**CMR
INSTITUTE OF
TECHNOLOGY**

**Improvement Test – November 2016**

# SCHEME

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sub:** | **Data Base Management Systems** | | | | | | | | **Code:** | 10CS54 |
| Date: | 16/11/2016 | Duration: | 90 mins | Max Marks: | 50 | **Sem:** | V | | **Branch:** | CSE |

| **Note: Answer any five full questions** | | **Marks** | OBE | |
|---|---|---|---|---|
| | | | CO | RBT |
| 1 | Domain Constraint<br>Key Constraint [SK,CK,PK]<br>Not Null Constraint<br>Entity Integrity constraint<br>Referential Integrity constraint [FK] | 2*5 | CO4 | L1 |
| 2 | a) Set theoretic operations<br>Explanation of any two + Examples | 2<br>4 | CO4 | L2 |
| | b) Select<br>Project | 2<br>2 | CO4 | L2 |
| 3 | Explain two-phase locking[2PL] protocol used in concurrency control<br>Two Phases – [Growing and Shrinking] | 5*2 | CO4 | L1 |
| 4 | How a transaction is being supported in SQL. | 5M | CO6 | L1 |
| | Explain ACID properties of transaction. | 5M | CO6 | L1 |
| 5 | 1NF + Example<br>2NF + Example<br>3NF + Example | 3<br>3.5<br>3.5 | CO6 | L2 |
| 6 | Key<br>1NF<br>2NF<br>3NF | 1.5<br>1.5<br>2.5<br>2.5 | CO6 | L3 |
| 7 | Initial Matrix<br>Steps | 3<br>7 | CO6 | L3 |
| 8 | Consider the following schema for a COMPANY database:<br>EMPLOYEE(Fname, Lname, SSN, DOB, Sex, Salary, Super_Ssn, Dno)<br>DEPARTMENT(Dnumber, Dname, Mgr_ssn, Mgr_str_date)<br>DEPT_LOCATION(Dnumber, Dlocation)<br>PROJECT(Pname, Pnumber, Plocation, Dno)<br>WORKS-ON(Essn, Pno, Hours)<br>DEPENDENT(Essn, Depn_name, Sex, Relationship)<br><br>Write the SQL queries for the following.<br>i. Retrieve the names and salary of all employees who works for 'Finance' department.<br>ii. List the names of all employees with at least 2 dependents.<br>iii. For each project on which more than two employees work, retrieve the project number, project name and the number of employees who work on the project.<br>iv. Retrieve department number, the number of employees in the department and their average salary.<br>v. Select the name of employee whose 1st letter is R and 3rd letter is M. | 2*5 | CO5 | L3 |

**CMR INSTITUTE OF TECHNOLOGY**

**Improvement Test – November 2016**

# SOLUTION

| **Sub:** | Data Base Management Systems | | | | | | | **Code:** | 10CS54 |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 16/11/2016 | Duration: | 90 mins | Max Marks: | 50 | **Sem:** | V | **Branch:** | CSE |

| | **Questions and Answers.** | **Marks** | OBE | |
|---|---|---|---|---|
| | | | CO | RBT |
| 1 | **Domain Constraints:** Domain constraints specify that the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains typically include standard numeric data types for integers (such as short-integer, integer, long-integer) and real numbers (float and double-precision float). Characters, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and money data types.<br><br>**Key Constraints and Constraints on Null:** A relation is defined as a set of tuples. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes. Usually, there are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes. Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that<br>$t_1[SK] \neq t_2[SK]$<br>Any such set of attributes SK is called a super key of the relation schema R. A super key SK specifies a uniqueness constraint that no two distinct tuples in a state r of R can have the same value for SK. A super key can have redundant attributes, however, so a more useful concept is that of a key, which has no redundancy.<br>A key K of a relation schema R is a super key of R with the additional property that removing any attribute A from K leaves a set of attributes K' that is not a super key of R. Hence, a key is a minimal super key—that is, a super key from which we cannot remove any attributes and still have the uniqueness constraint hold.<br>In general, a relation schema may have more than one key. In this case, each of the keys is called a candidate key. One among of the candidate keys is designated as the primary key of the relation, which is used to uniquely identify a tuple in the relation.<br>Another constraint on attributes specifies whether null values are or are not permitted. For example, if every STUDENT tuple must have a valid, non-null value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.<br><br>**Entity Integrity constraint:** The entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values for the primary key implies that we cannot identify some tuples. If two or more tuples had null for their primary keys, we might not be able to distinguish them.<br>**Referential Integrity and Foreign Keys:** The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. | | CO4 | L1 |

