Page No.
Date: / /

Page No.
Date: / /

www.ankurgupta.net

Ankur Gupta

(DBMS)

Database Management Systems

These notes have been prepared from the following books:-

(1) Fundamentals of Database Systems
by Ramez Elmasri, Shamkant B. Navathe

(2) Database Systems Concepts
by Henry F. Korth, Abraham Silberschatz, S. Sudarshan

Kindly refer above books for more details.

Page No.
Date: / /

www.ankurgupta.net

Introduction

Page No.
Date: / /

DBMS :-

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database.

Characteristics of the Database Approach :-

The main characteristics of the database approach are the following :-

(1) Self describing nature of a database system.

(2) Insulation between programs and data, and data abstraction.

(3) Support of multiple views of the data.

(4) Sharing of data and multiuser transaction processing.

Advantages of using the DBMS Approach:-

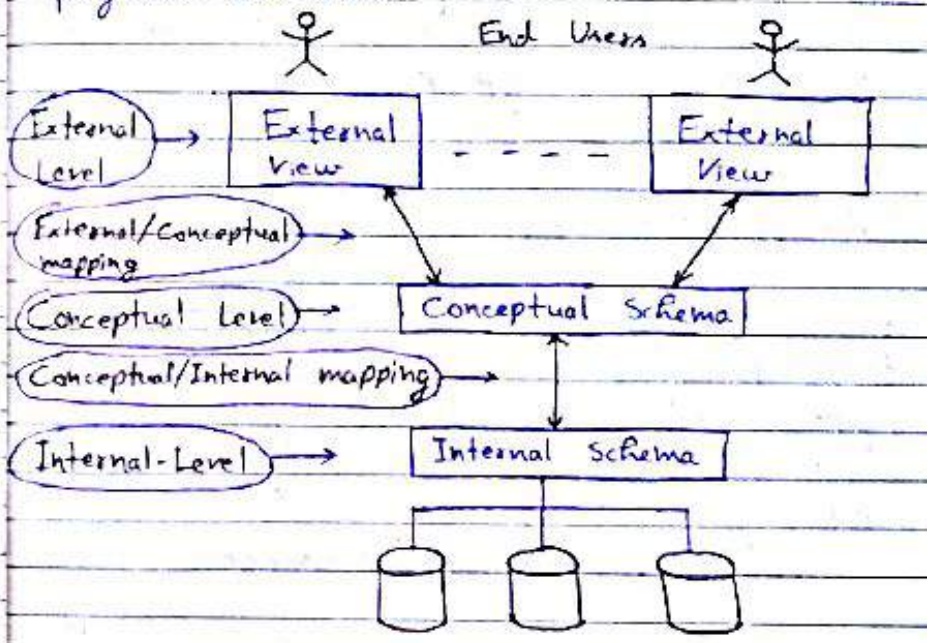
(1) Controlling Redundancy
(2) Restricting Unauthorized Access
(3) Providing storage structure for efficient query processing.
(4) Providing backup and recovery
(5) Providing multiple user interfaces.
(6) Representing complex relationships among data.
(7) Enforcing Integrity Constraints.

Deductive Database Systems:-

Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called deductive database systems.

The three-schema architecture:-

The goal of the three-schema architecture is to seperate the user-applications and the physical database.



(1) The internal level has an internal schema, which describes the physical storage structure of the database.
(2) The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users.

(3) The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

Data Independence :-

Data Independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

(1) **Logical data independence :-** Capacity to change the conceptual schema without having to change external schemas or user programs

(2) **Physical data independence :-** The capacity to change the internal schema without having to change the conceptual schema.

DBMS Languages :-

(i) DDL (Data Definition Language) is used if there is no strict seperation is maintained between Conceptual Schema and Internal Schema.

(ii) SDL (Storage Definition Language) is used to specify the Internal Schema.

Database Design Techniques :-

(1) **Top-down approach :-** We start defining the data set and then we go on defining data elements in those sets.

→ Entity - Relationship Modeling.

(2) **Bottom-Up Approach :-** We start defining required attribute first and then group those attributes to form the entities.

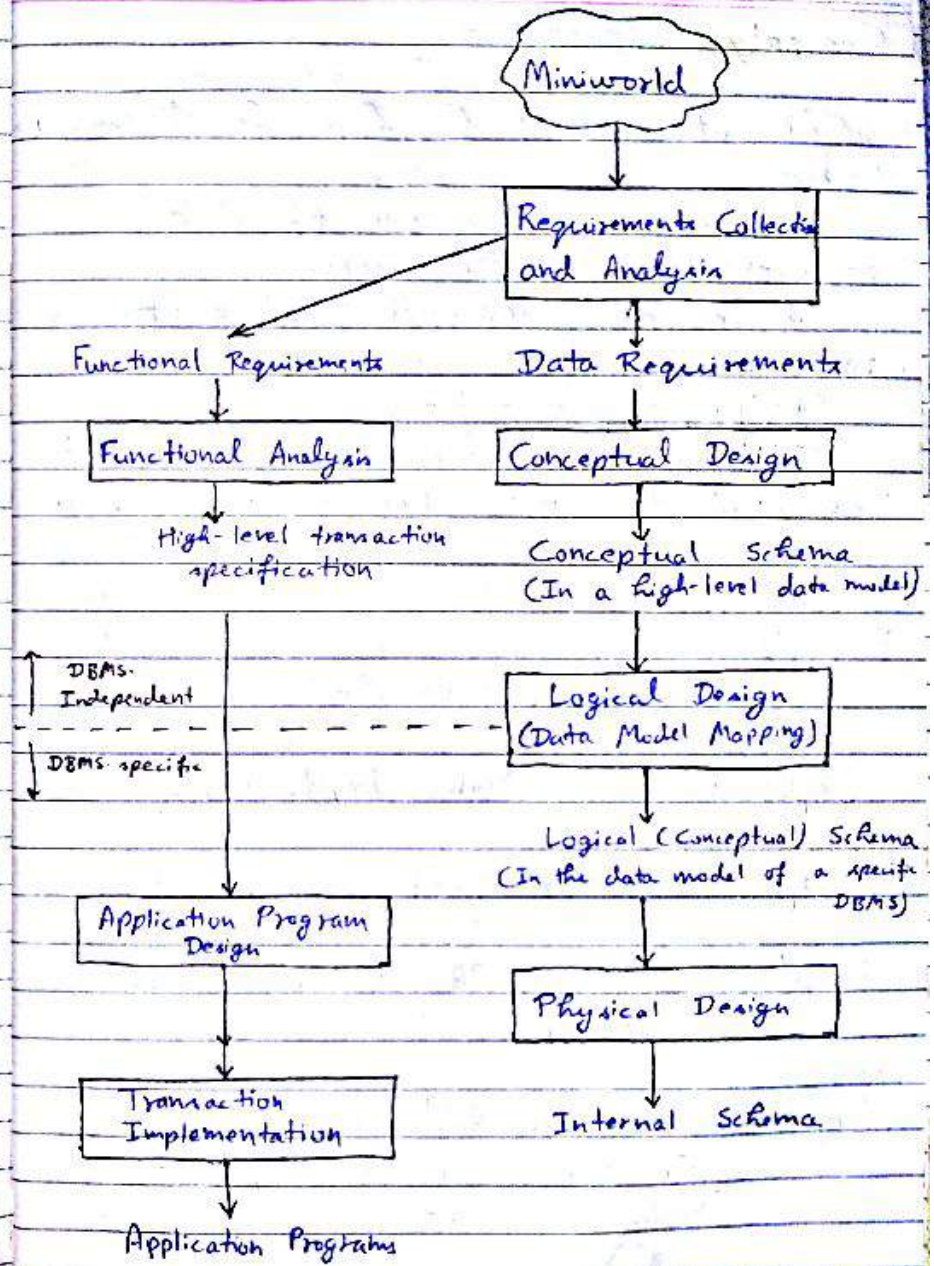
→ Normalization.

Page No.
Date: / /

www.ankurgupta.net

Conceptual Design

Page No.
Date: / /



Page No.
Date: / /

Conceptual Schema :-

(1) Outcome of the high level conceptual design.
(2) Concise description of data requirements of the users.
(3) Includes description of entity types relationships and constraints.
(4) No implementation details.
(5) Ease of understanding. Used to communicate with non-technical users.

Example :-
Company Database :-

Departments :- Name, Dept-id, Manager, ...

Projects :- Name, Proj-id, location

Employee :- Name, PAN, Address, Salary, Gender, DOB.

Dependents :- Name, Address, Gender, DOB, Relationship with employee

Page No.
Date: / /

Entity :-

An entity represents an object of the real world that has an independent existence.

An entity has attributes / properties that describes it.

Attribute Types :-

(1) Simple vs Composite :-

A composite attribute is made of one or more simple or composite attributes.
Example :- Name is made of first name and last name.

(2) Single valued vs Multivalued :-

A single valued attribute is described by one value (E.g. :- Age of a person). A multi-valued attribute is described by many values (E.g. :- The color of a multi colored bird)

(3) Stored vs. Derived :-

An attribute whose value can be inferred from the value of other attributes is called a derived attribute. (E.g :- age can be derived from date of birth and current date). If this is not the case, then it is a stored attribute.

(4) Null Values :-

Attributes that do not have any applicable values are said to have null values. Different from 0, unknown and missing values.

(5) Complex Attributes :-

Combination of composite and multivalued attributes. For example :- A person can have more than one address and each address can have different parts.

Entity types and entity sets :-

An entity type defines a collection of different entities having the same type.

Example :- "Department" defined earlier is an entity type defining several departments having the same attribute structure.

Any specific collection of entities of a particular type is called an Entity Set.

An entity type describes the schema or intension for an entity set.

