IAT 3
DBMS-18CS53(5 SEM) (AY 2021-22)- CSE

Q1(a): Explain the following (i). Candidate key (ii). Primary Key (iii). Super key(1 MARKS EACH + 1 MARK EXAMPLE)

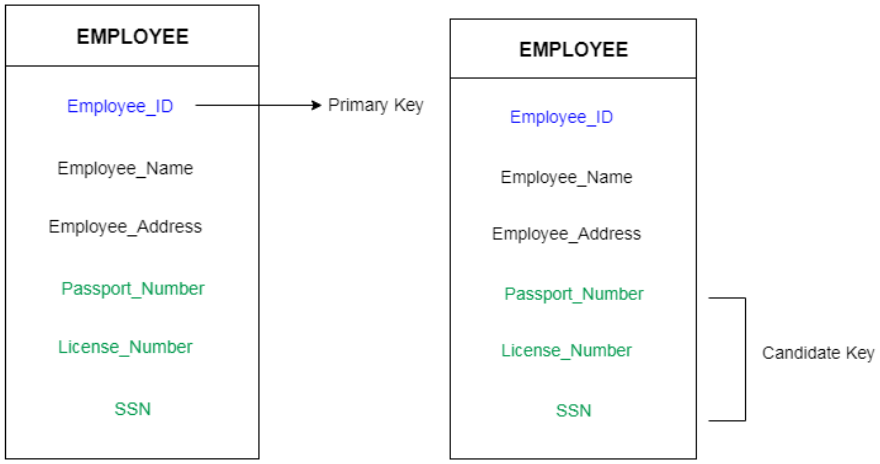
Candidate key:-Candidate key is a single key or a group of multiple keys that uniquely identify rows in a table. A Candidate key is a subset of Super keys and is devoid of any unnecessary attributes that are not important for uniquely identifying tuples. The value for the Candidate key is unique and non-null for all tuples.

Primary KeY:-A primary key, also called a primary keyword, is a key in a relational database that is unique for each record. It is a unique identifier, such as a driver license number, telephone number (including area code), or vehicle identification number (VIN). A relational database must always have one and only one primary key. Primary keys typically appear as columns in relational database tables.

Super key:-We can define a super key as a set of those keys that identify a row or a tuple uniquely. The word super denotes the superiority of a key. Thus, a super key is the superset of a key known as a Candidate key (discussed in the next section). It means a candidate key is obtained from a super key only.

For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

EXample:



Q1:B-Given below are two sets of FD's for a relation R (A, B, C, D, E, H).Are they equivalent? F= {A->C, AC->D, E->AD, E->H} and G= {A->CD, E->AH}--- 6 MARKS

Determining whether F covers G-

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set G

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set F

Step-03:

Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G.
- Thus, we conclude F covers G i.e. $F \supseteq G$.

Determining whether G covers F-

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set F
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set F

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set G
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set G

Step-03:

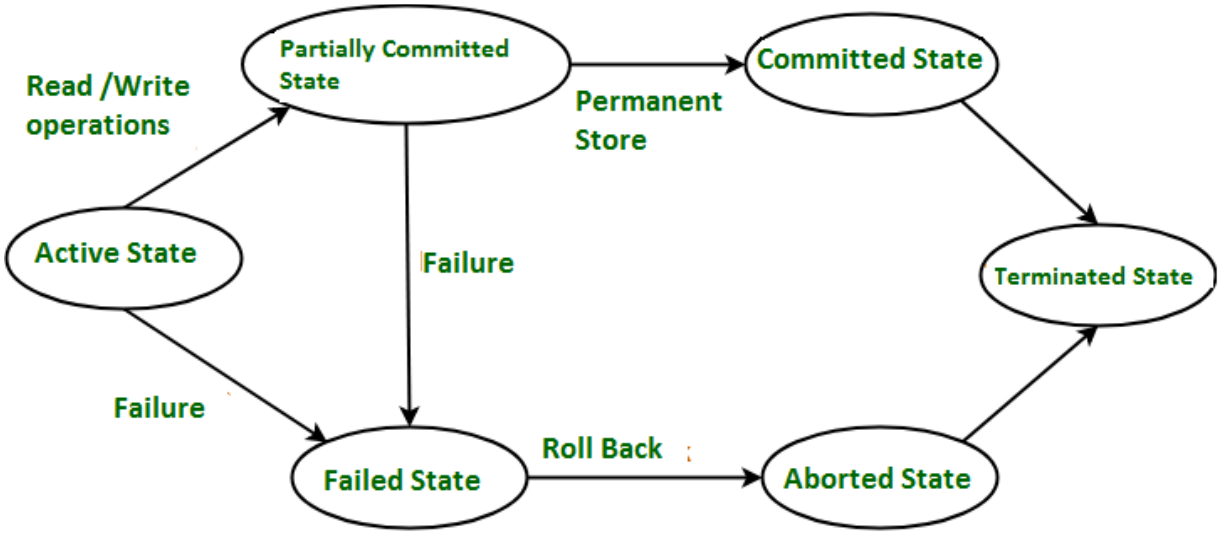
Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set G can determine all the attributes which have been determined by the functional dependencies of set F.
- Thus, we conclude G covers F i.e. G ⊇ F.

