Fifth Semester B.E. Degree Examination, Dec.2017/Jan.2018
Database Management Systems

Time: 3 hrs. Max. Marks: 80

Note: *Answer FIVE full questions, choosing one full question from each module.*

Module-1

1 a. Explain the main characteristics of the database approach versus the file processing approach. **(08 Marks)**

b. Explain the three – schema architecture with neat diagram. Why do we need mappings among schema levels? How do different schema definition languages support this architecture? **(08 Marks)**

OR

2 a. Discuss with examples, different types of attributes. **(07 Marks)**

b. Draw an ER diagram for a BANK database schema with atleast five entity types. Also specify primary key and structural constraints. **(09 Marks)**

Module-2

3 a. Describe the characteristics of relations with suitable example for each. **(08 Marks)**

b. What are the basic operations that can change the states of relations in the database? Explain how the basic operations deal with constraint violations. **(08 Marks)**

OR

4 a. Describe the steps of an algorithm for ER – to – relational mapping. **(10 Marks)**

b. In SQL which command is used for table creation? Explain how constraints are specified in SQL during table creation with suitable example. **(06 Marks)**

Module-3

5 Consider the COMPANY DATABASE

EMPLOYEE (Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, super-ssn, Dno)
DEPARTMENT (Dname, Dnumber, Mgr_ssn, Mgr_st_date)
DEPART_LOCATIONS(Dnumber, Dlocation)
PROJECT (Pname, Pnumber, Plocation, Dnum)
WORKS_ON (Essn, Pno, Hours)
DEPENDENT (Essn, Dependent_name, Sex, Bdate, Relationship).

Specify the following queries in SQL on the database schema given above :

a. For every project located in Stafford, list the project number the controlling department number and the department manager's last name, address and birth date. **(04 Marks)**

b. List the names of all employees who have a dependent with the same first name as themselves. **(02 Marks)**

c. For each project, list the project name and the total hours per week (by all employees) spent on that project. **(04 Marks)**

d. Retrieve the name of each employee who works on all the projects controlled by 'Research' department. **(06 Marks)**

OR

6 a. Define Stored Procedure. Explain the creating and calling of stored procedure with suitable example. (08 Marks)

b. Explain the Single – tier and Client – server architecture, with neat diagram. (08 Marks)

Module-4

7 a. Explain the informal design guidelines used as measures to determine the quality of relation schema design. (08 Marks)

b. Define Normal form. Explain 1NF, 2NF and 3NF with suitable examples for each. (08 Marks)

OR

8 a. Define Minimal cover. Write an algorithm for finding a minimal cover F for a set of functional dependencies E. Find the minimal cover for the given set of FDs be (08 Marks)
$E : \{B \rightarrow A , D \rightarrow A , AB \rightarrow D\}$.

b. Consider the universal relation R = {A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies (08 Marks)
$F = \{\{A, B\} \rightarrow \{C\} , \{A\} \rightarrow \{D, E\} , \{B\} \rightarrow \{F\} , \{F\} \rightarrow \{G, H\} , \{D\} \rightarrow \{I, J\}\}$.
Determine whether each decomposition has the lossless join property with respect to F.
$D_1 = \{R_1, R_2, R_3\}$; $R_1 = \{A, B, C, D, E\}$; $R_2 = \{B, F, G, H\}$; $R_3 = \{D, I, J\}$.

Module-5

9 a. Why Concurrency control is needed demonstrate with example? (12 Marks)

b. Discuss the desirable properties of transactions. (04 Marks)

OR

10 a. When deadlock and starvation problems occurs? Explain how these problems can be resolved. (09 Marks)

b. Explain how shadow paging helps to recover from transaction failure. (07 Marks)

SOLUTIONS

1 a.	**MODULE 1**

Characteristics of the Database Approach

The main characteristics of database approach versus file-processing approach are the following:
1) Self-describing nature of database system
2) Insulation between programs & data, and data abstraction
3) Support of multiple views of the data
4) Sharing of data & multiuser transaction processing

① Self-describing nature of a database system:

Database system includes a complete definition of database structure and constraints in addition to data stored. This is stored in DBMS catalog – contains structure of each file, type and storage format of each data item, and various constraints on data. This is called meta-data (data about data). The catalog is used by DBMS software and database users.

In traditional file processing, data definition is part of application programs. These programs are constrained to work with only one specific database whose structure is declared in application programs. DBMS software can access diverse databases by extracting

RELATIONS

Relation name	No. of cols
STUDENT	4

COLUMNS

Col name	Data type	Belongs to relation

⇒ An eg of database catalog

database definitions from catalog.

② Insulation between Programs and Data, and Data Abstraction

Program-data Independence :- The structure of data files is stored in DBMS catalog separately from access programs. In traditional file system, any changes to the structure of a file may require changing all programs that access that file.