Association :-

An association b/w two attributes indicates that the values of the associated attributes are interdependent.

Association b/w the attributes of an entity is called Attribute Association.

Association b/w entities is called a Relationship.

Page No.
Date: / /

Key Attributes :-

→ An attribute or a set of attributes that uniquely identify entities in an entity set are called **Key Attributes**.

→ When a collection of attributes define the key, the set is called a **composite key**.

→ Composite keys should be minimal. No subset of a composite key should be a key itself.

→ Key attributes are shown underlined in the E-R - Diagram.

→ Key attributes should retain the unique definition property for all extensions (EER model) of the entity type.

→ There may be more than one key in a relation. All such keys are known as **Candidate Keys**. One of the candidate keys is chosen as the **primary key**, the others are known as **Alternate Keys**.

Page No.
Date: / /

→ An attribute that forms part of a candidate key of a relation is called a **prime attribute**.

→ Some entity types may have no key attributes. Entities of such types are called **weak entities**. Others are called **strong entities**.

Domains of Attributes :-

The domain of an attribute is the set of all valid values that the attribute can have.

Example :-

The domain of the attribute "age" of an employee is the set of all integers b/w 18 and 65.

If the composite attribute can have n values with domains $D_1, D_2, - - D_n$ then the domain of the attribute is :-

$$D_1 \times D_2 \times - - \times D_n$$

Relationships :-

Relationship types relate two (one) or more entity types and defines a relationship set. Any given depiction of a relationship type is called a relationship instance.

A relationship type R among n entity types $E_1, E_2, \ldots E_n$, defines a set of associations (a relationship set) among entities of this type.

Formally :-

$$R \subseteq E_1 \times E_2 \times \ldots \times E_n$$

A relationship instance is any $r_i \in R_i$, where $r_i = (e_1, e_2, \ldots, e_n)$, such that :-

$$e_1 \in E_1,\ e_2 \in E_2,\ \ldots,\ e_n \in E_n$$

Degree of a relationship type :-

The number of participating entity types in this relationship type.

Ex :- Degree of the relationship department "managed by" Employee is 2

Unary, Binary and Ternary relationships are special terms for relationships of degree 1, 2 and 3 respectively.

Relationships vs Attributes :-

Relationships as attributes are a concept used in data models called Functional Data Model.

In object databases - relationships are represented by object refrences.

In relational databases, foreign keys represent relationships as attributes.

Note :- A one-to-one relationship b/w two entity sets does not imply that for an occurance of an entity from one set at any time, there must be an occurance of an entity in the other set.

Page No.
Date: / /

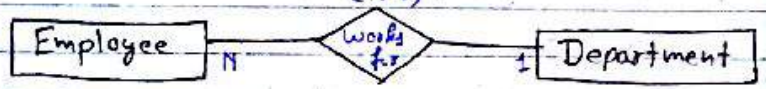
Constraints on Relationship types:-

Constraints limit the set of possible combinations of entities that can participate in the relationship type.

Two main kinds of constraints:-
(a) Cardinality Ratios
(b) Participation Constraints.
(c) Structural Constraints ((a) & (b) taken together).

Cardinality Ratio:-

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



Represents a cardinality constraint that while a department may have any number (N) of employees, each employee entity should be associated with only one department instance.

Participation Constraint:-

If every project has to be associated with a department, then the existence of an instance of Project entity type depends on the existence of an instance of the Handles relationship type.

The entity type Project is said to "**totally participate**" in the relationship type Handles. The relationship is also said to introduce an "**existence dependency**" on project.

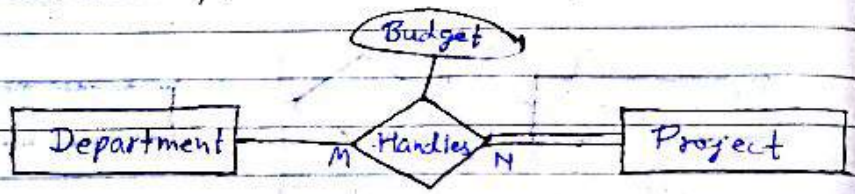
The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.

This constraint specifies the **minimum number of relationship instances** that each entity can participate in and is sometimes called the **minimum cardinality constraint.**

Two types:-
(i) Total Participation
(ii) Partial Participation

Page No.
Date: / /

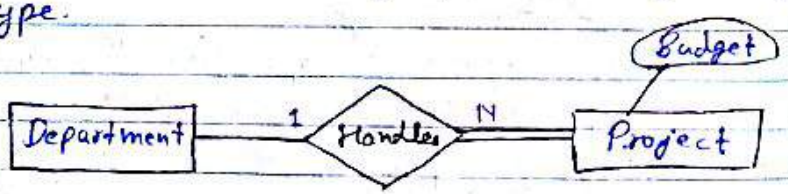
Attributes of relationship types :-



Budget refers to the allocated budget for a specific department for handling a specific project.

It belongs neither to the department nor to the project in its entirety.

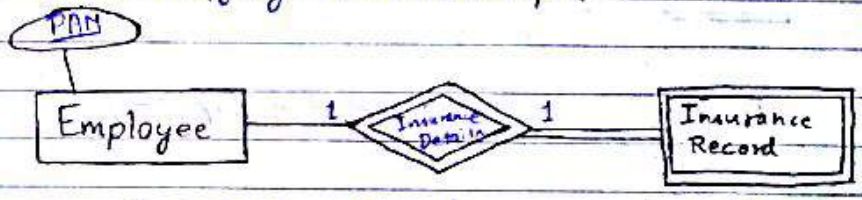
In 1:N or 1:1 relationships, relationship attributes can be moved to a corresponding participating entity type.



Page No.
Date: / /

Identifying Relationships :-

The relationship type that relates a weak entity type to its owner is called the identifying relationship.



Insurance record is a "weak entity type" with no key. It has no existence without its association with entity type Employee.

The relationship insurance details is an "identifying relationship."

Recursive Relationship :-

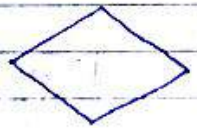
A relationship type, where the same entity type participates more than once in a different roles, is called Recursive Relationship.

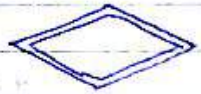
Page No.
Date: / /

Summary of ER-Notations :-

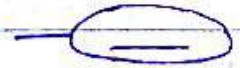

→ Entity type


→ Weak Entity Type

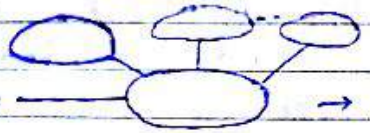

→ Relationship type


→ Identifying Relationship


→ Attribute

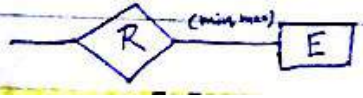

→ Key Attribute


→ Multivalued Attribute


→ Composite Attribute


→ Derived Attribute

→ Total participation of E_2 in R



→ Structural Constraint (min, max) on participation of E in R

Page No.
Date: / /

Enhanced Entity-Relationship (EER)

Subclasses and Inheritance :-

An entity class B is said to be a subclass of another entity class A, if it shares an "is-a" relationship with A.

Example-1 :- Car "is-a" vehicle.
Example-2 :- Manager "is-a" Employee.

An entity of class B is said to be a specialization of entities of class A. Conversely, entities of class A are generalizations of class B entities.

An entity can not exist in the database that merely belongs to a subclass. It has to belong to the super class as well.

~~Superclasses undergo~~
Subclasses undergo type inheritance of the super class. That is, each member of the subclass has the same attributes as the super class entities and participates in the same relationship types.

Page No
Date: / /

Example:-

⇒ Depicts the relation Manager "is-a" Employee
⇒ Manager inherits PAN as key attribute.
⇒ Other attributes in addition to PAN may form the key for Manager Entity.

Specialization and Generalization:-

The process of taking two or more

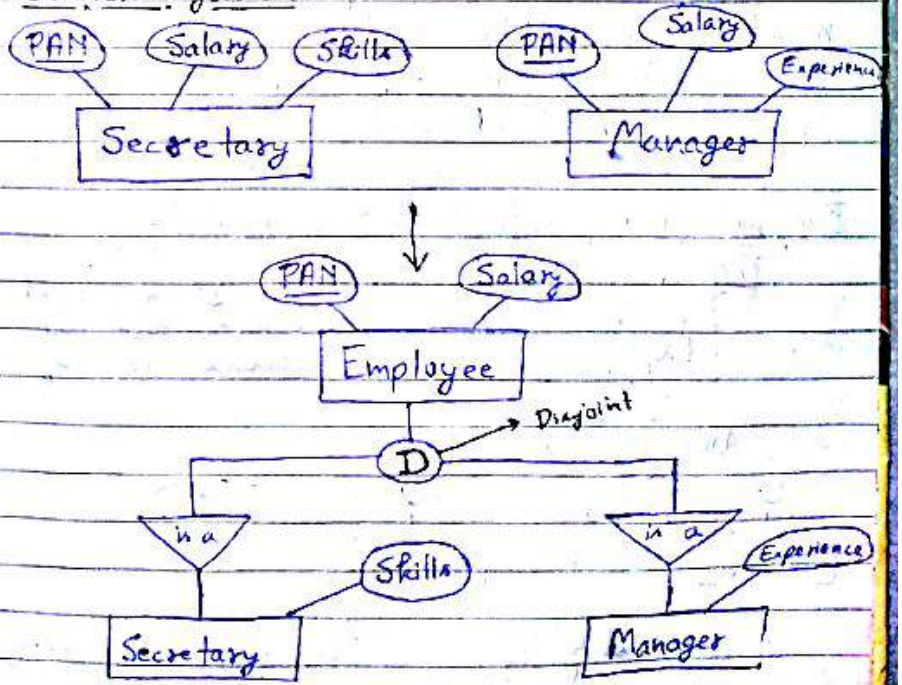
Page No.
Date: / /

Specialization and Generalization:-

The process of creating subclasses out of a given entity type is called specialization.

