CHAPTER FIVE 5

Entity Relationship Data Model

5.1 Entity Relationship (ER) Model

Entity Relationship Model/ Diagram is a conceptual model.

Entity Relationship Data Model

- It is based on a configuration of our real world perception into object sets called entities and the relationships among these objects.
- A logic tool is used for database scheme design.
- It does not include implementation details.
- It is described by an ER (Entity-Relationship) diagram.

Note

Implementation details are physical and have no place in an ER diagram. The ER diagram displays logical relationship, not physical relationship.

Entity

- An entity is a thing that has an independent existence.
- An entity is described by its attributes.
- An entity is determined by its instantiations.

(Instantiations are particular values for its attributes.)

■ **Example** A customer is an entity with the attributes:
Name
ID No
Address
Telephone No
An account is an entity with the attributes:
Account No
Balance

Entity Set (Entity Type)

- Define a set of entities of the same type (share the same structure)
- Denote by a rectangular box in ER diagram

- Identify entity by a list of attributes placed in ovals
- Identify key attributes (the set of attributes that uniquely identify entity type)

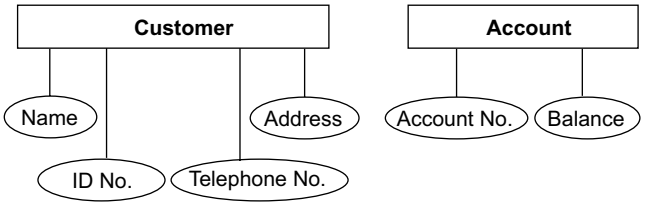


Figure 5.1 An entity representation with its attributes

Relationship and Relationship Set

A relationship is an association among several entities.

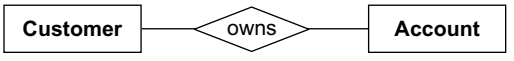
A relationship set is a set of relationships of the same type.

Let $E_1, E_2, \ldots E_n$ be a set of entity sets.

$< e_1, e_2, \ldots, e_n >$ is a relationship, where e_k is contained in E_k.

A subset of $E_1 \times E_2 \times \ldots \times E_n$ is a relationship set.

■ **Example**



Relationship Sets

Defines an association of entity sets.

Is a subset of cartesian product $\mathbf{E}_1 \times \mathbf{E}_2 \times \ldots \times \mathbf{E}_n$.

$\boldsymbol{E}_k$ is said to play a role in the relationship set.

Denoted by a diamond in the ER diagram as shown in Fig. 5.2.

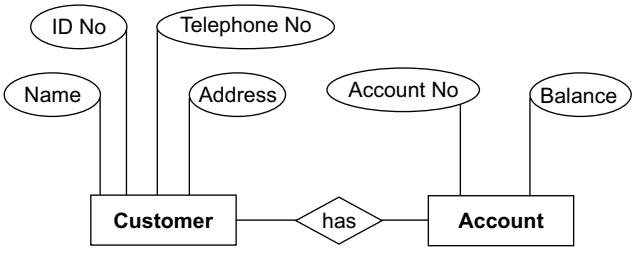


Figure 5.2 A relation

Descriptive Attributes

- Attributes of relationships

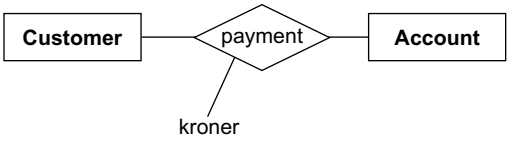


Figure 5.3 Payment relation (Please ask the people to remove the word kroner and the line which joinsit)

- Structural constraints

Degree: number of participating entity sets

Cardinality constraints: {1:1, 1:N, M:N}

Participation constraints: partial or total

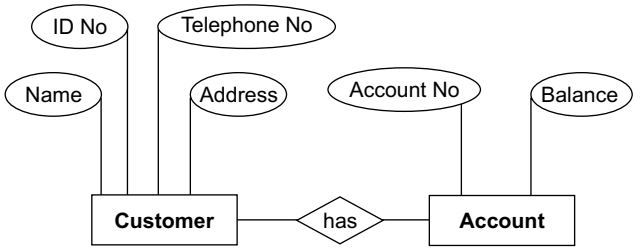


Figure 5.4 Customer and account relation

1:1 each customer has at most one account and each account is owned by at most one customer.
1:N each customer may have any number of accounts but each account is owned by at most one customer.
M:N each customer may have any number of accounts and each account may be owned by any number of customers.

5.1.1 Representing ER Model Using Tables

Basic rules

- One table for one entity set
- One column for one attribute

Superkey
A ***superkey*** is a set of one or more attributes which, taken collectively, allow us to identify uniquely an entity in the entity set.
■ **Example** In the entity set ***customer, customer-name and personal-number*** is a superkey. Note that ***customer-name*** alone is not a superkey, as two customers could have the same name.

Candidate Key
A superkey may contain extraneous attributes, and we are often interested in the smallest superkey. A superkey for which no subset is a superkey is called a ***candidate key.***

■ **Example** ***personal_number*** is a candidate key, as it is minimal and uniquely identifies a customer entity.

Primary Key
Primary key is a candidate key (there may be more than one) chosen by the DB designer to identify entities in an entity set. An entity set that does not possess sufficient attributes to form a primary key is called a ***weak entity set.*** One that does have a primary key is called a ***strong entity set.***

Relation: Customer

Customer No.	Name	Address	Post Code	Gender	Age	Date of Birth
Siddm02	Siddke	48 South St	3070	F	38	04/12/1954
Walsh01	H Walsh	2 Allen Crt	3065	M	44	04/16/1947
Foret13	T Forest	69 Black St	3145	M	24	06/12/1967
Richb76	B Rich	181 Kemp St	3507	M	50	09/12/1941

5.1.1.1 Conceptual Design/Developing the ER Model

- Do not think about the implementation.
- Conceptual model is independent of DBMS.

Basic ER Model Constructs
Entity

- An ***entity*** is an object that exists and is distinguishable from other objects (like a specific person, a company, a book).

Synonyms: Instance (an instantiation), member, individual
An Entity can be uniquely identified as one particular object in the universe. An entity may be **concrete** (like a person, a book) or **abstract** (like a country border, a holiday, etc.)

Entity Sets

- An entity set is a collection of entities of the same type (identified by the chosen properties).

■ **Example** The students at our university.

Synonym: Entity Class

Entity sets need not be disjoint. For example, the entity set ***employee*** and the entity set ***client*** may have members in common.

Entity Types

- **An entity type is defined by its attributes.**

Note that the entity type is **defined** by its attributes, whereas its instantiations are identified by these attributes.

A database is modeled as a collection of entities and relationships among them.

- **An entity set is a collection of entities that share the same properties.**

Synonym: Entity Class

- **An entity type is defined by its properties (attributes).**

A database is modeled as a collection of entities and relationships among them.

Attributes

- **Descriptive properties possessed by all members of an entity set.**

■ **Example** The set of students in a university form an entity set could be named **Student**.

The attributes of the Student could be: **name, ID No, GPA, …**

Formally, an attribute is a function that maps an entity set in a domain.

Every entity is described by a set of pairs in the form (attribute, data value), such as (street, Tröskaregatan), (city, Linköping)

This is akin to Class/Instantiation pattern for objects in OO programming.

Attribute Domain

The domain of an attribute is the set of permitted values for that attribute.

■ **Example** GPA has a range from 1 to 4.

Attribute Species (kinds of)

Simple vs composite
Single valued vs multivalued
Derived
Null valued
Identifier

- **A relationship may also have attributes.**

Defining a Set of ENTITIES and their RELATIONSHIPS

The definition for a set of entities and their relationships depends on how we deal with attributes. Suppose we have an entity set ***employee*** with attributes ***employee-name*** and ***phone-number.*** Is the phone an entity? If so, then we have two entity sets. Also, this new definition allows employees to have 0 or more phones.

This may be a better representation of the way things are.

Does employee have the same status?

Employee-name does NOT have the same status as phone.

The question of what constitutes an entity and what constitutes an attribute depends on how we interpret the world that is being modeled.

Relationships

Relationships associate one or more entities (usually 2).

5.1.1.2. Degree of Relationship

Unary relationship: associates entity with itself
Binary relationship: associates two entities
Ternary relationship: associates three entities
n-ary relationships: associates n entities

Cardinality and Structural Constraint

Relationships between entities have a cardinality associated with them. The [min:max] cardinality together are often referred to as the ***structural constraint.***

Sometimes we need additional attributes for a relationship, which leads to formation of an associative entity type.

Graphical Symbols used in ER diagrams

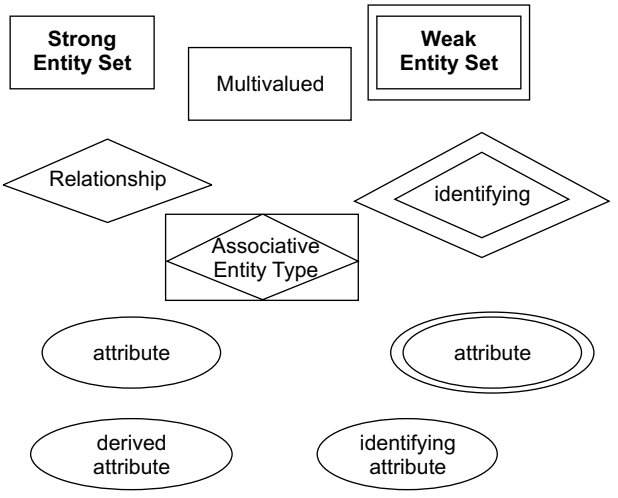


Figure 5.5 Symbols used in representing ER diagrams

Symbols used in representing cardinalities

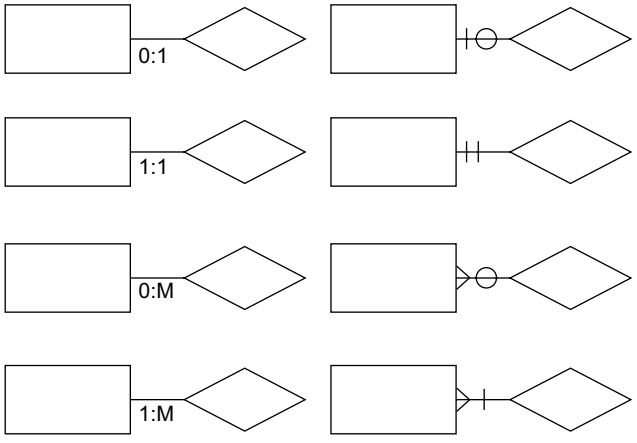


Figure 5.6 Symbolic representation of cardinalities

Existence Dependencies

If the existence of entity X depends on the existence of entity Y then X is said to be existence dependent on Y. (Or we say that Y is the dominant entity and X is the subordinate entity.)

■ **Example** Consider ***account*** and ***transaction*** entity sets, and a relationship ***log*** between them. This is a one-to-many from ***account*** to ***transaction.*** If an account entity is deleted, its associated transaction entities must also be deleted. Thus ***account*** is dominant and ***transaction*** is subordinate.

Strong/Weak Entity Sets

- Weak Entity Set: An entity set whose existence is dependent on one or more other strong entity sets (termed the 'identifying owner(s)')

Thus if an instance of the strong entity set is removed, so must the related instances of the weak entity set.

Strong is dominant.
Weak is subordinate.

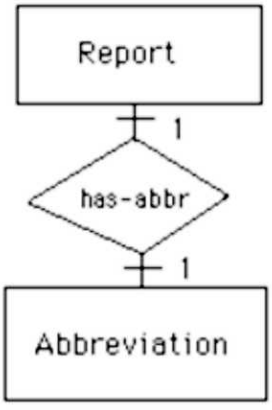
5.1.2 Entity Relationship Data Model

- Based on a configuration of our real world perception into object sets called entities, and the relationships among these objects.
- A logic tool used for database scheme design.
- Does not include implementation details.
- It is described by an ER (Entity-Relationship) diagram.

Note

That implementation details are physical and have no place in an ER diagram. The ER diagram displays logical relationships, not physical relationships.

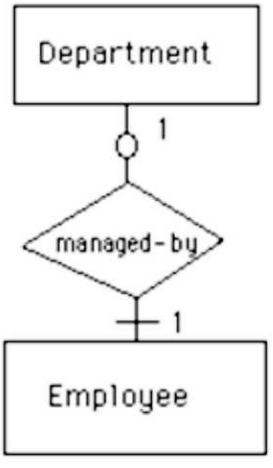
Examples for various cardinalities



(a) one-to-one, both entities mandatory

Every report has one abbreviation, and every abbreviation represents exactly one report.

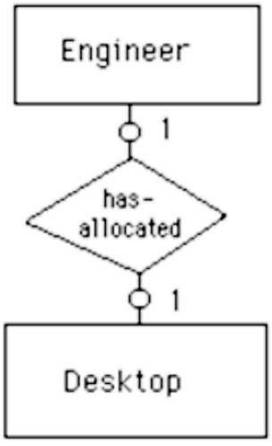
```
create table report
        (report_no  integer,
         report_name  varchar(256),
         primary key(report_no);
create table abbreviation
        (abbr_no  char(6),
         report_no  integer not null unique,
         primary key (abbr_no),
         foreign key (report_no ) references report
           on delete cascade on update cascade);
```



(b) one-to-one, one entity optional, one mandatory

Every department must have a manager, but an employee can be a manager of at most one department

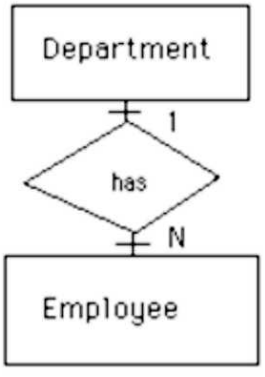
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         mgr_id  char(10) not null unique,
         primary key (dept_no),
         foreign key (mgr_id) references employee
           on delete set default on update cascade);
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         primary key (emp_id));
```



(c) one-to-one,
both entities optional

Some desktop computers are allocated to engineers, but not necessarily to all engineers.

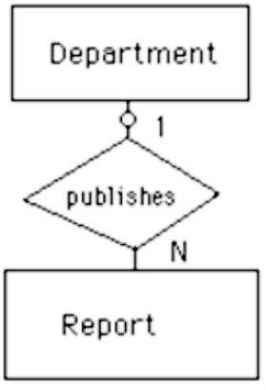
```
create table engineer
        (emp_id  char(10),
         desktop_no  integer,
         primary key (emp_id));

create table desktop
        (desktop_no  integer,
         emp_id  char(10),
         primary key (desktop_no),
         foreign key (emp_id) references engineer
           on delete set null on update cascade);
```



(d) one-to-many, both
entities mandatory

Every employee works in exactly one department, and each department has at least one employee.

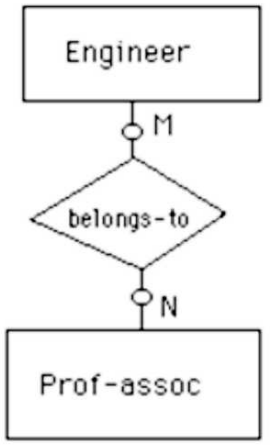
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         primary key (dept_no));
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         dept_no  integer not null,
         primary key (emp_id),
         foreign key (dept_no) references department
           on delete set default on update cascade);
```



(e) one-to-many, one entity
optional, one unknown

Each department publishes one or more reports. A given report may not necessarily be published by a department.

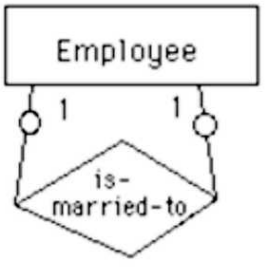
```
create table department
        (dept_no  integer,
         dept_name  char(20),
         primary key (dept_no));
create table report
        (report_no  integer,
         dept_no  integer,
         primary key (report_no),
         foreign key (dept_no) references department
           on delete set null on update cascade);
```



(f) many-to-many, both entities optional

Every professional association could have none, one, or many engineer members. Each engineer could be a member of none, one, or many professional associations.

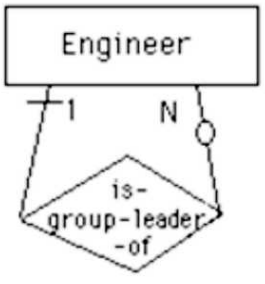
```
create table engineer
       (emp_id  char(10),
        primary key (emp_id));
create table prof_assoc
       (assoc_name  varchar(256),
        primary key (assoc_name));

create table belongs_to
       (emp_id  char(10),
        assoc_name  varchar(256),
        primary key (emp_id, assoc-name),
        foreign key (emp_id) references engineer
          on delete cascade on update cascade,
        foreign key (assoc_name ) references prof-assoc
          on delete cascade on update cascade);
```



(a) one-to-one, both sides optional

Any employee is allowed to be married to another employee in this company.

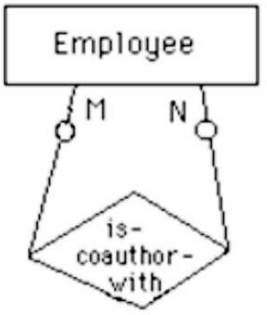
```
create table employee
       (emp_id  char(10),
        emp_name  char(20),
        spouse_id  char(10),
        primary key (emp_id),
        foreign key (spouse_id) references employee
          on delete set null on update cascade);
```



(b) one-to-many, one side mandatory, many side optional

Engineers are divided into groups for certain projects. Each group has a leader.

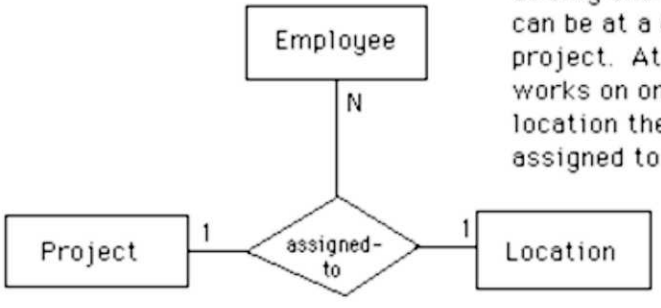
```
create table engineer
       (emp_id  char(10),
        leader_id  char(10) not null,
        primary key (emp_id),
        foreign key (leader_id) references engineer
          on delete set default on update cascade);
```



Each employee has the opportunity to coauthor a report with one or more other employees, or to write the report alone.

```
create table employee
        (emp_id  char(10),
         emp_name  char(20),
         primary key (emp_id));

create table coauthor
        (author_id  char(10),
         coauthor_id  char(10),
         primary key (author_id, coauthor-id),
         foreign key (author_id) references employee
           on delete cascade on update cascade,
         foreign key (coauthor_id) references employee
           on delete cascade on update cascade);
```

(c) many-to-many, both sides optional



Each employee assigned to a project works at only one location for that project, but can be at a different location for a different project. At a given location, an employee works on only one project. At a particular location there can be many employees assigned to a given project.

```
create table employee (emp_id  char(10),
                       emp_name  char(20),
                       primary key (emp_id));
create table project (project_name  char(20),
                      primary key (project_name));
create table location (loc_name  char(15),
                       primary key (loc_name));
create table assigned_to (emp_id  char(10),
                          project_name  char(20),
                          loc_name  char(15) not null,
                          primary key (emp_id, project_name),
                          foreign key (emp_id) references employee
                            on delete cascade on update cascade,
                          foreign key (project_name) references project
                            on delete cascade on update cascade,
                          foreign key (loc_name) references location
                            on delete cascade on update cascade),
                          unique (emp_id, loc_name));
```

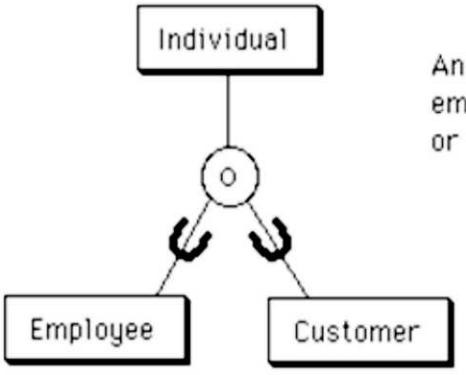
assigned_to

emp_id	project_name	loc_name
48101	forest	B66
48101	ocean	E71
20702	ocean	A12
20702	river	D54
51266	river	G14
51266	ocean	A12
76323	hills	B66

Functional dependencies

emp_id, loc_name -> project_name
emp_id, project_name -> loc_name

(b) one-to-one-to-many ternary relationships



An individual may be either an employee or a customer, or both, or neither.

```
create table individual (indiv_id  char(10),
                         indiv_name  char(20),
                         indiv_addr  char(20),
                         primary key (indiv_id));
create table employee (emp_id  char(10),
                       job_title  char(15),
                       primary key (emp_id),
                       foreign key (emp_id) references individual
                         on delete cascade on update cascade);
create table customer (cust_no  char(10),
                       cust_credit  char(12),
                       primary key (cust_no),
                       foreign key (cust_no) references individual
                         on delete cascade on update cascade);
```

5.1.3 Transforming the Conceptual Data Model to SQL Tables

* **Entity** – directly to a SQL table
* **Many-to-many binary relationship** – directly to a SQL table, taking the 2 primary keys in the 2 entities associated with this relationship as foreign keys in the new table.