Program-operation Independence :- In some databases systems, operations can be defined on data as part of database definitions. Operation (function or method) is specified in 2 parts :

a) Interface (or signature) includes operation name and data types of its arguments
b) Implementation (or method) can be changed without affecting the interface.

Programs can operate on data by invoking these operations through names irrespective of how the operations are implemented.

Data Abstraction : The characteristic that allows program-data independence & program-operation independence

- DBMS provides users with conceptual representation of data → does not include details of how data is stored or how operations are implemented. Data model is a type of data abstraction used to provide conceptual representation (hides storage & implementation details)

③ Support of Multiple Views of Data

Different users have different "views" or perspectives on the database.
A view may be a subset of the database or it may contain virtual data derived from database files but not explicitly stored.
For eg, one user may be interested only in accessing transcript of each student and another user may require only the details of prerequisites.

④ Sharing of Data and Multi User Transaction Processing:

Multiuser database allows multiple users to access database at the same time.
The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled

manner So that results of updates is correct
eg: Online airline reservation System.
These types of applications are Online transaction processing applications (OLTP)
A transaction is an executing program or process that includes one or more database accesses. The isolation property ensures that each transaction appears to execute in isolation from other transactions.
The atomicity property ensures that either all the database operations are executed or none are.

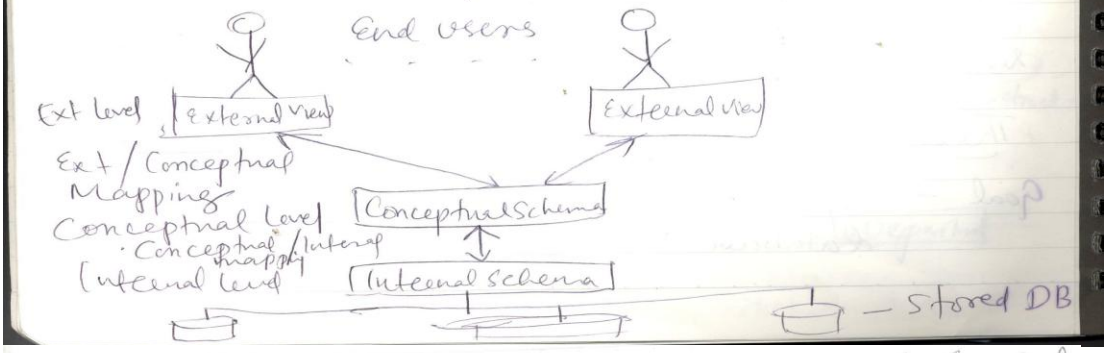
1 b.

Schema can be defined at three levels :—

① Internal level has — internal schema — describes physical storage structure of db. Uses physical data model & describes complete details of data storage & access paths.

② Conceptual level has — conceptual schema — describes structure of whole database for a community of users. Focuses on describing entities, data types, relationships. Uses representational data model.

③ External or View level — includes external schemas or user views. Each external schema describes part of the database that a particular user group is interested in and hides rest of the database from that user group.



The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. The processes of transforming requests and results between levels are called mappings.

Data Independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only mapping between the two levels is changed.

The 3-schema architecture can make it easier to achieve true data independence.

DBMS Languages :-

DBMS supports variety of users and must provide appropriate languages & interfaces for each category of users.

DBMS languages :-

1) DDL (Data Definition Language) – Used by DBA and database designers to define conceptual and internal schemas.

2) SDL (Storage Definition Language) – Used to specify internal schema.

3) VDL (View Definition Language) – Used to specify external schemas (user views)

4) DML (Data Manipulation Language) – Used for performing operations such as retrieval, insertion, deletion & modification.

Two main types of DML :

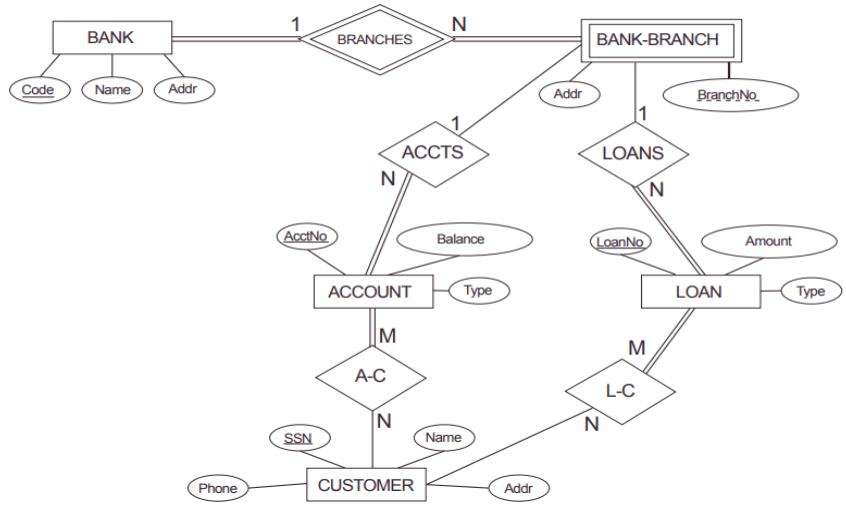
i) High-level or non procedural DML – can be used on its own to specify complex database operations concisely. (set-at-a-time DMLs)

