
DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 01- INTRODUCTION TO DBMS AND DATABASE DESIGN PROCESS

LECTURE CONTENT

- Introduction to databases, DBMS and their benefits
- Database design process
- Requirement analysis
- Conceptual modelling using EER diagrams
- Design traps

LEARNING OUTCOMES

- Explain what is a database and a DBMS.
- Identify situations where using a database would be beneficial.
- Explain the database design process.
- Draw a EER diagram for a given scenario.

WHAT IS A DATABASE & A DBMS?

- A database is a collection of related data.
- A database management systems (DBMS) is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.
- What do you understand by defining, constructing, manipulating and sharing databases?

WHY A DATABASE?

- Traditionally data are stored in files. So why don't we store all data in files.
- Example :
 - Scenario 1 :A several thousand of students are enrolled for courses at SLIIT. How do you recommend to store these students' data and why?
 - Scenario 2 :During the New Year season you need to buy various groceries and presents for family. How do you recommend to store your shopping list and why?

DATABASE APPROACH VS FILE PROCESSING SYSTEM

- Database system is of self-describing nature
- Database enables insulation between programs and data, and data abstraction
- Databases support of multiple views of the data
- Sharing of data and multiuser transaction processing
- Security and access levels

ADVANTAGES OF DATABASE APPROACH

- **Data independence** – application programs are independent of the way the data is structured and stored
- **Efficient data access** – make the data retrieval efficient using sophisticated techniques to store and access data
- **Enforcing integrity** – provide capabilities to define and enforce constraints
 - Ex: Data type for a name should be string
- **Restricting unauthorized access** – allows only authorized users to access data
- **Providing backup and recovery** – protects users from the effect of system failures
- **Concurrent access** – simultaneous access of data by different users

DATABASE DESIGN PROCESS

- There are six main phases of the process to develop a database
 - Requirement collection and analysis
 - Conceptual database design
 - Logical database design
 - Schema refinement
 - Physical database design
 - Security design

REQUIREMENT COLLECTION AND ANALYSIS

- The purpose of the phase is to collect and analyze the expectations of the users & the intended uses of the database.
- The process would include interviewing clients and analyzing documents such as files used to record data and reports to be generated.
- At the end of the requirement collection, the database developer should identify any unclear or incomplete requirements, redundant information and eliminate them.

REQUIREMENT ANALYSIS (CONTD.)

- Aspects to consider include
 - What data is to be stored in the database?
 - What applications are to be built?
 - What operations have to be performed?

CONCEPTUAL DATABASE DESIGN

- The result of the requirement analysis step is a concisely written set of users' requirements.
- Once, this step is completed, the next step is to create a **conceptual database schema** for the database, using a high-level conceptual data model.
- This step is called **conceptual database design**.
- **Entity-Relationship (ER) model** is a high-level conceptual data model.

ER MODEL - ENTITIES & ATTRIBUTES

- You already know these !
 - Check the handout for the definitions and examples for entities, different types of attributes and keys.
- Select an important entity in a context you are familiar. Add simple attributes, a multivalued attribute, a composite attribute and a key to the entity you identified.
 - Try not to use the attributes shown in the handouts.
- Exchange what you have drawn with your peer. What have they written?

ER MODEL - BINARY RELATIONSHIPS

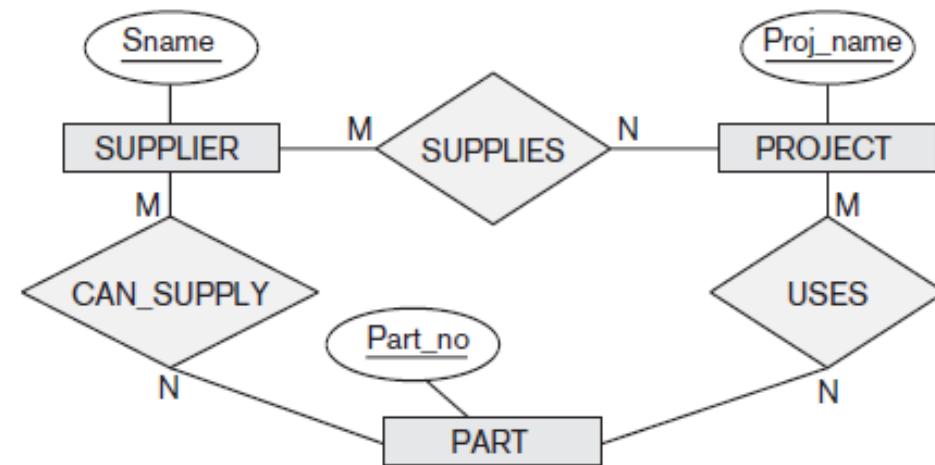
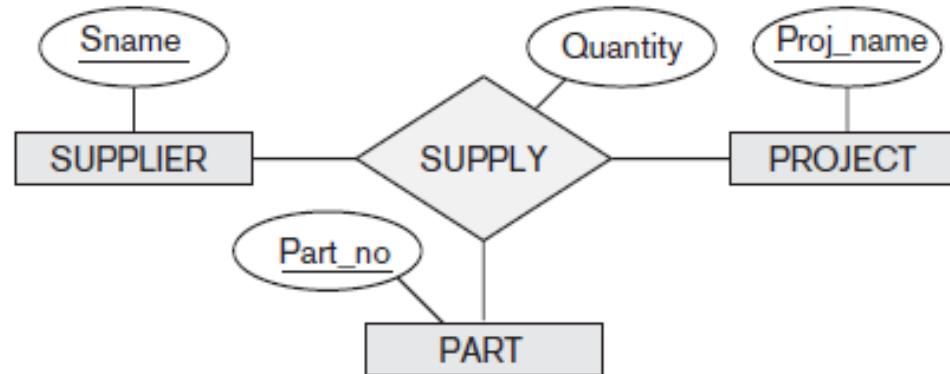
- You already know these too !
 - Check the handout for the definitions and examples for different cardinalities in binary relationships such as 1:1, 1:N and M:N
 - Now draw examples for each cardinality ratio above associated with binary relationships.
 - Exchange what you have drawn with your peer.What has he/she drawn?

ER MODEL - WEAK ENTITIES

- Weak entities are entities that cannot be uniquely identified alone in a domain.
- Following restrictions must hold with relevance to weak entities
 - The owner entity set and the weak entity set must participate in a one-to-many relationship set (one owner entity is associated with one or more weak entities, but each weak entity has a single owner). This relationship set is called the **identifying relationship set** of the weak entity set.
 - The weak entity set must have total participation in the identifying relationship set.
- Can you think of an weak entity in a domain you know?

ER MODEL - TERNARY RELATIONSHIPS

- A ternary relationship is when three entities participate in the relationship.
- When to use ternary vs binary?



- Ternary relationships could be used when there is a common attribute that needs all three entities together, (ex: quantity)
- Include the ternary relationship *plus* one or more of the binary relationships, if they represent different meanings and if all are needed

ACTIVITY

- Draw an ER diagram for the scenario below.
 - A Library is organized into several sections such as fiction, children and technology. Each section has a name and a number(unique) and its headed by a head librarian.
 - Each book belong to a section and has a title, authors, ISBN, year and a publisher.
 - A book may have several copies. Each copy is identified by an access number.
 - For each copy borrowed, current borrower and due date should be tracked.
 - Members have a membership number, an address and a phone number.
 - Members can borrow 5 books and could put hold request on a book
 - Librarian has a name, id number, phone and an address.

EER MODEL

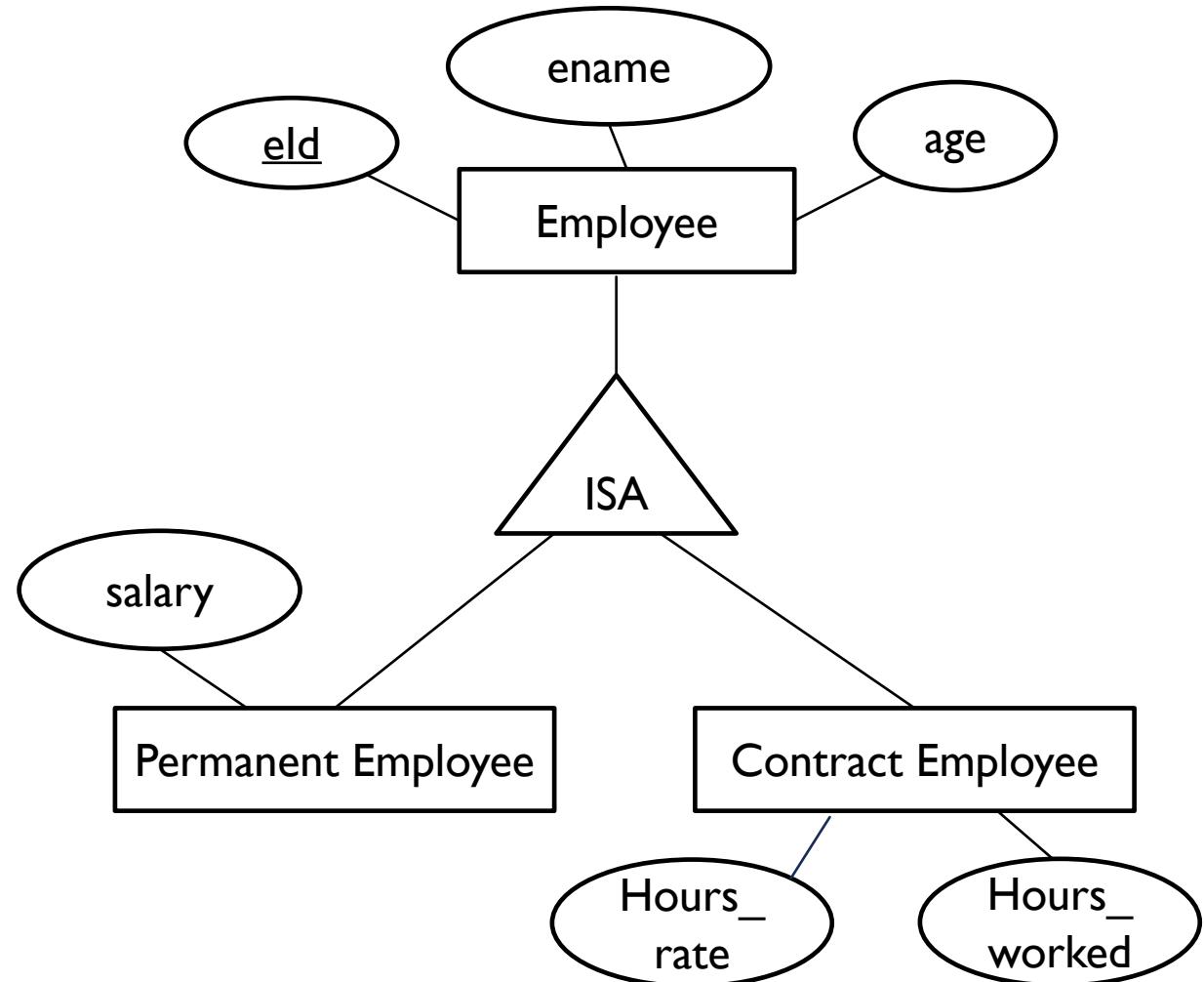
- ER model we discussed so far has been enhanced by adding several new concepts leading to the development of the EER model.
- An important extension included in the EER model is the specialization and generalization concepts.
 - Specialization is the process of defining a set of subclasses of an entity type.
 - Employee & permanent employee
 - Generalization is the process of identifying commonalities between entity types and grouping them as super-classes.

EER MODEL - ISA RELATIONSHIPS

- In many cases an entity type has numerous subgroupings or subtypes of its entities that are meaningful and need to be represented explicitly because of their significance to the database application.
 - Ex: the entities that are members of the EMPLOYEE entity type may be distinguished further into contract employees and permanent employees
- Such subtypes could be represented in EER diagrams using 'ISA' relationships

EER MODEL - ISA RELATIONSHIPS (CONTD.)

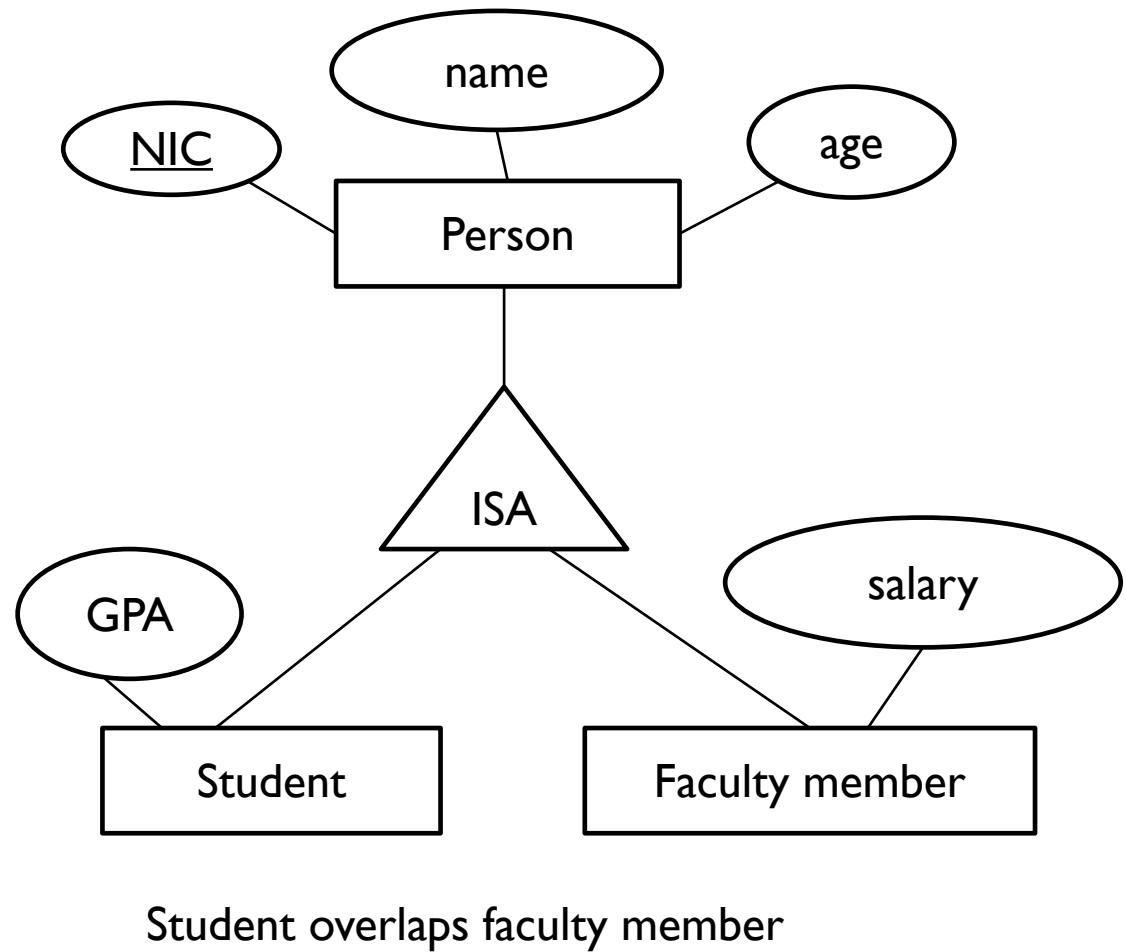
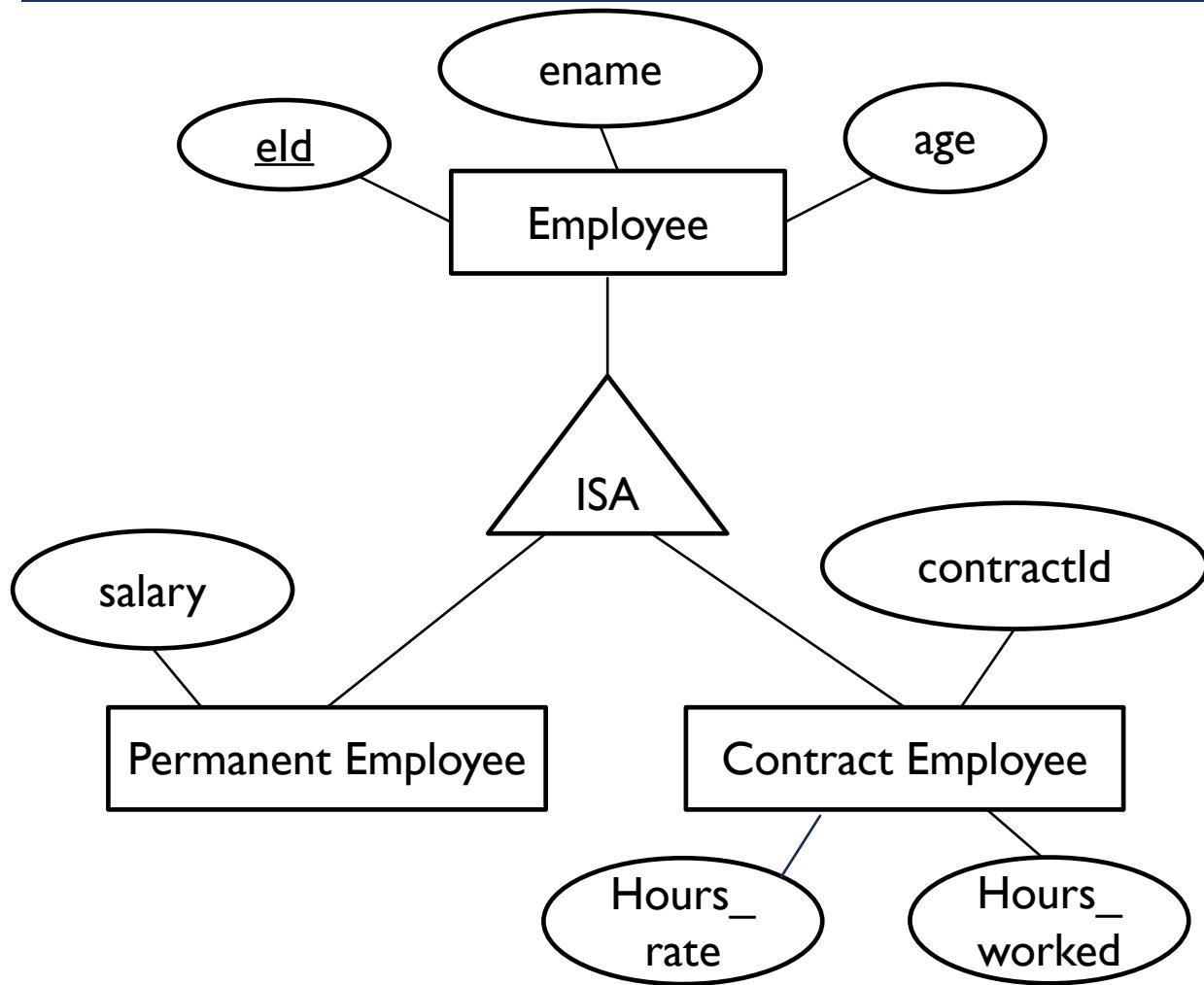
- Note that the subclasses may have their own attributes and relationships.
- Every entity in the subclass is also an employee entity and have all the have all of the attributes of Employees entity.
- Thus attributes of the permanent employee include all attributes of employee entity and those of permanent employee.



OVERLAPPING CONSTRAINT

- **Overlapping constraint** determine whether two subclasses are allowed to contain the same entity.
 - For example can an employee E be a permanent employee and a contract employee? Probably not. Therefore, the permanent employee subclass and the contract employee subclass are **disjoint**.
 - Can a person P in a university environment be a student and a faculty member at the same time. If it is so, we denote this by writing **student overlaps faculty member**. In absence on such a statement we assume that the sub classes are disjoint.

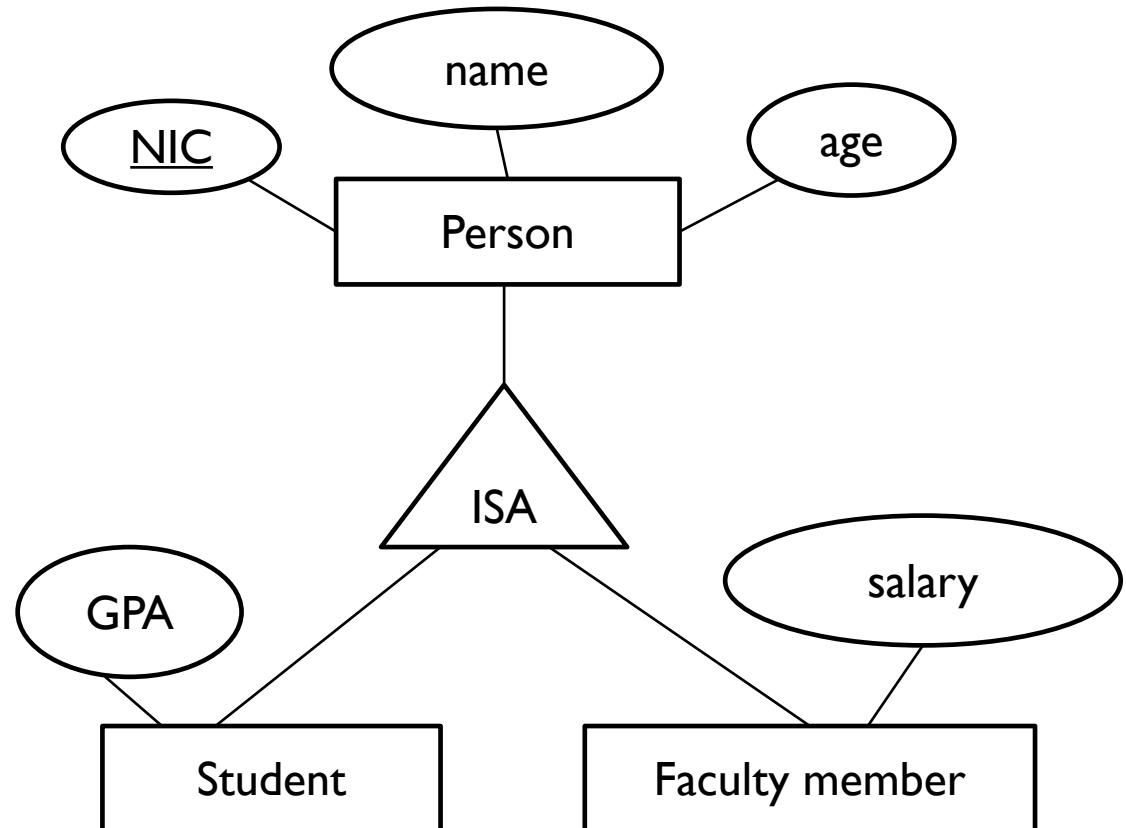
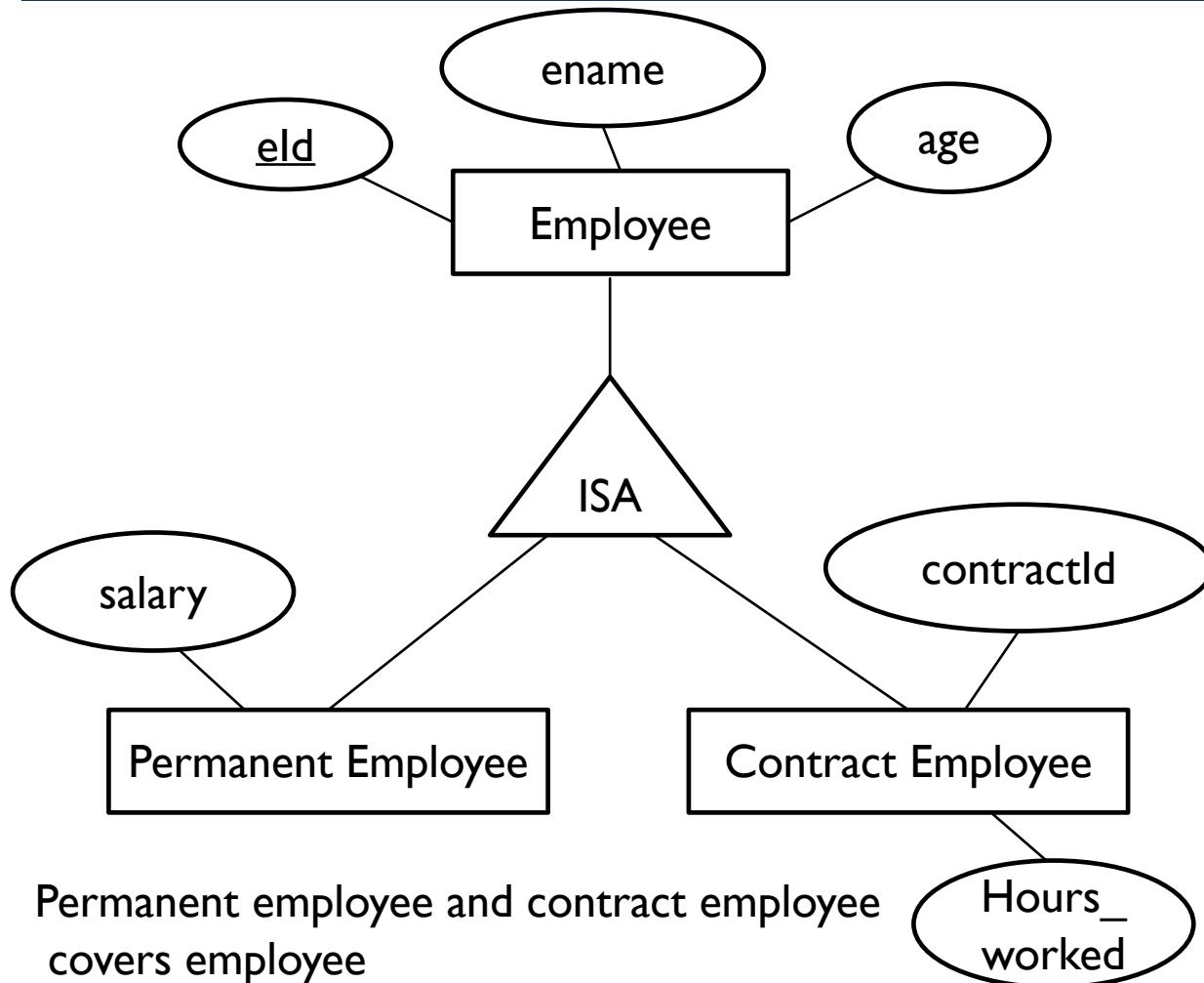
OVERLAPPING CONSTRAINT (CONTD.)



