Mock Exam – Answers – Database Fundamental and Applications

General remarks. In this exam you are **NOT** allowed to use a

- calculator
- book
- dictionary

You have two hours for this exam.

PART I: Concepts [30 points]

Question 1. [3 points] Describe how a theta join works in terms of relations, attributes, and tuples. Now, given relations $R(A, B)$ and $S(B, C)$ also describe how a theta join on the condition $R.B = S.B$ differs from a natural join.

Answer. The theta join can be considered as the Cartesian product of two relations, where we consider only a subset of the tuples produced by this product. That is, the tuples that meet the condition specified in the theta join. Now, given relations $R(A, B)$ and $S(B, C)$, a theta join on the condition $R.B = S.B$ differs from a natural join in the sense that the natural join keeps only one of the two attributes $R.B$ and $S.B$ with identical values—the other attribute is projected out. Hence, the schema for the theta join on this condition has attributes $A, R.B, S.B, C$ whereas the natural join has attributes A, B, C.

Question 2. [3 points] In the relational model of data, relations are conceptualised as sets of tuples. SQL, however, treats relations as bags of tuples instead. Explain the differences between sets of tuples and bags of tuples, and give two advantages of the bag-based approach.

Answer. When relations are conceptualised as sets of tuples, duplicates are not permitted, whereas for bags of tuples, a given tuple may occur multiple times. The bag-approach has two important advantages: (i) the elimination of duplicates is a computationally expensive operation that we better make sure not to carry out unless absolutely necessary and (ii) for certain aggregations (e.g. counting the number of tuples for which a certain condition holds true) we actually want to include the duplicates in our aggregation.

Question 3. [3 points] Consider a relation $R(A, B, C, D, E)$ that is decomposed into relations $R_1(A, B, C)$ and $R_2(C, D, E)$. What does it mean for the natural join of R_1 and R_2 not to be lossless?

A. There may be tuples t in R that are not found in $R_1 \bowtie R_2$.

B. There may be tuples t in $R_1 \bowtie R_2$ that are not found in R.

C. Both A and B.

D. There must be tuples t in $R_1 \bowtie R_2$ that are not found in R and, for some instances of R, there may be tuples t in R that are not found in $R_1 \bowtie R_2$.

Answer. When a join is not lossless this says nothing about particular instances of involved relations. When we say the join of R_1 and R_2 is lossless with respect to R, this means that for any instance of R, any tuple t in the join, $R_1 \bowtie R_2$, must also be present in the original relation, R. Hence, when the join is not lossless we say that B holds.

Question 4. [3 points] Describe the difference between relations and relationships.

Answer. Relations are the main data structure involved in the relational model: relations can be conceptualised either as sets or bags of tuples. Relationships, on the other hand, are a central concept in ER diagrams: relationships are used to connect entity sets (which can be conceptualised as abstract objects). E.g. a car (entity of entity set Cars) has a relationship ownerOf which connects it to an owner (entity of entity set Owners). Hence, relationships and relations represent two entirely different concepts. However, when moulding an ER diagram into a relational model, under some circumstance a relationship can in fact be represented as a relation, where the key attributes of each involved entity set are brought together (e.g. license plate no. and owner social-security no.).

Question 5. [3 points] Explain the difference between SQL tables, virtual views, and materialised views.

Answer. Tables (or base tables) are the essence of a SQL database (e.g. tables with Customers, Products, Orders, Invoices, etc.). SQL tables are part of the relational schema of the database (i.e. they are created using the SQL data-definition language or DDL). However, in addition to their schemas being hard-coded in the database, so are their contents—the actual tuples in the base tables are stored by the DBMS. Virtual views and materialised views are also both a part of the relational schema (i.e. they are also created using the SQL DDL). However, both types of views, in terms of the tuples contained therein, are 'derived products', in the sense that the tuples they contain are based on a query of the database. Hence, when one of the base tables changes, so can the content of the view. Now, the tuples of the virtual view are never stored by the DBMS—the tuples of the virtual view are computed when they are needed (e.g. when the view is used instead of a subquery). The tuples of a materialised view, on the other hand, are stored. For a materialised view the question then becomes how to update the stored tuples as the base tables, on which the view is based, change. There are two ways to go about this: (i) recompute all tuples in the materialised view periodically (e.g. once a week) or (ii) update/change the materialised view whenever the underlying base tables are changed. A final note is in order here: a virtual table is simply a table of which the tuples are never stored by the DBMS. Hence, a virtual view is a special case of a virtual table: the virtual view is a special case in the sense that its definition is part of the relational schema. Finally, note that (base) tables, virtual tables, virtual views, and materialised views are all relations.