* **One-to-many binary relationship** – primary key on "one" side entity copied as a foreign key in the "many" side entity's table.
* **Recursive binary relationship** – same rules as other binary relationships.
* **Ternary relationship** – directly to a SQL table, taking the 3 primary keys of the 3 entities associated with this relationship as foreign keys in the new table.
* A**ttribute of an entity** – directly to be an attribute of the table transformed from this entity.
* **Generalisation super-class (super-type) entity** – directly to a SQL table.
* **Generalisation subclass (subtype) entity** – directly to a SQL table, but with the primary key of its super-class (super-type) propagated down as a foreign key into its table.
* **Mandatory constraint (1 lower bound) on the "one" side of a one-to-many relationship** – the foreign key in the "many" side table associated with the primary key in the "one" side table should be set as "not null" (when the lower bound is 0, nulls are allowed as the default in SQL)

5.1.4 Enhanced ER Model

The following extensions will be added in enhanced-ER model:

1. Class/subclass relationships and type inheritance.
2. Specialisation and generalisation.
3. Constraints on specialisation and generalisation.
4. Union constructs.

5.1.5 File Structures (Sequential files, indexing, B and B + trees)

Read Operating Systems and Data Structures notes

5.2 Introduction to Tuple Relational Calculus (TRC)

Although relational algebra is useful in the analysis of query evaluation, SQL is actualy based on a different query language: relational caclulus. There are two important relational calculus is used in practice namingly 1. Tuple relational calculus and 2. Domain relational calculus. A typical TRC query looks the following:

- **{T/Condition}** - return all tuples T that satisfy the condition **Condition**.
- **{T/R(T)}** - returns all tuples T such that T is a tuple in relation R.
- **{T.name/FACULTY(T) AND T.DeptId = 'CS'}** – returns the values of name field of all faculty tuples with the value 'CS' in their department id field.
  - ⇒ T is the target – a variable that ranges over some relation (its values are tuples of the relation)
  - ⇒ Condition is the body of the query involving T and other variables and evaluates to true or false if a specific tuple is substituted for T.
  - ⇒ The result of a TRC query with respect to a given database is the set of all choices of tuples for the variable T that make the query condition a true statement about the database.
  - ⇒ The variable T is said to be free since it is not bound by a quantifier such as there exists (∃), for all (∀).
  - ⇒ Each variable T ranges over all possible tuples in the universe.
  - ⇒ Tuple variables are also called as range variables.

5.2.1 Relation Algebra vs. Relational Calculus

Although the relational algebra and calculus are equivalent in their expressive power the following important differences can be noted.

- Relational algebra provides a collection of explicit operations – join, union, projection, etc.
- The relational algebriac operations are used to tell the system **how** to build some desired relation in terms of other relations.

- The calculus merely provides a notation for formulating the **definition** of that desired relation in terms of those given relations.
- Relational Algebra is procedural; it is more like a programming language;
- Relational calculus is nonprocedural. it is more close to a natural language.
- The calculus formation is descriptive while the algebraic one is prescriptive.
- SQL does not require the explicit introduction of a tuple variable, it allows the relation name S to serve as an implicit tuple variable.

For example, suppose you want to query:
Get supplier numbers for suppliers who supply part P2.
An algebraic version of this query might follow these steps:

1. Form the natural join of relations SUP (supplier) and P (Part) on S#;
2. Next, restrict the result of that join to tuples for part P2;
3. Finally, project the result of that restriction on S#.

A calculus formulation might look like:
Get S# for suppliers such that there exists a shipment Part with the same S# value and with P# value P2.
That it,
{ t | ∃ s ε SUP(t[S#]=P[S#]) ∧ ∃ u ε P(u[P#]='P2') }

5.2.2 Why it is called relational calculus?

It is founded on a branch of mathematical logic called the predicate calculus.

5.2.3 Relationally Complete Language

A relational query language L is called as relational complete if we can express in L any query that can be expressed in relational calculus.

Formal Definition
Consider a simple relational calculus expression:
{T.name/FACULTY(T) AND T.DeptId='CS'}
which returns the values of name field of all faculty tuples with the value 'CS' in their department id field.
Informally, we need to specify the following information in a TRC expression, namingly:

1. For each tuple variable t, the range relation (for example in the above expression relation FACULTY is the range relation).

   Moreover, variables can be constrained by quantified statements to tuples in a single relation:

- **Esistential Quantifier.** ∃T ε R(cond) will succeed if Cond for atleast one tuple in T is true.
- **Universal Quantifier:** ∃T ε R(Cond) will succeed if Cond succeeds for all tuples in T.
- Any variable that is not bound by a quantifier is said to be free variable otherwise bound variable.
- A TRC expression may contain at most one free variable.

{T.name/FACULTY(T) AND T.DeptId = 'CS'} can be read as: " Find all tuples T such that T is a tuple in the FACULTY relation and the value of DeptId field is 'CS'. Return a tuple with a single field name which is equivalent to the name field of one such T tuple".
The same expression can be alternatively written as:
{R| ∃T ε FACULTY (T.DeptId = 'CS' AND R.name = T.name)} which can be read as " Find all tuples R such that there exists a tuple T in FACULTY with the DeptId field of R is equivalent to the name field of this tuple T}". The same can be alternatively stated as " Find all tuples R that can be obtained by copying the name field of SOME tuple in FACULTY with the value 'CS' in its Dept Id attribute.

Please note that if in the first query instead of T.name if we write only T then the query displays all attributes of the selected tuples.

2. A condition to select particular combinations of tuples. As tuple variables range over their respective range relations, the condition is evaluated for every possible combination of tuples to identify the selected combinations for which

the condition evaluates true. The conditions are also called as functions which are considered to be made of **atoms** (Details are given below).

3. A set of attributes to be retrieved, the requested attributes (in the above example T.name). The values of these attributes are retrieved for each selected combination of tuples.

In addition, the following examples may outline equivalence of TRC and SQL statement.
{T|TEACHING(T) AND T.semister = 'Fall2000'}
is equivalent to
SELECT *
FROM TEACHING T
WHERE T.semister = 'Fall2000';
Here,
Target T corresponds to SELECT list: the query result contains the entire tuple.
TEACHING(T) corresponds to FROM clause, i.e range or domain tuples.
T.semister='Fall2000' corresponds to WHERE clause which is condition.

Atoms

An atom has one one of the following forms

- s ε r where s is a tuple variable and r is a relation.
- s[x] **op** u[y], where s and u are tuple variables, x is an attribute on which s is defined, y is attribute on which u is defined, and op can be >, <, <=, >=, =, ≠.
- s[x] **op** c, where s[x] is as above and c is a constant and op is also same as above.

Formulas are created by joining atoms using the following rules.

- An atom is a formula.
- If P1 is a formula, then so are ¬P1 and (P1)
- If P1 and P2 are formulae then so are P1^P2, P1Vp2, and P1 ⇒ P2.
- If P1(s) is a formula containing a free tuple variable s and r is a relation then ∃ sεr (P1(s)) and ∀s ε r(P1(s)) are also formulae.
- P1^P2 is equivalent to ¬(¬P1, ∨ ¬ P2)
- ∀tε r (P1(t)) is equivalent to ¬ ∃t ε r (¬P1(t))
- P1⇒P2 is equivalent to ¬P1∨P2.

Safety of Expressions

Main drawback of TRC is it may generate an infinite relation. For example, a query like the following may generate infinitley many tuples which may not even appear in the database!.

{t | ¬ (t ε loan)}

Here we are trying to genrate the tuples which are not in the loan relation! Thus, domain of a tuple relational formula is used to solve this problem.

Important Points

- { T| STUDENT(T) AND FAACULTY(T)} will valuate to true if T is a tuple in both the relations. However, this is not possible since the schema of the two relations are different. Two tuples can never be identical.
- If we use attribute which is not available in the tuple then the result is NULL.
- {T|T.A>5} is unbounded expression which is not allowed. All tuple variables should be restricted to the tuples of a specific relation, even if they are not quantified.
- If a tuple variable T is bound to a relation R, then it only has values for the attributes in R. All other attribute values are null.
- A well formed query will have a single unbounded variable. All other variables will have a quantifier over them.
- Bound variables are used to make assertions about tuples in database (used in conditions).
- Free variables designate the tuples to be returned by the query.
- SQL has no quantifiers. Rather it uses some conventions such as:

  - Universal quantifiers are not allowed (but SQL 1999 introduced a limited form).
  - Makes esistential quantifiers implicit: any tuple variable that does not occur in SELECT is assumed to implicitly quantified with ∃.
- Adjacent existential quantifiers and adjacent universal quantifiers commute.
- Adjacent existential and universal quantifiers do not commute.
- A quantifier defines the scope of the quantified variable.
- Relational calculus comes from the first order predicate calculus.
- R(s), where R is a relation name and s is a tuple variable then this atom stands for assertion that s is a tuple in relation R.
- (∃s) (R(s)) says that relation R is not empty. That is, there exists a tuple s in R.
- A free variable is more like a global variable of high level programming languages, that is, a variable defined outside the current procedure. Where as "bound variable" is like a local variable, one that is defined in the procedure at hand and can not be referenced from the outside.
- Relational calculus based languages are higher-level than the algebraic languages. Calculus base languages leaves it to a compiler or interpreter to determine the most efficient order of evaluation.
- Parentheses may be placed around formulas as needed. We assume the order of evaluation of precedence is: arithmetic comparison operators highest, then the quantifiers, then NOT, ∨, ∧, in that order.
- {T| R(T) ∨ S(T)} makes sense if both relations R and S are of same arity.
- {T| R(T) ∧¬S(T)} indicates set difference R-S if both relations are of same arity.
- {T| (∃u) (R(T) ^ R(u) ^ (t[1] ≠ u[1] ≠ t[2] ≠ u[2])) } denotes R if R has two or more members and denotes the empty relation if R is empty or has only one member.
- If E is a relational algebraic expression then there is a safe expression in TRC equivalent to E.
- Let A (branch_name, loan_no, amount), B (cust_name, loan_no) are two relations. The following TRC query displays names of all customers who have loan from "Delhi" branch.

  {T|∃s ε B(T[cust_name] = s[cust_name]

  ∧∃ u ε A (u[loan_no] = s[loan_no] ∧ u[branch_name] = 'Delhi')) }
- The following displays names of faculty who belongs to CS department.

  {T|∃R ε FACULTY (R.DeptId = 'CS' ∧ T.name = R.name) }

  or

  {T.name| FACULTY(T) ∧ T.DeptId = 'CS' }
- The following TRC query may display name, social security number of those people who are staff and simultaneously students.

  {T|STUDENT(T) ∧∃ R ε FACULTY(T[SSN]=R[SSN] ∧ T[Name]=R[Name]) }
- Let A (cust_name, accountno), B (cust_name, loan_no) are two relations. The following TRC query displays names of all customers who have both loan and account

  {T|∃s ε B(T[cust_name]=s[cust_name] ∧ ∃ u ε A (T[cust_name]=u[cust_name])) }
- Let A (cust_name, account no), B (cust_name, loan_no) are two relations. The following TRC query displays names of all customers who have loan or account

  {T|∃s ε B(T[cust_name]=s[cust_name] ∧¬ u ε A (T[cust_name]=u[cust_name])) }
- Let A (cust_name, accountno), B (cust_name, loan_no) are two relations. The following TRC query displays names of all customers who have loan but not account

  {T|∃s ε B(T[cust_name]=s[cust_name] ∧¬∃ u ε A(T[cust_name]=u[cust_name])) }
- Let A(cust_name, accountno), B(cust_name, loan_no) are two relations. The following TRC query displays names of all customers who have account but no loan.

  {T|∃s ε A(T[cust_name]=s[cust_name] ∧¬∃ u ε B(T[cust_name]=u[cust_name])) }
- {T|∃F ε FACULTY(∃C ε CLASS(F.Id=C.Instructor.Id ∧ C.Year='2002' ∧ T.Name=F.Name ∧ T.Course_code=C. Course_Code))}

The above command displays faculty names, Course codes who taught in 2002.

- Write equivalent TRC queries for the following SQL statement :

```
SELECT DISTINCT F.Name
FROM FACULTY F
WHERE NOT EXISTS
  (SELECT * FROM CLASS C
  WHERE F.Id = C.InstructorId AND C.Year = '2002');
```

$\{F.Name|FACULTY(F) \wedge \neg(\exists C \varepsilon CLASS (F.Id=C.InstructorId \wedge C.Year='2002'))\}$

or

$\{F.Name|FACULTY(F) \wedge (\forall C \varepsilon CLASS (F.Id <> C.InstructorId \vee C.Year <> '2002')) \}$

- Find all students who have taken all the courses required by 'CSE432'.

  $\{S.Name \mid STUDENTS(S) \wedge \forall R \varepsilon REQUIRES(R.CrsCode <> 'CSC432' \vee (\exists T \varepsilon TRANSCRIPT(T.StudId=S.StudId \wedge T.CrsCode= R.PreReqCrsCode))\}$

- Find all students (names) who have never taken a course from 'Acorn'.

  $\{S.Name \mid STUDENTS(S) \wedge \forall C \varepsilon CLASS(\exists F \varepsilon FACULTY (F.Id=C.InstructorId \wedge (\neg(F.Name='Acorn' \vee \neg(\exists T \varepsilon TRANSCRIPT(S.Id=T.StudId \wedge C.CrsCode=T.CrsCode \wedge C.Year=T.Year))))) \}$

- Find all course tuples corresponding to all the courses that have been taken by all students.

  $\{E.Name \mid COURSE(E) \wedge \forall S \varepsilon STUDENT(\exists T \varepsilon TRANSCRIPT(T.StudId = S.StudId \wedge E.CrsCode= T. CrsCode)) \}$

- Find all students. Who has registered for course CS308

  $\{S \mid STUDENT(S) \wedge (\exists T \varepsilon TRANSCRIPT(S.Id = TStudId \wedge T.CrsCode= 'CS308')) \}$

5.2.4 Domain Relational Calculus

This is second form of relational calculus which uses domain variables that take on values from attributes domain, rather than values for an entire tuple. This is however, clisely related to TRC.

Formal Definition
An expression in DRC is expressed as

$\{<x1,x2,\ldots xn> | P(x1,x2,\ldots.xn)\}$

$<x1,x2,\ldots,xn>$ represented domain variables and P represents a formula composed of atoms, which are same as TRC.

Atoms
An atom has one one of the following forms

- $<x1,x2,\ldots xn> \varepsilon$ r where r is a relation on n attributes.
- s **op** u, where s and u are domain variables, and op can be >, <, <=, >=, =, ≠. We require that s and u have domains that can be compared by the above operations.
- s **op** c, where s is as domain variable and c is a constant and op is also same as above.

Formulas are created by joining atoms using the following rules.

- An atom is a formula.
- If P1 is a formula, then so are ¬P1 and (P1)
- If P1 and P2 are formulae then so are P1^P2, P1Vp2, and P1⇒ P2.

Find the branch name, loan number and amount for loans over 1300.

$\{<b,l,a> | <b,l,a> \varepsilon$ loan $\wedge$ a>1300$\}$

Find the names all customers who have a loan, an account or both at "Delhi" branch.

$\{<c> \mid \exists l (<c,l> \varepsilon$ borrower
$\wedge \exists b,a (<b,l,a> \varepsilon$ loan $\wedge$b="Delhi"))
$\vee \exists a (<c,a> \varepsilon$ depositor
$\wedge \exists b,n (<b,a,n> \varepsilon$ amount $\wedge$ b="Delhi"))$\}$

Tuple Relational Calculus

RA vs. TRC

- **Selection :**
  Algebra : $\sigma_{Cond}(R)$
  Calculus : $\{T | R(T)$ AND $Cond(T)\}$, i.e. replace attributes A in $Cond$ with $T.A$ to obtain $Cond(T)$.
- **Projection :**
  Algebra : $\Pi_{A_1,....,A_k}(R)$
  Calculus : $\{T | A_1,.... T.A_k | R(T)\}$
- **Cartesion Product:** Given $R(A_1,...., A_n)$ and $S(B_1,...., A_m)$
  Algebra : $R \times S$
  Calculus : $\{T | \exists T1 \in R, \exists T2 \in R ($
  $T.A_1 = T1.A_1$ AND $\cdots$ AND $T.A_n = T1.A_n$ AND
  $T.B_1 = T2.B_1$ AND $\cdots$ AND $T.B_m = T2.B_m)\}$
- **Union:**
  Algebra : $R \cup S$
  Calculus : $\{T | R(T)$ AND $S(T)$
- **Set Difference:**
  Algebra : $R - S$
  Calculus : $\{T | R(T)$ AND $\forall T1 \in S, (T1 <> T)\}$
  where $T <> T1$ is a shorthand for
  $T.A_1 <> T1.A_1$ OR $\cdots$ ORT. $A_n <> T1.A_n$

5.3 Integrity Constraints

- One row of a relation is called as a tuple
- Domain of an attribute is the set of values which it can take
- Domain Integrity Constraints : While developing the database system we can include a integrity constraint such that a specified attribute in a table will be made to accept either a set of range of values or a set of values.

  While we insert a new record, this constraint will be validated by the database system before accepting
- Attribute type Integrity Constraint : Normally while creating the tables every database management system supports freedom to specify the type of the attributes i.e, integer type or date type or string type etc.

  When we propagate (insertions) the database automatically enforces the type.
- Arity of a relation is the no of attributes.
- Cardinality means the no of tuples in that relation instance
- Candidate Key : Normally Candidate keys are the minimal subset of the attribute Set.

  For a relation, more than one candidate key can exist.

  One of the candidate keys is taken as Primary key. Then other keys become the Alternate keys for that relation. The attributes which are members of the candidate keys are called as Prime attributes. Primary Key is the one which is employed during the storage, retrieval of the Records

  If we happen to have more than one candidate key for a table, their selection is based on how they make the logical records physically distributed and what is its consequence on access times.

  * Super Key : is the one which may not be minimal set.

    If we add one attribute to a Candidate key or a primary key, the resulting set will be obviously a key and it is more appropriate to call it as super key.

    For a table if there exists at least one candidate key then, all the attributes of that table makes the largest possible super key.

If for a table there is no candidate key exist for a table then there will not be any super key also.

* Foreign Key :

Dept (Dept_id, Dept_name, Location)

10	Sales	Miami
20	Purchase	AKP

Emp (Empno, ename, emp_mgr, title, Emp_dept)

111	10
113	17
114	NULL

In the above tables, for the first table Dept_id is the primary key and Emp_dept is the foreign key for the second table.

If it is so, first tuple's insertion in the second Table is accepted as the last attribute value is in the domain of the Dept_id of the First table.

The insertion of the second tuple is not accepted as the attr value of employee dept 17 is not in the domain of Dept_id and is also not null.

