Question Paper Solutions

UNIT-I

1) Define the following terms: 20 Marks (Jun/July 2013)

i) **Data base**

Ans: A database is a collection of related data, where data means recorded facts. Computer-based repositories for data.It is logically coherent data to which some meaning is attached.

ii) Canned transaction:

Ans: These are the transactions that are carefully programmed and tested in advance. Ex bank teller, check account balances post withdrawals/deposits.

iii) Data model:

Ans: It is a collection of concepts that can be used to describe the concepts/logical structure of a database which provides the necessary means to achieve their abstraction.

iv) Meta data:

Ans: It is the data about data which is stored in system catalog which contains description of the structure of each file the type and the storage format of each file and the various constraints on the data.

v) Data base designer:

Ans: They arte responsible for identifying the data to be stored and for choosing an appropriate way to organize it. They also define views for different categories of users.

2) Explain the characteristics of data base approach.

4 Marks (Jun /July2014/June/July 2015)

1. ***Self-Describing Nature of a Database System*** - it has a complete definition or description of the database structure and constraints. This definition is stored in the system **catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. This information stored in the system catalog is called, **Meta-data** and it describes the structure of the primary database. This allows the DBMS software to work with different databases (Fig 1.1)
2. ***Insulation between Programs and Data and Data Abstraction*** - Called **program-data**

independence. Allows changing data storage structures and operations without having to change the DBMS access programs. The structure of data files is stored in the DBMS catalog separately from the access programs.

3. ***Data Abstraction***: A data model is used to hide storage details and present the users with a conceptual view of the database.
4. ***Support of Multiple Views of the Data*** - Each users may see a different view of the database, which describes only the data of interest to that user. (fig 1.4)
5. ***Sharing of Data and Multi-user*** *Transaction Processing* - the DBMS must include *concurrency control* software to ensure that the result of multi-user access is correct.

3) What are the responsibilities of a data base administrators.

4 Marks (Dec/Jan 2014, Jun / July2013)

Ans: A Data base administrator oversees and manages the data base system. Duties include authorizing users to access the data base coordinating/monitoring its use, acquiring H/W and S/w for upgrade. DBA must have supporting staff. The DBA is responsible for problems such as breach of security or poor system response time.

4) Explain the typical components of a DBMS with a neat diagram.

8 Marks (Dec /Jan 2013, Dec/Jan 2014, Dec2014/Jan2015)

DBMS Component Modules

- A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored on disk.
- The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- The run-time database processor handles database accesses at run time. The query compiler handles high-level queries that are entered interactively.
- The pre-compiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access

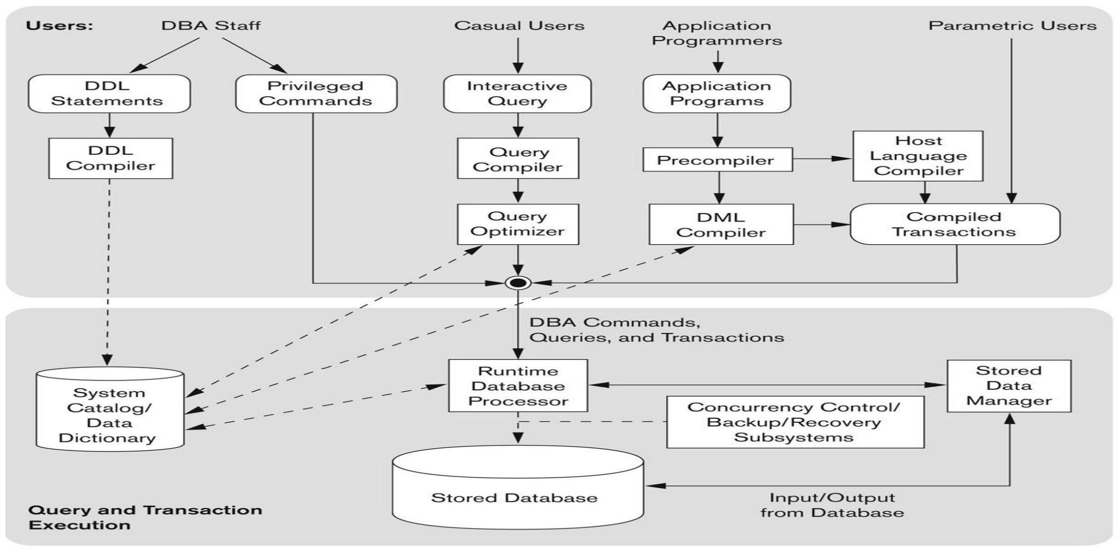


Figure 2.3
Component modules of a DBMS and their interactions.

5) Define and explain the following terms with an example for each.

8 Marks (Dec /Jan 2013, Jun / July2014)

i) Snapshot: The data in the database at a particular moment in time is called a database *state or snapshot.* A database state (also called *instances*) changes every time data is inserted, deleted, or modified

ii) Intension: The distinction between database schema and database state is that when we define a new database, we specify the database schema only to the DBMS The schema is sometimes called the intension.

iii) Extension: The distinction between database schema and database state is that when we define a new database, we specify the database schema only to the DBMS (the current state of the database is the *empty state* with no data). The database state is called an **extension** of the schema

6) What is meant by "persistent storage for program objects". Explain.

4 Marks (Dec/Jan 2013)

Providing Persistent Storage for Program Objects and Data Structures - objects survive the termination of program execution and can later be retrieved The persistent storage of programs objects and data structures is an important function of database systems.

7) Breifly discuss the advantages of using the DBMS 8 Marks (Dec/Jan 2014)

1. **Controlling Redundancy:** Data redundancy (such as tends to occur in the "file processing" approach) leads to **wasted storage space**, **duplication of effort** (when multiple copies of a datum need to be updated), and a higher liklihood of the introduction of **inconsistency**.On the other hand, redundancy can be used to improve performance of queries. Indexes, for example, are entirely redundant, but help the DBMS in processing queries more quickly.

Another example of using redundancy to improve performance is to store an "extra" field in order to avoid the need to access other tables (as when doing a JOIN, for example). See Figure 1.6 (page 18): the StudentName and CourseNumber fields need not be there.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated. (Figure 1.6 again, in which Student_name does not match Student_number.)

2. **Restricting Unauthorized Access:** A DBMS should provide a **security and authorization subsystem**, which is used for specifying restrictions on user accounts. Common kinds of restrictions are to allow read-only access (no updating), or access only to a subset of the data (e.g., recall the Bursar's and Registrar's office examples from above).
3. **Providing Persistent Storage for Program Objects:** Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.
4. **Providing Storage Structures for Efficient Query Processing:** The DBMS maintains indexes (typically in the form of trees and/or hash tables) that are utilized to improve the execution time of queries and updates. (The choice of which indexes to create and maintain is part of *physical database design and tuning* (see Chapter 16) and is the responsibility of the DBA.

   The **query processing and optimization** module is responsible for choosing an efficient query execution plan for each query submitted to the system. (See Chapter 15.)
5. **Providing Backup and Recovery:** The subsystem having this responsibility ensures that recovery is possible in the case of a system crash during execution of one or more transactions.
6. **Providing Multiple User Interfaces:** For example, query languages for casual users,

programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone users.

7. **Representing Complex Relationships Among Data:** A DBMS should have the capability to represent such relationships and to retrieve related data quickly.
8. **Enforcing Integrity Constraints:** Most database applications are such that the semantics (i.e., meaning) of the data require that it satisfy certain restrictions in order to make sense. Perhaps the most fundamental constraint on a data item is its data type, which specifies the universe of values from which its value may be drawn. (E.g., a Grade field could be defined to be of type Grade_Type, which, say, we have defined as including precisely the values in the set { "A", "A-", "B+", ..., "F" }.

   Another kind of constraint is *referential integrity*, which says that if the database includes an entity that refers to another one, the latter entity must exist in the database. For example, if (R56547, CIL102) is a tuple in the Enrolled_In relation, indicating that a student with ID R56547 is taking a course with ID CIL102, there *must be* a tuple in the Student relation corresponding to a student with that ID.
9. **Permitting Inferencing and Actions Via Rules:** In a **deductive** database system, one may specify *declarative* rules that allow the database to infer new data! E.g., Figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

   Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

8) Discuss the main Characteristics of the database approach. How does it differ from Traditional file systems? ` 8 Marks (Jun / July 2013, Jun / July2014)

Characteristics of the Database Approach are

Self-Describing Nature of a Database System - it has a complete definition or description of the database structure and constraints. This definition is stored in the system **catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. This information stored in the system catalog is called, **Meta-data** and it describes the structure of the primary database. This allows the DBMS software to work with different databases (Fig 1.1)

Insulation between Programs and Data and Data Abstraction - Called **program-data independence**. Allows changing data storage structures and operations without having to change the DBMS access programs. The structure of data files is stored in the DBMS catalog separately from the access programs.

Data Abstraction: A data model is used to hide storage details and present the users with a conceptual view of the database.

Support of Multiple Views of the Data - Each users may see a different view of the database, which describes only the data of interest to that user. (fig 1.4)

Sharing of Data and Multiuser *Transaction Processing* - the DBMS must include *concurrency control* software to ensure that the result of multiuser access is correct.

9) Explain the difference between Logical and physical data Independence?

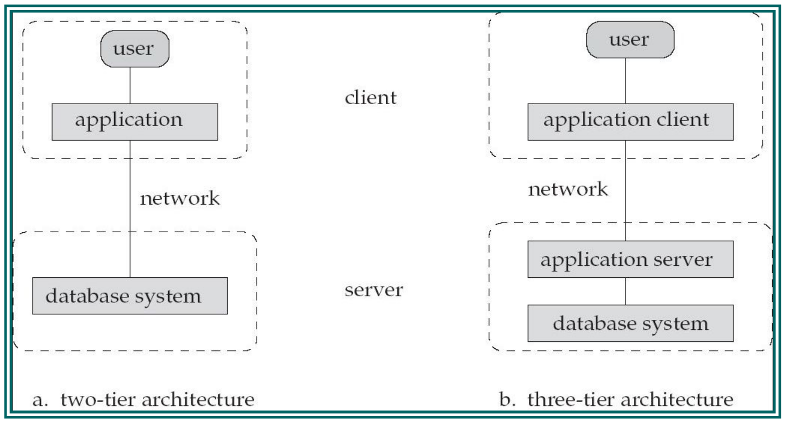
8 Marks (Jun / July 2013)

One important characteristic of the database approach was the insulation of programs and data (see Chapter 1). We can define two types of data independence:

logical data independence - the capacity to change the conceptual scheme without having to change external schemas or application programs.

Physical data independence - the capacity to change the internal scheme without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized.

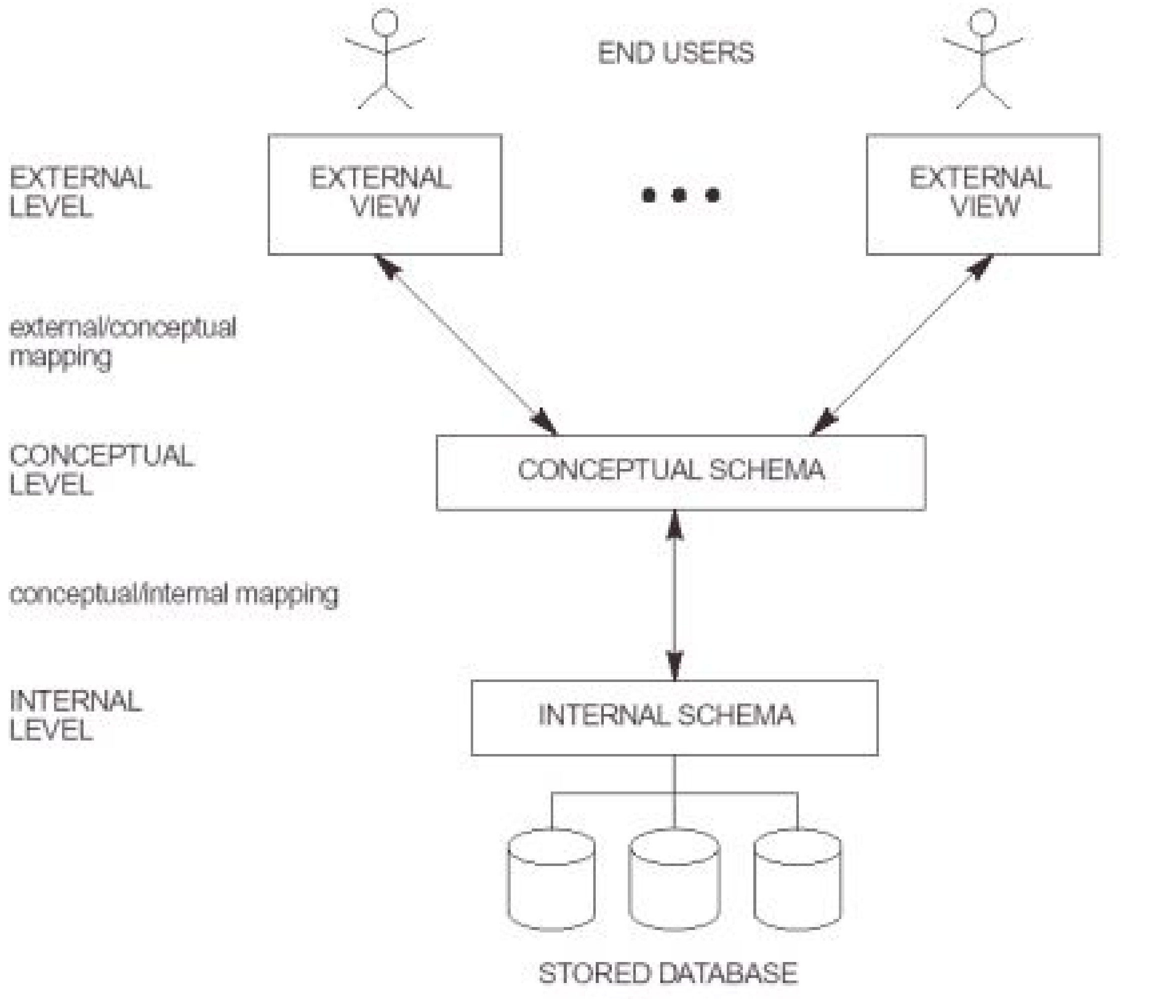
c) Explain the Operation of Two-Tier Client/Server architecture for RDBMS?



A client program may connect to several DBMSs.

Other variations of clients are possible: e.g., in some DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc. In such situations the server may be called the **Data Server**

10. **Explain the three-schema architecture. What is the logical data independence and physical data independence? 8 marks (June/July 2015)**



One important characteristic of the database approach was the insulation of programs and data (see Chapter 1). We can define two types of data independence:

logical data independence - the capacity to change the conceptual scheme without having to change external schemas or application programs.

Physical data independence - the capacity to change the internal scheme without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized.

11) Define the database and briefly explain the implicit properties of the database?

4 Marks (June/July 2015)

The main implicit properties of a database are listed below:

1. A database represents some aspect of the real world, sometimes called the miniworld or universe of discourse (UoD). Changes to the miniworld are reflected in the database.
2. A database is a logically coherent collection of data with some inherent meaning.
3. A database is designed, built, and populated with data for a specific purpose.

12) Define the following with examples: 10 marks (Dec 14/Jan 15)

i) Value set it consists of set of values

ii) Complex attributes We refer to an attribute that involves some combination of multi-valued *and* compositeness as a **complex** attribute.

iii) Data model a collection of concepts that can be used to describe the conceptual/logical structure of a database--- provides the necessary means to achieve this abstraction

iv) Schema constructs which is specified during design and is not expected to change often.

v) Meta data (i.e., data about data) is stored in the so-called **system catalog**,

UNIT-II

Entity-Relationship Model

1. List the summary of the notations for ER diagrams. Include symbols used in ER diagram and their meaning. 8 Marks (Jun / July2014)

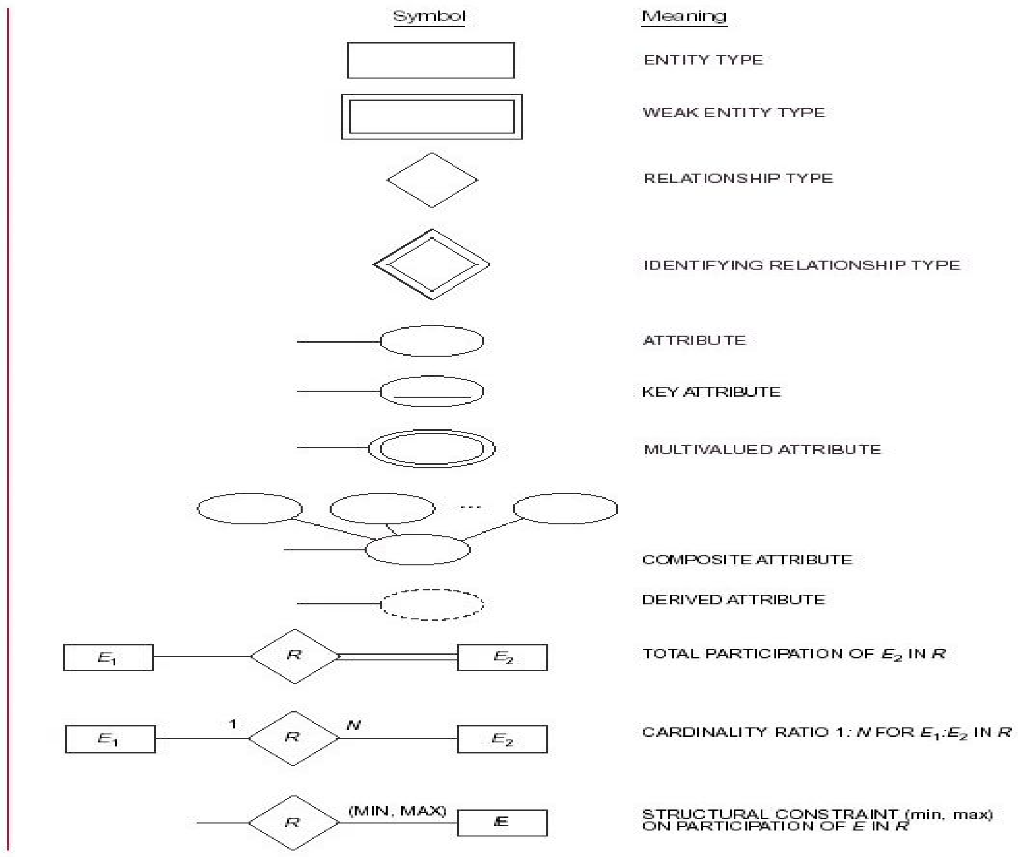

Symbol
Meaning
ENTITY TYPE
WEAK ENTITY TYPE
RELATIONSHIP TYPE
IDENTIFYING RELATIONSHIP TYPE
ATTRIBUTE
KEY ATTRIBUTE
MULTIVALUED ATTRIBUTE
...
COMPOSITE ATTRIBUTE
DERIVED ATTRIBUTE
E_1
R
E_2
TOTAL PARTICIPATION OF E_2 IN R
1
N
CARDINALITY RATIO 1:N FOR E_1:E_2 IN R
(MIN, MAX)
E
STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

Figure 3.14 Summary of ER diagram notation.