To define referential integrity more formally, we first define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R1 and R2. A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following two rules:
1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is null. In the former case, we have $t_1$[FK] = $t_2$[PK], and we say that the tuple t1 references or refers to the tuple t2. R1 is called the referencing relation and R2 is the referenced relation.
If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.

2 a) The following are the set theoretic operations are used to merge the elements of two sets in various ways in relational algebra,

- UNION
- INTERSECTION
- SET DIFFERENCE
- CARTESIAN PRODUCT

When these operations are adapted to relational databases, the two relations on which any of the above three operations are applied must have the same type of tuples; this condition is called union compatibility.

**UNION:** The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples is eliminated.
Example:

R:

| **Name** |
|---|
| Ram |
| Raju |
| Rakesh |

S:

| **Nam** |
|---|
| Ram |
| Rajesh |
| Ramu |

R ∪ S:

| **Name** |
|---|
| Ram |
| Raju |
| Rakesh |
| Rajesh |
| Ramu |

CO4 L2

**INTERSECTION:** The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S.
Example:

R:

| **Name** |
|---|
| Ram |
| Raju |
| Rakesh |

S:

| **Name** |
|---|
| Ram |
| Rajesh |
| Ramu |

R ∩ S:

| **Name** |
|---|
| Ram |

**SET DIFFERENCE:** The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S.
Example:

R:

| **Name** |
|---|
| Ram |
| Raju |
| Rakesh |

S:

| **Name** |
|---|
| Ram |
| Rajesh |
| Ramu |

R – S:

| **Name** |
|---|
| Raju |
| Rakesh |

b) SELECT Operation:
The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.3 One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to restrict the tuples in a relation to only those tuples that satisfy the condition. The SELECT operation can also be visualized as a horizontal

CO4 L2

partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

It is denoted by $\sigma$<selection condition>(R)
the selection condition is a Boolean expression (condition) specified on the attributes of relation R.

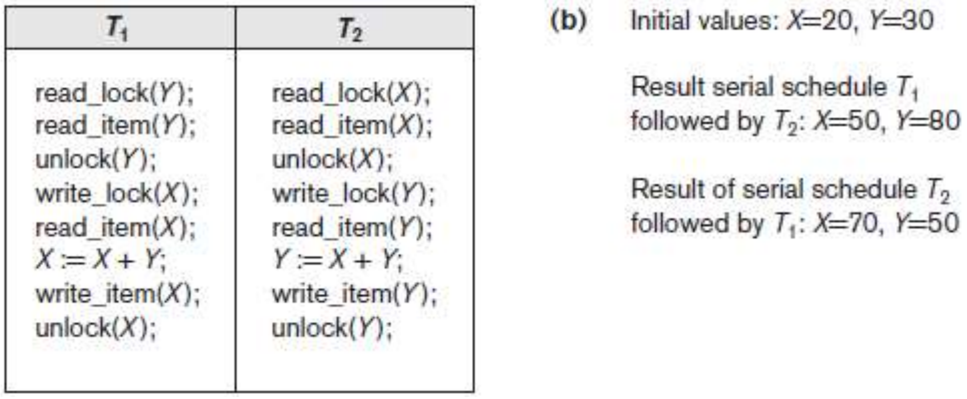
PROJECT Operation:
The PROJECT operation, selects certain columns from the table and discards the other columns. If we are interested in only certain attributes of a relation, we use the PROJECT operation to project the relation over these attributes only. Therefore, the result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.