The third tuples insertion is also accepted as the Emp_dept can take NULL.

This is known as Referential Integrity.

The foreign key can even exist for a single table also. For example, in the above Emp Table, Mgr_Id can be considered as a foreign key as the manager should also be an employee. Thus, its probable values (domain) will be same as the domain of the Emp_id.

5.4 Database Design and Normalisation

During the database design, we will be carrying out analysis of the tables which can be called as Normalisation.

The Normalisation is especially meant to eliminate the following anomalies,

(i) Insertion anomaly
(ii) Deletion anomaly
(iii) Update anomaly
(iv) Join anomaly

Even the objective of the normalisation includes elimination of redundancy in Database tables.

Redundancy may be one reason for some type of anomalies such as update anomalies.

Normalisation is even employed to impose some integrity constraints.

Goals of normalisation

1. Integrity
2. Maintainability

Side effects of normalisation

- Reduced storage space required (usually, but it could increase)
- Simpler queries (sometimes, but some could be more complex)
- Simpler updates (sometimes, but some could be more complex)

5.4.1 Functional Dependencies

If X, Y are two attribute sets, R is the relation then in this relation the FD

$X \rightarrow Y$ is said to be existing if for any two tuples t1, t2 if

T1 [x] = T2 [x] implies T1 [y] = T2 [y]

X functionally determines Y (or) Y is functionally dependent on X.

■ **Example** Emp (ID, Name, Dept, Grade, Sal, Age, Addr)

In the above table the functional dependency ID → Name is very well valid for this relation. However, Name → ID may not be a valid dependency, as there is a possibility that there can be two employees having same names but differ in ID no's.

* If R satisfies X → Y then PI Z (R) also satisfies X → → Y if X, Y subset of Z

■ **Example** Order (Order_no, part, supplier_name, Supplier_addr, Qty, Price)

Order_no Supplier_name → Supplier_addr

In the above table though the functional dependency is valid one. However, the supplier address is very much depends on Supplier name rather than Order no.

Thus, we can say Supplier_addr partially depends on the order no and Supplier Name

- If F is the set of functional dependencies meaningful in the relation and " f " is one FD from F then its lowercase " f " can be said as redundant FD if the set of FD's F – { f } implies the FD " f ".

■ **Example** In the FD set F given by X → Y

Y → Z

X → Z

X → Z is a redundant FD.

- The set of all FD's implied by " F " is called as Closure of F and is denoted as F^+
- If F^+ is same as F then F is called as full family of dependencies.

■ **Example** ID → Name
ID → Dept
ID Grade Age → Salary
ID → Age
ID → Address

{ ID } +
X = { ID }
X = { ID, Name }
X = { ID, Name, Dept }
X = { ID, Name, Dept, Age, Addr }
X = { ID, Name, Dept, Age, Addr, Grade, Salary }

- If X is a key for a database then X^+, contains all the attributes of that relation.

Superkey Rule 1. Any FD involving all attributes of a table defines a super-key on the LHS of the FD.

Given: Any FD containing all attributes in the table R(W, X, Y, Z) i.e., XY → WZ.
Proof:

(1) XY → WZ given
(2) XY → XY by the reflexivity axiom
(3) XY → XYWZ by the union axiom
(4) XY uniquely determines every attribute in table R, as shown in (3)
(5) XY uniquely defines table R, by the definition of a table as having no duplicate rows
(6) XY is therefore a super-key, by the definition of a super-key.

Super-key Rule 2. Any attribute that functionally determines a Super-key of a table, is also a super-key for that table.

Given: Attribute A is a super-key for table R(A, B, C, D, E) and E → A.
Proof:

(1) Attribute A uniquely defines each row in table R, by the def. of a super-key
(2) A → ABCDE by the definition of a super-key and a relational table
(3) E → A given
(4) E → ABCDE by the transitivity axiom
(5) E is a super-key for table R, by the definition of a super-key.

- Algorithm to find out whether a given FD X → Y is Valid in a given relation or not
  (i) Project X, Y from R
  (ii) Sort the tuples of the table using { X }
  (iii) Check every adjacent tuple X values and if they are same then check their Y values if they are matching continue else fail.
  O (n log n) for sorting
  O (n) for comparison where 'n' is the cardinality of the projected relation
  So, Time Complexity is O (n log n)

5.4.2 Armstrong's Axioms

(i) If Y subset of X then X → Y
(ii) If X → Y then XW → YW
(iii) If X → Y, Y→ Z then X → Z
(iv) If X → Y, YW → Z then XW → Z
(v) If X → Z, X → Y then X → YZ
(vi) If X → YZ then X → Y, X → Z

The FD of the first rule type is said to be trivial FD.

A FD X → Y is said to be redundant if the remaining FD's logically implies This FD X → Y.

One way to find that the FD is redundant or not is to prove that by using all of the inference axioms and the remaining FD's, this FD is implied.

The Set of inference axioms are complete and sound. Here the Complete indicates they can be used to enumerate all the possible acceptable FD's. Here the Sound indicates they will not produce or derive non-acceptable Functional dependencies.

■ **Example** F = { X → YW, XW → Z, Z → Y, XY → Z }
Find whether XY → Z is redundant or not ?

G = { X → YW, XW → Z, Z → Y }
T1 = { XY }
T2 = { XYW }
T3 = { XYWZ }

As the dependent of XY i.e, Z belongs to T3.
So, XY → Z is redundant.

- If G = Phi (empty) then we can conclude that XY → Z is not redundant.
- If we go on remove the redundant FD from a FD set till we cannot remove any more FD, the remaining set of FD's is called as Non- Redundant Cover of that relation.
- Though the Cover is Non-Redundant, the FD's may have some extraneous attributes either left side or right side. Removing these extraneous attributes. Also is very much needed. This operation is called as left Reduction, right Reduction, respectively.
- Let F is a Set of FD, XY → W is one FD, then X variable can be said as Extraneous if F – { XY → W } + { Y → W } explains the same cover.
- In order to check X → Y is redundant or not the following steps has to be taken,
  (i) Have G = F – { X → Y }
  (ii) T = { X }
  (iii) For each FD A → B in G do the following
  If A is subset of T T = T union { B }
  If Y is subset of T then X → Y can be said as redundant
  Else Remove A→ B from G.

(iv) if G is NULL then X→ Y can be said as redundant.

* If Covers before and after removal of an attribute X on the left side of the FD are same then X can be said as extraneous otherwise not.

■ **Example** Find Non – Redundant Cover of

$F = \{ X \to Y, Y \to Z, Z \to Y, X \to Z, Z \to X \}$

Take $X \to Y$:

$G = \{ Y \to X, Y \to Z, Z \to Y, X \to Z, Z \to X \}$

As $X^+ = \{ XYZ \}$ contains Y so, $X \to Y$ is redundant.

Take $Y \to X$:

$G = \{ Y \to Z, Z \to Y, X \to Z, Z \to X \}$

As Y+ = { XYZ } contain X so, $Y \to X$ is redundant.

Take $Y \to Z$:

$G = \{ Z \to Y, X \to Z, Z \to X \}$

As Y+ = {Y} do not contain Z. so, $Y \to Z$ is not redundant.

Similar procedure clearly confirms that $Z \to Y$, $X \to Z$, $Z \to X$ are not redundant.

Final FD set = $\{ Y \to Z, Z \to Y, X \to Z, Z \to X \}$

■ **Example** $F = \{X \to Z, XY \to WP, XY \to ZWQ, XZ \to R\}$

Selecting $XY \to WP$:

Choose X as extraneous.

Y+ = { Y }, WP not subset of Y+. So, X is not extraneous.

Choose Y as extraneous.

X+ = { XZR }, WP not subset of X +, Y is also not extraneous.

Selecting $XY \to ZWQ$:

Similarly, X, Y are not extraneous.

Selecting $XZ \to R$:

X is not extraneous and Z is extraneous.

So, left reduced final set = $\{ X \to Z, XY \to WP, XY \to ZWQ, X \to R \}$

- Minimal Cover or Canonical Cover : is the one in which,

(i) Every FD is simple (RHS of any FD should have single attribute)

(ii) It is left reduced

(iii) It is non-redundant

■ **Example** Find the minimal cover for the following set of functional dependencies.

$A \to B\ C$

$AC \to D$

$D \to A\ B$

$A\ B \to D$

In $AB \to D$, B is extraneous

In $AC \to D$, C is extraneous

$A \to B\ C$, $A \to D$, $D \to A\ B$ are left reduced.

$\{A \to D, D \to B, A \to C, D \to A\}$ is the minimal cover.

{A }, {D } are the candidate keys.

In the above set of functional dependencies A,D are prime attributes.

■ **Example** $A \to B\ C$, $B \to C$, $A \to B$, $A\ B \to C$. Find the minimal cover ?

In $A\ B \to C$, B is extraneous.

$\{ A \to B, B \to C \}$ is the minimal cover.

{ A } is the primary key.

* Normalisation : is especially aimed at to make the relations to be free from undesirable anamolies such as insertion anamoly or updation or deletion anamoly.

Normalisation is to obtain powerful relational retrieval algorithms which are based on a collection of relational primitive operators.

Normalisation is also to reduce the need for restructuring of the relations as new datatypes are added.

- If a table contains entries with multi values then the table is said to be unnormalised.
- Normally, such a type of table can be normalized by flattening the table i.e, for each value of an entry which is having multiple values we create a new tuple by simply copying the other attribute values as same.
- We can bring an unnormalised table with entries multiple values to 1 NF by either flattening the table or by decomposing the table.

A table is said to be in 2 NF if

(i) it is in 1 NF

(ii) no non-prime attribute is partially dependent on key or each non-prime attribute should fully dependent on every candidate key.

- A table is said to be in 3 NF if

(i) The relation should be in 2 NF

(ii) No non-prime attribute functionally determines any other non-prime attributes

- Boyce-Codd normal form (BCNF)

A table is in BCNF if, for every nontrivial FD X → A,

(1) attribute X is a super-key.

TABLE PART

PNUM	PNAME	WT
P1	NUT	12
P2	BOLT	17
P3	WRENCH	17
P4	WRENCH	24
P5	CLAMP	12
P6	LEVEL	19

We find the following FD as valid in this relation.

PNUM → PNAME, WT

Also, PNUM is key. Therefore, it is in 3NF.

■ **Example**

Table P1. R (X, Y, Z) is decomposed into R1 (X, Y) R2 (Y, Z)

F D = { X → Y, Z → Y }

	X	Y	Z
R1	A1	A2	B13
R2	B21	A2	A3

This is a lossy decomposition.

- If at all any row contains all A's then it is lossless decomposition.

R (X, Y, Z, W, P, Q) is decomposed into R1 (Z,P,Q) R2 (X, Y, W, P, Q).

FD = { XY → W, XW → P, PQ → Z, XY → Q }

	X	Y	Z	W	P	Q
R1	B1	B12	A3	B14	A5	A6
R2	A1	A2	A3	A4	A5	A6

This is Lossless decomposition.

- This algorithm is an iterative one and you proceed till there is no change in the table and it does not matter with the order of the FD's taken, result is same.

- While carrying out 2 NF Normalisation, we have to first of all find out all possible Candidate keys (Prime attrs) and Non-prime attrs. Also, we have to find out whether there are any partial dependencies of the non-prime attrs. Then, we decompose this table such that all fully dependent attrs along with Prime attrs into one table and partially dependent Non-Prime attrs, the attrs on which they depend into another table. This guarantees 2 NF requirements However, transitive dependencies may exist.
- ABU algorithm is very much suitable to identify whether the decomposition is Lossless or not. Especially, if finally there is no row having all a's in the above tables then we can say that the decomposition is lossy. The reverse is not strong, rather we have to apply extra data dependency analysis.

1. R (A, B, C) FD = { A → B, B → C }
A is the key for the database and all the non-prime attrs depends fully on A.
Thus, it is in 2 NF.
However, A → C is a transitive dependency. Thus, it cannot be in 3 NF.

2. R (X, Y, Z,W) FD = { Y → W, W → Y, X Y → Z }
XY is a Key.
XW is a Key.
So, XYW are Prime attributes.
Z is non- prime attribute
The relation is in 3 NF.

3. R (A,B,C, D, E) FD = { AB → CE, E → AB, C → D }
Keys : { AB }, {E }
This relation is in 2 NF but not 3 NF.

4. Emp (Id, Name, Dept, Hrly_rate)
Dept → Hrly_rate
Id → Dept
Id → Name
This relation is in 1 NF but not in 2 NF, not in 3 NF.

- Every 2 attribute relation is in BCNF.

5. Emp (ID, Name, Dept, Hrly_rate)
ID → Dept, Dept → Hrly_rate
This relation is in 2 NF.

6. Order (SSN, PNO, HRS, ENAME, PNAME, PLOC)
SSN PNO → HRS
SSN → ENAME
PNO → PNAME
PNO → PLOC
Key is (SSN, PNO)
HRS is fully dependent on SSN. ENAME is partially dependent on SSN.
So, this is not in 2 NF.

- Inorder to bring a table to 2 NF, we can decompose the table such that in one table all prime attrs and in another table non-prime attrs and the prime attrs on which they depend.

■ **Example** 2NF decomposition of the above relation is given by
(SSN, PNO, HRS)
(SSN, ENAME)
(PNO, PNAME, PLOC)

7. Emp (SSN, ENAME, BDATE, ADDR, DNO, DNAME, DMGRSSN)
DNO → DNAME
DNO → DMGRSSN

- In order to get the table into 3 NF, we can decompose such that in one table key, all those attrs which fully depends on a key are kept and in other table, we may have the attrs which depends transitively on the key and the attributes via which transitive dependency exists.

■ **Example** The 3 NF decomposition of the above relation is given by

(SSN, ENAME, BDATE, ADDR, DNO)

(DNO, DNAME, DMGRSSN)

- Projection of a set of FD's on to a set of attributes is defined as those set of FD's which are valid with this selected set of attrs.

Let F be projection onto a set of attributes T denoted as PI_T (F)

= { X → Y belongs to F+ / XY subset of T }

Algorithm:

X → Y

X, Y can be composite attributes whose union should be proper subset of T.

To calculate Projection Consider all proper subsets X of T (X subset of T, X !=T)

That appear as the determinant of FD's.

For each Set

1. Calculate X +.
2. For each set of attributes of Y of X+ that satisfies simultaneously the following conditions

A. Y subset of T

b. Y subset of X+

c. Y not equal to T

include X → Y as one of the FD's of Projection of FD's on T.

■ **Example** R (X, Y, Z, W, Q)

FD = { XY → WQ, Z → Q, W → Z, Q → X }

T = { X, Y, Z }

{X }, { Y }, {Z }, {XY }, {XZ }, { YZ }

{X}+ = {X }

{Y}+ = { Y }

{ Z }+ = { ZQX }

so, Z → X is implied.

{ XY } + = { XYZWQ }

so, XY → Z is acceptable

{ YZ } + = { YZWQX }

so, YZ → X is implied

{ XZ } + = { XZQ }

The projected FD's = { XY→ Z, Z → X }

- If a constraint on a relation R states that there cannot be more than one tuple with A given X value then X can be called as a Candidate key.
- If a Primary Key contains a single attribute then the relation can evidently be in 2 NF.

1. R (A, B, C, D)

FD = { AB → C, BC → D }

{AB } is the key.

The largest acceptable normal state is 2 NF.

2. R (A, B, C)

A → B, B → C What are the anomalies you will have?

There is no guarantee always that when we bring a table from 1 NF to 2 NF all the anomalies are eliminated. Only redundancy is reduced.

Under some special conditions in 3 NF also we may find insertion, updation anomalies etc.

■ **Example** Given a set of FDs H, determine a minimal set of tables in 3NF, while preserving all FDs and maintaining only lossless decomposition/joins.

H:	AB →C	DM→NP	D→KL
	A→DEFG	D→M	
	E→G	L→D	
	F→DJ	PR→S	
	G→DI	PQR→ST	

Step 1: Eliminate any extraneous attributes in the left hand sides of the FDs. We want to reduce the left hand sides of as many FDs as possible.

In general: XY→Z and X→Z ⇒ Y is extraneous **(Reduction Rule 1)**

XYZ→W and X→Y ⇒ Y is extraneous **(Reduction Rule 2)**

For this example we mix left side reduction with the union and decomposition axioms:

DM → NP ⇒ D → NP ⇒ D → MNP

D → M D → M

PQR → ST ⇒ PQR → S, PQR → T ⇒ PQR → .T

PR → S PR → S PR → S

Step 2: Find a non-redundant cover H' of H, i.e. eliminate any FD derivable from others in H using the inference rules (most frequently the transitivity axiom).

A → E → G ⇒ eliminate A → G from the cover

A → F → D ⇒ eliminate A → D from the cover

Step 3: Partition H' into tables such that all FDs with the same left side are in one table, thus eliminating any non-fully functional FDs. (Note: creating tables at this point would be a feasible solution for 3NF, but not necessarily minimal.)

R1: AB → C	R4: G → DI	R7: L → D
R2: A → EF	R5: F → DJ	R8: PQR → T
R3: E → G	R6: D → KLMNP	R9: PR → S

Step 4: Merge equivalent keys, i.e., merge tables where all FD's satisfy 3NF.

4.1 Write out the closure of all LHS attributes resulting from Step 3, based on transitivities.

4.2 Using the closures, find tables that are subsets of other groups and try to merge them. Use Rule 1 and Rule 2 to establish if the merge will result in FDs with super-keys on the LHS. If not, try using the axioms to modify the FDs to fit the definition of super-keys.

4.3 After the subsets are exhausted, look for any overlaps among tables and apply Rules 1 and 2 (and the axioms) again.

In this example, note that R7 (L→D) has a subset of the attributes of R6 (D→KLMNP). Therefore, we merge to a single table with FDs D →KLMNP, L→D because it satisfies 3NF: D is a super-key by Rule 1 and L is a super-key by Rule 2.

Final 3NF (and BCNF) table attributes, FDs, and candidate keys:

R1: ABC (AB → C with key AB)	R5:DFJ (F → DJ with key F)
R2: AEF (A → EF with key A)	R6: DKLMNP (D → KLMNP, L → D, w/keys D, L)
R3: EG (E → G with key E)	R7: PQRT (PQR → T with key PQR)
R4: DGI (G → DI with key G)	R8: PRS (PR → S with key PR)

Step 4a. Check to see whether all tables are also BCNF. For any table that is not BCNF, add the appropriate partially redundant table to eliminate the delete anomaly.

5.5 Transactions and Concurrency Control

5.5.1 What is a Transaction?

- A transaction is a logical unit of work –

  It may consist of a simple SELECT to generate a list of table contents, or a series of related UPDATE command sequences.

  A **database request** is the equivalent of a single SQL statement in an application program or transaction.

  - Must be either entirely completed or aborted –

    To sell a product to a customer, the transaction includes updating the inventory by subtracting the number of units sold from the PRODUCT table's available quantity on hand, and updating the accounts receivable table in order to bill the customer later.

  - No intermediate states are acceptable –

    Updating only the inventory or only the accounts receivable is not acceptable.

- **Example Transaction –**

  Consider a transaction that updates a database table by subtracting 10 (units sold) from an already stored value of 40 (units in stock), which leaves 30 (units of stock in inventory).

  - A **consistent database state** is one in which all data integrity constraints are satisfied.
  - At this transaction is taking place, the DBMS must ensure that no other transaction access X.

