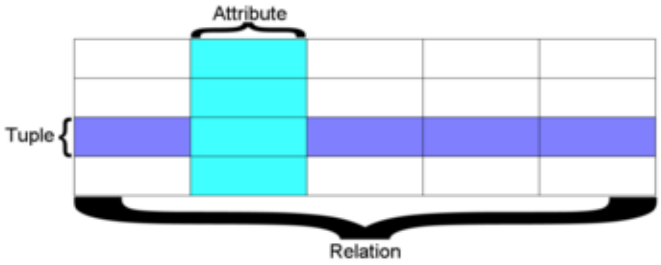
DBMS_Unit 2
Relational model and Relational Algebra

- Relational Model Concepts
- Relational Constraints and Relational database Schemas
- Update Operations
- Transactions and Dealing with Constraint Violations
- Unary Relational Operations: Select and Project
- Relational Algebra Operations from Set Theory
- Binary Relational Operations: JOIN and DIVISION
- Aggregate Functions, Generalized Projections
- Examples of Queries in Relational Algebra
- Relational database Design Using ER-to-Relational Mapping

Relational Model Concepts

- The relational model is based on a collection of mathematical principles drawn primarily from set theory and predicate logic.
- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper:
  "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.
- A relation is a table consisting of rows (tuples) and columns (attributes).
- Each row represents a fact that corresponds to a real-world entity or relationship.
  Each row has a value of an item or set of items that uniquely identifies that row in the table.
- Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
- Name of table and columns are necessary to interpret the meaning of values.
- Each column typically is called by its column name or column header or attribute name.



- **Domain:**
  - A set of atomic (indivisible) values for each attribute.
  - A data type, format is specified for each column
  - Thus a domain has a name, data type & format.
  - Additional information for interpreting the values of a domain can also be given e.g. dollar, kg etc.

  e.g.

  Phone_number: Set of all Phone Numbers in India having 10 digits

  Employee age: All possible ages (20 – 70)

Relational Schema: relation name and a list of attributes

e.g.

- R(A1, A2, A3, ……..An)
  - R → Relation
  - A1, A2, A3, …An → set of attributes in relation R
- dom(Ai): domain of attribute of Ai
- dom (A1): domain of attribute A1
- Degree (arity) of relation is number of attributes.

STUDENT	USN	Name	Address	DOB	GPA

- Relation name: STUDENT
- List of attributes: USN, Name, Address, DOB, GPA
- Dom (name): All possible Names
- Degree of relation: 5

Using data type of each attribute

STUDENT (USN: string, Name: string, Address: string, DOB: Date, GPA: real)

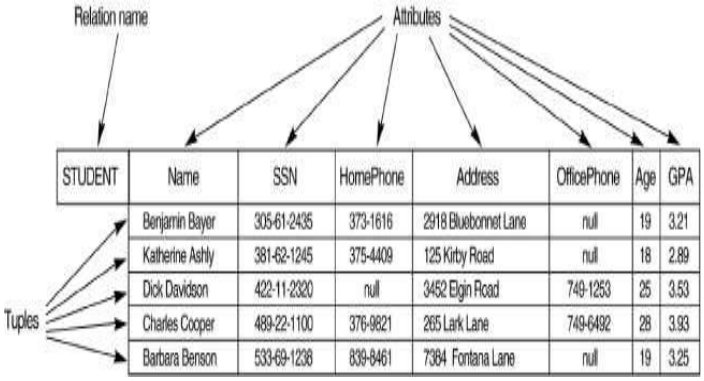
- *r* (*R*) denotes a *relation r* on the *relation schema*

R (A_1, A_2, A_3…..A_n)

- and is a set of n-tuples

  r= {t_1, t_2, t_3, ….., t_m}

  where t = {v_1, v_2, v_3,….v_n} $v_i = t[A_i]$
- R: schema of the relation
- r of R: a specific "value" or population of R.
- R is also called the intension of a relation
- r is also called the extension of a relation



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

- A relation r(R) is a mathematical relation of degree n on the domains dom(A_1), dom(A_2),….,dom(A_n).
- It is a subset of the Cartesian product of the domains that define R
- r(R) (dom *(A1)* * dom (A2) * .... * dom(An))
- The Cartesian product specifies all possible combinations of the values from the underlying domains.
- Total no of tuples in the Cartesian product is

$|dom(A_1)| * |dom(A_2)| * \ldots\ldots * |dom(A_n)|$

where cardinality of the domain (finite) is |D|.

- r(R) is the current relation state i.e. valid tuples that represent a particular state of the real world (or few tuples out of the combinations resulted from Cartesian product) .
- Relation schema is relatively much more stable.
- Relation state changes as the state of the real world changes.

Characteristics of Relations

- Each value is derived from an appropriate domain.
- A relation may be regarded as a *set of tuples* (rows).
- Columns in a table are also called attributes of the relation.
- **Ordering of tuples in a relation**
  - The tuples are *not* considered to be ordered physically, even though they appear to be in the tabular form.
  - Many logically orders can be specified in a relation .
- **Ordering of attributes in a relation schema R (and of values within each tuple):**
  - A tuple is an ordered list of n values

    e.g. the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> are *ordered*.

    At the logical level order of attributes and their values is not important as long as correspondence between attributes and values is maintained.
- **Values and NULLs in the Tuples**
  - Each value is an atomic value, composite and multivalued attributes are not allowed (due to first normal form).
  - Multivalued attributes must be represented by separate relations.
  - Composite attributes are represented by their simple component attributes in the basic relational model.
- **NULL represents the values of attributes because**
  - Value unknown
  - Value exists but not available
  - Attribute not applicable

Relational Model Notation

- A relation schema R of degree n: R(A1,A2,…..An)
- Tuple t in a relation: t = <V1, V2,…. Vn>
  - V_i belongs to A_i
- Q, R, S → Relation names
- q, r, s → Relation states
- t, u, v → tuples
- E.g. Student (USN, Name, Address) → Relation schema
- Student (111, Meena, #508 RRNagar) → one tuple from the current Relation state
- R.A → Student.Name Student.Address

Example

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  is a tuple belonging to the CUSTOMER relation.