ii) Low-level or procedural DML – Must be embedded in general purpose programming language. Retrieves individual records or objects from the database & processes each separately. (Record-at-a-time DMLs)

2a. Attribute—the particular properties that describe an entity. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.

Several types of attributes occur in the ER model: simple versus composite, singlevalued versus multivalued, and stored versus derived.

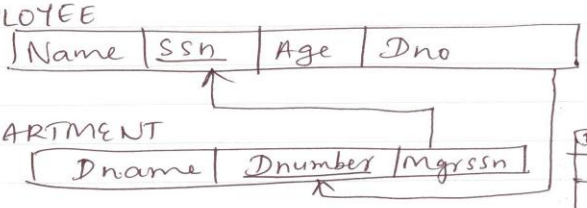
Composite versus Simple (Atomic) Attributes: Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip,3 with the values '2311 Kirby', 'Houston', 'Texas', and '77001.' Attributes that are not divisible are called simple or atomic

	attributes. Composite attributes can form a hierarchy; for example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number, **Single-Valued versus Multivalued Attributes**: Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a single-valued attribute of a person. one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different numbers of values for the College_degrees attribute. Such attributes are called multivalued. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. **Stored versus Derived Attributes**: In some cases, two (or more) attribute values are related—for example, the Age and Birth_date attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date. The Age attribute is hence called a derived attribute and is said to be derivable from the Birth_date attribute, which is called a stored attribute.
2b.	**Figure 3.17** An ER diagram for a BANK database. 
3a.	**MODULE 2** **Ordering of Tuples**: A relation is a *set* of tuples; hence, there is no order associated with them. That is, it makes no sense to refer to, for example, the 5th tuple in a relation. When a relation is depicted as a table, the tuples are necessarily listed in *some* order, of course, but you should attach no significance to that order. Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content. **Ordering of Attributes**: A tuple is best viewed as a mapping from its attributes (i.e., the names we give to the roles played by the values comprising the tuple) to the corresponding values. Hence, the order in which the attributes are listed in a table is irrelevant. The **Null** value: used for *don't know*, *not applicable*. **Interpretation of a Relation**: Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact. Example (see Figure 5.1): The first tuple listed means: There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc. Keep in mind that some relations represent facts about entities (e.g., students) whereas others represent facts

	about relationships (between entities). (e.g., students and course sections).The **closed world assumption** states that the only true facts about the miniworld are those represented by whatever tuples currently populate the database.
3 b.	Operations of relational model can be categorized into retrievals and updates. 3 basic operations that can change the states of relations in database: Insert, Delete, Update (or Modify) ↓ ↓ ↓ to Insert new tuples — delete tuples — change the value of some attributes in existing tuples **Insert Operation** Insert can violate any of the four types of constraints :- Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.

Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
Entity integrity can be violated if any part of the primary key of the new tuple is NULL.
Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

eg: EMPLOYEE



DEPARTMENT

Name	Ssn	Age	Dno
Ron	1234	30	1
Dev	5678	20	2
Smith	1110	50	3

Dname	Dnumber	Mgrssn
CS	1	1234
EC	2	5678
CVL	3	1110

* Insert into EMPLOYEE values ('John', NULL, 30, 1);
  Result: Violates entity integrity constraint (NULL for p.k)
* Insert into EMPLOYEE values ('Alicia', 1234, 35, 2);
  Result: Violates key constraint because another tuple with same Ssn value already exists in EMPLOYEE.
* Insert into EMPLOYEE values ('Cecilia', 8910, 33, 4);
  Result: Violates referential integrity constraint specified on Dno in EMPLOYEE because no tuple exists in DEPARTMENT with Dnumber = 4.

The default option is to reject the insertion if an insertion violates constraints. Another option is to provide a reason to the user for rejection & attempt to correct it.

DELETE Operation

Delete operation can violate only referential integrity. This occurs if tuple being deleted is referenced by foreign keys from other tuples in the database.

eg: Delete from employee where ssn = 1234;
— Not acceptable because there are tuples in DEPARTMENT that refer to this tuple.

