# Guião 7

## 7.1

**a)** Está na primeira forma normal, pois não tem atributos compostos, multivalor ou nested relations. No entanto, como tem dependências parciais (*Afiliacao_autor* depende apenas de *Nome_autor*), não está na segunda forma normal.

**b)** Para ficar na 3FN, será necessário remover dependências parciais e dependências transitivas do cenário original:

Livro (Titulo_Livro, Nome_Autor, Afiliacao_Autor, Tipo_Livro, Preco, NoPaginas, Editor, Endereco_Editor, Ano_Publicacao)

Titulo_Livro, Nome_Autor -> Editor, Tipo_Livro, NoPaginas, Ano_Publicacao
Tipo_Livro, NoPaginas -> Preco
Nome_Autor -> Afiliacao_Autor
Editor -> Endereco_Editor

1. Criar nova relação para armazenar dados do autor, com *Nome_Autor* como chave primária, e uma nova relação análoga para os dados do editor, com *Editor* como chave primária. A relação está agora na 2FN.

   Livro (Titulo_Livro, Nome_Autor, Tipo_Livro, Preco, NoPaginas, Editor, Ano_Publicacao)
   Titulo_Livro, Nome_Autor -> Editor, Tipo_Livro, NoPaginas, Ano_Publicacao
   Tipo_Livro, NoPaginas -> Preco

   Autor(Nome_Autor, Afiliacao_Autor)
   Nome_Autor -> Afiliacao_Autor

   Editor(Editor, Endereco_Editor)
   Editor -> Endereco_Editor

2. Criar nova relação com *Tipo_Livro* e *NoPaginas* como chave primária, para remover a dependência transitiva do atributo *Preco*. A relação está agora na 3FN.

Livro (Titulo_Livro, Nome_Autor, Tipo_Livro, NoPaginas, Editor, Ano_Publicacao)
Titulo_Livro, Nome_Autor -> Editor, Tipo_Livro, NoPaginas, Ano_Publicacao

PrecoLivro (Tipo_Livro, NoPaginas, Preco)
Tipo_Livro, NoPaginas -> Preco

Autor(Nome_Autor, Afiliacao_Autor)
Nome_Autor -> Afiliacao_Autor

Editor(Editor, Endereco_Editor)
Editor -> Endereco_Editor

# 7.2

**a)** {A, B}

**b)** Para ficar na 2FN, será necessário remover atributos compostos, multivalor, nested relations e dependências parciais do cenário original:

Relação R={A,B,C,D,E,F,G,H,I,J}, com as dependências { {A, B}→{C}, {A}→{D, E}, {B}→{F}, {F}→{G, H}, {D}→{I, J} }

1. Criar novas relações para armazenar dados do que dependem exclusivamente de A ou B.

R={A,B,C}, com as dependências { {A, B}→{C} }

R1={A,D,E,I,J}, com as dependências { {A}→{D, E}, {D}→{I, J} }

R2={B,F,G,H}, com as dependências { {B}→{F}, {F}→{G, H} }

**c)** Para ficar na 3FN, será necessário remover ainda as dependências transitivas da 2FN:

R={A,B,C}, com as dependências { {A, B}→{C} }
R1={A,D,E,I,J}, com as dependências { {A}→{D, E}, {D}→{I, J} }
R2={B,F,G,H}, com as dependências { {B}→{F}, {F}→{G, H} }

1. Criar novas relações para remover as dependências transitivas de D e F.

R={A,B,C}, com as dependências { {A, B}→{C} }

R1={A,D,E}, com as dependências { {A}→{D, E} }

R3={D,I,J}, com as dependências { {D}→{I, J} }

R2={B,F}, com as dependências { {B}→{F} }

R4={F,G,H}, com as dependências { {F}→{G, H} }

# 7.3

**a)** {A,B}

**b)** Para decompor R até 3FN é necessário remover as dependências transitivas.

> R={A,B,C,D,E} com as dependências funcionais F={ {A, B}→{C,D,E}, {D}→{ E}, {C}→{A}}