2) With respect to ER model explain with example. 4 Marks (Dec/Jan 2013)

i) **Strong entity**: Regular entity types that do have a key attribute are sometimes called *strong entity types.*

ii) **Weak entity**: entity types that do not have key attributes of their own are called *weak entity types*

iii) **Participation constraints:** It specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifiesthe minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint.

iv) **Cardinality ratio:** The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

v) **Recurring relationships:** In some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called recursive relationship

Atomic Atribute:

Simple. Attribute that are not divisible are called simple or atomic attribute.

For example, Street_name or Door_number. i.e. the division of composite attribute.

Recursive Relationship Type

An relationship type whose with the same participating entity type in distinct roles Example: In the SUPERVISION relationship EMPLOYEE participates twice in two distinct roles: supervisor (or boss) role

supervisee (or subordinate) role Each relationship instance relates two distinct EMPLOYEE entities:

One employee in supervisor roleOne employee in supervisee role

Weak Entity Types

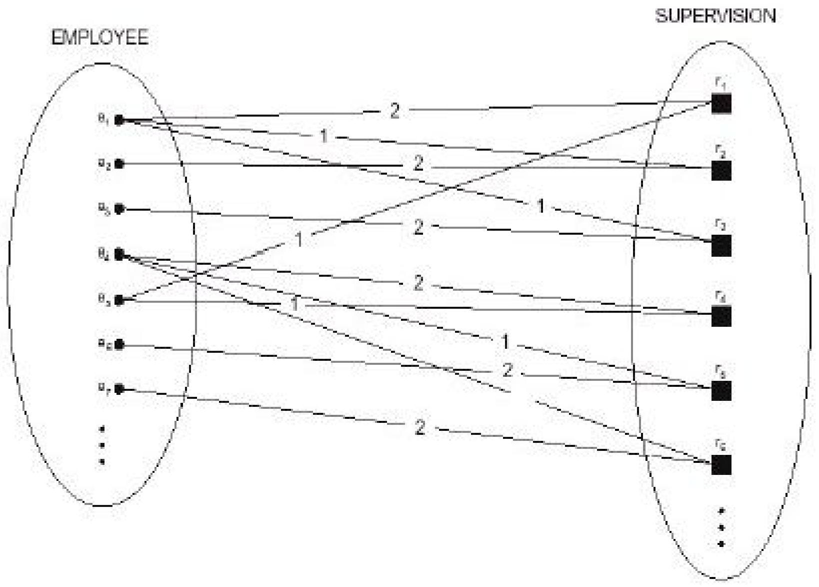
An entity that does not have a key attribute. A weak entity must participate in an identifying relationship type with an owner or identifying entity type. Entities are identified by the combination of: A partial key of the weak entity type The particular entity they are related to in the identifying entity type.

3. Explain hoe role names are assigned in case of recursive relationships? Illustrate this concept with an example. . **8 Marks (Jun / July 2013)**

A relationship can relate two entities of the same entity type ; for example, a SUPERVISION

relationship type relates one EMPLOYEE (in the role of supervisee) to another EMPLOYEE (in the role of supervisor). This is called a recursive relationship type

Figure 3.11 The recursive relationship SUPERVISION, where the EMPLOYEE entity type plays the two roles of supervisor (1) and supervisee (2).



4) What is meant by partial key? Explain. **4 Marks (Jun / July2014)**

A weak entity type must participate in an **identifying relationship type** with an **owner** or **identifying entity type**

Entities are identified by the combination of:

A partial key of the weak entity type

The particular entity they are related to in the identifying entity type

A weak entity type usually has a *partial key*, which is the set of attributes that can uniquely identify weak entities that are *related to the same owner entity.* For example, Name of

DEPENDENT is the partial key

5. Define an entity and an attribute,explain the different types of attributes that occur in an ERdiagram model,with an example **8 Marks (Dec /Jan 2013, Dec/Jan 2014)**

Our focus now is on the second phase, **conceptual design**, for which The **Entity-Relationship**

(ER) Model is a popular high-level conceptual data model.

In the ER model, the main concepts are **entity**, **attribute**, and **relationship**.

Entities and Attributes

Entity: An entity represents some "thing" (in the miniworld) that is of interest to us, i.e., about which we want to maintain some data. An entity could represent a physical object (e.g., house, person, automobile, widget) or a less tangible concept (e.g., company, job, academic course).

Attribute: An entity is described by its attributes, which are properties characterizing it. Each attribute has a **value** drawn from some **domain** (set of meaningful values).

Example: A *PERSON* entity might be described by *Name*, *BirthDate*, *Sex*, etc., attributes, each having a particular value.

What distinguishes an entity from an attribute is that the latter is strictly for the purpose of describing the former and is not, in and of itself, of interest to us. It is sometimes said that an entity has an independent existence, whereas an attribute does not. In performing data modeling, however, it is not always clear whether a particular concept deserves to be classified as an entity or "only" as an attribute.

We can classify attributes along these dimensions:

- simple/atomic vs. composite
- single-valued vs. multi-valued (or set-valued)
- stored vs. derived (*Note from instructor:* this seems like an implementational detail that ought not be considered at this (high) level of abstraction.)

A **composite** attribute is one that is *composed* of smaller parts. An **atomic** attribute is indivisible or indecomposable.

- **Example 1**: A *BirthDate* attribute can be viewed as being composed of (sub-) attributes for month, day, and year.
- **Example 2**: An *Address* attribute (Figure 3.4, page 64) can be viewed as being composed of (sub-)attributes for street address, city, state, and zip code. A street address can itself be viewed as being composed of a number, street name, and apartment number. As thissuggests, composition can extend to a depth of two (as here) or more. To describe the structure of a composite attribute, one can draw a tree (as in the aforementioned Figure 3.4). In case we are limited to using text, it is customary to write its name followed by a parenthesized list of its sub-attributes. For the examples mentioned above, we would write

BirthDate(Month, Day, Year)

Address(StreetAddr(StrNum, StrName, AptNum), City, State, Zip)

Single- vs. multi-valued attribute: Consider a *PERSON* entity. The person it represents has (one) *SSN*, (one) *date of birth*, (one, although composite) *name*, etc. But that person may have zero or more academic degrees, dependents, or (if the person is a male living in Utah) spouses! How can we model this via attributes *AcademicDegrees*, *Dependents*, and *Spouses*? One way is to allow such attributes to be *multi-valued* (perhaps *set-valued* is a better term), which is to say that we assign to them a (possibly empty) *set* of values rather than a single value. To distinguish a multi-valued attribute from a single-valued one, it is customary to enclose the former within curly braces (which makes sense, as such an attribute has a value that is a set, and curly braces are traditionally used to denote sets). Using the *PERSON* example from above, we would depict its structure in text as

PERSON(SSN, Name, BirthDate(Month, Day, Year), { AcademicDegrees(School, Level, Year) },

{ Dependents }, ...)

Here we have taken the liberty to assume that each academic degree is described by a school, level (e.g., B.S., Ph.D.), and year. Thus, *AcademicDegrees* is not only multi-valued but also composite. We refer to an attribute that involves some combination of multi-valuedness *and* compositeness as a **complex** attribute.

A more complicated example of a complex attribute is *AddressPhone* in Figure 3.5 (page 65). This attribute is for recording data regarding addresses and phone numbers of a business. The structure of this attribute allows for the business to have several offices, each described by an address and a set of phone numbers that ring into that office. Its structure is given by

{ AddressPhone({ Phone(AreaCode, Number) }, Address(StrAddr(StrNum, StrName, AptNum),

City, State, Zip)) }

Stored vs. derived attribute: Perhaps *independent* and *derivable* would be better terms for these (or *non-redundant* and *redundant*). In any case, a *derived* attribute is one whose value can be calculated from the values of other attributes, and hence need not be stored. **Example:** *Age* can be calculated from *BirthDate*, assuming that the current date is accessible. **The Null value**: In some cases a particular entity might not have an applicable value for a particular attribute. Or that value may be unknown. Or, in the case of a multi-valued attribute, the appropriate value might be the empty set.

Example: The attribute *DateOfDeath* is not applicable to a living person and its correct value may be unknown for some persons who have died.

In such cases, we use a special attribute value (non-value?), called **null**. There has been some argument in the database literature about whether a different approach (such as having distinct values for *not applicable* and *unknown*) would be superior

6. Define the following with an example. 20 Marks (Jun/July 2013 & June/July 2105)

Weak Entity Types: An entity type that has no set of attributes that qualify as a key is called **weak**. (Ones that do are **strong**.). An entity of a weak identity type is uniquely identified by the specific entity to which it is related (by a so-called **identifying relationship** that relates the weak entity type with its so-called **identifying** or **owner entity type**) in combination with some set of its own attributes (called a *partial key*).

Example: A *DEPENDENT* entity is identified by its first name together with the *EMPLOYEE* entity to which it is related via DEPENDS_ON. (Note that this wouldn't work for former heavyweight boxing champion George Foreman's sons, as they all have the name "George"!)

Because an entity of a weak entity type cannot be identified otherwise, that type has a **total participation constraint** (i.e., **existence dependency**) with respect to the identifying relationship.

This should not be taken to mean that any entity type on which a total participation constraint exists is weak. For example, DEPARTMENT has a total participation constraint with respect to MANAGES, but it is not weak.

In an ER diagram, a weak entity type is depicted with a double rectangle and an identifying relationship type is depicted with a double diamond

ii)Participation constraint

- **participation**: specifies whether or not the existence of an entity depends upon its being related to another entity via the relationship.

total participation (or existence dependency): To say that entity type *A* is constrained to **participate totally** in relationship *R* is to say that if (at some moment in time) *R*'s instance set is

$\{ (a_1, b_1), (a_2, b_2), \ldots (a_m, b_m) \}$,

then (at that same moment) *A*'s instance set must be $\{ a_1, a_2, \ldots, a_m \}$. In other words, there can be no member of *A*'s instance set that does not participate in at least one instance of *R*. According to our informal description of COMPANY, every employee must be assigned to some department. That is, every employee instance must participate in at least one instance of WORKS_FOR, which is to say that *EMPLOYEE* satisfies the total participation constraint with respect to the WORKS_FOR relationship. In an ER diagram, if entity type *A* must participate totally in relationship type R, the two are connected by a double line.

partial participation: the absence of the total participation constraint! (E.g., not every employee has to participate in MANAGES; hence we say that, with respect to MANAGES, *EMPLOYEE* participates partially. This is not to say that for all employees to be managers is not allowed; it only says that it need not be the case that all employees are managers.

i) Cardinality ratio

Often, in order to make a relationship type be an accurate model of the miniworld concepts that it is intended to represent, we impose certain constraints that limit the possible corresponding relationship sets. (That is, a constraint may make "invalid" a particular set of instances for a relationship type.)

There are two main kinds of relationship constraints (on binary relationships). For illustration, let *R* be a relationship set consisting of ordered pairs of instances of entity types *A* and *B*, respectively.

Cardinality ratio: **1:1 (one-to-one)**: Under this constraint, no instance of *A* may particpate in more than one instance of *R*; similarly for instances of *B*. In other words, if (a_1, b_1) and (a_2, b_2) are (distinct) instances of *R*, then neither $a_1 = a_2$ nor $b_1 = b_2$.

Example: Our informal description of COMPANY says that every department has one employee who manages it. If we also stipulate that an employee may not (simultaneously) play the role of manager for more than one department, it follows that MANAGES is 1:1.

1:N (one-to-many): Under this constraint, no instance of *B* may participate in more than one instance of *R*, but instances of *A* are under no such restriction. In other words, if (a_1, b_1) and (a_2, b_2) are (distinct) instances of *R*, then it cannot be the case that $b_1 = b_2$.

Example: CONTROLS is 1:N because no project may be controlled by more than one department. On the other hand, a department may control any number of projects, so there is no restriction on the number of relationship instances in which a particular department instance may participate. For similar reasons, is also 1:N. **N:1 (many-to-one)**: This is just the same as 1:N but with roles of the two entity types reversed.

Example: WORKS_FOR and DEPENDS_ON are N:1.

M:N (many-to-many): Under this constraint, there are no restrictions. (Hence, the term applies to the absence of a constraint!)

Example: WORKS_ON is M:N, because an employee may work on any number of projects and a project may have any number of employees who work on it. Notice the notation in Figure 3.2 for indicating each relationship type's cardinality ratio. Suppose that, in designing a database, we decide to include a binary relationship *R* as described above (which relates entity types *A* and *B*, respectively). To determine how *R* should be constrained, with respect to cardinality ratio, the questions you should ask are

these:May a given entity of type B be related to multiple entities of type A?

May a given entity of type A be related to multiple entities of type B?

The pair of answers you get maps into the four possible cardinality ratios as follows:

(yes, yes) --> M:N

(yes, no) --> N:1

(no, yes) --> 1:N

(no, no) --> 1:1

VI) Ternary relationship

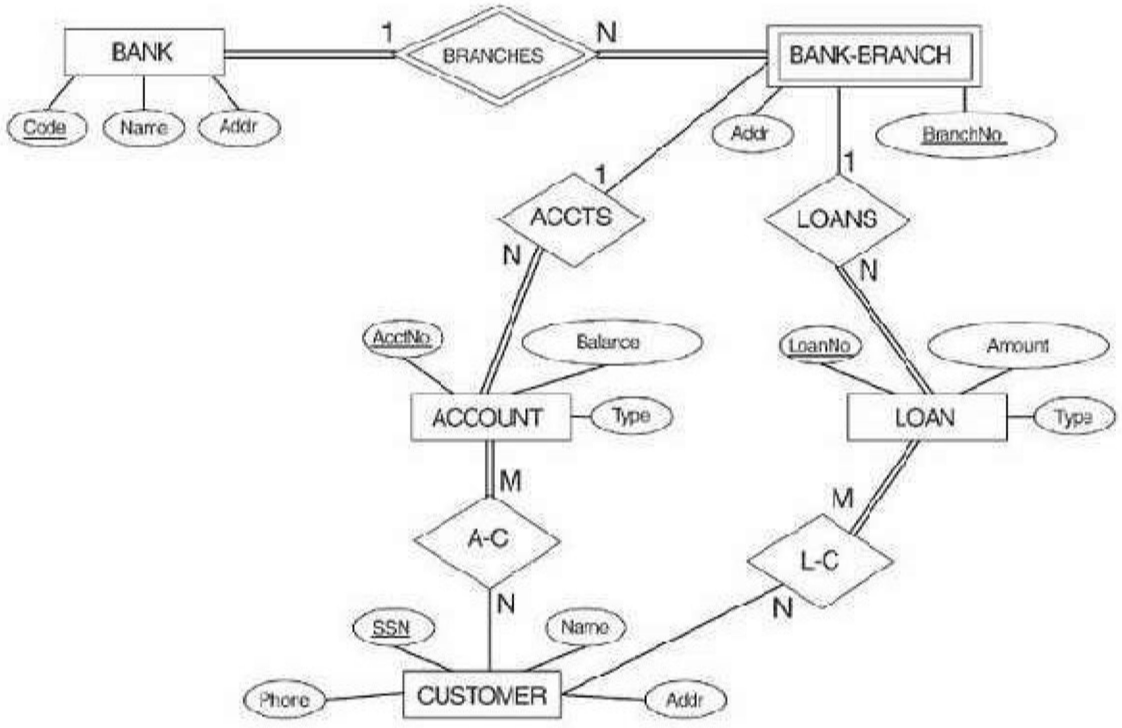
Also note that, in our COMPANY example, all relationship instances will be ordered pairs, as each relationship associates an instance from one entity type with an instance of another (or the same, in the case of SUPERVISES) relationship type. Such relationships are said to be *binary*, orto have *degree* two. Relationships with degree three (called *ternary*) or more are also possible, although not as common.

SUPPLY (perhaps not the best choice for a name) has as instances ordered triples of suppliers, parts, and projects, with the intent being that inclusion of the ordered triple *(s_2, p_4, j_1)*, for example, indicates that supplier s_2 supplied part p_4 to project j_1*)*.

V) Recursive relationship

An exception to this rule occurs when the same entity type plays two (or more) roles in the same relationship. (Such relationships are said to be *reCURsive*, which I find to be a misleading use of that term. A better term might be *self-referential*.) For example, in each instance of a SUPERVISES relationship set, one employee plays the role of *supervisor* and the other plays the role of *supervisee*.

7) Design an ER Diagram for keeping track of Information about Bank Database,Taking into account 4 entities? 8 Marks (Dec /Jan 2013)



8) Describe how to map the following Scenario's in ER Model to schema,with suitable example: 8 Marks (Dec/Jan 2014)

i) Strong Entity:-

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes.
- For composite attributes in E, convert them to simple attributes in R by converting each component to an attribute.
- For multi-value attributes in E, leave them to step 6
- Choose one of the key attributes of E as the primary key

  for R
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R

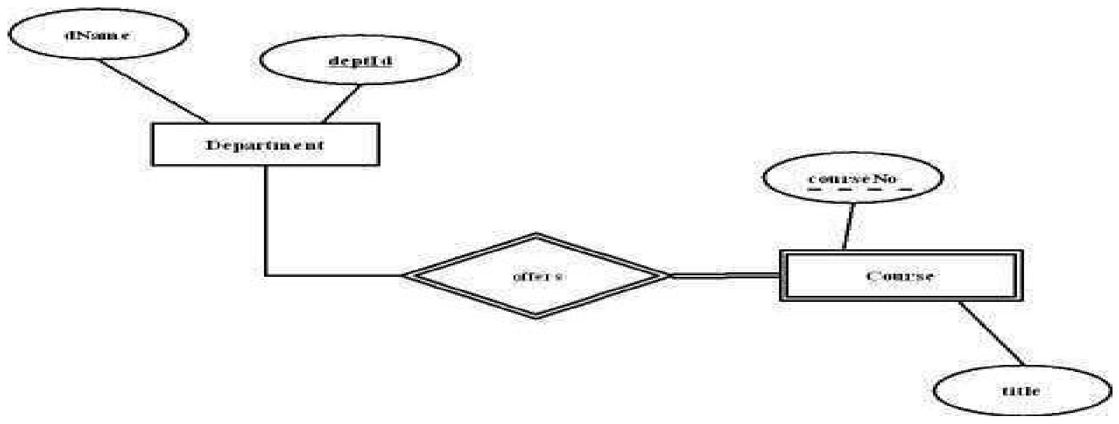
ii) One-to-one Relationship

- Mapping of Binary Relation Types with 1:1 cardinality ratio (no new relation is created) – 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.