Options available if a deletion causes a violation:

① Restrict, — reject deletion.
② Cascade — attempt to propagate deletion by deleting tuples that reference the tuple that is being deleted.
③ Set null or set default — Modify the referencing attribute values that cause violations; each such value is either set to NULL or changed to another default valid tuple.

UPDATE Operation

Update (or Modify) is used to change the values of one or more attributes in a tuple.

	Updating an attribute that is neither part of primary key nor part of foreign key usually causes no problems. Modifying a primary key value is similar to deleting one tuple and inserting another in its place. So key constraints & entity integrity issues If a foreign key attribute is modified, the DBMS must make sure that new value refers to an existing tuple in the referenced relation.
4 a.	Step 1: For each **regular (strong) entity type** E in the ER schema, create a relation R that includes all the simple attributes of E. Step 2: For each **regular (strong) entity type** E in the ER schema, create a relation R that includes all the simple attributes of E. Step 3: For each **binary 1:1 relationship type** R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S. Step 4: For each regular **binary 1:N relationship type** R identify the relation (N) relation S. Include the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S. Step 5: For each **binary M:N relationship type** R, create a relation S. Include the primary keys of participant relations as foreign keys in S. Their combination will be the primary key for S. Simple attributes of R become attributes of S. Step 6: For each **multi-valued attribute A**, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation (entity type or relationship type) as a foreign key in R. The primary key of R is the combination of A and K. Step 7: For each **n-ary relationship type R**, where n>2, create a new relation S to represent R. Include the primary keys of the relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint > 1.
4b.	CREATE command is used for table creation. The basic constraints that can be specified when you create a table in SQL are: • Not Null • Unique • Default • Primary Key • Foreign Key • Check Not Null: By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column. Unique: The UNIQUE Constraint prevents two records from having identical values in a particular column. Default: The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value. Primary Key: A primary key is a single field or combination of fields that uniquely identify a record. The fields that are part of the primary key cannot contain a NULL value and must be Unique. Each table should have a primary key, and each table can have only ONE primary key.

	Foreign Key: A foreign key is a key used to link two tables together. Foreign Key is a column or a combination of columns whose values match a Primary Key of another table. The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table. Example: CREATE TABLE EMPLOYEE (SSN NUMBER(10) PRIMARY KEY, NAME VARCHAR(10) NOT NULL, PHONE NUMBER(10) UNIQUE, AGE NUMBER(2) CHECK AGE>16, SALARY NUMBER(5) DEFAULT 10000, DEPTNO NUMBER(2) REFERENCES DEPARTMENT(DNO))
	MODULE 3
5a.	STAFFORD_PROJS ← σPlocation='Stafford'(PROJECT) CONTR_DEPTS ← (STAFFORD_PROJS Dnum=DnumberDEPARTMENT) PROJ_DEPT_MGRS ← (CONTR_DEPTS Mgr_ssn=SsnEMPLOYEE) RESULT ← πPnumber, Dnum, Lname, Address, Bdate(PROJ_DEPT_MGRS)
5b.	$emp\_with\_Deps \leftarrow employee \bowtie_{ssn=essn\ and\ fname=dependent\_name} dependent$ $answer \leftarrow \pi_{fname,minit,lname}(emp\_with\_Deps)$
5c.	$proj\_hours(pno, total\_hours) \leftarrow_{pno} \mathcal{F}_{sum\ hours}(works\_on)$ $answer \leftarrow \pi_{pname,total\_hours}(projs\_hours \bowtie_{pno=pnumber} project)$
5d.	deptno ← πDnum(σDname='Research'(DEPARTMENT)) DEPT5_PROJS ← ρ(Pno)(πPnumber(σDnum=deptno(PROJECT))) EMP_PROJ ← ρ(Ssn, Pno)(πEssn, Pno(WORKS_ON)) RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS RESULT ← πLname, Fname(RESULT_EMP_SSNS * EMPLOYEE)
6a.	<u>Database stored procedures and SQL/PSM</u> • Persistent procedures/functions (modules) are stored locally and executed by the database server. As opposed to execution by clients. • Advantages: If the procedure is needed by many applications, it can be invoked by • any of them (thus reduce duplications) • Execution by the server reduces communication costs • Enhance the modeling power of views • Disadvantages: Every DBMS has its own syntax and this can make the system less portable A stored procedure EXAMPLE CREATE PROCEDURE procedure-name (params) local-declarations procedure-body; A stored function CREATE FUNCTION fun-name (params) RETRUNS return-type local-declarations function-body; Calling a procedure or function: CALL procedure-name/fun-name (arguments);

E.g.,

```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)
RETURNS VARCHAR[7]
DECLARE TOT_EMPS INTEGER;
SELECT COUNT (*) INTO TOT_EMPS
FROM SELECT EMPLOYEE
WHERE DNO = deptno;
IF TOT_EMPS > 100
  THEN RETURN —HUGE
ELSEIF TOT_EMPS > 50
  THEN RETURN —LARGE
ELSEIF TOT_EMPS > 30
  THEN RETURN —MEDIUM
ELSE RETURN SMALL
ENDIF
```

6b.