Relational Model constraints & Relational Database Schema

Relational Model constraints

Constraints are *restrictions* specified on a relational database and must hold on *all* valid relation instances.

Constraints can be divided into three main categories:

1. **Inherent model-based or implicit constraints**
   Constraints that are inherent in the data model
2. **Schema based or explicit constraints**
   Constraints that are directly expressed in schema of the data model specifying in DDL
3. **Application-based or semantic constraints or business rules**
   Constraints that can not be directly expressed in schema of the data model and hence must be expressed and enforced by the application programs

- **There are following main types of constraints which can be specified in DDL :**
  - Domain Constraint
  - Key Constraint
  - Entity Integrity
  - Referential Integrity

Domain Constraint

- Within each tuple, the value of each attribute must be an atomic value from that domain.
- Data type
  - Integers, real
  - Characters, fixed & variable length strings
  - Date, time etc.
- Range of values can be specified for each attribute

Key Constraint

- A relation is a set of tuples and all elements of a set are distinct. Thus, all tuples in a relation state must be distinct.
- Superkey is set of attributes that identifies the tuples uniquely in a valid relation state.
- Any such set of attributes is called a Superkey
- Default superkey is a set of all attributes
  t1[SK] # t2[SK]

- Minimal superkey is a superkey from which we can not remove any attributes and still have uniqueness constraint.

or

- A "minimal" superkey is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.
- A minimal superkey is called a key attribute of the relation.

Key Constraint (Example)

Relation schema

- Student {USN, Name, Address}
- Default Superkey →
  - {USN, Name, Address}

  Removing redundant attributes
  - {USN, Name} → Superkey
  - {USN} → Minimal Superkey (or key)

Relation schema

- Order_details {Order_no, item_no, Quantity }
- Default Superkey →
  - {Order_no, item_no, Quantity }

  Removing redundant attributes
  - {Order_no, item_no} → Minimal Superkey (or key)
- **Candidate keys** are those key attributes which can uniquely identifies tuples separately.
- Candidate key must satisfy the following properties:
  - **Uniqueness property:**

    No two distinct tuples have the same value for the key.
  - **Minimality property:**

    None of the attributes of the key (in case of composite key) can be discarded from the key without destroying the uniqueness property.
- **Primary Key** is the candidate key that is chosen to identify the tuples uniquely.
  - Only one candidate key can be chosen as Primary key.
  - It should be underlined.
  - Value of primary key can not be null.
  - It may be single attribute or a small set of attributes. **(**if more than one attribute can identify the tuples uniquely together, those attributes together form composite primary key**)**

The car relation with two candidate keys: License number and engine serial number

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

NULL Constraint

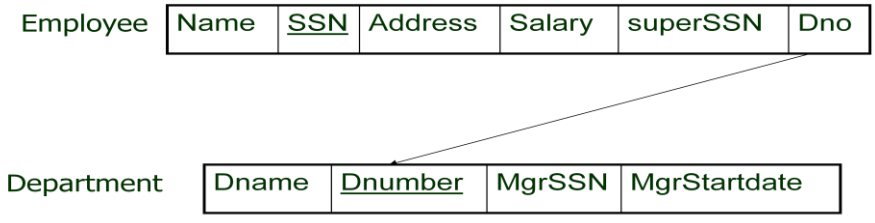
- Some of the Attributes can be restricted not to allow NULL values.
- NOT NULL constraint can be declared.

Entity Integrity Constraints

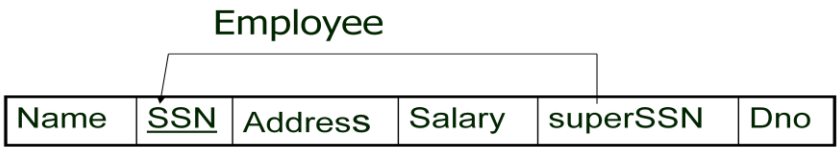
- **Entity Integrity:**
  - The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).
  - This is because primary key values are used to *identify* the individual tuples.
- Key Constraints & Entity Integrity Constraints are specified on individual relations
- Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.
- Referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations.

Foreign Key

- The foreign key is a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.
  - Dno of Employee refers to Dnumber of Department
- An attribute in a relation R1 is a foreign key [FK] of R1 that references PK of relation R2
- The attribute in FK have the same domains as the primary key [PK] attribute of R2



- Foreign Key can refer to its own relation
  - SuperSSN in Employee refers to SSN of Employee



- R1 → referencing relation
- R2 → referenced relation

- t1[FK]=t2[PK]
  - If t1 is from r1(R1)
  - If t2 is from r2(R2)
  - t1 of R1 refers to t2 of R2
- A tuple *t1* in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].
- However null is possible in FK.

Semantic Integrity Constraints

- Semantic Integrity Constraints can be specified and enforced on a relation database.
  - Salary of an employee should not exceed than 10,00,000
  - The max. no. of hours per employee
    for all projects he or she works on is 56 hrs
    per week
- A general purpose constraint specification language (triggers and assertions) can be used.

Relational Algebra Operations

- Operations can be categorized into retrievals and updates. Further, it can be classified as unary or binary operations. If operation is performed on one table it is unary and if operation is performed on two tables it is binary.
- Update operations are:
  - INSERT a tuple.
  - DELETE a tuple.
  - MODIFY a tuple.
- Retrieval operations are:
  - Select .
  - Project.

Update Operations on Relations

- Integrity constraints should not be violated during the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically. Thus, it may be necessary to maintain integrity constraints.

Insert operation

- Insert can violate any of the four types of constraints:
  - Domain constraints
    - If an attribute value does not belong to appropriate domain
  - Key constraints
    - If value of primary key of new tuple already exists in another tuple of the relation
  - Entity constraints
    - If value of primary key of new tuple is null
  - Referential integrity constraints
    - If value of foreign key refers to a tuple that does not exist.