1. Criar novas relações para remover as dependências transitivas de C e D.

> R={<u>A</u>,<u>B</u>,C,D,E} com as dependências funcionais {A, B}→{C,D,E}
> R={<u>D</u>,E} com as dependências funcionais {D}→{E}
> R={<u>C</u>,A} com a dependências funcionais {C}→{A}

**c)** Para ficar na BCNF, será necessário garantir que todos os atributos dependem funcionalmente apenas de toda a chave da relação:

> R={<u>A</u>,<u>B</u>,C,D,E} com as dependências funcionais {A, B}→{C,D,E}
> R={<u>D</u>,E} com as dependências funcionais {D}→{E}
> R={<u>C</u>,A} com a dependências funcionais {C}→{A}

1. Modificar as relações para ficarem na BCNF. Perdeu-se a relação {A, B}→{D}

> R={<u>B</u>,C} com as dependências funcionais {B}→{C}
> R={<u>D</u>,E} com as dependências funcionais {D}→{E}
> R={<u>C</u>,A} com a dependências funcionais {C}→{A}

# 7.4

**a)** {A, B}

**b)** Para ficar na 2FN, será necessário remover atributos compostos, multivalor, nested relations e dependências parciais do cenário original:

Relação R={A,B,C,D,E}, com as dependências { {A, B}→{C,D,E}, {A}→{C}, {C}→{D} }

1. Criar novas relações para armazenar os dados do que dependem exclusivamente de A.

   R={A,B,C,D,E}, com as dependências { {A, B}→{C,D,E}, {C}→{D} }
   R1={A,C,D}, com as dependências { {A}→{C}, {C}→{D} }

**c)** Para ficar na 3FN, será necessário remover ainda as dependências transitivas da 2FN:

R={A,B,C,D,E}, com as dependências { {A, B}→{C,D,E}, {C}→{D} }
R1={A,C,D}, com as dependências { {A}→{C}, {C}→{D} }

1. Criar novas relações para remover a dependência transitiva de D.

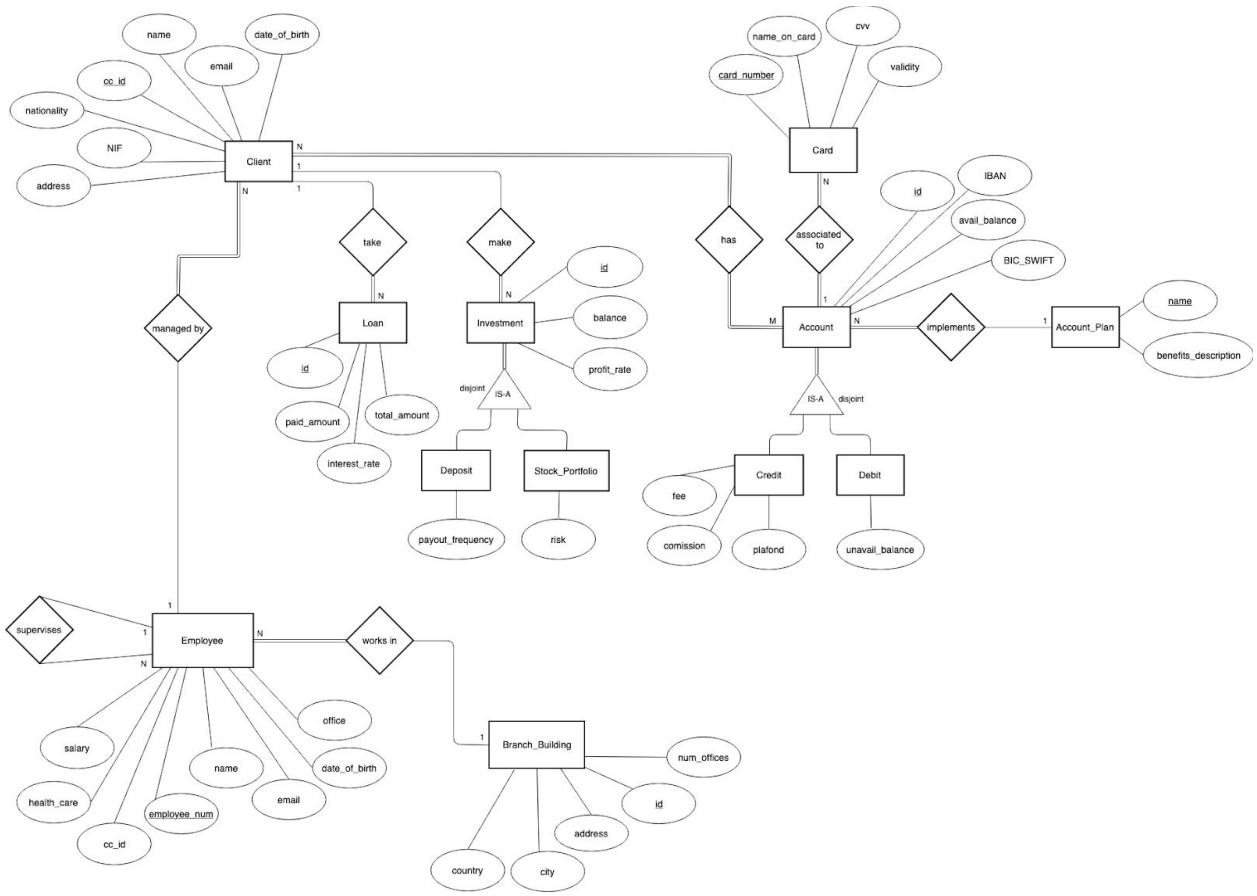
   R={A,B,C,D,E}, com as dependências { {A, B}→{C,D,E} }
   R1={A,C}, com as dependências { {A}→{C} }
   R2={C,D}, com as dependências { {C}→{D} }