  Evaluating Transaction Results

- Examine current account balance

```
SELECT ACC_NUM, ACC_BALANCE
FROM CHECKACC
WHERE ACC_NUM = '0908110638';
```

  - SQL code represents a transaction because of accessing the database
  - Consistent state after transaction
  - No changes made to Database

- Register credit sale of 100 units of product X to customer Y for $500:

  Reducing product X's quality on and (QOH) by 100

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH - 100
WHERE PROD_CODE = 'X';
```

  Adding $500 to customer Y's accounts receivable

```
UPDATE ACCT_RECEIVABLE
SET ACCT_BALANCE = ACCT_BALANCE + 500
WHERE ACCT_NUM = 'Y';
```

  - If both transactions are not completely executed, the transaction yields an inconsistent database.
  - Consistent state only if both transactions are fully completed
  - DBMS doesn't guarantee transaction represents real-world event but it must be able to recover the database to a previous consistent state. (For instance, the accountant inputs a wrong amount.)

5.5.2 Transaction Properties

All transactions must display **atomicity**, **durability**, **serialisability**, and **isolation**.

Atomicity

The Atomicity property of a transaction implies that it will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database or the database has been changed in a consistent manner.

The basic idea behind ensuring atomicity is as follows. The database keeps a track of the old values of any database on which a transaction performs a write, and if the transaction does not complete its execution, the old values are restored to make it appear as though the transaction never executed.

Ensuring atomicity is the responsibility of the database system itself; it is handled by a component called the **Transaction Management Component.**

Consistency

The consistency property of a transaction implies that if the database was in a consistent state before the start of a transaction, then on termination of the transaction, the database will also be in a consistent state.

Ensuring consistency for an individual transaction is the responsibility of the Application Manager who codes the transaction.

Isolation

The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in same order.

Thus, in a way it means that the actions performed by a transaction will be isolated or hidden from outside the transaction until the transaction terminates.

This property gives the transaction a measure of relative independence.

Ensuring the isolation property is the responsibility of a component of a database system called the Concurrency Control Component.

Durability

The durability property guarantees that, once a transaction completes successfully, all the updates that it carried out on the database persists even if there is a system failure after the transaction completes execution.

Durability can be guaranteed by ensuring that either:

(a) The updates carried out by the transaction have been written to the disk before the transaction completes.

(b) Information about updates carried out by the transaction and written to the disk is sufficient to enable the database to re-construct the updates when the database system is restored after the failure.

Ensuring durability is the responsibility of the component of the DBMS called the Recovery Management Component.

In a nutshell,:

- Atomicity –
  - All transaction operations must be completed

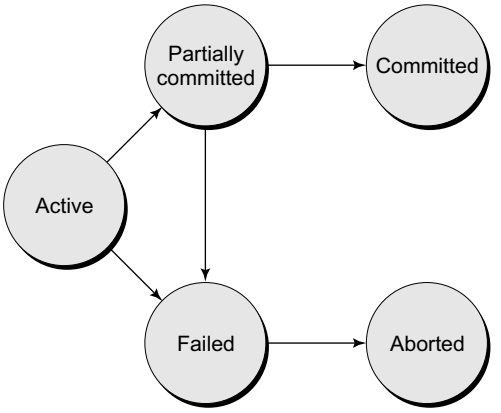
Incomplete transactions aborted

- Durability –
  - Permanence of consistent database state
- Serialisability –
  - Conducts transactions in serial order
  - Important in multi-user and distributed databases
- Isolation
  - Transaction data cannot be reused until its execution complete
- Consistency – (To preserve integrity of data, the database system must ensure: **atomicity, consistency, isolation, and durability (ACID).)**

Execution of a transaction in isolation preserves the consistency of the database.

- A single-user database system automatically ensures serialisability and isolation of the database because only one transaction is executed at a time.
- The atomicity and durability of transactions must be guaranteed by the single-user DBMS.

- The multi-user DBMS must implement controls to ensure serialisability and isolation of transactions – in addition to atomicity and durability – in order to guard the database's consistency and integrity.



5.5.3 Transaction State

- **Active**, the initial state; the transaction stays in this state while it is executing.
- **Partially committed,** after the final statement has been executed.
- **Failed,** after the discovery that normal execution can no longer proceed.
- **Aborted**, after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - **Restart** the transaction – only if no internal logical error but hardware or software failure.
  - **Kill** the transaction – once internal logical error occurs like incorrect data input.
- **Committed**, after successful completion. The transaction is terminated once it is aborted or committed.

5.5.4 Transaction Management with SQL

- Defined by ANSI, the standards of SQL transaction support: `COMMIT` & `ROLLBACK`
- User initiated transaction sequence must continue until one of following four events occurs:

1. `COMMIT` statement is reached—all changes are permanently recorded within the database.
2. `ROLLBACK` statement is reached—all the changes are aborted and the database is rolled back to its previous consistent state.
3. End of a program reached—all changes are permanently recorded within the database.
4. Program reaches abnormal termination—the changes made in the database are aborted and the database is rolled back to its previous consistent state.

For example:

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH - 100
WHERE PROD_CODE = '345TYX';
UPDATE ACCT_RECEIVABLE
SET ACCT_BALANCE = ACCT_BALANCE + 3500
WHERE ACCT_NUM = '60120010';
COMMIT;
```

In fact, the COMMINT statement used in this example is not necessary if the UPDATE statement is the application's last action and the application terminates normally.

Transaction Log

- The DBMS use transaction log to track all transactions that update database.
  - May be used by ROLLBACK command for triggering recovery requirement.
  - May be used to recover from system failure like network discrepancy or disk crash.
  - While DBMS executes transactions that modify the database, it also updates the transaction log. The log stores:
    - Record for beginning of transaction
    - Each SQL statement
      - The type of operation being performed (update, delete, insert).
      - The names of objects affected by the transaction (the name of the table).
      - The "before" and "after" values for updated fields
      - Pointers to previous and next entries for the same transaction.
    - Commit Statement – the ending of the transaction.

Note

Committed transactions are not rolled back.

1. If a system failure occurs, the DBMS will examine the transaction log for all uncommitted or incomplete transactions, and it will restore (ROLLBACK) the database to its previous state on the basis of this information.
2. If a ROLLBACK is issued before the termination of a transaction, the DBMS will restore the database only for that particular transaction, rather than for all transactions, in order to maintain the durability of the previous transactions.

Concurrency Control

- Coordinates simultaneous transaction execution in multiprocessing database.
- Ensure serialisability of transactions in multiuser database environment.
- Potential problems in multiuser environments.
- Three main problems: **lost updates**, **uncommitted data**, and **inconsistent retrievals**.

Lost updates

- Assume that two concurrent transactions (T1, T2) occur in a PRODUCT table which records a product's quantity on hand (PROD_QOH). The transactions are:

Transaction	Computation
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH - 30

Normal Execution of Two Transactions

Note

This table shows the serial execution of these transactions under normal circumstances, yielding the correct answer, PROD_QOH=105.

Lost Updates

Note

The addition of 100 units is "lost" during the process.

1. Suppose that a transaction is able to read a product's PROD_QOH value from the table before a previous transaction has been committed.
2. The first transaction (T1) has not yet been committed when the second transaction (T2) is executed.
3. T2 sill operates on the value 35, and its subtraction yields 5 in memory.
4. T1 writes the value 135 to disk, which is promptly overwritten by T2.

Uncommitted Data

- When two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data – thus violating the isolation property of transactions. The transactions are:

Transaction	Computation
T1: Purchase 100 units **(Rollback)**	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH - 30

Correct Execution of Two Transactions

Note

The serial execution of these transactions yields the correct answer.

An Uncommitted Data Problem

Note

The uncommitted data problem can arise when the ROLLBACK is completed after T2 has begun its execution.

Inconsistent Retrievals

- When a transaction calculates some summary (aggregate) functions over a set of data while other transactions are updating the data.
- The transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.
  1. T1 calculates the total quantity on hand of the products stored in the PRODUCT table.
  2. T2 updates PROD_QOH for two of the PRODUCT table's products.

Retrieval During Update

Note

T1 calculates PROD_QOH but T2 represents the correction of a typing error, the user added 30 units to product 345TYX's PROD_QOH, but meant to add the 30 units to product '123TYZ's PROD_QOH. To correct the problem, the user executes 30 from product 345TYX's PROD_QOH and adds 30 to product 125TYZ's PROD_QOH.

Transaction Results: Data Entry Correction

Note

The initial and final PROD_QOH values while T2 makes the correction—same results but different transaction process.

Transaction Result: Data Entry Correction

Note

- The transaction table demonstrates that inconsistent retrievals are possible during the transaction execution, making the result of T1's execution incorrect.
- Unless the DBMS exercises concurrency control, a multi-user database environment can create chaos within the information system.

5.5.5 The Scheduler – Schedule, Serialisability, Recovery, Isolation

- Previous examples executed the operations within a transaction in an arbitrary order:
  - As long as two transactions, T1 and T2, access unrelated data, there is no conflict, and the order of execution is irrelevant to the final outcome.
  - If the transactions operate on related (or the same) data, conflict is possible among the transaction components, and the selection of one operational order over another may have some undesirable consequences.
- Establishes order of concurrent transaction execution.
- Interleaves execution of database operations to ensure serialisability.
- Bases actions on concurrency control algorithms
  - Locking
  - Time stamping
  - Ensures efficient use of computer's CPU
  - First-come-first-served basis (FCFS) – executed for all transactions if no way to schedule the execution of transactions.
  - Within multi-user DBMS environment, FCFS scheduling tends to yield unacceptable response times.
  - READ and/or WRITE actions that can produce conflicts.

Read/Write Conflict Scenarios: Conflicting Database Operations Matrix

Note

The table below show the possible conflict scenarios if two transactions, T1 and T2, are executed concurrently over the same data.

- **Schedules** – sequences that indicate the chronological order in which instructions of concurrent transactions are executed
  - a schedule for a set of transactions must consist of all instructions of those transactions
  - must preserve the order in which the instructions appear in each individual transaction.
  - Example of schedules (refer right figures 5.7–5.9)
  - Schedule 1 (see Figure 5.7): Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B. The following is a serial schedule, in which T_1 is followed by T_2.
    - Schedule 2 (see Figure 5.8): Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is **equivalent** to Schedule 1.

T_1	T_2
read(A) A := A – 50 write (A) read(B) B := B + 50 write(B)	read(A) temp := A * 0.1 A := A – *temp* write(A) read(B) B := B + *temp* write(B)

Figure 5.7 A sample schedule

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	
	read(B) $B := B + temp$ write(B)

Figure 5.8 Concurrent schedule of schedule in Figure 5.7

- Schedule 3 (see Figure 5.9): The following concurrent schedule does not preserve the value of the sum A + B.

T_1	T_2
read(A) $A := A - 50$	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B)
write(A) read(B) $B := B + 50$ write(B)	
	$B := B + temp$ write(B)

Figure 5.9 Another Concurrent schudule of schedule in Figure 5.7

5.5.5.1 Serialisability

A (possibly concurrent) schedule is serialisable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:

1. conflict serialisability
2. view serialisability

■ Conflict Serialisability: Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.

1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict.
2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
3. I_i = **write**(Q), I_j = **read**(Q). They conflict
4. I_i = **write**(Q), I_j = **write**(Q). They conflict

- If a schedule S can be transformed into a schedule S′ by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**.
- We say that a schedule S is **conflict serialisable** if it is conflict equivalent to a serial schedule.

■ View Serialisability: Let S and S′ be two schedules with the same set of transactions. S and S ′ are view equivalent if the following three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
2. For each data item Q if transaction T_i executes **read** (Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i must in schedule S' also read the value of Q that was produced by transaction T_j.

3. For each data item Q, the transaction (if any) that performs the final **write**(Q) operation in schedule S must perform the final **write**(Q) operation in schedule S′.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- As can be seen, view equivalence is also based purely on reads and writes alone.
- A schedule S is view serialisable if it is view equivalent to a serial schedule.
- Every conflict serialisable schedule is also view serialisable.

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$B := A - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Figure 5.10 View Serializable schedule of schedule in Figure 5.7

- Schedule in Figure 5.10 – a schedule which is view-serialisable but not conflict serialisable.
- Every view serialisable schedule that is not conflict serialisable has blind writes.

■ Other Notions of Serialisability

- Schedule in Figure 5.10 produces same outcome as the serial schedule < T_1, T_5 >, yet is not conflict equivalent or view equivalent to it.
- Determining such equivalence requires analysis of operations other than read and write.

5.5.5.2 Recoverability

Need to address the effect of transaction failures on concurrently running transactions

■ *Recoverable schedule* – if a transaction T_j reads a data items previously written by a transaction T_i, the commit operation of Ti appears before the commit operation of T_j.

- The schedule in Figure 5.11 is not recoverable if T_9 commits immediately after the read.

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Figure 5.11 An example irrecoverable schedule

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence database must ensure that schedules are recoverable.

■ *Cascading rollback* – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)
- If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work.

■ *Cascadeless schedules*– cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless

➢ Implementation of Isolation

■ Schedules must be conflict or view serialisable, and recoverable, for the sake of database consistency, and preferably cascadeless.

■ A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency.

■ Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur.

■ Some schemes allow only conflict-serialisable schedules to be generated, while others allow view-serialisable schedules that are not conflict-serialisable.

5.5.6 Concurrency Control with Locking Methods

- Lock guarantees current transaction exclusive use of data item, i.e., transaction T_2 does not have access to a data item that is currently being used by transaction T_1.
- Acquires lock prior to access.
- Lock released when transaction is completed.
- DBMS automatically initiates and enforces locking procedures.
- All lock information is managed by lock manager.

Lock Granularity

- Lock granularity indicates level of lock use: database, table, page, row, or field (attribute).

Database-Level

- The entire database is locked.
- Transaction T2 is prevented to use any tables in the database while T1 is being executed.
- Good for batch processes, but unsuitable for online multi-user DBMSs.
- Figure 5.7 transactions T1 and T2 cannot access the same database concurrently, even if they use different tables. (The access is very slow!)

Table-Level

- The entire table is locked. If a transaction requires access to several tables, each table may be locked.
- Transaction T2 is prevented to use any row in the table while T1 is being executed.
- Two transactions can access the same database as long as they access different tables.
- It causes traffic jams when many transactions are waiting to access the same table.
- Table-level locks are not suitable for multi-user DBMSs.
- Figure 5.8 transaction T1 and T2 cannot access the same table even if they try to use different rows; T2 must wait until T1 unlocks the table.

Page-Level

- The DBMS will lock an entire diskpage (or page), which is the equivalent of a diskblock as a (referenced) section of a disk.

- A page has a fixed size and a table can span several pages while a page can contain several rows of one or more tables.
- Page-level lock is currently the most frequently used multi-user DBMS locking method.

  Shows that T1 and T2 access the same table while locking different diskpages.
- T2 must wait for using a locked page which locates a row, if T1 is using it.

Row-Level

- With less restriction respect to previous discussion, it allows concurrent transactions to access different rows of the same table even if the rows are located on the same page.
- It improves the availability of data, but requires high overhead cost for management.

  For row-level lock.

Field-Level

- It allows concurrent transactions to access the same row, as long as they require the use of different fields (attributes) within a row.
- The most flexible multi-user data access, but cost extremely high level of computer overhead.

Lock Types

- The DBMS may use different lock types: binary or shared/exclusive locks.
- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Binary Locks

- Two states: locked (1) or unlocked (0).
- Locked objects are unavailable to other objects.
- Unlocked objects are open to any transaction.
- Transaction unlocks object when complete.
- Every transaction requires a lock and unlock operation for each data item that is accessed.

Example of Binary Lock Table

Note

The lock and unlock features eliminate the lost update problem encountered. However, binary locks are now considered too restrictive to yield optimal concurrency conditions.

Shared/Exclusive Locks

- Shared (S Mode)
  - Exists when concurrent transactions granted READ access
  - Produces no conflict for read-only transactions
  - Issued when transaction wants to read and exclusive lock not held on item
- Exclusive (X Mode)
  - Exists when access reserved for locking transaction
  - Used when potential for conflict exists.

	S	X
S	true	false
X	false	false

  - Issued when transaction wants to update unlocked data
- Lock-compatibility matrix
  - A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
  - Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

➢ Reasons to increasing manager's overhead

- The type of lock held must be known before a lock can be granted
- Three lock operations exist: READ_LOCK (to check the type of lock), WRITE_LOCK (to issue the lock), and UNLOCK (to release the lock).
- The schema has been enhanced to allow a lock upgrade (from shared to exclusive) and a lock downgrade (from exclusive to shared).

➢ Problems with Locking

- Transaction schedule may not be serialisable
  - Managed through two-phase locking
- Schedule may create deadlocks
  - Managed by using deadlock detection and prevention techniques

5.5.6.1 Two-Phase Locking

Two-phase locking defines how transactions acquire and relinquish (or revoke) locks.

1. Growing phase – acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. Shrinking phase – releases all locks and cannot obtain any new lock.

➢ Governing rules

- Two transactions cannot have conflicting locks
- No unlock operation can precede a lock operation in the same transaction
- No data are affected until all locks are obtained

➢ In the example for two-phase locking protocol the transaction acquires all the locks it needs (two locks are required) until it reaches its locked point.

- When the locked point is reached, the data are modified to conform to the transaction requirements.
- The transaction is completed as it released all of the locks it acquired in the first phase.

➢ Updates for two-phase locking protocols:

- Two-phase locking does not ensure freedom from deadlocks.
- Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called strict two-phase locking. Here a transaction must hold all its exclusive locks till it commits/aborts.
- Rigorous two-phase locking is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.
- There can be conflict serialisable schedules that cannot be obtained if two-phase locking is used.
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict Serialisability in the following sense:

  Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serialisable.

5.5.6.2 Deadlocks

Occurs when two transactions wait for each other to unlock data. For example:

T1 = access data items X and Y

T2 = access data items Y and X

➢ Deadly embrace – if T_1 has not unlocked data item Y, T_2 cannot begin; if T_2 has not unlocked data item X, T_1 cannot continue.

➢ Starvation is also possible if concurrency control manager is badly designed.

- For example, a transaction may be waiting for an X-lock (exclusive mode) on an item, while a sequence of other transactions request and are granted an S-lock (shared mode) on the same item.
- The same transaction is repeatedly rolled back due to deadlocks.

➢ Control techniques

- Deadlock prevention – a transaction requesting a new lock is aborted if there is the possibility that a deadlock can occur.
- If the transaction is aborted, all the changes made by this transaction are rolled back, and all locks obtained by the transaction are released.
- It works because it avoids the conditions that lead to deadlocking.
- Deadlock detection – the DBMS periodically tests the database for deadlocks.
- If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and restarted), and the other transaction continues.
- Deadlock avoidance – the transaction must obtain all the locks it needs before it can be executed.
- The technique avoids rollback of conflicting transactions by requiring that locks be obtained in succession.
- The serial lock assignment required in deadlock avoidance increase action response times.

➢ Control Choices

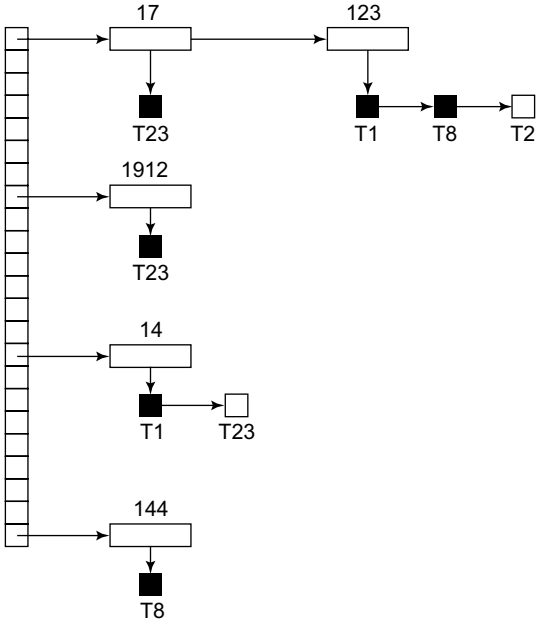
- If the probability of deadlocks is low, deadlock detection is recommended.
- If the probability of deadlocks is high, deadlock prevention is recommended.
- If response time is not high on the system priority list, deadlock avoidance might be employed.

Implementation of Locking

- A Lock manager can be implemented as a separate process to which transactions send lock and unlock requests.
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock).
- The requesting transaction waits until its request is answered.
- The lock manager maintains a datastructure called a lock table to record granted locks and pending requests.
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked.

