Database Management Systems

Introduction

Introduction

What is a Database?

A collection of related pieces of data:

- Representing/capturing the information about a real-world enterprise or part of an enterprise.
- Collected and maintained to serve specific data management needs of the enterprise.
- Activities of the enterprise are supported by the database and continually update the database.

An Example

University Database:

Data about students, faculty, courses, research-laboratories, course registration/enrollment etc.

Reflects the state of affairs of the academic aspects of the university.

Purpose: To keep an accurate track of the academic activities of the university.

Database Management System (DBMS)

A *general purpose* software system enabling:

- Creation of large disk-resident databases.
- Posing of data retrieval queries in a standard manner.
- Retrieval of query results efficiently.
- Concurrent use of the system by a large number of users in a consistent manner.
- Guaranteed availability of data irrespective of system failures.

OS File System Storage Based Approach

- Files of records – used for data storage
  - data redundancy – wastage of space
  - maintaining consistency becomes difficult
- Record structures – hard coded into the programs
  - structure modifications – hard to perform
- Each different data access request (a query)
  - performed by a separate program
  - difficult to anticipate all such requests
- Creating the system
  - requires a lot of effort
- Managing concurrent access and failure recovery are difficult

DBMS Approach

DBMS

->separation of data and metadata

->flexibility of changing metadata

->program-data independence

Data access language

->standardized – SQL

->ad-hoc query formulation –

easy System development

- less effort required
- concentration on logical level design is enough
- components to organize data storage

process queries, manage concurrent access, recovery from failures, manage access control are all available

Data Model

Collection of conceptual tools to describe the database at a certain level of abstraction.

- *Conceptual Data Model*
  - a high level description
  - useful for requirements understanding.
- *Representational Data Model*
  - describing the logical representation of data without giving details of physical representation.
- *Physical Data Model*
  - description giving details about record formats, file structures etc.

E/R (Entity/Relationship) Model

- A conceptual level data model.
- Provides the concepts of *entities*, *relationships* and *attributes*.

The University Database Context

Entities: *student*, *faculty member*, *course*, *departments* etc.

Relationships: *enrollment* relationship between student & course,
employment relationship between faculty member, department etc.

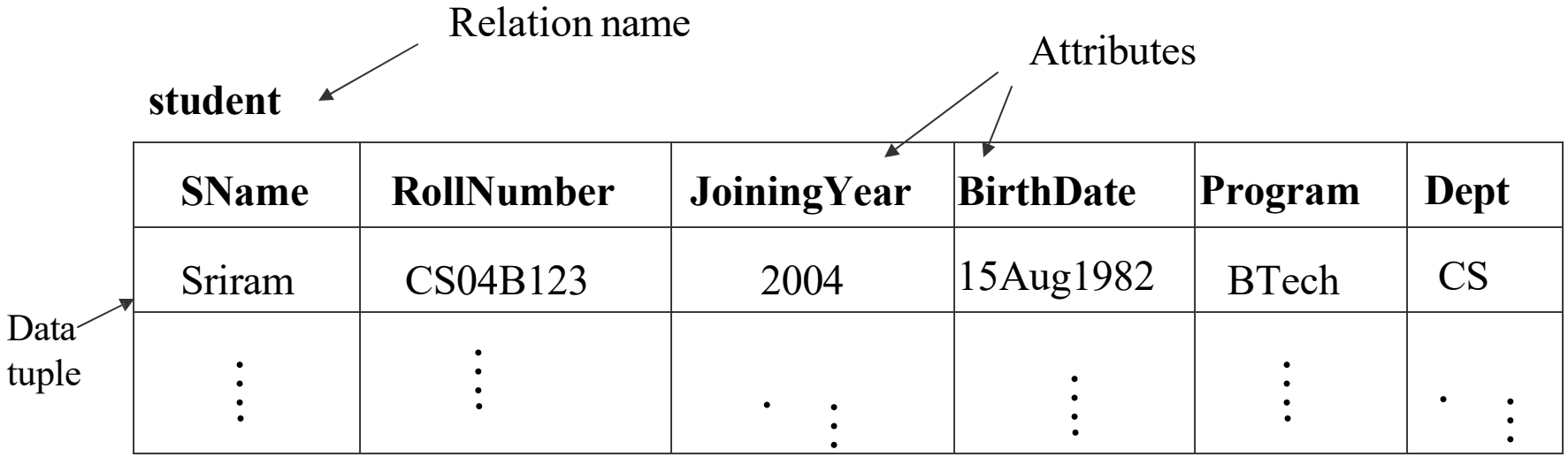
Attributes: *name*, *rollNumber*, *address* etc., of *student* entity,
name, *empNumber*, *phoneNumber* etc., of faculty entity etc.

More details will be given in the E/R Model Module.

Representational Level Data Model

Relational Model : Provides the concept of a relation.