Initially, data-intensive applications were combined into a single tier, including the DBMS, application logic, and user interface, a" illustrated in Figure 7.5. The application typically ran on a mainframe, and users

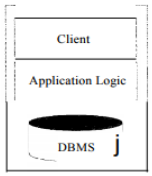


Figure 7.5 A Single-Tier Architecture

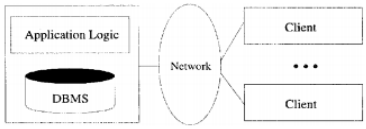


Figure 7.6 A Two-Server Architecture: Thin Clients

accessed it through dumb teT'minals that could perform only data input and display. This approach ha..s the benefit of being easily maintained by a central administrator.

Single-tier architectures have a,n important drawback: Users expect graphical interfaces that require much more computational power than simple dumb terminals. Centralized computation of the graphical displays of such interfaces requires much more computational power than a single server hclS available, and thus single-tier architectures do not scale to thousands of users. The commoditization of the PC and the availability of cheap client computers led to the developlnent of the two-tier architecture. Two-tier architectures, often also referred to a<; client-server architectures, consist of a client computer and a server computer, which interact through a well-defined protocol. What part of the functionality the client implements, and what part is left to the server, can vary. In the traditional clientserver architecture, the client implements just the graphical user interface, and the server. implements both the business logic and the data management; such clients are often called thin clients, and this architecture is illustra,ted in Figure 7.6

Compared to the single-tier architecture, two-tier architectures physically separate the user interface from the data management layer. To implement twotier architectures, we can no longer have dumb terminals on the client side; we require computers that run sophisticated presentation code (and possibly, application logic).

MODULE 4

7a.

GUIDELINE 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, the meaning tends to be clear.

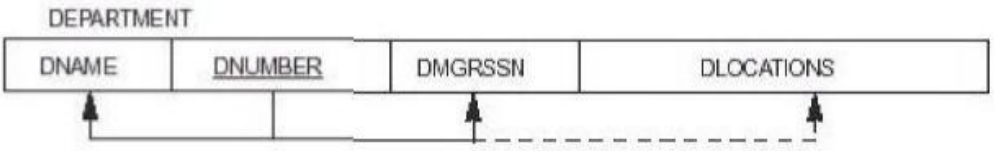
GUIDELINE 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

GUIDELINE 3: As far as possible, avoid placing attributes in a base relation whose values may frequently

	be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation. Having too many Null values can be waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level. Another problem with nulls is how to account for them when aggregate operations such as COUNT or SUM are applied. Moreover, nulls can have multiple interpretations. 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 tuples are generated. Do not have relations that contain matching attributes other than foreign key-primary key combinations. If such relations are unavoidable, do not join them on such attributes, because the join may produce spurious tuples.
7b.	**Normalization:** ■ The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations **Normal form:** ■ Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form First Normal Form (1NF) First normal form is now considered to be part of the formal definition of a relation; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domains of attributes 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. Practical Rule: "Eliminate Repeating Groups," i.e., make a separate table for each set of related attributes, and give each table a primary key. Formal Definition: A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only.

Figure 14.8 Normalization into 1NF. (a) Relation schema that is not in 1NF. (b) Example relation instance. (c) 1NF relation with redundancy.

(a)



(b) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Second Normal Form (2NF)

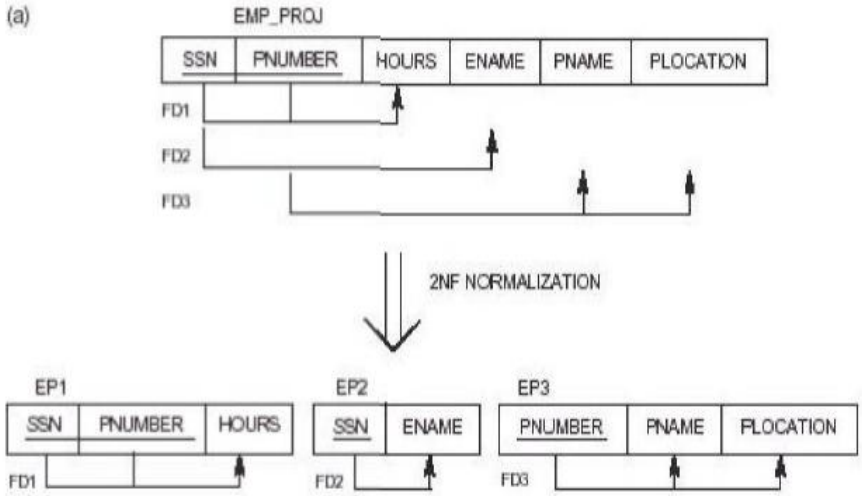
Second normal form is based on the concept of fully functional dependency. A functional X ->Y is a fully functional dependency is removal of any attribute A from X means that the dependency does not hold any more. A relation schema is in 2NF if every nonprime attribute in relation is fully functionally dependent on the primary key of the relation. It also can be restated as: a relation schema is in 2NF if every nonprime attribute in relation is not partially dependent on any key of the relation.