➢ Lock Table

- Black rectangles indicate granted locks, white ones indicate waiting requests.



- Lock table also records the type of lock granted or requested.
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks.
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted.
- If transaction aborts, all waiting or granted requests of the transaction are deleted.
  - Lock manager may keep a list of locks held by each transaction, to implement this efficiently.

5.5.7 Concurrency Control with Time Stamping Methods

- Assigns global unique time stamp to each transaction
- Produces order for transaction submission
- Properties
- Uniqueness: ensures that no equal time stamp values can exist.
- Monotonicity: ensures that time stamp values always increase.
- DBMS executes conflicting operations in time stamp order to ensure serialisability of the transaction.
- If two transactions conflict, one often is stopped, rolled back, and assigned a new time stamp value.
- Each value requires two additional time stamps fields
- Last time field read
- Last update
- Time stamping tends to demand a lot of system resources because there is a possibility that many transactions may have to be stopped, rescheduled, and re-stamped.

5.5.7.1 Timestamp-Based Protocols

- Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that TS(T_i) < TS(T_j).
- The protocol manages concurrent execution such that the time-stamps determine the serialisability order.
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
- **W-timestamp**(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
- **R-timestamp**(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.
- The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- Suppose a transaction Ti issues a **read**(Q)
  1. If TS(T_i) ≦ **W-timestamp**(Q), then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T_i is rolled back.
  2. If TS(T_i) ≧ **W-timestamp**(Q), then the read operation is executed, and **R-timestamp**(Q) is set to the maximum of **R-timestamp**(Q) and TS(T_i).
- Suppose that transaction T_i issues **write**(Q).
  1. If TS(T_i) < **R-timestamp**(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and T_i is rolled back.
  2. If TS(T_i) < **W-timestamp**(Q), then T_i is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and T_i is rolled back.
  3. Otherwise, the **write** operation is executed, and W-timestamp(Q) is set to TS(T_i).

Concurrency Control with Optimistic Methods

- A **validation-based protocol** that assumes most database operations do not conflict.
- No requirement on locking or time stamping techniques.

- Transaction executed without restrictions until committed and fully in the hope that all will go well during validation.
- Two or three Phases:
  - **Read (and Execution) Phase** – the transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values.
  - **Validation Phase** – the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database.
  - **Write Phase** – the changes are permanently applied to the database.

➢ The optimistic approach is acceptable for mostly read or query database system that require very few update transactions.

➢ Each transaction T_i has 3 timestamps
  - **Start**(T_i) : the time when T_i started its execution
  - **Validation**(T_i): the time when T_i entered its validation phase
  - **Finish**(T_i): the time when T_i finished its write phase

➢ Serialisability order is determined by timestamp given at validation time, to increase concurrency. Thus TS(T_i) is given the value of **Validation**(T_i).

➢ This protocol is useful and gives greater degree of concurrency if probability of conflicts is low. That is because the serialisability order is not pre-decided and relatively less transactions will have to be rolled back.

5.5.8 Database Recovery Management

➢ Restores a database to previously consistent state, usually inconsistent, to a previous consistent state.

➢ Based on the **atomic transaction property**: all portions of the transaction must be treated as a single logical unit of work, in which all operations must be applied and completed to produce a consistent database.

➢ Level of backup
  - Full backup – dump of the database.
  - Differential backup – only the last modifications done to the database are copied.
  - Transaction log – only the transaction log operations that are not reflected in a previous backup copy of the database.

➢ The database backup is stored in a secure place, usually in a different building, and protected against dangers such as file, theft, flood, and other potential calamities.

➢ Causes of Database Failure
  - Software – be traceable to the operating system, the DBMS software, application programs, or virus.
  - Hardware – include memory chip errors, disk crashes, bad disk sectors, disk full errors.
  - Programming Exemption – application programs or end users may roll back transactions when certain conditions are defined.
  - Transaction – the system detects deadlocks and aborts one of the transactions.
  - External – a system suffers complete destruction due to fire, earthquake, flood, etc.

5.5.8.1 Transaction Recovery

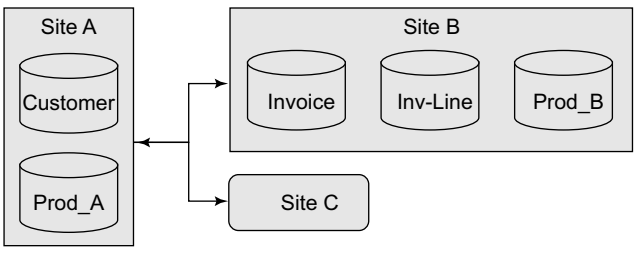
➢ Four important concepts to affect recovery process –
  - Write-ahead-log protocol – ensures that transaction logs are always written before any database data are actually updated.
  - Redundant transaction logs – ensure that a disk physical failure will not impair the DBMS ability to recover data.
  - Database buffers – create temporary storage area in primary memory used to speed up disk operations and improve processing time.
  - Database checkpoint – setup an operation in which the DBMS writes all of its updated buffers to disk and registered in the transaction log.

- Transaction recovery procedure generally make use of deferred-write and write-through techniques.
  - Deferred-write (or Deferred-update)
    - Changes are written to the transaction log, not physical database.
    - Database updated after transaction reaches commit point.
    - Steps:
      1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
      2. For a transaction that started and committed before the last checkpoint, nothing needs to be done, because the data are already saved.
      3. For a transaction that performed a commit operation after the last checkpoint, the DBMS uses the transaction log records to redo the transaction and to update the database, using "after" values in the transaction log. The changes are made in ascending order, from the oldest to the newest.
      4. For any transaction with a RP::BACK operation after the last checkpoint or that was left active (with neither a `COMMIT` nor a `ROLLBACK`) before the failure occurred, nothing needs to be done because the database was never updated.
  - Write-through (or immediate update)
    - Immediately updated by during execution
    - Before the transaction reaches its commit point
    - Transaction log also updated
    - Transaction fails, database uses log information to `ROLLBACK`
    - Steps:
      1. Identify the last checkpoint in the transaction log. This is the last time transaction data was physically saved to disk.
      2. For a transaction that started and committed before the last checkpoint, nothing needs to be done, because the data are already saved.
      3. For a transaction that committed after the last checkpoint, the DBMS redoes the transaction, using "after" values in the transaction log. Changes are applied in ascending order, from the oldest to the newest.
      4. For any transaction with a `ROLLBACK` operation after the last checkpoint or that was left active (with neither a `COMMIT` nor a `ROLLBACK`) before the failure occurred, the DBMS uses the transaction log records to `ROLLBACK` or undo the operations, using the "before" values in the transaction log. Changes are applied in reverse order, from the newest to the oldest.

5.6 Solved Questions

1. The following question is based on the DDBMS scenario in the following Figure.

Table	Fragments	Location
Customer	N/A	A
Product	Prod_A	A
	Prod_B	B
INVOICE	N/A	B
INV_LINE	N/A	B



Specify the types of operations the database must support (remote request, remote transaction, distributed transaction, or distributed request) in order to perform the following operations:

To answer the following questions, remember that the key to each answer is in the number of remote data processors that are accessed by each request/transaction. Remember that a distributed request is necessary if a single SQL statement is to access more than one remote DP site.

Use the following summary:

	Number of remote DPs	
Operation	1	> 1
Request	Remote	Distributed
Transaction	Remote	Distributed

Based on this summary, the following questions have to be answered.

At Site C:

A. **SELECT ***
FROM CUSTOMER;

Answer: This SQL sequence represents a *remote request.*

B. **SELECT ***
FROM INVOICE
WHERE INV_TOTAL > 1000;

Answer: This SQL sequence represents a *remote request.*

C. **SELECT ***
FROM PRODUCT
WHERE PROD_QOH < 10;

Answer: This SQL sequence represents a distributed request. Note that the distributed request is required when a single request must access two DP sites. The PRODUCT table is fragmented across two sites, A and B. In order for this SQL sequence to run properly, it must access the data at both sites.

D. **BEGIN WORK;**
UPDATE CUSTOMER
SET CUS_BALANCE = CUS_BALANCE + 100
WHERE CUS_NUM='10936';
INSERT INTO INVOICE(INV_NUM, CUS_NUM, INV_DATE, INV_TOTAL)
VALUES ('986391', '10936', '15-FEB-2002', 100);
INSERT INTO INVLINE(INV_NUM, PROD_CODE, LINE_PRICE)
VALUES ('986391', '1023', 100);
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH – 1
WHERE PROD_CODE = '1023';
COMMIT WORK;

Answer: This SQL sequence represents a *distributed request.*

Note that UPDATE CUSTOMER and the two INSERT statements only require remote request capabilities. However, the entire transaction must access more than one remote DP site, so we also need distributed transaction capability. The last UPDATE PRODUCT statement accesses two remote sites because the PRODUCT table is divided into two fragments located at two remote DP sites. Therefore, the transaction as a whole requires distributed request capability.

E. **BEGIN WORK;**
INSERT CUSTOMER(CUS_NUM, CUS_NAME, CUS_ADDRESS CUS_BALANCE)
VALUES ('34210",'Victor Ephanor', '123 Main St', 0.00);
INSERT INTO INVOICE(INV_NUM, CUS_NUM, INV_DATE, INV_TOTAL)
VALUES ('986434', '34210', '10-AUG-1999', 2.00);
COMMIT WORK;

Answer: This SQL sequence represents a *distributed transaction*. Note that, in this transaction, each individual request requires only remote request capabilities. However, the transaction as a whole accesses two remote sites. Therefore, distributed request capability is required.

A**t Site A:**

F. **SELECT CUS_NUM, CUS_NAME, INV_TOTAL**
FROM CUSTOMER, INVOICE
WHERE CUSTOMER.CUS_NUM = INVOICE.CUS_NUM;

Answer: This SQL sequence represents a *remote request*. Note that the request accesses only one remote DP site; therefore only remote request capability is needed.

G. **SELECT ***
FROM INVOICE
WHERE INV_TOTAL > 1000;

Answer: This SQL sequence represents a *remote request*, because it accesses only one remote DP site.

H. **SELECT ***
FROM PRODUCT
WHERE PROD_QOH < 10;

Answer: This SQL sequence represents a *distributed request.* In this case, the PRODUCT table is partitioned between two DP sites, A and B. Although the request accesses only one remote DP site, it accesses a table that is partitioned into two fragments: PROD A and PROD B. Only if the DBMS supports distributed requests a single request can access a partitioned table.

A**t Site B:**

I. **SELECT ***
FROM CUSTOMER;

Answer: This SQL sequence represents a *remote request.*

J. **SELECT CUS_NAME, INV_TOTAL**
FROM CUSTOMER, INVOICE
WHERE INV_TOTAL > 1000;

Answer: This SQL sequence represents a *remote request.*

K. **SELECT ***
FROM PRODUCT
WHERE PROD_QOH < 10;

Answer: This SQL sequence represents a *distributed request*. (See explanation for Part h.)

2. The following data structure and constraints exist for a magazine publishing company.

- The company publishes one regional magazine each in Florida (FL), South Carolina (SC), Georgia (GA), and Tennessee (TN).
- The company has 300,000 customers (subscribers) distributed throughout the four states listed in Part a.
- On the first of each month an annual subscription INVOICE is printed and sent to all customers whose subscription is due (CUS_SUB_DATE) for renewal. The INVOICE entity contains a REGION attribute to indicate the state (FL, SC, GA, TN) in which the customer resides.

CUSTOMER (CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_CITY, CUS_STATE, CUS_SUB_DATE)

INVOICE (INV_NUM, REG_CODE, CUS_NUM, INV_DATE, INV_TOTAL)

The company's management is aware of the problems associated with centralised management and has decided that it is time to decentralise the management of the subscriptions in its four regional subsidiaries. Each subscription site will handle its own customer and invoice data. The company's management, however, wants to have access to customer and invoice data to generate annual reports and to issue ad hoc queries, such as:

- List all current customers by region.
- List all new customers by region.
- Report all invoices by customer and by region.

A. Given these requirements, how must you partition the database?

Answer: The CUSTOMER table must be partitioned horizontally by state.

B. What recommendations will you make regarding the type and characteristics of the required database system?

Answer: The Magazine Publishing Company requires a distributed system with distributed database capabilities. The distributed system will be distributed among the company locations in South Carolina, Georgia, Florida, and Tennessee.

The DDBMS must be able to support distributed transparency features, such as fragmentation transparency, replica transparency, transaction transparency, and performance transparency. Heterogeneous capability is not a mandatory feature since we assume there is no existing DBMS in place and that the company wants to standardise on a single DBMS.

C. What type of data fragmentation is needed for each table?

Answer: The database must be horizontally partitioned, using the STATE attribute for the CUSTOMER table and the REGION attribute for the INVOICE table.

D. What must be the criteria used to partition each database?

Answer: The following fragmentation segments reflect the criteria used to partition each database:

Horizontal Fragmentation of the CUSTOMER Table By State

Fragment Name	Location	Condition	Node name
C1	Tennessee	CUS_STATE = 'TN'	NAS
C2	Georgia	CUS_STATE = 'GA'	ATL
C3	Florida	CUS_STATE = 'FL'	TAM
C4	South Carolina	CUS_STATE = 'SC'	CHA

Horizontal Fragmentation of the INVOICE Table By Region

Fragment Name	Location	Condition	Node name
I1	Tennessee	CUS_STATE = 'TN'	NAS
I2	Georgia	CUS_STATE = 'GA'	ATL
I3	Florida	CUS_STATE = 'FL'	TAM
I4	South Carolina	CUS_STATE = 'SC'	CHA

E. Design the database fragments. Show an example with node names, location, fragment names, attribute names, and demonstration data.

Answer:

Fragment C1 Location: Tennessee Node: NAS

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_CITY	CUS_STATE	CUS_SUB_DATE
10884	James D. Burger	123 Court Avenue	Memphis	TN	8-DEC-2000
10993	Lisa B. Barnette	910 Eagle Street	Nashville	TN	12-MAR-2000

Fragment C2 Location: Georgia Node: ATL

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_CITY	CUS_STATE	CUS_SUB_DATE
11887	Ginny E. Stratton	335 Main Street	Atlanta	GA	11-AUG-2000
13558	Anna H. Ariona	657 Mason Ave.	Dalton	GA	23-JUN-2000

Fragment C3 Location: Florida Node: TAM

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_CITY	CUS_STATE	CUS_SUB_DATE
10014	John T. Chi	456 Brent Avenue	Miami	FL	18-NOV-2000
15998	Lisa B. Barnette	234 Ramala Street	Tampa	FL	23-MAR-2000

Fragment C4 Location: South Carolina Node: CHA

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_CITY	CUS_STATE	CUS_SUB_DATE
21562	Thomas F. Matto	45 N. Pratt Circle	Charleston	SC	2-DEC-2000
18776	Mary B. Smith	526 Boone Pike	Charleston	SC	28-OCT-2000

Fragment I1 Location: Tennessee Node: NAS

INV_NUM	REGION-CODE	CUS_NUM	INV_DATE	INV_TOTAL
213342	TN	10884	1-NOV-2000	45.95
209987	TN	10993	15-APR-2000	45.95

Fragment I2 Location: Georgia Node: ATL

INV_NUM	REGION-CODE	CUS_NUM	INV_DATE	INV_TOTAL
198893	GA	11887	15-AUG-2000	70.45
224345	GA	13558	1-JUN-2000	45.95

Fragment I3 Location: Florida Node: TAM

INV_NUM	REGION-CODE	CUS_NUM	INV_DATE	INV_TOTAL
200915	FL	10014	1-NOV-2000	45.95
231148	FL	15998	1-MAR-2000	24.95

Fragment I4 Location: South Carolina Node: CHA

INV_NUM	REGION-CODE	CUS_NUM	INV_DATE	INV_TOTAL
243312	SC	21562	15-NOV-2000	45.95
231156	SC	18776	1-OCT-2000	45.95

F. What type of distributed database operations must be supported at each remote site?

Answer: To answer this question, we must first draw a map of the locations, the fragments at each location, and the type of transaction or request support required to access the data in the distributed database.

	Node				
Fragment	NAS	ATL	TAM	CHA	Headquarters
CUSTOMER	C1	C2	C3	C4	
INVOICE	I1	I2	I3	I4	
Distributed Operations Required	none	none	none	none	distributed request

Given the problem's specifications, we conclude that no interstate access of CUSTOMER or INVOICE data is required. Therefore, no distributed database access is required in the four nodes. For the headquarters, the manager wants to be able to access the data in all four nodes through a single SQL request. Therefore, the DDBMS must support distributed requests.

G. What type of distributed database operations must be supported at the headquarters site?

Answer: See the answer for part f.

1. What is meant by a recursive relationship type? Give some examples of recursive relationship types.

Answer: Recursive relationship is the entity type participates more than once in a relationship type in different role. For example a department manger is the supervisor of the employee work for the department or a supervisee of a general manger of a company.

2. Example of a 3NF table that is not BCNF,

Consder a problem involving students, courses and instructors. For each course, each student is taught by only one instructor. A course may be taught by more than one instructor. Each instructor teaches only one course. Assuming, S = student, C = course, I = instructor. The following FD's are seen valid in the relation.

SC → I

I → C

This table is 3NF with a candidate key SC:

student	**course**	**instructor**
Sutton	Math	Von Neumann
Sutton	Journalism	Murrow
Niven	Math	Von Neumann
Niven	Physics	Fermi
Wilson	Physics	Einstein

Delete anomaly: If Sutton drops Journalism, then we have no record of Murrow teaching Journalism.

How can we decompose this table into BCNF?

Decomposition 1 (bad).....eliminates the delete anomaly

SC (no FDs) and I -> C (two tables)

Problems -

1. lossy join
2. dependency SC -> I is not preserved

SC	student	course IC	instructor	course
	Sutton	Math	Von Neumann	Math
	Sutton	Journalism	Murrow	Journalism
	Niven	Math	Fermi	Physics
	Niven	Physics	Einstein	Physics
	Wilson	Physics		

---------- join SC and IC ----------

SCI'	student	course	instructor
	Sutton	Math	Von Neumann
	Sutton	Journalism	Murrow
	Niven	Math	Von Neumann
	Niven	Physics	Fermi
	Niven	Physics	Einstein **(spurious row)**
	Wilson	Physics	Fermi **(spurious row)**
	Wilson	Physics	Einstein

Decomposition 2 (better).....eliminates the delete anomaly

SI (no FD) and I → C

Advantages – eliminates the delete anomaly, lossless

Disadvantage – dependency SC → I is not preserved

SI	student	instructor IC	instructor	course
	Sutton	Von Neumann	Von Neumann	Math
	Sutton	Murrow	Murrow	Journalism
	Niven	Von Neumann	Fermi	Physics
	Niven	Fermi	Einstein	Physics
	Wilson	Einstein	Dantzig	Math **(new)**
	Sutton	Dantzig **(new)**		

The new row is allowed in SI using unique (student, instructor) in the create table command, and the join of SI and IC is lossless. However, a join of SI and IC now produces the following two rows:

student	**course**	**instructor**
Sutton	Math	Von Neumann
Sutton	Math	Dantzig which violates the FD SC → I.

Decomposition 3 (tradeoff between integrity and performance)

SC → I and I → C (two tables with redundant data)

Problems – extra updates and storage cost

3. What is a composite entity, and when is it used?

Answer: A composite entity, also known as a bridge entity, is one that is used to transform M:N relation-

ships into sets of 1:M relationships. The composite entity's primary key consists of the combination of primary keys from the entities it connects. For a detailed review of the role played by composite entities, refer to the second discussion focus question **(How are M:N relationships handled in the development of an E-R diagram?)**

6. What two courses of action are available to a designer when a multivalued attribute is encountered?

Answer: The designer can split the multivalued attributes into its components and keep these components in the same entity.

The designer may also create a new entity composed of the multivalued attribute's components and link this new entity to the entity in which the multivalued attributes occurred originally. This second option is especially desirable when the number of outcomes in the multivalued attribute is, for all practical purposes, unlimited. For example, employees classified as "technical" may have certifications in many different areas and at many different levels.