It is denoted by $\Pi$<attribute list>(R)
Where <attribute list> is the desired sublist of attributes from the attributes of relation R.

3 Two-Phase Locking protocol - This protocol has two rules:
a) If a transaction T wants to read (respectively, modify) an object, it first request a shared (respectively, exclusive) lock on the object.
b) A transaction cannot request additional locks once it releases any lock.
2PL allow transactions to release locks before the end, that is, before the commit or abort action.

A transaction is said to follow 2PL if all locking operations (read_lock, wrote_lock) precede the first unlock operation in the transaction. Such transactions can be divided into two phases:
**A Growing/expanding phase:** in which new locks can be acquired, but none can be released.
**A Shrinking phase:** which is followed by growing phase, where existing locks can be released but none can be acquired.
If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase. Hence, a read_lock(X) operation that downgrades an already held write lock on X can appear only in the shrinking phase.
Transactions T1 and T2 below, do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T2.

CO4 L1



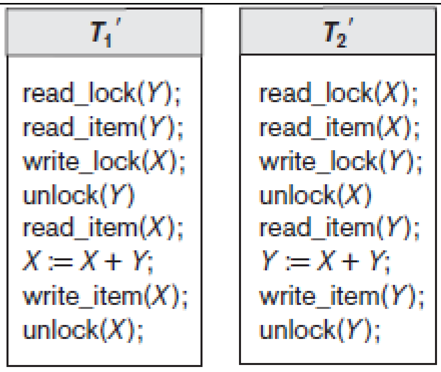
| $T_1$ | $T_2$ |
|---|---|
| read_lock(Y); | read_lock(X); |
| read_item(Y); | read_item(X); |
| unlock(Y); | unlock(X); |
| write_lock(X); | write_lock(Y); |
| read_item(X); | read_item(Y); |
| $X := X + Y$; | $Y := X + Y$; |
| write_item(X); | write_item(Y); |
| unlock(X); | unlock(Y); |

(b) Initial values: $X$=20, $Y$=30

Result serial schedule $T_1$ followed by $T_2$: $X$=50, $Y$=80

Result of serial schedule $T_2$ followed by $T_1$: $X$=70, $Y$=50

In order to enforce two-phase locking, the transactions can be rewritten as T1` and T2`, as shown.



| $T_1'$ | $T_2'$ |
|---|---|
| read_lock($Y$); | read_lock($X$); |
| read_item($Y$); | read_item($X$); |
| write_lock($X$); | write_lock($Y$); |
| unlock($Y$) | unlock($X$) |
| read_item($X$); | read_item($Y$); |
| $X := X + Y$; | $Y := X + Y$; |
| write_item($X$); | write_item($Y$); |
| unlock($X$); | unlock($Y$); |