Practical Rule:

"Eliminate Redundant Data," i.e., if an attribute depends on only part of a multivalued key, remove it to a separate table.

Formal Definition:

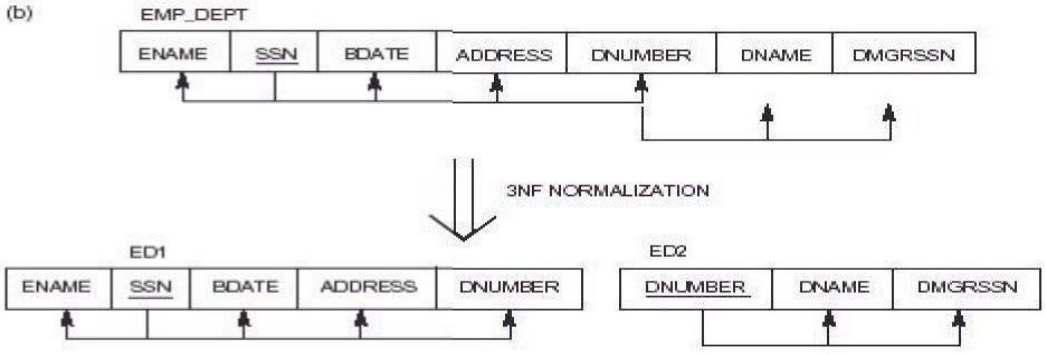
A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.



Practical Rule: "Eliminate Columns not Dependent on Key," i.e., if attributes do not contribute to a description of a key, remove them to a separate table.
Formal Definition:
A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

Third Normal Form (3NF)

Third normal form is based on the concept of transitive dependency. A functional dependency X Y in a relation is a transitive dependency if there is a set of attributes Z that is not a subset of any key of the relation, and both X ->Z and Z -> Y hold. In other words, a relation is in 3NF if,
whenever a functional dependency X-> A holds in the relation, either
(a) X is a superkey of the relation,
or (b) A is a prime attribute of the relation.
Practical Rule: "Eliminate Columns not Dependent on Key," i.e., if attributes do not contribute to a description of a key, remove them to a separate table.
Formal Definition:
A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.



8a	A set of FDs is minimal if it satisfies the following conditions: 1. Every dependency in F has a single attribute for its RHS. 2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F. A→3. We cannot replace any dependency X A, where Y is a→in F with a dependency Y proper-subset-of X and still have a set of dependencies that is equivalent to F. Algorithm 15.2. Finding a Minimal Cover F for a Set of Functional Dependencies E » Input: A set of functional dependencies E. 1. Set F:=E. 2. Replace each functional dependency X → {A1, A2, ..., An} in F by the n functional dependencies X →A1, X →A2, ..., X → An. 3. For each functional dependency X → A in F for each attribute B that is an element of X if { {F – {X → A} } ∪ { (X – {B}) → A} } is equivalent to F then replace X → A with (X – {B}) → A in F. (* The above constitutes a removal of the extraneous attribute B from X *) 4. For each remaining functional dependency X → A in F if {F – {X → A} } is equivalent to F, then remove X → A from F. (* The above constitutes a removal of the redundant dependency A from F *)→X given set of FDs E : {B → A, D → A, AB → D}.We have to find the minimum cover of E. ■ All above dependencies are in canonical form; so we have completed step 1 of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine if AB → D has any redundant attribute on the left-hand side; that is, can it be replaced by B → D or A → D? ■ Since B → A, by augmenting with B on both sides (IR2), we have BB → AB, or B → AB (i). However, AB → D as given (ii). ■ Hence by the transitive rule (IR3), we get from (i) and (ii), B → D. Hence AB → D may be replaced by B → D. ■ We now have a set equivalent to original E , say E′ : {B → A, D → A, B → D}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side. ■ In step 3 we look for a redundant FD in E′. By using the transitive rule on B → D and D → A, we derive B → A. Hence B → A is redundant in E' and can be eliminated. ■ Hence the minimum cover of E is {B → D, D → A}

8b.