Question 6. [3 points] When is a relation said to be in Boyce-Codd Normal Form (BCNF)? What is the use of decomposing a relation into this form? Explain what the downsides are, if any, to decomposing a relation into BCNF.

Answer. A relation R is said to be in BCNF when all its non-trivial functional dependencies (FDs) have a superkey on the left-hand side. That is to say, there is no non-trivial FD for which something 'less' than the key 'explains' some other attributes. Hence, a BCNF violation constitutes a scenario where an attribute is 'explained' by something less than the key (i.e. by something that is not necessarily unique), leading to redundancy. A proper BCNF-decomposition algorithm decomposes relation R into $R_1, \ldots, R_m$ such that the FDs, when projected on R_i for $i = 1, \ldots, m$, are no longer BCNF violations, while ensuring the join of R_i for $i = 1, \ldots, m$ is lossless. The first downside of this approach is that some of the original FDs may not be preserved in the decomposition. The second downside of this approach is that when not using the proper BCNF-decomposition algorithm but instead e.g. splitting R into relations with only two attributes, the join of this decomposition need not be lossless.

Note: there was no Question 7 in the mock exam.

Question 8. [3 points] Explain what a dirty read is, and also give both an advantage and disadvantage of allowing dirty reads. You may use practical examples in your explanation.

Answer. Say we have two concurrent transactions, T_1 and T_2. If transaction T_1 is allowed read uncommitted changes to the database, dirty reads can occur. In this case, T_1 can read the database, including changes made by T_2 that have not been committed yet.

Now consider a case where T_2 transfers money to bank account A, and T_1 tries to transfer money from bank account A to C. Now, prior to T_2 taking place, account A had insufficient funds for T_1. However, when dirty reads are permitted, the moment T_2 has transferred money to A, T_1 can take place even though the transfer of T_2 has not been committed yet. Now, if T_1 is committed, but T_2 is rolled back, this leads to an inconsistent state of the database: money has been transferred from A to C, even though A had insufficient funds.

In short, dirty reads can lead to a scenario where things are based on information that is ultimately lost, which is often undesirable, especially for transactions that both read from and write into the database. However, Example 45, in Chapter 6 of GUM, shows a case where the dirty reads have less detrimental effects, and may actually be advantageous, as no seat-reservation transactions has to wait for another transaction to complete, thereby, speeding up the average processing time for booking requests.

Question 9. [3 points] Proper transactions should meet the so-called ACID properties. Explain what these properties are.

Answer.

- A = *atomicity*: transactions should be either completed as a whole or not at all.
- C = *consistency*: transactions should respect the constraints of the database (e.g. `CHECK`s, key constraints, referential-integrity constraints, etc.)
- I = *isolation*: transactions should be carried out at the proper isolation level. This is the only property for which there is something to choose: there are various isolation levels, and depending on how much we care about concurrent transactions possibly affecting the transaction at hand, we set the right isolation level for the given transaction.
- D = *durability*: the result of transactions should be durable. Hence, there should be mechanisms in place to bring the database back into the state it was after all committed transactions e.g. in case of a power outage.

Question 10. [3 points] Explain how indices can help query your tables more efficiently, and also explain what trade-offs are involved when deciding on which and how many (sets of) attributes to set as indices.

Answer. When a set of attributes is set as an index, a tree-like data structure, called a B-tree, is created, in which we can easily find the nodes that correspond to the tuples with a certain value for the attributes that constitute the index. E.g. if `customerID` is an index for some table R, it is easy to find the node in this B-tree that corresponds to `customerID = 282693`.

Each node, in turn, contains pointers, showing where the full data in the corresponding tuples is stored on the storage device. In the example, the DBMS gets the physical location of all tuples `WHERE customerID = 282693` on the storage device, and, hence, knows where to look on this device to retrieve the desired data.

Hence, the overall cost of retrieving data using attributes that constitute an index, is – crudely speaking – equal to price of searching through the B-tree (low costs) plus looking up the full data, based on the pointers to physical locations on the storage device (also at relatively low costs).