- There are three possible approaches:
- Foreign Key approach: Choose one of the relations – say S – and include a foreign key in S the primary key of T. It is better to
- choose an entity type with total participation in R in the role of S.
- Include in S all the attributes from R
- Example: 1:1 relation MANAGES is mapped by choosing the
- participating entity type DEPARTMENT to serve in the role of S, because
- its participation in the MANAGES relationship type is total.

9) Define and explain Partial Key, with example? 4 Marks (Dec/Jan 2014, Jun / July2013)

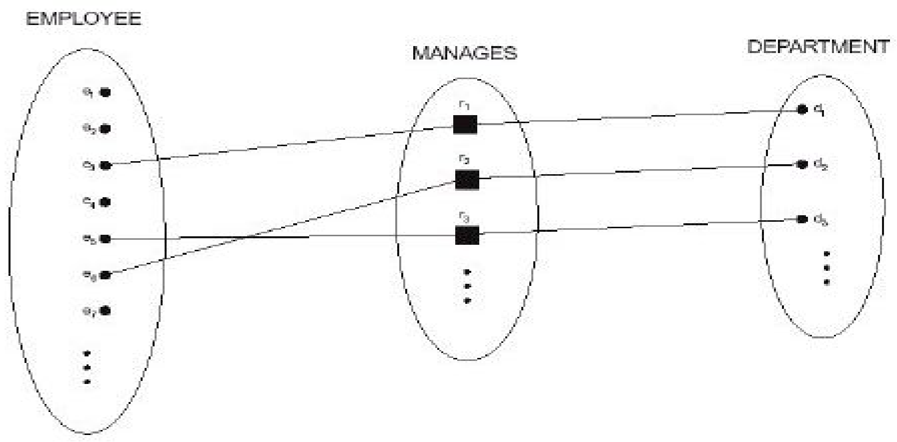
An attribute is a Partial Key if a Key from a related entity type must be used in conjunction with the attribute in question to uniquely identify instances of a corresponding entity set.



10) What is meant by recursive relationship?Bring out the importance of role names in ecursive relationship, with an example? 8 Marks (Jun / July 2013)

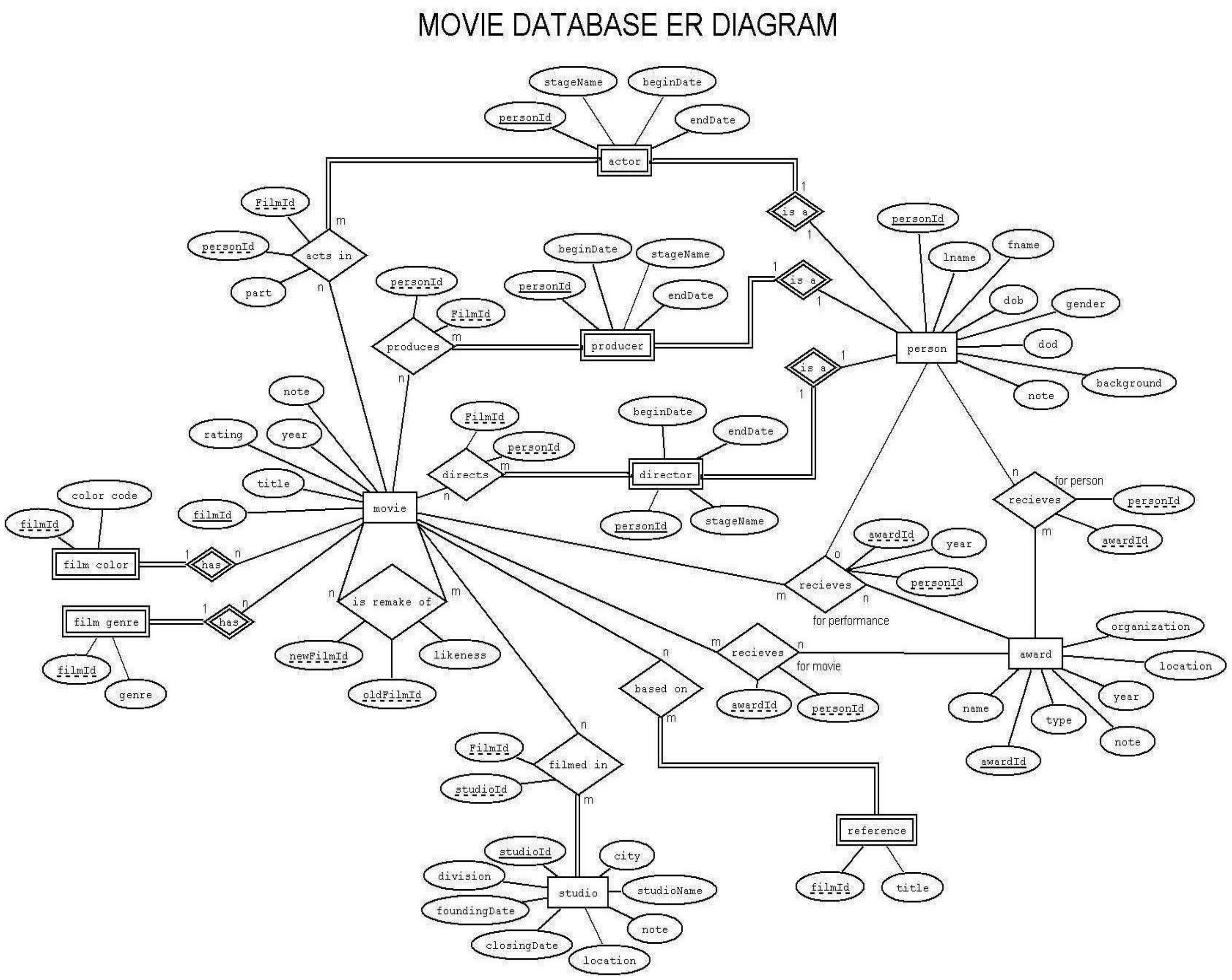
A relationship can relate two entities of the same entity type ; for example, a SUPERVISION relationship type relates one EMPLOYEE (in the role of supervisee) to another EMPLOYEE (in the role of supervisor). This is called a recursive relationship type. A relationship type can have attributes; for example, HoursPerWeek of WORKS_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT. Structural constraints on relationships: Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N. Participation constraint (on each participating entity type): total (called existence dependency) or partial.

Figure 3.12 The 1:1 relationship MANAGES, with partial participation of employee and total participation of DEPARTMENT.

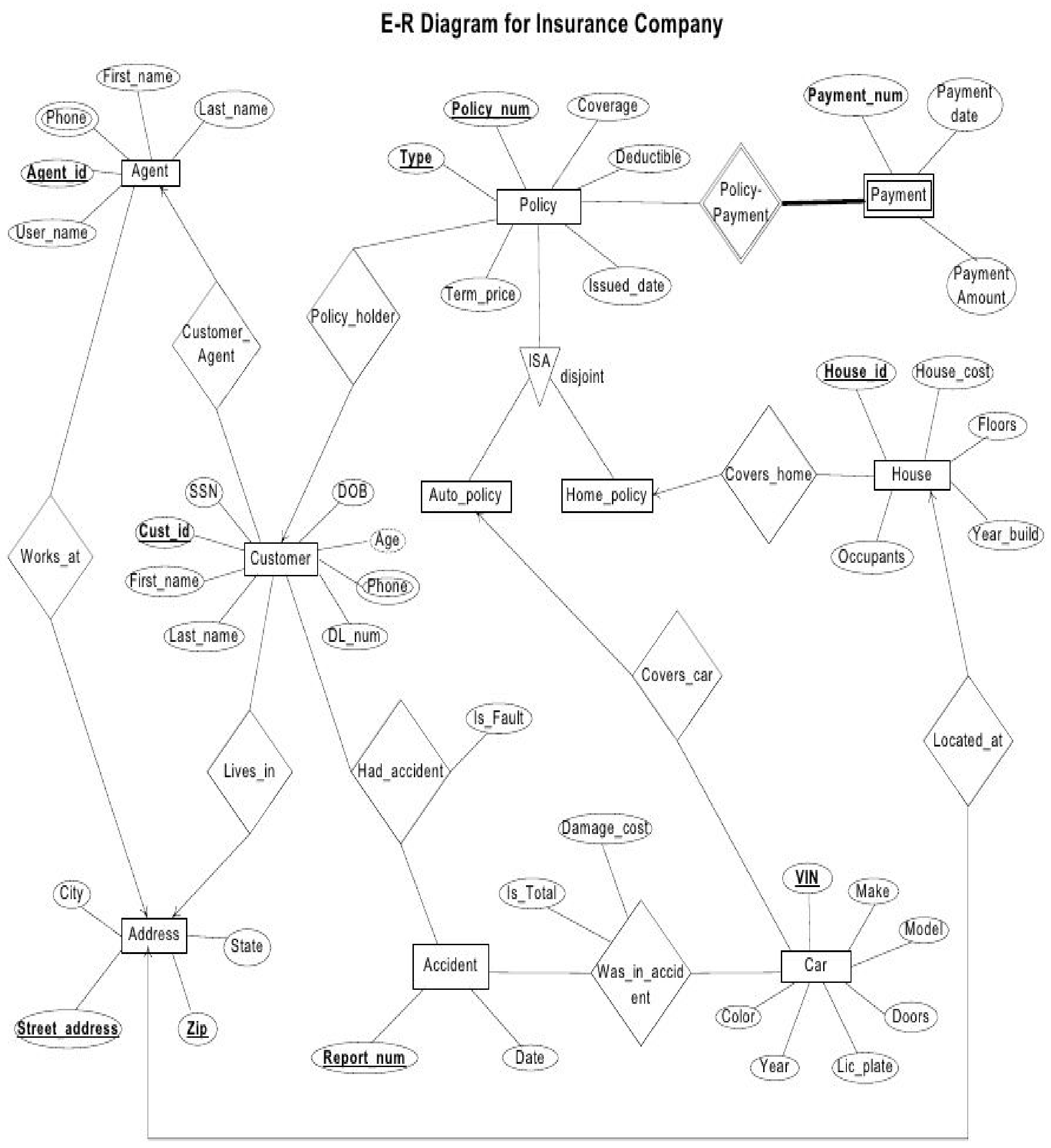


Alternative (min, max) notation for relationship structural constraints: Specified on each participation of an entity type E in a relationship type R. Specifies that each entity e in E participates in at least min and at most max relationship instances in R. Default(no constraint): min=0, max=n.

11) Draw the ER –diagram of musician who performs for album. Assume any four entities. Indicate all key and constraints and assumptions that are made? 8 Marks (Jun / July2014)



12. Design an ER diagram for an insurance company. Assume suitable entity types like CUSTOMER, AGENT, BRANCH, POLICY, PAYMENT and the relation between them. **10 marks (June/ July 2015) (Dec 14/Jan 15)**



13 What are structural constraints on relationship types? Explain with an example?

10 Marks (Dec 14/Jan 15)

participation: specifies whether or not the existence of an entity depends upon its being related to another entity via the relationship.

total participation (or existence dependency): To say that entity type *A* is constrained to **participate totally** in relationship *R* is to say that if (at some moment in time) *R*'s instance set is

$\{ (a_1, b_1), (a_2, b_2), \ldots (a_m, b_m) \}$,

then (at that same moment) *A*'s instance set must be $\{ a_1, a_2, \ldots, a_m \}$. In other words, here can be no member of *A*'s instance set that does not participate in at least one instance of *R*. According to our informal description of COMPANY, every employee must be assigned to some department. That is, every employee instance must participate in at least one instance of WORKS_FOR, which is to say that *EMPLOYEE* satisfies the total participation constraint with respect to the WORKS_FOR relationship. In an ER diagram, if entity type *A* must participate totally in relationship type R, the two are connected by a double line.

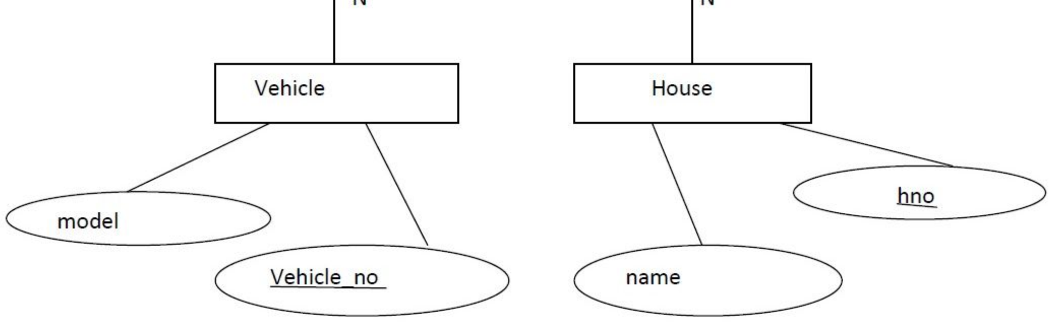
partial participation: the absence of the total participation constraint! (E.g., not every employee has to participate in MANAGES; hence we say that, with respect to MANAGES, *EMPLOYEE* participates partially. This is not to say that for all employees to be managers is not allowed; it only says that it need not be the case that all employees are managers.

14) What are weak entity type? Explain the role of partial key in design of weak entity type?

5 Marks (Dec 14/Jan 15)

An entity type that has no set of attributes that qualify as a key is called **weak**. (Ones that do are **strong**.) An entity of a weak identity type is uniquely identified by the specific entity to which it is related (by a so-called **identifying relationship** that relates the weak entity type with its so-called **identifying** or **owner entity type**) in combination with some set of its own attributes (called a *partial key*).

Example: A *DEPENDENT* entity is identified by its first name together with the *EMPLOYEE* entity to which it is related via DEPENDS_ON. (Note that this wouldn't work for former heavyweight boxing champion George Foreman's sons, as



UNIT-III

Relational Model and Relational Algebra

1) Define the following terms with an example for each. 8 Marks (Jun / July2014)

Super key: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t_1 and t_2 in r(R), for any two distinct tuples t_1 and t_2 in a relation state *r* of *R* we have (the subset of attributes *SK* is called a *superkey* of the relation schema *R*).

Domain: A set/universe of atomic values, where by "atomic" we mean simply that from the point of view of the database each value in the domain is indivisible.

Tuple: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity in the miniworld.

Nulls: The null value is don't know, not applicable

Relational database schema: It is a set of schemas for its relations together with a set of integrity constraints.

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 keys implies that we cannot identify some tuples. Key constraints and entity constraints are specified on individual relations.

2) Explain the following terms 8 Marks (Dec/Jan 2014)

Domain constraint: Each attribute value must be NULL or drawn from the domain of that attribute. Note that some DBMS"s allow you to impose the not null constraint upon an attribute, which is to say that attribute may not have the value-null.

Semantic integrity constraint: These are application specific restrictions that are unlikely to be expressible in DDL.

Functional dependency constraint : It is a constraint b/w the two sets of attributes from two relations.

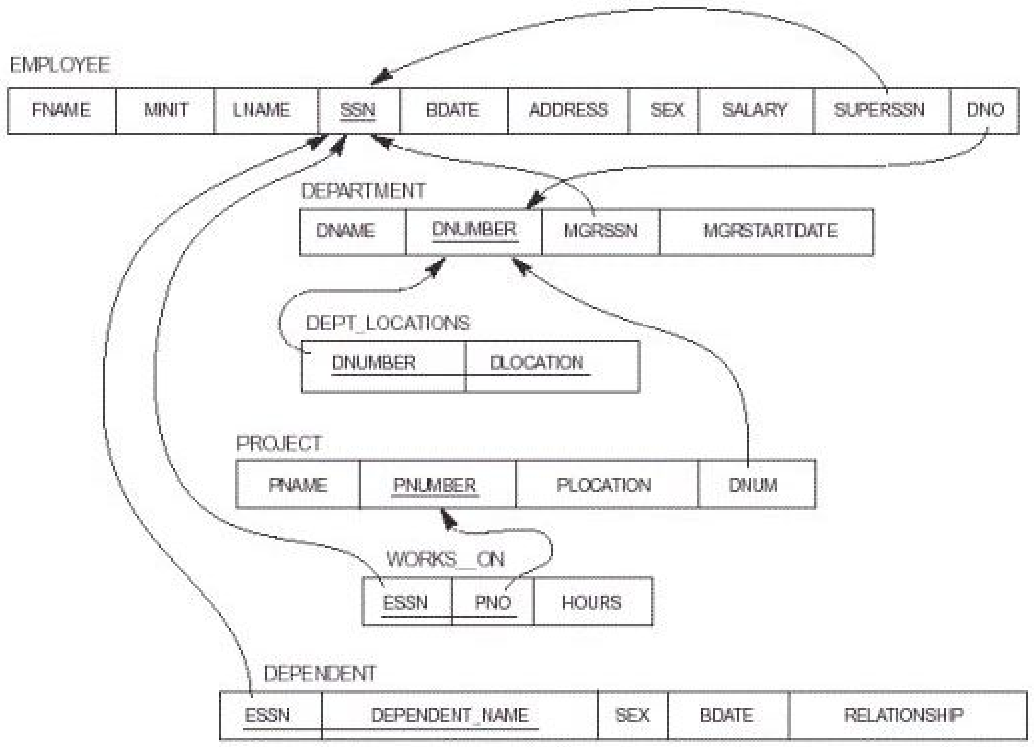
3) Define referential integrity constraint. Explain the importance of referential integrity constraint. How this constraint is implemented

8 Marks (Jun / July 2013) (5 Marks Dec 14/Jan 15)

Referential integrity constraint - The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. A tuple in one relation that refers to another relation must refer to an existing tuple in that relation. For example, the value of attribute DNO in every EMPLOYEE tuple must match the

DNUMBER value of some tuple in the DEPARTMENT relation.

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



4) Explain: 12 Marks (Jun/July 2013)

Domain constraint: Each attribute value must be NULL or drawn from the domain of that attribute. Note that some DBMS"s allow you to impose the not null constraint upon an attribute, which is to say that attribute may not have the value-null.

Semantic integrity constraint: These are application specific restrictions that are unlikely to be expressible in DDL.

Functional dependency constraint: It is a constraint b/w the two sets of attributes from two relations.

5) Discuss the characteristics of a relation, with an example 4 Marks (Dec/Jan 2013)

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. (Note that,

unfortunately, the set theoretic operations in relational algebra (at least how E&N define them) make implicit use of the order of the attributes. Hence, E&N view attributes as being arranged as a sequence rather than a set.)

Values of Attributes: For a relation to be in *First Normal Form*, each of its attribute domains

must consist of atomic (neither composite nor multi-valued) values. Much of the theory underlying the relational model was based upon this assumption.

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

: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.

Relational Model Notation: page 152 *R*(A_1, A_2, ..., A_n) is a relational schema of degree *n* denoting that there is a relation *R* having as its attributes A_1, A_2, ..., A_n.