7. Consider the following schema for a relational database:

Relation	Attributes
professor	ssn, profname, status,salary
course	crscode, crsname, credits
taught	croscode, semester, ssn

Assumption: (1) Each course has only one instructor in each semester; (2) all professors have different salaries; (3) all professors have different names; (4) all courses have different names; (5) status can take values from "Full", "Associate", and "Assistant".

For each of the following queries, give an expression in *relational algebra:*

(a) Return those professors who have taught both 'csc6710' and 'csc 7710'.

$\pi_{ssn}(\sigma_{crscode='csc6710'}(Taught)) \cap \pi_{ssn}(\sigma_{crscode = 'csc7710'}(Taught))$

(b) Return those professors who have taught 'csc7710'.

$\pi_{ssn}(Professor)-\pi_{ssn}(\sigma_{crscode = 'csc7710'}(Taught))$

(c) Return those professors who taught 'CSC6710' and 'CSC 7710' in the same semester

$\pi_{ssn}(\sigma_{crscode1='csc6710'}(Taught[crscode\ 1, ssn\ semester])) \bowtie$

$\sigma_{crscode2 = 'csc7710'}(Taught[crscode\ 2, ssn\ semester])$

(d) Return those professors who taught 'CSC6710' or 'CSC 7710' but not both

$\pi_{ssn}(\sigma_{crscode='csc6710'v\ crscode = 'csc\ 7710'}(Taught)) - \pi_{ssn}(\sigma_{crscode='csc6710'}(Taught) \cap (\pi_{ssn}(\sigma_{crscode='csc7710'}(Taught)))$

(e) Return those courses that have never been taught.

$\pi_{crscode}(Course)-\pi_{crscode}(Taught)$

(f) Return those courses that have never been taught at least in two semesters.

$\pi_{crscode}(\sigma_{semester\ 1 <> semester\ 2}(Taught[crscode, ssn1\ semester1] \bowtie Taught[crscode, ssn2\ semester2]))$

(g) Return the names of professors who ever taught 'CSC6710'.

$\pi_{profname}(\sigma_{crscode = 'csc6710'}(Taught) \bowtie Professor)$

(h) Return the names of full professors who ever taught 'CSC6710'.

$\pi_{profname}(\sigma_{crscode = 'csc6710'}(Taught) \bowtie \sigma_{status = 'full'}(Professor)$

(i) Return the names of full professors who ever taught at least two courses in one semester.

$\pi_{profname}(\pi_{ssn}(\sigma_{crscode1<>crscode2}(Taught[crscode\ 1, ssn\ semester] \bowtie (Taught[crscode\ 2, ssn\ semester])) \bowtie \sigma_{status = 'full'}(Professor))$

(j) List all the course names that professors 'Smith' taught in fall of 2007.

$\pi_{crsname}(\sigma_{profname='Smith'}(Professor) \bowtie (\sigma_{semester='f2007'}(Taught) \bowtie (Course)$

(k) List the names of those courses that professor Smith have never taught.

$\pi_{crsname}(Course)-\pi_{crsname}(\sigma_{profname='Smith'}(Professor) \bowtie (Taught) \bowtie (Course)$

(l) Return those courses that have been taught by all professors.

$\pi_{crscode,\ ssn}(Taught)/\pi_{ssn}(Professor)$

(m) Return those courses that have been taught in all semesters.

$\pi_{crscode,\ semester}(Taught)/\pi_{semester}(Taught)$

(n) Return those courses that have been taught ONLY by assisitant professors.

$\pi_{crscode}(Course)-\pi_{crscode}(\sigma_{status \neq 'Assistant'}(Professor) \bowtie Taught)$

(o) Return those professors who have taught 'csc6710' but never 'csc7710'.

$\pi_{ssn}(\sigma_{crscode='csc\ 6710'}(Taught))-\pi_{ssn}(\sigma_{crscode='csc7710'}(Taught)$

For each of the following queries, give an expression in SQL:

(p) Return those professors who have taught both 'csc6710' and 'csc7710'.

```
SELECT T1.ssn
From Taught T1, Taught T2,
Where T1.crscode = 'CSC6710' AND
T2.crscode='CSC7710' AND T1.ssn=T2.ssn
```

(q) Return those professors who have never taught 'csc7710'.

```
(SELECT ssn
From Professor)
EXCEPT
(SELECT ssn
From Taught T
Where T.crscode = 'CSC7710')
```

(r) Return those professors who taught 'CSC6710' and 'CSC7710" in the same semester

```
SELECT T1.ssn
From Taught T1, Taught T2, Where T1.crscode = 'CSC6710' AND T2.crscode='CSC7710' AND T1.ssn=T2.ssn
AND T1.semester=T2.semester
```

(s) Return those professors who taught 'CSC6710' or 'CSC7710" but not both.

```
(SELECT ssn
FROM Taught T
WHERE T.crscode='CSC6710' OR T.crscode = 'CSC7710')
Except
(SELECT T1.ssn
From Taught T1, Taught T2,
Where T1.crscode = 'CSC6710') AND T2.crscode ='CSC7710' AND T1.ssn=T2.ssn)
```

(t) Return those courses that have been taught at least in 10 semesters.

```
SELECT crscode
FROM Taught
GROUP BY crscode
HAVING COUNT(*) >= 10
```

(u) Return those courses that have been taught by at least 5 different professors.

```
SELECT crscode
FROM (SELECT DISTINCT crscode, ssn FROM TAUGHT)
GROUP BY crscode
HAVING COUNT(*) >= 5
SELECT crscode
FROM Course C
WHERE (SELECT COUNT(DISTINCT *) FROM Taught T
WHERE T.crscode = C.crscode
) >=5.
```

(v) Return the names of full professors who ever taught at least two courses in one semester.

```
SELECT P.profname
FROM Professor P, Taught T1, Taught T2
WHERE P.status = 'Full' AND P.ssn = T1.ssn AND T1.ssn = T2.ssn
AND T1.crscode <> T2.crscode AND T1.semester = T2.semester
```

(w) In chronological order, list the number of courses that the professor with ssn ssn =
123456789 taught in each semester.

```
SELECT semester, COUNT(*)
FROM Taught
WHERE ssn = '123456789'
GROUP BY semester
ORDER BY semester ASC
```

(x) Delete those professors who taught less than 40 credits.

```
DELETE FROM Professor
WHERE ssn IN(
SELECT T.ssn
      FROM Taught T, Course C
WHERE T.crscode = C.crscode
GROUP BY ssn
HAVING SUM(C.credits) < 40
)
```

(y) Return the professors who taught the largest number of courses in Fall 2001.

```
SE
LECT *
FROM Professor P1
WHERE Not EXISTS
(
SELECT *
FROM Professor P2
WHERE(
(SELECT COUNT(*)
FROM Taught
WHERE Taught.ssn = P2.ssn AND Taught. semester='F2001')
>
(SELECT COUNT(*)
FROM Taught
WHERE Taught.ssn = P1.ssn AND Taught. semester='F2001')
)
```

(z) Change all the credits to 4 for those courses that are taught in f2006 semester.

```
UPDATE Course
SET credits = 4
WHERE crscode IN
(
SELECT crscode
FROM Taught
WHERE semester = 'f2006'
)
```

(aa) Return the name(s) of the professor(s) who taught the most number of courses in S2006.

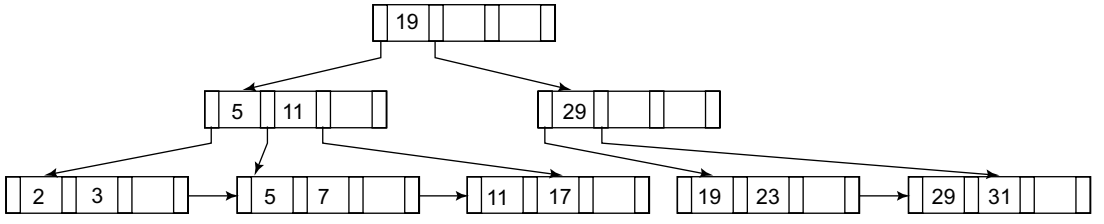
```
SELECT profname
FROM Professor
WHERE ssn IN(
SELECT ssn FROM Taught
WHERE semester = 'S2006'
GROUP BY ssn
HAVING COUNT(*) =
(SELECT MAX(Num)
FROM
(SELECT ssn, COUNT(*) as Num
FROM Taught
WHERE semester = 'S2006'
GROUP BY ssn)
)
    )
```

8. Construct a B+- tree for the following set of key values:
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

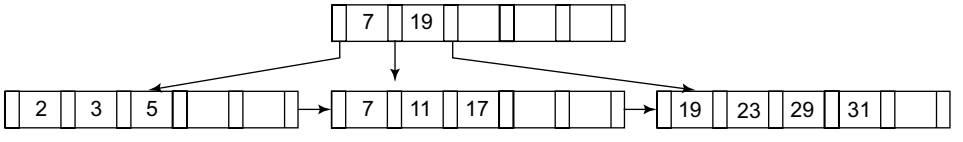
(i) Assume that the tree is initially empty and values are added in ascending order. Construct B+- trees for the cases where the number of pointers that will fit in one node is as follows:

A. Four B. Six C. Eight

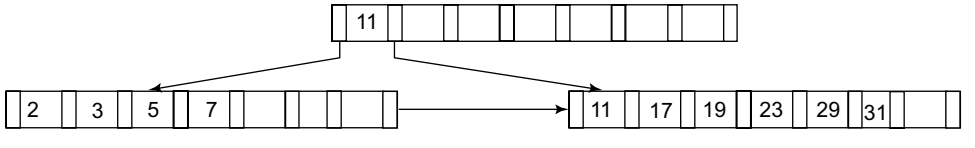
Answer: The following were generated by inserting values into the B+- tree in ascending order. A node (other than the root) was never allowed to have fewer than [n/2] values/pointers.



(a)



(b)



(c)

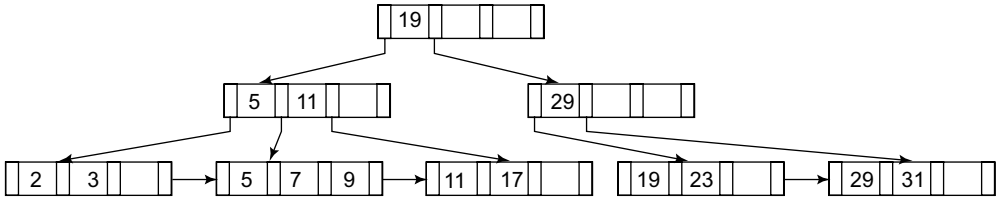
(ii) For each B+- tree in Question i), show the form of the tree after each of the following series of operations:

A. Insert 9 B. Insert 10 C. Insert 8 D. Delete 23

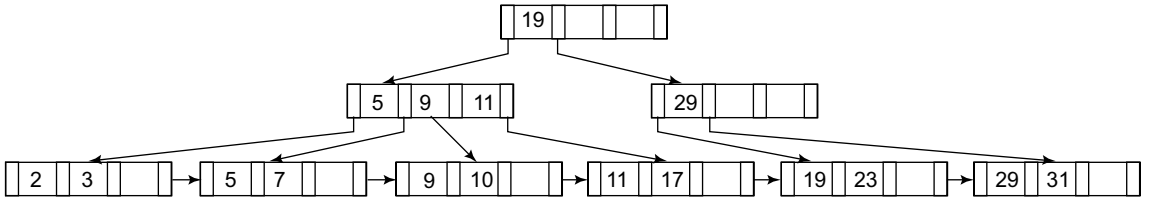
E. Delete 19

Answer:

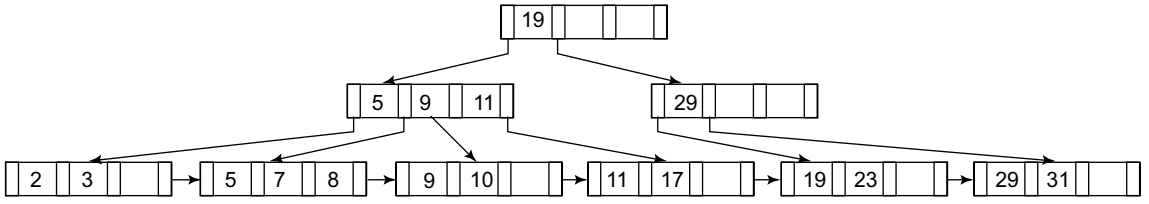
With n = 4



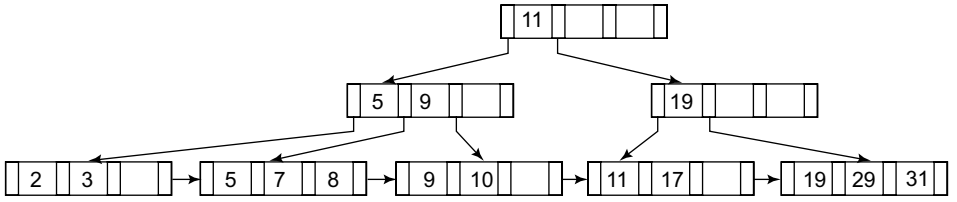
Insert 10:



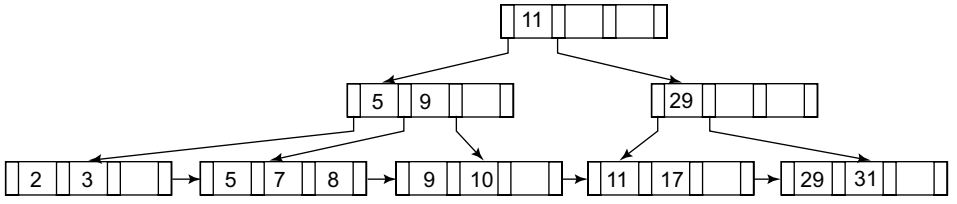
Insert 8:



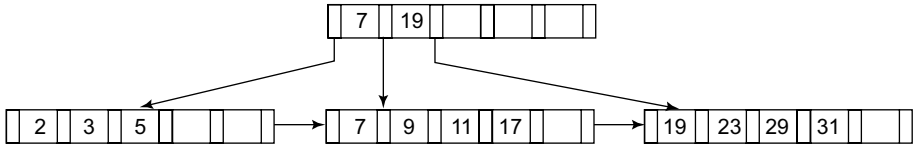
Delete 23:



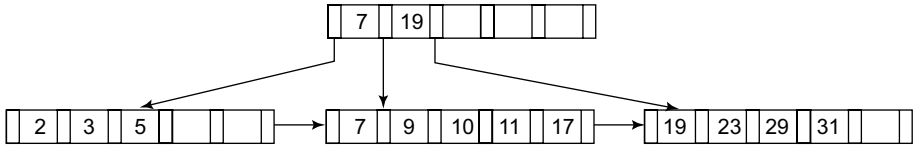
Delete 19:



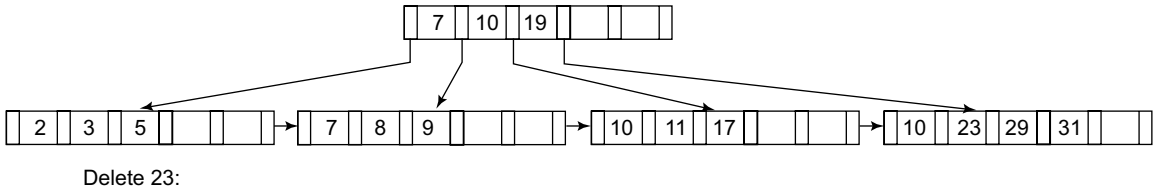
With n = 6



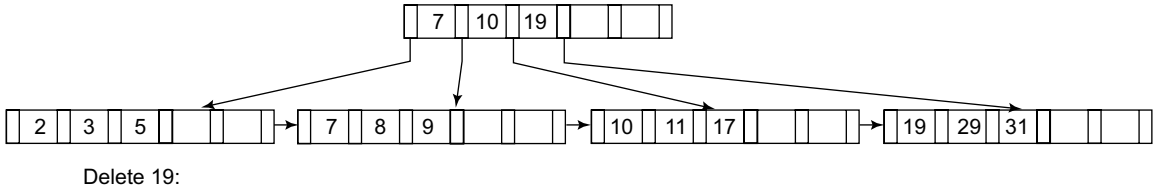
Insert 10:



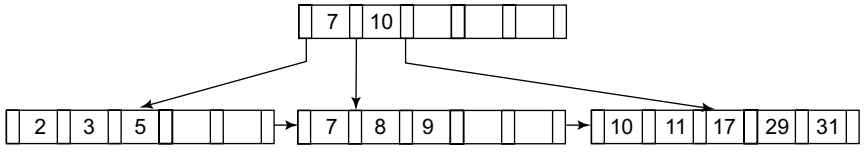
Insert 8:



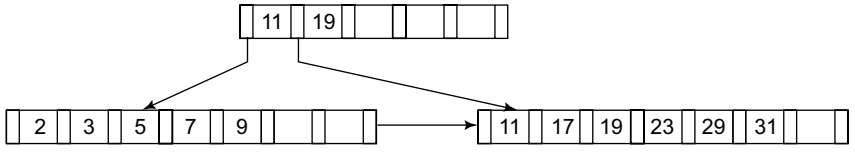
Delete 23:



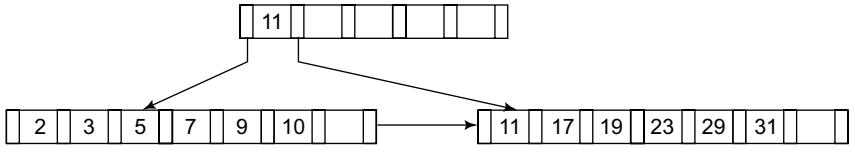
Delete 19:



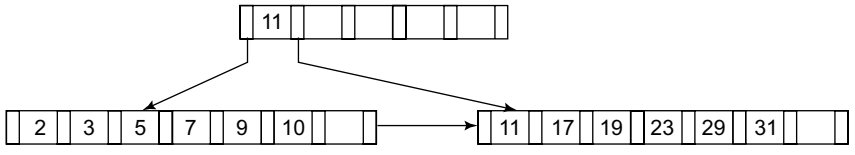
With n=8



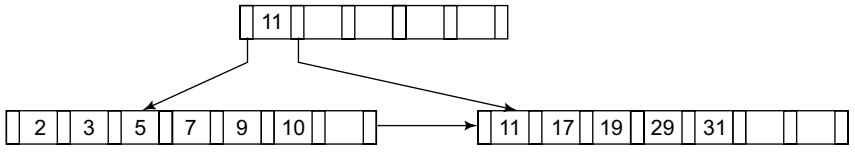
Insert 10:



Insert 8:



Delete 23:



Delete 19:

9. How does Tuple-oriented relational calculus differ from domain-oriented relational calculus?

The tuple-oriented calculus uses a tuple variables i.e., variable whose only permitted values are tuples of that relation. E.g. QUEL

The domain-oriented calculus has domain variables i.e., variables that range over the underlying domains instead of over relation. E.g. ILL, DEDUCE.

10. When is a functional dependency F said to be minimal?

- Every dependency in F has a single attribute for its right hand side.
- We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$ where Y is a proper subset of X and still have a set of dependency that is equivalent to F.

- We cannot remove any dependency from F and still have set of dependency that is equivalent to F.

11. What is Multivalued dependency?

Multivalued dependency denoted by $X \rightarrow\rightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation r of R: if two tuples t1 and t2 exist in r such that t1[X] = t2[X] then t3 and t4 should also exist in r with the following properties

- t3[x] = t4[X] = t1[X] = t2[X]
- t3[Y] = t1[Y] and t4[Y] = t2[Y]
- t3[Z] = t2[Z] and t4[Z] = t1[Z]
  where [Z = (R-(X U Y))]

12. What is Lossless join property?

It guarantees that the spurious tuple generation does not occur with respect to relation schemas after decomposition.

