ако ниеден услов не е исполнет тогаш резултат ќе биде оној наведен во можноста ELSE. Ако не е наведена можноста ELSE и притоа ниту еден услов не е исполнет, тогаш функцијата враќа NULL.

```
--8. Зголемете ја двојно количината на залихата на сини производи.

-- Да видиме како изгледа базата пред промената

     select sp.*,c.Name
     from StoreProduct sp inner join Color c on sp.ColorId=c.Id
     order by c.Name,sp.Id

-- Ја правиме промената

     update StoreProduct
     set ItemsOnStock=ItemsOnStock*2
     where ColorId = (select Id from Color where Color.Name='Blue')
```

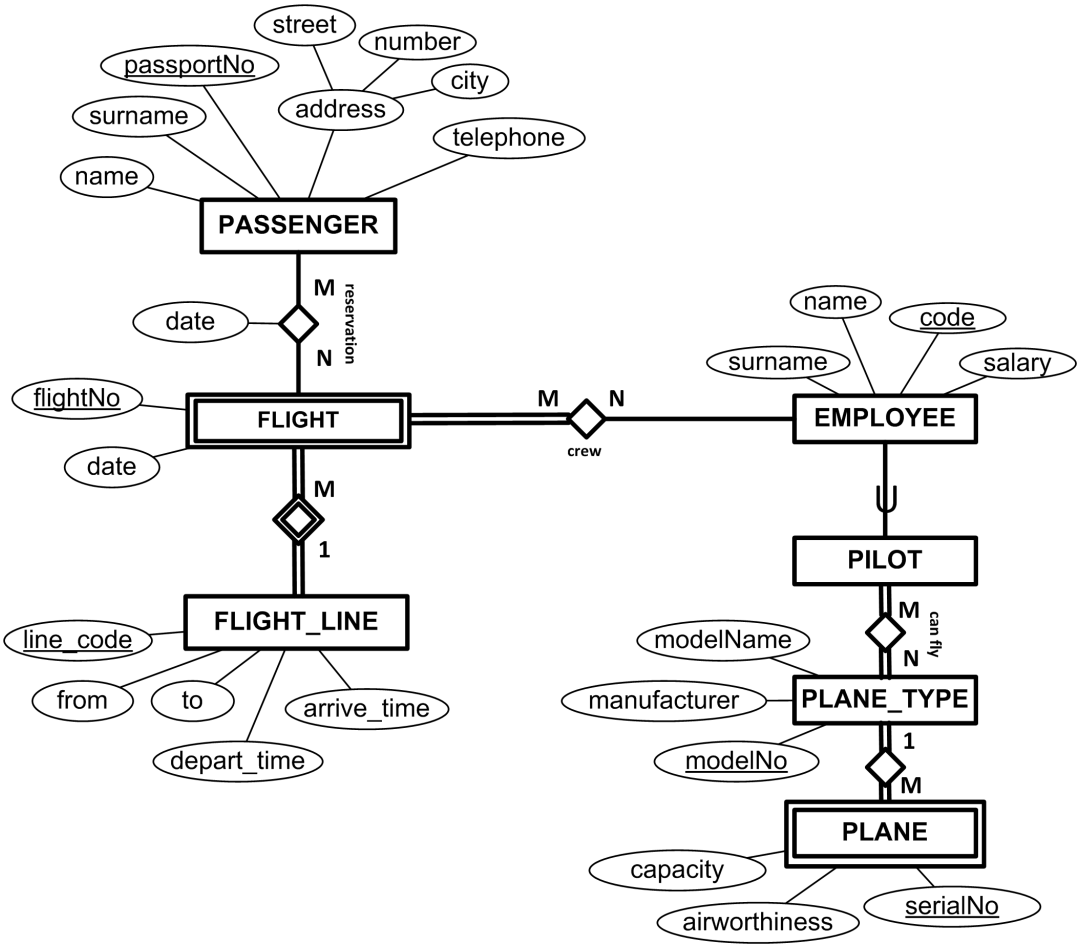
Кај голем број на комерцијални DBMS (MSSQL, PostgreSQL, …) дополнителните табели кои ни се потребни за условот на UPDATE наредбата можат да се наведат и во опционален FROM дел, веднаш после SET. Овде можат да се наведат и повеќе табели и истите да се спојуваат како во FROM делот на било кој друг SQL прашалник. Претходната наредба би била:

```
update StoreProduct
set ItemsOnStock=ItemsOnStock*2
from Color
where Color.Id=StoreProduct.ColorId and Color.Name='Blue'


-- Како изгледа базата после промената

select sp.*,c.Name
from StoreProduct sp inner join Color c on sp.ColorId=c.Id
order by c.Name,sp.Id
```