Determining whether both F and G cover each other-

- From Step-01, we conclude F covers G.
- From Step-02, we conclude G covers F.
- Thus, we conclude both F and G cover each other i.e. F = G.

Q2:(A)Explain transaction states with a neat diagram. 4 MARKS(2 MARKS- DIAGRAM+ 2 MARKS EXPLANATION)



Transaction States in DBMS

These are different types of Transaction States :

1. **Active State –**

When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

2. **Partially Committed –**

After completion of all the read and write operation the changes are made in main memory or local buffer. If the the changes are made permanent on the Data Base then the state will change to

"committed state" and in case of failure it will go to the "failed state".

3. **Failed State –**

When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Data Base.

4. **Aborted State –**

After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

5. **Committed State –**

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the "terminated state".

6. **Terminated State –**

If there isn't any roll-back or the transaction comes from the "committed state", then the system is consistent and ready for new transaction and the old transaction is terminated.

Q2:B:Write the algorithm to find the minimal cover for a sets of FD's Consider R={A,B,C,D,E,H} .FD's {A->C,AC->D,E->AD,E->H} Find the irreducible cover for this set of FD's(minimal cover).(6 MARKS- 3 MARKS ALGORITHM+ 3 MARKS PROBLEM)

Given:

R={A,B,C,D,E,H} .FD's {A->C,AC->D,E->AD,E->H}

Minimal cover steps:

a. Canonical form:

F: {A → C, AC→ D, E → A, E → D, E → H }

b. Removing extraneous attributes:

Since A→C is present, C in AC→D is extraneous attribute. Hence removing we can remove C from AC→D and replace it with A→D.

We get: F: {A →C, A→ D, E → A, E → D, E → H}

c. Removing redundant FDs:

Since E→A and A→D we get E→D using transitive property, hence E→D is redundant in F and can be removed from F.