COVERING CONSTRAINT

- **Covering constraints** determine whether the entities in the subclasses collectively include all entities in the super class.
 - For example, does every employee entity e, belong to one of its subclasses (i.e. permanent employee or contract employee)? If so we denote this by writing **permanent employee and contract employee covers employee**.
 - Does every person p in a university environment belong to either student sub class or the faculty member sub class? Probably not. Therefore, there is no covering constraint associated with the hierarchy.
- Existence of a covering constraint is also known as having a **total specialization**.
- Absence of a covering constraint in a class hierarchy is known as **partial specialization**.

COVERING CONSTRAINT (CONTD.)



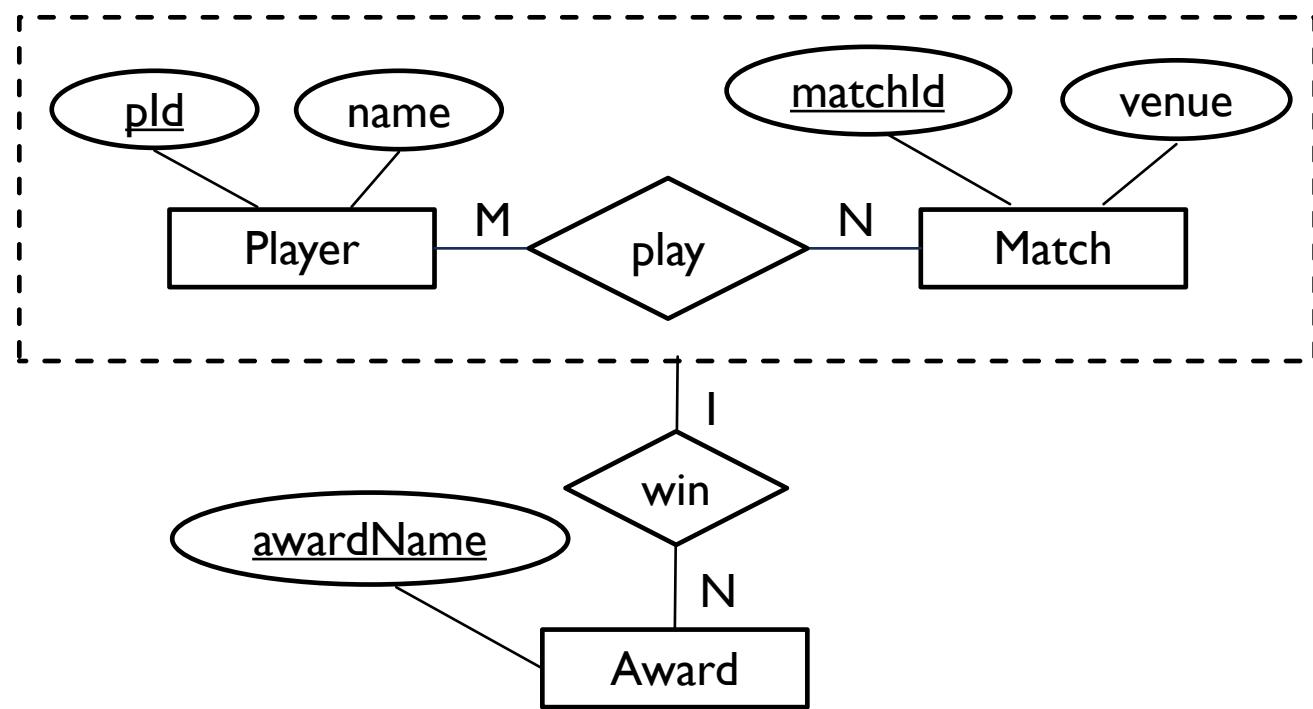
Permanent employee and contract employee
covers employee

ACTIVITY

- In a blank paper draw two ISA relationships, one which is having a covering constraint and another which is having a overlapping constraint.
- Exchange what you have drawn with your peer. What have they drawn?
- Have you understood the concepts properly?

EER MODEL - AGGREGATION

- **Aggregation** allows us to indicate that a relationship as above which is between a relationship set that set participates in another relationship set.
- For example, a cricket players play in cricket matches. When he plays a match for his performance he may win awards.
- Note that, the difference between ternary and aggregation is that aggregation contain two independent relationships whereas in ternary relationship there is one.



ACTIVITY

- Can you think of an aggregation relationship in a domain familiar to you?
- Exchange what you have drawn with your peer. What have they drawn?
- Have you understood the concepts properly?

ACTIVITY

- Draw an E-ER diagram for the following requirements.
 - Students contain an id (unique), name and an address.
 - There are academic semesters containing an semester id (unique), semester and year.
 - There are courses offered during academic semesters. A course has a *number* (unique), *name* and *credits*.
 - Students make payments. A payment has *receipt number* (unique), *amount* and *date*.
 - Payments can be classified into Tuition (semester payment), Examination and other (Library fine, Printouts).
 - A Tuition payment is made for an academic semester
 - For other payments description should be stored
 - Students register for courses offered during a particular semester. The registered date must be stored in the database.

DESIGN TRAPS

- There are several different "modeling traps" (called *connection traps*) that you can fall into when designing your ER model.
- Two connection traps that we will look at are:
 - Fan traps
 - Chasm traps

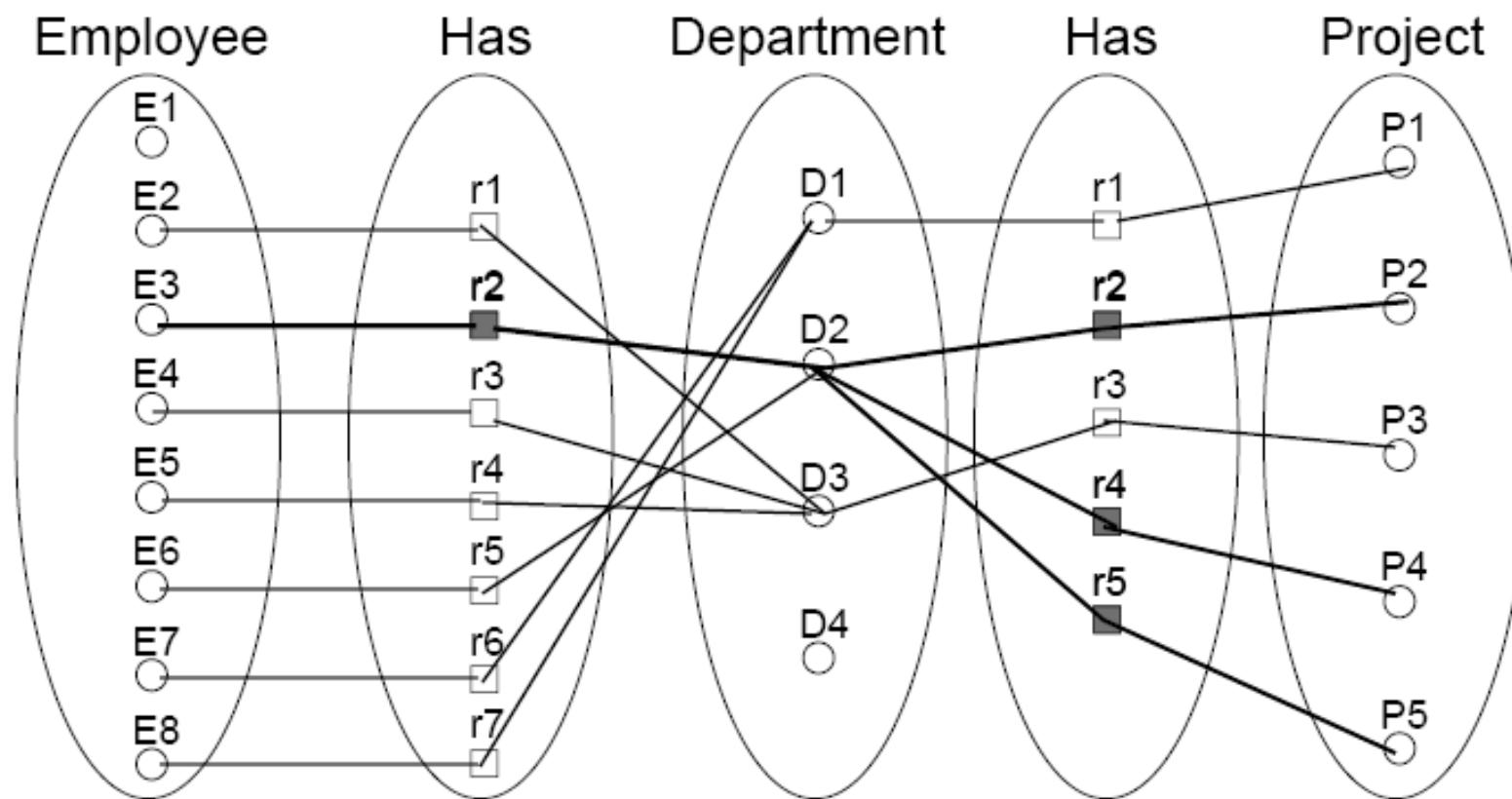
FAN TRAP

- A ***fan trap*** is when a model represents a relationship between entity types, but the pathway between certain entity instances is ambiguous.
 - Often occurs when two or more one-to-many relationships fan out (come from) the same entity type.
- Example: A department has multiple employees, a department has multiple projects, and each project has multiple employees.



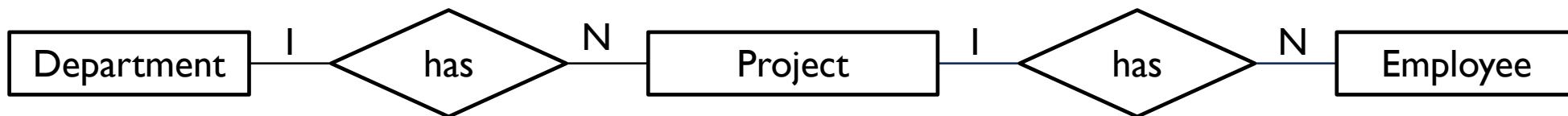
FAN TRAP (CONTD.)

- Which projects does employee E3 work on?



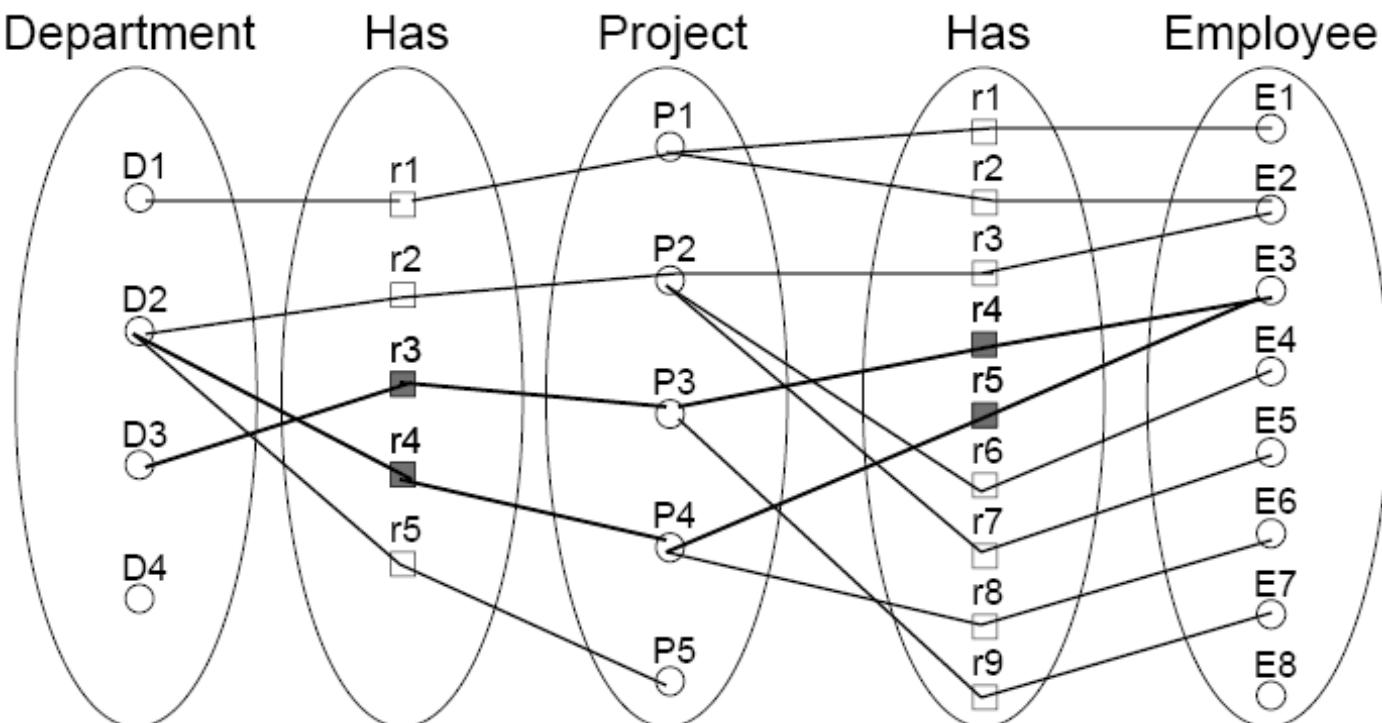
CHASM TRAP

- A ***chasm trap*** occurs when a model suggests that a relationship between entity types should be present, but the relationship does not actually exist. (*missing relationship*)
 - May occur when there is a path of optional relationships between entities.
- Example: A department has multiple employees, a department has multiple projects, and each project has multiple employees.



CHASM TRAP (CONTD.)

- Which department is employee E8 in?
- What are the employees of department D4?



WHAT YOU HAVE TO DO BY NEXT WEEK

- Try out the self-test questions on the course web.
- Try out tutorial and bring the answers to the class.
- Answer the questions at the end of chapter 2 of Database Management Systems by Ramakrishnan & Gehrke

DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 02- ER AND EER TO RELATIONAL MODEL MAPPING

LECTURE CONTENT

- Logical database design
- Relational model and its components
- ER to relational mapping

LEARNING OUTCOMES

- Explain the process of logical database design
- Explain relational model and its components
- Convert a complex ER model to the relational model

LOGICAL DATABASE DESIGN

- Once we finished the step of conceptual database design we next select a DBMS to implement our database design.
- We then convert the conceptual database design into a database schema in the data model of the chosen DBMS.
- Before 1970 most database systems were based on two older data models namely, hierarchical model and network model.
- Leading DBMS products nowadays are based on the relational model which introduced by Codd in 1970.

THE RELATIONAL MODEL

- The major advantage of relational model is its simplicity in data representation.
- A **relational database** is a collection of relations with distinct relation names.
- The main construct representing data in the relational model is the **relation**.
- A relation consists of a **relational schema** and a **relational instance**.

THE RELATIONAL MODEL (CONTD.)

- The relational schema describes the columns for a relation.
 - Ex: Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
 - The schema specifies the relation's name, the name of each **field** and the **domain** of each field.
- An **instance** of a relation is a set of **tuples**, also called **records**, in which each tuple has the same number of fields as the relation schema.
 - A relation instance can be thought of as a *table* in which each tuple is a *row*, and all rows have the same number of fields.

<i>sid</i>	<i>name</i>	<i>login</i>	<i>age</i>	<i>gpa</i>
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8

INTEGRITY CONSTRAINTS

- An integrity constraints (IC) is a condition specified on a database schema and restricts the data that could be stored in an instance of the database.
- If a database instance satisfies all the integrity constraints specified on the database schema, it is a **legal** instance.
- There are several types of integrity constraints
 - Domain constraints
 - Referential integrity constraints
 - Key constraints
 - Other constraints

INTEGRITY CONSTRAINTS (CONTD.)

- Domain Constraint
 - Domain constraint specifies that the values that appear in a column must be drawn from the domain associated with that column.
 - Relational database provides data types to specify valid domains.
- Key Constraint
 - The minimal set of attributes that uniquely identify a tuple is called the **key** of a relation
 - A set of fields that uniquely identifies a tuple according to a key constraint is called a candidate key for the relation; we often abbreviate this to just key.
 - One of the candidate keys is designated as the primary key.

INTEGRITY CONSTRAINTS (CONTD.)

- Referential integrity Constraint
 - The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations.
 - Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
 - Foreign keys enforce referential integrity constraints
 - Foreign key attributes in R₁ referring to R₂ have the following rules:
 - The FK attributes in R₁ have the *same domain(s)* as the primary key attributes if R₂
 - The value of FK in tuple t₁ in R₁ must reference an existing PK value in tuple t₂ of R₂
 - We can diagrammatically display the foreign keys by drawing an arrow from the foreign key to the primary key

INTEGRITY CONSTRAINTS (CONTD.)

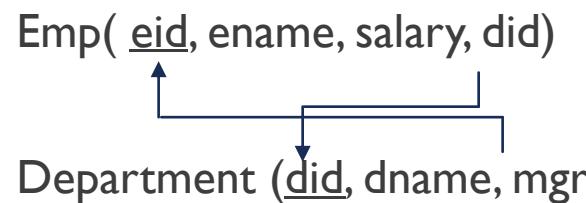
■ Other constraints

■ **Table Constraints** (constraints within tables)

- Example : Balance of the account should be greater than 0
- In SQL, CHECK constraint can be used

■ **Assertions** (constraints between multiple tables)

- For example: consider the following schema



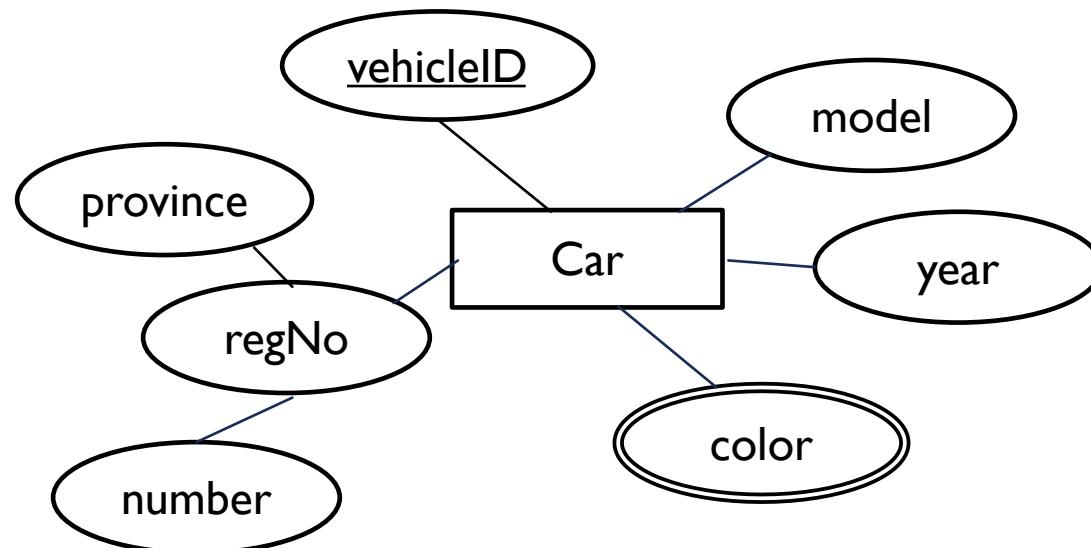
- Suppose that an employee's salary should not exceed his/her manager's salary
- Such constraints can be using specific language features such as triggers and assertions

ACTIVITY

- In two minutes summarize the integrity constraints discussed in today's lecture on a blank paper.
- Exchange what you have written with your peer and correct the answer.
- Have you understood each constraint well?