In the context of university database:



SName	RollNumber	JoiningYear	BirthDate	Program	Dept
Sriram	CS04B123	2004	15Aug1982	BTech	CS
⋮	⋮	⋮	⋮	⋮	⋮

Relation scheme: Attribute names of the relation.

Relation data/instance: set of data tuples.

More details will be given in Relational Data Model Module.

Data versus Schema or Meta-Data

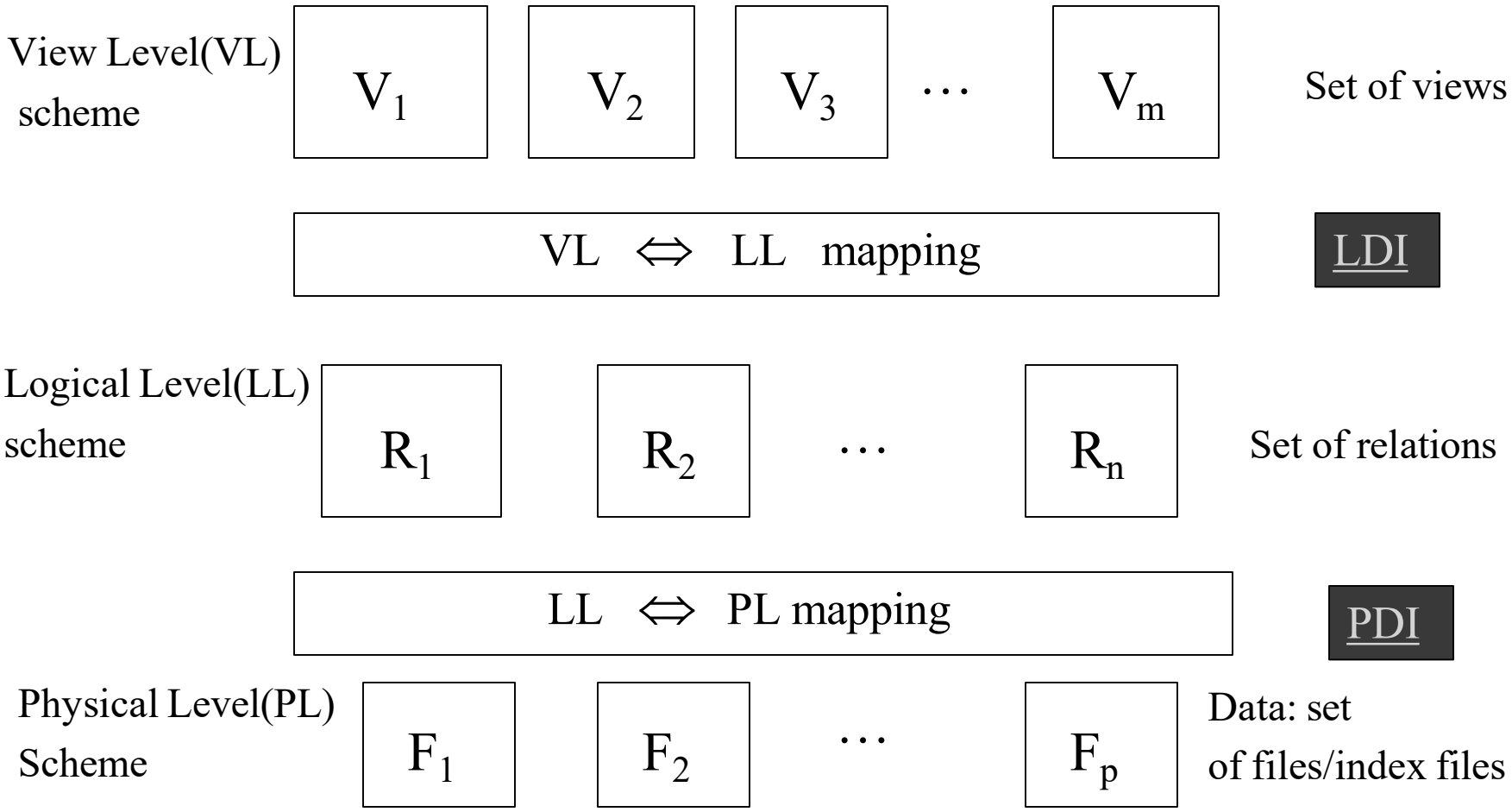
- DBMS is generic in nature
  - not tied to a single database
  - capable of managing several databases at a time
- Data and schema are stored separately.

- In RDBMS context:

  Schema – table names, attribute names with their data types for each table and constraints etc.

- Database definition – setting up the skeleton structure
- Database Loading/populating – storing data

Abstraction Levels in a DBMS: Three-Schema Architecture



Three-schema Architecture(1/2)

View Level Schema

Each view describes an aspect of the database relevant to a particular group of users.

For instance, in the context of a library database:

- Books Purchase Section
- Issue/Returns Management Section
- Users Management Section

Each section views/uses a portion of the entire data.