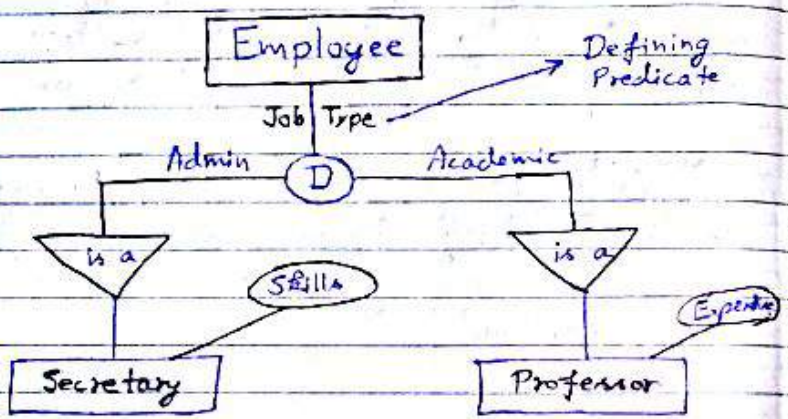
The reverse process of taking two or more entity types and clubbing them under a common super class is called generalization.

Generalization:-



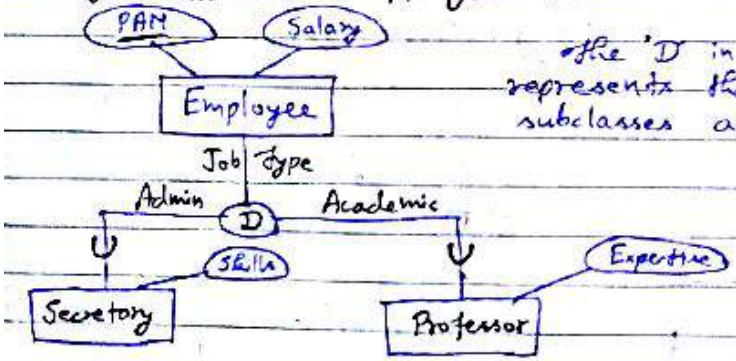
Creation of Generalized Class

Predicate Defined Subclasses:-



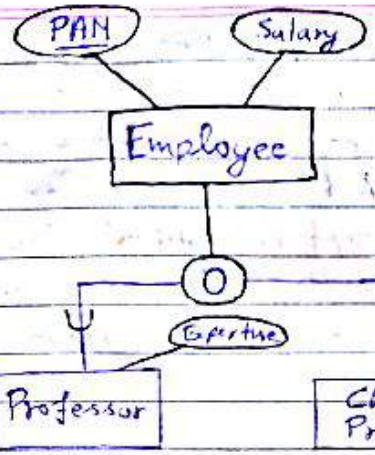
Subclasses entity types whoose members can be defined based on the value of an attribute are called 'predicate-defined' subclasses.

Disjoint and Overlapping Subclasses:-



The 'D' in the circle represents that the subclasses are disjoint

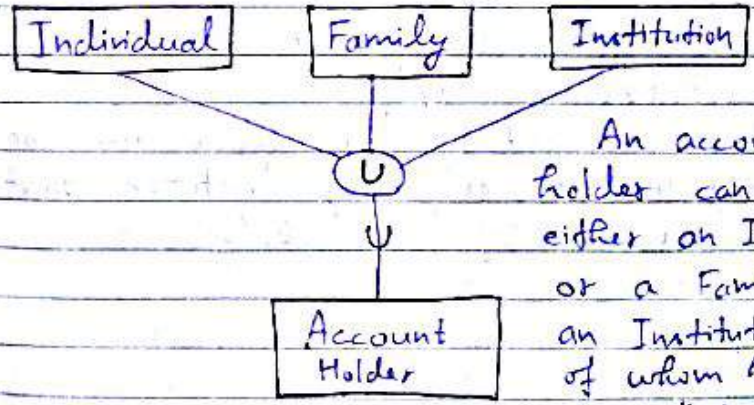
An entity of type Employee may belong to at most one of the subclasses.



Subclasses that are not disjoint are said to overlap. This is represented by an "O" in the inheritance circle.

Disjoint or Overlap may be either partial or total.

Union Types or Categories:-



An account holder can be either an Individual or a Family or an Institution, each of whom have their own attributes and keys.

A Union type is also called a category.

In the previous example, not every individual, family or institution listed in the database may be an account holder.

In such cases, Account Holder is said to be a "partial union". Usually a constraint is specified that determines which entity participates in the union.

Reasons for using the concept of Specialization

(1) Some attributes may apply to some but not all entities of the superclass. A subclass is defined in order to group the entities to which these attributes apply.

(2) Some relationship types may be participated in only by entities that are members of the subclass.

Specialization and Generalization Hierarchies and Lattices:-

A subclass may have further subclasses specified on it, forming a hierarchy or a lattice of specializations.

(Structural Constraint)

Page No.
Date: / /

Alternative Notations for ER Diagram:-

This notation involves associating a pair of integers (min, max) with each participation of an entity type E in a relationship type R, where $0 \leq min \leq max$ and $max \geq 1$

The numbers mean that, for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point of time.

Here each employee must work in exactly one department and each department should have minimum 4 employees.

Here, min = 0 implies partial participation and min > 0 implies total participation.

Page No.
Date: / /

www.ankurgupta.net

Relational Model

Page No.
Date: / /

The relational model, like all data models, consists of three basic components:-

(1) A set of domains and a set of relations.

(2) Operation on relations

(3) Integrity rules.

Background:-

→ Introduced by Ted Codd of IBM Research in 1970.

→ Concept of mathematical relation as the underline basis

→ The standard database model for most transactional database today.

Relational Model Concepts:-

→ A relational database is a collection of relation.

→ A relation resembles a "Table" or a "Flat File" of records

→ Each row or a relational instance is called a tuple and each column is called attribute of the relation.

Some Definitions:-

→ A data value is said to be atomic if it can not be subdivided into smaller data values.

→ A Domain is a set of atomic values.

→ A **Relational Schema** denoted by $R(A_1, A_2 \ldots A_n)$ is made up of relation name R and a list of attributes $A_1, A_2, \ldots A_n$. Each A_i is the name of role played by some domain D in the schema R. The domain of A_i is denoted by $dom(A_i)$.

→ The **Cardinality** of the relation is the number of tuples of the relation.

→ The **Degree** of the relation is the number of attributes of its relation schema. It is also called as **Arity**.

→ A relation r of a relation schema $R(A_1, A_2 \ldots A_n)$ is a set of n-tuples of the form $[t_1, t_2 \ldots, t_n]$ where each $t_i \in dom(A_i)$.

→ The Relation can be defined as:-

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n))$$

where $\times$ is the cartesian product over sets.

Characteristics of Relation:-

(i) **Ordering of Tuples:-**

Mathematically tuples in relation don't have any ordering (although in reality they do have order when stored in files or disks).

(ii) **Ordering of attributes:-**

Attributes within a tuple have to maintain order if they are stored as tuples. However a relation need not necessiarily be maintained as a set of ordered tuples as long as a mapping between attribute and value is maintained in each relation instance.

(iii) Values of tuples:-

Each value that makes up a tuple is atomic and can not be further subdivided. This is also called the First Normal Form assumption.

Relational Constraints:-

(1) Domain Constraints:-

Domain Constraints specify that value of each attribute A must be atomic and a member of dom(A).

(2) Key Constraints:-

Given a relational schema $R = (A_i)$, $i = 1, \ldots, n$, a subset of attributes K is called a 'superkey' if the values of attributes in K is distinct for every relation instance. This is, for any two tuples t_i, t_j, $(t_i \neq t_j) \rightarrow (t_i[K] \neq t_j[K])$

A "key" k of a relation is a subset of the superkey, $k \subseteq K$, such that removal of any attribute in k removes the superkey property for k. They are also called "Minimal Superkeys".

A relation schema may have more than one key. Each key is called a candidate key.

Usually one of the candidate key is chosen to uniquely identify tuples in a relation such a key is called primary key for a relation.

(3) Entity Integrity Constraint:-

The primary key of a tuple may never be null.

(4) Referential Integrity Constraint:-

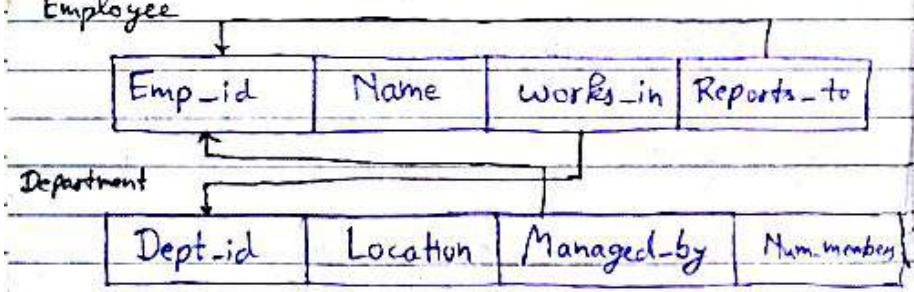
Informally, a tuple that refers to another tuple from another relation should refer to an existing tuple.

In a given relation R_1, a set of attributes FK is said to be a foreign key of R_1 referencing another relation R_2, if the following rules hold:-

(a) The attribute in FK have the same domain as the primary key of R_2.

(b) For every tuple in R_1, the attributes in its foreign key FK, either reference to a tuple in R_2 or is null.

Example:-



Foreign Keys are diagramatically depicted as arrows.