- By convention, *Q*, *R*, and *S* denote relation names.
- By convention, *q*, *r*, and *s* denote relation states. For example, *r(R)* denotes one possible state of relation *R*. If *R* is understood from context, this could be written, more simply, as
- By convention, *t*, *u*, and *v* denote tuples.
- The "dot notation" *R.A* (e.g., STUDENT.Name) is used to qualify an attribute name, usually for the purpose of distinguishing it from a same-named attribute in a different relation

(e.g., DEPARTMENT.Name).

6) Briefly discuss the different types of update operations on relational database. show an example of a violation of referential integrity in each of the update operation For each of the *update* operations (Insert, Delete, and Update), we consider what kinds constraint violations may result from applying it and how we might choose to react.

8 Marks (Dec /Jan 2013, Dec/Jan 2014)

Insert:

- domain constraint violation: some attribute value is not of correct domain
- entity integrity violation: key of new tuple is **null**
- key constraint violation: key of new tuple is same as existing one
- referential integrity violation: foreign key of new tuple refers to non-existent tuple

Ways of dealing with it: reject the attempt to insert! Or give user opportunity to try again with different attribute values.

Delete:

- referential integrity violation: a tuple referring to the deleted one exists.

Three options for dealing with it:

- Reject the deletion
- Attempt to **cascade** (or **propagate**) by deleting any referencing tuples (plus those that

reference them, etc., etc.)

- modify the foreign key attribute values in referencing tuples to **null** or to some valid value referencing a different tuple

Update:

- Key constraint violation: primary key is changed so as to become same as another tuple's
- referential integrity violation:
  - o foreign key is changed and new one refers to nonexistent tuple
  - o primary key is changed and now other tuples that had referred to this one violate the constraint

7) What is valid state and an invalid state, with respect to a database

4 Marks (Dec/Jan 2014, Jun / July2013)

A database state that does not obey all integrity constraints is called an invalid state and a state that satisfies all the constraints in IC is called valid state.

8) List the characteristics of relation? Discuss any one? 4 Marks (Jun / July2014)

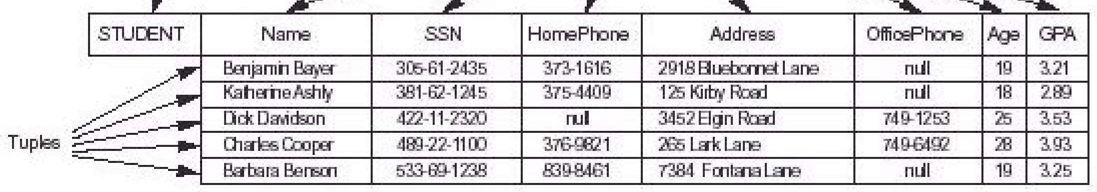
Characteristics of Relation

- A *tuple* can be considered as a set of (<attribute>, <value>) pairs. Thus the following two tuples are *identical*:
- t1 = <(Name, B. Bayer),(SSN, 305-61-2435),(HPhone, 3731616),

  (Address, 291 Blue Lane),(WPhone, null),(Age, 23),(GPA, 3.25)>
- t2 = <(HPhone, 3731616),(WPhone, null),(Name, B. Bayer),(Age, 23),

  (Address, 291 Blue Lane),(SSN, 305-61-2435),(GPA, 3.25)>
- *Tuple* ordering is not a part of relation, that is the following relation is *identical* to that of

  Table taken below.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

The attributes and tuples of a relation STUDENT

9) Discuss Various types of Inner Join Operations?
8 Marks (Jun / July2014) (5 marks Dec 14/Jan 15)

inner join

The cartesian product example above combined each employee with each department. If we only keep those lines where the **dept** attribute for the employee is equal to the **dnr** (the department number) of the department, we get a nice list of the employees, and the department.

- A very common and useful operation.
- Equivalent to a cartesian product followed by a select.
- Inside a relational DBMS, it is usually much more efficient to calculate a join directly, instead of calculating a cartesian product and then throwing away most of the lines.
- Note that the same SQL query can be translated to several different relational algebra expressions, which all give the same result.
- If we assume that these relational algebra expressions are executed, inside a relational DBMS which uses relational algebra operations as its lower-level internal operations, different relational algebra expressions can take very different time (and memory) to execute.

Consider the following schema

Sailor(Sal-id,Sal-name,rating,age) Reserves(Sal-id,Boat-id,day) Boats(Boat_id,boat-name,color)

I. Find the names of sailors ,who have reserved all boats,called Interlake?

Select Sal-name from Sailor s, Reserves r,Boats b where

r.sal-id==s.sal-id and r.boat-id==b.boat-id and boat-name=”Interlake”;

II. Find the Sid of the sailor with age over 20 who haven’t reserved the boat.

Select sid from sailor s where NOT EXISTS(select S.Sal-id FROM Sailor S ,Reserves r ,Boats b where r.bid=b.boat-id) and age>20;

III Find the names of sailors,who have reserved atleast Two Boats

Select S.sal-name from sailors S,Reserves r,boats b where r.salid=s.sal-id and b.boat-id=r.boat-id having count(*)>=2

10) Explain Foreign key and its importance. Can foreign key exist, only for single table explain? 8 Marks (Jun / July 2013)

A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

```
CREATE TABLE ORDERS
(Order_ID integer,
Order_Date date,
Customer_SID integer,
Amount double,
Primary Key (Order_ID),
Foreign Key (Customer_SID) references CUSTOMER (SID));
```

Yes foreign key can exists for a single Table.

11) How an Intersection Operator can be implemented using Union and Minus operator?

8 Marks (Jun/July 2013)

```
select Rollno From (
Select 1 AS dummy,Rollno From StageShow
Union ALL
Select 2 AS dummy,Rollno From Sports
) X group By Rollno having count(*)=2
```

12) Write queries in Relational Algebra? 8 Marks (Dec /Jan 2013)

i) **Retrieve the number of dependents for an employee "RAM"?**

Name, SSN, Dname(SSN=SSN2(Employee x   SSN2, Dname(Dependents))

ii)**Find the name of manager of each department?**

Department $\bowtie_{MGRSSN=SSN}$ Employee

iii)**Retrieve the names of employee who works in same department as that of "RAJ"?**

Π <name> (σ <dno=5>(<Employee>))

13. Briefly discuss how the different update operations on a relation deal with constraint violations? 8 marks (June/July 2015)

There are three types of constraints on relational database:

• Domain Constraints

• Primary Key Constraints

• Integrity Constraints

Domain Constraints:

It specifies that each attribute in a relation must contain an atomic value only from the corresponding domains. Domain constraint specifies the condition that we want to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column.

Key Constraints:

This constraints states that the key attribute value in each tuple must be unique, i.e., no two tuples contain the same value for the key attribute. This is because the value of the primary key is used to identify the tuples in the relation.

Integrity Constraints:

There are two types of integrity constraints:

• Entity Integrity Constraint

• Referential Integrity Constraint

Entity Integrity Constraint:

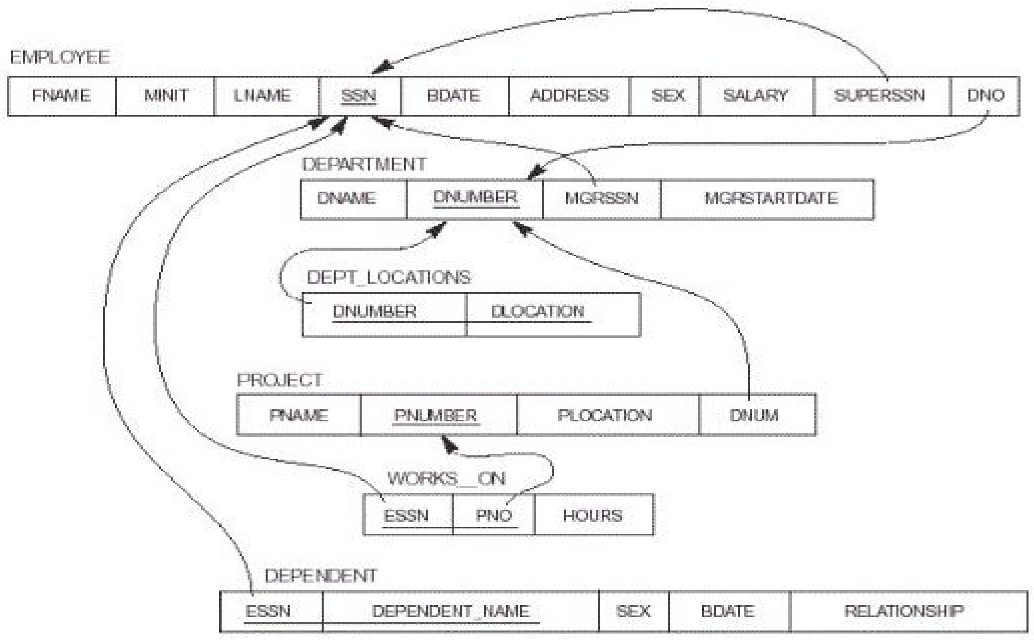
It states that no primary key value can be null. This is because the primary key is used to identify individual tuple in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes.

Referential Integrity Constraint:

It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraint is specified on two relations, not necessarily distinct.

14 Give the schema **10 marks (Dec 14/Jan 15)**

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



UNIT-IV

SQL – 1

1) Given the schema

14 Marks (Jun/July 2013 & June/July 2015)

EMP (Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

DEP T(Dname, Dnumber, MgrSSN, MGrstartdate)

DEPT-LOC (Dnumber, Dloc) PROJECT(Pname, Pnumber, Ploc,Dnum)

WORKS-ON (ESSN,PNo,Hours)

Give the relation algebra expression for the following:

1. List female employees from Dno=20 earning more than 50000

Temp $\longleftarrow \sigma_{\text{Dno=20 and salary<50000}}$ (EMP)

Temp2 $\longleftarrow \sigma_{\text{sex='F'}}$(Temp)

Result $\longleftarrow$ $\Pi_{\text{Lname, Fname}}$ (Temp2)

2. List 'CSE' department details.

Result $\longleftarrow \sigma_{Dname='CSE'}$(Department)

3. Retrieve the first name, last name and salary of all employees who work in departmental number 50

Dep-Emps $\longleftarrow \sigma_{Dno='50'}$ (Emp)

4. Retrieve the name of the manager of each department.

MGRS $\longleftarrow \Pi_{MgrSSn}$ (Department)

Result $\longleftarrow \Pi$ Lname,Fname (MGRS ∞ EMP)

5. Retrieve the name and address of all employees who work for sports department.

Sport-dept $\longleftarrow \sigma_{Dname='sports'}$(Dept)

Sport-Emps $\longleftarrow$(Sport-dept $\infty_{Dnumber=Dno}$ EMP)

2) With respect to SQL, explain with example. 4 Marks (Dec/Jan 2013)

i) **Drop table:** It is used to remove a relation (Base table) and its definition. The relation can no longer be used for querying, updates or any other commands since its description no longer exists

Ex: Drop table dependent;

ii) **Alter table**: It is used to add an attribute to one of the base relations. The new attribute will have NULL's in all tuples of the relation right after the command is executed, hence the NOT NULL constraint is not allowed for such an attribute.

Ex: Alter table Employee add JOB varchar2 (12);

3) Explain IN and EXISTS operations with an example.

6 Marks (Jun/July 2013, Jun/July 2013)

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

Ex: Query : Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q: SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       EXISTS  (SELECT    *
                    FROM          DEPENDENT  WHERE        SSN=ESSN
                            AND FNAME=DEPENDENT_NAME)
```

The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V

Ex: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q: SELECT     E.FNAME, E.LNAME
       FROM          EMPLOYEEASE
       WHERE         E.SSN IN
                            (SELECT      ESSN
                            FROM         DEPENDENT
                            WHERE        ESSN=E.SSN AND
                            E.FNAME=DEPENDENT_NAME)
```

4) How does SQL inplement the entity integrity constraints of relational Data Model? Explain with an example?

4 Marks (Dec/Jan 2014, Jun / July2013)(8 marks Dec 14/Jan 15)

In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table

More on Views

We can make the following observations:

- A view with a single defining table is updatable if we view contain PK or CK of the base table
- View on multiple tables using joins are not updatable
- View defined using grouping/aggregate are not updatable
- Specifying General Constraints
- Users can specify certain constraints such as semantics constraints

CREATE ASSERTION SALARY_CONSTRAINT

CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D WHERE E.SALARY > M. SALARY **AND** E.DNO=D.NUMBER **AND** D.MGRSSN=M.SSN))

5) Consider the following two tables T1 and T2 show the result of the following

operations

T1$\infty_{T1.p=T2.A}$ T2

T1$\infty_{T1.Q=T2.B}$ T2

T1$\infty_{T1.p=T2.A}$ T2

T1UT2

(ASSUME T1 AND T2 ARE UNION COMPATIBILITY)

8 Marks (Dec /Jan 2013, Jun / July2014)

Table T2

Table T1

P	Q	R
10	a	5
15	b	8
25	a	6

A	B	C
10	b	6
25	c	3
10	b	5

6. Explain with an example , the basic constraints that can be specified, when you create a table in SQL. 4 Marks (Jun / July2014)

Tuples in the *referencing relation* R_1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if

$t_1[FK] = t_2[PK]$

A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.

- *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 keys implies that we cannot identify some tuples. Key constraints and entity constraints are specified on individual relations.

- *Referential integrity constraint* - The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations. A tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

- *Foreign key* is a set of attributes of one relation R_1 whose values are required to match values of some candidate key of some relation R_2 .

For example, the value of attribute DNO in every EMPLOYEE tuple must match the DNUMBER value of some tuple in the DEPARTMENT relation.

7) Explain how groupBy clause works?What is the difference between WHERE and Having? In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)

8 Marks (Dec/Jan 2014 & June/July 2015)

The function is applied to each subgroup independently

SQL has a GROUP BY clause for specifying the grouping attributes, which must also appear in the SELECT clause

```
SELECT PNUMBER, PNAME, COUNT(*)
  FROM PROJECT, WORKS_ON
  WHERE PNUMBER=PNO
  GROUP BY PNUMBER, PNAME
```

In the EMPLOYEE tuples are divided into groupseach group having the same value for the Grouping attribute DNO The COUNT and AVG functions are applied to each such group of tuples separately.The SELECTclause includes only the grouping attribute and the functions to be applied on each group of tuples. A join condition can be used in conjunction with grouping.

8) How does SQL implement the entity integrity constraints of relational Data Model? Explain with an example? (8 Marks (Jun / July 2013)

In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table

More on Views

We can make the following observations:

- A view with a single defining table is updatable if we view contain PK or CK of the base table
- View on multiple tables using joins are not updatable
- View defined using grouping/aggregate are not updatable
- Specifying General Constraints
- Users can specify certain constraints such as semantics constraints

CREATE ASSERTION SALARY_CONSTRAINT

CHECK (N O T EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M,

DEPARTMENT D

WHERE E.SALARY > M. SALARY **AND** E.DNO=D.NUMBER **AND** D.MGRSSN=M.SSN))

9) Explain all possible options that are specified when referential integrity constraint is violated using suitable example for all options? 8 Marks (Jun/July 2013, Jun / July2014)

An addition to the original standard allows specification of primary and candidate keys and foreign keys as part of the **create table** command:

- **primary key** clause includes a list of attributes forming the primary key.
- **unique key** clause includes a list of attributes forming a candidate key.
- **foreign key** clause includes a list of attributes forming the foreign key, and the name of the relation referenced by the foreign key.
- When a referential integrity constraint is violated, the normal procedure is to reject the action. But a foreign key clause in SQL-92 can specify steps to be taken to change the tuples in the referenced relation to restore the constraint.

Example.

create table *account*

foreign key (*bname*) **references** *branch*

on delete cascade

on insert cascade,

If a delete of a tuple in *branch* results in the preceding referential integrity constraints being violated, the delete is not rejected, but the delete ``cascade" to the *account* relation, deleting the tuple that refers to the branch that was deleted. Update will be cascaded to the new value of the branch!

- SQL-92 also allows the **foreign key** clause to specify actions other than cascade, such as setting the refencing field to null, or to a default value, if the constraint is violated.
- If there is a chain of foreign key dependencies across multiple relations, a deletion or

update at one end of the chain can propagate across the entire chain.If a cascading update or delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction and all the changes caused by the transaction and its cascading actions are undone. Given and complexity and arbitrary nature of the way constraints in SQL behave with null values, it is the best to ensure that all columns of **unique** and **foreign** key specifications are declared to be non null.

10) Write queries in SQL for the following

8 Marks (Dec /Jan 2013, Dec/Jan 2014)

Select all EMPLOYEE SSNs , and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME ?

```
SELECT   SSN, DNAME
FROM       EMPLOYEE, DEPARTMENT
```

Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT   FNAME, LNAME, 1.1*SALARY
FROM     EMPLOYEE, WORKS_ON, PROJECT
WHERE    SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Make a list of Project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manger of the department that controls the project

```
SELECT DISTINCT PNUMBER   FROM PROJECT  WHERE PNUMBER IN
(SELECT PNUMBER  FROM PROJECT,  DEPARTMENT, EMPLOYEE   WHERE
DNUM=DNUMBER AND  MGRSSN=SSN AND LNAME=‘Smith’
```

10) List and explain the basic data types available for attributes in SQL and give example.?

5 marks (June/July 2015)

Data type	**Description**
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 expo notation. The size argument for this type consists of a single number specifying t precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16

DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

1 1) Given the schema

12 Marks (Dec 14/Jan 15)

EMP (Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

DEP T(Dname, Dnumber, MgrSSN, MGrstartdate)

DEPT-LOC (Dnumber, Dloc) PROJECT(Pname, Pnumber, Ploc,Dnum)

WORKS-ON (ESSN,PNo,Hours)

Give the relation algebra expression for the following:

1. List female employees from Dno=20 earning more than 50000

Temp $\longleftarrow \sigma_{\text{Dno=20 and salary<50000}}$ (EMP)

Temp2 $\longleftarrow \sigma_{\text{sex='F'}}$(Temp)

Result $\longleftarrow$ $\Pi_{\text{Lname, Fname}}$ (Temp2)

2. List 'CSE' department details.

Result $\longleftarrow \sigma_{\text{Dname='CSE'}}$(Department)

3. Retrieve the first name, last name and salary of all employees who work

in departmental number 50

Dep-Emps $\longleftarrow \sigma_{Dno='50'}$ (Emp)

4. Retrieve the name of the manager of each department.

MGRS $\longleftarrow \Pi_{MgrSSn}$ (Department)

Result $\longleftarrow \Pi$ Lname,Fname (MGRS ∞ EMP)

5. Retrieve the name and address of all employees who work for sports department.

Sport-dept $\longleftarrow \sigma_{Dname='sports'}$(Dept)

Sport-Emps $\longleftarrow$(Sport-dept $\infty_{Dnumber=Dno}$ EMP)

UNIT-V
SQL – 2

1. Explain insert, delete and update statements in SQL with example.

8 **Marks** **(Dec/Jan 2013 & June/July 2015)**

The Insert Operation

This operation can violate all constraints in a relational database.Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain. Key constraints can be violated if a key value in the new tuple already exits in another tuple in the relation R. Entity constraints can be violated if 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. Examples of this operation:

- Insert <\`Ahmad',\`Jabbar', null, \`60/04/05',F,28000,null,4> into EMPLOYEE.