MAPPING ENTITIES AND ATTRIBUTES

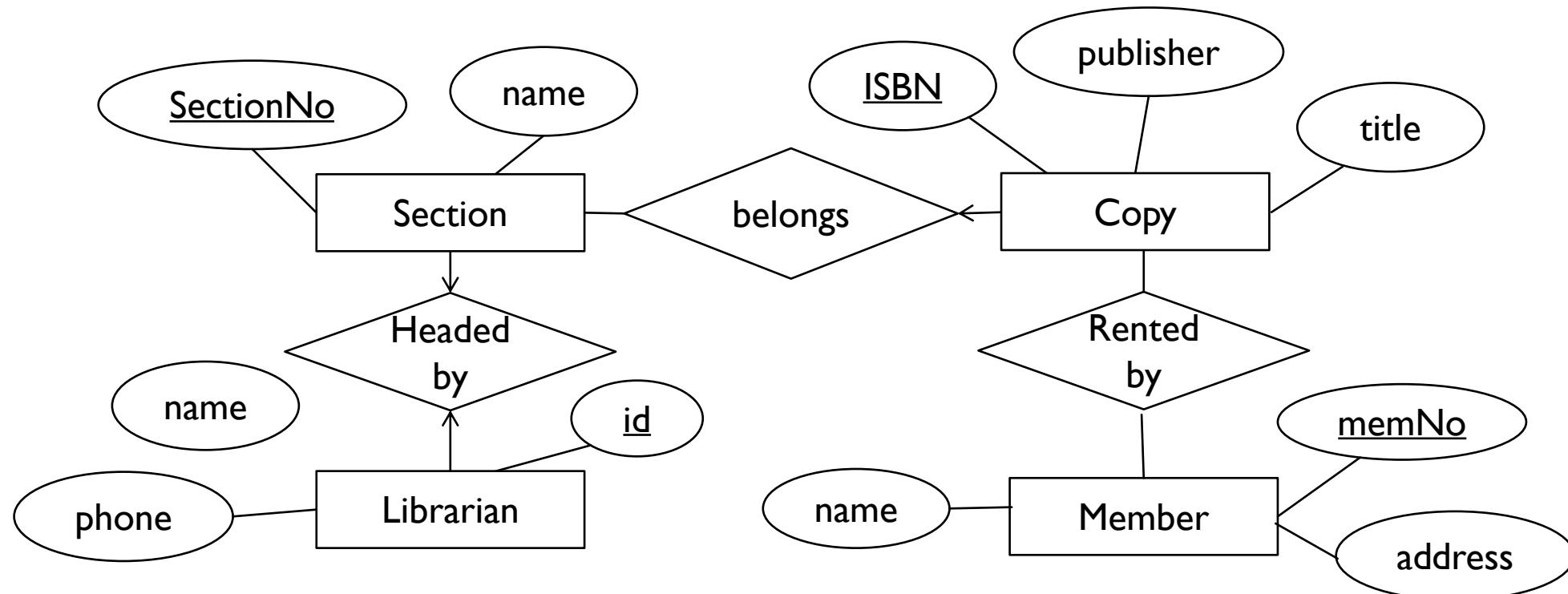
- Go through page 1 of the handout 2.
- Map the entity below to relational model.



- Compare what you have written with your peer.

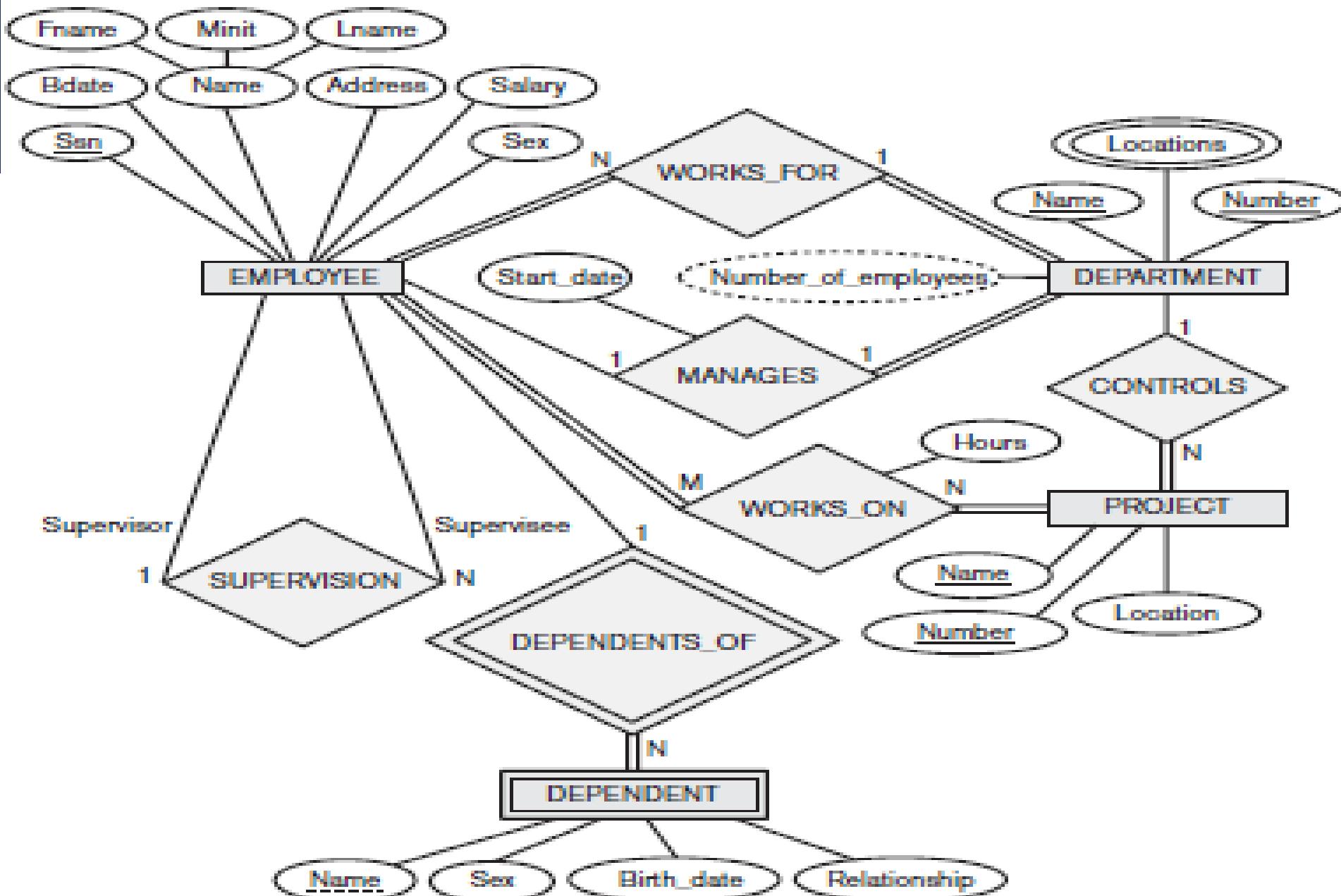
MAPPING BINARY RELATIONSHIPS

- Go through page 1 of the handout 2.
- Now lets map the following ER diagram to relational model



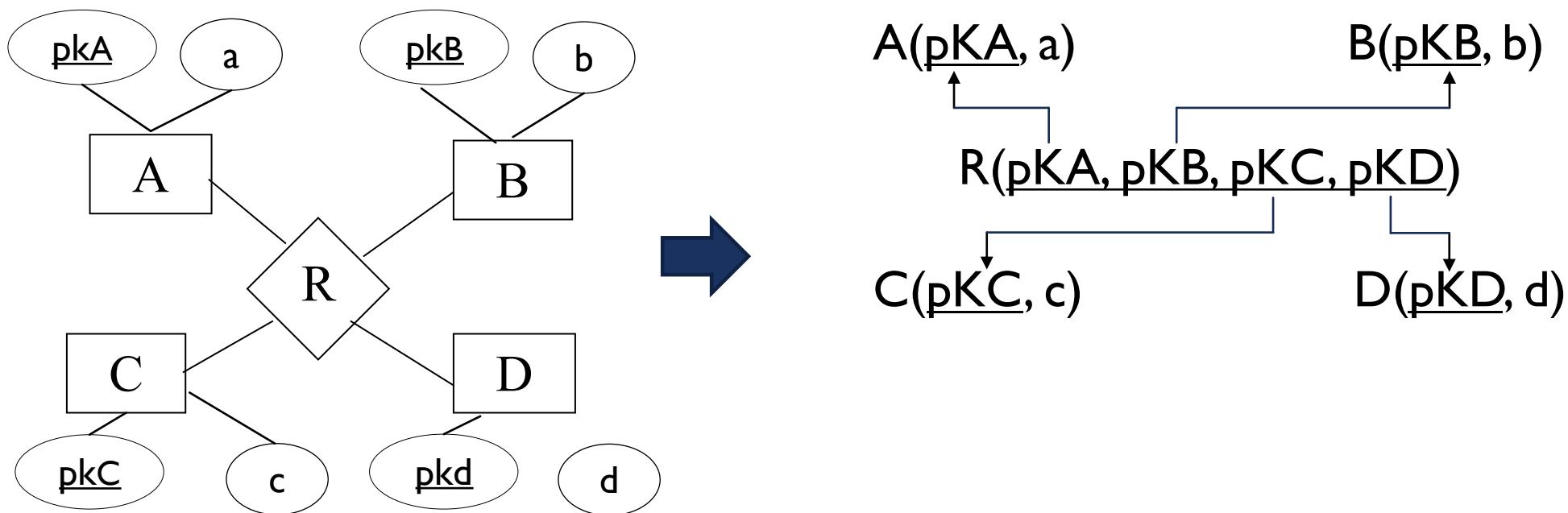
ACTIVITY

- Map the ER diagram to relational model



MAPPING N-ARY RELATIONSHIPS

- N-ary relationship is mapped in to a “Relationship” relation and foreign keys

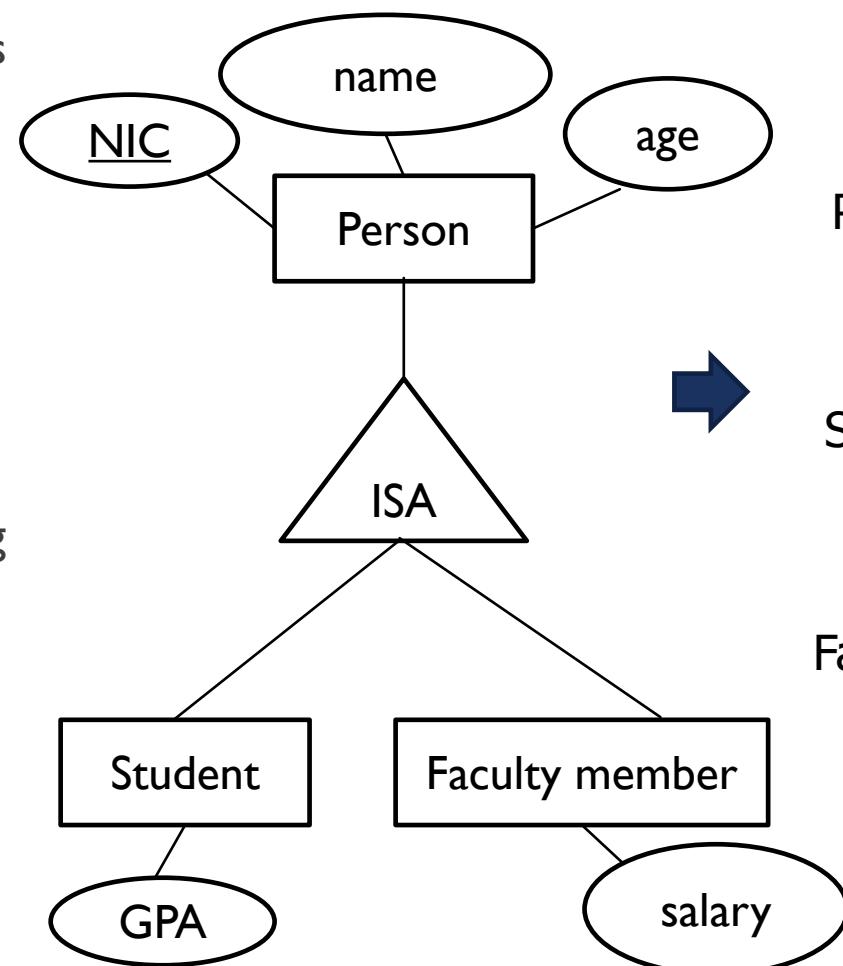


MAPPING ISA-RELATIONSHIPS

- There are four different options for mapping ISA relationships.
 - Multi-relation options : option 1 & option 2
 - Single-relation options : option 3 & option 4
- Each option is suitable for specific situations.

ISA MAPPING – OPTION I

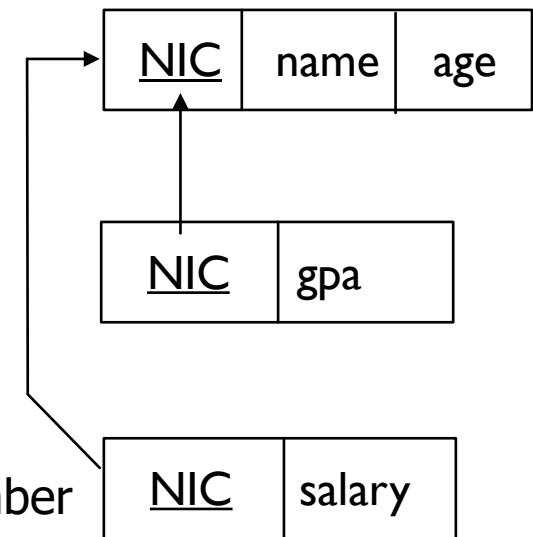
- Create a relation for the superclass with its attributes. Primary key of the superclass becomes the primary key of the relation.
- Create separate relations for the sub classes with their attributes. Primary key of the superclass is also primary key of each subclass. They are also foreign keys referring to the primary key of the relation created for the super class.
- **Option I works for all constraints disjoint, overlapping, total and partial**



Person

Student

Faculty member



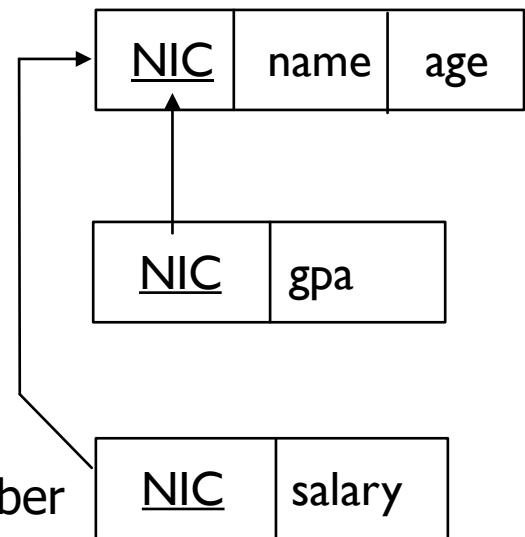
ISA MAPPING – OPTION I (CONTD.)

- Now think how you would store information of following people
 - A person who is not a student or a faculty member(i.e when the ISA relationship is parial)
 - A person who is a student but not a faculty member (i.e disjoint)
 - A person who is both a student and a faculty member (i.e: overlapping classes)

Person

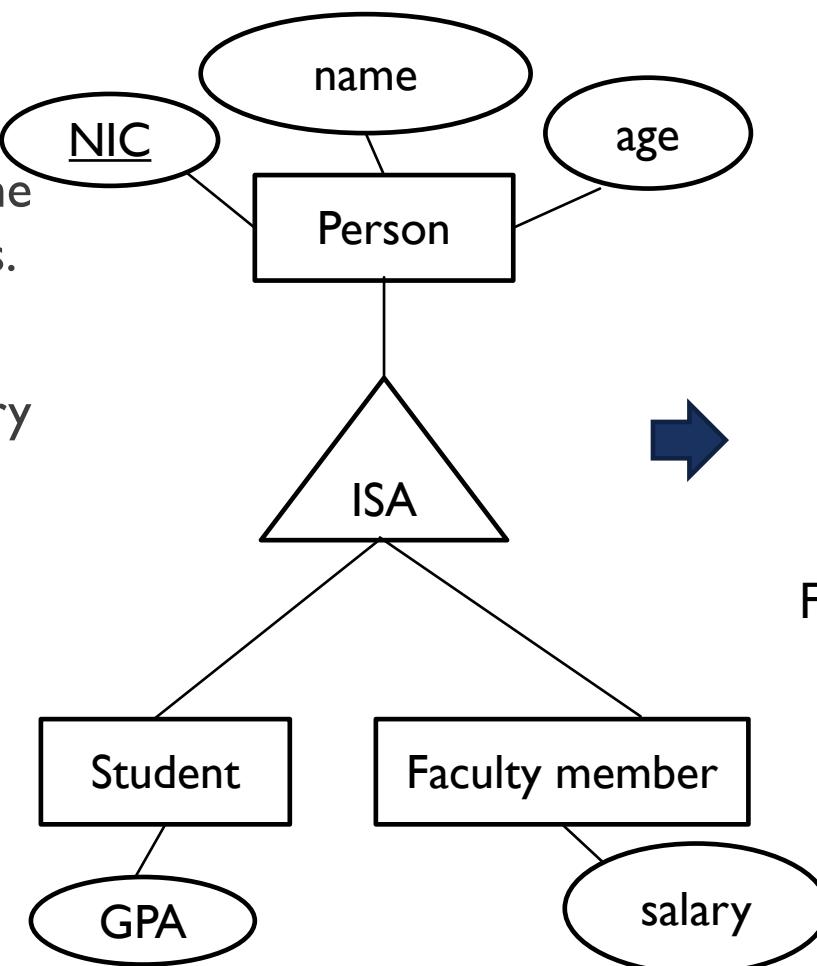
Student

Faculty member



ISA MAPPING – OPTION 2

- Create separate relations for all the subclasses with their own attributes and the attributes of the superclass.
- Primary key of the superclass becomes primary key of the subclasses.
- **The ISA relationship must be total (i.e. subclasses must cover the super class)**



Student

NIC	name	age	gpa
-----	------	-----	-----

Faculty member

NIC	name	age	salary
-----	------	-----	--------

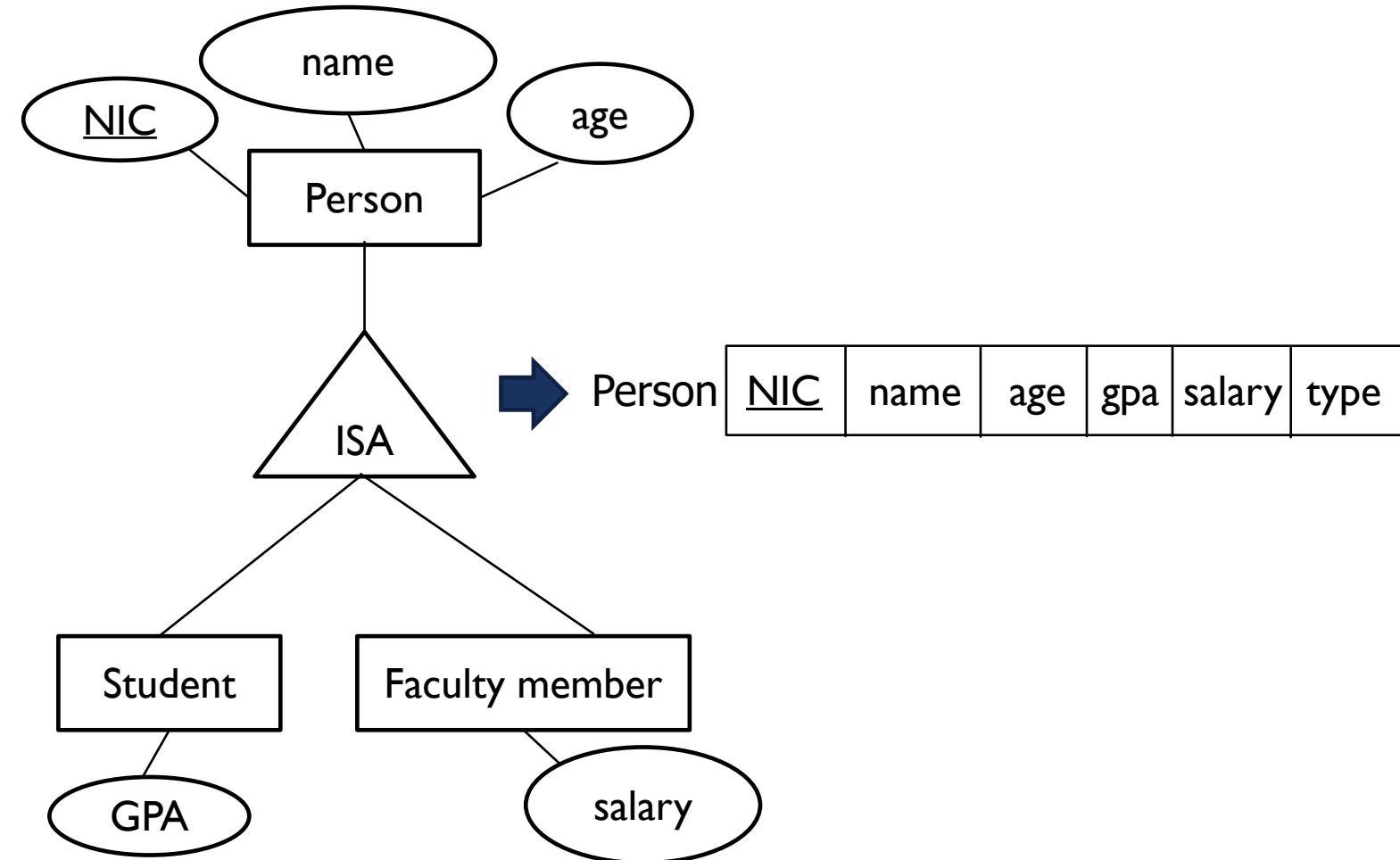
ISA MAPPING – OPTION 2 (CONTD.)

- Now think how you would store information of following people
 - A person who is not a student or a faculty member(i.e when the ISA relationship is parial)
 - A person who is a student but not a faculty member (i.e disjoint)
 - A person who is both a student and a faculty member (i.e: overlapping classes)

Student	<table border="1"><tr><td>NIC</td><td>name</td><td>age</td><td>gpa</td></tr></table>	NIC	name	age	gpa
NIC	name	age	gpa		
Faculty member	<table border="1"><tr><td>NIC</td><td>name</td><td>age</td><td>salary</td></tr></table>	NIC	name	age	salary
NIC	name	age	salary		

ISA MAPPING – OPTION 3

- Create a single relation including attributes of the superclass as well as attributes of all subclasses.
- Include an attribute named type for specifying which subclass the entity belongs if any.
- Primary key of the superclass becomes primary key of the relation.
- The specialization/generalization relationship must be **disjoint**
- Good if subclasses have **few attributes**



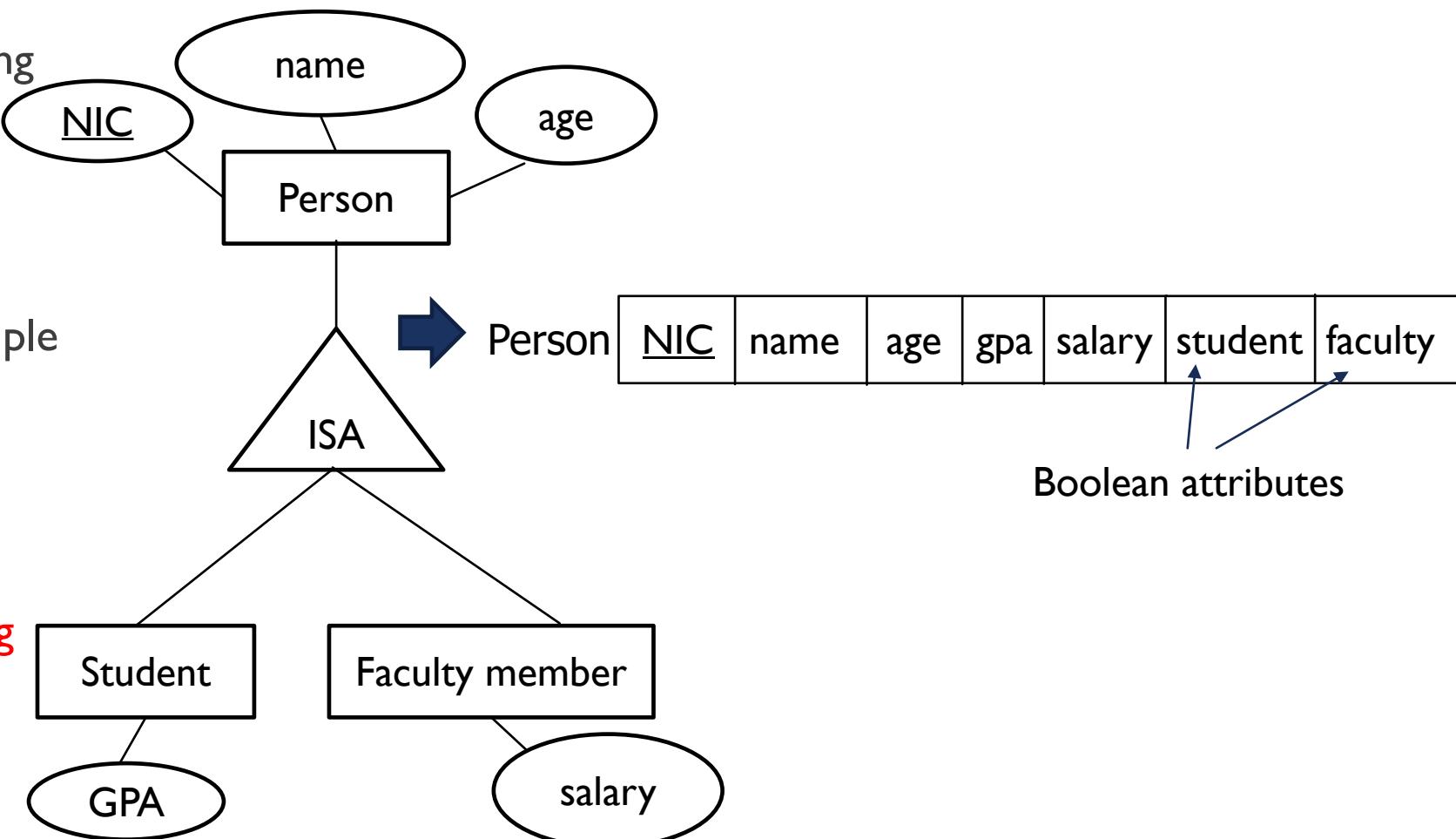
ISA MAPPING – OPTION 3 (CONTD.)