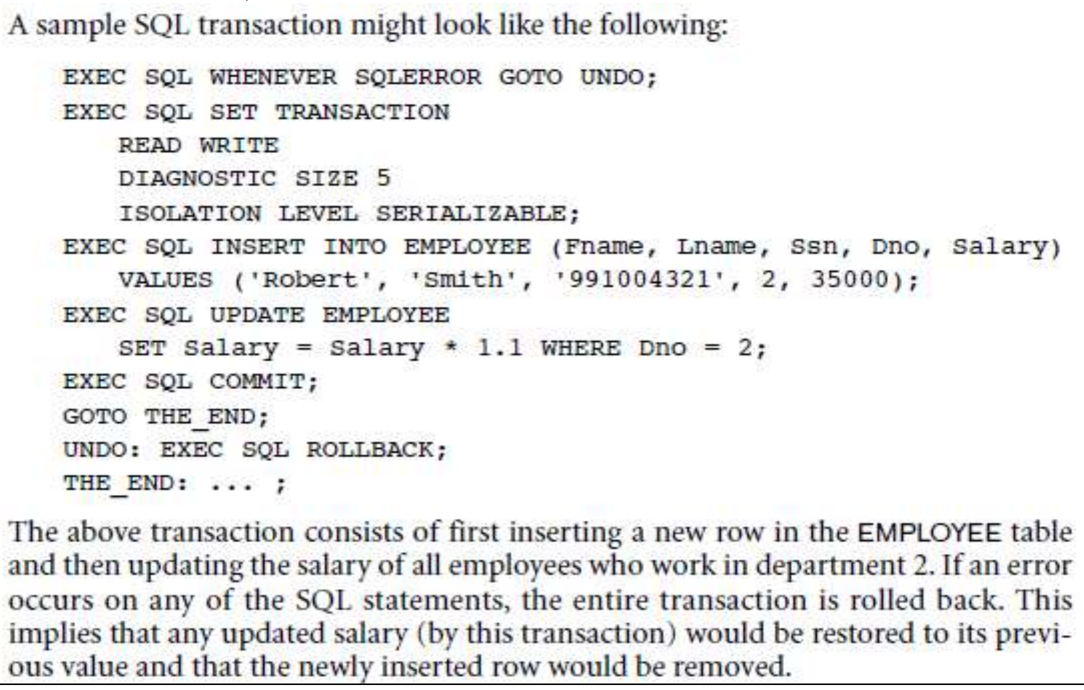
It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules. The locking protocol, by enforcing two-phase locking rules, also enforces serializability.

4 a) Transaction initiation is done implicitly when particular SQL statements are encountered. However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK. Every transaction has certain characteristics attributed to it. These characteristics are specified by a SET TRANSACTION statement in SQL. The characteristics are the access mode, the diagnostic area size, and the isolation level.

a. The access mode can be specified as READ ONLY or READ WRITE. The default is READ WRITE, unless the isolation level of READ UNCOMMITTED is specified (see below), in which case READ ONLY is assumed. A mode of READ WRITE allows select, update, insert, delete, and create commands to be executed. A mode of READ ONLY, as the name implies, is simply for data retrieval.

b. The diagnostic area size option, DIAGNOSTIC SIZE n, specifies an integer value n, which indicates the number of conditions that can be held simultaneously in the diagnostic area. These conditions supply feedback information (errors or exceptions) to the user or program on the n most recently executed SQL statement.

c. The isolation level option is specified using the statement ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE



A sample SQL transaction might look like the following:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
    VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

The above transaction consists of first inserting a new row in the EMPLOYEE table and then updating the salary of all employees who work in department 2. If an error occurs on any of the SQL statements, the entire transaction is rolled back. This implies that any updated salary (by this transaction) would be restored to its previous value and that the newly inserted row would be removed.

CO6 L1

b) Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

a. Atomicity. A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

b. Consistency preservation. A transaction should be consistency preserving, meaning that if it is

CO6 L1

completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

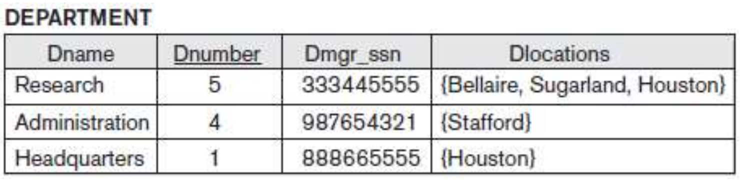
c. Isolation. A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

d. Durability or permanency. The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

5 **First Normal Form:** "It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute."

1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

Consider the DEPARTMENT relation schema shown below, whose primary key is Dnumber, As it is shown, a department can have number of locations and hence Dlocations can have multiple values for a department. Thus the relation DEPARTMENT his is not in 1NF because Dlocations is not an atomic attribute.