* Foreign key can be from a relation to itself.

(5) Semantic Integrity Constraints:-

Constraints on the values of attributes.

Example:- The minimum age of an employee is 18 and maximum is 65.

Relation Schema (R) → Intension

A relation state (r) of R → Extension.

Page No.
Date: / /

www.ankurgupta.net

Relational Algebra and Relational Calculus

Page No.
Date: / /

Relational Algebra is a collection of operations to manipulate relations.

The result of each of these relations is also a relation.

Relational Algebra is a procedural language. It specifies the operations to be performed on existing relations to derive result relations

The Select Operation:-

The SELECT operation is used to select (retrieve) a subset of the tuples from a relation that matches the "selection condition".

$\sigma_{salary > 3000}(Employee)$

Selects all tuples of relation 'Employee' which matches the criteria:-

Employee . Salary > 3000

Page No.
Date: / /

Properties of 'SELECT' Operation:-

→ The SELECT operation is unary. It operates on only one relation.

→ Selection conditions are applied to each tuple individually in the relation. Hence the condition can not span more than one tuple.

→ The degree of the relation resulting from $\sigma_c(R)$ is same as that of R.

→ $|\sigma_c(R)| \leq |R|$

→ The SELECT operation is commutative:-

$$\sigma_{c_1}(\sigma_{c_2}(R)) \Leftrightarrow \sigma_{c_2}(\sigma_{c_1}(R))$$

Hence Select operation can be cascaded in any manner.

Page No.
Date: / /

The Project Operation:→

The project operation selects specific columns from the specified relation.

$$\Pi_{\text{Name, Salary}}(\text{Employee})$$

returns a relation comprising of only the Employee.Name and Employee.Salary attributes.

The operation is said to have "projected" Employee relation onto the required relation.

Properties of Project Operation:-

→ The Project operation removes duplicate in its results.

→ The number of tuples returned by Project is less than or equal to number of tuples is the specified relation.

→ When attribute list of PROJECT includes the super key, then the number of tuples returned is equal to the number of tuples in the specified relation.

→ PROJECT is not commutative.

$\Pi_{l1}\Pi_{l2}(R) = \Pi_{l}(R)$ if $l1$ is a substring of l_2, Else the operation in an incorrect operation.

Page No.
Date: / /

Composition and Assignment :-

→ Since the input and output of the relational operators is a single relation, they can be composed in any fashion with the output of one operation being the input of the other.

$\Pi_{Name, Salary}(\sigma_{Salary>3000}(Employee))$

returns Name and Salary field of all records of Employee, where salary is greater than 3000

→ Results of a query can also be assigned to a name to form a relation by that name.

$Salary\ Statement \leftarrow \Pi_{Name, Salary}(\sigma_{salary>3000}(Employee))$

creates a new relation Salary Statement out of the specified query.

Page No.
Date: / /

Cartesian Join (x) :-

The Join operator allows the combining of two relations to form a single new relation.

Join is basically the cartesian product of the relations followed by a selection operation.

Properties of Cartesian Join :-

→ The cartesian join represents a Canonical Join of two or more relations.

→ If relation R having $|R|$ tuples and relation S having $|S|$ tuples are joined using X, then $|R \times S| = |R| . |S|$

→ Cartesian Join is too efficient for most queries involving multiple relations.

Page No.
Date: / /

Theta Join ($\bowtie_\theta$) :-

Theta join is used to combine related tuples from two or more relations (specified by the condition θ) to form a single tuple.

Consider the Student and Lab relation :-

The relation :-

$$\text{Student} \bowtie_{\text{Student.Lab} = \text{Lab.Name}} \text{Lab}$$

returns the same result as :-

$$\sigma_{\text{Student.Lab} = \text{Lab Name}} (\text{Student} \times \text{Lab})$$

* This is most efficient when Student.Lab is a foreign key into Lab (and Lab Name is a primary key of Lab) and referential integrity is maintained.

* Tuples whose join attributes are null do not appear in the final result.

Page No.
Date: / /

Properties of Theta Join :-

→ Theta join, where the only comparison operator used is the equals (=) sign, are called equijoins.

→ A natural join denoted by * is an equijoin, where the attribute names at both ends of the equality sign are the same. In this case, the attribute is not repeated in the final result.

Consider the Student and the Lab example. Let the Student relation schema is modified as Student (RollNo, Name, LabName) and Lab relation schema be modified as Lab (LabName, Faculty, Dept)

The natural join Student * Lab has a schema :-

(RollNo, Name, LabName, Faculty, Dept)

Renaming Operator :-

$TA(id, LabName) \leftarrow \Pi_{RollNo, Lab}(Student)$

The above relation projects RollNo and Lab attributes from Students and renames them as id and LabName respectively in the output relation. It also names the output relation as "TA".

The above can also be achieved by "rename" (ρ) operator as :-

$\rho_{TA(id, LabName)}(\Pi_{RollNo, Lab}(Student))$

→ The general form of the rename operator is as follows :-

$$\rho_{S(S_1, S_2, \ldots S_n)}(R)$$

or

$$\rho_S(R)$$

or

$$\rho_{(S1, S2, \ldots, Sn)}(R)$$

The first form renames both relation name and attribute names, the second form renames only the relation name and the third form renames only attribute names

Set Theoretic Operations :-

Set theoretic operation like UNION($\cup$) INTERSECTION($\cap$) and SET DIFFERENCE can be applied to compatible relations.

Compatibility :- Two relations $R(A_1, A_2, \ldots A_n)$ and $S(B_1, B_2, \ldots B_n)$ are compatible if they have the same number of attributes and for all A_i, B_i, $1 \leq i \leq n$, $dom(A_i) = dom(B_i)$

→ The Union operator $R \cup S$ returns the set of all tuples that are present in either R or S or both. Duplicate tuples will be eliminated.

→ The Intersection operator $R \cap S$ returns the set of all tuples that are present in both R and S.

→ The Set Difference Operator $R - S$ returns the set of all tuples in R but not in S.

$$R \cup S = S \cup R$$
$$R \cap S = S \cap R$$
$$R - S \neq S - R$$

Page No.
Date: / /

The Division Operator :-

The Division Operator (÷) is used to denote the conditions where a given relation R is to be split based on whether its association with every tuple in an another relation S.

Let the set of attributes of R be Z and the set of attributes of S be X. Let $Y = Z - X$, that is the set of all attributes of R not in S.

The result of the operation :-

$T(Y) = R(Z) \div S(X)$ is as follows:-

R

A	B
a_1	b_1
a_2	b_1
a_2	b_1
a_1	b_2
a_3	b_2
a_1	b_3
a_2	b_2

S

A
a_1
a_2

T = R ÷ S

B
b_1
b_3

Page No.
Date: / /

Other Relational Operators :-

Left outer join ⟕
Right outer join ⟖
Full outer join ⟗

Outer Join :-

A normal join operation ⋈ requires referential integrity. Every attribute in R that refers S should refer to an existing tuple in S. If the referencing attribute of S is null, the tuple is not returned.

Outer join is used when all tuples are required even when referential integrity is not met. Left outer join includes tuples even when the referencing attribute is null, and Right outer join includes tuples even when the referenced attribute is null or referenced tuple does not exist.

An outer join which is both left and right is called a Full Outer Join.

Outer Union :-

Outer Union computes the union of partially compatible relations. Two relations R(X) and S(Z) are said to be partially compatible if there exist $W \subseteq X$ and $Y \subseteq Z$ such that for every $A_i \in W$ and $B_i \in Y$, $dom(A_i) = dom(B_i)$.

Page No.
Date: / /

Outer Union of R and S creates a new relation T(X ∪ Z) that includes all attributes from both relation. Null values fill up all undefined attributes.

The "Complete Set" of the Operators:-

The following set of relational operators:- (π, σ, ∪, −, ×) is called the "complete set" of the relational operators because all other operators can be mathematically expressed as a sequence of the above operators

For example:-

$$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$$

$$R \bowtie_{condition} S = \sigma_{condition}(R \times S)$$

Page No.
Date: / /

Relational Calculus:-

Relational Calculus is a query system wherein queries are expressed as variables and formulas on these variables.

A calculus expression specifies what to be retrieved rather than how to retrieve it. Therefore, the relational calculus is considered to be a nonprocedural language.

It has been shown that any retrieval that can be specified in the basic relational algebra can also be specified in the relational calculus, and vice-versa, in other words, the expressive power of the two languages is identical.

Relational Calculus can be categorised in two parts:-

(1) Tuple Calculus
(2) Domain Calculus

Page No.
Date: / /

Tuple Calculus :-

Queries in tuple calculus are expressed by a tuple calculus expression. A tuple calculus expression is of the form :-

$$\{ t \mid COND(t) \}$$

where t is a tuple variable and COND(t) is a conditional expression involving t. The result of such a query is the set of all tuples t that satisfy COND(t).

For example :- to find all employees whoose salary is above Rs 50,000, we can write the following tuple calculus expression :-

$$\{ t \mid Employee(t) \wedge t.salary > 50000 \}$$

To retrive only some of the attributes :-

$$\{ t.fname, t.LName \mid Employee(t) \wedge t.salary > 50000 \}$$

Page No.
Date: / /

Informally, we need to specify the following information in a tuple calculus expression :-

(1) For each tuple variable t, the range relation R of t. This value is specified by a condition of the form R(t).

(2) A condition to select a particular combinations of tuples.

(3) A set of attributes to be retrieved, the requested attributes.

Example :- t.attr_name

Expressions and Formulas in Tuple Relational Calculus :-

A general expression of the tuple relational calculus is of the form :-

$$\{ t_1.A_j, t_2.A_k, \ldots, t_n.A_m \mid COND(t_1, t_2, \ldots, t_n, t_{n+1}, \ldots, t_{n+m}) \}$$

where $t_1, t_2, \ldots t_{n+m}$ are tuple variables, each A_i is an attribute of the relation on which t_i ranges and COND is a condition or formula of the tuple relational calculus. It is also called a well formed formula (wff).