Person	<u>NIC</u>	name	age	gpa	salary	type
--------	------------	------	-----	-----	--------	------

- Now think how you would store information of following people
 - A person who is not a student or a faculty member(i.e when the ISA relationship is parial)
 - A person who is a student but not a faculty member (i.e disjoint)
 - A person who is both a student and a faculty member (i.e: overlapping classes)

ISA MAPPING – OPTION 4

- Create a single relation including attributes of the superclass as well as attributes of all sub classes.
- Include a Boolean attribute to indicate which subclass each tuple belongs to
- Primary key of the superclass becomes primary key of the relation.
- This relation allows overlapping constraints for specialization/generalization relationship



ISA MAPPING – OPTION 4 (CONTD.)

Person	<u>NIC</u>	name	age	gpa	salary	student	faculty
--------	------------	------	-----	-----	--------	---------	---------

- Now think how you would store information of following people
 - A person who is not a student or a faculty member(i.e when the ISA relationship is parial)
 - A person who is a student but not a faculty member (i.e disjoint)
 - A person who is both a student and a faculty member (i.e: overlapping classes)

ACTIVITY

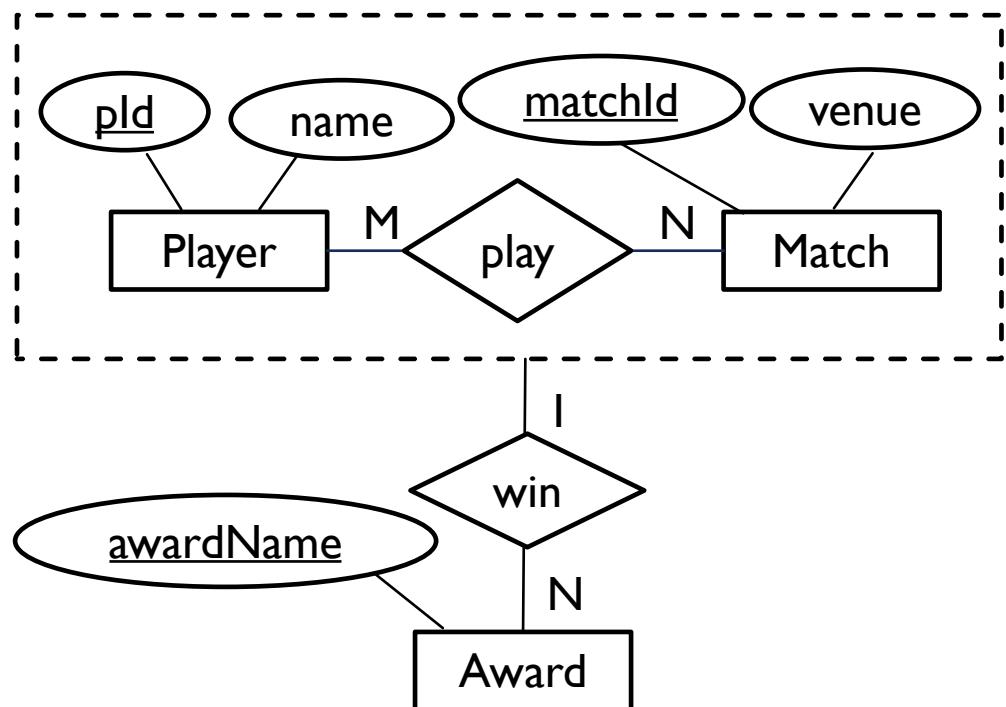
- Fill the table below indicating the which constraints work with which option.

	Overlapping	Disjoint	Total	Partial
Option 1				
Option 2				
Option 3				
Option 4				

MAPPING AGGREGATION RELATIONSHIPS

- Aggregation mapping could be performed in two steps.
 - Step 1: First map the aggregation relationship R.
 - Step 2 :To map relationship set involving aggregation of R, treat the aggregation like an entity set whose primary key is the primary key of the table for R

MAPPING AGGREGATION RELATIONSHIPS (CONTD.)



- Step 1: First map the aggregation relationship R.

Player (pId, name)
Match (matchId, venue)
Play (pId, matchId)

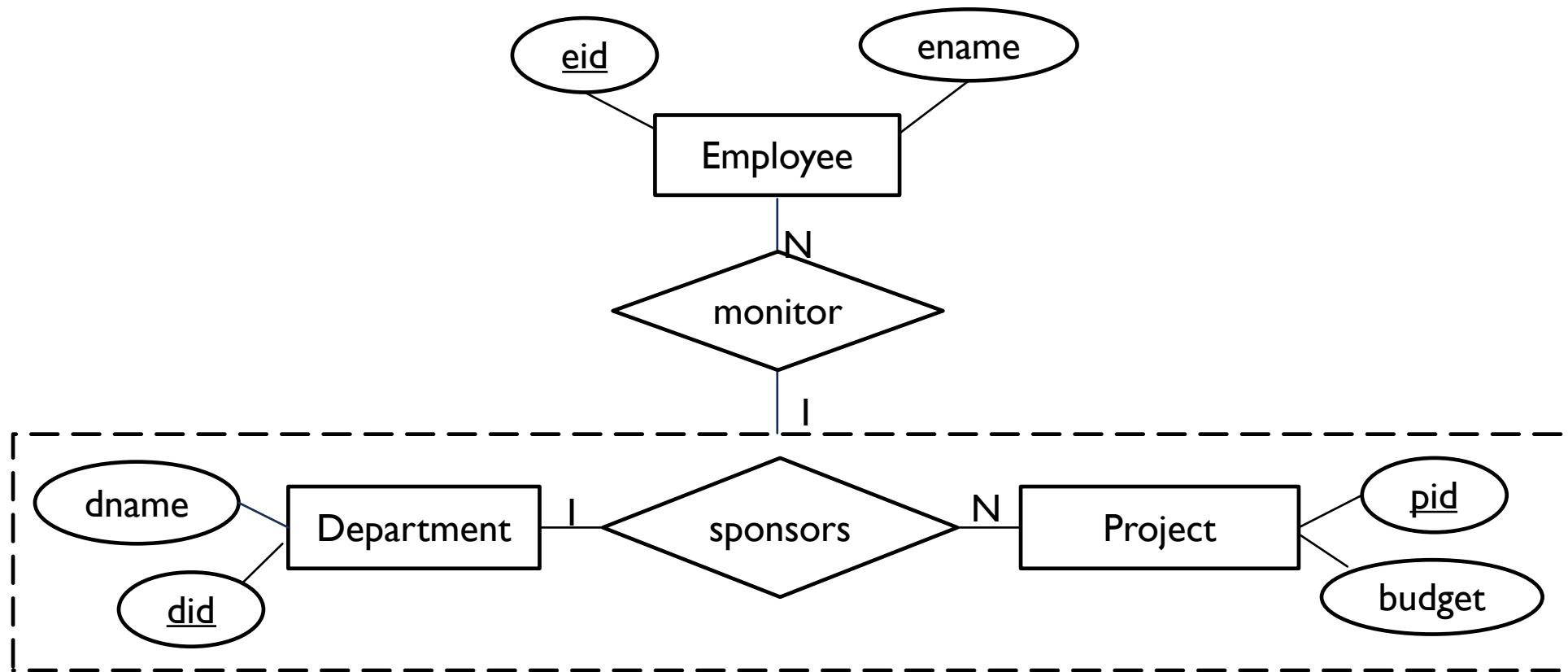
- Step 2 : To map relationship set involving aggregation of R, treat the aggregation like an entity set whose primary key is the primary key of the table for R.

Table for aggregation is 'Play'. Thus, the relationship win should be mapped considering 1:N relationship between play (primary key : pId,matchId) and award,

Award (awardName, pId, matchId)

MAPPING AGGREGATION RELATIONSHIPS (CONTD.)

- Map the following EER diagram to relational model.



WHAT YOU HAVE TO DO BY NEXT WEEK

- Try out the self-test questions on the course web.
- Complete the tutorial.
- Complete the lab sheet

DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 03- SCHEMA REFINEMENT

LECTURE CONTENT

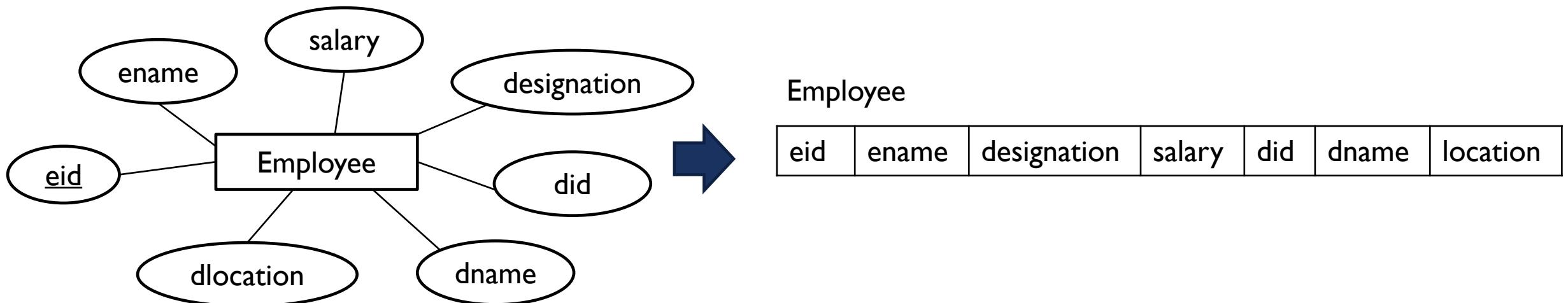
- Why schema refinement?
- Properties of good decomposition
- Functional dependencies
- Computing keys of relations
- Normalization and normal forms

LEARNING OUTCOMES

- Explain the pitfalls of incorrect grouping of attributes in relations.
- Explain the properties of a good decomposition.
- Compute keys from a given set of FDs in a relation
- Normalize a given schema to BCNF.

WHY SCHEMA REFINEMENT?

- The relations resulted through the logical database design may not be very good if your conceptual database design is not good.
- For example, what is wrong with the following schema resulted through mapping of an ER diagram



WHY SCHEMA REFINEMENT? (CONTD.)

- Schemas such as in the previous slide lead to several anomalies while inserting, updating and deleting and wasting of space due to redundancies of data.
- Can you spot where data are duplicated and issues may cause during insert/update & delete?

<u>eid</u>	Ename	Designation	Salary	did	dname	location
1000	Ajith	Lecturer	60000	1	Academic	malabe
1001	Sunil	Executive	45000	3	Maintenance	Kandy
1002	Kamal	Lecturer	75000	1	Academic	malabe
1003	Piyumi	Manager	50000	2	Admin	metro
1004	Roshan	Lecturer	35000	1	Academic	malabe
1005	Nuwan	Lecturer	80000	1	Academic	malabe
1006	Jayamini	Assistant	25000	2	Admin	metro
1007	Nishani	Lecturer	42000	1	Academic	malabe
1008	Amal	Assistant	28000	4	ITSD	Matara

WHY SCHEMA REFINEMENT? (CONTD.)

- Insertion Anomaly
 - Inserting a new employee to the emp table
 - Department information is repeated (ensure that correct department information is inserted).
 - Inserting a department with no employees
 - Impossible since eid cannot be null
- Deletion Anomaly
 - Deleting the last employee from the department will lead to loosing information about the department
- Update Anomaly
 - Updating the department's location needs to be done for all employees working for that department

WHY SCHEMA REFINEMENT? (CONTD.)

- To solve the issues discussed previously, we should decompose the relations to smaller relations.
- For example, we can decompose the emp relation discussed previously into two relations as below to overcome the issues of redundancies and anomalies.

<u>eid</u>	Ename	Designation	Salary	did
1000	Ajith	Lecturer	60000	1
1001	Sunil	Executive	45000	3
1002	Kamal	Lecturer	75000	1
1003	Piyumi	Manager	50000	2
1004	Roshan	Lecturer	35000	1
1005	Nuwan	Lecturer	80000	1
1006	Jayamini	Assistant	25000	2
1007	Nishani	Lecturer	42000	1
1008	Amal	Assistant	28000	4

did	dname	location
1	Academic	malabe
2	Admin	metro
3	Maintenance	Kandy
4	ITSD	Matara

WHY SCHEMA REFINEMENT? (CONTD.)

- Random decompositions however, may introduce new problems.
- Two properties that could be looked up to ensure that the relations resulted from decomposition are good are as follows:
 - Loss-less join property
 - Dependency preserving property

WHY SCHEMA REFINEMENT? (CONTD.)

- **Loss-less join property** : the property enables recovery of original relation from a set of smaller relations resulted through decomposition.
 - For example, suppose S is decomposed in to R₁ and R₂. When we join R₁ and R₂ do we get S? if so, we say the decomposition is loss-less.

	S	P	D
S	S1	P1	D1
	S2	P2	D2
	S3	P1	D3

R ₁	S	P
	S1	P1
	S2	P2
	S3	P1

R ₂	P	D
	P1	D1
	P2	D2
	P1	D3

- **Dependency preserving property** : enables to enforce any constraint on the original relation simply enforcing some constraints on each of the smaller relation.

SCHEMA REFINEMENT

- Schema refinement can be considered a systematic process for analyzing a relational schema with the aim of minimizing redundancies and minimizing insertion, deletion and update anomalies.
- The process performs a series of tests called **normal form tests** to check whether the schemas meet certain conditions, and those relations that are unsatisfactory are decomposed into smaller relations.
- Normalization is based on functional dependencies

FUNCTIONAL DEPENDENCY IN GENERAL TERMS

- Functional dependency is a relationship that exists when one attribute uniquely determine another attribute.
- For example: Suppose we have a student table with attributes: id name, age.
 - Here id attribute uniquely identifies the name attribute of student table because if we know the student id we can tell the student name associated with it.
 - This is known as functional dependency and can be written as $\text{id} \rightarrow \text{name}$ or in words we can say name is functionally dependent on id.
- Redundancies in relations are based on functional dependencies

FUNCTIONAL DEPENDENCIES

- Mathematical definition of functional dependency (FD) :
 - A functional dependency, denoted by $X \rightarrow Y$, where X and Y are sets of attributes in relation R , specifies the following constraint:
 - Let t_1 and t_2 be tuples of relation R for any given instance
Whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$
where $t_i[X]$ represents the values for X in tuple t_i

ACTIVITY

- Consider a person entity. With respect to attributes you know a person holds write three functional dependencies that can exist in a person table.
- Exchange what you have written with your peers.
- What have they written?

KEYS AND FDS

- A key constraint is a special case of a FD where the attributes in the key play the role of X and the set of all attributes play the role of Y.
- Normalization process analyzes schemas based on keys and on the functional dependencies among their attributes.
- Thus, to start normalization it is essential to find keys of a given relation.
- Attribute closure of an attribute set X, denoted by X^+ can be defined as set of all attributes which can be functionally determined from it.
- If $X^+ = \text{all attributes}$, then X is a key.

COMPUTING KEYS USING AMSTRONG AXIOMS

- Attribute closure could be computed using Armstrong Axioms.
- X, Y, Z are sets of attributes:

Reflexivity: If $X \subseteq Y$, then $Y \rightarrow X$

Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z

Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- Couple of additional rules (that follow from Armstrong Axioms):

Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

COMPUTING KEYS USING AMSTRONG AXIOMS (CONTD.)

- Consider a relation R (A, B, C, D), with the following set of functional dependencies over R:

- $F = \{A \rightarrow B, B \rightarrow C, B \rightarrow D\}$

$A \rightarrow A$ (reflexivity rule)

$A \rightarrow B$ (given)

$A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$ (transitivity)

$A \rightarrow B$ and $B \rightarrow D$ then $A \rightarrow D$ (transitivity)

$A \rightarrow ABCD$ (Union rule) , $[A]^+ = \{ABCD\}$

$B^+ = \{BCD\}$, $C^+ = \{C\}$, $D^+ = \{D\}$

Therefore A is the key

ACTIVITY

- Consider a relation R (A, B, C, D, E), with the following set of functional dependencies over R:
 - $\{A \rightarrow B, A \rightarrow C, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$
 - Compute the keys for relation R.

REVISIT TO SOME DEFINITIONS

- **Superkey:** Set of attributes S in relation R such that no two distinct tuples t_1 and t_2 will have $t_1[S] = t_2[S]$
- **Key:** A key is a superkey with the additional property that removal of any attributes from the key will not satisfy the key condition
- **Candidate Key:** Each key of a relation is called a candidate key
- **Primary Key:** A candidate key is chosen to be the primary key
- **Prime Attribute:** an attribute which is a member of a candidate key
- **Nonprime Attribute:** An attribute which is not prime

NORMAL FORMS

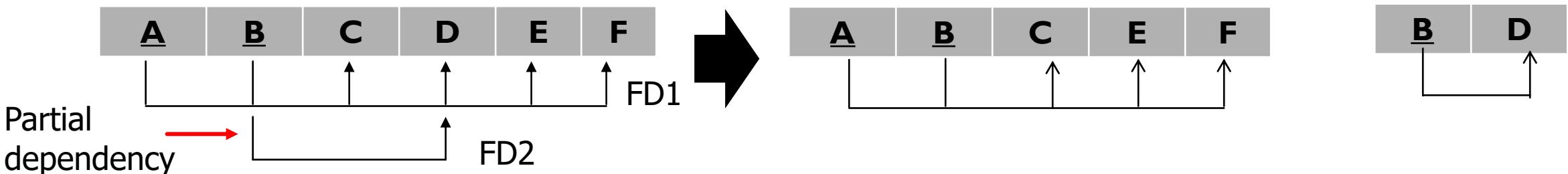
- Normal forms refers to a series of tests performed on relational schemas to improve their goodness.
- We discuss four normal forms namely,
 - 1st Normal Form
 - 2nd Normal Form
 - 3rd Normal Form
 - Boyce Codd Normal Form
- Test for each normal form is performed in a top-down fashion.
- The Normal form of a relation refers to the highest normal form condition it meets.

1ST NORMAL FORM

- A relation R is in first normal form (1NF) if domains of all attributes in the relation are *atomic* (simple & indivisible).
- 1NF is now considered to be part of the formal definition of a relation in the relational model since it allows only atomic values and disallows multivalued attributes and composite attributes.

2ND NORMAL FORM

- A relation R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.
- Second normal form (2NF) is based on the concept of full functional dependency.
 - A functional dependency $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.
 - For example, in a relation R (ABCDE) where $AB \rightarrow \{ABCDE\}$, if $A \rightarrow C$, $A \rightarrow C$ is a partial dependency (not fully functional dependent)
- To normalize the relation to 2NF decomposition is performed as follows.



ACTIVITY

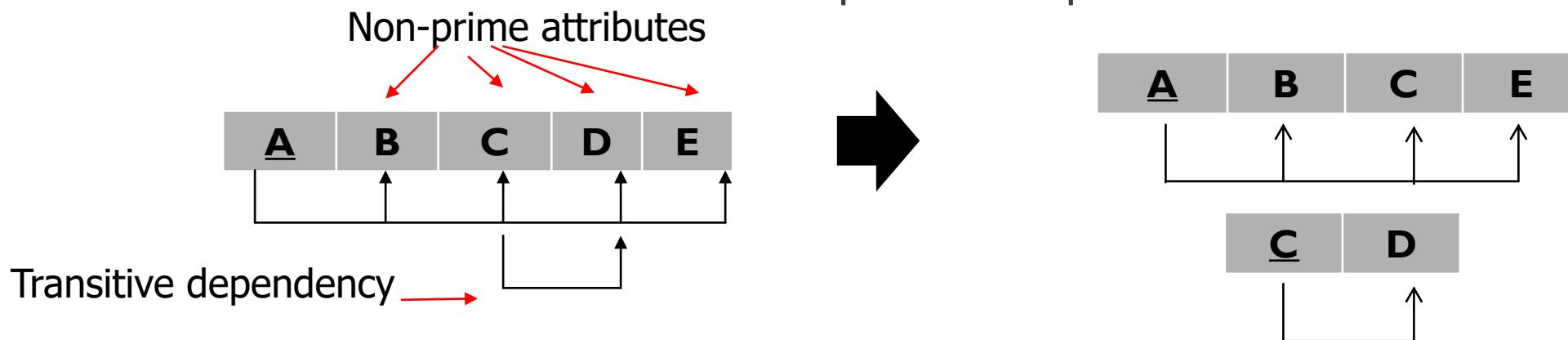
- What normal form the relation in the slide is? If the relation is not in 2NF normalize the relation to 2NF.

EMP_PROJ	<u>NIC</u>	<u>PNUM</u>	HOURS	ENAME	PNAME	LOC
FD1						
FD2						
FD3						

```
graph TD; NIC[NIC] -- FD1 --> PNUM[PNUM]; PNUM -- FD2 --> HOURS[HOURS]; HOURS -- FD3 --> ENAME[ENAME]
```

3RD NORMAL FORM

- A relation R is in 3rd normal form (3NF) if every
 - R is in 2NF, and no nonprime attribute is transitively dependent on any key
- Third normal form is based on the concept of transitive dependency.
 - A functional dependency $X \rightarrow Y$ in a relational schema R is a transitive dependency if there is a set of non-prime attributes Z where both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.
- To normalize the relation to 2NF decomposition is performed as follows.