**d)** Para ficar na BCNF, será necessário garantir que todos os atributos dependem funcionalmente apenas de toda a chave da relação:

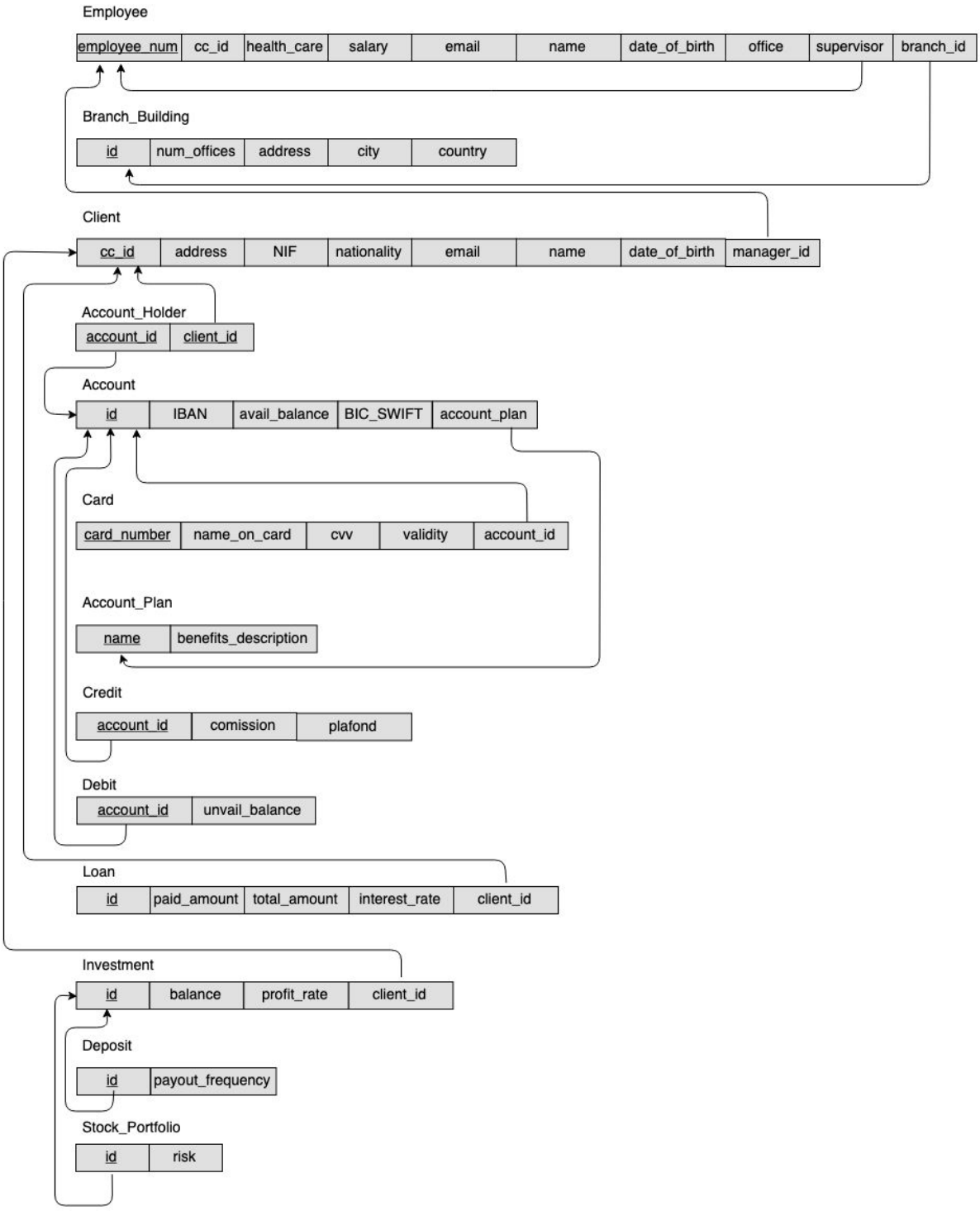
R={A,B,C,D,E}, com as dependências { {A, B}→{C,D,E} }
R1={A,C}, com as dependências { {A}→{C} }
R2={C,D}, com as dependências { {C}→{D} }