ЗАДАЧА 3. Нека ја имаме следната база на податоци составена од следните табели:

PASSENGER (passportNo, name, surname, street, number, city, telephone)
FLIGHT_LINE (line_code, from, to, depart_time, arrive_time)
FLIGHT (line_code*, flightNo, date)
RESERVATION (passportNo*, line_code*, flightNo*, date)
EMPLOYEE (code, name, surname, salary)
CREW (line_code*, flightNo*, employeeCode*)
PILOT (code*)
PLANE_TYPE (modelNo, manufacturer, modelName)
PLANE (serialNo, modelNo*, capacity, airworthiness)
CAN_FLY (pilotCode*, modelNo*)

која одговара на следниот ЕР дијаграм:



Дајте ги SQL изразите кои одговараат на следните барања:

1. Најдете ги имињата на сите патници кои немаат полетување на 25.12.2015.
2. Прикажете го вкупниот број на полетувања во кои учествувал секој пилот, вклучително и оние пилоти кои немаат ниту едно полетување.
3. Најдете ги броевите на пасоши на патниците кои немале полетување управувано од пилотот со код 23412.

***РЕШЕНИЕ*:**

1.
```
-- Решение 1: Со користење на привремена табела

  SELECT R.passportNo, R.flightNo
  INTO #RES_FLIGHT
  FROM RESERVATION R
  INNER JOIN FLIGHT FL ON R.flightNo = FL.flightNo
  WHERE FL.date='2015-12-25'

  SELECT PA.name
  FROM PASSENGER PA LEFT JOIN
  #RES_FLIGHT RF
  ON RF.passportNo=PA.passportNo
  WHERE RF.flightNo is NULL
```

```
-- Решение 2: Со користење на CTE

 WITH RES_FLIGHT AS
( SELECT R.passportNo, R.flightNo
  FROM RESERVATION R
       INNER JOIN FLIGHT FL ON R.flightNo = FL.flightNo
   WHERE FL.date='2015-12-25')

SELECT PA.name
FROM PASSENGER PA LEFT JOIN RES_FLIGHT RF
     ON RF.passportNo=PA.passportNo
WHERE RF.flightNo is NULL


-- За овој прашалник не треба да се користи поглед затоа што условот е
премногу специфичен и мала е веројатноста дека ќе имаме повеќекратна
употреба
```

2.
```
SELECT P.code, COUNT(C.line_code)
FROM CREW C RIGHT JOIN PILOT P ON P.code=C.employeeCode
GROUP BY P.code
ORDER BY P.code
```

3.
```
CREATE VIEW PILOT_FLIGHT(code,line_code,flightNo)
SELECT P.code, C.line_code, C.flightNo
FROM CREW C INNER JOIN PILOT P ON P.code=C.employeeCode


CREATE VIEW PASSENGER_FLIGHT(passportNo,line_code,flightNo)
SELECT P.passportNo,R.line_code,R.flightNo
FROM PASSENGER P INNER JOIN RESERVATION R ON
     P.passportNo=R.passportNo


SELECT PAF.passportNo
FROM PASSENGER_FLIGHT PAF LEFT JOIN PILOT_FLIGHT PIF ON
     PAF.line_code=PIF.line_code AND PAF.flightNo=PIF.flightNo
WHERE PIF.code=23412
GROUP BY PAF.passportNo
HAVING COUNT(PIF.code)=0
 -- Последното може да се реши и со is NULL проверка
```