13. What is 4NF?

A relation schema R is said to be in 4NF if for every Multivalued dependency $X \rightarrow\rightarrow Y$ that holds over R, one of following is true

- X is subset or equal to (or) XY = R.
- X is a super key.

14. What is 5NF?

A Relation schema R is said to be 5NF if for every join dependency {R1, R2,..., Rn} that holds R, one the following is true

- Ri = R for some i.
- The join dependency is implied by the set of FD, over R in which the left side is key of R.

15. What is Domain-Key Normal Form?

A relation is said to be in DKNF if all constraints and dependencies that should hold on the the constraint can be enforced by simply enforcing the domain constraint and key constraint on the relation.

16. What do you mean by atomicity and aggregation?

A*tomicity:*

Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions. DBMS ensures this by undoing the actions of incomplete transactions.

A*ggregation*:

A concept which is used to model a relationship between a collection of entities and relationships. It is used when we need to express a relationship among relationships.

17. What is a Phantom Deadlock?

In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts.

18. What is a checkpoint and when does it occur?

A checkpoint is like a snapshot of the DBMS state. By taking checkpoints, the DBMS can reduce the amount of work to be done during restart in the event of subsequent crashes.

19. How can you find the minimal key of relational schema?

Minimal key is one which can identify each tuple of the given relation schema uniquely. For finding the minimal key it is required to find the closure that is the set of all attributes that are dependent on any given set of attributes under the given set of functional dependency.

A**lgo. I** Determining X^+, closure for X, given set of FDs F

1. Set $X^+ = X$
2. Set Old $X^+ = X^+$
3. For each FD $Y \rightarrow Z$ in F and if Y belongs to X^+ then add Z to X^+
4. Repeat steps 2 and 3 until Old $X^+ = X^+$

A**lgo.II** Determining minimal K for relation schema R, given set of FDs F

1. Set K to R that is make K a set of all attributes in R
2. For each attribute A in K
   A. Compute $(K - A)^+$ with respect to F
   b. If $(K - A)^+ = R$ then set $K = (K - A)^+$

20. What do you understand by dependency preservation?

Given a relation R and a set of FDs F, dependency preservation states that the closure of the union of the projection of F on each decomposed relation Ri is equal to the closure of F. i.e.,

$((\Pi_{R1}(F)) \cup \ldots \cup (\Pi_{Rn}(F)))^+ = F^+$

if decomposition is not dependency preserving, then some dependency is lost in the decomposition.

OBJECTIVE TYPE QUESTIONS

1. Software that defines a database, stores the data, supports a query language, produces reports and creates data entry screens is a:

A. Data dictionary
B. Database management system (DBMS)
C. Decision support system
D. Relational database

2. The separation of the data definition from the program is known as:

A. Data dictionary
B. Data independence
C. Data integrity
D. Referential integrity

3. In the client / server model, the database:
A. Is downloaded to the client upon request
B. Is shared by both the client and server
C. Resides on the client side
D. Resides on the server side

4. The traditional storage of data that is organised by customer, stored in separate folders in filing cabinets is an example of what type of 'database' system?
A. Hierarchical B. Network
C. Object oriented D. Relational

5. The database design that consists of multiple tables that are linked together through matching data stored in each table is called a:
A. Hierarchical database
B. Network database
C. Object oriented database
D. Relational database

6. What is the main limitation of Hierarchical Databases?
A. Limited capacity (unable to hold much data.
B. Limited flexibility in accessing data
C. Overhead associated with maintaining indexes
D. The performance of the database is poor

7. An abstract data type is used to:
A. Link data from remote databases
B. Prevent users from getting to database security information
C. Provide a conceptual view of the data so it is easier to understand
D. Store complex data structure to represent the properties of objects

8. Which component of the database management system (DBMS) most affects the ability to handle large problems (scalability)?
A. Data Storage Subsystem
B. Database Engine
C. Query Processor
D. Security Subsystem

9. The primary difference between the Relational database (RDB) and Object Oriented database (OODB) models is:
A. OODB incorporates methods in with the definition of the data structure, while RDB does not
B. OODB supports multiple objects in the same database while RDB only supports a single table per database
C. RDB allows the definition of the relationships between the different tables, while OODB does not allow the relationships to be defined between objects
D. RDB supports indexes, while OODB does not support indexes

10. Which of the following items is not the advantage of a DBMS?
A. Improved ability to enforce standards
B. Improved data consistency
C. Local control over the data
D. Minimal data redundancy

11. When building a database, the data dealing with an entity is modeled as a:
A. Attribute B. Class
C. Object D. Table

12. Database system modelers use this type of diagram to graphically represent both the data structure and how the different objects are interrelated.
A. Class Diagram B. Data Diagram
C. Object Diagram
D. Table Relationship Diagram

13. In relational database model, after conceptually designing your database, the information contained in a single class would be stored in a:
A. Database B. Field
C. Property D. Table

14. The property (or set of properties) that uniquely defines each row in a table is called the:
A. Identifier B. Index
C. Primary key D. Symmetric key

15. Business rules can be represented in the database through:
A. Associations (or relationships)
B. Attributes C. Properties
D. Secondary keys

16. The association role defines:
A. How tables are related in the database
B. The relationship between the class diagram and the tables in the database
C. The tables that contains each attribute
D. Which attribute is the table's primary key

17. The purpose of an N-Ary association is:
- A. To capture a parent-child relationship
- B. To deal with one to many relationships
- C. To deal with relationships that involve more than two tables
- D. To represent an inheritance relationship

18. A reflexive association is one where one class is:
- A. Broken down into special cases
- B. Combined with multiple other classes
- C. Combined with one other class
- D. Linked back to itself

19. Which of the following statements is not correct?
- A. A primary goal of a database system is to share data with multiple users
- B. It is possible to change a method or property inherited from a higher level class
- C. While companies collect data all the time, the structure of the data changes very often.
- D. In a client / server environment, data independence causes client side applications to be essentially independent of the database stored on the server side.

20. Which of the following statements is not correct?
- A. Data Normalisation is the process of defining the table structure
- B. The purpose of class diagrams is to model the interrelationships between the different classes in the database
- C. Individual objects are stored as rows in a table
- D. Properties of an object are stored as columns in a table.

21. Which of the following statements is not correct?
- A. The primary key must be unique for a given table
- B. Specifying a zero (0) for the lower bound for the association multiplicity on a class diagram indicates that the item is required
- C. Specifying a one (1) for the lower bound for the association multiplicity on a class diagram indicates that the item is required
- D. Most databases allow multiple records that are identical (i.e., records that have the same values for all properties).

22. Which of the following statements is not correct?
- A. All many-to-many relationships must be converted to a set of one-to-many relationships by adding a new entity
- B. In a one-to-one relationship between two classes, the two classes are generally described by one table in relational database model
- C. Encapsulation provides some security and control features
- D. Properties and functions can be protected from other areas of the applications

23. There is a relational schema which has k attributes. The domain of each attribute consists of exactly 2 elements. A table is defined as subset of tuples where in each tuple, a value is defined for each of the k attributes. The minimum value of k needed for the number of distinct tuples to exceed 10^9 is

A. 5 B. 9 C. 17 D. 32

E. 24

24. The relation PAYMENT(Cust-ID, Account, Amount_Paid, Date_Paid, Type_Payment, Discount). Assume that a customer may have more than one account and that he or she can make several payments on any day but not more than one payment per day can be applied to each account. The key for the relation can be:
- A. Cust-ID, Account
- B. Account, Date_Paid
- C. Cust-ID, Account, Date_Paid
- D. None

25. The relation STORE(Location, No-of-Employees, Total-Monthly-Sales, Manager, City)
- A. No key available for this
- B. If we assign STORE_ID as an extra attribute it can act like PK
- C. If we restrict a person to manage only one store then Manager attribute can become as PK
- D. If we restrict a person can manage all the stores in a City then Manager attribute can become as PK
- E. None

26. Candidate keys for a relation R(X, Y, Z, W, P) are

A. 1 B. 2 C. 3 D. 4

E. None

27. The valid FD's in the following relation are

A	B	C
f	e	e
d	e	e
b	c	e
a	c	d
a	b	c

A. $A \rightarrow B$ B. $AB \rightarrow C$

C. $A \rightarrow C$ D. $AC \rightarrow B$

E. None

28. Out of the following FD'S F={ $A \rightarrow BC$, $E \rightarrow C$, $D \rightarrow AEF$, $ABF \rightarrow BD$}. Then the left reduced set is

A. { $A \rightarrow BC$, $E \rightarrow C$, $D \rightarrow AEF$, $ABF \rightarrow BD$}

B. { A→BC, E→C, D→AE, AB→BD}
C. { A→BC, E→C, D→AEF, AF→BD}
D. { A→BC, E→C, D→AEF, AB→BD}
E. None

29. The canonical cover of following FD'S F={ A → BC, E → C, D → AEF, ABF → BD}.
A. {A→B, A→C, E→C, D→A, D→E, E→F, AF→D}
b. {A→B, A→C, E→C, D→A, D→E, D→F, AF→D}
c. {A→B, A→C, E→C, D→A, D→E, E→F, A→D}
d. {A→B, A→C, E→C, D→A, D→E, E→F, AF→D}
E. None

30. If a relation contains only one key (single attribute) and no transitive dependencies then select more valid one.
A. 1NF B. 2NF
C. Every non prime attribute will be fully functionally dependent on key
D. 3NF
E. None

31. The number of candidate keys for a relation (ABCDEFGH) with the given FD's {A→C, B→D,G→H, E→F, C→G}
A. 1 B. 2 C. 3 D. 4
E. None

32. The canonical cover of set of FD's F={ A→BC, B→C, A→B, AB→C}
A. A→C A→B B. A→B A→B
C. A→B B→C D. None

33. List the candidate keys for the relation R(ABCDE) with FD's F={ A→BC, B→D, CD→E, E→A}
A. A B. CD C. E D. None

34. Construct a B-Tree for the following data (2,3,5,7,11,17,19,23,29,31). Assume in a node at most 4 keys can be stored and atleast 2. number of splitting operations required are
A. 1 B. 2 C. 3 D. 4

ANSWER KEY

1. B	**2.** B	**3.** D	**4.** A
5. D	**6.** B	**7.** D	**8.** B
9. A	**10.** C	**11.** A	**12.** A
13. D	**14.** C	**15.** A	**16.** A
17. C	**18.** D	**19.** C	**20.** A
21. B	**22.** B	**23.** D	**24.** C
25. B	**26.** B	**27.** B	**28.** C
29. B	**30.** D	**31.** E	**32.** C
33. A	**34.** B		

Previous Years' GATE Questions

1. An index is clustered, if **(GATE 2013)**
A. It is on a set of fields that form a candidate key.
B. It is on a set of fields that include the primary key.
C. The data records of the file are organised in the same order as the data entries of the index.
D. The data records of the file are organised not in the same order as the data entries of the index.

2. Consider the following relational schema. **(GATE 2013)**

Students(rollno: integer, sname: string)
Courses(courseno: integer, cname: string)
Registration(rollno: integer, courseno: integer, percent: real)

Which of the following queries are equivalent to this query in English?

"Find the distinct names of all students who score more than 90% in the course numbered 107"

(I) SELECT DISTINCT S.sname
FROM Students as S. Registration as R
WHERE R. rollno = S.rollno AND R. Courseno = 107 AND R. percent > 90

(II) $\Pi_{sname}\ (\sigma_{courseno\ =\ 107\ \wedge\ percent\ >\ 90}$ (Registration $\bowtie$ Students)

(III) $\{T \mid \exists S \in$ Students, $\exists R \in$ Registration (S. rollno = R. rollno $\wedge$ R. courseno = 107 $\wedge$ R. percent > 90 $\wedge$ T. sname = S. sname)$\}$

(IV) $\{<S_N> \mid \exists S_R\ \exists R_P\ (<S_R, S_N> \in$ Students $\wedge <S_R, 107, R_P> \in$ Registration $\wedge\ R_P > 90)\}$

A. I, II, III and IV B. I, II and III only
C. I, II and IV only D. II, III and IV only

Relation *R* has eight attributes *ABCDEFGH*. Fields of R contain only atomic values.

F={CH→G, A→BC, B→CFH, E→A, F→EG} is a set of functional dependencies (FDs) so that *F+* is exactly the set of FDs that hold for R.

3. How many candidate keys does the relation R have? **(GATE 2013)**
A. 3 B. 4 C. 5 D. 6

4. The relation R is **(GATE 2013)**
A. In 1NF, but not in 2NF.
B. In 2NF, but not in 3NF.
C. In 3NF, but not in BCNF.
D. In BCNF.

5. Which of the following statements are true about an SQL query? **(GATE 2012)**

P: An SQL query can contain a HAVING clause even if it does not a GROUP BY clause

Q: An SQL query can contain a HAVING clause only if it has a GROUP BY clause

R: All attributes used in the GROUP BY clause must appear in the SELECT clause

S: Not all attributes used in the GROUP BY clause need to apper in the SELECT clause

A. P and R B. P and S

C. Q and R D. Q and S

6. Given the basic ER and relational models, which of the following is INCORRECT? **(GATE 2012)**

A. An attributes of an entity can have more that one value

B. An attribute of an entity can be composite

C. In a row of a relational table, an attribute can have more than one value

D. In a row of a relational table, an attribute can have exactly one value or a NULL value

7. Suppose (A, B) and (C, D) are two relation schemas. Let r1 and r2 be the corresponding relation instances. B is a foreign key that refers to C in r2. If data in r1 and r2 satisfy referential integrity constraints, which of the following is always true? **(GATE 2012)**

A. $\Pi_B(r_1) - \Pi_C(r_2) = \varnothing$ B. $\Pi_C(r_2) - \Pi_B(r_1) = \varnothing$

C. $\Pi_B(r_1) = \Pi_C(r_2)$ D. $\Pi_B(r_1) - \Pi_C(r_2) \neq \varnothing$

Explanation:

B is a foreign key in r1 that refers to C in r2. r1 and r2 satisfy referential integrity constraints. So every value that exists in column B of r1 must also exist in column C of r2.

8. Which of the following is true? **(GATE 2012)**

A. Every relation in 2NF is also in BCNF

B. A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R

C. Every relation in BCNF is also in 3NF

D. No relation can be in both BCNF and 3NF

Explanation:

BCNF is a stronger version 3NF. So every relation in BCNF will also be in 3NF.

9. Consider the following transactions with data items P and Q initialised to zero:

T1: read (P) ;
read (Q) ;
if P = 0 then Q : = Q + 1 ;
write (Q) ;

T2: read (Q) ;
read (P) ;
if Q = 0 then P : = P + 1 ;
write (P) ;

Any non-serial interleaving of T1 and T2 for concurrent execution leads to **(GATE 2012)**

A. A serialisable schedule

B. A schedule that is not conflict serialisable

C. A conflict serialisable schedule

D. A schedule for which a precedence graph cannot be drawn

Explanation:

Two or more actions are said to be in conflict if:

1. The actions belong to different transactions.
2. At least one of the actions is a write operation.
3. The actions access the same object (read or write).

The schedules S1 and S2 are said to be conflict-equivalent if the following conditions are satisfied:

1. Both schedules S1 and S2 involve the same set of transactions (including ordering of actions within each transaction).
2. The order of each pair of conflicting actions in S1 and S2 are the same.

A schedule is said to be conflict-serialisable when the schedule is conflict-equivalent to one or more serial schedules.

In the given scenario, there are two possible serial schedules:

1. T1 followed by T2
2. T2 followed by T1.

In both of the serial schedules, one of the transactions reads the value written by other transaction as a first step. Therefore, any non-serial interleaving of T1 and T2 will not be conflict serialisable.

10. Consider the following relations A, B, C. How many tuples does the result of the following relational algebra expression contain? Assume that the schema of A U B is the same as that of A. **(GATE 2012)**

$(A \cup B) \bowtie_{A.Id>40 \text{ V } CId<15} C$

Table A

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

Table B

Id Name Age

---

15 Shreya 24

25 Hari 40

98 Rohit 20

99 Rohit 11

Table C

Id Phone Area

---

10 2200 02

99 2100 01

A. 7 B. 4 C. 5 D. 9

Explanation :

Result of AUB will be following table

Id Name Age

---

12 Arun 60

15 Shreya 24

99 Rohit 11

25 Hari 40

98 Rohit 20

The result of given relational algebra expression will be

Id Name	Age	Id	Phone	Area
12 Arun	60	10	2200	02
15 Shreya	24	10	2200	02
99 Rohit	11	10	2200	02
25 Hari	40	10	2200	02
98 Rohit	20	10	2200	02
99 Rohit	11	99	2100	01
98 Rohit	20	99	2100	01

11. Consider the above tables A, B and C. How many tuples does the result of the following SQL query contains? **(GATE 2012)**

```
SELECT A.id
FROM A
WHERE A.age > ALL (SELECT B.age
FROM B
WHERE B. name = "arun")
```

A. 4 B. 3

C. 0 D. 1

Explanation:

The meaning of "ALL" is the A.Age should be greater than all the values returned by the subquery. There is no entry with name "arun" in table B. So the subquery will return null. If a subquery returns null, then the condition becomes true for all rows of A. So all rows of table A are selected.

12. Consider a relational table with a single record for each registered student with the following attributes.

1. Registration_Number:< Unique registration number for each registered student
2. UID: Unique Identity number, unique at the national level for each citizen
3. BankAccount_Number: Unique account number at the bank. A student can have multiple accounts or joint accounts. This attributes stores the primary account number
4. Name: Name of the Student
5. Hostel_Room: Room number of the hostel

Which of the following options is incorrect?

(GATE 2011)

A. BankAccount_Number is a candidate key

B. Registration_Number can be a primary key

C. UID is a candidate key if all students are from the same country

D. If S is a superkey such that $S \cap UID$ is null then $S \cup UID$ is also a superkey

Explanation:

A Candidate Key value must uniquely identify the corresponding row in table. BankAccount_Number is not a candidate key. As per the question "A student can have multiple accounts or joint accounts. This attributes stores the primary account number". If two students have a joint account and if the joint account is their primary account, then BankAccount_Number value cannot uniquely identify a row.

13. Consider a relational table r with sufficient number of records, having attributes A1, A2,…, An and let $1 \Leftarrow p \Leftarrow n$. Two queries Q1 and Q2 are given below.

(GATE 2011)

Q1 : $\pi_{A1.....A_n}(\sigma_{A_p = c}(r)$ where c is a const

Q1 : $\pi_{A1.....A_n}(\sigma_{c_1 \le A_p \le c_2}(r)$ where c_1 and c_2 are constants.

The database can be configured to do ordered indexing on Ap or hashing on Ap. Which of the following statements is true?

A. Ordered indexing will always outperform hashing for both queries

B. Hashing will always outperform ordered indexing for both queries

C. Hashing will outperform ordered indexing on Q1, but not on Q2

D. Hashing will outperform ordered indexing on Q2, but not on Q1.

Explanation:

If record are accessed for a particular value from table, hashing will do better. If records are accessed in a range of values, ordered indexing will perform better.