# 7.6

**a)** A segunda versão do DER, melhorado em função da sessão de apresentação é a seguinte:



**b)** A segunda versão do ER, melhorado em função da sessão de apresentação é a seguinte:



**c)**

As dependências funcionais de cada relação do ER, convertidas para a forma BCNF, são as seguintes:

i) Employee
   1) *employee_num* -> {name, email, cc_id, health_care, salary, date_of_birth, office, supervisor, branch_id}
   2) *cc_id* -> {name, email, employee_num, health_care, salary, date_of_birth, office, supervisor, branch_id}
   3) *email* -> {name, employee_num, cc_id, health_care, salary, date_of_birth, office, supervisor, branch_id}

ii) Branch_Building
   1) *id* -> {num_offices, address, city, country}

iii) Client
   1) *cc_id* -> {address, NIF, nationality, email, name, date_of_birth, manager_id}
   2) *email* -> {cc_id, address, NIF, nationality, name, date_of_birth, manager_id}
   3) *NIF* -> {cc_id, email, address, nationality, name, date_of_birth, manager_id}

iv) Account_Holder

v) Account
   1) id -> {IBAN, avail_balance, BIC_SWIFT, account_plan}
   2) IBAN -> {id, avail_balance, BIC_SWIFT, account_plan}

vi) Card
   1) card_number -> {name_on_card, cvv, validity, account_id}

vii) Account_Plan
   1) name -> {benefits_description}

viii) Credit
   1) account_id -> {commission, plafond}

ix) Debit
   1) account_id -> {unavail_balance}

x) Loan
   1) id -> {paid_amount, total_amount, interest_rate, client_id

xi) Investment}
   1) id -> {balance, profit_rate, client_id}

xii) Deposit
   1) id -> {payout_frequency}

xiii) Stock Portfolio
   1) id -> {risk}

**d)** O código SQL para a criação das relações/tabelas é o seguinte:

```
CREATE TABLE BRANCH_BUILDING(
	id DECIMAL(5,0) IDENTITY (1,1) NOT NULL,
	city VARCHAR(20) NOT NULL,
	country VARCHAR(20) NOT NULL,
	num_offices DECIMAL(3,0),
	"address" VARCHAR(150),
	PRIMARY KEY(id)
);

CREATE TABLE EMPLOYEE (
	employee_num DECIMAL(10,0) NOT NULL IDENTITY (1,1),
	cc_id DECIMAL(9,0) UNIQUE CHECK(cc_id > 0),
	email VARCHAR(30) UNIQUE NOT NULL,
	"name" VARCHAR(100) NOT NULL,
	salary DECIMAL(10,2),
	date_of_birth DATE,
	health_care VARCHAR(30),
	office DECIMAL(3,0),
	supervisor DECIMAL(10,0),
	branch_id DECIMAL(5,0) NOT NULL,
	PRIMARY KEY(employee_num),
	FOREIGN KEY(supervisor) REFERENCES EMPLOYEE(employee_num),
	FOREIGN KEY(branch_id) REFERENCES BRANCH_BUILDING(id)
);

CREATE TABLE CLIENT(
	cc_id DECIMAL(9,0) NOT NULL CHECK(cc_id > 0),
	NIF DECIMAL(9,0) UNIQUE CHECK(NIF > 0),
	email VARCHAR(30) UNIQUE,