Views can be set up for each section of users.

Three-schema Architecture(2/2)

Logical Level Schema

- Describes the logical structure of the entire database.
- No physical level details are given.

Physical Level Schema

- Describes the physical structure of data in terms of record formats, file structures, indexes etc.

Remarks

- Views are optional
  - Can be set up if the DB system is very large and if easily identifiable user-groups exist
- The logical scheme is essential
- Modern RDBMS's hide details of the physical layer

Physical Data Independence

The ability to modify physical level schema without affecting the logical or view level schema.

Performance tuning – modification at physical level creating a new index etc.

Physical Data Independence – modification is localized

- achieved by suitably modifying PL-LL mapping.
- a very important feature of modern DBMS.

Three Schema Arch

Logical Data Independence

The ability to change the logical level scheme without affecting the view level schemes or application programs

Adding a new attribute to some relation

- no need to change the programs or views that don't require to use the new attribute

Deleting an attribute

- no need to change the programs or views that use the remaining data
- view definitions in VL-LL mapping only need to be changed for views that use the deleted attribute

Three-schema Architecture

Development Process of a Database System (1/2)

Step 1. Requirements collection

- *Data model requirements*
  - various pieces of data to be stored and the interrelationships.
  - presented using a conceptual data model such as E/R model.
- *Functional requirements*
  - various operations that need to be performed as part of running the enterprise.
    - acquiring a new book, enrolling a new user, issuing a book to the user, recording the return of a book etc.

Development process of a database system (2/2)

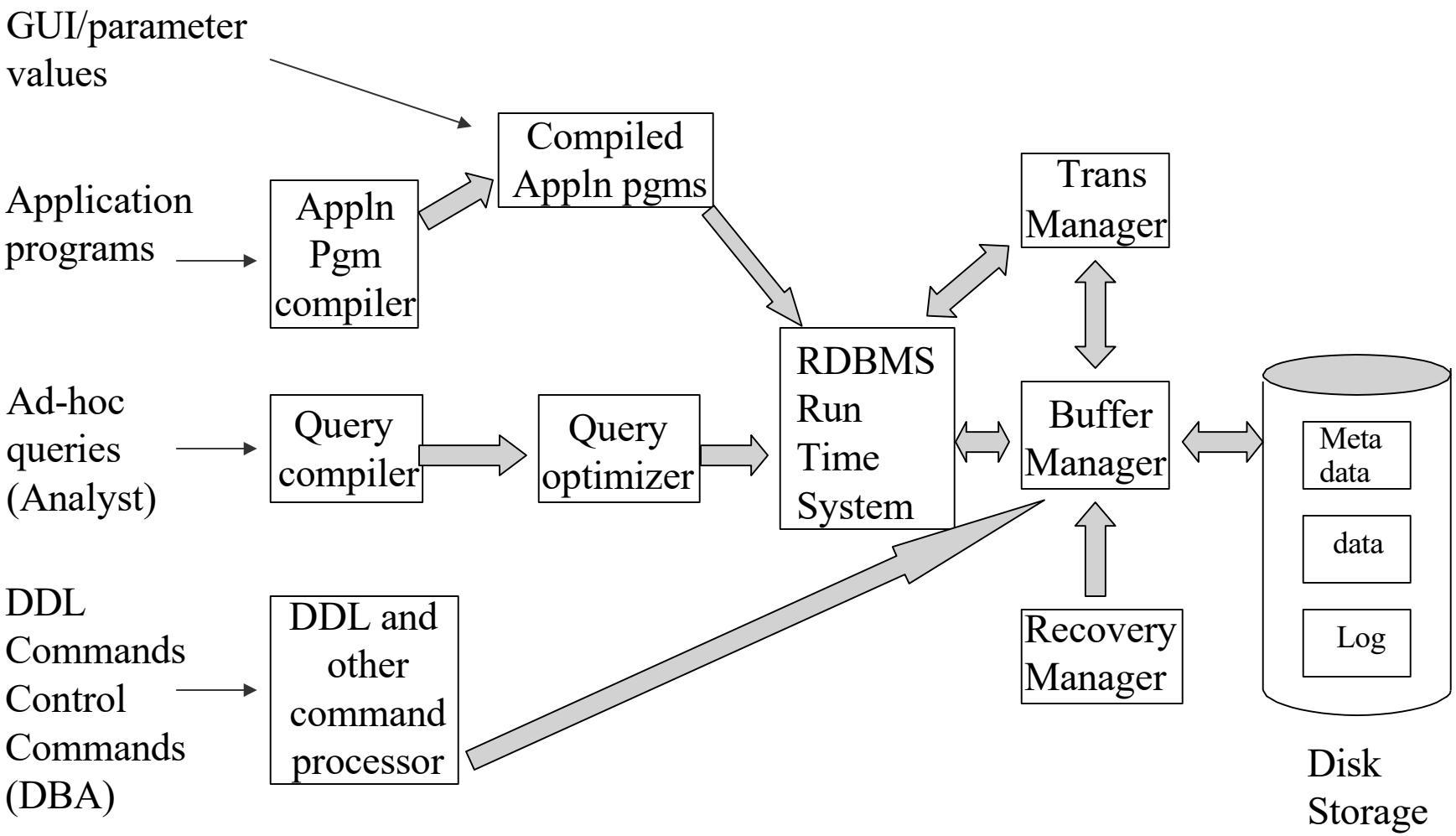
Step 2. Convert the data model into a representational level model

- typically relational data model.
- choose an RDBMS system and create the database.