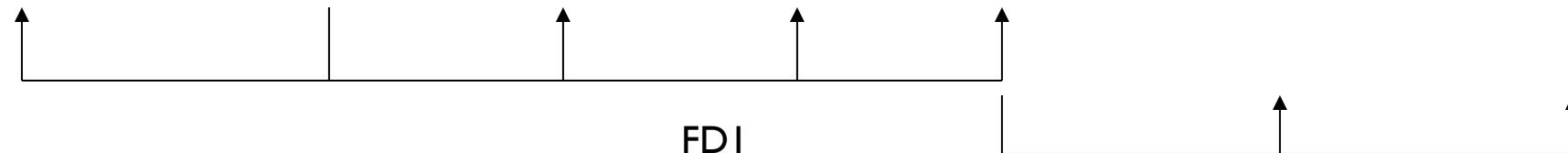


ACTIVITY

- What normal form the relation in the slide is?
- If the relation is not in 3NF normalize the relation to 3NF.

EMP_DEPT

ENAME	<u>SSN</u>	BDATE	ADD	DNUM	DNAME	DMGR
-------	------------	-------	-----	------	-------	------

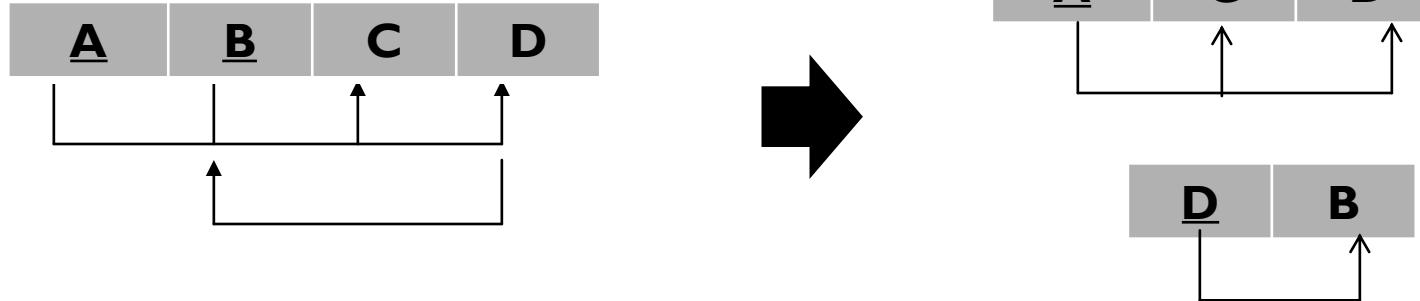


FD1

FD2

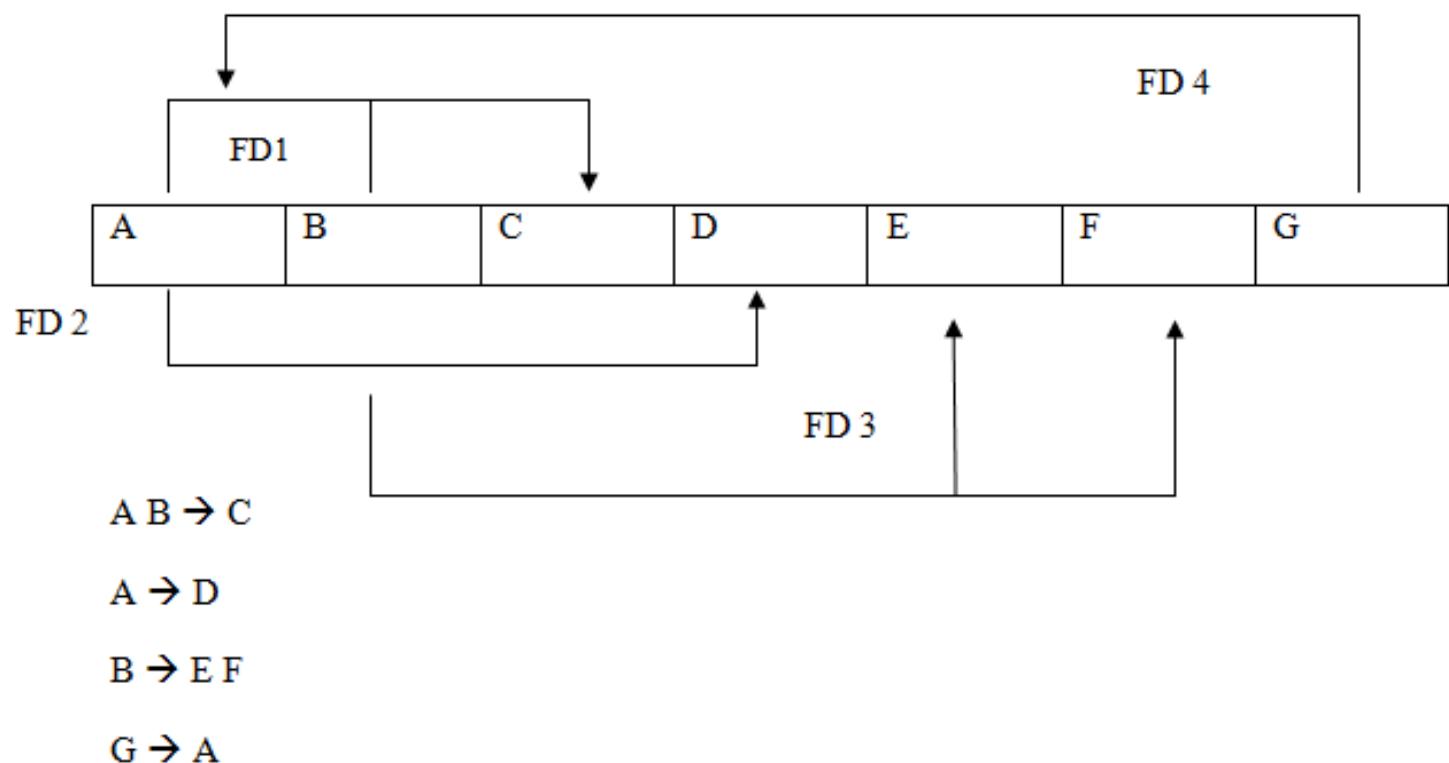
BOYCE-CODD NORMAL FORM

- A relation schema is in Boyce-Codd Normal Form
 - If every nontrivial functional dependency $X \rightarrow A$ hold in R, then X is a superkey of R
- Decomposition into BCNF:
 - Consider relation R with FDs F. If $X \rightarrow Y$ violates BCNF, decompose R

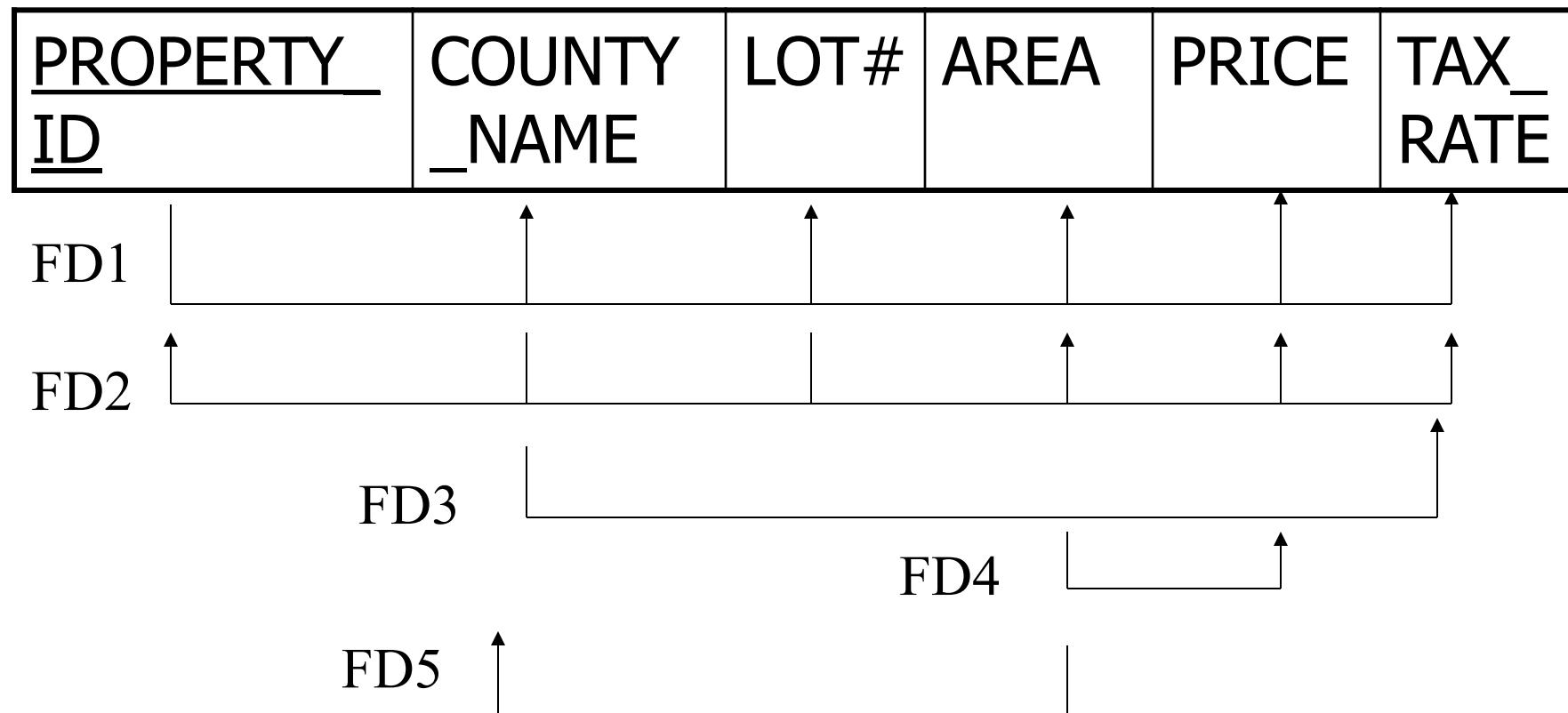


ACTIVITY

- Consider the following relational schema for R:
 $R(A, B, C, D, E, F, G)$
- AB is the primary key in the relation.
- What normal form is the relation in?
- If R is not in BCNF, convert it to BCNF.



ACTIVITY



WHAT YOU HAVE TO DO BY NEXT WEEK

- Try out the self-test questions on the course web.
- Complete the tutorial.
- Read Chapter

DATABASE MANAGEMENT SYSTEMS (IT2040)

LECTURE 04 - SQL



LECTURE CONTENT

- Introduction to SQL
- Data definition language
- Data manipulation language

LECTURE CONTENT

- At the end of this lecture students should be able to
 - Write syntactically correct SQL statements to create and modify relations in a RDBMS
 - Write syntactically correct SQL statements to answer user defined queries in a RDBMS

SQL

- SQL Initially called SEQUEL (for Structured English QUErY Language)
- It was developed for an experimental relational database system called System R
- A joint effort between ANSI (American National Standard Institute) and ISO (International Standards Organization) led to a standard version of SQL in 1986 (SQLI, SQL-86, etc.)
- Major revisions have been proposed and SQL2 (also called SQL-92) has subsequently been developed

RELATIONAL MODEL VS. SQL

■ Terminology

Relational Model	SQL
Relation	Table
Attribute	Column
Tuple	Row

SQL: REVIEW (CONTD.)

SQL is a comprehensive database language:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Facilities for security & authorization
- Facilities for transaction processing
- Facilities for embedding SQL in general purpose languages (Embedded SQL)
- ...

DATA DEFINITION LANGUAGE (DDL)

- DDL is the subset of SQL that supports the creation, deletion and modifications for tables and views.
- Constraints can be defined on the tables

Constraint	Purpose
Not Null	Ensure that column doesn't have null values
Unique	Ensure that column doesn't have duplicate values
Primary key	Defines the primary key
Foreign key	Defines a foreign key
Default	Defines a default value for a column (When no values are given)
Check	Validates data in a column

CREATING A TABLE - EXAMPLE

```
CREATE TABLE STUDENT
(
    studentId INTEGER PRIMARY KEY,
    sName VARCHAR (30) NOT NULL,
    nic CHAR(10) UNIQUE,
    gpa FLOAT,
    proglId VARCHAR(10) DEFAULT 'IT',
    CONSTRAINT student_prog_fk FOREIGN KEY (proglId)
        REFERENCES programs(id) ON DELETE SET DEFAULT ON
        UPDATE CASCADE,
    CONSTRAINT gpa_ck CHECK (gpa<= 4.0 )
)
```

MODIFICATIONS TO TABLES

- ALTER commands – to alter the definition of the object
 - Ex :Adding a new column to a table
 - **ALTER TABLE** student **ADD** age **INT**
 - Ex :Adding a new constraint to a column
 - **ALTER TABLE** student **ADD CONSTRAINT** chk_age **CHECK** (age > 18)
 - Ex :removing a column from a table
 - **ALTER TABLE** student **DROP COLUMN** age
- DROP commands – for dropping objects
 - Ex: Deleting a table
 - **DROP TABLE** Employee

DATA MANIPULATION LANGUAGE (DML)

- DML is the subset of SQL that allows users to write statements to insert, delete, modify and display rows.
 - Inserting a row
 - **INSERT INTO** student**VALUES** (1000,‘Amal’,‘123456789V’,3.2,‘BM’)
 - **INSERT INTO** student(studentId, sName, nic) **VALUES** (1001,‘Nimali’,‘234567890V’)

StudentID	SName	nic	gpa	progId
1000	Amal	123456789V	3.2	BM
1001	Nimali	234567890V	Null	IT

DATA MANIPULATION LANGUAGE (DML)(CONTD.)

- Deleting a row
 - **DELETE** student **WHERE** studentId=1000
- Updating a row
 - **UPDATE** student
SET gpa=2.8
WHERE studentId=1001

SELECT CLAUSE

- Select clause in SQL is the basic statement for retrieving information from a database

- Basic form

SELECT <attributes>

FROM <one or more relations>

WHERE <conditions>

- Ex : display ids of all students whose gpa is above 3.0

- Select StudentId from student where gpa > 3.0

CLAUSES AND OPERATORS USED WITH SELECT

- LIKE operator
- IS [NOT] NULL operator
- DISTINCT operators
- BETWEEN operator
- ORDER BY clause
- Joins (inner & outer)
- Nested query (IN/SOME/ANY,ALL), [NOT] EXISTS
- Aggregate functions
- GROUP BY – HAVING clauses

LIKE OPERATOR

- Used for matching patterns
- Syntax : <string> LIKE <pattern>
 - <pattern> may contain two special symbols:
 - % = any sequence of characters
 - _ = any single character
- Ex : Find students whose name starts with a 'A'
 - Select Name From student where Name Like 'A%'

Student

StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	Null	IT
1002	Aruni	3.0	SE
1003	Surani	2.5	IT

Name
Amal
Aruni

IS [NOT] NULL OPERATOR

- IS NULL :Used to check whether attribute value is null
- Ex :Find studentIDs of the students who have not completed a semester yet.
 - Select studentId

From student

Where gpa IS NULL



StudentID
1001
1004

Student			
StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	Null	IT
1002	Aruni	3.0	SE
1003	Surani	2.5	IT
1004	Imali	Null	BM

DISTINCT OPERATOR

- In a table, a column may contain many duplicate values.
- Duplicates in results can be eliminated using DISTINCT operator
- Ex :
Select proglId
From student

ProglId
BM
IT
SE
IT
BM

Student

StudentID	Name	gpa	proglId
1000	Amal	3.2	BM
1001	Nimali	Null	IT
1002	Aruni	3.0	SE
1003	Surani	2.5	IT
1004	Imali	Null	BM

Select DISTINCT proglId
From student

ProglId
BM
IT
SE

BETWEEN OPERATOR

- Used to check whether attribute value is within a range
- Ex :Find the students who will be obtaining a first class ($3.7 \leq gpa \leq 4.0$)

Select studentID

From student

Where gpa between 3.7 and 4.00

Student

StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	Null	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	IT
1004	Imali	4.0	BM



StudentID
1002
1004

ORDER BY CLAUSE

- Used to order results based on a given field
- Ordering is ascending (ASC), unless you specify the DESC keyword
- Ex : display the student names and gpa's in the ascending order of gpa's.

Select Name, gpa 

From student

Order by gpa

Name	gpa
Surani	2.5
Nimali	2.8
Amal	3.2
Aruni	3.8
Imali	4.0

Student

StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	IT
1004	Imali	4.0	BM

[INNER] JOIN

- Joins two tables based on a certain condition
- Ex : Find the names of students who follow programs offered by SLIIT

Select s.Name

From student s, program P

where s.pid=p.ProgId and offerBy='SLIIT'

Or

Select s.Name

From student s INNER JOIN program p on

s.pid=p.progId

Where offerBy='SLIIT'

Student

SID	Name	gpa	pid
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	IT

Program

progId	years	Offer By
BM	3	Curtin
IT	4	SLIIT
SE	3	SHU

Name
Nimali
Surani

LEFT OUTER JOIN

- Returns all rows from the table on the left hand side of join, with the matching rows in the table on the right hand side of the join.
- The result is NULL in the right side when there is no match.
 - Ex : For all the students display the name and the offering institute

Select s.Name, p.offerBy

From student s LEFT OUTER JOIN program p
on s.pid=p.progId

Student

SID	Name	gpa	pid
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	IT

Program

progId	years	Offer By
BM	3	Curtin
IT	4	SLIIT



Name	offerBy
Amal	Curtin
Nimali	SLIIT
Aruni	NULL
Surani	SLIIT

RIGHT OUTER JOIN

- Returns all rows from the table on the right hand side of join, with the matching rows in the table on the left hand side of the join.
- The result is NULL in the left side when there is no match.
 - Ex : For all the programs display the offering institute and names of the students following

Select s.Name, p.offerBy

From student s RIGHT OUTER JOIN program p
on s.pid=p.progId

Student

SID	Name	gpa	pid
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	NULL
1003	Surani	2.5	IT

Program

progId	years	Offer By
BM	3	Curtin
IT	4	SLIIT
SE	3	SHU



Name	offerBy
Amal	Curtin
Nimali	SLIIT
Surani	SLIIT
NULL	SHU

IN OPERATOR

- Used to check whether attribute value matches any value within a value list
- Ex : Find the students who has obtained a 'A'.

Select s.Name

From Student s

Where s.SID IN (Select SID →

from Grades

Where Grade='A')

Student

SID	Name	gpa
1000	Amal	3.2
1001	Nimali	2.8
1002	Aruni	3.8

Grades

SID	cid	Grade
1000	IT102	A
1000	IT100	B
1001	IT102	A
1002	IT102	C
1002	IT200	C

Name
Amal
Nimali

EXISTS OPERATOR

- Used to check if subquery returns any rows
- Ex : Find the students who has obtained a 'A'.

Select s.Name

From Student s

Where EXISTS (Select *

from grades g

Where g.SID=s.SID and
g.Grade='A')

Student

SID	Name	gpa
1000	Amal	3.2
1001	Nimali	2.8
1002	Aruni	3.8

Grades

SID	cid	Grade
1000	IT102	A
1000	IT100	B
1001	IT102	A
1002	IT102	C
1002	IT200	C

Name
Amal
Nimali

COMPARISON OPERATORS WITH SOME, ANY & ALL

- Comparison operators such as =, <>, >, >= , < and <= could be modified using operators SOME, ANY and ALL.
 - SOME and ANY : used to compare a value to a list or subquery. Return true if at least one of the comparison evaluates as true.
 - ALL : used to compare a value to a list or subquery. If all of the comparisons evaluate to true then the result of the ALL expression will be true. Otherwise the result will be false

ANY AND SOME OPERATORS

- Ex : Find BM students who has gpa greater than any of the IT students.

Select s.Name

From student s

Where s.progID='BM' and

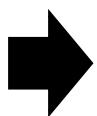
s.gpa > ANY (

select s1.gpa

from student s1 where
s1.progId='IT')

Student

StudentID	Name	Gpa	progId
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	BM
1004	Imali	4.0	IT



Name
Amal

ALL OPERATOR

- Ex : Find IT students who has **Student** gpa greater than all the BM students.

Select s.Name

From student s

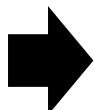
Where s.progID='IT' and

s.gpa > ALL (

select s1.gpa

from student s1 where
s1.progId='BM')

StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	BM
1004	Imali	4.0	IT



Name
Imali

AGGREGATION

- An aggregate function summarizes the results of an expression over a number of rows, returning a single value.
- Some of the commonly used aggregate functions are SUM, COUNT, AVG, MIN and MAX

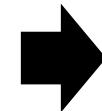
AGGREGATION (CONTD.)

- Ex : Find the average, minimum, maximum gpa of students

Student

StudentID	Name	gpa	progId
1000	Amal	3.2	BM
1001	Nimali	2.8	IT
1002	Aruni	3.8	SE
1003	Surani	2.5	BM
1004	Imali	4.0	IT

Select AVG(gpa), MIN(gpa), MAX(gpa)
From student



AVG(gpa)	MIN(gpa)	MAX(gpa)
3.26	2.5	4.0

GROUPING (GROUP BY CLAUSE)

- Groups the data in tables and produces a single summary row for each group.
- Grouping is done based on values in a given field
- When using group by
 - Each item in the SELECT list must be single valued per group.
 - SELECT clause may only contain Columns names, aggregate function, constants or an expression involving combinations of the above.
 - All column names in SELECT list must appear in the GROUP BY clause unless the name is used only in the aggregate function.

GROUPING (CONTD.)

- Ex: Count the number of students who has followed each module.

Student

SID	CID	Grade
1000	DBII	A
1000	SEI	B
1001	DBII	A
1002	DBII	C
1002	SPD	C

SID	CID	Grade
1000	DBII	A
1001	DBII	A
1002	DBII	C
1000	SEI	B
1002	SPD	C

Select CID, Count(SID)
From Student
Group by CID



CID	Count (SID)
DBII	3
SEI	1
SPD	1

HAVING CLAUSE

- Used to apply conditions on the groupings
- Ex: Find courses which is followed by more than two students

Select CID, Count(SID)

From course

Group by CID

Having count(SID)>2

CID	Count (SID)
DBII	3

Student

SID	CID	Grade
1000	DBII	A
1000	SEI	B
1001	DBII	A
1002	DBII	C
1002	SPD	C

SID	CID	Grade
1000	DBII	A
1001	DBII	A
1002	DBII	C
1000	SEI	B
1002	SPD	C

SQL: REVIEW (CONTD.)

Summary of SQL Queries...

```
SELECT <attribute-list>
FROM  <table-list>
[WHERE  <condition>]
[GROUP BY  <group attribute(s)>]
[HAVING  <group condition>]
[ORDER  BY <attribute list>];
```

WHAT YOU HAVE TO DO BY NEXT WEEK

- Try out the self-test questions on the course web.
- Complete the tutorial.

DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 05 – DATABASE PROGRAMMING

LECTURE CONTENT

- Views
- SQL extensions (specifically T-SQL)
- Functions & Procedures
- Triggers

LEARNING OUTCOMES

- At the end of this lecture, students should be able to
 - Identify situations where views, functions, stored procedures and triggers are applicable.
 - Write syntactically correct sql statements to create functions, procedures and triggers on RDBMS to cater user requirements.

VIEWS

- A **view** is a virtual table which is derived from other tables which are called **base tables** or **defining tables**.
- A view does not necessarily exist in a physical form.
- In SQL, CREATE VIEW statement is used to define views.
- Syntax :

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

VIEWS (CONTD.)