DEPT

DNUM	**DNAME**
1	MCA
2	MBA

EMPLOYEE

ENUM	**ENAME**	**Salary**	**DNO**
E001	**Seema**	**30000**	**1**
E002	**Reena**	**32000**	**2**

- **Domain constraints**
  - Insert into EMPLOYEE values(‘e003’, ‘Teena’, ‘asfdf’, ‘1’);
  - → Not accepted
- **Key constraints**
  - Insert into EMPLOYEE values(‘e001’, ‘Teena’, 31000, ‘1’);
  - → Not accepted
- **Entity constraints**
  - Insert into EMPLOYEE values(‘’, ‘Teena’, 31000, ‘1’);
  - → Not accepted
- **Referential integrity constraints**
  - Insert into EMPLOYEE values(‘’, ‘Teena’, 31000, ‘3’);
  - → Not accepted
- Insert into EMPLOYEE values(‘e003’, ‘Teena’, 31000, ‘1’);
  - → Accepted

Delete operation

- Delete can violate only Referential integrity constraints
  - If the tuple being deleted is referenced by the foreign key from other relation’s tuples
  - delete from DEPT where dname = ‘MBA’;
  - → Not accepted
- Due to delete operation violation, three options are possible:

1. Reject the deletion
2. Cascade deletion – delete tuples from different relations that refer the tuple that being deleted
3. Modify the referencing attribute values

   . Null

   . Some valid value

- If FK is part of PK, it can not be NULL

Cascade deletion

- delete from DEPT where dname = ‘MBA’;
- Delete the tuples from DEPT and EMPLOYEE also wherever dname = ‘MBA’ is satisfied, (but not feasible)

or

- Delete the tuples from DEPT where dname = 'MBA'
  and tuples from EMPLOYEE will be updated to NULL for DNO where dname = 'MBA' (feasible)

Update (modify) operation

- Generally, it is necessary to specify a condition on the attributes of the relation to select the tuple to be modified
  - Updating an attribute other than PK, FK –

  ----only domain constraint will be violated

  ----i.e. check to confirm that the new value is of the correct data type and domain
- Modifying a PK → deleting one tuple + inserting a new tuple
  - Constraint violations due to Delete
  - Constraint violations due to Insert
  - Modifying a FK → new value should refer to an existing tuple in the referenced relation
  - Referential Integrity Violation

Select operation

- Select operation is used to select a subset of the tuples from a relation that satisfy a selection condition.
- It is a horizontal partitioning of the relation:
  - One partition satisfying the condition and will be shown as result.
  - Other partition not satisfying the condition will be discarded.

  SELECT operation is unary
- σ <selection condition> (R)
  - Where σ (sigma) is for select operation
  - selection condition is a boolean expression specified on the attributes of Relation
- Clauses of selection condition
  - <Attribute name> <comparison operator> <constant value>
  - <Attribute name> <comparison operator> <Attribute name>

Employee

σ salary>20,000 (Employee)

Empid	**Name**	**Salary**
101	**Ram**	**20,000**
102	**Shyam**	**22,000**
103	**Geeta**	**24,000**
104	**Geeta**	**25,000**
105	**Pam**	**18,000**

Resultant table:

Employee

Empid	Name	Salary
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

- **Comparison operator**
  - =, <, >, ≠, ≥,≤ for numeric value or date
  - =, ≠ for strings of characters
  - Constant value
  - Value from attribute domain
- Degree of resulting relation is same as that of R
- No. of tuples in resulting relation <= no. of tuples in R
  - i.e. |σc (R)| <= |R|
- Selection condition may be more than one
  - i.e. (cond1 and cond2) → true if both are true
  - i.e. (cond1 or cond2) → true if any one or both are true
  - NOT cond →true if cond is false
- Select operation is Commutative:
  - σ<cond1> (σ<cond2> (R)) = σ<cond2> (σ<cond1> (R))
  - A sequence of Selects can be applied in any order.

- Cascade of Selects can be used through one select using Conjunctive (AND)
  - σ<cond1> (σ<cond2> (….(σ<condn>(R))……))
    = σ<cond1> AND<cond2>AND…… AND<condn> (R)

Select operation: Commutative

Employee

σ salary>20,000

(σ Dno=2 (Employee))

Empid	Name	Dno	Salary
101	Ram	1	20,000
102	Shyam	2	22,000
103	Geeta	3	24,000
104	Geeta	2	25,000
105	Pam	2	18,000

Empid	**Name**	**Dno**	**Salary**

Resultant table:

102	**Shyam**	**2**	**22,000**
104	**Geeta**	**2**	**25,000**

Employee

Project Operation:

- Project operation selects certain column from relation and discards rest.
- It is a vertical partitioning of the relation:
  - One partition satisfying the condition and will be shown as result.
  - Other partition not satisfying the condition and will be discarded.
    PROJECT operation is unary.

Employee

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

Π empid, salary (Employee)

Resultant table:

Empid	Salary
101	20,000
102	22,000
103	24,000
104	25,000

Employee

Employee

Empid	**Name**	**Salary**
101	**Ram**	**20,000**
102	**Shyam**	**22,000**
103	**Geeta**	**25,000**
104	**Geeta**	**25,000**

Π name, salary (Employee)

Resultant table:

Employee

Name	Salary
Ram	20,000
Shyam	22,000
Geeta	25,000

- Π <attribute list> (R)
  - Where Π (pi) is for project operation
- Attribute list is the desired list of attributes from relation R
- Resulting relation has attribute only those specified in list and in same order
- Degree of resulting relation is same as no. of attributes in list
- No. of tuples in resulting relation?
  - If attribute list has Primary key, the no. of tuples will be same as in Relation R
  - If attribute list does not have Primary key, the duplicate tuples will be removed to give valid relation
  - Thus, no. of tuples in resulting relation <= no. of tuples in relation R
- Project operation is not Commutative:
  - Π<list1> (Π<list2> (R)) ≠ Π<list2> (Π<list1> (R))
  - Π<list1> (Π<list2> (R)) = Π<list1> (R)

Project Operation: not Commutative

Employee

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

Π empid (Π empid, salary (Employee))

Resultant table:

Empid
101
102
103
104

EMPLOYEE

Π salary (Π empid, salary(Employee))

Employee →

Salary
20,000
22,000
24,000
25,000

Project Operation: not Commutative

Employee

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

Π empid (Π empid, salary (Employee))

Resultant table:

Empid
101
102
103
104

EMPLOYEE

Π empid (Employee))