  This insertion violates the entity integrity constraints (SSN=null).

  Insert <\`Alicia',\`Zelaya',\`999887777',\`60/04/05',F ,28000,\`987654321',4> into EMPLOYEE.

This insertion violates the key constraint (SSN=999887777 is already exists).

- Insert <\`Ahmad',\`Jabbar',\`677678989',\`60/04/05',M, 28000,\`987654321',7> into EMPLOYEE.

This insertion violates referential integrity (DNO=7, there is no DEPNUMBER=7 in DEPARTMENT relation).

- Insert <`Ahmad',`Jabbar',`987987987',`69/03/29',M,2800,null,4> into EMPLOYEE.

This insertion is acceptable.

The Delete Operation

This operation can violate only referential integrity. Examples of delete operation:

- Delete the WORKS_ON tuple with SSN=`123456789' and PNO=1.

  This deletion is acceptable.

- Delete the EMPLOYEE tuple with SSN=`999887777'.

This deletion violates the referential integrity (tuples in WORKS_ON refer to this tuple).

- Delete the EMPLOYEE tuple with SSN=`333445555'.

  This deletion violates referential integrity (tuples in EMPLOYEE, DEPARTMENT,

WORKS_ON, and DEPENDENT refer to this tuple).

The Update Operation

The new value must be of the correct data type and domain. Examples of update operation:

- Update the SALARY of the EMPLOYEE tuple with SSN=`999887777' to 28000.

This update is acceptable.

- Update the DNO of the EMPLOYEE tuple with SSN=`999887777' to 7.

This update violates referential integrity (there is no DNUMBER=7 in DEPARTMENT relation).

- Update the SSN of the EMPLOYEE tuple with SSN=`999887777' to `987654321'.

2. Write a note on aggregate functions in SQL and Views in SQL with examples.

4 Marks (Jun / July2014 & June/July 2015)

AGGREGATE FUNCTIONS ($\Im$)

Functions such as SUM, COUNT, AVERAGE, MIN, MAX are often applied to sets of values or sets of tuples in database applications

<grouping attributes> $\Im_{\text{<function list>}}$(R)

The grouping attributes are optional

Example 1: Retrieve the average salary of all employees (no grouping needed):

$\rho(\text{AVGSAL}) \leftarrow \mathcal{F}_{\text{AVERAGE SALARY}}(\text{EMPLOYEE})$

Example 2: For each department, retrieve the department number, the number of employees, and

the average salary (in the department):

$\rho_{(\text{DNO,NUMEMPS,AVGSAL})} \leftarrow {}_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE SALARY}}(\text{EMPLOYEE})$

DNO is called the *grouping attribute* in the above example

SQL Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

SQL CREATE VIEW Examples

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued=No
```

3) How is a view created and dropped? What problems are associated with updating of views? (Dec 11/Jan 12)(6 marks Dec 14/Jan 15)

A view refers to a single table that is derived from other tables

```
CREATE VIEW WORKS_ON1
AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON WHERE SSN=ESSN AND
PNO=PNUMBER
```

A view can be dropped as shown below

```
DROP VIEW WORKS_ON1
```

Updating of Views

Updating the views can be complicated and ambiguous In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table A view with a single defining table is updatable if we view contain PK or CK of the base table View on multiple tables using joins are not updatable View defined using grouping/aggregate are not updatable

4) What is embedded SQL? With an example explain how would you Connect to a database, fetch records and display. Also explain the concept of stored procedure in brief. (Dec 11/Jan 13/ Jan 2014) (10 marks Dec 14/ Jan 15)

The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements. In PL/I the keywords EXEC SQL precede any SQL statement. In some implementations, SQL statements are passed as parameters in procedure calls. We will use PASCAL as the host programming language, and a "$" sign to identify SQL statements in the program. Within an embedded SQL command, we may refer to program variables, which are prefixed by a "%" sign. The programmer should declare program variables to match the data types of the database attributes that the program will process. These program variables may or may not have names that are identical to their corresponding attributes

Example: Write a program segment (loop) that reads a social security number and prints out some information from the corresponding EMPLOYEE tuple

```
E1:     LOOP:= 'Y';
         while LOOP = 'Y' do
                 begin
                 writeln('input social security number:');
                 readln(SOC_SEC_NUM);
                 $SELECT FNAME, MINIT, LNAME, SSN, BDATE,
ADDRES
                   INTO %E.FNAME, %E.MINIT, %E.LNAME, %E.SSN,
                          %E.BDATE, %E.ADDRESS, %E.SALARY
                   FROM EMPLOYEE
                   WHERE SSN=%SOC_SEC_NUM ;
                 writeln( E.FNAME, E.MINIT, E.LNAME, E.SSN,
E.BDAT
```

```
writeln('more social security numbers (Y or N)? ');
readln(LOOP)
end;
```

Stored procedures are the database program modules ,procedures or functions that are stored and executed by the DBMS at the database server.

<u>Advantages of stored procedures</u>

1. It can be invoked by many application programs simultaneously.
2. It can reduce data transfer and communication cost b/w client and server .
3. it can enhance modeling power provided by views by allowing more complex types of derived data.

5) Explain the syntax of a SELECT statement in SQL.write the SQL query for the following relation algebra expression. (MAY/JULY 2013/ 2014) (4 Marks Dec 14/Jan 15)

*bdate,address(σFname='John' and Lname='smith'(Employee))

The SELECT Statement

The syntax of this command is:

SELECT <attribute list>

FROM <table list>

WHERE <Condition>;

```
:           SELECT BDATE, ADDRESS
                  FROM    EMPLOYEE
        WHERE FNAME = 'John' AND  MINIT ='B' AND   LNAME = 'SMITH'
```

6) Explain the drop command with an example (MAY/JULY 2013)

Used to remove a relation (base table) and its definition

The relation can no longer be used in queries, updates, or any other commands since its description no longer exists

Example:

```
DROP TABLE  DEPENDENT;
```

7. Consider the following tables: (May/July 2013,DEC/ Jan2014)

WORKS(PNAME,CNAME,SALARY)

LIVE(PNAME,STREET,CITY)

LOCATEDI-IN(CNAME,CITY)

MNAGER(PNAME,MGRNAME)

Write the sql query for the following:

i) Find the names of all persons who live in city 'mumbai'.

ii) Retrieve the names of all persons of Infosis' whose salary is between rs.30,000 and rs 50,000.

iii) Find the names of all persons who live and work in the same city.

iv) list the names of the people who work for 'wipro' along with the cities they live in.

v) find the average salary of all 'Infosians'

Select Pname from MNAGER,LIVE where CITY='MUMBAI';

Select pname from MNAGER M ,WORKS W WHERE W.pname =M.pname and SALARY IN

Select salary from Works where (salary >30000 and salary <50000)

Select pname from MNAGER M,LIVE L,WORK W where

M.pname=L.Pname and City in(Select pname from LOCATED where City='Bombay';

Select Pname,city from Works W,Located-in L1 where W.pname=L1.pname and CNAME='wipro' .

Select avg(Salary) from Works W where CNAME='infosys';

8) List the approaches to DB Programming. Main issues involved in DB Programming?

(Dec /Jan 11/ Jan 2014)

Embedded commands:

- Database commands are embedded in a general-purpose programming language
- Library of database functions:
- Available to the host language for database calls; known as an *API*
- *API* standards for Application Program Interface
- A brand new, full-fledged language
- Minimizes impedance mismatch

9) What is Impedence Mismatch problem? Which of the three programming approaches minimizes this problem? (Dec 12/Jan 13)

- Incompatibilities between a host programming language and the database model, e.g.,type mismatch and incompatibilities; requires a new binding for each language

- set vs. record-at-a-time processing
- need special integrators to loop over query results and manipulate individual values

Steps:

- Client program *opens a connection* to the database server
- Client program *submits queries to and/or updates* the database
- When database access is no longer needed, client program *closes (terminates) the* connection

Embedded SQL

- Most SQL statements can be embedded in a general-purpose host programming language such as COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between **EXEC SQL** or **EXEC SQL BEGIN** and a matching **END-EXEC** or

  EXEC SQL END (or semicolon)
- Syntax may vary with language
- Shared variables (used in both languages) usually prefixed with a colon (:) in SQL

10) How are Triggers and assertions defined in SQL?Explain (Dec 11/Jan 13)

Constraints as Assertions

General constraints: constraints that do not fit in the basic SQL categories (presented in Mechanism: **CREAT ASSERTION** Components include:

- a constraint name,
- followed by CHECK,
- followed by a condition

The salary of an employee must not be greater than the salary of the manager of the department that the employee works for''

```
CREAT ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
                    FROM EMPLOYEE E, EMPLOYEE M,
     DEPARTMENT D
                    WHERE E.SALARY > M.SALARY AND
          E.DNO=D.NUMBER AND
     D.MGRSSN=M.SSN))
```

SQL Triggers:

Triggers are expressed in a syntax similar to assertions and include the following:

Event

Such as an insert, deleted, or update operation

Condition

Action

To be taken when the condition is satisfied

A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
        SALARY, SUPERVISOR_SSN ON EMPLOYEE
        FOR EACH ROW
                WHEN
                (NEW.SALARY> (SELECT SALARY FROM EMPLOYEE
                 WHERE SSN=NEW.SUPERVISOR_SSN))
                INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN);
```

11) Discuss the significance of assertion? Write an assertion to specify constraint such that the salary of an employee must not be greater than the salary of the manager of the department that employees works for? (DEC/JAN 2013/ JAN- 2014)

```
CREAT ASSERTION SALARY_CONSTRAINTCHECK (NOT EXISTS (SELECT *FROM
EMPLOYEE E, EMPLOYEE M, DEPARTMENT DWHERE E.SALARY > M.SALARY
ANDE.DNO=D.NUMBER  AND D.MGRSSN=M.SSN))
```

An assertion is a piece of SQL which makes sure a condition is satisfied or it stops action being taken on a **database object**. It could mean locking out the whole table or even the whole database.

To make matters more confusing - a trigger could be used to enforce a check constraint and in some DBs can take the place of an assertion (by allowing you to run code un-related to the table being modified). A common mistake for beginners is to use a check constraint when a trigger is required or a trigger when a check constraint is required.

12. What is Impendence Mismatch? Explain (DEC/JAN 2013)

The concept of query languages developed in 1970-ties assumed no pragmatic universality. However, because eventually such universality is inevitable in real applications, there was an assumption that a query language is a "sublanguage" that is to be embedded in a universal programming language. A "sub-language" determines the access to a database only. The rest of the entire application has to be programmed in a typical programming language. This assumption requires joining a query language with a programming language in such a way that:

Queries can be used inside programs;

Queries can be parameterized through values calculated within programs;

Results of queries are to be passed to programs.

Difference between language concepts cause significant technical difficulties in accomplishing this kind of connection. A lot of programmers and computer professionals were also disappointed by the technical, aesthetic and conceptual degradation of the programming environment. This degradation is commonly referred to as *impedance mismatch*. This term denotes a bunch of disadvantageous features that are implied by the eclectic mix of a query language (in particular SQL) with a programming language (such as C, C++ or Java). Below welist and comment these features.

Syntax: In the same code the programmer must use two programming styles and must follow two different grammars. Similar concepts are denoted differently (for instance, strings in C are written within "…", in SQL – '…') and different concepts are denoted similarly (for instance, in C = denotes an assignment, in SQL – a comparison).

Typing: Types and denotations of types assumed by query and programming languages differ, as a rule. This concerns atomic types such as integer, real, boolean, etc. Representation of atomic types in programming languages and in databases can be significantly different, even if the types are denoted by the same keyword, e.g. *integer*. Alossless conversion between such types could be impossible and might imply some

performance overhead. This also concerns complex types, such as tables (a basic data type constructor in SQL, absent in programming languages). Popular programming languages introduce static (compile time) type checking, which is impossible e.g. in SQL (because query languages are based on dynamic rather than static binding).

Semantics and language paradigms: The concept of semantics of both languages is totally different. Query languages are based on the declarative style (*what* is to be retrieved rather than *how*), while programming languages are based on the imperative style (a sequence of commands to a machine, which accomplishes *what*).

Abstraction level: Query languages free the programmers from majority of the details concerning data organization and implementation, for instance, organization of collections, presence or absence indices, etc. In programming languages these details usually are explicit(although may be covered by some libraries).

Binding phases and mechanisms: Query languages are based on late (run-time) binding of all the names that occur in queries, while programming languages are based on early (compile and linking time) binding. Thus, from the point of view of a program, queries are simply strings of characters.

Name spaces and scope rules: Queries do not see names occurring in programs and v/v. Because eventually there must be some intersection of these name spaces (e.g. program variables must parameterize queries) additional facilities, with own syntax, semantics and pragmatics, are required. These facilities are the burden for the size and legibility of the program code. Moreover, in programming languages name scopes are organized hierarchically and processed by special rules based on stacks. These rules are ignored by a query language. This leads e.g. to problems with recursive procedure calls (a well-known example concerns SQL cursors that severely reduce the possibility of recursion). Another disadvantage of separated name spaces concerns automatic refactoring of programs, which cannot be performed on queries.

Collections: Databases store collections (e.g. tables) which are processed by queries. In programming languages collections are absent or severely limited. Hence collections returned by queries have no direct counterparts in a programming language and must be processed by special constructs with own syntax and semantics.

Null values: Databases and their query languages have specialized features for storing and processing null values. Such features are absent in programming languages, thus some substitutes must be introduced. For instance, if some value in a relational database can be null, mapping it to a programming language requires two variables: one for storing information about null and another one for storing the value.

13) Create View which will display the dname,no of employees working and total salary of each department? (DEC/JAN 2013/July 2013)

CREATE VIEW WORKS_ON1

AS SELECT FNAME, LNAME, PNAME, HOURS ,SALARY

FROM EMPLOYEE, PROJECT, WORKS_ON WHERE SSN=ESSN AND PNO=PNUMBER .

a) Embedded SQL

```
loop = 1;
while (loop) {
        prompt ("Enter SSN: ", ssn);
        EXEC SQL
                select FNAME, LNAME, ADDRESS, SALARY
                into :fname, :lname, :address, :salary
                from EMPLOYEE where SSN == :ssn;
                if (SQLCODE == 0) printf(fname, …);
                else printf("SSN does not exist: ", ssn);
                prompt("More SSN? (1=yes, 0=no): ", loop);
        END-EXEC
}
```

A **cursor** (iterator) is needed to process multiple tuples

FETCH commands move the cursor to the *next* tuple

CLOSE CURSOR indicates that the processing of query results has been completed

All these things are embedded in a language.

UNIT – 6

Database Design – 1

1) What is the need for normalization? Explain the first,second and third normal forms with examples. (JUN/JULY 2013)

The purpose of normalization.

- The problems associated with redundant data.
- The identification of various types of update anomalies such as insertion, deletion,

and modification anomalies.

- How to recognize the appropriateness or quality of the design of relations.
- The concept of functional dependency, the main tool for measuring the appropriateness of attribute groupings in relations.
- How functional dependencies can be used to group attributes into relations that are in a known normal form.
- How to define normal forms for relations.
- How to undertake the process of normalization.
- How to identify the most commonly used normal forms, namely first (1NF), second (2NF), and third (3NF) normal forms, and Boyce-Codd normal form (BCNF).
- How to identify fourth (4NF), and fifth (5NF) normal forms.

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.

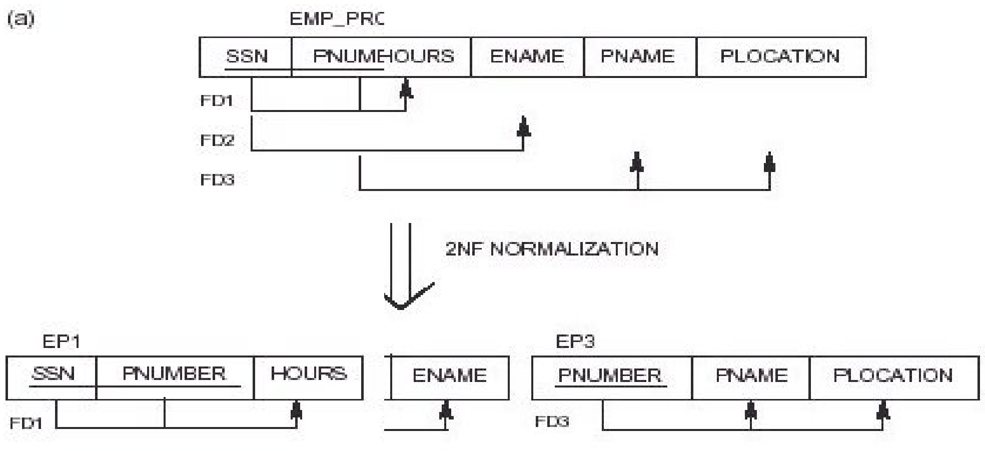
Second Normal Form (2NF)

Second normal form is based on the concept of fully functional dependency. A functional X $\rightarrow$

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.



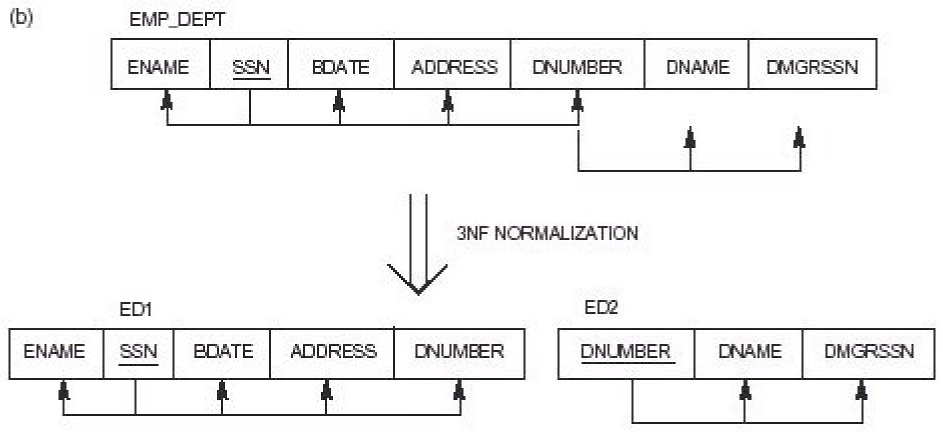
Third Normal Form (3NF)

Third normal form is based on the concept of transitive dependency. A functional dependency X $\rightarrow$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 $\rightarrow$ Z and Z$\rightarrow$ Y hold. In other words, a relation is in 3NF if, whenever a functional dependency

X $\rightarrow$ 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.