- emp (eid, ename, age, salary) ,dept (did, dname, budget,mangerId), works (eid, did, pct_time)
- Create a view named dept_info that contains name of the department, budget and manager's name

```
CREATEVIEW dept_info(dname,budget,manager)
AS
SELECT d.dname, d.budget, e.ename FROM emp e,dept d
WHERE e.eid=d.managerId
```

EXERCISE I

- Create a view named emp_info which contains eid, name, salary and total percentage of time.

emp (eid, ename, age, salary)

dept (did, dname, budget, mangerId)

works (eid, did, pct_time)

QUERYING AND DELETING A VIEW

- Querying a view can be done similar to a table.
 - Ex : `SELECT * FROM dept_info`
 - Ex : `SELECT dname FROM dep_info WHERE budget > 500,000`
- Dropping a view
 - Ex : `DROPVIEW dept_info`

UPDATING VIEWS (CONTD.)

- Updating a view can be ambiguous...
 - Views containing aggregate functions are not updateable

- Ex :

```
UPDATE emp_info  
SET tot_pct= 90  
WHERE eid = 1000
```

**This is not possible

UPDATING VIEWS (CONTD.)

- Views containing a join can be ambiguous

A	B
b	1

B	C
1	d
1	e

A	B	C
b	1	d
b	1	e

UPDATE VI

SET A = a

WHERE C = e

- Thus, in many DBMSs, views are updateable only if they are defined on a single base table.

WHY VIEWS ?

- Advantages
 - **Security** : Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data
 - **Query Simplicity** : A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- Disadvantages
 - **Performance** : Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time
 - **Update restrictions**

PROGRAMMING IN T-SQL

- Similar to a programming language, certain extensions have been made in SQL to program simple server-side logic.
- Some of the statements include:
 - Variables
 - Selection conditions
 - IF (...)... ELSE ...
 - Looping
 - WHILE (...)

T-SQL:VARIABLES

- A Transact-SQL local variable is an object that can hold a single data value of a specific type.
- Variables in scripts are typically used:
 - As a counter either to count the number of times a loop is performed or to control how many times the loop is performed
 - To hold a data value to be tested by a control-of-flow statement
 - To save a data value to be returned by a stored procedure return code.

T-SQL:VARIABLES (CONTD.)

- The DECLARE statement initializes a Transact-SQL variable.
 - Syntax: `DECLARE @<variable name> <data type>`
 - Ex: `DECLARE @DName VARCHAR(20)`
 - The created variable will be holding a null value
- To assign a value to a variable, use the SET statement.
 - Syntax : `SET @<variable name> =<value>`
 - `SET @DName = 'SESD'`

T-SQL:VARIABLES (CONTD.)

- The declared variables could be used in scripts
- Ex :
 - ```
SELECT budget
 FROM Dept
 WHERE dname = @DName
```
  - ```
DECLARE @empId INT
          SELECT @empId = MAX(eid)
          FROM emp
```

T-SQL: IF STATEMENT

- Imposes conditions on the execution of a Transact-SQL statement.
- Ex:

```
IF (SELECT count(eid) FROM emp) > 1000
BEGIN
    PRINT 'Inside the IF statement'
    PRINT 'There are lesser than 1000 employees '
END
ELSE
    PRINT 'There are more than 1000 employees !'
```

T-SQL: WHILE STATEMENT

- Sets a condition for the repeated execution of an SQL statement or statement block.
- The statements are executed repeatedly as long as the specified condition is true.
- The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords.

T-SQL: WHILE STATEMENT (CONTD.)

- **BREAK**
 - Causes an exit from the innermost WHILE loop. Any statements appearing after the END keyword, marking the end of the loop, are executed.
- **CONTINUE**
 - Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

T-SQL: WHILE STATEMENT(CONTD.)

- Ex :

```
WHILE @count<=100
BEGIN
    INSERT INTO Employees VALUES(@count,CONCAT('Employee',@count))
    SET @count=@count+1
END
```

STORED FUNCTIONS/PROCEDURES

- Business logic is maintained in database tier for data intensive operations
 - E.g. Calculating all interest earned in bank accounts
- In SQL Server 2005, Stored Procedure/ Functions can be written in
 - T-SQL (we will study only this)
 - Any .NET Language

STORED FUNCTIONS/PROCEDURES (CONTD.)

- Syntax of a function

```
CREATE FUNCTION <function name> (parameters)
RETURNS <return type>
<function body>
```

- Parameter mode of parameters for functions is IN which parameters allow the calling code to pass values into the procedure

- Syntax of a procedure

```
CREATE PROCEDURE <procedure name> (parameters)
<procedure body>
```

- Each parameter to a procedure should have a data type and a parameter mode (IN or OUT).

- IN: This is the default mode. IN parameters allow the calling code to pass values into the procedure
- OUT: OUT parameters allow the procedure to pass values back to the calling code

FUNCTIONS

- Ex : Create a function that returns the number of employees in a given department.

Function name → create function **getEmpCount** (@did char(12)) returns int ← Return data type
Input parameters

as

begin

```
declare @ecount int  
select @ecount=count(*)  
from works w  
where w.did=@did  
return @ecount
```

end

FUNCTIONS (CONTD.)

- Calling the function created previously

```
declare @result int  
exec @result=get_empCount 'Admin'  
print @result
```

EXERCISE 2

- Create a function to return the total percentage of time a person works given the employee id.

STORED PROCEDURES

- Ex : Create a procedure to give a salary increment to all the employee by a given percentage from their existing salary

```
CREATE PROCEDURE increaseSalary (@pct float)
```

as

begin

 Update emp

 Set salary=salary+salary * (pct/100)

end;

STORED PROCEDURES (CONTD.)

- Calling the procedure

```
exec increaseSalary 10
```

STORED PROCEDURES (CONTD.)

- Ex 2: create a procedure that outputs statistics of salary (min, max) for a given department.

```
create procedure get_stats(@did varchar(12),@maxm real output,@minm real output)
as
begin
    select @maxm=max(e.salary),@minm=min(e.salary)      from dept d, works w, emp e
    where d.did=w.did and w.eid=e.eid and d.did=@did
end
```

STORED PROCEDURES (CONTD.)

- Calling the procedure

```
declare @max int,@min int  
exec get_stats 'Admin', @max output,@min output  
print @max  
print @min
```

EXERCISE 3

- Create a procedure that outputs the name of the manager and his salary in a given department.

TRIGGERS

- Triggers are useful in enforcing business rules and data integrity.
- They are more powerful than general constraints.
- For example,
 - The employees salary is always less than his/her manager's salary

T-SQL:TRIGGERS

- A trigger is a special type of stored procedure that automatically takes effect when the data in a specified table is modified.
- A trigger is invoked in response to a
 - DDL statement (CREATE,ALTER etc.) or
 - DML statement (INSERT, UPDATE, or DELETE statement).‘

T-SQL:TRIGGER SYNTAX

- We will learn DML triggers...
- Syntax :

```
CREATE TRIGGER trigger_name
ON { table | view }
{
    { { FOR | AFTER | INSTEAD OF }
    { [ INSERT ] [,] [ UPDATE ] [,]
        [DELETE] }

    AS
        sql_statement [ ...n ]
    }
}
```

T-SQL:TRIGGER SYNTAX (CONTD.)

- **FOR|AFTER**
 - AFTER specifies that the DML trigger is fired only when all operations specified in the triggering SQL statement have executed successfully.
 - AFTER is the default when FOR is the only keyword specified.
 - AFTER triggers cannot be defined on views.
- **INSTEAD OF**
 - Specifies that the trigger is executed *instead of* the triggering SQL statement, thus overriding the actions of the triggering statements.
 - Specifies At most, one INSTEAD OF trigger per INSERT, UPDATE, or DELETE statement can be defined on a table or view.

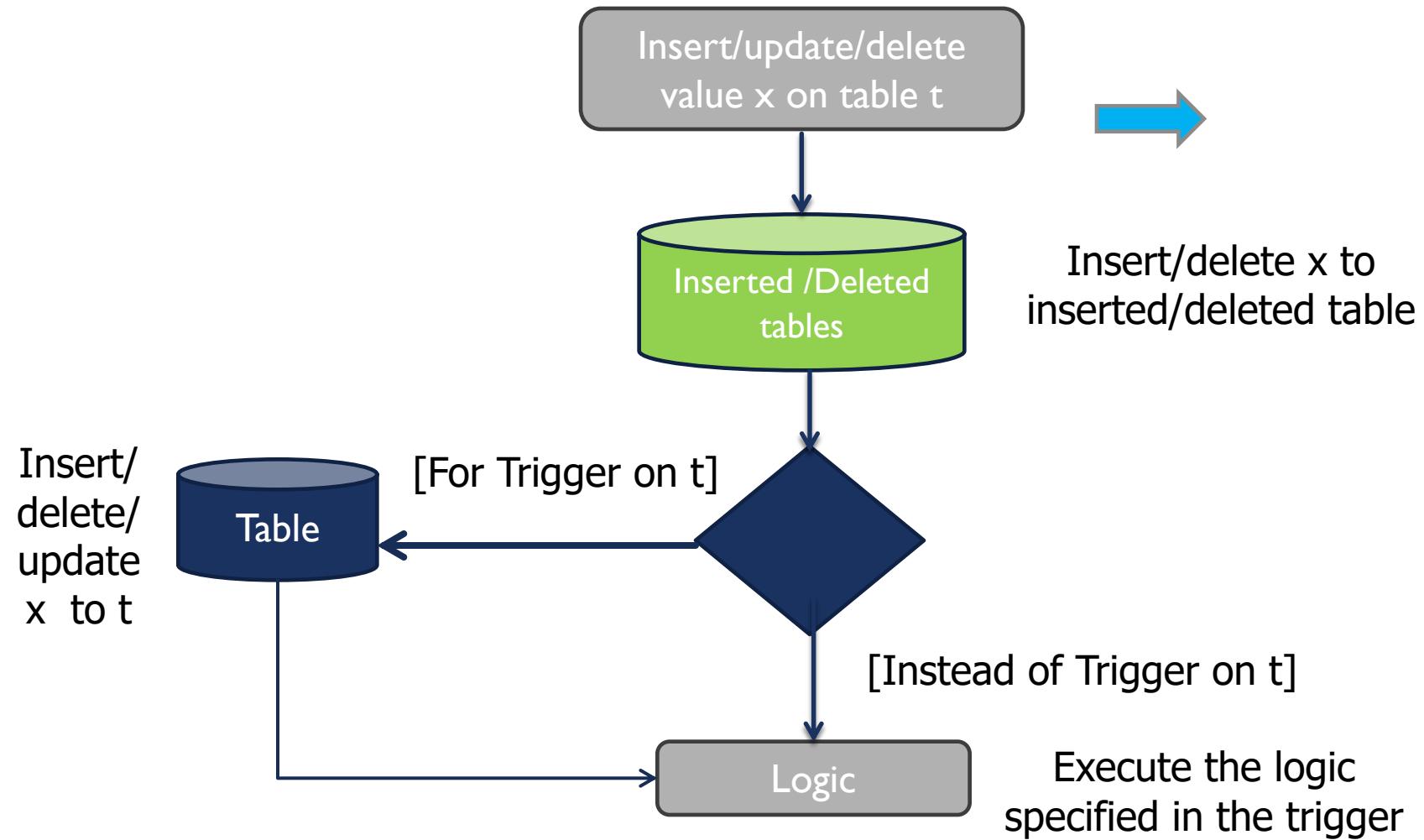
T-SQL:TRIGGER SYNTAX (CONTD.)

- [DELETE] [,] [INSERT] [,] [UPDATE]
 - Are keywords that specify which data modification statements, when attempted against this table or view, activate the trigger. At least one option must be specified.

TRIGGERS : INSERTED/DELETED TABLES

- When a trigger is executed SQL Server creates two virtual tables called **INSERTED** and **DELETED**.
- The **deleted** table stores copies of the affected rows during **DELETE** and **UPDATE** statements
- The **inserted** table stores copies of the affected rows during **INSERT** and **UPDATE** statements
 - For example when inserting a record to a table, SQL Server creates a virtual table call **INSERTED** and loads data into the inserted table then executes the trigger statements and writes the related data pages.
- The format of the inserted and deleted tables is the same as the format of the table on which the trigger is defined
- Each column in the inserted and deleted tables maps directly to a column in the base table

TRIGGERS : HOW DO THEY WORK?



T-SQL:TRIGGERS(CONTD.)

- Example I:
 - Consider tables below
 - Account (accountNo, custId, branch, balance)
 - AccountAudit (accountNo, balance, date)
 - Create a trigger to track all inserts/updates done to the balance field of an Account table at a bank in the AccountAudit table

T-SQL:TRIGGERS (CONTD.)

Create trigger account_audit_trigg

On Account

For Insert, update

As

Begin

 Declare @ano int

 Declare @balance float

 Select @ano=accountNo,@balance=balance from inserted

 Insert into accountAudit(@ano,@balance,getdate())

end

T-SQL:TRIGGERS (CONTD.)

- Example 2:
 - Consider following tables :
 - Emp(eid ,ename, age, salary)
 - Works (eid, did, pct-time)
 - Dept(did, budget, managerid)
 - Create a trigger to ensure that an employee doesn't work in more than 2 departments

EXERCISE 4

- Consider the following table
 - Transaction(tid, accountNo, type,amount,date)
 - Type may contain ‘credit’ or ‘debit’
- Assuming that the bank’s maximum withdrawal limit per day is 40000, write a trigger to ensure that no customer withdraws more than the given limit.

EXERCISE 5

- Consider the tables given below
 - Employee(nic, name, salary, dno)
 - Dept (dno, dname, mgrNic)
- Create a trigger to ensure that no employee has a salary greater than his/her manager.

SUMMARY

- Views
- Transaction Basics
- T-SQL extensions
- Stored Procedures
- Triggers

DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 06 – JAVA DATABASE CONNECTIVITY

LECTURE CONTENT

- JDBC

LEARNING OUTCOMES

- At the end of this lecture students would be able to
 - Write a command line program in java to access different databases

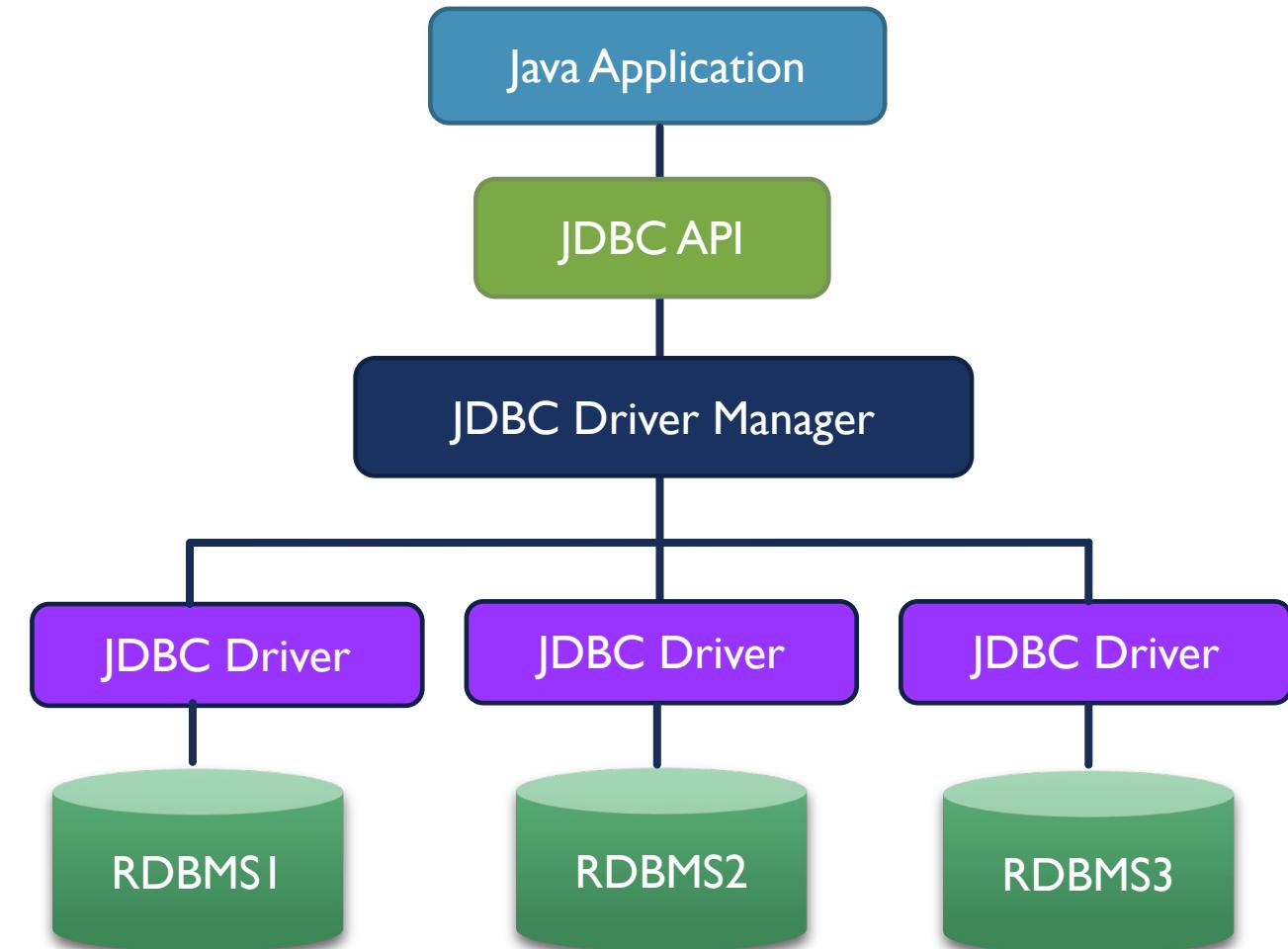
INTRODUCTION

- SQL is the standard common language used for storing in and obtaining information from relational databases.
- Even though SQL was standardized, still different applications were needed to be developed to access different DBMSs.
- A common way for an application to access a relational database was highly desirable.

JDBC

- Java Database Connectivity (JDBC) is an application program interface (API) specification for connecting programs written in Java to the data in popular databases.
- The application program interface lets you to connect to a database, to interact with that database via SQL and retrieve results from the database that could be used within a java application.

JDBC ARCHITECTURE



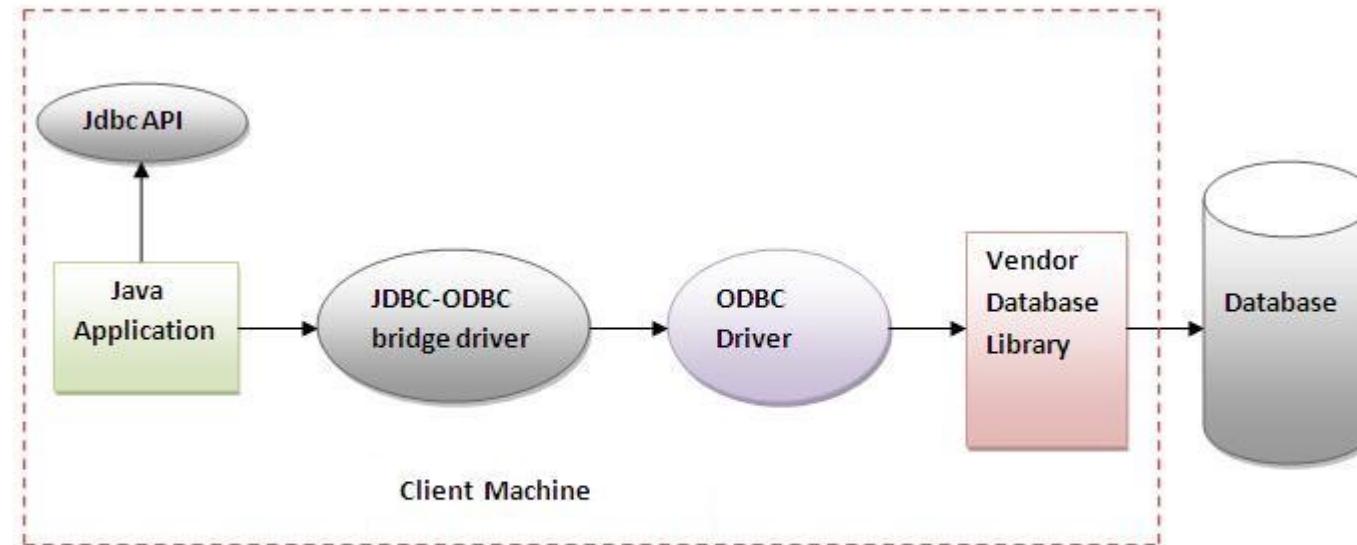
- The JDBC API provides the application-to-JDBC Manager connection.
- The JDBC driver manager supports the JDBC Manager-to-Driver Connection. It manages a list of database drivers. Matches connection requests from the java application with the proper database driver.
- JDBC Driver handles the communications with the database server.

TYPES OF JDBC DRIVERS

- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation in to four categories which are :
 - Type 1: JDBC-ODBC Bridge Driver
 - Type 2: Native API Driver
 - Type 3: Network Pure Java Driver
 - Type 4:Thin Driver(100% Pure Java Driver)

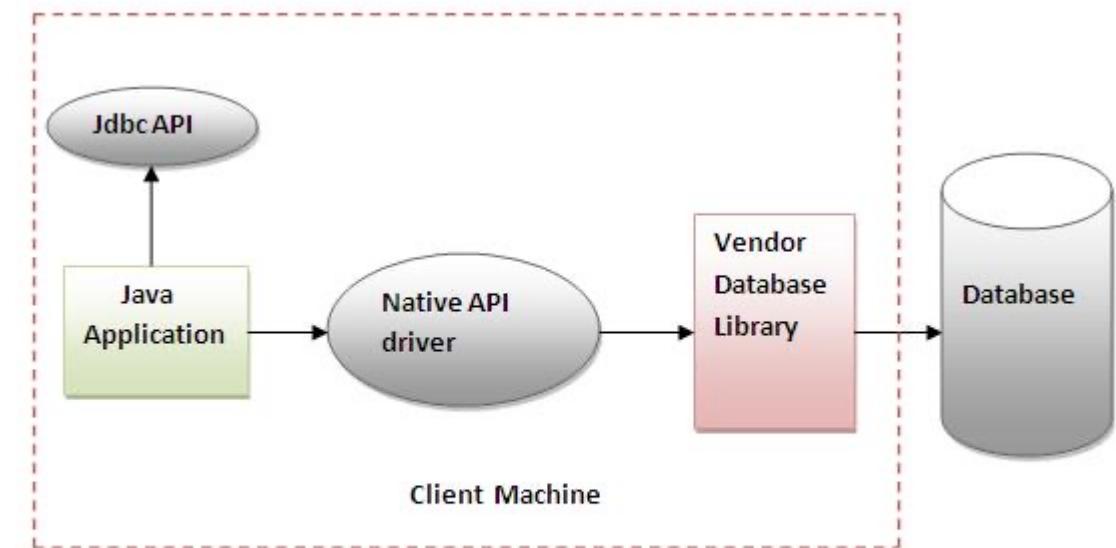
JDBC-ODBC BRIDGE DRIVER

- The JDBC-ODBC bridge driver uses ODBC driver installed in a client machine to connect to the database.
- The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.