Employee→

Empid
101
102
103
104

Sequence of operations Operation

- Several algebraic expressions:
  - As a single relational expression by nesting operations
  - Π<list1> (σ<cond1> (R))

Or

  - Name the intermediate relation and get final result as a series of operations
  - temp ← σ<cond1> (R)
  - result ← Π<list1> (temp)

Employee

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

Π empid (σ salary>20,000 (Employee))

Empid

Resultant table:

Employee

102
103
104

Employee

Π empid (σ salary>20,000 (Employee))

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000
104	Geeta	25,000

temp

temp ← σ salary>20,000 (Employee)

Empid	**Name**	**Salary**
102	**Shyam**	**22,000**
103	**Geeta**	**24,000**
104	**Geeta**	**25,000**

empid (temp)

Employee

Empid
102
103
104

Rename

- Attributes can be renamed in resulting relation
  - temp ← σ<cond1> (Employee)
  - Emp(empid, empname, empsalary) ← Π<eid, ename, salary> (temp)
  - If no renaming is applied, resulting relation will have same names and order of attributes as parent relation has.

Sequence of operations Operation: Rename resulting relation

Empid	Name	Salary
101	Ram	20,000
102	Shyam	22,000
103	Geeta	24,000

Employee

104	Geeta	25,000

Employee

temp ← σ salary>20,000 (Employee)

Empid	**Name**	**Salary**
101	**Ram**	**20,000**
102	**Shyam**	**22,000**
103	**Geeta**	**24,000**
104	**Geeta**	**25,000**

EMP(EID) ← Π empid (temp)

EMP

EID
102
103
104

Set Theoretic Operation

- Binary operation
- Both relational should be union compatible
- Union compatible has following characteristics:
  - i.e. for R(A1, A2, ….An) & S(B1,B2,….Bn)
  - dom(Ai) = dom(Bi) for 1 <= i <=n
  - Degree of both relations should be same.
  - UNION (RUS)
- INTERSECTION (R ∩S)
- MINUS (R-S)
- UNION (RUS)
  - It includes all tuples that are either in R or S or in both
  - Duplicate tuples are eliminated.

UNION

Student U instructor

Student

Id	Name
101	**Ram**
102	**Shyam**
103	**Geeta**
104	**Rita**

Id	Name
101	Ram
102	Shyam
103	Geeta
104	Rita
107	Smith

Instructor

Id	Name
102	**Shyam**
103	**Geeta**
107	**Smith**

- INTERSECTION (R∩S)
  - It includes all tuples that are I n R & S both

Intersection

Student ∩instructor

Student

Id	Name
101	**Ram**
102	**Shyam**
103	**Geeta**
104	**Rita**

Id	Name
102	Shyam
103	Geeta

Instructor

Id	Name
102	**Shyam**
103	**Geeta**
107	**Smith**

Difference

- (R-S)
  - It includes all tuples that are in R but not in S
- (S-R)
  - It includes all tuples that are in S but not in R

(R-S) ≠ (S-R)

DIFFERENCE (S-R)

Student

Id	Name
101	**Ram**
102	**Shyam**
103	**Geeta**
104	**Rita**

Instructor

Id	Name
102	**Shyam**
103	**Geeta**
107	**Smith**

Student -instructor

Id	Name
101	Ram
104	Rita

DIFFERENCE: not commutative

Student – Instructor

Id	Name
101	**Ram**
104	**Rita**

Instructor - Student

Id	Name
107	**Smith**

- Union & Intersection are:

1. Commutative
   - (R U S) = (S U R)
   - (R ∩ S) = (S ∩ R)
2. Associative
   - (R U (S U T) = (R U S) U T)
   - (R ∩ (S ∩ T) = (R ∩ S) ∩ T)

Student UNION

Id	Name
101	Ram
102	Shyam
103	Geeta
104	Rita

Instructor

Id	Name
102	**Shyam**
103	**Geeta**
107	**Smith**

Student U Instructor

Id	Name
101	Ram
102	Shyam
103	Geeta
104	Rita
107	Smith

Instructor U Student

Id	Name
101	Ram
102	Shyam
103	Geeta
104	Rita
107	Smith

INTERSECTION

Student

Id	Name
101	Ram
102	Shyam
103	Geeta
104	Rita

Instructor

Id	Name
102	Shyam
103	Geeta
107	Smith

Student ∩ Instructor

Id	Name
102	Shyam
103	Geeta

Instructor ∩ Student

Id	Name
102	Shyam
103	Geeta

Student

Id	Name
101	Ram
102	Shyam
103	Geeta

Player

Id	Name
103	Geeta
102	Shyam

Instructor

Id	Name
102	Shyam
104	Smith
105	Nita

(student U instructor) U player

Id	Name
101	Ram
103	Geeta
104	Rita
105	John
102	Shyam

student U (instructor U player)

Id	Name
101	Ram
103	Geeta
104	Rita
105	John
102	Shyam

Cartesian Product

- Also known as cross product or cross join
- Binary Operation
- R x S
- no. of tuples in Resultant relation = no. of tuples in R x no. of tuples in S
- It includes one tuple from each combination
- No. of attributes in Resultant relation = n + m
- R(A1, A2,…,An) x S(B1, B2,...,Bn) = Q(A1, A2,…,An, B1, B2,….,Bn)
- Result will have some irrelevant tuple which can be further processed by select & project operation

EMPLOYEE

Emp_id	Name
101	Ram
102	Shyam
103	Geeta

DEPENDANT

Dep_name	Bdate
Meena	23-02-1988
Raju	23-02-1990

Employee x dependant

Emp_id	Name	Dep_name	Bdate
101	**Ram**	**Meena**	**23-02-1988**
101	**Ram**	**Raju**	**23-02-1990**
102	**Shyam**	**Meena**	**23-02-1990**
102	**Shyam**	**Raju**	**23-02-1990**
103	**Geeta**	**Meena**	**23-02-1990**
103	**Geeta**	**Raju**	**23-02-1990**

Binary relational Operations

- JOIN
- DIVISION

JOIN Operation

- R⋈ S It is used to combine related tuples from two relations into single tuples.
- It allows to process relationships among relations
- R⋈ <join condition> S
  - Attributes in Resultant relation = n + m
  - R(A1, A2,…..,An) S(B1, B2,….,Bn) = Q(A1, A2,…..,An, B1, B2,….,Bn)
  - No. of tuples wherever join condition is satisfied.