F: {A →C, A→ D, E → A, E → H}

Minimal Cover of given F is : F: {A →C, A→ D, E → A, E → H}

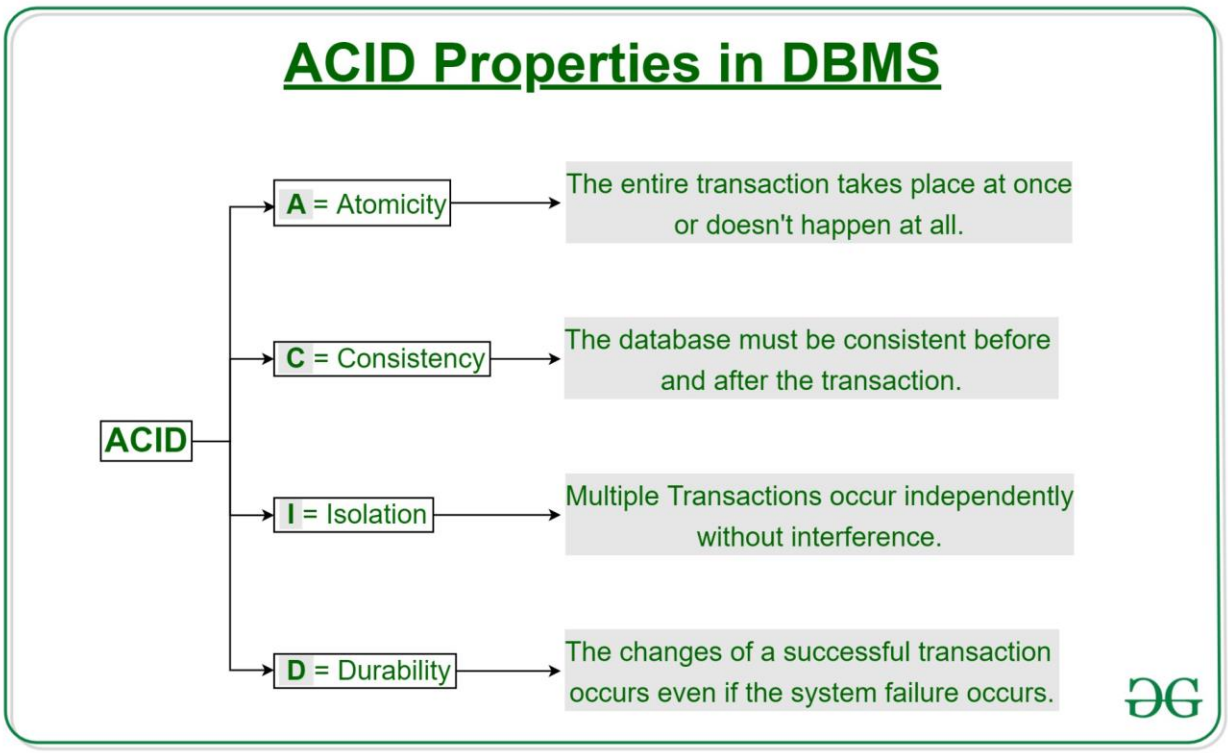
Algorithm:

Input: A set of functional dependencies E.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (**comment**).

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$. (*This places the FDs in a canonical form for subsequent testing*)
3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B that is an element of X
   if $\{ \{F - \{X \rightarrow A\}\} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to F
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
   (*This constitutes removal of an extraneous attribute B contained in the left-hand side X of a functional dependency $X \rightarrow A$ when possible*)
4. For each remaining functional dependency $X \rightarrow A$ in F
   if $\{F - \{X \rightarrow A\}\}$ is equivalent to F,
   then remove $X \rightarrow A$ from F. (*This constitutes removal of a redundant functional dependency $X \rightarrow A$ from F when possible*)

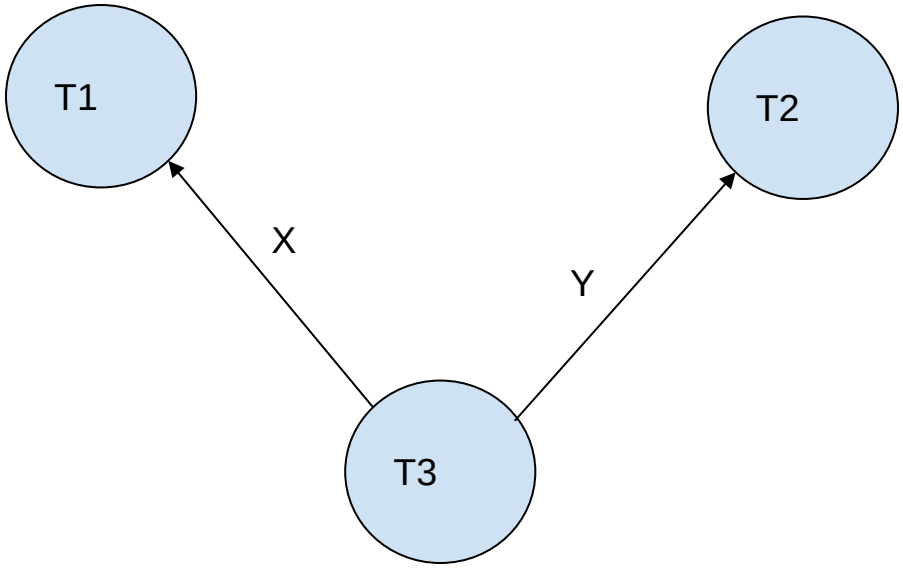
Q3:(A) List and explain ACID Properties. 4 MARKS



Q3:(B)Check whether the given schedule is serializable or not using a precedence graph. Explain with an algorithm. S1:R1(X) R2(Z) R1(Z) R3(X) R3(Y) W1(X) W3(Y) R2(Y) W2(Z) W2(Y) (6 MARKS)- 4 MARKS- ALGORITHM STEP BY STEP EXPLANATION+2 MARKS GRAPH

Algorithm:

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.