Page No.
Date: / /

A variable appearing in a formula is said to be **free** unless it is quantified by the existential quantifier ($\exists$) or the universal quantifier ($\forall$), otherwise it is said to be **bound**.

Tuple calculus formulas are built from **atoms**. An atom is either of the forms given below:-

(1) An atom of the form $R(t_i)$, where R is a relation name and t_i is a tuple variable.

(2) An atom of the form $t_i.A$ op $t_j.B$ where op is one of the comparison operators in the set $\{=, <, \leq, >, \geq, \neq\}$

(3) An atom of the form $t_i.A$ op c or c op $t_j.B$, where op is one of the comparison operators in the set $\{=, <, \leq, >, \geq, \neq\}$ and c is a constant.

Each of the preceding atoms evaluates to either TRUE or FALSE for a specific combination of tuples, this is called the truth value of an atom.

Page No.
Date: / /

Formulas (wff) are built from atoms using the following rules:-

(1) Every atom is a formula.

(2) If f and g are formulas, then $\neg f$, (f), $f \vee g$, $f \wedge g$, $f \rightarrow g$ are also formulas

(3) If $f(x)$ is a formula where x is free, then $\exists x(f(x))$ and $\forall x(f(x))$ are also formulas, however x is now bound.

The logical implication expression $f \rightarrow g$, meaning if f then g, is equivalent to $\neg f \vee g$.

The formula $\exists x(f(x))$ is TRUE if the formula $f(x)$ evaluates to true for some (at least one) tuple assigned to free occurances of x in $f(x)$, otherwise it is false.

The formula $\forall x(f(x))$ is TRUE if the formula $f(x)$ evaluates evaluates to TRUE for every tuple assigned to free occurances of x in $f(x)$, otherwise it is false.

Transforming the Universal and Existential Quantifiers :-

(1) $(\forall x)(\neg(P(x))) \equiv \neg(\exists x)(P(x))$

(2) $\neg(\forall x)P(x) \equiv (\exists x)(\neg(P(x)))$

(3) $(\forall x)(P(x)) \equiv \neg(\exists x)(\neg(P(x)))$

(4) $(\exists x)(P(x)) \equiv \neg(\forall x)(\neg(P(x)))$

(5) $(\forall x)(P(x) \wedge Q(x)) \equiv \neg(\exists x)(\neg(P(x)) \vee \neg(Q(x)))$

(6) $(\forall x)(P(x) \vee Q(x)) \equiv \neg(\exists x)(\neg(P(x)) \wedge \neg(Q(x)))$

(7) $(\exists x)(P(x) \wedge Q(x)) \equiv \neg(\forall x)(\neg(P(x)) \vee \neg(Q(x)))$

(8) $(\exists x)(P(x) \vee Q(x)) \equiv \neg(\forall x)(\neg(P(x)) \wedge \neg(Q(x)))$

(9) $(\forall x)(P(x)) \rightarrow (\exists x)(P(x))$

(10) $\neg(\exists x)(P(x)) \rightarrow \neg(\forall x)(P(x))$

Safe Expressions :-

A safe expression in relational calculus is one that is guaranted to yield a finite number of tuples as its result, otherwise the expression is called unsafe.

An expression is said to be safe if all values in its result are from the domain of the expression Ex:- $\{t \mid \text{NOT}(\text{Employee}(t))\}$ is unsafe.

Domain Calculus :-

Domain Calculus differs from tuple calculus in the type of variables used in formulas :- Rather than having variables range over tuples, the variables range over single values from domains of attributes. To form a relation of degree n for a query result, we must have n of these domains variables - one for each attribute.

An expression of the domain calculus is of the form :-

$$\{x_1, x_2, \ldots, x_n \mid \text{COND}(x_1, x_2, \ldots x_n, x_{n+1}, \ldots x_{n+m})\}$$

where $x_1, x_2, \ldots, x_n, x_{n+1}, \ldots x_{n+m}$ are domain variables that range over domains and COND is a condition or formula of the domain relational calculus

Note :-

SQL is based on tuple relational calculus, while QBE is related to domain relational calculus. Both of these languages were developed by IBM Research.

SQL - Structured Query Language

QBE - Query by example.

Page No.
Date: / /

A formula is made up of atoms. The atoms of a formula are slightly different from those for the tuple calculus and can be one of the following:-

(1) An atom of the form $R(x_1, x_2, \ldots x_j)$, where R is the name of a relation of degree j and each x_i, $1 \le i \le j$, is a domain variable.

(2) An atom of the form x_i op x_j, where op is one of the comparison operators, and x_i & x_j are domain variables.

(3) An atom of the form x_i op c or c op x_j, where op is a comparison operator, x_i & x_j are domain variables and c is a constant value.

The rules of formulas are same as for tuple calculus.

Rational Completeness:-

Rational completeness means that a query language can express all the queries that can be expressed in relational algebra. It does not mean that the language can express any desired query.

Page No.
Date: / /

Relational Algebra has the same expressive power as safe relational calculus.

Page No.
Date: / /

www.ankurgupta.net

Relational Database Design

Page No.
Date: / /

Informal Design Guidelines for Relation Schemas :-

(1) Semantics of the Relation Attributes :-

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

(2) Reducing the redundant values in tuples :-

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

(3) Reducing the null values in tuples :-

As far as possible, avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

(4) Disallowing the possibility of generating spurious tuples :-

Design relation schemas so that they can be joined together with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not foreign key, primary key combinations, because joining on such attributes may produce spurious tuples.

☝ For more information, refer Navathe.

Some Definitions :-

(1) Redundancy :- Information is repeated across tuples.

(2) Update Anomalies :- If information is repeated across tuples, then ~~any~~ an update of such information has to performed across all tuples containing the information.

(3) Deletion Anomalies :- If information is repeated across tuples, deletion of information has to be performed across all these tuples.

Functional Dependencies :-

→ Framework for systematic design and optimization of relational schemas.

→ Generalization over the notion of keys.

→ Crucial in obtaining correct normalized schemas.

Definition of Functional Dependency :-

In any relation R, if there exists a set of attributes $A_1, A_2, \ldots, A_n$ and an attribute B such that if any two tuples have the same value for $A_1, A_2, \ldots A_n$, then they also have the same value for B.

A functional dependency (FD) of the above form is written as :-

$$A_1, A_2, \ldots, A_n \longrightarrow B$$

Functional dependencies define properties of the schema and not of any particular tuple in the schema.

☝ If $A \rightarrow B$, this does not mean $B \rightarrow A$

If $A_1, A_2, \ldots A_n$ can uniquely determine many attributes, they can all be clubbed together in one expression.

$A_1, A_2, \ldots A_n \longrightarrow B_1$
$A_1, A_2, \ldots A_n \longrightarrow B_2$
$A_1, A_2, \ldots A_n \longrightarrow B_3$
$\vdots$
$A_1, A_2, \ldots A_n \longrightarrow B_m$

$\rightarrow$

$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2 \ldots B_m$

If there is a functional dependency:-

$A \longrightarrow B$

this means that the value of B is determined by the value of A or the value of a A uniquely determines B.

OR

There is a functional dependency from A to B or B is functionally dependent on A.

Example:- Consider the relation:-

Movies (title, year, length, filmType, studio, star)

We can identify some FDs as the following:-

title, year → length
title, year → studio

However note that

title, year → star

may not always be true!

Trivial and Non-trivial Functional Dependency:-

Note that in the Movies relation:-

title, year → title

An FD where the right hand side is contained within the left hand side is called a Trivial FD.

Example:- $X \rightarrow Y$ where $X \supseteq Y$

If there is at least one element on R.H.S. that is not contained in L.H.S., it is called non-trivial, and if none of the elements of R.H.S. are contained in L.H.S., it is called completely non-trivial FD.

Inference Rules for Functional Dependencies:-

→ FDs which are specified are called **stated FDs** and FDs which are derived are called **inferred FDs.**

→ The set of FDs that are specified on relation schema R, is represented by F.

→ We use the notation $F \models X \rightarrow Y$ to denote that the functional dependency $X \rightarrow Y$ is inferred from the set of functional dependencies F.

Inference Rule 1:-
(Reflexive Rule)

If $X \supseteq Y$, then $X \rightarrow Y$

Inference Rule 2:-
(Augmentation Rule)

$\{X \rightarrow Y\} \models XZ \rightarrow YZ$

Inference Rule 3:-
(Transitive Rule)

$\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

Inference Rule 4:-
(Decomposition Rule)

$\{X \rightarrow YZ\} \models X \rightarrow Y, X \rightarrow Z$

Inference Rule 5:-
(Union Rule)

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

Inference Rule 6:-
(Pseudotransitive Rule)

$\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Proof:-

$X \rightarrow Y$	(given)	—(i)
$WY \rightarrow Z$	(given)	—(ii)
$WX \rightarrow WY$	(using IR_2 on 1)	—(iii)
$WX \rightarrow Z$	(using IR_3 on 3 & 2)	—(iv)

Inference Rules IR1 through IR3 are known as Armstrong's Inference Rules.

Specified Functional Dependency:-
Certain FDs can be specified without referring to a specific relation, but as a property of those attributes

Page No.
Date: / /

Closure of FDs :-

The set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F. It is denoted by F^+.

Here F is the set of functional dependencies that are specified on relation schema R.

Example :-

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

then

$F^+ = \{ X \to Y, X \to Z, X \to YZ, X \to X \}$

Armstrong's Axioms (AA) :-

Armstrong showed that Inference Rules (1), (2), and (3) are sound and complete. These are called Armstrong's Axioms (AA).