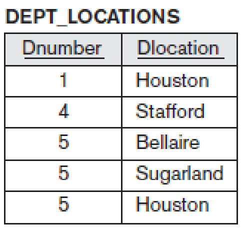


DEPARTMENT

| Dname | Dnumber | Dmgr_ssn | Dlocations |
|---|---|---|---|
| Research | 5 | 333445555 | {Bellaire, Sugarland, Houston} |
| Administration | 4 | 987654321 | {Stafford} |
| Headquarters | 1 | 888665555 | {Houston} |

There are three main techniques to achieve first normal form for such a relation:

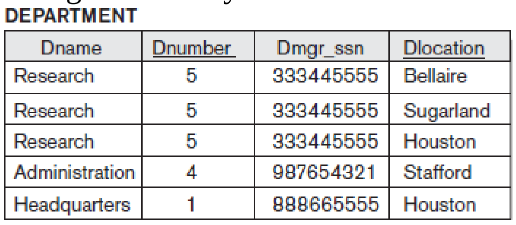
1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation} as shown below. This decomposes the non-1NF relation into two 1NF relations.



DEPT_LOCATIONS

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |

2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown below. In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing redundancy in the relation.



DEPARTMENT

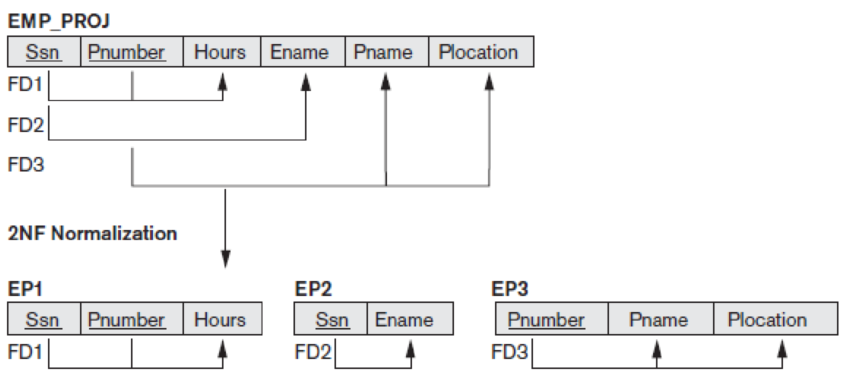
| Dname | Dnumber | Dmgr_ssn | Dlocation |
|---|---|---|---|
| Research | 5 | 333445555 | Bellaire |
| Research | 5 | 333445555 | Sugarland |
| Research | 5 | 333445555 | Houston |
| Administration | 4 | 987654321 | Stafford |
| Headquarters | 1 | 888665555 | Houston |

3. If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations.

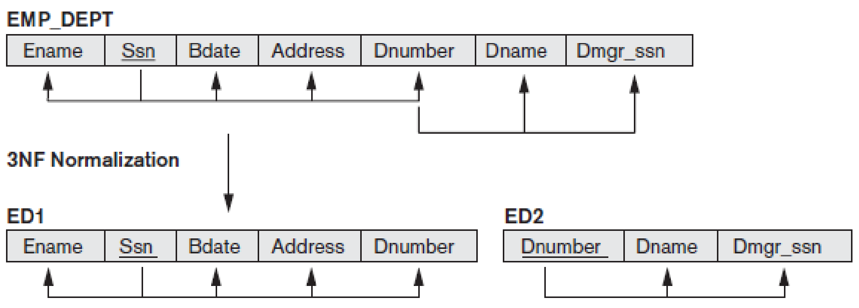
**Second normal form (2NF):** A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