JOIN Operation and Cartesian product

- In Join, combinations of tuples satisfying joining condition appear in the result.
- Whereas in Cartesian product, all combinations of tuples are included in the result.
- Join operation can be stated as Cartesian product operation followed by Select operation
- A general JOIN condition is of the form
  - <condition> AND <condition> AND ….AND <condition>
  - Each condition is of the form Ai θ Bi
- Theta join:
  - Condition is of the form Ai θ Bi where θ is one of the {=,<,>,≥,≤,≠}
  - dom(Ai) = dom(Bi)
  - Tuples whose join attributes are null do not appear in the result
  - S1 $\bowtie_{S1.sid < R1.sid}$ R1

- EQUI JOIN
  - When comparison Operator used is =, is called an EQUI JOIN
- One or more pairs of attributes that have identical values

EQUI Join operation

EMPLOYEE

emp_id	Name
101	Ram
102	Shyam
103	Geeta

Employee ⋈ emp_id=eidDependant

emp_id	eid	Name	Dep_name
101	101	Ram	Meena
102	102	Shyam	Raju

DEPENDANT

Dep_name	eid
Meena	101
Raju	102

- In EQUI JOIN one pair of attribute have identical values in each tuple i.e. one attribute is superfluous, a new JOIN can be applied:
  - Natural JOIN i.e. R * S
- Both the tables should have one same attribute as joining attribute with same name
- If attribute is same but name is different (first perform rename than natural join)

Join operation: Natural JOIN

EMPLOYEE

emp_id	Name
101	Ram
102	Shyam
103	Geeta

DEPENDANT

Dep_name	eid _id
Meena	101
Raju	102

Employee * Dependant

emp_id	Name	Dep_name
101	Ram	Meena
102	Shyam	Raju

- If join condition is not satisfied → no tuple in resultant relation
- If no. of tuples in R = n_R
- If no. of tuples in S = n_S
- Tuples in resultant of EQUI JOIN are in between zero and $n_R * n_S$
- Join selectivity ratio = expected size of join result / maximum size

Outer Join Operations

- Outer joins are used to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.
- This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.
- The join operations where only matching tuples are kept in the result, are called inner joins. JOIN
- LEFT OUTER JOIN operation keeps every tuple present in the first (left) relation in the result of R ⟕ S.
- If no matching tuple is found in S, then the attributes of S in the join result are filled with null values.

Outer Join Operations

Relation *loan*

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *borrower*

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

Left Outer Join

loan ⟕ *loan.loan_number=borrower.loan_number* *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

- RIGHT OUTER JOIN keeps every tuple present in the second (right) relation S in the result of R ⟖ S.

Outer Join Operations

Relation *loan*

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *borrower*

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

Right Outer Join

loan ⟖ *loan.loan_number=borrower.loan_number* *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

141

- FULL OUTER JOIN keeps all tuples present in both the left and the right relations.

R ⟗ S

Outer Join Operations

Relation *loan*

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *borrower*

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

Full Outer Join

loan ⟗ *loan.loan_number=borrower.loan_number* *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

143

Division operation

- Division operation is suited to queries that include the phrase "for all"
- Denoted by ÷
- Operation is applied to two relations

  R(Z) ÷ S(X)

- Division operation is applied to two relations R(Z) ÷ S(X) where X is subset of Z and attribute Y of R that is not attribute of S.

PNO
1
2

Division

R

ESSN	PNO
12345	1
2345	2
23456	2
23456	1
35346	1
21336	2

S

ESSN
12345
23456

R ÷ S

Additional Relational Operations

Generalized Projection

- Generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list.
- $\Pi_{F1, F2,\ldots,Fn}$ (R)

  Where F_1, F_2,….,F_n are functions over the attributes in the relation R and may involve constants.

Generalized Projection (Example)

Relation Given

- EMPLOYEE (Ssn, Salary, Deduction, Years_service)

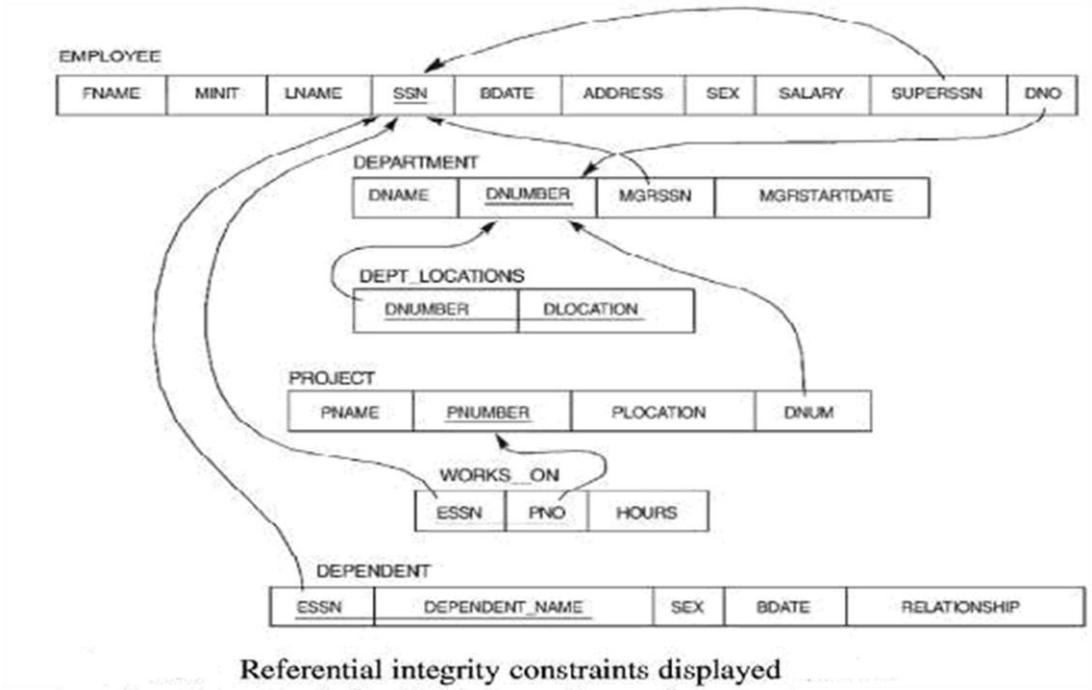
Result Expected
Report (Ssn, Net_salary, Bonus, Tax)