Soundness :- (Free from defect)

Every new FD $X \to Y$ derived from a given set of FDs F using Armstrong's Axioms is such that :- $F \models \{X \to Y\}$

Completeness :-

Any FD $X \to Y$ logically implied by F (i.e $F \models \{X \to Y\}$) can be derived from F using Armstrong's Axioms.

Page No.
Date: / /

Computing Closures :-

The size of F^+ can sometimes be exponential in the size of F.

For instance,

$F = \{ A \to B_1, A \to B_2, \ldots, A \to B_n \}$

$F^+ = \{ A \to X \}$, where $X \subseteq \{ B_1, B_2, \ldots, B_n \}$

Thus $|F^+| = 2^n$

Since the method of computing F^+ is computationally expensive, however there is an alternative method to check $F \models X \to Y$ without generating F^+.

Checking if $X \to Y \in F^+$ can be done by checking if $Y \subseteq X_F^+$, where X_F^+ is the closure of X under F.

Definition of X_F^+ :-

The closure of X w.r.t. F, written as X_F^+, is the set of all attributes that are "eventually" defined by X.

Let :-

$A \to B$, $B \to C$, D & $BUD \to E$;

then :-

$A^+ \supseteq A \cup B \cup C \cup D \cup E$

Page No.
Date: / /

Now let,

$A' \subseteq A^+$ and

$A' \rightarrow F$, then

$A^+ = A^+ \cup F$

Algorithm to compute X^+ :-

Given a set of functional dependencies F and a set of attributes X, the closure of X under F (X_F^+) can be computed by the following algorithm:-

(1) Initially $X^+ = X$

(2) For every $X' \subseteq X$, if there exists an FD of the form $X' \rightarrow Y$ and $Y \not\subseteq X$, then $X^+ = X^+ \cup Y$

(3) Repeat step 2 untill no more attributes can be added to closure (X).

A functional dependency is a property of the relation schema (intension) R, not of a particular legal relation state (extension) r of R. Hence an FD can not be inferred automatically from a given relation extension r, but must be defined explicitly by someone who knows the semantics of the attributes of R.

Page No.
Date: / /

Covers and Equivalence of FDs :-

Covers :-

A set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F^+, that is, if every dependency in E can be inferred from F, alternatively we can say that E is covered by F.

Equivalence :-

Two set of functional dependencies E and F are equivalent if $E^+ = F^+$. Hence, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions E covers F and F covers E hold.

Determining whether F covers E :-

Calculate X^+ with respect to F for each FD $X \rightarrow Y$ in E, and then checking whether this X^+ includes the attributes in Y. If this is the case for every FD in E, then F covers E.

Page No.
Date: / /

Nonredundant Covers :-

Given two sets of functional dependencies E and E, if E covers F and if no proper subset E' (E'⊂E) covers F, E is called a **non-redundant cover** of F.

OR

Given a set of FDs F, we say that it is non-redundant if no proper subset F' of F is equivalent to F, i.e. no F' exists such that

$$F'^{+} = F^{+}$$

If a proper subset F' of F covers F, then F is **redundant** and we can remove some FD, say $X \to Y$, from F to find a non-redundant cover of F.

If F is a set of FDs and if E is a nonredundant cover of F, then it is not true that E has the minimum no. of FDs.

Page No.
Date: / /

Algorithm to find a Non-Redundant Cover :-

(1) Initially E = F
(2) for each FD, $X \to Y$ in E do
(3) if $X \to Y \in \{F - (X \to Y)\}^{+}$
(4) then $F = \{F - (X \to Y)\}$
(5) E = F

Canonical Cover or Minimal Cover :-

A set of FDs F is called a canonical cover or minimal cover, if it satisfies the following conditions :-

(1) Every dependency in F has a single attribute for its right hand side
(2) We can not replace any dependency $X \to A$ in F with a dependency $Y \to A$, where $Y \subset X$ and still have a set of dependencies that is equivalent to F.
(3) We can not remove any dependency from F and still have a set of dependencies that is equivalent to F.

Page No.
Date: / /

Algorithm to find a minimal cover F for a set of FDs E:-

(1) Set F = E

(2) Replace each functional dependency $X \to \{A_1, A_2, \ldots A_n\}$ in F by the n functional dependencies $X \to A_1$, $X \to A_2$, ... $X \to A_n$.

(3) For each functional dependency $X \to A$ in F
for each attribute B that is an element of X
if $\{\{F - \{X \to A\}\} \cup \{\{(X - \{B\}) \to A\}\}$ is equivalent to F, then:-
Replace $X \to A$ with $(X - \{B\}) \to A$ in F.

(4) For each remaining FDs $X \to A$ in F
if $\{F - \{X \to A\}\}$ is equivalent to F, then remove $X \to A$ from F.

Full Functional Dependency:-

Given a relational scheme R and an FD $X \to Y$, Y is fully functionally dependent on X if there is no Z, where $Z \subset X$ such that $Z \to Y$.

If there exist some Z, where $Z \subset X$, such that $Z \to Y$, then $X \to Y$ is a partial dependency.

Page No.
Date: / /

Normalization of Relations:-

Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of:-
(1) minimizing redundancy
(2) minimizing the insertion, deletion and update anomalies.

The normal form of a relation refers to the highest normal form condition that it meets.

The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:-
(1) The lossless join or nonadditive join property, which guarantees that the spurious tuple generation problem does not occur.
(2) The Dependency Preservation Property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

* The non additive join property is extremely critical and must be achieved at any cost.

First Normal Form :-

A relation schema is said to be in First Normal Form (1NF), if the values in the domain of each attribute of the relation are atomic.

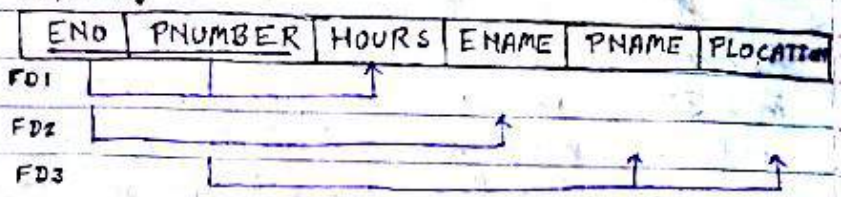
In other words only one value is associated with each attribute and the value is not a set of values or a list of values.

Second Normal Form :- (Considers only primary key)

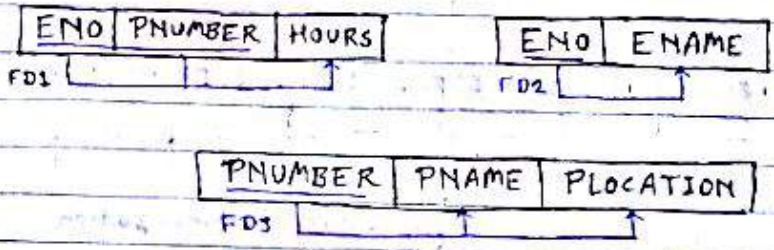
→ Second Normal Form (2NF) is based on the concept of full functional dependency.

A relation schema R is in 2NF, if every non-prime attribute A in R is fully functionally dependent on the primary key of R.

Emp_Proj



↓ 2NF Normalization



Third Normal Form :- (Considers only primary key)

3NF is based on the concept of transitive dependency.

A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency, if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

The general definition of 2NF & 3NF considers pastial, full and transitive FDs, with respect to all candidate keys of a relation.

Page No.
Date: / /

According to Codd's original definition, a relation schema R is in 3NF, if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

Emp_Dept

ENAME	ENO	DOB	ADDRESS	DNUMBER	DNAME	DMGRENO

↓ 3NF Normalization

ENAME	ENO	DOB	ADDRESS	DNUMBER

DNUMBER	DNAME	DMGRENO

General Definition of 3NF:- (Considers all candidate keys)

A relation schema R is in third normal form (3NF) if, whenever a non-trivial functional dependency $X \rightarrow A$ holds in R, either

(a) X is a superkey of R.

or

(b) A is a prime attribute of R.

Page No.
Date: / /

Boyce-Codd Normal Form:-

→ Every relation in BCNF is also in 3NF, however a relation in 3NF is not necessarily in BCNF.

A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.

Imp:- Trick to check the NF of a Relation:-

(1) First of all calculate all candidate keys of the relation using given FDs.

(2) Use candidate keys and the given FDs to check for NF.

2-attribute Relations:-

Any 2-attribute relation of the form R(A, B) is always in BCNF.

Proof:- Consider the following cases:-

(1) There are no FDs b/w A & B, in which case only trivial FDs exist and R is in BCNF.

(2) $A \rightarrow B$, but there is no FD of the form $B \rightarrow A$. In this case, A is a key and R is in BCNF.

(3) $B \rightarrow A$, but there is no FDs of the form $A \rightarrow B$. In this case, B is the key and R is in BCNF.

(4) $A \rightarrow B$ and $B \rightarrow A$. Both A and B are keys, this does not violate the BCNF condition.

In the definition of normal forms, an attribute that is part of any candidate key will be considered as prime attribute.

General Definition of 2NF :- (Considers all candidate keys)

A relation schema R is in second normal form (2NF), if every nonprime attribute A in R is not partially dependent on any key of R.

OR

A relation schema R is in 2NF, if every non-prime attribute A in R is fully functional dependent on every key of R.

Note :-

In general definitions of 2NF and 3NF, partial and full functional dependencies and transitive dependencies are considered with respect to all candidate keys of a relation.

Properties of Relational Decompositions :-

Attribute Preservation Property of Decomposition :-

Let R be a relation schema. and let $D = \{R_1, R_2, \ldots, R_m\}$ be the decomposition of R into $R_1, R_2, \ldots, R_m$, then :

We must make sure that each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are lost.