1NF: R is in 1NF iff all domain values are atomic.

2NF: R is in 2 NF iff R is in 1NF and every nonkey attribute is fully dependent on the key.

3NF: R is in 3NF iff R is 2NF and every nonkey attribute is non-transitively dependent on the key

2) Explain informal design guidelines for relation schemas. **(Dec /Jan 12)**

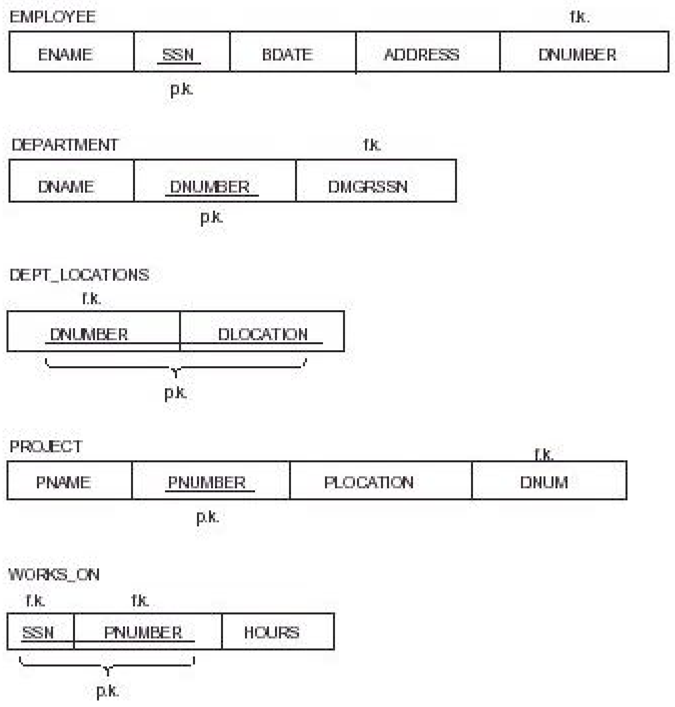
The four informal measures of quality for relation schema

- Semantics of the attributes
- Reducing the redundant values in tuples
- Reducing the null values in tuples
- Disallowing the possibility of generating spurious tuples

Semantics of relations attributes

Specifies how to interpret the attributes values stored in a tuple of the relation. In other words, how the attribute value in a tuple relate to one another.

Figure 14.1 Simplified version of the COMPANY relational database schema.



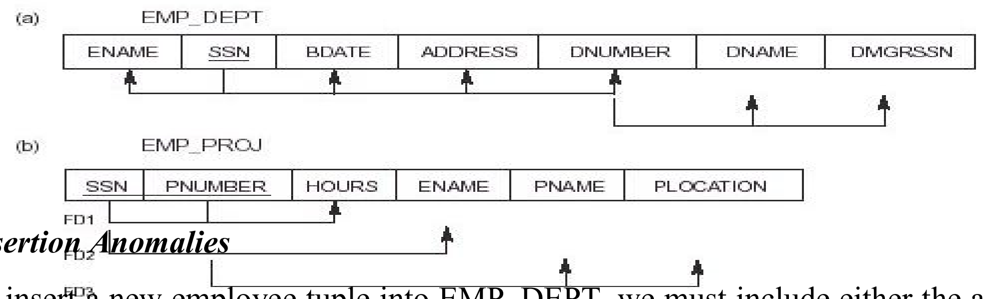
Guideline 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Reducing redundant

values in tuples. Save storage space and avoid update anomalies.

3. Discuss insertion, deletion, and modification anomalies. Why they are bad? Illustrate with example? (Jan 2014)

- Insertion anomalies.
- Deletion anomalies.
- Modification anomalies.

Figure 14.3 Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.



Insertion Anomalies

To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for that department that the employee works for, or nulls. It's difficult to insert a new department that has no employee as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity - not a department entity.

Deletion Anomalies

If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

Modification Anomalies

In EMP_DEPT, if we change the value of one of the attributes of a particular department- say the manager of department 5- we must update the tuples of all employees who work in that department.

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies occur. Reducing the null values in tuples. e.g., if 10% of employees have offices, it is better to have a separate relation, EMP_OFFICE, rather than an attribute

OFFICE_NUMBER in EMPLOYEE.

Guideline 3: Avoid placing attributes in a base relation whose values are mostly null. Disallowing spurious tuples.

Spurious tuples - tuples that are not in the original relation but generated by natural join of decomposed subrelations.

Example: decompose EMP_PROJ into EMP_LOCS and EMP_PROJ1.

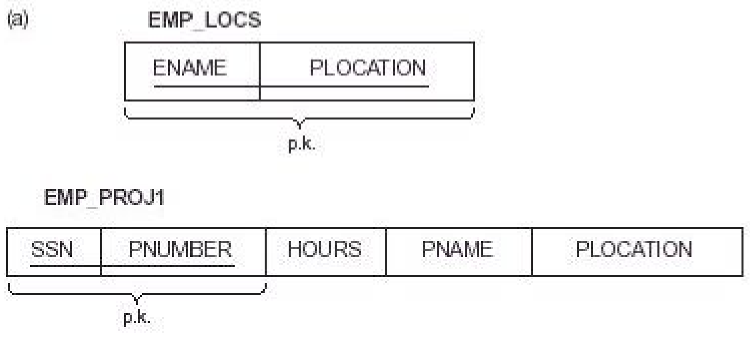
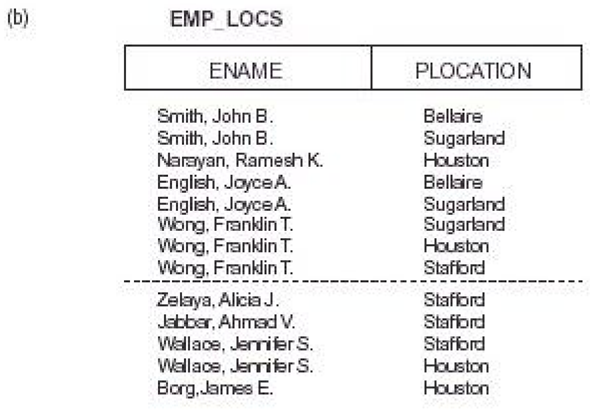


Fig. 14.5a


(b) EMP_LOCS

ENAME	PLOCATION
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

Guideline 4: Design relation schemas so that they can be naturally JOINed on primary keys or foreign keys in a way that guarantees no spurious tuples are generated.

4) Which normal form is based on the concept of transitive dependency? Explain with an example the decomposition into 3NF <u>**(July2013 /Jan 2013)**</u>

Third normal form is based on the concept of transitive dependency. A functional dependency X

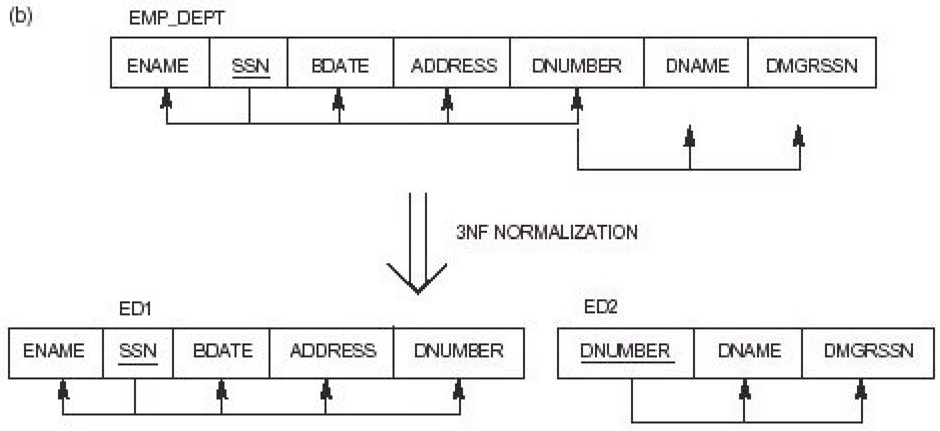
→Y in a relation is a transitive dependency if there is a set of attributes Z that is not a subset of

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 $\rightarrow$ 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.



5) **Explain multivalued dependency. Explain 4NF with an exampl**e **(July2013/Jan 2013)**

A **multivalued dependency** (MVD) X □□□□Y specified on R, where X, and Y are both subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1 [Z]$

Fourth Normal Form (4NF)

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s).

A Relation is in 4NF if it is in 3NF and there is no multivalued dependencies.

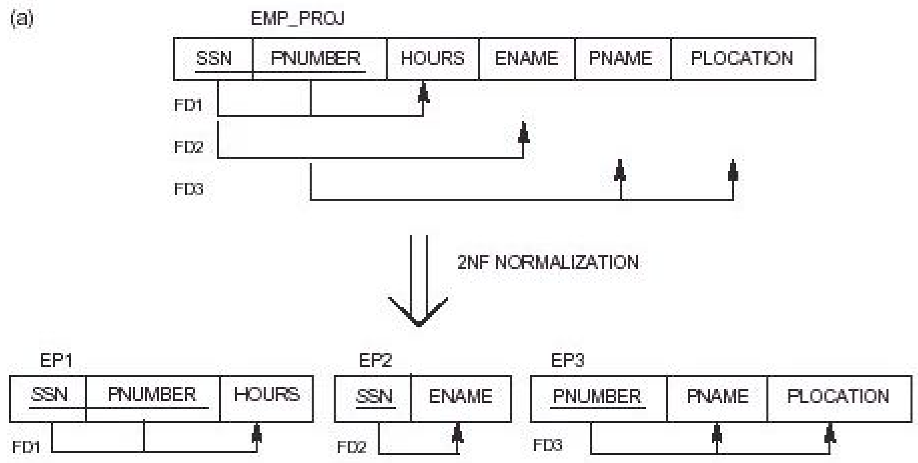
6) what is the need for normalization ?explain second normal form (Jan 2013)

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.



7. Explain any Two informal quality measures employed for a relation schema Design?

(Jan -2013)

Informal Design Guidelines are:-

Semantics of the Relation Attributes

Redundant Information in Tuples and Update Anomalies

Null Values in Tuples

Spurious Tuples

Informally, each tuple in a relation should represent one entity or relationship instance. (Applie to individual relations and their attributes).

Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation

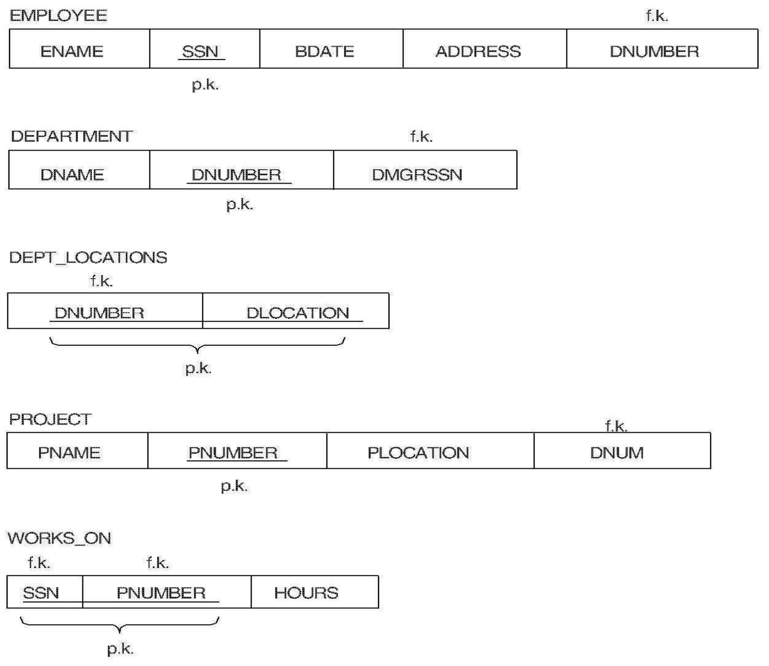
Only foreign keys should be used to refer to other entities

Entity and relationship attributes should be kept apart as much as possible.

Bottom Line: Design a schema that can be explained easily relation by relation. The semantics

of attributes should be easy to interpret.

Figure 14.1 Simplified version of the COMPANY relational database schema.



© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Mixing attributes of multiple entities may cause problems

Information is stored redundantly wasting storage

Problems with update anomalies

– Insertion anomalies

– Deletion anomalies

– Modification anomalies

Consider the relation:

EMP_PROJ (<u>Emp#, Proj#,</u> Ename, Pname, No_hours)

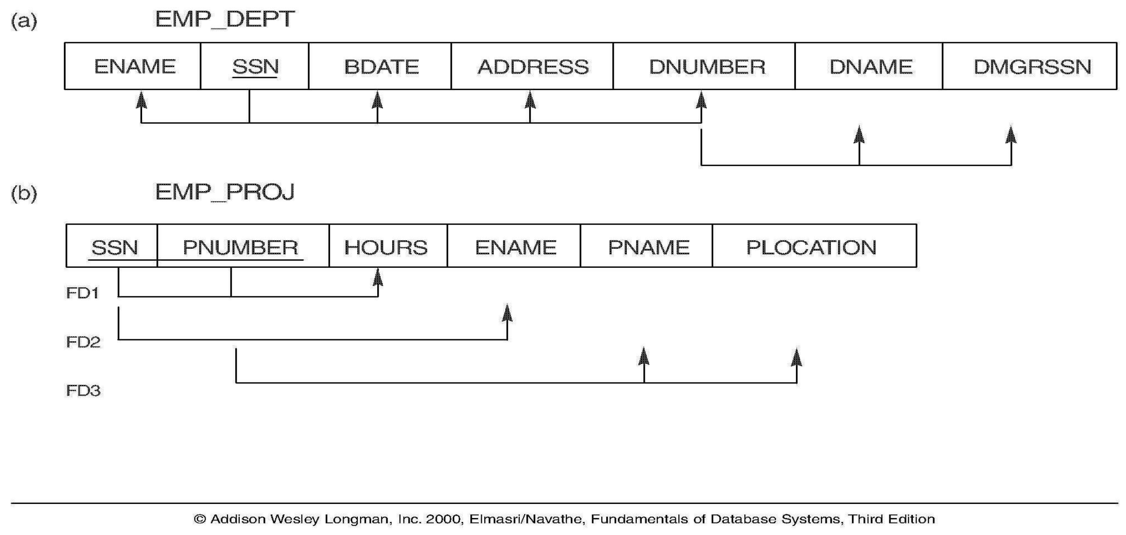
Update Anomaly: Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.

Insert Anomaly: Cannot insert a project unless an employee is assigned to .

Inversely - Cannot insert an employee unless an he/she is assigned to a project.

Delete Anomaly: When a project is deleted, it will result in deletingall the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Figure 14.3 Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.



8).Consider the following relations: Car_sale (car_no, date-sold, salemanno, commission%,discount). Assume a car can be sold by multiple salesman and hence primary key is {car-no, salesman} additional dependencies are: Date-sold□discount and salesmanno. Commission? (JAN -2014)

I) Yes this relation is in 1NF

C) Discuss the minimal sets of FD'S?

Every set of FDs has an equivalent minimal set

There can be several equivalent minimal sets

There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs

To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal sets.

9) Suggest and explain three different techniques to achieve 1nf using suitable example? (DEC/JAN 2013)

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.

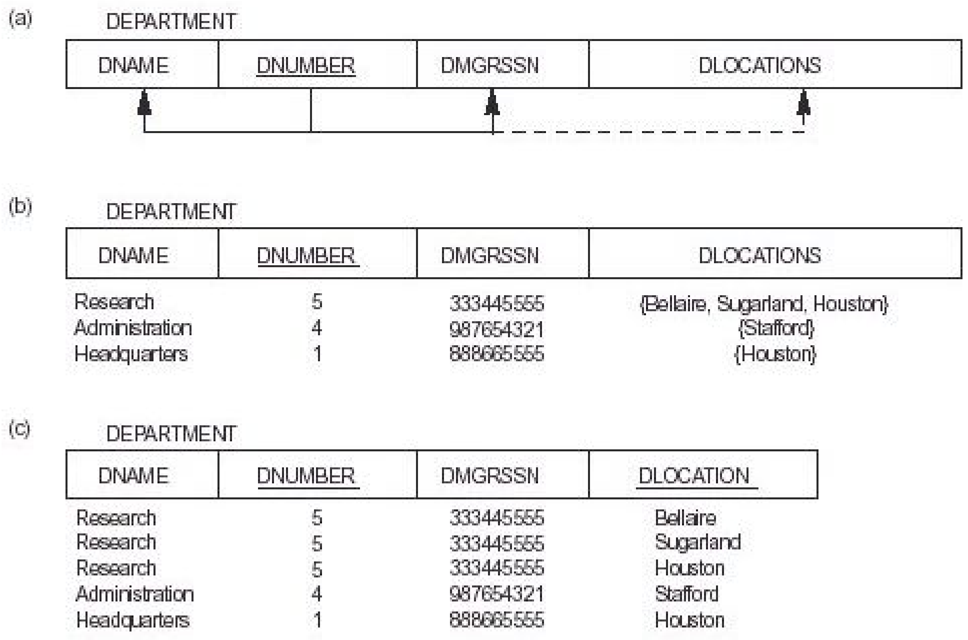


Figure 14.8 Normalization into 1NF. (a) Relation schema that is not in 1NF. (b) Example relation instance. (c) 1NF relation with redundancy.

(a) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS

(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

10. Differentiate between Prime and Non-prime attributes? (DEC/JAN 2013)

A **non-prime attribute** of R is an attribute that does not belong to any candidate key of R. A transitive dependency is a functional dependency in which $X \rightarrow Z$ (X determines Z) indirectly, by
virtue of $X \rightarrow Y$ and $Y \rightarrow Z$ (where it is not the case that $Y \rightarrow X$).

A-X, the set difference between A and X is a **prime attribute** (i.e., *A-X* is contained within a candidate key

c) Consider the Relation R and FD A->B,C->DF,AC->E,D->F?WHAT IS KEY AND Highest normal form? if it is not in 3nf find decomposition that is lossless and dependency preserving?

8 marks

Ans: The key here is E because it has no incoming only outgoing edges.

A->B,C->DF,AC->E,D->F

12) what is functional dependencies? Who specifies functional dependencies for the attributes

UNIT – 7

Database Design -2

1. **Explain multivalued dependency and fourth normal form 4NF with examples.**

(JUN/JULY 2013 / JAN 2014)

A **multivalued dependency** (MVD) X □□□□Y specified on R, where X, and Y are both subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1 [Z]$

Fourth Normal Form (4NF)

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s). A Relation is in 4NF if it is in 3NF and there is no multivalued dependencies.