However, as updating such a B-tree is rather expensive, and updating existing tuples and inserting new tuples into our table, typically require this B-tree to be updated, it is evident that tables that primarily receive updates and tuples to be inserted would do better with less indices than tables that are primarily being read.

Finally, when deciding on specific indices it makes little sense to set an attribute as index which is hardly ever used in a `WHERE` clause—if we have to choose between setting attribute set A or B as index, we prefer the attribute set that is most often used for selection.

PART II: Applications of theory [30 points]

Question 11. [15 points] Consider relation $R(A, B, C, D, E)$ with the following functional dependencies (FDs): $A \to D$, $BC \to E$, $E \to A$.

Answers to questions specified below.

i) Find all non-trivial, implied FDs with one attribute on the right-hand side.

$\{A\}^+ = \{A, D\}$
$\{B\}^+ = \{B\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
$\{E\}^+ = \{A, D, E\}$
$\{A, B\}^+ = \{A, B, D\}$
$\{A, C\}^+ = \{A, C, D\}$
$\{A, D\}^+ = \{A, D\}$
$\{A, E\}^+ = \{A, D, E\}$
$\{B, C\}^+ = \{A, B, C, D, E\}$
$\{B, D\}^+ = \{B, D\}$
$\{B, E\}^+ = \{A, B, E, D\}$
$\{C, D\}^+ = \{C, D\}$
$\{C, E\}^+ = \{A, C, D, E\}$
$\{D, E\}^+ = \{A, D, E\}$
$\{A, B, C\}^+ = \{A, B, C, D, E\}$
$\{A, B, D\}^+ = \{A, B, D\}$
$\{A, B, E\}^+ = \{A, B, D, E\}$
$\{A, C, D\}^+ = \{A, C, D\}$
$\{A, C, E\}^+ = \{A, C, D, E\}$
$\{A, D, E\}^+ = \{A, D, E\}$
$\{B, C, D\}^+ = \{A, B, C, D, E\}$
$\{B, C, E\}^+ = \{A, B, C, D, E\}$
$\{B, D, E\}^+ = \{A, B, D, E\}$
$\{C, D, E\}^+ = \{A, C, D, E\}$
$\{A, B, C, D\}^+ = \{A, B, C, D, E\}$
$\{A, B, C, E\}^+ = \{A, B, C, D, E\}$
$\{A, B, D, E\}^+ = \{A, B, D, E\}$
$\{A, C, D, E\}^+ = \{A, C, D, E\}$
$\{B, C, D, E\}^+ = \{A, B, C, D, E\}$

The full set of non-trivial FDs with single-attribute right-hand sides is now given by:

$A \to D$
$E \to A$
$E \to D$
$AB \to D$
$AC \to D$
$AE \to D$
$BC \to A$
$BC \to D$
$BC \to E$
$BE \to A$
$BE \to D$
$CE \to A$
$CE \to D$
$DE \to A$
$ABC \to D$
$ABC \to E$
$ABE \to D$
$ACE \to D$
$BCD \to A$
$BCD \to E$
$BCE \to A$
$BCE \to D$
$BDE \to A$
$CDE \to A$
$ABCD \to E$
$ABCE \to D$
$BCDE \to A$

ii) List all keys.

There is only one key: $K = \{B, C\}$

iii) Indicate which FDs violate Boyce-Codd Normal Form.

The following FDs violate BCNF, as their left-hand side does not include a superkey:

- $A \rightarrow D$
- $E \rightarrow A$
- $E \rightarrow D$
- $AB \rightarrow D$
- $AC \rightarrow D$
- $AE \rightarrow D$
- $BE \rightarrow A$
- $BE \rightarrow D$
- $CE \rightarrow A$
- $CE \rightarrow D$
- $DE \rightarrow A$
- $ABE \rightarrow D$
- $ACE \rightarrow D$
- $BDE \rightarrow A$
- $CDE \rightarrow A$

iv) Decompose R into $R_1(B, C, E)$ and $R_2(A, D, E)$. Project all given and derived FDs onto R_1 and R_2.

For $R_1(B, C, E)$ only one FD holds: $BC \rightarrow E$. For $R_2(A, D, E)$ the following FDs hold:

- $A \rightarrow D$
- $E \rightarrow A$
- $E \rightarrow D$
- $AE \rightarrow D$
- $DE \rightarrow A$

v) Are the decomposed relations in BCNF?