Step 3. Convert the functional requirements into application programs

- programs in a high-level language that use embedded SQL to interact with the database and carry out the required tasks.

Architecture of an RDBMS system



Architecture Details (1/3)

Disk Storage:

Meta-data – schema

- table definition, view definitions, mappings

Data – relation instances, index structures

statistics about data

Log – record of database update operations

essential for failure recovery

DDL and other command processor:

Commands for relation scheme creation

Constraints setting

Commands for handling authorization and data access control

Architecture Details (2/3)

Query compiler

Compiles
- SQL adhoc queries
- update / delete commands

Query optimizers

Selects a near optimal plan for executing a query

- relation properties and index structures are utilized

Application Program Compiler

Preprocess to separate embedded SQL commands

Use host language compiler to compile rest of the program

Integrate the compiled program with the libraries for SQL commands supplied by RDBMS

Architecture Details (3/3)

RDBMS Run Time System:

- Executes Compiled queries, Compiled application programs
- Interacts with Transaction Manager, Buffer Manager

Transaction Manager:

- Keeps track of start, end of each transaction
- Enforces concurrency control protocols

Buffer Manager:

- Manages disk space
- Implements paging mechanism

Recovery Manager:

- Takes control as restart after a failure
- Brings the system to a consistent state before it can be resumed

Roles for people in an Info System Management (1/2)

Naive users / Data entry operators

- Use the GUI provided by an application program
- Feed-in the data and invoke an operation
  - e.g., person at the train reservation counter,
    person at library issue / return counter
- No deep knowledge of the IS required

Application Programmers

- Embed SQL in a high-level language and develop programs to handle functional requirements of an IS
- Should thoroughly understand the logical schema or relevant views
- Meticulous testing of programs - necessary

Roles for people in an Info System management (2/2)

Sophisticated user / data analyst:

- Uses SQL to generate answers for complex queries

DBA (Database Administrator)

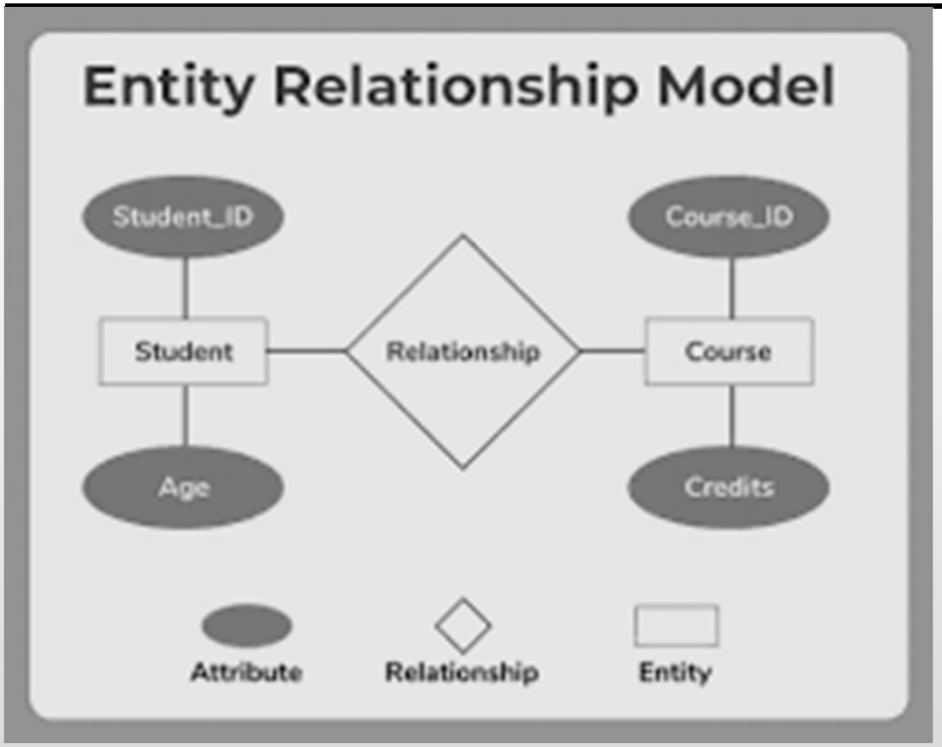
- Designing the logical scheme
- Creating the structure of the entire database
- Monitor usage and create necessary index structures to speed up query execution
- Grant / Revoke data access permissions to other users etc.