2) Explain lossless join property. (Dec /Jan 12)

Lossless (Non-additive) Join Property of a Decomposition:

Definition: Lossless join property: a decomposition D = {R1, R2, ..., Rm} of R has the **lossless (nonadditive) join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where * is the natural join of all the relations in D:

$* (\pi R1(r), ..., \pi Rm(r)) = r$

Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for "loss of information" a better term is "**addition of spurious information**"

Algorithm 11.1: Testing for Lossless Join Property

Input: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and aset F of functional dependencies.

1. Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.

2. Set S(i,j):=bij for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j) *).

3. For each row i representing relation schema Ri

```
        {for each column j representing attribute Aj
            {if (relation Ri includes attribute Aj) then set S(i,j):= aj;};};
    (* each aj is a distinct symbol associated with index (j) *)
```

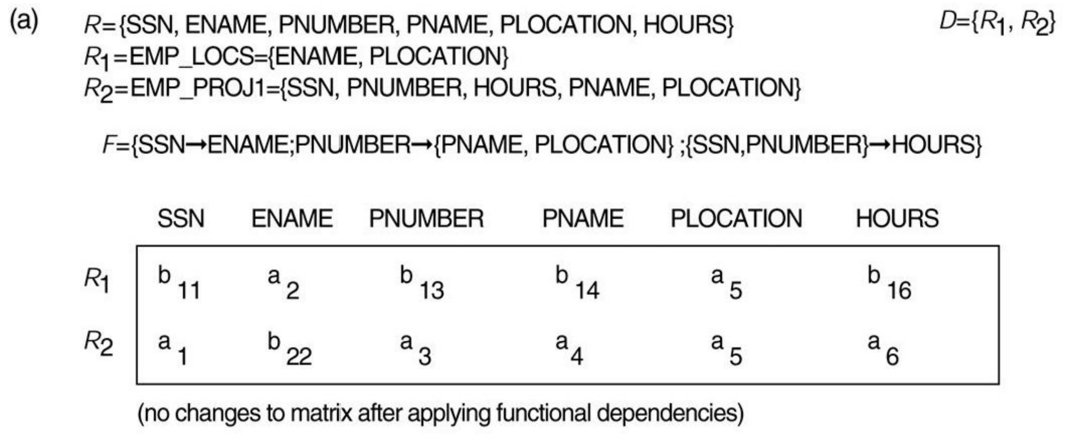
Algorithm 11.1: Testing for Lossless Join Property

4. Repeat the following loop until a complete loop execution results in no changes to S

```
    {for each functional dependency X Y in F
{for all rows in S which have the same symbols in the columns corresponding to attributes in X{make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: If any of the rows has an "a" symbol for the column, set the other rows to that same "a" symbol in the column. If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that same "b" symbol in the column ;};
        };
    };
```

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it does not.

Lossless (nonadditive) join test for *n*-ary decompositions.
(a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test.
(b) A decomposition of EMP_PROJ that has the lossless join property

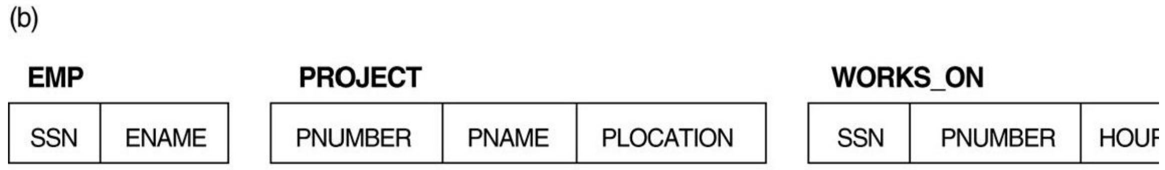


(a) R={SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS} $D=\{R_1, R_2\}$
R_1=EMP_LOCS={ENAME, PLOCATION}
R_2=EMP_PROJ1={SSN, PNUMBER, HOURS, PNAME, PLOCATION}

F={SSN→ENAME;PNUMBER→{PNAME, PLOCATION} ;{SSN,PNUMBER}→HOURS}

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(no changes to matrix after applying functional dependencies)



(b)

EMP

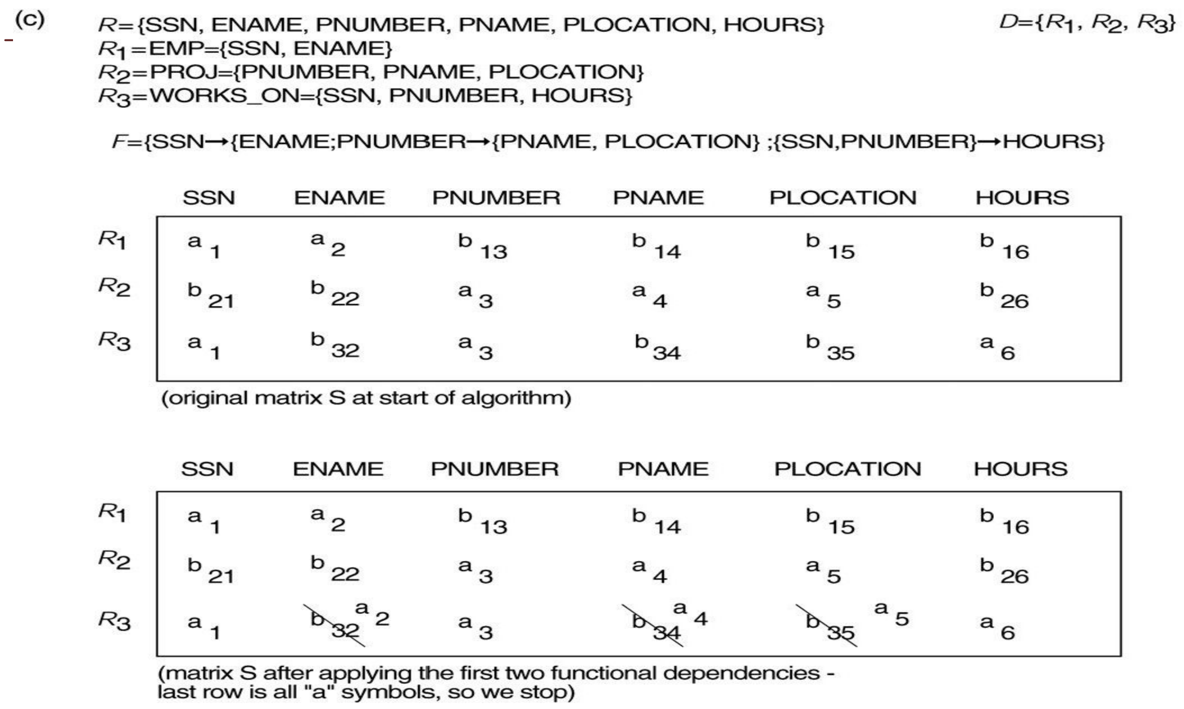
SSN	ENAME

PROJECT

PNUMBER	PNAME	PLOCATION

WORKS_ON

SSN	PNUMBER	HOUR



(c) R={SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS} $D=\{R_1, R_2, R_3\}$
R_1=EMP={SSN, ENAME}
R_2=PROJ={PNUMBER, PNAME, PLOCATION}
R_3=WORKS_ON={SSN, PNUMBER, HOURS}

F={SSN→{ENAME;PNUMBER→{PNAME, PLOCATION} ;{SSN,PNUMBER}→HOURS}

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(original matrix S at start of algorithm)

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	$\cancel{b_{32}}$ a_2	a_3	$\cancel{b_{34}}$ a_4	$\cancel{b_{35}}$ a_5	a_6

(matrix S after applying the first two functional dependencies - last row is all "a" symbols, so we stop)

Lossless (nonadditive) join test for n-ary decompositions.
(c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test

3) Explain multivalued dependency and fourth normal form,with an example .

(JULY 2013 / July 2015)

Multivalued Dependencies and Fourth Normal Form (4NF)

4NF associated with a dependency called **multi-valued dependency** (MVD). MVDs in a relation are due to first normal form (1NF), which disallows an attribute in a row from having a set of values. MVD represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B, and a set of values for C. However, the set of values for B and C are independent of each other.MVD between attributes A, B, and C in a relation using the following notation

A - > - > B (A multidetermines B)

A - > - > *C*

Formal Definition of Multivalued Dependency

A **multivalued dependency** (MVD) X □□□□Y specified on R, where X, and Y are both subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1 [Z]$

Fourth Normal Form (4NF)

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s).

4) Explain lossless join property **(MAY/JULY 2013)**

Lossless (Non-additive) Join Property of a Decomposition:

Definition: Lossless join property: a decomposition D = {R1, R2, ..., Rm} of R has the **lossless (nonadditive) join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where * is the natural join of all the relations in D:

* (π R1(r), ..., πRm(r)) = r

Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for "loss of information" a better term is "**addition of spurious information**"

Algorithm 11.1: Testing for Lossless Join Property

Input: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.

2. Set S(i,j):=bij for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j)*).

```
3. For each row i representing relation schema Ri
        {for each column j representing attribute Aj
            {if (relation Ri includes attribute Aj) then set S(i,j):= aj;};};
      (* each aj is a distinct symbol associated with index (j) *)
```

Algorithm 11.1: Testing for Lossless Join Property

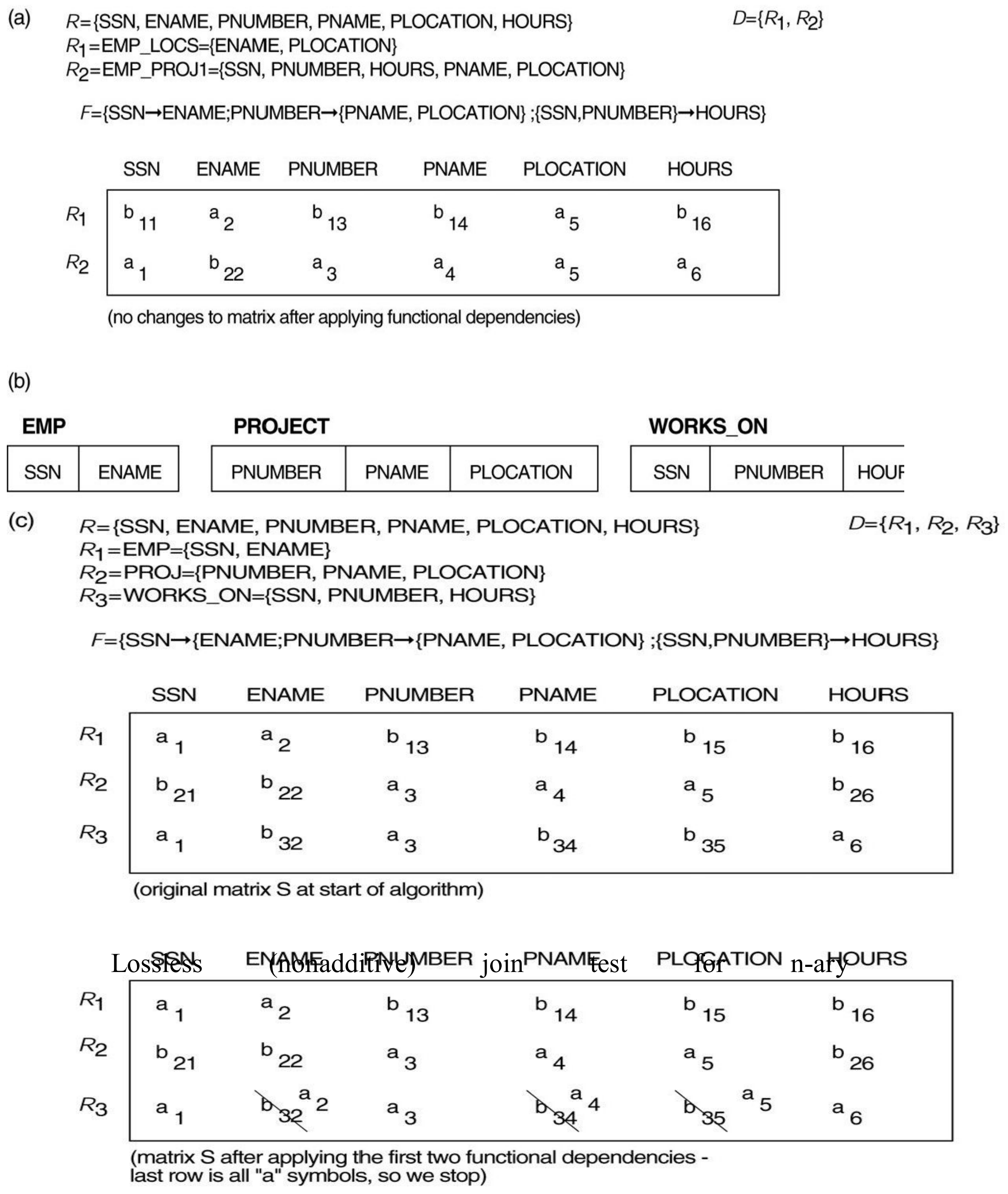
```
4. Repeat the following loop until a complete loop execution results in no changes to S
    {for each functional dependency X →Y in F
        {for all rows in S which have the same symbols in the columns
         corresponding to attributes in X
            {make the symbols in each column that correspond to an attribute in Y be the same in all
             these rows as follows:
             If any of the rows has an "a" symbol for the column, set the other rows to that same "a"
             symbol in the column.
             If no "a" symbol exists for the attribute in any of the rows, choose
             one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to
             that same "b" symbol in the column ;};
        };
    };
```

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it does not.

Lossless (nonadditive) join test for *n*-ary decompositions.

(a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test.

(b) A decomposition of EMP_PROJ that has the lossless join property



(a) R={SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS} $D=\{R_1, R_2\}$

R_1=EMP_LOCS={ENAME, PLOCATION}

R_2=EMP_PROJ1={SSN, PNUMBER, HOURS, PNAME, PLOCATION}

F={SSN→ENAME;PNUMBER→{PNAME, PLOCATION} ;{SSN,PNUMBER}→HOURS}

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(no changes to matrix after applying functional dependencies)

(b)

EMP

SSN	ENAME

PROJECT

PNUMBER	PNAME	PLOCATION

WORKS_ON

SSN	PNUMBER	HOUF

(c) R={SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS} $D=\{R_1, R_2, R_3\}$

R_1=EMP={SSN, ENAME}

R_2=PROJ={PNUMBER, PNAME, PLOCATION}

R_3=WORKS_ON={SSN, PNUMBER, HOURS}

F={SSN→{ENAME;PNUMBER→{PNAME, PLOCATION} ;{SSN,PNUMBER}→HOURS}

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(original matrix S at start of algorithm)

Lossless (nonadditive) join test for n-ary

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	$\cancel{b_{32}}\ a_2$	a_3	$\cancel{b_{34}}\ a_4$	$\cancel{b_{35}}\ a_5$	a_6

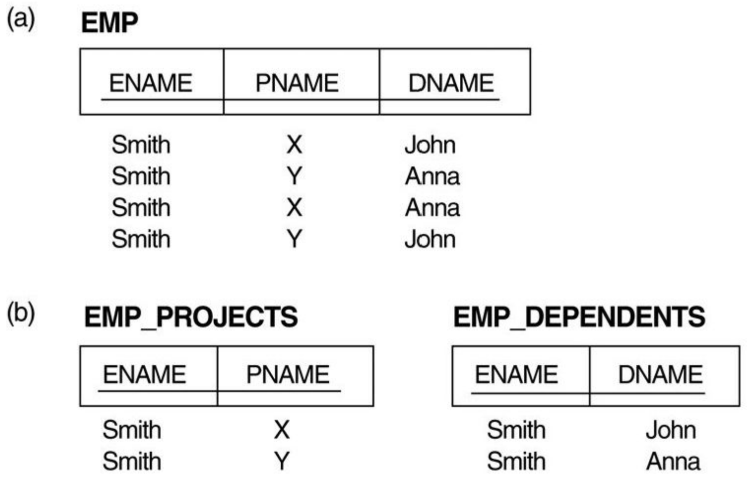
(matrix S after applying the first two functional dependencies - last row is all "a" symbols, so we stop) decompositions.

(c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test

5. What is Multivalve Dependency **(Dec /Jan 11)**

(a) The EMP relation with two MVDs: ENAME —>> PNAME and ENAME —>> DNAME.

(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.



(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

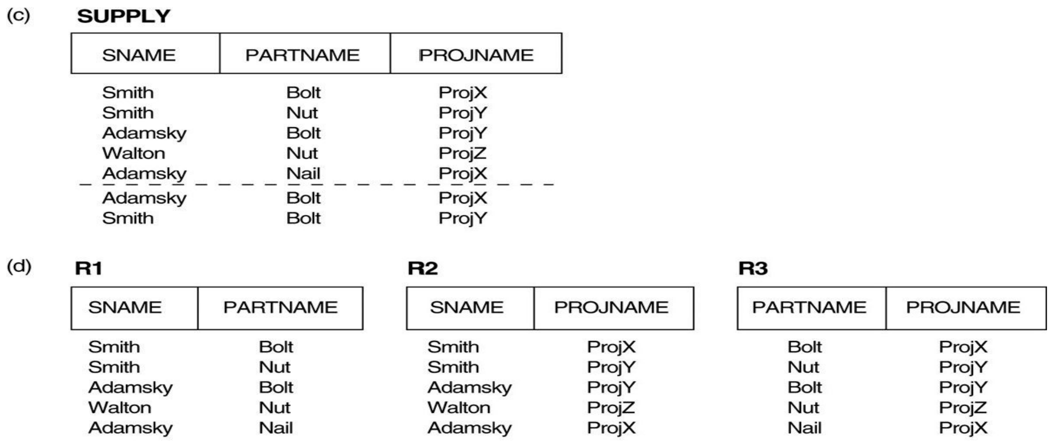
(b) **EMP_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3).
(d) Decomposing the relation SUPPLY into the 5NF relations R1, R2, and R3.



(c) **SUPPLY**

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) **R1**

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

6) **Which Normal form is based on a concept of MVD?explain the same with example?**

(Jan 2013/July 2013/Jan 2014)

Multivalued Dependencies and Fourth Normal Form (4NF)

4NF associated with a dependency called **multi-valued dependency** (MVD). MVDs in a relation are due to first normal form (1NF), which disallows an attribute in a row from having a set of values. MVD represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B, and a set of values for C. However, the set of values for B and C are independent of each other. MVD between attributes A, B, and C in a relation using the following notation

A ->->B (A multidetermines B)

A ->->C

Formal Definition of Multivalued Dependency

A **multivalued dependency** (MVD) X Y specified on R, where X, and Y are both

subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1 [Z]$

<u>Fourth Normal Form (4NF)</u>

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s). A Relation is in 4NF if it is in 3NF and there is no multivalued dependencies.