Since we have R3(X) and W1(X) as conflicting operation, we add an edge from T3 to T1.
Since we have W3(Y) and R2(Y) as conflicting operation, we add an edge from T3 to T2.
Since there are no more conflicting operations, we don't get any more edges in the graph. Finally, as there is no cycle in the precedence graph, the Schedule is serializabe as below:

T3 → T1 → T2 OR T3→ T2 → T1

Q4:(A)Define Multivalued dependency. Explain 4NF with an example. 4 MARKS: MVD- 2 MARKS+4NF 2 MARKS

A multivalued dependency is best illustrated using an example. ... This is an example of multivalued dependency: An item depends on more than one value. In this example, the course depends on both lecturer and book. Thus, 4NF states that a table should not have more than one of these dependencies.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY

21	**Computer**	**Dancing**
21	**Math**	**Singing**
34	**Chemistry**	**Dancing**
74	**Biology**	**Cricket**
59	**Physics**	**Hockey**

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Q4:(B)Explain the problems that can occur when concurrent transactions are executed. 6 MARKS

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. These problems are commonly referred to as concurrency problems in a database environment. The five concurrency problems that can occur in the database are:

- Temporary Update Problem
- Incorrect Summary Problem
- Lost Update Problem
- Unrepeatable Read Problem
- Phantom Read Problem

These are explained as following below.

Temporary Update Problem:

Temporary update or dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

Example:

T1	T2
read_item(X) X = X - N write_item(X) read_item(Y)	 read_item(X) X = X + M write_item(X)

In the above example, if transaction 1 fails for some reason then X will revert back to its previous value. But transaction 2 has already read the incorrect value of X.

Incorrect Summary Problem:

Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.

Example:

T1	T2
   read_item(X) X = X - N write_item(X) read_item(Y) Y = Y + N write_item(Y)	sum = 0 read_item(A) sum = sum + A read_item(X) sum = sum + X read_item(Y) sum = sum + Y

In the above example, transaction 2 is calculating the sum of some records while transaction 1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.

Lost Update Problem:

In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Example:

T1	T2
read_item(X) X = X + N	X = X + 10 write_item(X)

In the above example, transaction 1 changes the value of X but it gets overwritten by the update done by transaction 2 on X. Therefore, the update done by transaction 1 is lost.

Unrepeatable Read Problem:

The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

Example:

T1	T2
Read(X)	
	Read(X)
Write(X)	
	Read(X)

In the above example, once transaction 2 reads the variable X, a write operation in transaction 1 changes the value of the variable X. Thus, when another read operation is performed by transaction 2, it reads the new value of X which was updated by transaction 1**.**

Q5:Explain informal design guidelines for relation schemas.(10 MARKS)

The Four informal design guidelines that may be used as measures to determine the quality of relation schema design

Guideline 1: Each record in a relation should represent one entity or relationship instance

This guideline is to design a relational schema so that its meaning is easily explained. Another goal of this guideline is to not combine attributes from multiple relationships and entity types into a single relation. Information should not be stored redundantly therefore preventing storage being wasted, only foreign keys should be used when referring to other entities.