CO6 L2

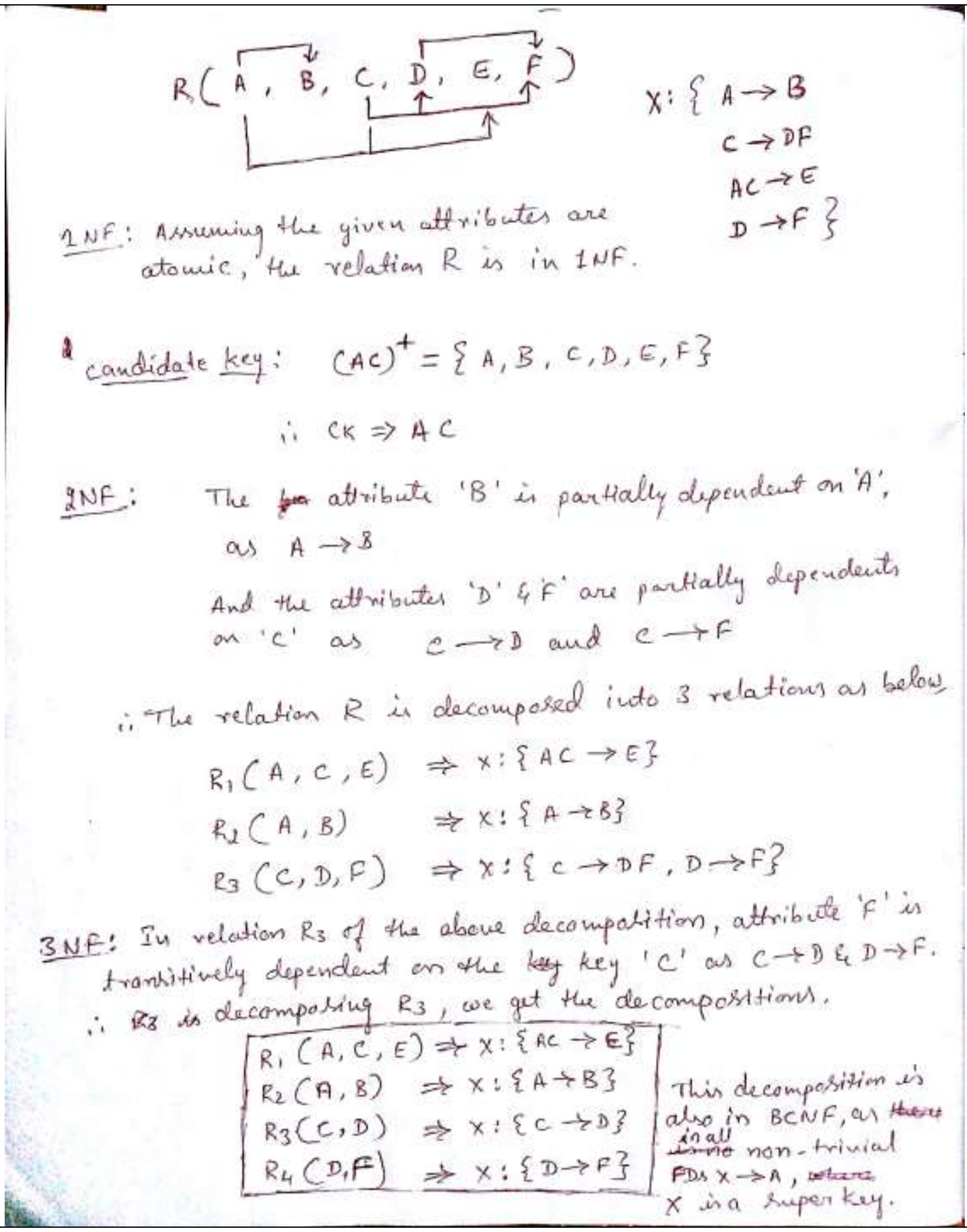
A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ε X, (X – {A}) does not functionally determine Y. A functional dependency X→Y is a partial dependency if some attribute A ε X can be removed from X and the dependency still holds; that is, for some A ε X, (X – {A}) → Y.
For example in the belwo figure, In the relation EMP_PROJ {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds). However, the dependency {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds. Thus EMP_PROJ is not in 2NF. The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3. The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key {Ssn, Pnumber} of EMP_PROJ, thus violating the 2NF test.
The functional dependencies FD1, FD2, and FD3 lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown below, each of which is in 2NF.



**Third normal form (3NF):** "A relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key."
A functional dependency X→Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X→Z and Z→Y hold.
For example in the below figure of relation EMP_DEPT, The dependency Ssn→Dmgr_ssn is transitive through Dnumber because both the dependencies Ssn → Dnumber and Dnumber → Dmgr_ssn hold and Dnumber is neither a key itself nor a subset of the key of EMP_DEPT. The relation schema EMP_DEPT is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown below.