R_1 has one key, $K_1 = \{B, C\}$. Hence, it is in BCNF, as its only FD has K_1 on the left-hand side.

R_2 has one key $K_2 = \{E\}$. Hence, it is not in BCNF—it has one FD (i.e. $A \to D$) that does not contain K_2 as subset on the left-hand side.

vi) For the FDs projected onto R_1: find the minimal basis. Do the same for the FDs projected onto R_2.

The minimal basis of FDs for R_1 is simply the single FD itself, i.e. $B_1 = \{BC \to E\}$.

The minimal basis of FDs for R_2 is given by $B_2 = \{A \to D, E \to A\}$.

vii) Based on these minimal bases for the FDs of R_1 and R_2, what set of FDs do they imply for $R_1 \bowtie R_2$? Is this set equivalent to the set of FDs that holds for R?

The FDs in the join, are simply given by the union of B_1 and B_2, and then using the closure algorithm to find all implied FDs. Now, by merely considering $B_1 \cup B_2$, we already see we retrieve the initial three FDs for R that we started with. Hence, the complete set of implied FDs by $B_1 \cup B_2$ will be identical to the complete set of implied FDs for R. Hence, all FDs are preserved in this decomposition.

viii) Use the chase test to assert whether the join of R_1 and R_2 is lossless.

We begin by setting up a tableau, representing an instance of R, where arbitrary tuple $t = (a, b, c, d, e)$, which is for sure present in the join, is 'spread' over as many rows as possible in R, where we aim to show t must actually be a single tuple in R, proving the join is lossless. Initial tableau:

A	B	C	D	E
a_1	b	c	d_1	e
a	b_2	c_2	d	e

$E \to A \Rightarrow$

A	B	C	D	E
a	b	c	d_1	e
a	b_2	c_2	d	e

$A \to D \Rightarrow$

A	B	C	D	E
a	b	c	d	e
a	b_2	c_2	d	e

We see in the final tableau that any tuple t in the join is for sure also a tuple in R. Hence, the join is lossless.

ix) Would there be any advantage to decomposing this relation into third normal form (3NF) using the synthesis algorithm? Explain why or why not.

The big advantage of the 3NF-synthesis algorithm is that it eliminates much redundancy from FDs (though not necessarily all) while preserving for sure all FDs from the original relation. We see that our decomposition of R into R_1 and R_2 also preserves all original FDs in R. Hence, regarding FD preservation, the 3NF-synthesis algorithm provides no advantage here.

However, regarding redundancy, we can observe that the 3NF-synthesis algorithm would have yielded three relations, $R_1(A, D)$, $R_2(E, A)$, and $R_3(B, C, E)$. These three relations preserve FDs and, by virtue of the decomposition algorithm, have a lossless join. Furthermore, we can observe these three relations are all in BCNF. Hence, one could argue that even though our decomposition preserves the FDs and has a lossless join, just like the 3FN-synthesis algorithm, the 3NF approach in this particular case is likely to have less redundancy in the decomposition than our decomposition has.

Question 12. [15 points] The director of a theatre asks you to set up a database system for the sales of tickets to customers for different performances of a show. The director provides you with the following information:

- Each show has a name and producer.
- For each show there are one or more performances (unless the show is cancelled altogether).
- In each performance exactly one show is performed.
- Each performance has a date, time, and hall in which it takes place.
- Each performance has a crew consisting of one or more crew members.
- Each member of the crew has a name, birth date, and role.
- Tickets are sold for each performance.
- Each ticket constitutes a reservation of one seat for one performance.
- Each ticket has a price and a seat number.
- Each ticket can be bought by at most one customer.
- Each customer can hold multiple tickets. Each customer has a name and address in the system.

Answers to questions specified below. Using this information:

i) Draw an appropriate entity-relationship (ER) diagram for this database.

Figure 1 shows a possible ER diagram that meets the set requirements. Note that the question left some room for ambiguity. Whatever you decide e.g. in terms of key attributes should simply make sense and/or be explained.

Note that for Shows, I assumed the name to be the only key attribute. For CrewMembers, name and birthDate constitute the key. For Customers, name and address form the key.

ii) Are there any weak entity sets in your diagram? If so, explain why they are weak entity sets.