Text Books

- Ramez Elmasri and Shamkant B Navathe, *Fundamentals of Database Systems*, 3rd Edition, Addison Wesley, 2000.
- A Silberschatz, H F Korth and S Sudarshan, *Database System Concepts*, 5th Edition, 2006.

ENTITY-RELATIONSHIP (E/R) MODEL…..

ENTITY-RELATIONSHIP (E/R) MODEL



ENTITY-RELATIONSHIP (E/R) MODEL

- Widely used conceptual level *data model*
  - proposed by Peter P Chen in 1970s
- Data model to describe the database system at the requirements collection stage
  - high level description.
  - easy to understand for the enterprise managers.
  - rigorous enough to be used for system building.
- Concepts available in the model
  - entities and attributes of entities.
  - relationships between entities.
  - diagrammatic notation.

- Entities

- *Entity* - a thing (animate or inanimate) of independent physical or conceptual existence and *distinguishable*.

  - In the University database context, an individual *student*, *faculty member*, a *class room,* a *course* are entities.

-

- *Entity Set* or *Entity Type*-

  - Collection of entities all having the same properties. *Student* entity set – collection of all *student* entities. *Course* entity set – collection of all *course* entities.

ATTRIBUTES

- Each entity is described by a set of attributes/properties that have associated values
- student entity
  - StudName – name of the student.
  - RollNumber – the roll number of the student.
  - Sex – the gender of the student etc.
- All entities in an Entity set/type have the same set of attributes. Chosen set of attributes – amount of detail in modeling.

TYPES OF ATTRIBUTES (1/2)

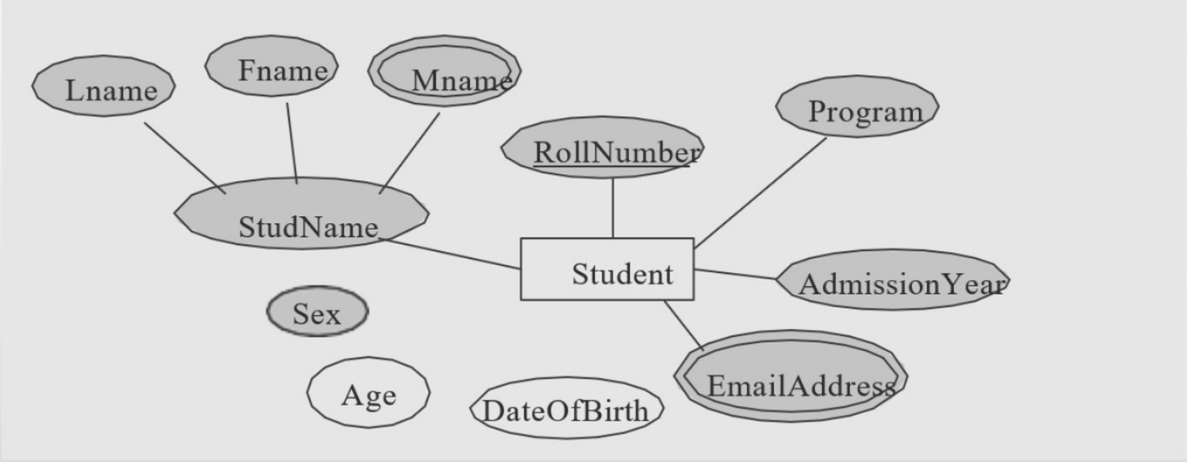
- Simple Attributes
  - having atomic or indivisible values.
    - `example`: *Dept* – a string
      - *PhoneNumber* – a ten digit number
    - Composite Attributes
      - having several components in the value.
      - `example`: *Qualification* with components (*DegreeName, Year*, *UniversityName*)
    - Derived Attributes
      - Attribute value is dependent on some other attribute.
      - `example`: *Age* depends on *DateOfBirth*. So age is a derived attribute.

TYPES OF ATTRIBUTES (2/2)

- Single-valued
  - having only one value rather than a set of values.
  - for instance, *PlaceOfBirth* – single string value.
- Multi-valued
  - having a set of values rather than a single value.
  - for instance, *CoursesEnrolled* attribute for student *EmailAddress* attribute for student *PreviousDegree* attribute for student.
- Attributes can be:
  - simple single-valued, simple multi-valued,
  - composite single-valued or composite multi-valued.