CustomerDetails

Customer-Firstname	Customer-Lastname	Customer-Email
Josh	Robb	Josh@email.com
john	doe	john@email.com
Jim	Bates	jim@email.com

VehicleRental

Customer-Email	RentID	Authorised-by	Discount	Renting-Price	VehicleRegistration-No
Josh@email.com	1	Dave	£15	£150	BD51 SMK
Josh@email.com	2	Bob	£0	£129	WR44 FNR
john@email.com	3	Dave	£5	£160	WR44 FNR
jim@email.com	4	Bob	£44	£432	BD51 SMK
jim@email.com	5	Steve	£53	£235	BD51 SMK

Vehicles

Vehicle-Model	Vehicle-Colour	Vehicle-Type	VehicleRegistration-No
Ford	White	Van	WR44 FNR
VW	Red	Campervan	BD51 SMK

A brief and basic overview of the desired database design

Guideline 2: No insertion, deletion, or modification anomalies are present in the relations

This guideline is to help design the database relation schema so that there are no insertion, deletion or modification anomalies present in the relations, improper groupings of attributes into a relation schema will result in wasted storage, insert anomalies, delete anomalies and modification anomalies.

Guideline 3: Relations should be designed such that their records will have as few NULL values as possible

This guideline is to help prevent placing attributes in a database relation which value may frequently be null, however, if nulls are unavoidable they should apply in exceptional cases only and not the greater part of values in a relation. Problems with having null values are as follows: wasting of storage

space, Problems with understanding the meaning of attributes, Problems with applying certain aggregate function and also problems with JOIN operations.

As a result of applying normalisation the schema has got minimal attributes which may be null, the only attribute that may be null is the Discount attribute within the table VehicleRental.

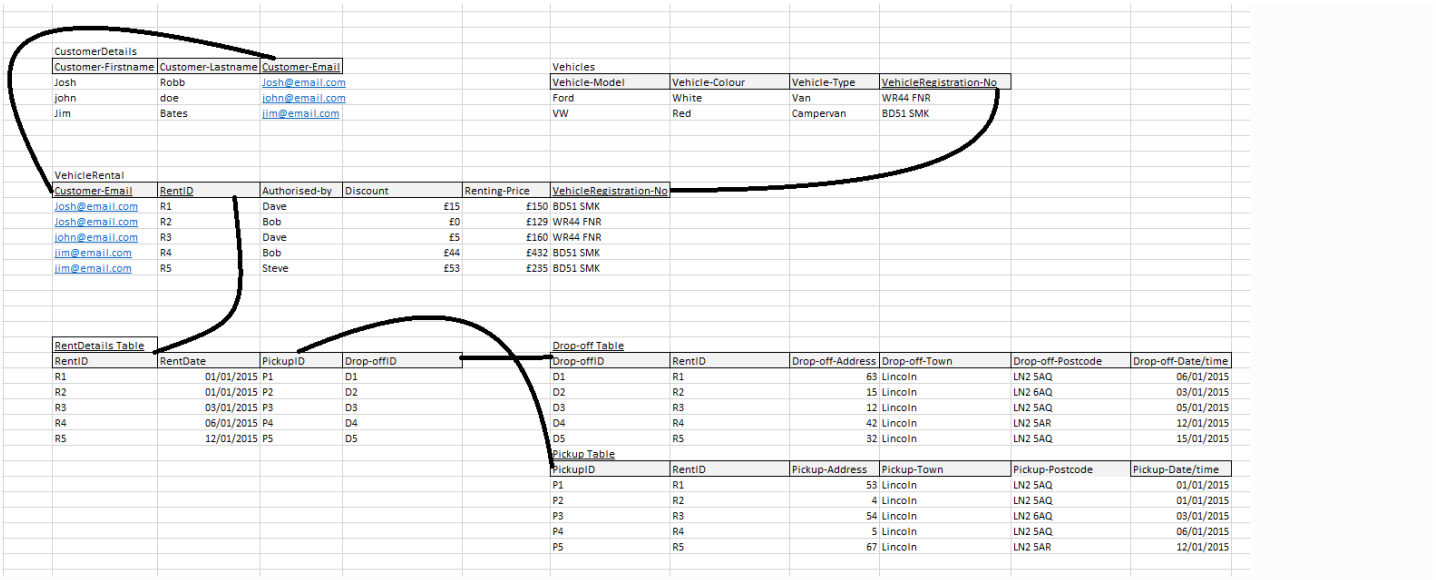
	CustomerDetails Table
	Customer-Firstname
	Customer-Lastname
PK	Customer-Email
	VehicleRental Table
PK	Customer-Email
PK	RentID
	Authorised-by
	Discount
	Renting-Price
PK	VehicleRegistration-No
	Vehicle Table
	Vehicle-Model
	Vehicle-Colour
	Vehicle-Type
PK	VehicleRegistration-No
	RentDetails Table
PK	RentID
	RentDate
FK	PickupID
FK	Drop-offID