- REPORT← $\rho_{(Ssn,\ Net\_salary,\ Bonus,\ Tax)}$
  $(\Pi_{Ssn,\ Salary\ -\ Deduction,\ 2000*\ Years\_service,\ 0.25*Salary}$ (EMPLOYEE)

Aggregate functions and Grouping

- To specify mathematical aggregate functions on collections of values from the database
- Sum, Average, Maximum and Minimum can be applied to collections of numeric values.
- The COUNT function is used to count tuples or values.
  - avg: average value
  - min: minimum value
  - max: maximum value
  - sum: sum of values
  - count: number of values
- Tuples can be grouped in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.
- An AGGREGATE FUNCTION operation, using the symbol ℱ (pronounced 'script F'), to specify these types of operations as follows:
- <grouping attributes> F <function list> (R)
- Where <grouping attributes>is a list of attributes of the relation specified in R for which grouping is required,
- and <function list> is a list of (<function><attributes>) pairs.
- <grouping attributes> ℱ <function list> (R)
- Dno F COUNTSsn (EMPLOYEE)
- Dno F COUNTSsn , AVERAGESalary(EMPLOYEE)
- Dno F COUNTSsn, AVERAGESalary(EMPLOYEE)
- Result:
  R(DNO, COUNT_SSN, AVERAGE_SALARY)
- F COUNTSsn, AVERAGESalary (EMPLOYEE)
- Result:
  R(COUNT_SSN, AVERAGE_SALARY)
- Retrieve each department number, the number of the employee in the department, their average salary, while renaming the resulting attributes
- $\rho_{R(DNO,\ NO\_OF\_EMPS,\ AVG\_SAL)}$
  (DNO F COUNTSsn,AVERAGESalary(EMPLOYEE))
- Result: R(DNO, NO_OF_EMPS, AVG_SAL)

Examples of queries



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

Q1: Retrieve the name and address of all employees who work for the 'Research' department.

Solution:

- RESEARCH_DEPT <- $\sigma_{DNAME='Research'}$ (DEPARTMENT)
- RESEARCH_EMPS <- (RESEARCH_DEPT $\bowtie_{DNUMBER = DNO}$ EMPLOYEE)
- RESULT <- $\pi_{fname, lname, address}$ (RESEARCH_EMPS)

Q2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address and birth date.

Solution:

STAFFORD_PROJS<- $\sigma_{PLOCATION='Stafford'}$ (PROJECT)

- CONTR_DEPT <- (STAFFORD_PROJS $\bowtie_{DNUM = DNUMBER}$ DEPARTMENT)
- PRJ_DPT_MGR <- CONTR_DEPT $\bowtie_{MGRSSN = SSN}$ EMPLOYEE)
- RESULT <- $\Pi_{PNUMBER,DNUM, LNAME, ADDRESS,BDATE}$ (PRJ_DPT_MGR)

Q3: Find the names of employee who work on all the projects controlled by department no. 5

Solution:

DEPT5_PROJS(PNO)<-$\Pi_{PNUMBER}$ ($\sigma_{DNUM=5}$(PROJECT))

- EMP_PROJ(SSN, PNO)<- $\Pi_{ESSN, PNO}$(WORKS_ON)

RESULT_EMP_SSNS<- EMP_PROJ ÷ DEPT5_PROJS

- RESULT <- $\Pi_{LNAME, FNAME}$(RESULT_EMP_SSNS * EMPLOYEE)

Q4: Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

Solution:

- SMITHS(ESSN) <- $\Pi_{SSN}(\sigma_{LNAME='Smith'}$(EMLPOYEE))
- SMITH_WORKER_PROJ <- Π_{PNO}(WORKS_ON * SMITHS)
- MGRS <- $\Pi_{LNAME, DNUMBER}$(EMPLOYEE ⋈ $_{SSN=MGRSSN}$ DEPARTMENT)

- SMITH_MANAGED_DEPTS(DNUM) <- $\Pi_{DNUMBER}$ ($\sigma_{LNAME='Smith'}$(MGRS))
- SMITH_MGR_PROJS(PNO) <-$\Pi_{PNUMBER}$ (SMITH_MANAGED_DEPTS * PROJECT)
- RESULT <- (SMITH_WORKER_PROJ U SMITH_MGR_PROJS)

Q5**: List the names of all employees with two or more dependents.**

- **(Strictly speaking, this query cannot be done in the *basic relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* DEPENDENT_NAME values.)**

Solution:

- T_1(SSN, NO_OF_DEPTS) <- $_{ESSN}$ ℑ COUNT $_{DEPENDENT\_NAME}$ (DEPENDENT)
- T_2<- $\sigma_{NO\_OF\_DEPTS >=2}$ (T1)

RESULT <- $\Pi_{LNAME, FNAME}$(T_2 * EMPLOYEE

Q6: Retrieve the names of employees who have no dependents.

Solution:

- ALL_EMPS <- Π_{SSN}(EMPLOYEE)
- EMPS_WITH_DEPS(SSN) <- Π_{ESSN} (DEPENDENT)
- EMP_WITHOUT_DEPS <- (ALL_EMPS - EMPS_WITH_DEPS)
- RESULT<- $\Pi_{LNAME, FNAME}$(EMPS_WITHOUT_DEPS * EMPLOYEE)

Q7: List the names of managers who have at least one dependent.

Solution:

- MGRS (SSN) <- Π_{MGRSSN} (DEPARTMENT)

- EMPS_WITH_DPS (SSN) <- Π_{ESSN} (DEPENDENT)
- MGRS_WITH_DEPS <- (MGRS ∩ EMPS_WITH_DPS)
- RESULT <- $\Pi_{LNAME, FNAME}$(MGRS_WITH_DEPS * EMPOYEE).