DIAGRAMMATIC NOTATION FOR ENTITIES

- *entity* - rectangle
- *attribute* - ellipse connected to rectangle
- *multi-valued attribute* - double ellipse
- *composite attribute* - ellipse connected to ellipse
- *derived attribute* - dashed ellipse



DOMAINS OF ATTRIBUTES

- Each attribute takes values from a set called its domain
- For instance, studentAge – {17,18, …, 55}
- HomeAddress – character strings of length 35
- Domain of composite attributes –
  - cross product of domains of component attributes
- Domain of multi-valued attributes –
  - set of subsets of values from the basic domain

ENTITY SETS AND KEY ATTRIBUTES

- *Key* – an attribute or a collection of attributes whose value(s) uniquely identify an entity in the entity set.
- For instance,
  - *RollNumber* - Key for *Student* entity set
  - *EmpID* - Key for *Faculty* entity set
  - *HostelName, RoomNo* - Key for *Student* entity set (assuming that each student gets to stay in a single room)
- A key for an entity set may have more than one attribute.
- An entity set may have more than one key.
- Keys can be determined only from the meaning of the attributes in the entity type.
  - Determined by the designers

RELATIONSHIPS

- When two or more entities are associated with each other, we have an instance of a *Relationship*.
- E.g.: *student* Ramesh *enrolls* in Discrete Mathematics *course*
- Relationship *enrolls* has *Student* and *Course* as the
- *participating* entity sets.

- Formally, *enrolls* $\subseteq$ *Student* $\times$ *Course*
  - (s,c) $\in$ *enrolls* $\Leftrightarrow$ Student 's' has enrolled in Course 'c'
  - Tuples in *enrolls* – relationship instances
  - *enrolls* is called a relationship Type/Set.

DEGREE OF A RELATIONSHIP

- *Degree* : the number of participating entities.
  - Degree 2: *binary*
  - Degree 3: *ternary*
  - Degree n: *n-ary*
- Binary relationships are very common and widely used.

DIAGRAMMATIC NOTATION FOR RELATIONSHIPS

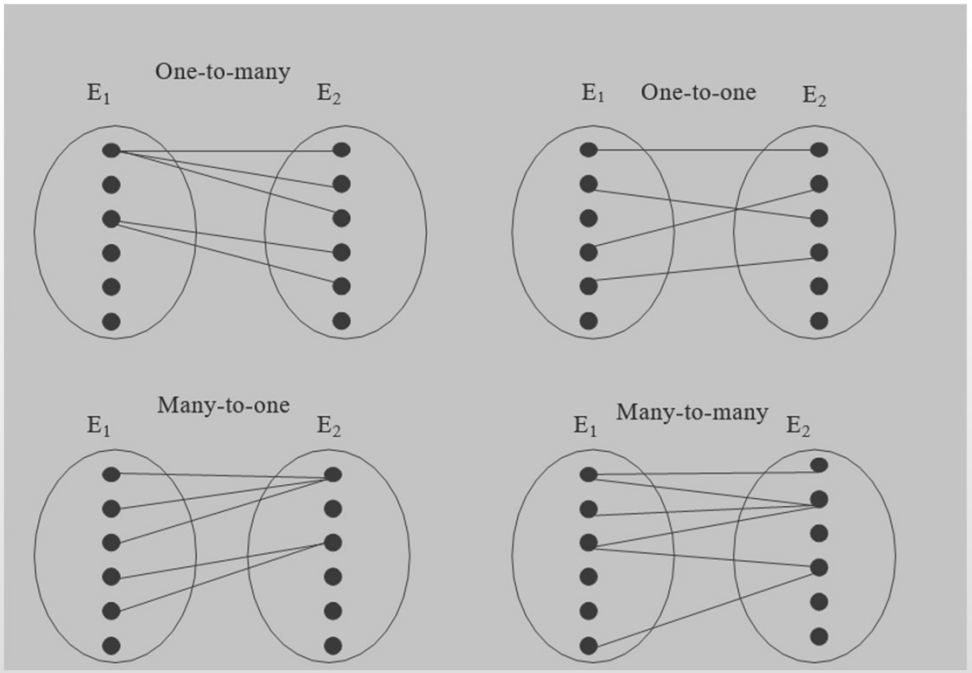
- Relationship – diamond shaped box
  - Rectangle of each participating entity is connected by a line to this diamond. Name of the relationship is written in the box.