Similar to seat reservations on a flight, here the Tickets for seats are a weak ES. That is to say, for a given ticket in Tickets, for a given seatNo., bought at a given price, by a given customer in Customers, there is exactly one corresponding performance in Performances. Moreover, a ticket cannot be identified merely on the basis of seatNo. and/or price. However, seatNo. in combination with the key attributes of the supporting ES Performances, suffice to identify Tickets.

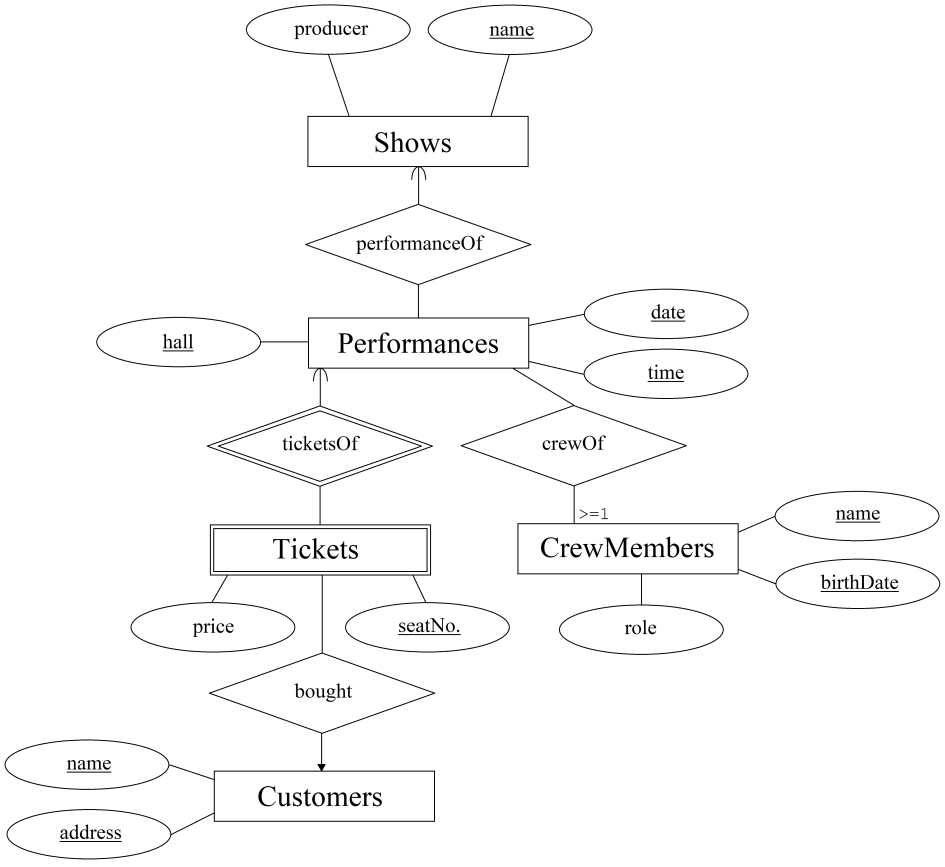

Figure 1. ER diagram following specifications in Question 12, Point i.

iii) What are the referential-integrity constraints?

Obviously, as Tickets is weak ES, it places a referential-integrity constraint on its supporting ES, Performances. Moreover, ES Performances places a referential-integrity constraint on Shows. Finally, Tickets does not place a referential-integrity constraint on Customers—even though the relationship is many-one from Tickets to Customers, the specification allows for there to be tickets without a customer (yet!).

iv) Convert the ER-diagram to a set of relations, in accordance with the rules you have learnt in Ch. 4.

Based on the relevant many-one relationship, and the fact that Customers only has key attributes, we can absorb Customers into our relation for Tickets. Moreover, as Tickets is a weak ES, we must also include the key attributes from Performances.

As Performances places a referential-integrity constraint on Shows, and Shows has an attribute that is not key (i.e. producer), we can only incorporate the relationship performanceOf in Performances.

In addition, we need separate relations for crewOf and crewMembers, as Performances and CrewMembers have a many-many relationship.

Finally, note that even though it appears to be possible to include Performances in the relation for Tickets as a whole (as it has no non-key attributes), there is an important reason to decide against this: the relation for Performances will incorporate the performanceOf relationship and, therefore, from the perspective of Performances, it will have an attribute that is not key for that relation: name of the show. Hence, inclusion of Performances in the relation for Tickets will lead to unnecessary redundancy, while having it as a separate relation will not lead to such redundancy.