	nationality VARCHAR(25),
	"address" VARCHAR(150),
	"name" VARCHAR(100),
	date_of_birth DATE,
	manager_id DECIMAL(10,0) NOT NULL,
	PRIMARY KEY(cc_id),
	FOREIGN KEY(manager_id) REFERENCES EMPLOYEE(employee_num)
);

CREATE TABLE ACCOUNT_PLAN(
	"name" CHAR(3) NOT NULL,
	benefits_description VARCHAR(80) NOT NULL,
	PRIMARY KEY([name])
);
```

```
CREATE TABLE ACCOUNT (
	id DECIMAL(14,0) NOT NULL IDENTITY (1,1),
	IBAN CHAR(25) UNIQUE,
	avail_balance DECIMAL(15,2),
	BIC_SWIFT CHAR(11) NOT NULL,
	account_plan CHAR(3),
	PRIMARY KEY(id),
	FOREIGN KEY(account_plan) REFERENCES ACCOUNT_PLAN([name])
);

CREATE TABLE ACCOUNT_HOLDER(
	client_id DECIMAL(9,0) NOT NULL,
	account_id DECIMAL(14,0) NOT NULL,
	PRIMARY KEY(client_id, account_id),
	FOREIGN KEY(client_id) REFERENCES CLIENT(cc_id),
	FOREIGN KEY(account_id) REFERENCES ACCOUNT(id)
);

CREATE TABLE "CARD"(
	card_number DECIMAL(16,0) NOT NULL CHECK(card_number > 0),
	name_on_card VARCHAR(20) NOT NULL,
	cvv DECIMAL(3,0) NOT NULL CHECK(cvv > 0),
	validity DATE NOT NULL,
	account_id DECIMAL(14,0) NOT NULL,
	PRIMARY KEY(card_number),
	FOREIGN KEY(account_id) REFERENCES ACCOUNT(id)
);

CREATE TABLE CREDIT(
	account_id DECIMAL(14,0) NOT NULL,
	commission DECIMAL(5,2) NOT NULL CHECK(commission > 0),
	plafond DECIMAL(8,2) NOT NULL,
	PRIMARY KEY(account_id),
	FOREIGN KEY(account_id) REFERENCES ACCOUNT(id)
);

CREATE TABLE DEBIT(
	account_id DECIMAL(14,0) NOT NULL,
	unavail_balance DECIMAL(15,2),
	PRIMARY KEY(account_id),
	FOREIGN KEY(account_id) REFERENCES ACCOUNT(id)
);
```

```
CREATE TABLE LOAN(
	id DECIMAL(14,0) NOT NULL IDENTITY (1,1),
	paid_amount DECIMAL(9,2) DEFAULT 0,
	total_amount DECIMAL(9,2) NOT NULL CHECK(total_amount > 0),
	interest_rate DECIMAL(5,2) NOT NULL CHECK(interest_rate > 0),
	client_id DECIMAL(9,0) NOT NULL,
	PRIMARY KEY(id),
	FOREIGN KEY(client_id) REFERENCES CLIENT(cc_id)
);

CREATE TABLE INVESTMENT(
	id DECIMAL(14,0) NOT NULL IDENTITY (1,1),
	balance DECIMAL(15,2) NOT NULL,
	profit_rate DECIMAL(5,2) NOT NULL CHECK(profit_rate > 0),
	client_id DECIMAL(9,0) NOT NULL,
	PRIMARY KEY(id),
	FOREIGN KEY(client_id) REFERENCES CLIENT(cc_id)
);

CREATE TABLE DEPOSIT(
	id DECIMAL(14,0) NOT NULL,
	payout_frequency DECIMAL(3,0) NOT NULL CHECK(payout_frequency > 0),
	PRIMARY KEY(id),
	FOREIGN KEY(id) REFERENCES INVESTMENT(id)
);

CREATE TABLE STOCK_PORTFOLIO(
	id DECIMAL(14,0) NOT NULL,
	risk DECIMAL(5,2) NOT NULL CHECK(risk > 0),
	PRIMARY KEY(id),
	FOREIGN KEY(id) REFERENCES INVESTMENT(id)
);
```