BINARY RELATIONSHIPS AND CARDINALITY RATIO

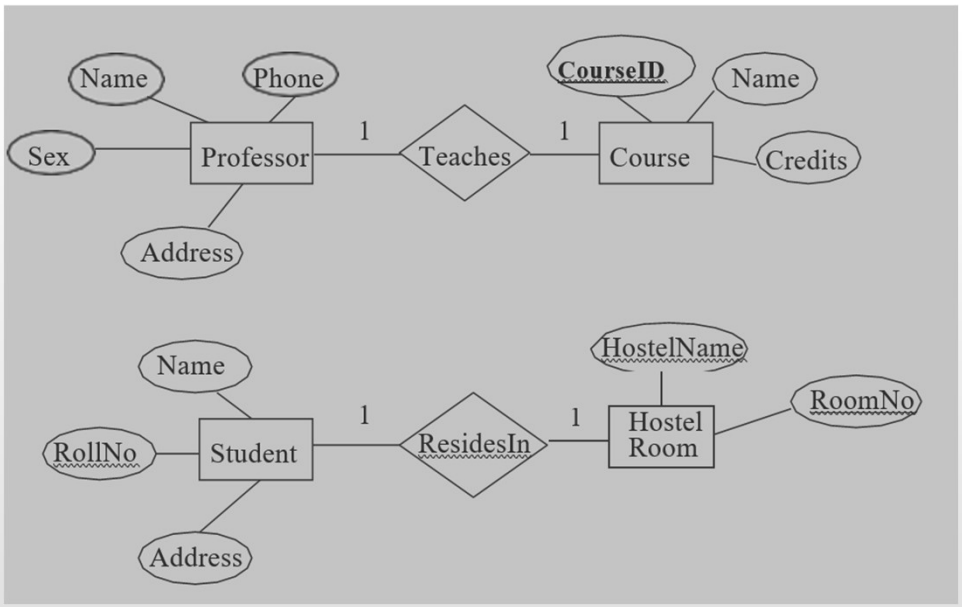
- The *maximum* number of entities from E_2 that an entity from E_1 can possibly be associated thru R (and vice-versa) determines the *cardinality ratio* of R.
-
- Four possibilities are usually specified:
  - *one-to-one (1:1)*
  - *one-to-many (1:N)*
  - *many-to-one (N:1)*
  - *many-to-many (M:N)*

CARDINALITY RATIOS

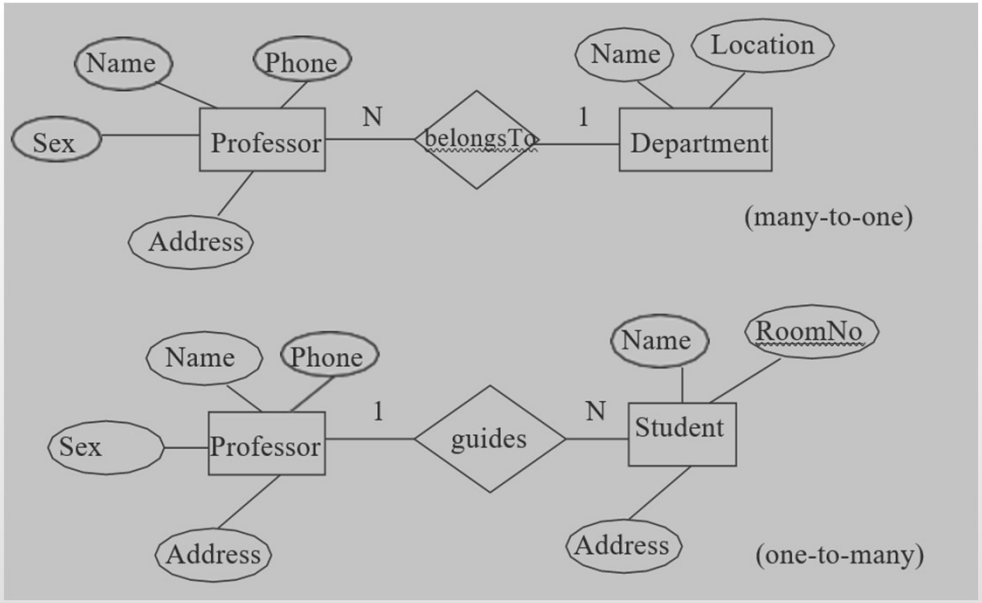
- *One-to-one:* An E_1 entity may be associated with at
  - most one E_2 entity and similarly
  - an E_2 entity may be associated with at most one E_1 entity.
- *One-to-many:* An E_1 entity may be associated with
  - many *E2* entities whereas an *E2* entity may be associated with at most one E_1 entity.
- *Many-to-one:* An E_2 entity may be associated with
  - many *E1* entities whereas an *E1* entity may be associated with at most one E_2 entity.
- *Many-to-many:* Many E_1 entities may be associated with a
  - single *E2* entity and a single *E1* entity may be associated with many E_2 entities.