Step 1: Original matrix at start of algorithm

	A	B	C	D	E	F	G	H	I	J
R1	a_1	a_2	a_3	a_4	a_5	b_{16}	b_{17}	b_{18}	b_{19}	b_{110}
R2	b_{21}	a_2	b_{23}	b_{24}	b_{25}	a_6	a_7	a_8	b_{29}	b_{210}
R3	b_{31}	b_{32}	b_{34}	a_4	b_{36}	b_{37}	b_{38}	b_{39}	a_9	a_{10}

Step 2: Matrix after applying the functional dependencies

	A	B	C	D	E	F	G	H	I	J
R1	a_1	a_2	a_3	a_4	a_5	~~b_{16}~~ a_6	~~b_{17}~~ a_7	~~b_{18}~~ a_8	~~b_{19}~~ a_9	~~b_{110}~~
R2	b_{21}	a_2	b_{23}	b_{24}	b_{25}	a_6	a_7	a_8	b_{29}	b_2
R3	b_{31}	b_{32}	b_{34}	a_4	b_{35}	b_{36}	b_{37}	b_{38}	a_9	a_1

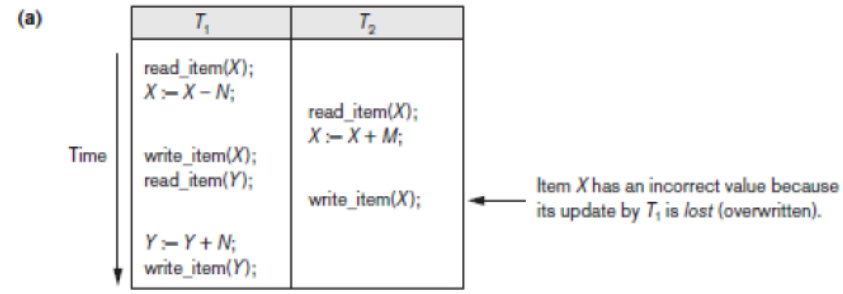
First row is all "a" symbols, hence it satisfies lossless join property.

MODULE 5

9a. Several problems can occur when concurrent transactions execute in an uncontrolled manner. Consider two transactions below:

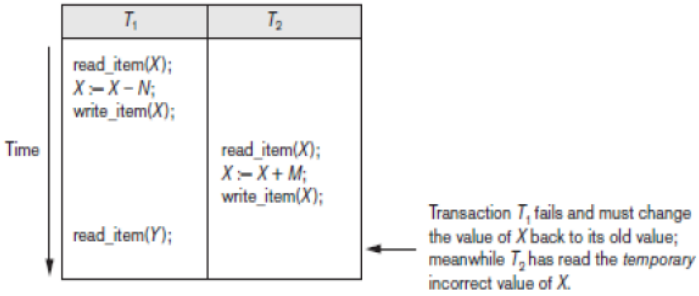
i) The Lost Update Problem:

(a)



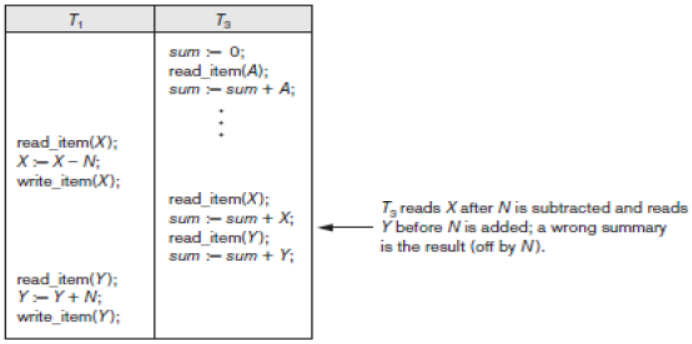
This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. Suppose that transactions *T*1 and *T*2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in Figure above; then the final value of item *X* is incorrect because *T*2 reads the value of *X before T*1 changes it in the database, and hence the updated value resulting from *T*1 is lost.

ii) The Temporary Update (or Dirty Read) Problem:



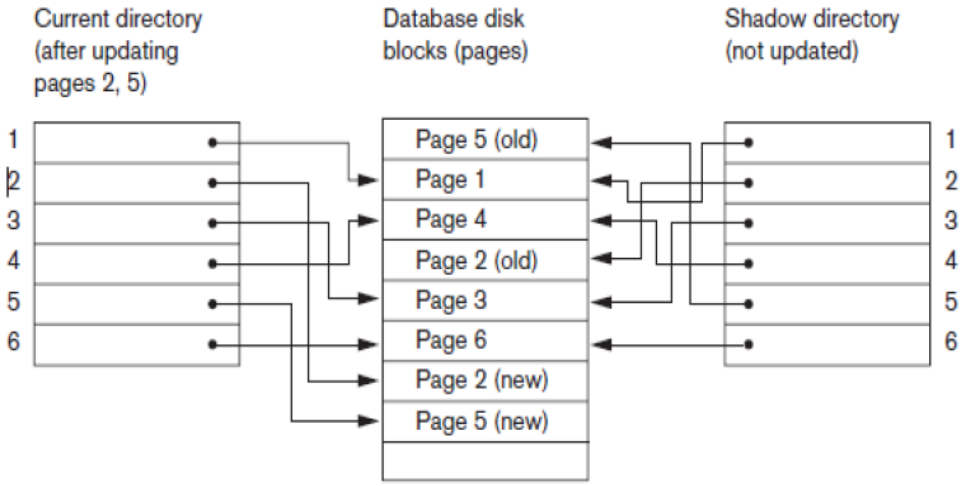
This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value. Figure shows an example where *T*1 updates item *X* and then fails before completion, so the system must roll back *X* to its original value. Before it can do so, however, transaction *T*2 reads the *temporary* value of *X*, which will not be recorded permanently in the database because of the failure of *T*1. The value of item *X* that is read by *T*2 is called *dirty data* because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the *dirty read problem*

iii) The Incorrect Summary Problem:



If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

	iv) The Unrepeatable Read Problem: Another problem that may occur is called *unrepeatable read*, where a transaction *T* reads the same item twice and the item is changed by another transaction *T'* between the two reads. Hence, *T* receives *different values* for its two reads of the same item. This may occur, for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights. When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.
9b.	Transactions should possess several properties, often called the **ACID** properties The following are the ACID properties: ■ **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all. ■ **Consistency preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, **it should take the database from one consistent state to another.** ■ **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.** ■ **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.
10a.	**Deadlock** occurs when *each* transaction *T* in a set of *two or more transactions* is waiting for some item that is locked by some other transaction *T'* in the set. Hence, each transaction in the set is in a waiting queue, waiting for one of the other transactions in the set to release the lock on an item. **Deadlock Prevention Protocols:** One way to prevent deadlock is to use a **deadlock-prevention protocol.** One deadlock prevention protocol, which is used in conservative two-phase locking, requires that every transaction lock *all the items it needs in advance.* A second protocol, which also limits concurrency, involves *ordering all the items* in the database and making sure that a transaction that needs several items will lock them according to that order. Some of these techniques use the concept of **transaction timestamp** TS(*T*), which is a unique identifier assigned to each transaction. Two schemes that prevent deadlock are called ***wait-die*** and ***wound-wait.*** □ **Wait-die.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) *Ti* is allowed to wait; otherwise (*Ti* younger than *Tj*) abort *Ti* (*Ti dies*) and restart it later *with the same timestamp.* □ ■ **Wound-wait.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) abort *Tj* (*Ti wounds Tj*) and restart it later *with the same timestamp;* otherwise (*Ti* younger than *Tj*) *Ti* is allowed to wait. In wait-die, an older transaction is allowed to *wait for a younger transaction*, whereas a younger transaction requesting an item held by an older transaction is aborted and restarted. Wound-wait does the opposite. Another group of protocols that prevent deadlock do not require timestamps. These include the **no waiting (NW)** and **cautious waiting (CW)** algorithms. In the **no waiting algorithm**, if a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not. Suppose that transaction *Ti* tries to lock an item *X* but is not able to do so because *X* is locked by some other transaction *Tj* with a conflicting lock. The cautious waiting rule is as follows: **Cautious waiting:** If *Tj* is not blocked (not waiting for some other locked item), then *Ti* is blocked and allowed to wait; otherwise abort *Ti*. **Deadlock Detection:** A simple way to detect a state of deadlock is for the system to construct and maintain a **wait-for graph**. One node is created in the wait-for graph for each transaction that is currently executing. Whenever a transaction *Ti* is waiting to lock an item *X* that is currently locked by a transaction *Tj*, a directed edge (*Ti* -> *Tj*) is created in the wait-for graph. When *Tj* releases the lock(s) on the items that *Ti* was waiting for, the directed edge is dropped from the wait-for graph. We have a state of deadlock if and only if the wait-for graph has a cycle. Another simple scheme to deal with deadlock is the use of **timeouts**. In this method, if a transaction

	waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it. **Starvation:** □ Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further. □ In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back. □ This limitation is inherent in all priority based scheduling mechanisms. In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.
10b.	Shadow paging considers the database to be made up of a number of fixed size disk pages (or disk blocks)—say, *n*—for recovery purposes. A **directory** with *n* entries is constructed, where the *i*th entry points to the *i*th database page on disk. The directory is kept in main memory if it is not too large, and all references—reads or writes—to database pages on disk go through it. When a transaction begins executing, the **current directory**—whose entries point to the most recent or current database pages on disk—is copied into a **shadow directory**. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory is *never* modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.* Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block.  For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory and the new version by the current directory.