Relational Database Design by ER- to-Relational Mapping

ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relation Types
Step 4: Mapping of Binary 1:N Relationship Types.
Step 5: Mapping of Binary M:N Relationship Types.
Step 6: Mapping of Multi-valued attributes.
Step 7: Mapping of N-ary Relationship Types.

Step 1: Mapping of Regular Entity Types

- For each regular (strong) entity type (Ei), create a relation that includes all the simple attributes of that Entity.
- Includes only the simple component attributes of a composite attribute (if any) (as required).
- Choose one of the key attributes of Entity as primary key for Relation.
- If the chosen key of Entity is composite , the set of simple attributes together form the primary key of Relation.
- If multiple key attributes exist, information about all secondary keys are kept for indexing and other types of analyses.
- If multiple key attributes exist, information about all secondary keys are kept for indexing and other types of analyses.

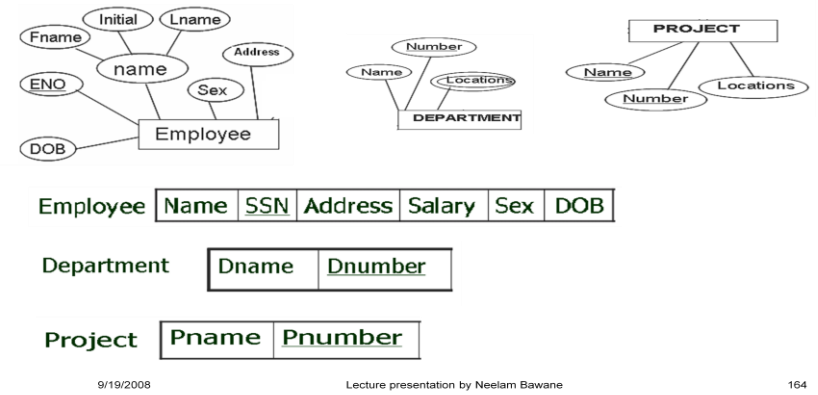
  Example:
- Create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
- SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT.

Step 2: Mapping of Weak Entity Types

- For each weak entity type in the ER schema with owner entity type , create a relation.
- Include all simple attributes (or simple components of the composite attributes) of Weak entity as attributes of Relation .
- Include the primary key attribute (s) of owner entity as foreign key in Weak entity relation.
- The primary key for weak entity relation is the combination of the primary key (s) of the owner (s) and the partial key of the weak entity type .

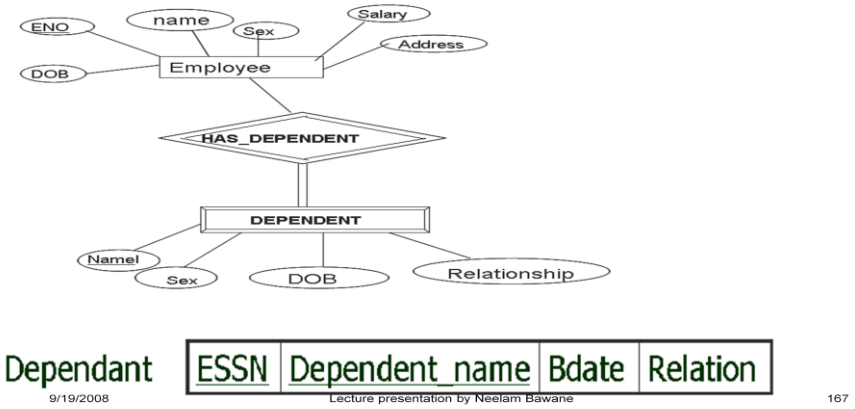
  Example:

Step 1: Mapping of Regular Entity Types.



- Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.
- Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
- The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

Step 2: Mapping of Weak Entity Types



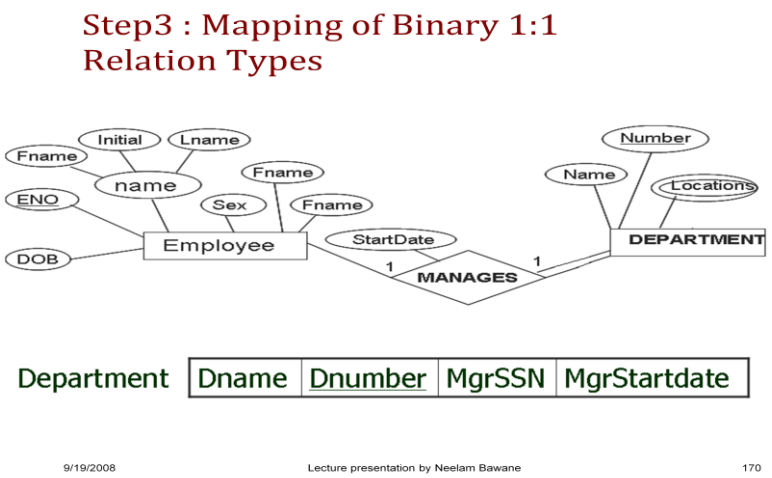
Step3 : Mapping of Binary 1:1 Relation Types

- For each binary 1:1 relation type R in the ER schema, identify the relations S and T that correspond to the entity types participating in the R. There are three possible approaches:

1) the foreign key approach ,
2) the emerged relationship approach , and
3) the cross-reference or relationship relation approach

The foreign key approach

- Mostly used
- Include Primary key of one relation as foreign key in other relation (generally in the relation which has total participation)
- e.g. MGRSSN as foreign key in DEPARTMENT which is primary key in EMPLOYEE due to relation of MANAGES
- Because DEPARTMENT entity participation in manages is total.