**e)** O código SQL para a inserção de dados nas tabelas encontra-se no ficheiro `ex6/insercoes.sql`.

**f)** As consultas necessárias para o sistema ManABank incluem:

1. Consultar a quantidade total de bens e obrigações de um cliente.

```
SELECT c.cc_id, c.[name], SUM(l.total_amount) - SUM(l.paid_amount) AS obligations, SUM(i.balance) AS
goods
FROM INVESTMENT as i
        JOIN CLIENT AS c ON c.cc_id = i.client_id
        JOIN LOAN as l ON c.cc_id = l.client_id
GROUP BY c.cc_id, c.[name];
```

2. Consultar o número de trabalhadores num pólo (branch) do banco.

```
SELECT b.id, b.city, b.country, COUNT(*)
FROM EMPLOYEE AS e
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country;
```

3. Obter o salário mínimo e máximo de trabalhadores por pólo.

```
SELECT b.id, b.city, b.country, MIN(e.salary) AS minimum_wages, MAX(e.salary) AS maximum_wages
FROM EMPLOYEE AS e
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country;
```

4. Saber quantos titulares cada dada conta tem.

```
SELECT account_id, COUNT(*) AS num_titulares
FROM ACCOUNT_HOLDER
GROUP BY account_id;
```

5. Saber em que cidades e países é que o banco tem pólos.

```
SELECT b.city, b.country
FROM BRANCH_BUILDING AS b
GROUP BY b.city, b.country;
```

6. Saber qual é número de clientes por pólo.

```
SELECT b.id, b.city, b.country, COUNT(*) AS num_clients
FROM CLIENT AS c
        JOIN EMPLOYEE AS e ON e.employee_num = c.manager_id
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country;
```

7. Saber o nível total de despesas em salários por pólo.

```
SELECT  b.id, b.city, b.country, SUM(e.salary) AS total_montly_salary_expenditure
FROM EMPLOYEE AS e
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country;
```

8. Saber o número médio de cartões por cliente.

```
SELECT AVG(num_cards) AS average_card_num
FROM (  SELECT ah.client_id, COUNT(*) AS num_cards
        FROM [CARD] AS c
                JOIN ACCOUNT AS a ON c.account_id = a.id
                JOIN ACCOUNT_HOLDER AS ah ON a.id = ah.account_id
        GROUP BY ah.client_id) AS CARDS_PER_CLIENT;
```

9. Saber quantos clientes cada dado gestor de contas gere.

```
SELECT b.id, b.city, b.country, e.employee_num, e.email, COUNT(*) AS num_clients
FROM CLIENT AS c
        JOIN EMPLOYEE AS e ON e.employee_num = c.manager_id
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country, e.employee_num, e.email;
```

10. Saber quantos clientes têm um empréstimo ativo.

```
SELECT COUNT(client_id) AS num_active_loans
FROM LOAN;

-- Extra: per branch
SELECT b.id, b.city, b.country, COUNT(client_id) AS num_active_loans
FROM LOAN AS l
        JOIN CLIENT AS c ON c.cc_id = l.client_id
        JOIN EMPLOYEE AS e ON c.manager_id = e.employee_num
        JOIN BRANCH_BUILDING AS b ON e.branch_id = b.id
GROUP BY b.id, b.city, b.country;
```