Hence, the final set of relations is as follows:

- Performances(<u>hall</u>, <u>date</u>, <u>time</u>, nameShow)
- Shows(<u>name</u>, producer)
- crewOf(<u>hallPerf</u>, <u>datePerf</u>, <u>timePerf</u>, <u>nameCrwMmbr</u>, <u>birthDateCrwMmbr</u>)
- CrewMembers(<u>name</u>, <u>birthDate</u>, role)
- Tickets(<u>seatNo.</u>, <u>hallPerf</u>, <u>datePerf</u>, <u>timePerf</u>, price, nameCustomer, addressCustomer)

v) Setting aside all ESs other than Shows: draw an ER-diagram focussing exclusively on Shows, which allows you to distinguish between a ballet, a musical, and a regular play, where

- a ballet has a fixed crew of one or more ballet dancers as well as an orchestra consisting of multiple musicians,
- a musical has a fixed crew of one or more actors as well as an orchestra consisting of multiple musicians, and
- a regular play has a fixed crew of one or more actors.

Figure 2 shows the appropriate hierarchy for Shows, assuming both Dancers, Musicians, and Actors have only two attributes: name and birthDate, both being key attributes.

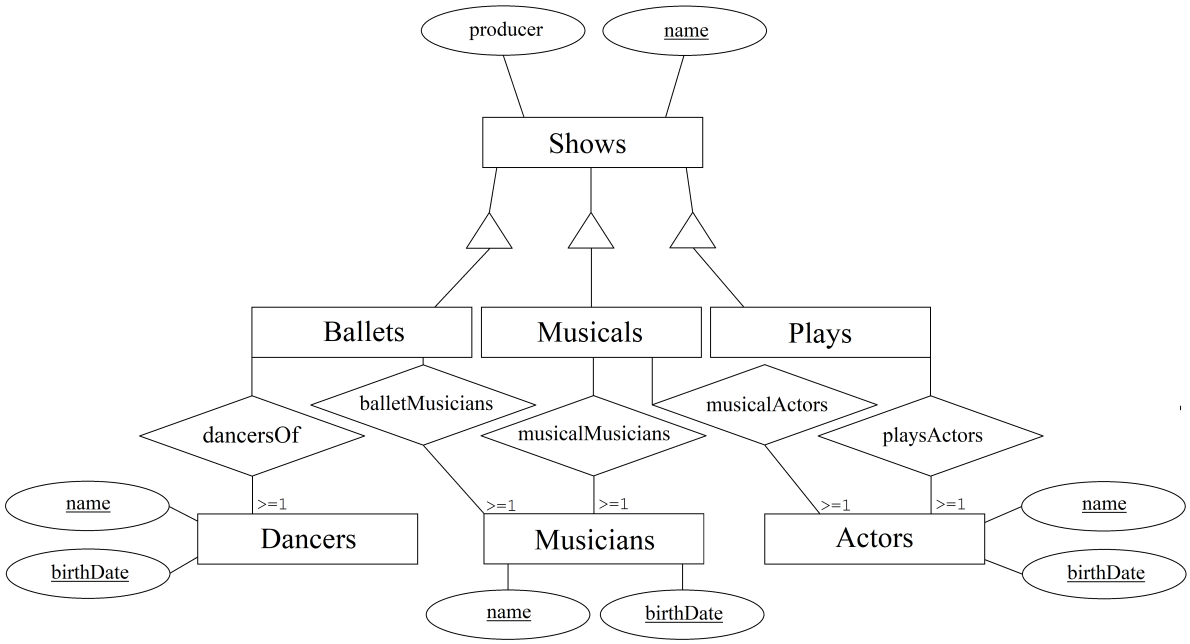

Figure 2. ER diagram following specifications in Question 12, Point v, focussing on the hierarchy for Shows.