Formally, we have :-

$$\bigcup_{i=1}^{m} R_i = R$$

This is called the attribute preservation property of decomposition.

Page No.
Date: / /

Dependency Preservation Property of a Decomposition :-

If each functional dependency $X \rightarrow Y$ specified in F either appears directly in one of the relation schemas R_i in the decomposition D or could be inferred from the dependencies that appear in some R_i. Informally, this is the dependency preservation condition.

→ Given a set of dependencies F on R, the projection of F on R_i, denoted by $\pi_{R_i}(F)$, where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \rightarrow Y$ are all contained in R_i.

A decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is dependency preserving with respect to F, if the union of the projections of F on each R_i in D is equivalent to F; that is :-

$$((\pi_{R_1}(F)) \cup (\pi_{R_2}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$$

Page No.
Date: / /

It is always possible to find a dependency-preserving decomposition D with respect to F such that relation R_i in D is in 3NF.

Lossless (Nonadditive) Join Property of a Decomposition :-

A decomposition $D = \{R_1, R_2, \ldots R_m\}$ of R has the lossless (non additive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the Natural Join of all the relations in D :-

$$*(\pi_{R_1}(r), \ldots, \pi_{R_m}(r)) = r$$

The word loss refers to loss of information not to loss of tuples

Algorithm for testing for Lossless (nonadditive) Join Property :-

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

(1) Create an initial matrix S with one row i for each relation R_i in D and one column j for each attribute A_j in R.

(2) Set $S(i,j) = b_{ij}$ for all matrix entries.

(3) For each row i representing relation schema R_i

{ for each column j representing attribute A_j

{

If (relation R_i includes attribute A_j) then set $S(i,j) = a_j$

}

}

(4) Repeat the following loop until a complete loop execution results in no change to S

{ for each FD X → Y in F

{ for all rows in S that have the same symbols in the columns corresponding the attributes in X

{ make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows :-

If any of the rows has an "a" symbol for the column, set the other rows to that same "a" symbol in the column.

If no "a" symbols exists for the attribute in any of rows, choose one of the "b" symbols that appears in one of the rows for the attribute and set the other rows to that same "b" symbol in the column.

} } }

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

Example :-

R = {ENO, ENAME, PNUMBER, PNAME, PLOCATION, HOURS}

R_1 = {ENO, ENAME}

R_2 = {PNUMBER, PNAME, PLOCATION}

R_3 = {ENO, PNUMBER, HOURS}

F = {ENO → ENAME, PNUMBER → {PNAME, PLOCATION}, {ENO, PNUMBER} → HOURS}

Page No.
Date: / /

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

(Original matrix S at start of algorithm)

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

Since here last row is all 'a' symbols, so it is a lossless join decomposition.

Testing binary decomposition for the Nonadditive Join Property :-

This method is applicable to binary decompositions only.

A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property w.r.t. a set of FDs F on R if and only if either

The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+

OR

The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+

Page No.
Date: / /

Successive Lossless (Non additive) Join Decompositions :-

If a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R has the non additive join property with respect to a set of f.d.s F on R and if a decomposition $D_j = \{Q_1, Q_2, \ldots Q_k\}$ of R_i has the non additive join property w.r.t. the projection of F on R_i, then the decomposition $D_2 = \{R_1, R_2, \ldots R_{i-1}, Q_1, Q_2 \ldots Q_k, R_{i+1} \ldots R_m\}$ of R has the non additive join property w.r.t. F.

Page No
Date: / /

Algorithms for creating a relational decomposition :-

Dependency-Preserving Decomposition into 3NF Schemas :-

Input :- A universal relation R and a set of FDs F on the attributes of R.

(1) Find a minimal cover G for F.

(2) For each L.H.S. X of an FD that appears in G, create a relation schema in D with attributes $[X \cup \{A\} \cup \{A\} \cup \dots \cup \{A_k\}]$ where $X \to A_1, X \to A_2, \dots, X \to A_k$ are the only dependencies in G with X as the L.H.S.

(3) Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

Page No.
Date: / /

Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas :-

Input :- A universal relation R and a set of FDs F.

(1) Find a minimal cover G of F.

(2) For each L.H.S. X of an F.D. that appears in G, create a relation schema in D with attributes $[X \cup \{A_1\} \cup \{A_2\} \cup \dots \cup \{A_k\}]$, where $X \to A_1, X \to A_2, \dots, X \to A_k$ are the only dependencies in G with X as L.H.S.

(3) If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R.

Algorithm to find a key K of R :-

Input :- A relation R and a set of FDs F.

(1) Set $K = R$

(2) For each attribute A in K

{

Compute $(K-A)^+$ w.r.t. F.

If $(K-A)^+$ contains all attributes in R, then set $K = K-\{A\}$

}

Page No.
Date: / /

Lossless (Nonadditive) Join Decomposition into BCNF Schemas :-

Input :- A universal relation R and a set of FDs F.

(1) Set D = {R}

(2) While there is a relation schema Q in D that is not in BCNF, do :-

{

choose a relation schema Q in D that is not in BCNF.

find an FD X → Y in Q that violates BCNF.

Replace Q in D by two relation schemas (Q − Y) and (X ∪ Y).

}

The theory of lossless join decomposition is based on the assumption that no null values are allowed for the join attributes.

The algorithms for normalization are not deterministic.

Page No.
Date: / /

We use 3NF, if we require efficiency in :-

INSERT, UPDATE and DELETE operations

We use 2NF, if we have more data retrieval operations (SELECT), than insert update or delete operation.

Normal forms are insufficient on their own as criteria for a good relational database schema design. The relations must collectively satisfy two additional properties :-

(i) Dependency Preservation Property.

(ii) Lossless or non-additive join property.

Page No.
Date: / /

www.ankurgupta.net

Questions for FDs and Normal Forms

Page No.
Date: / /

(1) $A \rightarrow C$, $AC \rightarrow D$, $E \rightarrow AD$, $E \rightarrow H$

(2) $A \rightarrow CD$, $E \rightarrow AH$

(3) $A \rightarrow BCDEF$, $BC \rightarrow ADEF$, $B \rightarrow F$, $D \rightarrow E$

(4) $AB \rightarrow C$, $A \rightarrow DE$, $B \rightarrow F$, $F \rightarrow GH$, $D \rightarrow IJ$

(5) $AB \rightarrow C$, $BD \rightarrow EF$, $AD \rightarrow GH$, $A \rightarrow I$, $H \rightarrow J$

(6) $AB \rightarrow C$, $CD \rightarrow E$, $DE \rightarrow B$

(7) $A \rightarrow CDE$, $ABGH \rightarrow FIJK$, $GHIJ \rightarrow ABF$

(8) $AC \rightarrow BDE$, $B \rightarrow E$, $C \rightarrow D$

(9) $A \rightarrow CF$, $C \rightarrow D$, $B \rightarrow E$

(10) $A \rightarrow D$, $AB \rightarrow C$, $D \rightarrow E$

(11) $AB \rightarrow C$, $A \rightarrow D$, $B \rightarrow EF$

PageNo
Date: / /

www.ankurgupta.net

Nested Loop Join and Block Nested Join

PageNo
Date: / /

Nested Loop Join:-

For each tuple r in R do
  For each tuple s in S do
    if r and s satisfy the join condition
      then output the tuple $\langle r, s \rangle$

This algorithm involves $n_r \times b_s + b_r$ block transfers, where:-

n_r = No. of records in R.
b_s = No. of blocks used by S.

Block Nested Loop Join:-

For each block B_r in R do
  For each tuple s in S do
    For each tuple r in block B_r do
      if r and s satisfy the join condition
        then output the tuple $\langle r, s \rangle$

This algorithm involves $b_r \times b_s + b_r$ block transfers.

Page No.
Date: / /

www.ankurgupta.net

Transaction Processing and Concurrency Control

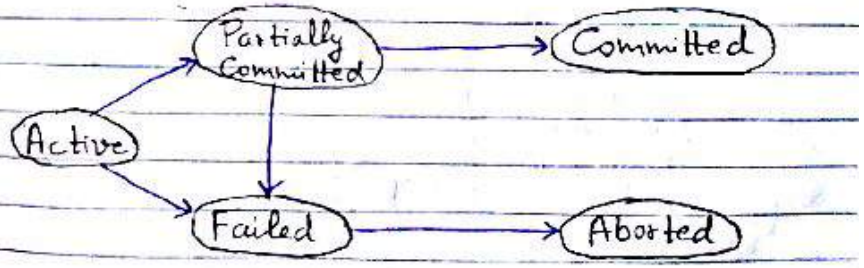
Page No.
Date: / /

Properties of Transaction :- (ACID)

(1) Atomicity → Transaction Management Component
(2) Consistency
(3) Isolation → Concurrency Control Component
(4) Durability → Recovery Management Component

Transaction State :-

(1) Active :- While executing.
(2) Partially Committed :- After the final statement has been executed.
(3) Failed :- When normal operation can no longer proceed.
(4) Aborted :- After the transaction has been rolled back and database has been restored to its state prior to start of the transaction.
(5) Committed :- After succesful completion.



Terminated :- If the transaction has either committed or aborted.

Page No.
Date: / /

Concurrent execution of transactions allows :-

(i) Improved throughput and Resource Utilization.

(ii) Reduced waiting time.

Schedules :-

Schedules represent the order in which instructions are executed in the system.

Serial Schedule :-

It consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.

Example :-

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).

Page No.
Date: / /

Conflict Serializability :-

→ Two instructions I_i and I_j conflict, if they are operations by different transactions on the same data item and at least one of these instructions is a write operation.

→ 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.

→ A schedule S is conflict serializable, if it is conflict equivalent to a serial schedule.