6



R( A , B, C, D, E, F )

X: { A → B
C → DF
AC → E
D → F }

1NF: Assuming the given attributes are atomic, the relation R is in 1NF.

candidate key: $(AC)^+ = \{A, B, C, D, E, F\}$

∴ CK ⇒ A C

2NF: The ~~[illegible]~~ attribute 'B' is partially dependent on 'A', as A → B

And the attributes 'D' & 'F' are partially dependents on 'C' as C → D and C → F

∴ The relation R is decomposed into 3 relations as below

$R_1(A, C, E) \Rightarrow X: \{AC \to E\}$

$R_2(A, B) \Rightarrow X: \{A \to B\}$

$R_3(C, D, F) \Rightarrow X: \{C \to DF, D \to F\}$

3NF: In relation $R_3$ of the above decompatition, attribute 'F' is transitively dependent on the ~~key~~ key 'C' as C → D & D → F.

∴ ~~R3 is~~ decomposing $R_3$, we get the decompositions.

$R_1(A, C, E) \Rightarrow X: \{AC \to E\}$

$R_2(A, B) \Rightarrow X: \{A \to B\}$

$R_3(C, D) \Rightarrow X: \{C \to D\}$

$R_4(D, F) \Rightarrow X: \{D \to F\}$

This decomposition is also in BCNF, as ~~there~~ in all ~~some~~ non-trivial FDs X → A, ~~where~~ X is a super key.

CO6 L3

7

Initial Matrix S:

| | SSN | Pno | Ename | Hrs | Pname | Ploc |
|---|---|---|---|---|---|---|
| R1 | $a_1$ | $b_{12}$ | $a_3$ | $b_{14}$ | $b_{15}$ | $b_{16}$ |
| R2 | $b_{21}$ | $a_2$ | $b_{23}$ | $b_{24}$ | $a_5$ | $a_6$ |
| R3 | $b_{31}$ | $b_{32}$ | $b_{33}$ | $a_4$ | $b_{35}$ | $b_{36}$ |

| | SSN | Pno | Ename | Hrs | Pname | Ploc |
|---|---|---|---|---|---|---|
| R1 | $a_1$ | $b_{12}$ | $a_3$ | $b_{14}$ | $b_{15}$ | $b_{16}$ |
| R2 | $b_{21}$ | $a_2$ | $b_{23}$ | $b_{24}$ | $a_5$ | $a_6$ |
| R3 | $a_1$ | $a_2$ | ~~$b_{33}$~~ $a_3$ | $a_4$ | ~~$b_{35}$~~ $a_5$ | ~~$b_{36}$~~ $a_6$ |

Matrix S after applying 2nd and 3rd FDs, Last row is completely made of 'a' symbol, hence we stop.
Therefore the given decomposition is lossless.

CO6 L3

| | | | | |
|---|---|---|---|---|
| 8 | i. SELECT E.FNAME, E.LNAME, E.SALARY<br>FROM EMPLOYEE E, DEPARTMENT D<br>WHERE E.DNO=D.DNUMBER AND DNAME=’Finance’;<br><br>ii. SELECT FNAME FROM EMPLOYEE<br>WHERE SSN IN (SELECT D.SSN FROM DEPENDENT D<br>GROUP BY D.SSN<br>HAVING COUNT(D.DEPN_NAME)>2);<br><br>iii. SELECT P.PNO,P.PNAME, COUNT(W.SSN)<br>FROM PROJECT P, WORKS_ON W<br>WHERE P.PNUMBER=W.PNO<br>GROUP BY P.PNO,P.PNAME<br><br>iv. SELECT E.DNO, COUNT(E.SSN), AVG(E.SALARY)<br>FROM EMPLOYEE E<br>GROUP BY E.DNO;<br><br>v. SELECT FNAME FROM EMPLOYEE WHERE FNAME LIKE ‘R_M%’; | | CO5 | L3 |