PART III: SQL queries [30 points]

We have data from an airline company on three relations, `Flights`, `Seats`, and `Bookings`, shown in Tables 1, 2, and 3 respectively.

flightNo	date	plane	time	dest
1031	2018-12-01	908	09:12	LCY
1032	2018-12-01	908	12:29	RTM
0134	2018-12-02	671	10:09	VIE
0135	2018-12-02	671	15:43	AMS

Table 1. Relation `Flights`

plane	row	number	class
671	1	A	business
671	1	B	business
671	1	E	business
671	1	F	business
671	2	A	business
671	2	B	business
671	2	E	business
671	2	F	business
671	3	A	economy
671	3	B	economy
671	3	E	economy
671	3	F	economy
671	4	A	economy
671	4	B	economy
671	4	E	economy
671	4	F	economy
908	1	A	business
908	1	B	business
908	1	E	business
908	1	F	business
908	2	A	business
908	2	B	business
908	2	E	business
908	2	F	business
908	3	A	economy
908	3	B	economy
908	3	E	economy
908	3	F	economy
908	4	A	economy
908	4	B	economy
908	4	E	economy
908	4	F	economy

Table 2. Relation `Seats`

flightNo	date	seatRow	seatNo	price	customerID
1031	2018-12-01	1	A	800	1209495
1031	2018-12-01	3	A	120	1689312
1031	2018-12-01	3	B	130	3429123
1031	2018-12-01	3	E	110	4292135
1031	2018-12-01	3	F	130	9329401
1031	2018-12-01	4	A	180	3949123
1031	2018-12-01	4	E	140	3126902
1032	2018-12-01	1	B	650	4783013
1032	2018-12-01	2	F	750	4539231
1032	2018-12-01	3	A	110	1263912
1032	2018-12-01	3	B	90	2131230
1032	2018-12-01	3	E	100	4502343
1032	2018-12-01	4	A	120	1839694
1032	2018-12-01	4	E	100	8924922
1032	2018-12-01	4	F	130	7255242
0134	2018-12-02	1	A	1500	7324723
0134	2018-12-02	1	E	1700	2139067
0134	2018-12-02	2	F	1100	6589123
0134	2018-12-02	3	E	210	4589212
0134	2018-12-02	4	B	190	9648324
0135	2018-12-02	1	B	1300	5839149
0135	2018-12-02	2	E	1200	8573271
0135	2018-12-02	3	A	180	5182542
0135	2018-12-02	3	B	180	6581298
0135	2018-12-02	4	A	150	1231205
0135	2018-12-02	4	B	160	8132984
0135	2018-12-02	4	F	170	3210935

Table 3. Relation `Bookings`

Question 13. [10 points] Write a SQL query to compute the total amount money spent on bookings for each flight and each class. Also provide the result of this query.

Answer.

```
SELECT Flights.flightNo, Flights.date, class, SUM(price) AS total
FROM Flights, Seats, Bookings
WHERE Flights.plane = Seats.plane AND Flights.flightNo = Bookings.flightNo
AND Flights.date = Bookings.date AND row = seatRow AND number = seatNumber
GROUP BY Flights.flightNo, Flights.date, class;
```

flightNo	date	class	total
1031	2018-12-01	business	800
1031	2018-12-01	economy	810
1032	2018-12-01	business	1400
1032	2018-12-01	economy	650
0134	2018-12-02	business	4300
0134	2018-12-02	economy	400
0135	2018-12-02	business	2500
0135	2018-12-02	economy	840

Question 14. [10 points] Write a SQL query to compute the number of seats available on each flight, in each class. Also provide the result of this query.

Answer.

```
SELECT flightNo, date, class, COUNT(number) AS availSeats
FROM Flights, Seats
WHERE Flights.plane = Seats.plane
AND (row, number) NOT IN (
SELECT seatRow, seatNo FROM Bookings
WHERE Flights.flightNo = Bookings.flightNo AND Flights.date = Bookings.date
)
GROUP BY flightNo, date, class;
```

flightNo	date	class	availSeats
1031	2018-12-01	business	7
1031	2018-12-01	economy	2
1032	2018-12-01	business	6
1032	2018-12-01	economy	2
0134	2018-12-02	business	5
0134	2018-12-02	economy	6
0135	2018-12-02	business	6
0135	2018-12-02	economy	3

Question 15. [**10 points**] Write a SQL query to show which rows on which flights are still completely empty (i.e. for which there is no single booking). Note that by flight I mean the combination of flightNo and date. Also provide the result of this query.

Answer.

```
SELECT DISTINCT flightNo, date, row
FROM Flights, Seats
WHERE Flights.plane = Seats.plane
AND row NOT IN (
SELECT seatRow FROM Bookings
WHERE Flights.flightNo = Bookings.flightNo AND Flights.date = Bookings.date
);
```

flightNo	date	row
1031	2018-12-01	2