	Drop-off Table
PK	Drop-offID
FK	RentID
	Drop-off-Address
	Drop-off-Town
	Drop-off-Postcode
	Drop-off-Date/time
	Pickup Table
PK	PickupID
FK	RentID
	Pickup-Address
	Pickup-Town
	Pickup-Postcode
	Pickup-Date/time

Primary and foreign keys are being identified with thee database design

Guideline 4: Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious records are generated

This guideline is to help design a relational schema so that it can be joined with equal conditions of attributes that are either primary keys or foreign keys in ways that will guarantee that no spurious records are generated. To stop generations spurious records, relations should no contain matching attributes other than the foreign/primary key relation, however, if such relations are unavoidable they should not be joined on such attributes because the join may create spurious records.



CustomerDetails

Customer-Firstname	Customer-Lastname	Customer-Email
Josh	Robb	Josh@email.com
john	doe	john@email.com
Jim	Bates	jim@email.com

Vehicles

Vehicle-Model	Vehicle-Colour	Vehicle-Type	VehicleRegistration-No
Ford	White	Van	WR44 FNR
VW	Red	Campervan	BD51 SMK

VehicleRental

Customer-Email	RentID	Authorised-by	Discount	Renting-Price	VehicleRegistration-No
Josh@email.com	R1	Dave	£15	£150	BD51 SMK
Josh@email.com	R2	Bob	£0	£129	WR44 FNR
john@email.com	R3	Dave	£5	£160	WR44 FNR
jim@email.com	R4	Bob	£44	£432	BD51 SMK
jim@email.com	R5	Steve	£53	£235	BD51 SMK

RentDetails Table

RentID	RentDate	PickupID	Drop-offID
R1	01/01/2015	P1	D1
R2	01/01/2015	P2	D2
R3	03/01/2015	P3	D3
R4	06/01/2015	P4	D4
R5	12/01/2015	P5	D5

Drop-off Table

Drop-offID	RentID	Drop-off-Address	Drop-off-Town	Drop-off-Postcode	Drop-off-Date/time
D1	R1	63	Lincoln	LN2 5AQ	06/01/2015
D2	R2	15	Lincoln	LN2 6AQ	03/01/2015
D3	R3	12	Lincoln	LN2 5AQ	05/01/2015
D4	R4	42	Lincoln	LN2 5AR	12/01/2015
D5	R5	32	Lincoln	LN2 5AQ	15/01/2015

Pickup Table

PickupID	RentID	Pickup-Address	Pickup-Town	Pickup-Postcode	Pickup-Date/time
P1	R1	53	Lincoln	LN2 5AQ	01/01/2015
P2	R2	4	Lincoln	LN2 5AQ	01/01/2015
P3	R3	54	Lincoln	LN2 6AQ	03/01/2015
P4	R4	5	Lincoln	LN2 5AQ	06/01/2015
P5	R5	67	Lincoln	LN2 5AR	12/01/2015

Q6:Consider R = {A,B,C,D,E,F) and D={R1,R2,R3} where, R1(A,B) R2(C,D,E) R3(A,C,E) The following functional dependencies hold on relation R.

F = {A→B; C→DE; AC→F}. Check if D is lossless decomposition. (10 MARKS)

Initial matrix:

	A	B	C	D	E	F
R1						
R2						
R3						

Step 1:

	A	B	C	D	E	F
R1	a_1	a_2	• b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

Step2:

	A	B	C	D	E	F
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

Since last row is completely made up of “a” symbol, we conclude that the decomposition D={R1,R2,R3} is lossless decomposition.

Q7:What is the need of normalization? Explain the first, second and third normal form with examples. (10 MARKS)(NORMALIZATION- 2 MARKS, 2 MARKS EACH NORMAL FORM+2 MARKS EXAMPLE)

Normalization is a technique for organizing data in a database. It is important that a database is normalized to minimize redundancy (duplicate data) and to ensure only related data is stored in each table. It also prevents any anomalies caused due to database modifications such as insertions, deletions, and updates.