14. Database table by name Loan_Records is given below. **(GATE 2011)**

Borrower	Bank Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT Count(*)
FROM ( (SELECT Borrower, Bank_Manager
   FROM Loan_Records) AS S
   NATURAL JOIN (SELECT Bank_Manager,
                Loan_Amount
          FROM Loan_Records) AS T );
```

A. 3 B. 9 C. 5 D. 6

Explanation :

Following will be contents of temporary table S

Borrower	Bank_Manager
Ramesh	Sunderajan
Suresh	Ramgqpal
Mahesh	Sunderjan

Following will be contents of temporary table T

Bank_Manager	Loan_Amount
Sunderajan	10000.00
Ramgopal	5000.00
Sunderjan	7000.00

Following will be the result of natural join of above two tables. The key thing to note is that the natural join happens on column name with same name which is Bank_Manager in the above example. "Sunderjan" appears two times in Bank_Manager column, so their will be four entries with Bank_Manager as "Sunderjan".

Borrower	Bank_Manager	Load_Amount
Ramesh	Sunderajan	10000.00
Ramesh	Sunderajan	7000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	10000.00
Mahesh	Sunderajan	7000.00

15. The table at any point in time. Using MX and MY, new records are inserted in the table 128 times with X and Y values being MX+1, 2*MY+1 respectively. It may be noted that each time after the insertion, values of MX and MY change. What will be the output of the following SQL query after the steps mentioned above are carried out? **(GATE 2011)**

SELECT Y FROM T WHERE X = 7;

A. 127 B. 255 C. 129 D. 257

Explanation:

```
X   Y
-------
1   1
2   3
3   7
4   15
5   31
6   63
7   127
  .....
  .....
```

16. A relational schema for a train reservation database is given below.

Passenger (pid, pname, age)

Reservation (pid, class, tid) **(GATE 2010)**

Table: Passenger

```
pid pname age
-----------------
0 Sachin   65
1 Rahul    66
2 Sourav   67
3 Anil     69
```

Table : Reservation

```
pid class tid
---------------
0 AC   8200
1 AC   8201
2 SC   8201
5 AC   8203
1 SC   8204
3 AC   8202
```

What pids are returned by the following SQL query for the above instance of the tables?

```
SLECT pid
FROM Reservation,
WHERE class 'AC' AND
  EXISTS (SELECT *
    FROM Passenger
    WHERE age > 65 AND
    Passenger. pid = Reservation.pid)
```

A. 1, 0 B. 1, 2 C. 1, 3 D. 1, 5

Explanation :

When a subquery uses values from outer query, the subquery is called correlated. The correlated subquery is evaluated once for each row processed by the outer query. The outer query selects 4 entries (with pids as 0, 1, 5, 3) from Reservation table. Out of these selected entries, the subquery returns Non-Null values only for 1 and 3.

17. Which of the following concurrency control protocols ensure both conflict serialzability and freedom from deadlock? **(GATE 2010)**

I. 2-phase locking

II. Time-stamp ordering

A. I only B. II only

C. Both I and II D. Neither I nor II

Explanation:

Two phase locking is a concurrency control method that guarantees Serialisability. The protocol utilises locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life. 2PL may be lead to deadlocks that result from the mutual blocking of two or more transactions. See the following situation, neither T3 nor T4 can make progress.

T_3	T_4
lock-X (B)	
read (B)	
$B := B - 50$	
write (B)	
	lock-S (A)
	read (A)
	lock-S (B)
lock-X(A)	

Timestamp based concurrency control algorithm is a non-lock concurrency control method. In Timestamp based method, deadlock cannot occur as no transaction ever waits.

18. Consider the following schedule for transactions T1, T2 and T3: **(GATE 2010)**

T1	T2	T3
Read (*X*)		
	Read (*Y*)	
		Read (*Y*)
	Write (*Y*)	
Write (*X*)		
		Write (*X*)
	Read (*X*)	
	Write (*X*)	

Which one of the schedules below is the correct serialization of the above?

A. T1 T3 T2 B. T2 T1 T3

C. T2 T3 T1 D. T3 T1 T2

Explanation:

T1 can complete before T2 and T3 as there is no conflict between Write(X) of T1 and the operations in T2 and T3 which occur before Write(X) of T1 in the above diagram.

T3 should can complete before T2 as the Read(Y) of T3 doesn't conflict with Read(Y) of T2. Similarly, Write(X) of T3 doesn't conflict with Read(Y) and Write(Y) operations of T2.

Another way to solve this question is to create a dependency graph and topologically sort the dependency graph. After topologically sorting, we can see the sequence T1, T3, T2.

19. The following functional dependencies hold for relations R(A, B, C) and S(B, D, E):

$B \rightarrow A$,

$A \rightarrow C$

The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural join $R \rhd\lhd S$ (R natural join S) **(GATE 2010)**

A. 100 B. 200

D. 300 D. 2000

Explanation:

From the given set of functional dependencies, it can be observed that B is a candidate key of R. So all 200 values of B must be unique in R. There is no functional dependency given for S. To get the maximum number of tuples in output, there can be two possibilities for S.

(1) All 100 values of B in S are same and there is an entry in R that matches with this value. In this case, we get 100 tuples in output.

(2) All 100 values of B in S are different and these values are present in R also. In this case also, we get 100 tuples.

20. Consider two transactions T1 and T2, and four schedules S1, S2, S3, S4 of T1 and T2 as given below: **(GATE 2009)**

T1 = R1[X] W1[X] W1[Y]

T2 = R2[X] R2[Y] W2[Y]

S1 = R1[X] R2[X] R2[Y] W1[X] W1[Y] W2[Y]

S2 = R1[X] R2[X] R2[Y] W1[X] W2[Y] W1[Y]

S3 = R1[X] W1[X] R2[X] W1[Y] R2[Y] W2[Y]

S1 = R1[X] R2[Y]R2[X]W1[X] W1[Y] W2[Y]

Which of the above schedules are conflict-serialisable?

A. S1 and S2 B. S2 and S3

C. S3 only D. S4 only

Explanation:

There can be two possible serial schedules T1 T2 and T2 T1. The serial schedule T1 T2 has the following sequence of operations

R1[X] W1[X] W1[Y] R2[X] R2[Y] W2[Y]

And the schedule T2 T1 has the following sequence of operations.

R2[X] R2[Y] W2[Y] R1[X] W1[X] W1[Y]

The Schedule S2 is conflict-equivalent to T2 T1 and S3 is conflict-equivalent to T1 T2.

21. Let R and S be relational schemes such that R={a,b,c} and S={c}. Now consider the following queries on the database: **(GATE 2009)**

I. $\pi_{R-S}(r) - \pi_{R-S}(\pi_{R-S}(r) \times S - \pi_{R-S, S}(r))$

II. $\{t | t \epsilon \pi_{R-S}(r) \wedge \forall u \epsilon_r (\exists v \epsilon s (u = v[s] \wedge t = v[R-S]))\}$

III. $\{t | t \epsilon \pi_{R-S}(r) \wedge \forall v \epsilon r (\exists u \epsilon s (u = v[s] \wedge t = v[R-S]))\}$

IV. SELECT R.a, R.b
FROM R,S
WHERE R.c = S.c

Which of the above queries are equivalent?

A. I and II B. I and III

C. II and IV D. III and IV

22. Consider the following relational schema:

Suppliers (sid:integer, sname:string, city:string, street:string)

Parts(pid:integer, pname:string, color:string)

Catalog(sid:integer, pid:integer, cost:real)

Consider the following relational query on the above database: **(GATE 2009)**

```
SELECT S.sname
  FROM Suppliers S
    WHERE S.sid NOT IN (SELECT C.sid
        FROM Catalog C
        WHERE C.pid NOT IN (SELECT P.pid
            FROM Parts P
            WHERE P.color<> 'blue'))
```

Assume that relations corresponding to the above schema are not empty. Which one of the following is the correct interpretation of the above query?

A. To find the names of all suppliers who have supplied a non-blue part.

B. To find the names of all suppliers who have not supplied a non-blue part.

C. To find the names of all suppliers who have supplied only blue parts.

D. To find the names of all suppliers who have not supplied only blue parts.

Explanation :

The subquery "*SELECT P.pid FROM Parts P WHERE P.color<> 'blue'*" gives pids of parts which are not blue. The bigger subquery "*SELECT C.sid FROM Catalog C WHERE C.pid NOT IN (SELECT P.pid FROM Parts P WHERE P.color<> 'blue')*" gives sids of all those suppliers who have supplied blue parts. The complete query gives the names of all suppliers who have not supplied a non-blue part

23. Assume that, in the suppliers relation above, each supplier and each street within a city has a unique name, and (sname, city) forms a candidate key. No other functional dependencies are implied other than those implied by primary and candidate keys. Which one of the following is true about the above schema? **(GATE 2009)**

A. The schema is in BCNF

B. The schema is in 3NF but not in BCNF

C. The schema is in 2NF but not in 3NF

D. The schema is not in 2NF

Explanation :

The schema is in BCNF as all attributes depend only on a superkey (Note that primary and candidate keys are also superkeys).

24. Let R and S be two relations with the following schema **(GATE 2008)**

R (P, Q, R1, R2, R3)

S (P, Q, S1, S2)

Where {P, Q} is the key for both schemas. Which of the following queries are equivalent?

I. $\Pi_P (R \bowtie S)$

II. $\Pi_P (R) \bowtie \Pi_P (S)$

III. $\Pi_P (\Pi_{P,Q} (R) \cap \Pi_{P,Q} (S))$

IV. $\Pi_P (\Pi_{P,Q} (R) - (\Pi_{P,Q} (R) - (\Pi_{P,Q} (S)))$

A. Only I and II B. Only I and III

C. Only I, II and III D. Only I, III and IV

Explanation:

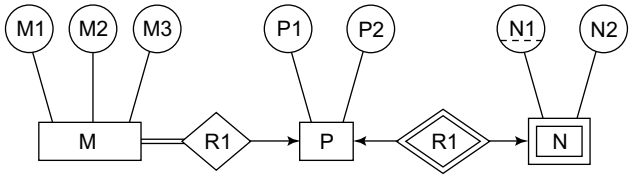
In I, Ps from natural join of R and S are selected.

In III, all Ps from intersection of (P, Q) pairs present in R and S.

IV is also equivalent to III because (R–(R–S)) = R ∩ S.

II is not equivalent as it may also include Ps where Qs are not same in R and S.

25. Consider the following ER diagram. **(GATE 2008)**



The minimum number of tables needed to represent M, N, P, R1, R2 is

A. 2 B. 3 C. 4 D. 5

Explanation:

Many-to-one and one-to-many relationship sets that the total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side. Since R1 is many to one and participation of M is total, M and R1 can be combined to form the table {M1, M2, M3, P1}. N is a week entity set, so it can be combined with P.

26. Which of the following is a correct attribute set for one of the tables for the correct answer to the above question? **(GATE 2008)**

A. {M1, M2, M3, P1} B. {M1, P1, N1, N2}

C. {M1, P1, N1} D. {M1, P1}

27. Consider the following relational schemes for a library database: **(GATE 2008)**

Book (Title, Author, Catalog_no, Publisher, Year, Price)

Collection (Title, Author, Catalog_no)

within the following functional dependencies:

I. Title Author → Catalog_no

II. Catalog_no → Title Author Publisher Year

III. Publisher Title Year → Price

Assume {Author, Title} is the key for both schemes. Which of the following statements is true?

(GATE 2008)

A. Both Book and Collection are in BCNF

B. Both Book and Collection are in 3NF only

C. Book is in 2NF and Collection is in 3NF

D. Both Book and Collection are in 2NF only

Explanation:

Table Collection is in BCNF as there is only one functional dependency "Title Author → Catalog_no" and {Author, Title} is key for collection. Book is not in BCNF because Catalog_no is not a key and there is a functional dependency "Catalog_no → Title Author Publisher Year". Book is not in 3NF because non-prime attributes (Publisher Year) are transitively dependent on key [Title, Author]. Book is in 2NF because every non-prime attribute of the table is either dependent on the key [Title, Author], or on another non prime attribute.

28. Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a

1. T1 start
2. T1 B old=12000 new=10000
3. T1 M old=0 new=2000
4. T1 commit
5. T2 start
6. T2 B old=10000 new=10500
7. T2 commit

Suppose the database system crashes just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure? mortagage payment and then apply a 5% interest. **(GATE 2006)**

A. We must redo log record 6 to set B to 10500

B. We must undo log record 6 to set B to 10000 and then redo log records 2 and 3

C. We need not redo log records 2 and 3 because transaction T1 has committed

D. We can apply redo and undo operations in arbitrary order because they are idempotent.

Explanation :

Once a transaction is committed, no need to redo or undo operations.

29. Consider the relation enrolled (student, course) in which (student, course) is the primary key, and the relation paid (student, amount) where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Given the following four queries:

Query1: select student from enrolled where student in (select student from paid)

Query2: select student from paid where student in (select student from enrolled)

Query3: select E.student from enrolled E, paid P where E.student = P.student

Query4: select student from paid where exists

(select * from enrolled where enrolled.student = paid. student)

Which one of the following statements is correct?

(GATE 2006)

A. All queries return identical row sets for any database

B. Query2 and Query4 return identical row sets for all databases but there exist databases for which Query1 and Query2 return different row sets.

C. There exist databases for which Query3 returns strictly fewer rows than Query2.

D. There exist databases for which Query4 will encounter an integrity violation at runtime.

Explanation:

The output of Query2, Query3 and Query4 will be identical. Query1 may produce duplicate rows. But rowset produced by all of them will be same.

Table enrolled

student course

---

abc c1

xyz c1

abc c2

pqr c1

Table paid

student amount

---

abc 20000

xyz 10000

rst 10000

Output of Query 1

abc

abc

xyz

Output of Query 2

abc

xyz

Output of Query 3

abc

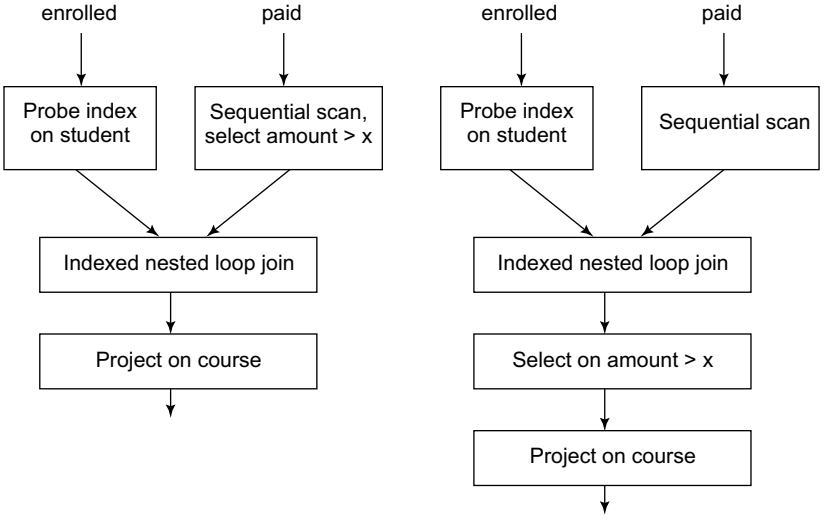
xyz

Output of Query 4

abc

xyz

30. Consider the relation enrolled (student, course) in which (student, course) is the primary key, and the relation paid (student, amount), where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Assume that amounts 6000, 7000, 8000, 9000 and 10000 were each paid by 20% of the students. Consider these query plans (Plan 1 on left, Plan 2 on right) to "list all courses taken by students who have paid more than x". **(GATE 2006)**



A disk seek takes 4ms, disk data transfer bandwidth is 300 MB/s and checking a tuple to see if amount is greater than x takes 10 micro-seconds. Which of the following statements is correct?

A. Plan 1 and Plan 2 will not output identical row sets for all databases.
B. A course may be listed more than once in the output of Plan 1 for some databases
C. For x = 5000, Plan 1 executes faster than Plan 2 for all databases.
D. For x = 9000, Plan I executes slower than Plan 2 for all databases.

Explanation :

Assuming that large enough memory is available for all data needed. Both plans need to load both tables courses and enrolled. So disk access time is same for both plans.

Plan 2 does lesser number of comparisons compared to plan 1.

1. Join operation will require more comparisons as the second table will have more rows in plan 2 compared to plan 1.
2. The joined table of two tables will will have more rows, so more comparisons are needed to find amounts greater than x.

31. The following functional dependencies are given: **(GATE 2006)**

$AB \rightarrow CD$, $AF \rightarrow D$, $DE \rightarrow F$, $C \rightarrow G$, $F \rightarrow E$, $G \rightarrow A$

Which one of the following options is false?

A. CF+ = {ACDEFG}
B. BG+ = {ABCDG}
C. AF+ = {ACDEFG}
D. AB+ = {ABCDFG}

Explanation :

Closure of AF or AF+ = {ADEF}, closure of AF doesn't contain C and G.

Option (D) Also looks correct. AB+ = {ABCDG}, closure of AB doesn't contain F.

32. Which one of the following statements about normal forms is false? **(GATE 2005)**

A. BCNF is stricter than 3NF
B. Lossless, dependency-preserving decomposition into 3NF is always possible
C. Lossless, dependency-preserving decomposition into BCNF is always possible
D. Any relation with two attributes is in BCNF

Explanation :

It is not always possible to decompose a table in BCNF and preserve dependencies. For example, a set of functional dependencies $\{AB \rightarrow C, C \rightarrow B\}$ cannot be decomposed in BCNF. See this for more details.

33. The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade. **(GATE 2005)**

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted is:

A. (3,4) and (6,4)
B. (5,2) and (7,2)
C. (5,2), (7,2) and (9,5)
D. (3,4), (4,3) and (6,4)

Explanation :

When (2,4) is deleted. Since C is a foreign key referring A with delete on cascade, all entries with value 2 in C must be deleted. So (5, 2) and (7, 2) are deleted. As a result of this 5 and 7 are deleted from A which causes (9, 5) to be deleted.

34. The relation book (<u>title</u>, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list? **(GATE 2005)**

```
select title
from book as B
where (select count(*)
    from book as T
    where T.price > B.price) < 5
```

A. Titles of the four most expensive books
B. Title of the fifth most inexpensive book
C. Title of the fifth most expensive book
D. Titles of the five most expensive books

Explanation:

When a subquery uses values from outer query, the subquery is called correlated subquery. The correlated subquery is evaluated once for each row processed by the outer query.

The outer query selects all titles from book table. For every selected book, the subquery returns count of those books which are more expensive than the se-

lected book. The where clause of outer query will be true for 5 most expensive books. For example count (*) will be 0 for the most expensive book and count(*) will be 1 for second most expensive book.

35. Let r be a relation instance with schema R = (A, B, C, D). We define r1 = 'select A,B,C from r' and r2 = 'select A, D from r'. Let s = r1 * r2 where * denotes natural join. Given that the decomposition of r into r1 and r2 is lossy, which one of the following is true? **(GATE 2005)**

A. s is subset of r B. r U s = r
C. r is a subset of s D. r * s = s

Explanation:

Consider the following example with lossy decomposition of r into r1 and r2. We can see that r is a subset of s.

Table r

A	B	C	D
1	10	100	1000
1	20	200	1000
1	20	200	1001

Table r1

A	B	C
1	10	100
1	20	200

Table r2

A	D
1	1000
1	1001

Table s (natural join of r1 and r2)

A	B	C	D
1	10	100	1000
1	20	200	1000
1	20	100	1001
1	20	200	1001

36. Let E1 and E2 be two entities in an E/R diagram with simple single-valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one-to-many and R2 is many-to-many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model? **(GATE 2005)**

A. 2 B. 3
C. 4 D. 5

Explanation:

The situation given can be expressed with following sample data.

E1
a
b
c

E2
x
y
z

R1

E1	E2
a	x
a	y
b	z

R2

E1	E2
a	x
a	y
b	y

37. Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A→B, BC → D, E → C, D → A}. What are the candidate keys of R? **(GATE 2005)**

A. AE, BE B. AE, BE, DE
C. AEH, BEH, BCH D. AEH, BEH, DEH

Explanation :

A set of attributes S is candidate key of relation R if the closure of S is all attributes of R and there is no subset of S whose closure is all attributes of R.

Closure of AEH, i.e. AEH+ = {ABCDEH}

Closure of BEH, i.e. BEH+ = {ABCDEH}

Closure of DEH, i.e. DEH+ = {ABCDEH}

ANSWER KEY

1. C	**2.** A	**3.** B	**4.** A
5. B	**6.** C	**7.** A	**8.** C
9. B	**10.** A	**11.** B	**12.** A
13. C	**14.** C	**15.** A	**16.** C
17. B	**18.** A	**19.** A	**20.** B
21. A	**22.** B	**23.** A	**24.** D
25. A	**26.** A	**27.** C	**28.** C
29. A	**30.** C	**31.** C	**32.** C
33. C	**34.** D	**35.** C	**36.** C
37. D			