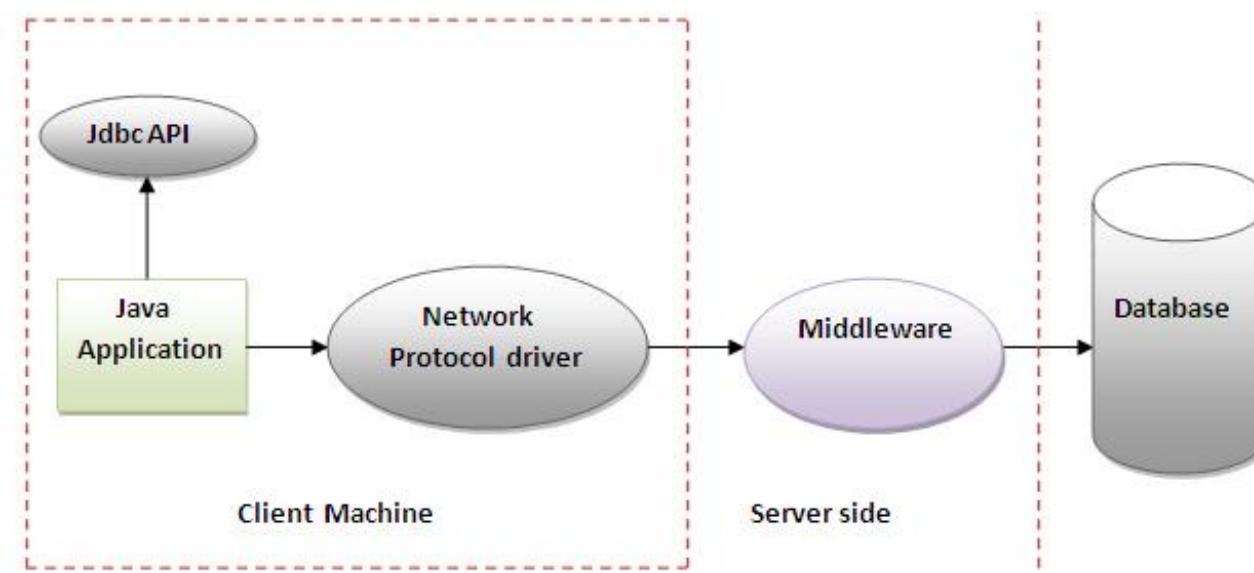
NATIVE API DRIVER

- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database a specific database, such as IBM, Informix, Oracle or Sybase.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
- The vendor-specific driver must be installed on each client machine.



NETWORK PURE JAVA DRIVER

- In a Type 3 driver, a three-tier approach is used to access databases.
- The JDBC clients translate JDBC calls in to middleware vendor's protocol which in return is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.



THIN DRIVER(100% PURE JAVA DRIVER)

- Written entirely in Java
- Converts JDBC calls into packets that are sent over the network in the proprietary format used by the specific database.
- This is the highest performance driver available for the database and is usually provided by the vendor itself.

CREATING A JDBC APPLICATION

- **Import the packages:** Import the packages containing the JDBC classes needed for database programming. Most often, using `import java.sql.*` will suffice.
- **Load the driverRegister the JDBC driver:** Initialize a driver in order to open a communication channel with the database.
- **Open a connection:** Create a Connection object, which represents a physical connection with the database.
- **Execute a query:** Using an object of type Statement for building and submitting an SQL statement to the database.
- **Extract data from result set:** retrieve the data from the result set.
- **Clean up the environment:** Closing all database resources versus relying on the JVM's garbage collection.

STEP I :IMPORT THE PACKAGES

- In this step the packages containing the JDBC classes needed for database programming are imported.
- The `java.sql` package contains the entire JDBC API that sends SQL statements to **relational databases** and **retrieves the results** of executing those SQL statements.
- Some commonly used interfaces in the package are as follows.
 - The `Driver` interface represents a specific JDBC implementation for a particular database system.
 - `Connection` represents a connection to a database.
 - The `Statement`, `PreparedStatement`, and `CallableStatement` interfaces support the execution of various kinds of SQL statements.
 - `ResultSet` is a set of results returned by the database in response to a SQL query.

STEP 2: REGISTERING DATABASE DRIVERS

- In order to use a driver, it should be registered in the program.
- The most common approach to register a driver is to use Java's **Class.forName()** method, to dynamically load the driver's class file into memory, which automatically registers it.
- Ex : `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();`

STEP 3 : OPEN A CONNECTION

- After the driver is loaded to the memory a connection to the database could be established using the **DriverManager.getConnection()** method.
- There are three parameters to the method which are URL of the database, user name and the password.
 - Ex:
 - String url = “jdbc:odbc:dbConn”
 - Connection c = DriverManager.getConnection(url,“user”,“password”);
- This connection should be closed later using **Connection.close()**

STEP 4 : EXECUTING QUERIES

- Once a connection is obtained we can interact with the database. The *JDBC Statement*, *CallableStatement*, and *PreparedStatement* interfaces define the methods and properties that enable you to send SQL commands and receive data from your database.
- A summary of statements is available below :

Interface	Recommended use
Statement	Useful when static SQL statements are used at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Suitable when SQL statements are used many times. The PreparedStatement interface accepts input parameters at runtime
CallableStatement	Used to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

STEP 4 : EXECUTING QUERIES – STATEMENT OBJECT

- Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.
 - boolean execute (String SQL): Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false.
 - int executeUpdate (String SQL): Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected
 - Ex: int rowsUpd = st.executeUpdate("INSERT INTO Customers (cid, cname) VALUES '123A', 'Sampath');
 - ResultSet executeQuery (String SQL): Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.
 - Ex: ResultSet s = st.executeQuery("SELECT cname FROM Customers");

STEP 4 : EXECUTING QUERIES – PREPARED STATEMENTS

- This statement gives you the flexibility of supplying arguments dynamically
- All parameters in JDBC are represented by the ? symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.
- The **setXXX()** methods bind values to the parameters, where **XXX** represents the Java data type of the value you wish to bind to the input parameter.
- All of the Statement object's methods for interacting with the database (a) `execute()`, (b) `executeQuery()`, and (c) `executeUpdate()` also work with the PreparedStatement object.

- Ex:

```
String sql = "UPDATE Employees set age=? WHERE id=?";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
stmt.setInt(1, 35); // This would set age  
stmt.setInt(2, 102); // This would set ID  
int rows = stmt.executeUpdate();
```

STEP 4 : EXECUTING QUERIES – CALLABLE STATEMENTS

- Following steps are used in calling a procedure within a java program
- Firstly, we must prepare a CallableStatement object
 - `CallableStatement cStmt = conn.prepareCall("exec getEmpName ?,?");`
- Bind parameters
 - `stmt.setInt(1, 102);`
- Register output parameters if any
 - `stmt.registerOutParameter(2, java.sql.Types.VARCHAR);`
- Execute the stored procedure
 - `stmt.execute();`
- Get any output parameters if any
 - `String empName = stmt.getString(2);`

STEP 5 : EXTRACT DATA FROM THE RESULT SET

- The SQL statements that read data from a database query, return the data in a result set.
- The `java.sql.ResultSet` interface represents the result set of a database query.
- A `ResultSet` object maintains a cursor that points to the current row in the result set.
- The `next()` method moves the cursor to the next row, and because it returns false when there are no more rows in the `ResultSet` object, it can be used in a while loop to iterate through the result set.
- There is a get method for each of the possible data types There is a get method for each of the possible data types. Get method could be called by providing the column index or column name
- Ex:
 - ```
rs = stmt.executeQuery("SELECT cname, salary FROM Customer");
while (rs.next())
{
 String name = rs.getString(1);
 float salary = rs.getFloat(2);
}
```

# SQL EXCEPTIONS

- JDBC Exception handling is very similar to the Java Exception handling but for JDBC, the most common exception you'll deal with is `java.sql.SQLException`.
- When an exception occurs, an object of type `SQLException` will be passed to the catch clause.
- The passed `SQLException` object has the following methods available for retrieving additional information about the exception:
  - A string describing the error: `getMessage()`
  - An integer error code that is specific to each vendor: `getErrorCode()`

# A SAMPLE PROGRAM

```
String url="jdbc:odbc:dsn";
String query="select ProductID,ProductName,UnitPrice from Products";
try{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
 Connection con=DriverManager.getConnection(url);
 Statement s=con.createStatement();
 ResultSet res=s.executeQuery(query);
 System.out.println("Product ID \t Product Name \t\t Unit Price");
 while(res.next())
 {
 int pID=res.getInt(1);
 String pName=res.getString(2);
 float price=res.getFloat(3);
 System.out.println(pID+"\t"+pName+"\t"+price);
 }
 con.close();
}
```

---

---

---

# DATABASE MANAGEMENT SYSTEMS (IT 2040)

## LECTURE 07 – DATABASE UTILITIES



# LECTURE CONTENT

- Transferring data between different sources
- Backup and restore of data
- Jobs and job schedules
- Database maintenance plans

# LEARNING OUTCOMES

- At the end of this week, students should be able to
  - Transfer data between a data source and a database
  - Create jobs in SQL server
  - Develop a simple database maintenance plan

# DATABASE UTILITIES

- Once the database is developed, you may need to perform several tasks on the created tables and stored data.
- SQL server provides many types of utilities for performing commonly used tasks.
- These include,
  - Data Transferring
  - Creating and managing jobs
  - Backup tools & etc.

# DATA TRANSFERRING

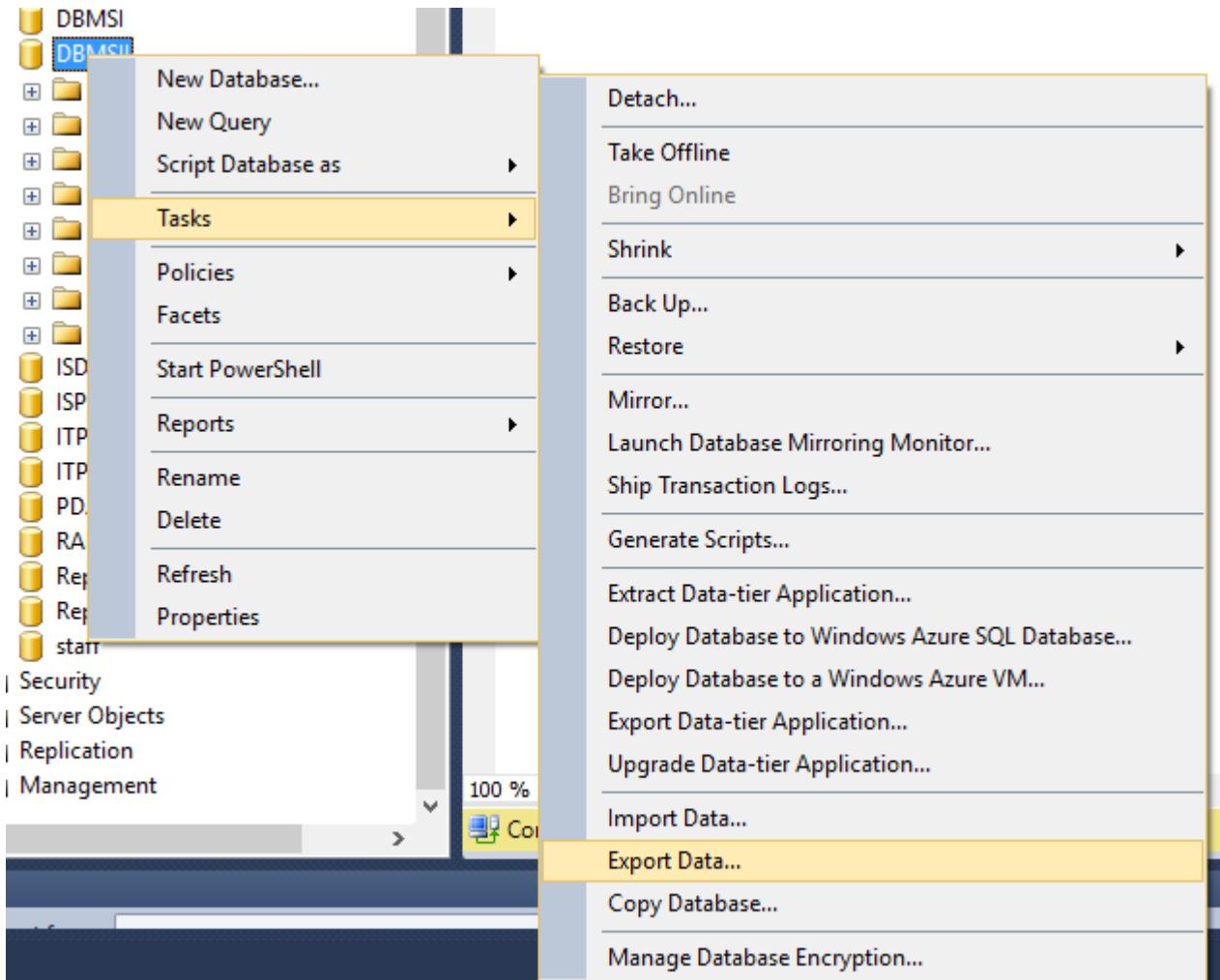
- Some times it is required for transferring data between two sources such as between tables and servers.
- SQL server offers a number of options to facilitate the above as follows :
  - SQL Server Integration Services (SSIS)
  - Using the SQL Server Import and Export Wizard
  - Using BCP to Import and Export Data
  - BULK INSERT
  - SELECT INTO command

# SQL SERVER IMPORT AND EXPORT WIZARD

- The SQL sever import and export wizard provides a quick way to move data and perform very light transformations of data.
- The wizard is available in all editions of SQL Server except the Local Database edition and Express.
- Next few slides of the lecture shows how data in an Excel file could be exported to a SQL server table in 5 steps.

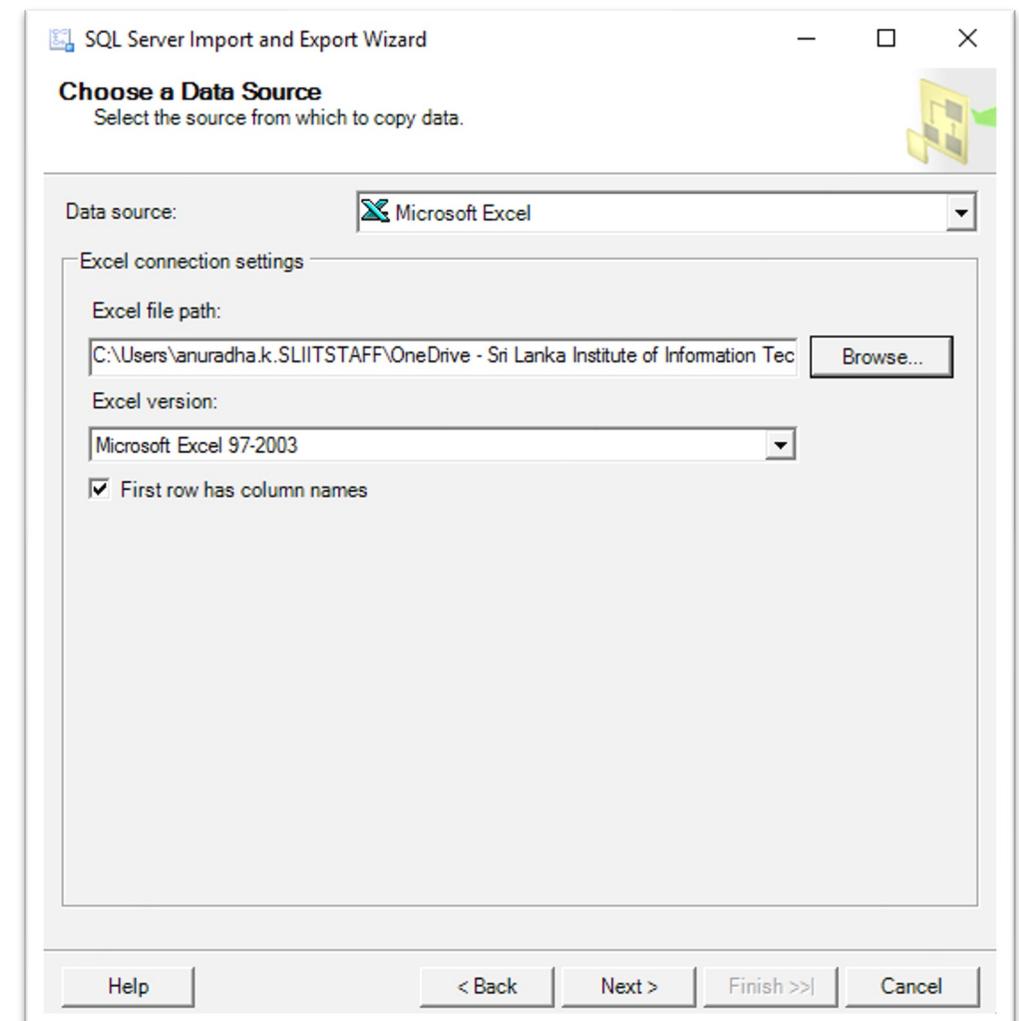
# STEP I – EXPORTING DATA FROM AN EXCEL FILE

- Log in to the SQL sever
- Right click on a database name and select export data sub menu from the task menu.
- Click Next in the welcome screen which appears.



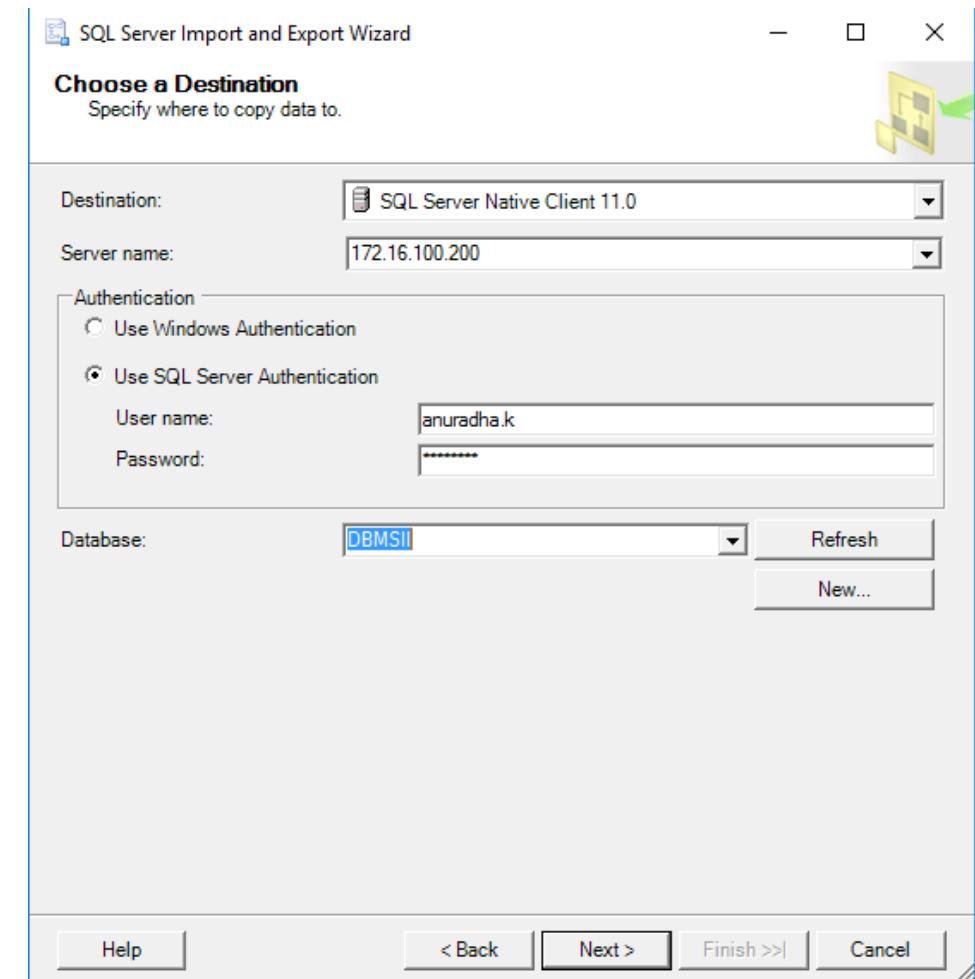
## STEP 2 – EXPORTING DATA FROM AN EXCEL FILE

- On the next screen select the data source.
- In this scenario the data source is an excel file.
- Browse and select the location of the file.
- Tick the check box on the bottom of the screen if the column names are in the first row of the file.
- Click Next.



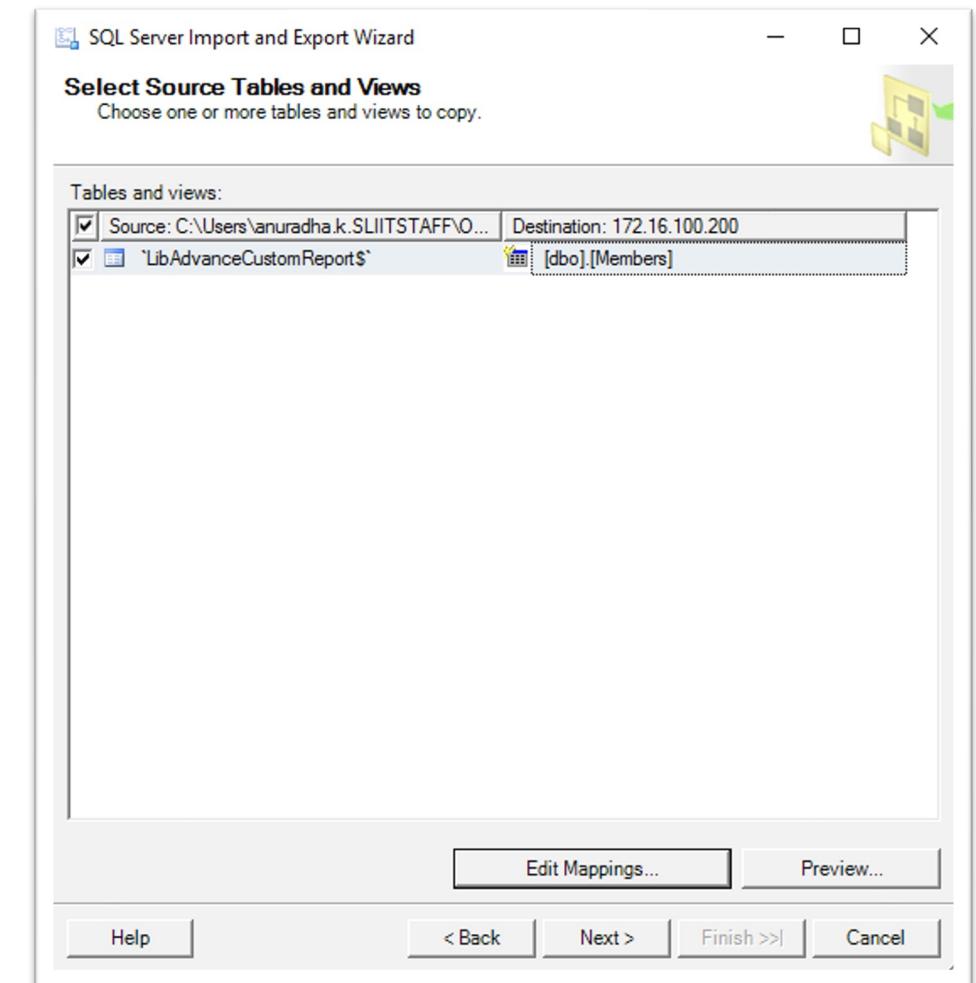
# STEP 3 – EXPORTING DATA FROM AN EXCEL FILE

- Select the destination for the exported data.
- In this case the destination is the SQL sever database.
- Provide the server name, authentication information and the name of the target database in this screen.
- On the next, select whether all data from a file should be copied or specific data should be drawn through querying.
- In this case all data from the file should be copied.