The Merged relationship approach

- Merge two entity types and their relationship type in a single relation .
- Suitable only when total participation of both the entity exists.
- But not widely used

The cross-reference or relationship relation approach

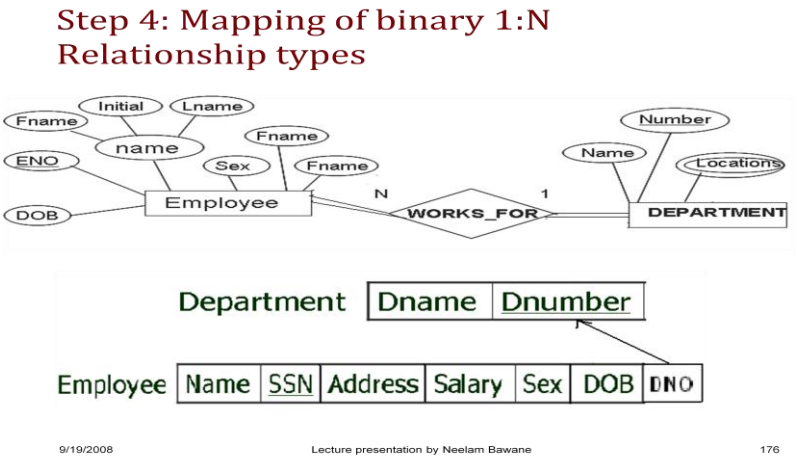
- Create a new relation for relationship type.
- Include Primary keys of both the relations as foreign key in new relation.
- Both primary key together works as composite primary key for new relation.
- Used when both entity participation is partial (to avoid more nulls)

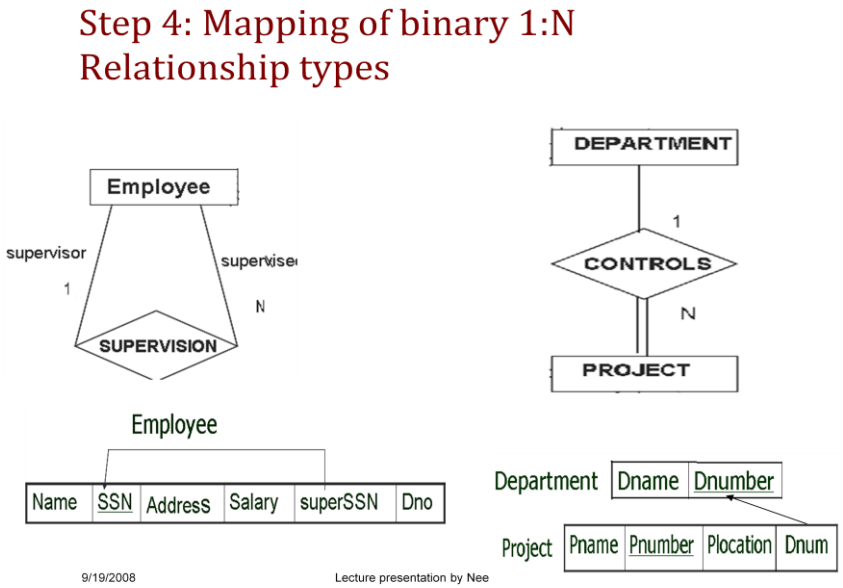
Step 4: Mapping of binary 1:N Relationship types

- For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type.
- Include the primary key of the relation T that represents the other entity type participating in R as foreign key in S.
- This is done because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type.
- Include any simple attributes (or simple components of the composite attributes) of the 1:N relationship type as attribute of S.

Example:

- 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure.
- For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.



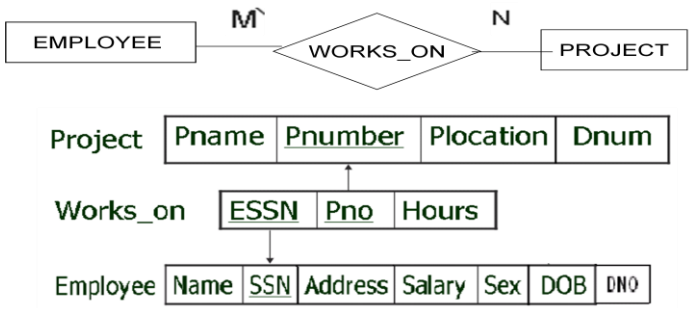


Step 5: Mapping of binary M:N Relationship Types

- For each binary M:N relationship type R, create a new relation S to represent R.
- Include the primary keys of the relations that represent the participating entity types as foreign key attributes in S
- Their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
- We cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate relation S.

Example:

- The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema. The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
- Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.



9/19/2008 Lecture presentation by Neelam Bawane 181

Step 6: Mapping of the multivalued Attributes

- For each multivalued attribute A, create a new relation R.
- This relation R will include an attribute corresponding to A, plus the primary key attribute K of main relation as a foreign key in new relation
- The primary key of R is the combination of A and K .
- If the multivalued attribute is composite, we include its simple components

Example:

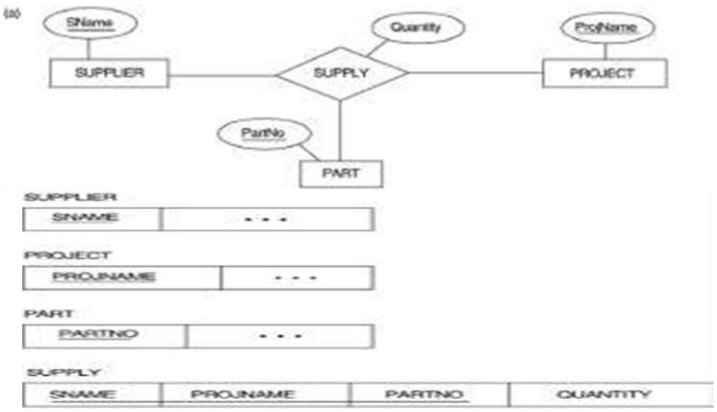
- The relation DEPT_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation. The primary key of R is the combination of {DNUMBER, DLOCATION}.

Step 7: Mapping of N-ary relationship Types

- For each n-ary relationship type , where n>2 , create a new relation S to represent Relationship type.
- Include the primary keys of the relations that represent the participating entity types as foreign key attributes in S .
- Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes for S .
- The primary key of S is usually a combination of all the foreign keys that references the relations representing the participating entity types .
- Step 7: Mapping of N-ary relationship Types

Example:

- The relationship type SUPPY in the ER below. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}


(a)
SName
Quantity
ProjName
SUPPLIER
SUPPLY
PROJECT
PartNo
PART
SUPPLIER
SNAME
- - -
PROJECT
PROJNAME
- - -
PART
PARTNO
- - -
SUPPLY
SNAME
PROJNAME
PARTNO
QUANTITY