First normal form

To be in first normal form (1NF), a table must have the following qualities:

- The table is two-dimensional with rows and columns.
- Each row contains data that pertains to some thing or portion of a thing.
- Each column contains data for a single attribute of the thing it’s describing.
- Each cell (intersection of a row and a column) of the table must have only a single value.
- Entries in any column must all be of the same kind. If, for example, the entry in one row of a column contains an employee name, all the other rows must contain employee names in that column, too.
- Each column must have a unique name.
- No two rows may be identical (that is, each row must be unique).

- The order of the columns and the order of the rows are not significant.

A table (relation) in first normal form is immune to some kinds of modification anomalies but is still subject to others. The SALES table is in first normal form, and the table is subject to deletion and insertion anomalies. First normal form may prove useful in some applications but unreliable in others.

SALES

Customer_ID	Product	Price
1001	Laundry detergent	12
1007	Toothpaste	3
1010	Chlorine bleach	4
1024	Toothpaste	3

Second normal form

To appreciate second normal form, you must understand the idea of functional dependency. A *functional dependency* is a relationship between or among attributes. One attribute is functionally dependent on another if the value of the second attribute determines the value of the first attribute. If you know the value of the second attribute, you can determine the value of the first attribute.
Suppose, for example, that a table has attributes (columns) StandardCharge, NumberOfTests, and TotalCharge that relate through the following equation:

```
TotalCharge = StandardCharge * NumberOfTests
```

TotalCharge is functionally dependent on both StandardCharge and NumberOfTests. If you know the values of StandardCharge and NumberOfTests, you can determine the value of TotalCharge.
Every table in first normal form must have a unique primary key. That key may consist of one or more than one column. A key consisting of more than one column is called a *composite key.* To be in second

normal form (2NF), all non-key attributes must depend on the entire key. Thus, every relation that is in 1NF with a single attribute key is automatically in second normal form.

If a relation has a composite key, all non-key attributes must depend on all components of the key. If you have a table where some non-key attributes don't depend on all components of the key, break the table up into two or more tables so that — in each of the new tables — all non-key attributes depend on all components of the primary key.

Sound confusing? Look at an example to clarify matters. Consider the SALES table. Instead of recording only a single purchase for each customer, you add a row every time a customer buys an item for the first time. An additional difference is that charter customers (those with Customer_ID values of 1001 to 1007) get a discount off the normal price.

SALES_TRACK

Customer_ID	Product	Price
1001	Laundry detergent	11.00
1007	Toothpaste	2.70
1010	Chlorine bleach	4.00
1024	Toothpaste	3.00
1010	Laundry detergent	12.00
1001	Toothpaste	2.70

Customer_ID does not uniquely identify a row. In two rows, Customer_ID is 1001. In two other rows, Customer_ID is 1010. The combination of the Customer_ID column and the Product column uniquely identifies a row. These two columns together are a composite key.

If not for the fact that some customers qualify for a discount and others don't, the table wouldn't be in second normal form, because Price (a non-key attribute) would depend only on part of the key (Product). Because some customers do qualify for a discount, Price depends on both CustomerID and Product, and the table is in second normal form.

Third normal form

Tables in second normal form are especially vulnerable to some types of modification anomalies — in particular, those that come from transitive dependencies.
A *transitive dependency* occurs when one attribute depends on a second attribute, which depends on a third attribute. Deletions in a table with such a dependency can cause unwanted information loss. A relation in third normal form is a relation in second normal form with no transitive dependencies.

Look again at the SALES table, which you know is in first normal form. As long as you constrain entries to permit only one row for each Customer_ID, you have a single-attribute primary key, and the table is in second normal form. However, the table is still subject to anomalies. What if customer 1010 is unhappy with the chlorine bleach, for example, and returns the item for a refund?

You want to remove the third row from the table, which records the fact that customer 1010 bought chlorine bleach. You have a problem: If you remove that row, you also lose the fact that chlorine bleach has a price of $4. This situation is an example of a transitive dependency. Price depends on Product, which, in turn, depends on the primary key Customer_ID.

Breaking the SALES table into two tables solves the transitive dependency problem. The two tables make up a database that’s in third normal form.