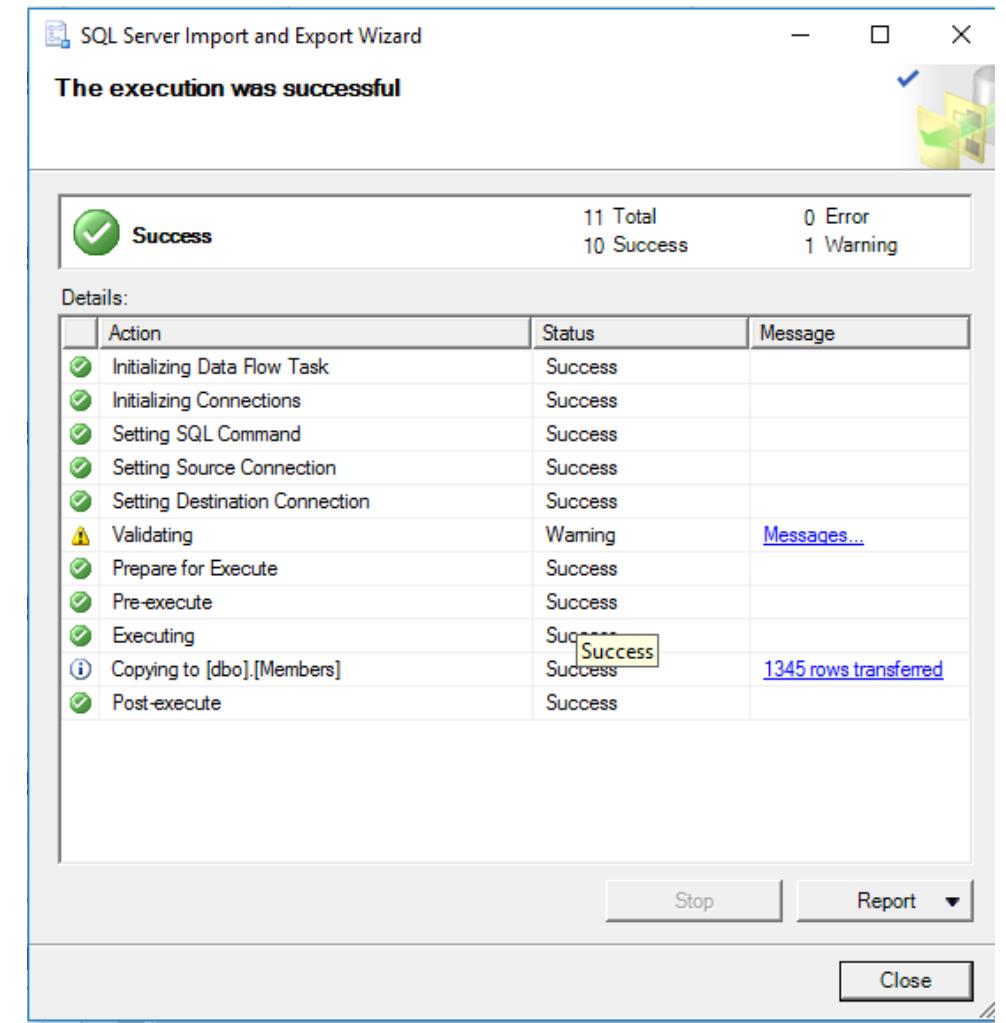
# STEP 4 – EXPORTING DATA FROM AN EXCEL FILE

- In this screen mapping between the source and destination could be done.
- Change the destination name (i.e. target table name) if it is required.
- Select edit mapping to provide more information on the mapping between the source and the destination
  - Ex: data types of the source table
- Click Next



# STEP 5 – EXPORTING DATA FROM AN EXCEL FILE

- Click finish to complete the migration on the next screen.
- Outcome of the data migration process would be displayed as shown on the screen.



# BCP COMMAND

- BCP is a powerful command line utility that enables us to transfer large number of records between a SQL instance and a data file using a special file format.
- This tool is installed by default with SQL Server.
- With BCP the data migrated can be a table or a SQL query result.
- Type BCP command on the command line to obtain the options that could be used with the command.

## BCP COMMAND (CONTD.)

- Format of the BCP command is as follows
  - `bcp {table|view|"query"} {out|queryout|in|format} {data_file|nul} {[optional_argument]}...`
  - Table|view|query represents the source of the data
  - out|queryout|in|format determines the command's mode (direction)
    - Ex: **out option** exports data from a table or view into a data file.
    - Ex: queriyout option exports data retrieved through a query into a data file.
  - *data\_file|nul* is the full path of the data file or, when a data file should not be specified, the **nul** value.

# BCP COMMAND – EXAMPLES

## ■ Copying data from a table to a file

- `bcp adventureworks2012.person.person out C:person.csv -T -c -S L-AC-08-0282`
- `-c` argument is used to perform operations using a character type.
- `-T` is used to connect using a Trusted connection (Windows Authentication).
- `-S` is used to specify the SQL Server name.

## ■ Exporting data from a SQL Server Query to a file.

- `bcp "select Name,color from AdventureWorks2012.Production.Product" queryout C:product.csv -T -c -S L-AC-08-0282`

```
Command Prompt
C:\Users\anuradha.k.SLIITSTAFF>bcp adventureworks2012.person.person out C:person.csv -T -c -S L-AC-08-0282
Starting copy...
SQLState = S1000, NativeError = 0
Error = [Microsoft][ODBC Driver 11 for SQL Server]Warning: BCP import with a format file will convert empty strings in delimited columns to NULL.
1000 rows successfully bulk-copied to host-file. Total received: 1000
1000 rows successfully bulk-copied to host-file. Total received: 2000
1000 rows successfully bulk-copied to host-file. Total received: 3000
1000 rows successfully bulk-copied to host-file. Total received: 4000
1000 rows successfully bulk-copied to host-file. Total received: 5000
1000 rows successfully bulk-copied to host-file. Total received: 6000
1000 rows successfully bulk-copied to host-file. Total received: 7000
1000 rows successfully bulk-copied to host-file. Total received: 8000
1000 rows successfully bulk-copied to host-file. Total received: 9000
1000 rows successfully bulk-copied to host-file. Total received: 10000
1000 rows successfully bulk-copied to host-file. Total received: 11000
1000 rows successfully bulk-copied to host-file. Total received: 12000
1000 rows successfully bulk-copied to host-file. Total received: 13000
1000 rows successfully bulk-copied to host-file. Total received: 14000
1000 rows successfully bulk-copied to host-file. Total received: 15000
1000 rows successfully bulk-copied to host-file. Total received: 16000
1000 rows successfully bulk-copied to host-file. Total received: 17000
```

```
Select Command Prompt
C:\Users\anuradha.k.SLIITSTAFF>bcp "select Name,color from AdventureWorks2012.Production.Product" queryout C:product.csv -T -c -S L-AC-08-0282
Starting copy...
SQLState = S1000, NativeError = 0
Error = [Microsoft][ODBC Driver 11 for SQL Server]Warning: BCP import with a format file will convert empty strings in delimited columns to NULL.
504 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total : 16 Average : (31500.00 rows per sec.)
C:\Users\anuradha.k.SLIITSTAFF>
```

# BULK INSERT COMMAND

- BULK INSERT loads data from a data file into a table.
- This functionality is similar to that provided by the **in** option of the **bcp** command; however, the data file is read by the SQL Server process.
- Example :

```
BULK INSERT emp2 from "D:\emps.csv"
WITH (
FIELDTERMINATOR = ',',
ROWTERMINATOR = '\n'
)
SELECT * FROM emp2
```

# SELECT INTO COMMAND

- SELECT INTO statement copies data from one table to another
- Tables can be on the same SQL Server or linked SQL servers or on different types of servers using distributed queries
- SELECT INTO operates just like Bulk Insert except it can't read from an external file
- SELECT INTO can also create the destination table automatically before copying the data
- Example :
  - `select * into emps from emp2`

# SECURING DATA WITH BACKUPS

- Backing up your SQL Server databases, protects you from potentially catastrophic data loss.
- With valid backups of a database, you can recover your data from many failures, such as:
  - User errors, for example, dropping a table by mistake.
  - Hardware failures, for example, a damaged disk drive or permanent loss of a server.
  - Natural disasters.

# BACKUP AND RESTORE OF DATABASES

- The SQL Server backup and restore component provides an essential safeguard for protecting critical data stored in your SQL Server databases.
- To minimize the risk of catastrophic data loss, you need to back up your databases to preserve modifications to your data on a regular basis.
- A well-planned backup and restore strategy helps protect databases against data loss caused by a variety of failures.

# TYPES OF BACKUPS

- Full Backups :Full database backup takes a copy of the entire database including the part of the transaction log file.
- Differential Backups :Differential database backup includes only extents which were changed since the last full database backup.
- Transaction Log Backups :Transaction log backup captures all the transaction log records that have been written after the last full database backup or last transaction log backup.



Demo

# JOBS IN SQL SERVER

- A job is a specified series of operations performed sequentially
- Use jobs to define an administrative task that can be run one or more times and monitored for success or failure. A job can run on one local server or on multiple remote servers.
- A job can perform a wide range of activities and can run repetitive or schedulable tasks
- Automatically notify users of job status

# RUNNING JOBS IN SQL SERVER

- Jobs can run in several ways:
  - According to one or more schedules.
  - In response to one or more alerts.
  - By executing the `sp_start_job` stored procedure.
- Jobs can be created by users in several roles including `sysadmin` user role
- A created job can be edited by only its owners or members of `sysadmin` role.

# SCHEDULES

- A *schedule* specifies when a job runs. More than one job can run on the same schedule, and more than one schedule can apply to the same job. A schedule can define the following conditions for the time when a job runs:
  - Whenever SQL Server Agent starts. (SQL server agent is the component of SQL sever responsible for automation)
  - Whenever CPU utilization of the computer is at a level you have defined as idle.
  - One time, at a specific date and time.
  - On a recurring schedule.



# Demo

# MAINTENANCE PLANS

- A database maintenance plan is a set of specific, proactive tasks that need to be performed regularly on **databases** to ensure their adequate performance and availability.
- Maintenance Plan Wizard and Designer is to cover those critical database maintenance tasks that, as a bare minimum, should be applied to all databases, to ensure *adequate* performance and availability.

# CORE MAINTENANCE PLAN TASKS

- Backup Databases
- Verify the Integrity of Database
- Maintain a Database's Indexes
- Remove Older Data from msdb
- Remove Old Backups



Demo



Thank you!

---

---

---

# DATABASE MANAGEMENT SYSTEMS (IT 2040)

## LECTURE 08 – DATABASE SECURITY

## LEARNING OUTCOMES

- At the end of the lecture, the students should be able to:
  - Explain the three important aspects of security
  - Identify different roles and permissions in SQL server at server and database level
  - Implement a security policy of a given database using access control mechanisms in SQL server

# LECTURE CONTENT

- Aspects of database security
- Access control mechanisms
- Roles, Users and Permissions in SQL server

# DATABASE SECURITY

- The data stored in a DBMS is often vital to the business interests of the organization and is regarded as a corporate asset.
- In addition to protecting the intrinsic value of the data, corporations must consider ways to ensure privacy and to control access to data that must not be revealed to certain groups of users for various reason.

# ASPECTS OF DATABASE SECURITY

- There are three main objectives to consider while designing a secure database application:
  - **Secrecy:** Information should not be disclosed to unauthorized users.
    - For example, a student should not be allowed to examine other students' grades.
  - **Integrity:** Only authorized users should be allowed to modify data.
    - For example, students may be allowed to see their grades, yet not allowed to modify them.
  - **Availability:** Authorized users should not be denied access.
    - For example, an instructor who wishes to change a grade should be allowed to do so.

# SECURITY POLICY & ACCESS CONTROLS

- To achieve the objectives in the previous slide, a clear and consistent security policy should be developed
- A security policy specifies who is authorized to do what.
- Most users need to access only a small part of the database to carry out their tasks.
- Allowing users unrestricted access to all the data can be undesirable.
- Two main mechanisms at the DBMS level to control access to data
  - Discretionary access control
  - Mandatory access control

# DISCRETIONARY ACCESS CONTROL

- Discretionary access control is based on the concept of access rights, or privileges, and mechanisms for giving users such privileges.
- A privilege allows a user to access some data object in a certain manner (e.g., to read or to modify).
- A user who creates a database object such as a table or a view automatically gets all applicable privileges on that object.
- The DBMS subsequently keeps track of how these privileges are granted to other users and ensures that at all times only users with the necessary privileges can access an object.

# SECURITY IN SQL SERVER

- A SQL Server instance contains a hierarchical collection of entities, starting with the server.
- Each server contains multiple databases, and each database contains a collection of securable objects.
- The SQL Server security framework manages access to securable entities through authentication and authorization
  - Authentication : verifying the identities of users trying to connect to a SQL Server instance
  - Authorization : determines which data resources that authorized users can access and what actions they can take.

## SECURITY IN SQL SERVER (CONTD.)

- Authentication and authorization are achieved in SQL Server through a combination of security principals, securable, and permissions.
- Together, these three component types provide a structure for authenticating and authorizing SQL Server users.
- You must grant each principal the appropriate permissions it needs on specific securables to enable users to access SQL Server resources.

# PRINCIPALS, SECURABLE AND PERMISSIONS

- **Principals :** Individuals, groups, or processes granted access to the SQL Server instance, either at the server level or database level.
  - Server-level principals include logins and server roles.
  - Database-level principals include users and database roles.
- **Securable:** Objects that make up the server and database environment. The objects can be broken into three hierarchical levels:
  - Server-level securables include such objects as databases and availability groups.
  - Database-level securables include such objects as schemas and full-text catalogs.
  - Schema-level securables include such objects as tables, views, functions, and stored procedures.
- **Permissions:** The types of access permitted to principals on specific securables. You can grant or deny permissions to securables at the server, database, or schema level.

## STEPS TO PROVIDE USERS WITH ACCESS TO SQL SERVER RESOURCES

1. At the server level, create a login for each user that should be able to log into SQL Server.
  - create Windows authentication logins that are associated with Windows user or group accounts,
  - create SQL Server authentication logins that are specific to that instance of SQL Server.
2. Create user-defined server roles if the fixed server roles do not meet your configuration requirements.
3. Assign logins to the appropriate server roles (either fixed or user-defined).
4. For each applicable server-level securable, grant or deny permissions to the logins and server roles.

## STEPS TO PROVIDE USERS WITH ACCESS TO SQL SERVER RESOURCES (CONTD.)

5. At the database level, create a database user for each login.
  - A database user can be associated with only one server login.
6. Create user-defined database roles if the fixed database roles do not meet your configuration requirements.
7. Assign users to the appropriate database roles (either fixed or user-defined).
8. For each applicable database-level or schema-level securable, grant or deny permissions to the database users and roles.

## SCENARIO

- A software company is assigned to create a software system for a super market.
- The project manager responsible to develop the system assigns the task of creating the databases related to the projects (inventory database, payroll database) to a senior DBA (Kamal)
- The senior DBA creates the databases and all logins for the users
- The senior dba then assigns a junior DBA (Namali & Janaka) to look after each database.
- The junior DBA designs the database and assigns the task of creating tables to a database developer (Sahan)
- The junior DBA also assigns the task of entering data to some data entry operators (Amali and Mihiran)

# CREATING A LOGIN USING WINDOWS AUTHENTICATION

- A login is an individual user account for logging into the SQL Server instance.
- Syntax

```
CREATE LOGIN [domain_name\login_name]
FROM WINDOWS
[WITH DEFAULT_DATABASE = database_name
| DEFAULT_LANGUAGE = language_name];
```

- Example

```
CREATE LOGIN [win10b\winuser01] FROM WINDOWS
WITH DEFAULT_DATABASE = master, DEFAULT_LANGUAGE = us_english;
```

# CREATING A LOGIN USING SQL SERVER

## ■ Syntax

```
CREATE LOGIN [Login Name] -- This is the User that you use for login
WITH PASSWORD = 'provide_password' MUST_CHANGE,
CHECK_EXPIRATION = ON, CHECK_POLICY = ON, DEFAULT_DATABASE = [Database Name],
DEFAULT_LANGUAGE = [Language Name];-- This is Optional
```

## ■ Example

```
CREATE LOGIN sqluser01
WITH PASSWORD = 'tempPW@56789'
MUST_CHANGE, CHECK_EXPIRATION = ON,
DEFAULT_DATABASE = master, DEFAULT_LANGUAGE = us_english;
```

# CREATING AND ASSIGNING SERVER ROLES

- With server roles logins could be grouped together in order to more easily manage server-level permissions.
- SQL Server supports fixed server roles and user-defined server roles.
- With fixed server roles logins could be assigned, but permissions cannot be changed.
- For user-defined roles logins can be assigned and permission can be changed

# ADDING LOGINS TO FIXED SERVER ROLES

## ■ Assigning logins to sever roles

- Syntax :`ALTER SERVER ROLE server_role_name ADD MEMBER login`
- Server role names :

| Role                 | Description                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Sysadmin</b>      | Members of the <b>sysadmin</b> fixed server role can perform any activity in the server.                                                             |
| <b>dbcreator</b>     | Members of the <b>dbcreator</b> fixed server role can create, alter, drop, and restore any database.                                                 |
| <b>securityadmin</b> | Members of the <b>securityadmin</b> fixed server role manage logins and their properties. They can GRANT, DENY, and REVOKE server-level permissions. |

- Ex :`ALTER SERVER ROLE diskadmin ADD MEMBER Ted`

# CREATING USER-DEFINED SERVER ROLES AND PERMISSIONS

- Creating a user defined sever role and assigning permissions
  - Syntax for creating a server role: CREATE SERVER ROLE role\_name
    - Ex: CREATE SERVER ROLE devops;
  - Syntax for granting permissions : GRANT permission TO grantee\_principal [WITH GRANT OPTION]
    - Permission Examples :
      - CREATE ANY DATABASE — Create a database on the server.
      - ALTER ANY DATABASE — Create, alter, or drop any login in the instance.
      - ALTER ANY LOGIN — Modify any login.
      - SHUTDOWN — Shut down the server.
    - Ex: GRANT ALTER ANY DATABASE TO devops;

## REVISIT TO SCENARIO

- Senior DBA, Junior DBA, Database Developer and Data entry operator should have logins.
- According to the scenario, Senior dba also should be able to create databases and logins.
- Therefore two methods could be used to create logins
  - All logins could be created by the project manager
  - Project manager could create the login of Senior DBA and Senior DBA could create the other logins. (we will use this approach)

## REVISIT TO SCENARIO (CONTD.)

- First create the login for the senior DBA using windows authentication of SQL server authentication as follows:

- Windows authentication

```
CREATE LOGIN PC2\kamal
```

```
FROM WINDOWS WITH DEFAULT_DATABASE = Master
```

- SQL server authentication

```
CREATE LOGIN Kamal
```

```
WITH PASSWORD = 'kamal@123', DEFAULT_DATABASE = Master
```

## REVISIT TO SCENARIO (CONTD.)

- Now let's give permission to the senior DBA to create databases and logins.
- This could be done using pre-defined sever roles or user defined server roles as follows

- Using pre-defined server roles

```
alter server role dbcreator add member kamal
```

```
alter server role securityadmin add member kamal
```

- Using user-defined server role

```
create server role seniordba
```

```
grant create any database,alter any login to seniordba
```

```
alter server role seniordba add member kamal
```

# DATABASE USERS

- Logins are created at the server level, while users are created at the database level.
- In other words, a login allows to connect to the SQL Server service and permissions inside the database are granted to the database users, not the logins.
- The logins will be assigned to server roles (for example, `serveradmin`)
- The database users will be assigned to roles within that database
- Syntax : `CREATE USER user_name FOR LOGIN login_name`
  - `user_name` :The name of the database user that you wish to create.
  - `login_name` :The Login used to connect to the SQL Server instance

## REVISIT TO SCENARIO

- Senior DBA can now create the databases and create a logins for the junior DBAs who are in-charge of the databases
- Create the login first

```
CREATE LOGIN namali
```

```
WITH PASSWORD = 'namali@123', DEFAULT_DATABASE = Inventory
```

- The senior DBA can also create all the other logins for the other roles here
- Now let's create the user for the login so that the user can be given permission at the database level

```
Create user namali for login namali
```

## CREATING DATABASE ROLES

- A database role is a group of users that share a common set of database-level permissions.
- As with server roles, SQL Server supports both fixed and user-defined database roles.
- To set up a user-defined database role, you must create the role, grant permissions to the role, and add members to the role (or add members and then grant permissions).

# ASSIGNING USERS TO FIXED DATABASE ROLES

- Some examples of database roles are as follows:

| Role                    | Description                                                                                                                                                                |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>db_owner</b>         | Members of the <b>db_owner</b> fixed database role can perform all configuration and maintenance activities on the database, and can also drop the database in SQL Server. |
| <b>db_securityadmin</b> | Members of the <b>db_securityadmin</b> fixed database role can modify role membership for custom roles only and manage permissions.                                        |
| <b>db_accessadmin</b>   | Members of the <b>db_accessadmin</b> fixed database role can add or remove access to the database                                                                          |
| <b>db_ddladmin</b>      | Members of the <b>db_ddladmin</b> fixed database role can run any Data Definition Language (DDL) command in a database.                                                    |

- Syntax for windows authentication :ALTER ROLE db\_datawriter ADD MEMBER <domain\username>
- Syntax for SQL authentication :ALTER ROLE db\_datawriter ADD MEMBER <username>;

## REVISIT TO SCENARIO

- The junior DBAs are expected to manage the databases.
- Therefore they could be assigned with the db\_owner role by the senior DBA as follows :

```
use inventoryDB
```

```
alter role db_owner add member namali
```

```
alter role db_securityadmin add member namali
```

```
alter role db_accessadmin add member namali
```

# ASSIGNING USERS TO USER DEFINED DATABASE ROLES

- Creating a user defined role
  - Syntax :CREATE ROLE <role\_name>
  - Example :CREATE ROLE student
- Assigning login to the role
  - Syntax :ALTER ROLE <role\_name> ADD MEMBER <username>
  - Example :ALTER ROLE student ADD MEMBER Brady

# PERMISSIONS

- Every SQL Server securable has associated permissions that can be granted to user.
- At the database level they are assigned to database users and database roles.
- Some permissions are as follows :

| Role         | Description                                                                                                   |
|--------------|---------------------------------------------------------------------------------------------------------------|
| ALTER        | Grants or denies the ability to alter the existing database.                                                  |
| CREATE TABLE | Grants or denies the ability to create a table.                                                               |
| INSERT       | Grants or denies the ability to issue the INSERT command against all applicable objects within the database.  |
| EXECUTE      | Grants or denies the ability to issue the EXECUTE command against all applicable objects within the database. |
| REFERENCES   | The REFERENCES permission on a table is needed to create a FOREIGN KEY constraint                             |

# CREATING AND REMOVING PERMISSIONS

- Creating and removing permissions could be done using **GRANT**, **DENY** and **REVOKE**
  - **GRANT** – Grants permissions on a securable to a principal.
  - **DENY** – Denies a permission to a principal.
  - **REVOKE** – Removes a previously granted or denied permission.

# GRANTING PERMISSIONS

- **Granting permissions to a role/user**
  - Ex: GRANT SELECT TO GrantSelectRole
  - Ex: GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES, EXECUTE ON Northwind TO sqluser01;
- **Denying permissions to a role/user**
  - Ex: DENY SELECT TO DenyTest
- **Revoking permissions assigned to a user**
  - Ex: REVOKE SELECT TO GrantSelectRole;

## REVISIT TO SCENARIO

- The database developers need to create tables which would probably have foreign keys
- Lets create a role names database developer for this
  - create role databaseDev
  - grant create table, alter, references on inventoryDB to databaseDev
  - alter role databaseDev add member Sahan
- Similarly create another role with select and insert for the data Entry operators

## VIEWS AND SECURITY

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Together with GRANT/REVOKE commands, views are a very powerful access control tool



Thank you!