Example :-

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
	Write (A)
Write (B)	
	Read (B)
	Write (B)

Page No.
Date: / /

It is possible to have two schedules that produce the same outcome, but that are not conflict equivalent.

View Serializability :-

The schedules S and S' are said to be view equivalent if 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 if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction T_j.

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

→ A schedule S is view serializable if it is view equivalent to a serial schedule.

Example :-

T_3	T_4	T_6
Read(Q)		
	Write(Q)	
Write(Q)		
		Write(Q)

→ Every conflict-serializable schedule is also view-serializable, but there are view-serializable schedules that are not conflict serializable.

→ The above schedule is view equivalent to the serial schedule $\langle T_3, T_4, T_6 \rangle$

→ In above schedule, transaction T_4, T_6 perform write(Q) operations without having performed a read(Q) operation. Writes of this sort are called blind writes.

→ Blind Writes appear in any view-serializable schedule that is not conflict serializable.

Page No.
Date: / /

Recoverable Schedules :-

A recoverable schedule is one where, 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 commit operation of T_j.

Cascadeless Schedules :-

The phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback.

A cascadeless schedule is one where 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.

Page No.
Date: / /

Testing for Conflict Serializability :-

Construct a directed graph, called a precedence graph, from schedule S. This graph consists of a pair $G = (V, E)$, where V is a set of vertices and E is a set of edges.

The set of vertices consists of all the transactions participating in the schedule.

The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions hold:-

(1) T_i executes write(Q) before T_j executes read(Q).

(2) T_i executes read(Q) before T_j executes write(Q).

(3) T_i executes write(Q) before T_j executes write(Q).

If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then, in any serial schedule S' equivalent to S, T_i must appear before T_j.

If the precedence graph for S has a cycle, then schedule S is not conflict serializable.

Testing for View Serializability :-

Testing for view serializability is rather complicated, and has been shown that this problem is NP-Complete.

Lock-based Protocols :-

→ If we do not use locking, or if we unlock data items as soon as possible after reading or writing them, we may get inconsistent states.

→ On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur.

→ Deadlocks are definitely preferable to inconsistent states.

The Two-Phase Locking Protocol :-

Each transaction issues lock and unlock requests in two phases :-

(1) Growing Phase :-
A transaction may obtain locks, but may not release any lock.

(2) Shrinking Phase :-
A transaction may release locks, but may not obtain any new locks.

→ Two phase locking protocol ensures conflict serializability.

→ It does not ensure freedom from deadlock.

→ Cascading rollback may occur under two phase locking.

→ Cascading rollbacks can be avoided by a modification of two phase locking called the strict two-phase locking protocol. Here all exclusive-mode locks taken by a transaction are held until that transaction commits.

Page No.
Date: / /

Rigorous Two-Phase Locking :-

It requires that all locks be held until the transaction commits.

Lock Conversion :-

We denote conversion from shared to exclusive modes by upgrade, and from exclusive to shared by downgrade.

→ Upgrading can take place in only the growing phase, whereas downgrading can take place in only the shrinking phase.

→ Two-phase locking with lock conversion generates only conflict-serializable schedules, and if exclusive locks are held until the end of the transaction, the schedules are cascadeless.

Page No.
Date: / /

Graph-Based Protocols :-

It depends upon the prior knowledge about the order in which the database items will be accessed.

In the tree protocol, the only lock instruction allowed is lock-X. Each transaction T_i can lock a data item at most once, and must observe the following rules:-

(1) The first lock by T_i may be on any data item.

(2) Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.

(3) Data items may be unlocked at any time.

(4) A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.

→ All schedules that are legal under the tree protocol are conflict serializable and are free from deadlock.

→ To ensure recoverability and cascadelessness, the protocol can be modified to not permit release of exclusive locks until the end of transaction.

→ There are schedules possible under the two-phase locking protocol that are not possible under the tree protocol, and vice versa.

Page No.
Date: / /

Timestamp-Based Protocols :-

The timestamp-ordering protocol operates as follows :-

(1) Suppose that transaction T_i issues read(Q).

(a) If $TS(T_i) < W\text{-timestamp}(Q)$, 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.

(b) If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and $TS(T_i)$.

(2) Suppose that transaction T_i issues write(Q).

(a) If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that the value would never be produced. Hence, the system rejects the write operation and rolls T_i back.

(b) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q. Hence the system rejects this write operation and rolls back T_i.

(c) Otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$.

→ If a transaction T_i is rolled back, by the system assigns it a new timestamp and restarts it.

→ There are schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and vice-versa.

→ It ensures conflict serializability.

→ It ensures freedom from deadlock, since no transaction ever waits.

→ Recoverability and cascadelessness can be ensured by performing all writes together at the end of the transaction.

→ Recoverability and cascadelessness can also be guaranted by using a limited form of locking, whereby reads of uncommitted items are postponed until the transaction that updated the item commits.

The number of concurrent schedules possible from two transaction containing n_1 and n_2 instructions is :-

$$\frac{(n_1 + n_2)!}{n_1!\, n_2!}$$

Conservative two-phase locking protocol :-

In C2PL protocol, transactions obtain all the locks they need before they begin.

Deadlock Prevention :-

(1) Wait-die Scheme :-

It is a nonpreemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j. Otherwise, T_i is rolled back.

(2) Wound-wait scheme :-

It is a preemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j. Otherwise T_j is rolled back.

Page No.
Date: / /

www.ankurgupta.net

Transaction Log

Page No.
Date: / /

Techniques to maintain Log Files :-

(1) Deferred Update
(2) Immediate Update

Occurence of failure in Deffered Update Scheme :-

$<T_0$ Starts$>$
$<T_0, A, 950>$
$<T_0, B, 2050>$
CRASH

Recovery :-
No Action

$<T_0$ Starts$>$
$<T_0, A, 950>$
$<T_0, B, 2050>$
$<T_0$ Commits$>$
$<T_1$ Starts$>$
$<T_1, C, 600>$
CRASH

Recovery :-
Redo (T_0)

$<T_0$ Starts$>$
$<T_0, A, 950>$
$<T_0, B, 2050>$
$<T_0$ Commits$>$
$<T_1$ Starts$>$
$<T_1, C, 600>$
$<T_1$ Commits$>$
CRASH

Recovery :-
Redo (T_0)
Redo (T_1)

Page No.
Date: / /

Occurence of failure in Immediate Update Scheme :-

<T0 Starts>
<T0, A, 950>
<T0, B, 2050>
CRASH

Recovery :-
~~No Redo~~
Undo (T_0)

<T0 Starts>
<T0, A, 950>
<T0, B, 2050>
<T0 Commits>
<T1 Starts>
<T1, C, 600>
CRASH

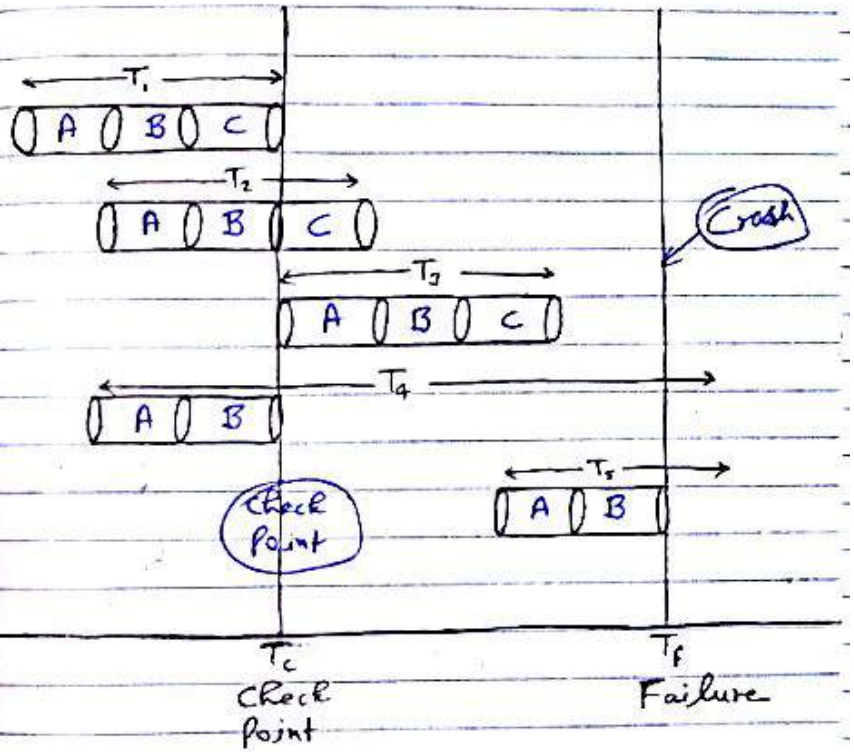
Recovery :-
Undo (T_1)
Redo (T_0)

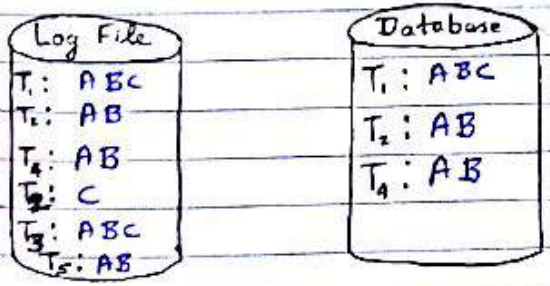
<T0 Starts>
<T0, A, 950>
<T0, B, 2050>
<T0 Commits>
<T1 Starts>
<T1, C, 600>
<T1 Commits>
CRASH

Recovery :-
Redo (T_0)
Redo (T_1)

Page No.
Date: / /

Checkpoints :-





Recovery :-
T_1 : No changes T_2 : Redo C, T_3 : Redo ABC
T_4 : Undo AB T_5 : Discard AB.