7 . Explain Fifth Normal Form **<u>(JUN/JULY 2013)</u>**

A **join dependency** (**JD**), denoted by JD{R_1, R_2, …, R_n}, specified on relation schema *R,* specifies a constraint on the states *r* of *R*. The constraint states that every legal state *r* of *R* should have a lossless join decomposition into R_1, R_2, …, R_n; that is, for every such *r* we have

$* (\pi_{R1}(r), (\pi_{R2}(r), \quad \dots (\pi_{Rn}(r)) = r$

Lossless-join property refers to when we decompose a relation into two relations - we can rejoin the resulting relations to produce the original relation. However, sometimes there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and 5NF.

5NF (or project-join normal form (PJNF))

A relation that has no join dependency.

Template Dependencies

The idea behind template dependencies is to specify a template—or example—that defines each constraint or dependency. There are two types of templates: tuple-generating templates and constraint-generating templates. A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the template conclusion. For tuple-generating templates, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there. For constraint-generating templates, the template conclusion is a condition that must hold on the hypothesis tuples.

Domain Key Normal Form

The idea behind **domain-key normal form (DKNF)** is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints. A relation is said to be in **DKNF** if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and key constraints on the relation. However, because of the difficulty of including complex constraints in a DKNF relation, its practical utility is limited, since it may be quite difficult to specify general integrity constraints.

For example, consider a relation CAR(MAKE, VIN#) (where VIN# is the vehicle identification number) and another relation MANUFACTURE(VIN#, COUNTRY) (where COUNTRY is the country of manufacture). A general constraint may be of the following form: "If the MAKE is either Toyota or Lexus, then the first character of the VIN# is a "*J*" if the country of manufacture is Japan; if the MAKE is Honda or Acura, the second character of the VIN# is a "*J*" if the country of manufacture is Japan." There is no simplified way to represent such constraints short of writing a procedure (or general assertions) to test them.

UNIT-VIII

Transaction Management

1) Explain properties of a transaction ? (JUN/JULY 2015)

Transactions should posses the following (ACID) properties:

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

1. **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
2. **Consistency preservation:** A transaction is consistency preserving if its complete execution take(s) the database from one consistent state to another.
3. **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **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.

The atomicity property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database.

2. What is a schedule? Explain with example serial, non serial and conflict serializable

schedules. (JUN/JULY 2013)

A **schedule** (or **history**) *S* of *n* transactions T1, T2, ..., Tn is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in *S,* the operations of Ti in *S* must appear in the same order in which they occur in Ti. Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in *S*. For now, consider the order of operations in *S* to be a *total ordering,* although it is possible theoretically to deal with schedules whose operations form *partial orders*.

$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

which we call Sb, can be written as follows, if we assume that transaction T1 aborted after its read_item(*Y*) operation:

$$S_b: r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$$

Two operations in a schedule are said to **conflict** if they satisfy all three of the following conditions:

1. they belong to different transactions;
2. they access the same item *X;* and
3. at least one of the operations is a write_item(*X*).

For example, in schedule S_a, the operations $r_1(X)$ and $w_2(X)$ conflict, as do the operations

$r_2(X)$ and $w_1(X)$,

and the operations w1(*X*) and w2(*X*). However, the operations r1(*X*) and

r2(*X*) do not conflict, since they are both read operations; the operations w2(*X*) and w1(*Y*) do not conflict, because they operate on distinct data items *X* and *Y;* and the operations r1(*X*) and w1(*X*) do not conflict, because they belong to the same transaction.

- A schedule *S* is *serial* if, for every transaction *T* all the operations of *T* are executed consecutively in the schedule.
- A schedule *S* of *n* transactions is *serializable* if it is equivalent to some serial schedule of the same *n* transactions

3.Write short notes on (Jan 2013/Jan2014)

i) Write a head log protocol:

When in-place update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by Write-Ahead Logging (WAL) protocol. WAL states that

For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

i) Transaction support in SQL: A single SQL statement is always considered to be atomic. Either the statement completes execution without error or it fails and leaves the database unchanged. With SQL, there is no explicit Begin Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered. Every transaction must have an explicit end statement, which is either a

COMMIT or ROLLBACK.
Characteristics specified by a SET TRANSACTION statement in SQL2:

Access mode: READ ONLY or READ WRITE.
The default is READ WRITE unless the isolation level of READ UNCOMITTED is specified, in which case READ ONLY is assumed.

Diagnostic size n, specifies an integer value n, indicating the number of conditions that can be held simultaneously in the diagnostic area.

Isolation level <isolation>, where <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ or SERIALIZABLE. The default is SERIALIZABLE. With SERIALIZABLE: the interleaved execution of transactions will adhere to our notion of serializability. However, if any transaction executes at a lower level, then serializability may be violated.
Potential problem with lower isolation levels:

Dirty Read:

Reading a value that was written by a transaction which failed.

Nonrepeatable Read:

Allowing another transaction to write a new value between multiple reads of one transaction.

A transaction T1 may read a given value from a table. If another transaction T2 later updates that value and T1 reads that value again, T1 will see a different value. Consider that T1 reads the employee salary for Smith. Next, T2 updates the salary for Smith. If T1 reads Smith's salary again, then it will see a different value for Smith's salary.

Phantoms:New rows being read using the same read with a condition. A transaction T1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE clause. Now suppose that a transaction T2 inserts a new row that also satisfies the WHERE clause condition of T1, into the table used by T1. If T1 is repeated, then T1 will see a row that previously did not exist, called a phantom.

iii) Two phase locking protocol:

Two-Phase Locking Techniques: Essential components

Two locks modes (a) shared (read) and (b) exclusive (write). Shared mode: shared lock (X). More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction. Exclusive mode: Write lock (X). Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Conflict matrix

Read	Write
Y	N
N	N

Lock Manager: Managing locks on data items.

Lock table: Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock

table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Database requires that all transactions should be well-formed. A transaction is well-formed if:

- It must lock the data item before it reads or writes to it.
- It must not lock an already locked data items and it must not try to unlock a free dataitem.

The following code performs the lock operation:

```
B:     if LOCK (X) = 0 (*item is unlocked*)
       then LOCK (X) ← 1 (*lock the item*)
       else begin
               wait (until lock (X) = 0) and
               the lock manager wakes up the transaction);
       goto B
       end;
```

The following code performs the unlock operation:

```
       LOCK (X) ← 0 (*unlock the item*)
       if any transactions are waiting then
               wake up one of the waiting the transactions;
```

iv) Time stamp Ordering:

Timestamp

A monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation. Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions

Basic Timestamp Ordering

1. Transaction T issues a write_item(X) operation:

a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then an younger transaction has already read the data item so abort and roll-back T and reject the operation.

If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).

2. Transaction T issues a read_item(X) operation:

If write_TS(X) > TS(T), then an younger transaction has already written to the data item so abort and roll-back T and reject the operation.

If write_TS(X) ≤ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X)

5 explain the problems that can occur when concurrent transaction are executed give examples <u>(JULY 2013/ Jan 2014/July 2015)</u>

Why Concurrency Control is needed:

The Lost Update Problem

This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

The Temporary Update (or Dirty Read) Problem

This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 17.1.4).

The updated item is accessed by another transaction before it is changed back to its original value.

The Incorrect Summary Problem

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

Two-Phase Locking Techniques

Locking is an operation which secures (a) permission to Read or (b) permission to Write a data item for a transaction. Example: Lock (X). Data item X is locked in behalf of the requesting transaction. Unlocking is an operation which removes these permissions from the data item.

Example: Unlock (X). Data item X is made available to all other transactions.

Lock and Unlock are Atomic operations.

Two-Phase Locking Techniques: Essential components

Two locks modes (a) shared (read) and (b) exclusive (write).

Shared mode: shared lock (X). More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

Exclusive mode: Write lock (X). Only one write lock on X can exist at any time and no

shared lock can be applied by any other transaction on X.

Conflict matrix

conflict matrix

Two-Phase Locking Techniques: Essential components

Lock Manager: Managing locks on data items.

Lock table: Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode Ptr	to next data item
T1	X1	Read	Next

Two-Phase Locking Techniques: Essential components

Database requires that all transactions should be well-formed. A transaction is well-formed if:

- It must lock the data item before it reads or writes to it.
- It must not lock an already locked data items and it must not try to unlock a free dataitem.

The following code performs the lock operation:

```
B:    if LOCK (X) = 0 (*item is unlocked*)
      then LOCK (X) ← 1 (*lock the item*)
      else begin
            wait (until lock (X) = 0) and
```

```
                 the lock manager wakes up the transaction);
         goto B
         end;
```

The following code performs the unlock operation:

```
         LOCK (X) ← 0 (*unlock the item*)
         if any transactions are waiting then
                 wake up one of the waiting the transactions
```

The following code performs the read operation:

```
        B: if LOCK (X) = “unlocked” then
begin LOCK (X) ← “read-locked”;
        no_of_reads (X) ← 1;
end
else if LOCK (X) ← “read-locked” then
            no_of_reads (X) ← no_of_reads (X) +1
          else begin wait (until LOCK (X) = “unlocked” and
                 the lock manager wakes up the transaction);
                 go to B
               end;
```

The following code performs the write lock operation:

```
        B: if LOCK (X) = “unlocked” then
 LOCK (X) ← “write-locked”;
else begin
     wait (until LOCK (X) = “unlocked” and
                 the lock manager wakes up the transaction);
                 go to B
               end;
```

The following code performs the unlock operation:

```
        if LOCK (X) = “write-locked” then
begin LOCK (X) ← “unlocked”;
         wakes up one of the transactions, if any
```

```
end
else if LOCK (X) ← "read-locked" then
    begin
        no_of_reads (X) ← no_of_reads (X) -1
        if  no_of_reads (X) = 0 then
        begin
            LOCK (X) = "unlocked";
            wake up one of the transactions, if any
        end
    end;
```

Lock conversion

Lock upgrade: existing read lock to write lock

```
        if Ti has a read-lock (X) and Tj has no read-lock (X) (i ≠ j) then
    convert read-lock (X) to write-lock (X)
else
    force Ti to wait until Tj unlocks X
```

Lock downgrade: existing write lock to read lock

```
        Ti has a write-lock (X)    (*no transaction can have any lock on X*)
    convert write-lock (X) to read-lock (X)
```

Two Phases: (a) Locking (Growing) (b) Unlocking (Shrinking).

Locking (Growing) Phase: A transaction applies locks (read or write) on desired data items one at a time.

Unlocking (Shrinking) Phase: A transaction unlocks its locked data items one at a time.

Requirement: For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Two-Phase Locking Techniques: The algorithm

T1	**T2**	**Result**
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2

write_lock (X);	Write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Two-Phase Locking Techniques: The algorithm

T1	**T2**	**<u>Result</u>**
read_lock (Y);		X=50; Y=50
read_item (Y);		Nonserializable because it.
unlock (Y);		violated two-phase policy.
	read_lock (X);	
	read_item (X);	
	unlock (X);	
	write_lock (Y);	
	read_item (Y);	
	Y:=X+Y;	
	write_item (Y);	
	unlock (Y);	
write_lock (X);		
read_item (X);		
X:=X+Y;		
write_item (X);		
unlock (X);		

Two-Phase Locking Techniques: The algorithm

<u>T'1</u>	**<u>T'2</u>**	
read_lock (Y);	read_lock (X);	T1 and T2 follow two-phase
read_item (Y);	read_item (X);	policy but they are subject to
write_lock (X);	Write_lock (Y);	deadlock, which must be
unlock (Y);	unlock (X);	dealt with.
read_item (X);	read_item (Y);	
X:=X+Y;	Y:=X+Y;	

```
        write_item (X);          write_item (Y);
        unlock (X);              unlock (Y);
```

Two-phase policy generates two locking algorithms (a) Basic and (b) Conservative.

Conservative: Prevents deadlock by locking all desired data items before transaction begins execution.

Basic: Transaction locks data items incrementally. This may cause deadlock which is dealt with.

Strict: A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

6. What is Serializibility?How can seriaizability?Justify your answer?

(Dec 11/July 13/ JAN 2014)

Serializability of Schedules

If no interleaving of operations is permitted, there are only two possible arrangement for transactions T1 and T2.

Execute all the operations of T1 (in sequence) followed by all the operations of T2 (in sequence).

Execute all the operations of T2 (in sequence) followed by all the operations of T1

A schedule *S* is *serial* if, for every transaction *T* all the operations of *T* are executed consecutively in the schedule.

A schedule *S* of *n* transactions is *serializable* if it is equivalent to some serial schedule of the same *n* transactions.

7. Write a short Notes (Dec /Jan 11)

a) 2PL Lock:

Locking is an operation which secures (a) permission to Read or (b) permission to Write a data item for a transaction. Example: Lock (X). Data item X is locked in behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item.

Example: Unlock (X). Data item X is made available to all other transactions.

Lock and Unlock are Atomic operations.

Two locks modes (a) shared (read) and (b) exclusive (write).

Shared mode: shared lock (X). More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

Exclusive mode: Write lock (X). Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Lock Manager: Managing locks on data items.

Lock table: Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Two-Phase Locking Techniques: Essential components

The following code performs the lock operation:

```
B:  if LOCK (X) = 0 (*item is unlocked*)
    then LOCK (X) ← 1 (*lock the item*)
    else begin
        wait (until lock (X) = 0) and
        the lock manager wakes up the transaction);
    goto B
    end;
```

Dead Locks:

Deadlock

T'1 **T'2**

read_lock (Y); T1 and T2 did follow two-phase read_item (Y);

```
                    policy but they are deadlock
                                read_lock (X);
                                read_item (Y);
write_lock (X);
(waits for X)                   write_lock (Y);
                                (waits for Y)
```

Deadlock (T'1 and T'2)

Deadlock prevention

A transaction locks all data items it refers to before it begins execution. This way of locking prevents deadlock since a transaction never waits for a data item. The conservative two-phase locking uses this approach.

Deadlock detection and resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.

A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle. One of the transaction of the cycle is selected and rolled back.

Deadlock avoidance

There are many variations of two-phase locking algorithm. Some avoid deadlock by not letting the cycle to complete. That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction. Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim

Starvation

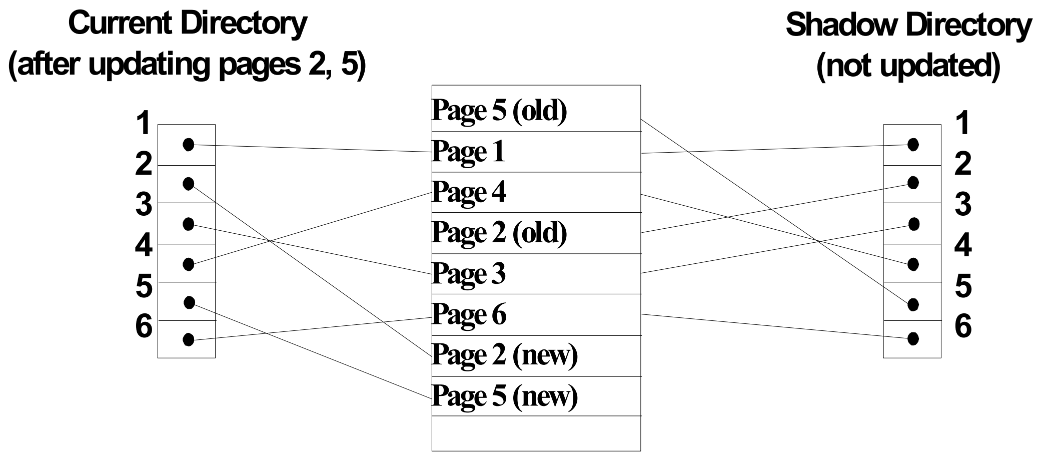
Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.. In Wound-Wait scheme a younger transaction may In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back. This limitation is inherent in all priority based scheduling mechanisms always be wounded (aborted) by a long running older transaction which may create starvation.

. ARIES: The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

X and Y: Shadow copies of data items

X` and Y`: Current copies of data items

To manage access of data items by concurrent transactions two directories (current and shadow) are used. The directory arrangement is illustrated below. Here a page is a data item



8 Explain two Phase Locking Protocol and its disadvantages?

(DEC/JAN 2013/ Jan 2014)

Locking is an operation which secures (a) permission to Read or (b) permission to Write a data item for a transaction. Example: Lock (X). Data item X is locked in behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item.

Example: Unlock (X). Data item X is made available to all other transactions.

Two-Phase Locking Techniques: Essential components

The following code performs the lock operation:

```
B:    if LOCK (X) = 0 (*item is unlocked*)
      then LOCK (X) ← 1 (*lock the item*)
      else begin
            wait (until lock (X) = 0) and
            the lock manager wakes up the transaction);
      goto B
```

end;

9 Explain Time stamp ordering algorithm. (JAN 2013)

a) Time stamp ordering algorithm.

A monotonically increasing variable (integer) indicating the age of an operation or a transaction.

A larger timestamp value indicates a more recent event or operation. Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions

Basic Timestamp Ordering

1. Transaction T issues a write_item(X) operation:

If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then an younger transaction has already read the data item so abort and roll-back T and reject the operation.

If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).

2. Transaction T issues a read_item(X) operation:

If write_TS(X) > TS(T), then an younger transaction has already written to the data item so abort and roll-back T and reject the operation.

If write_TS(X) ≤ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X

10. ARIES Algorithm (July 2013)

The ARIES Recovery Algorithm is based on:

1. WAL (Write Ahead Logging)
2. Repeating history during redo: ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.
3. Logging changes during undo: It will prevent ARIES from repeating the

completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

The ARIES recovery algorithm consists of three steps:

1. **Analysis**: step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash. The appropriate point in the log where redo is to start is also determined.
2. **Redo**: necessary redo operations are applied.
3. **Undo**: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

The Log and Log Sequence Number (LSN)

A log record is written for (a) data update, (b) transaction commit, (c) transaction abort, (d) undo, and (e) transaction end. In the case of undo a compensating log record is written. A unique LSN is associated with every log record. LSN increases monotonically and indicates the disk address of the log record it is associated with. In addition, each data page stores the LSN of the latest log record corresponding to a change for that page. A log record stores (a) the previous LSN of that transaction, (b) the transaction ID, and (c) the type of log record.

A log record stores:

1. Previous LSN of that transaction: It links the log record of each transaction. It is like a back pointer points to the previous record of the same transaction.
2. Transaction ID
3. Type of log record.
4. Page ID for the page that includes the item
5. Length of the updated item
6. Its offset from the beginning of the page
7. BFIM of the item

8. AFIM of the item

A checkpointing does the following:

1. Writes a *begin_checkpoint* record in the log
2. Writes an *end_checkpoint* record in the log. With this record the contents of transaction table and dirty page table are appended to the end of the log.
3. Writes the LSN of the *begin_checkpoint* record to a special file. This special file is accessed during recovery to locate the last checkpoint information.