One-to-many
E1
E2
E1
One-to-one
E2
Many-to-one
E1
E2
E1
Many-to-many
E2

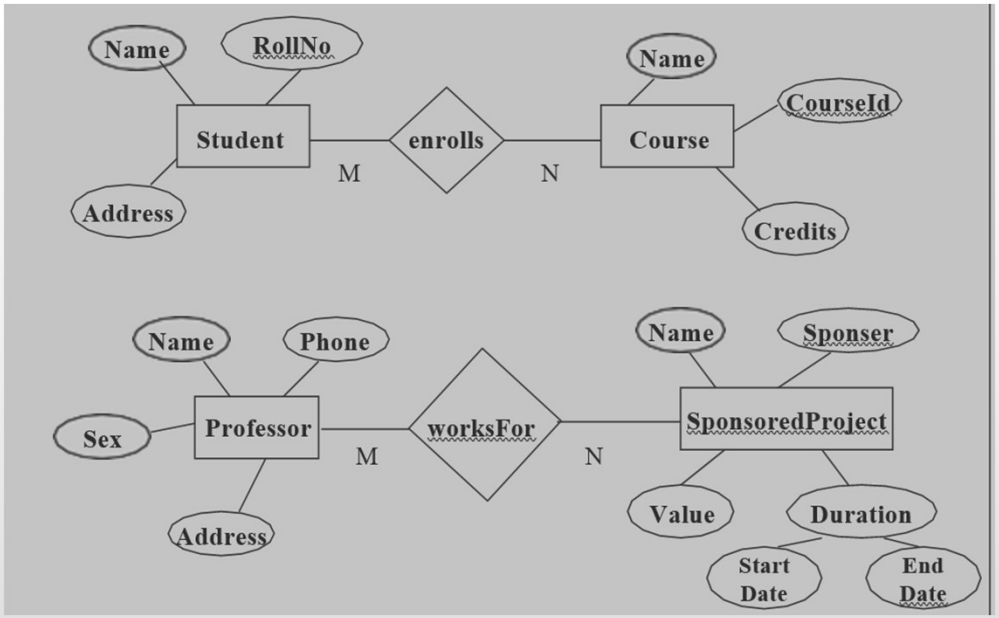
CARDINALITY RATIO – EXAMPLE (*ONE-TO-ONE*)



CARDINALITY RATIO – EXAMPLE (*MANY-TO-ONE*/*ONE-TO-MANY*)



CARDINALITY RATIO – EXAMPLE (*MANY-TO-MANY*)

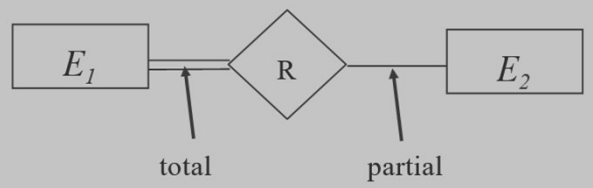


PARTICIPATION CONSTRAINTS

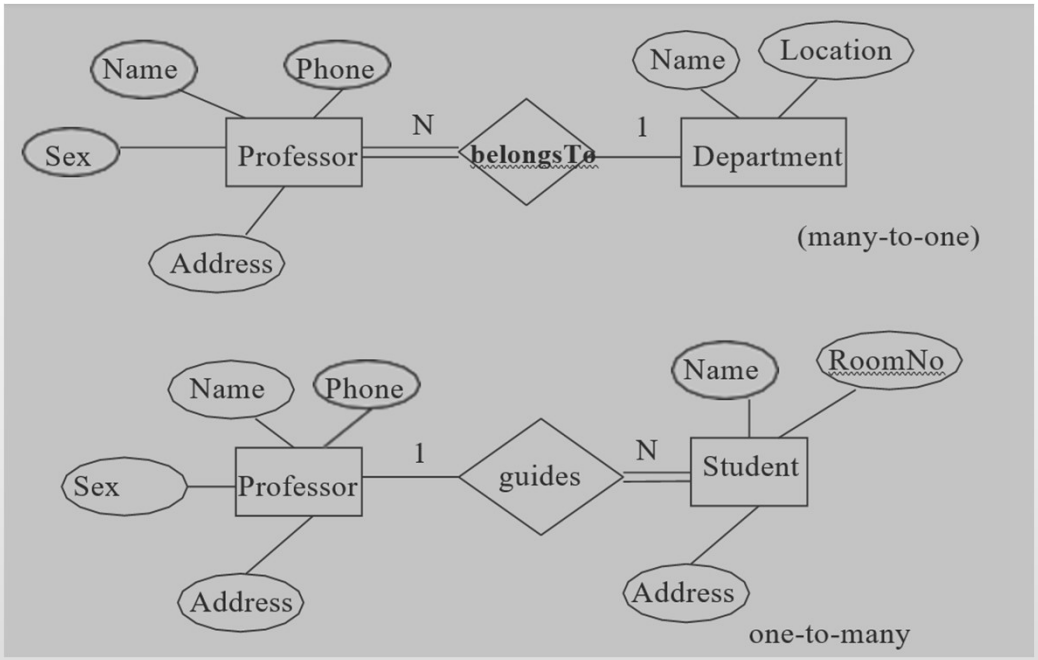
An entity set may participate in a relation either *totally* or

- *partially*.
  - *Total participation*: Every entity in the set is involved in some association (or tuple) of the relationship.
  - *Partial participation*: Not all entities in the set are involved in association (or tuples) of the relationship.

Notation:



EXAMPLE OF TOTAL/PARTIAL PARTICIPATION

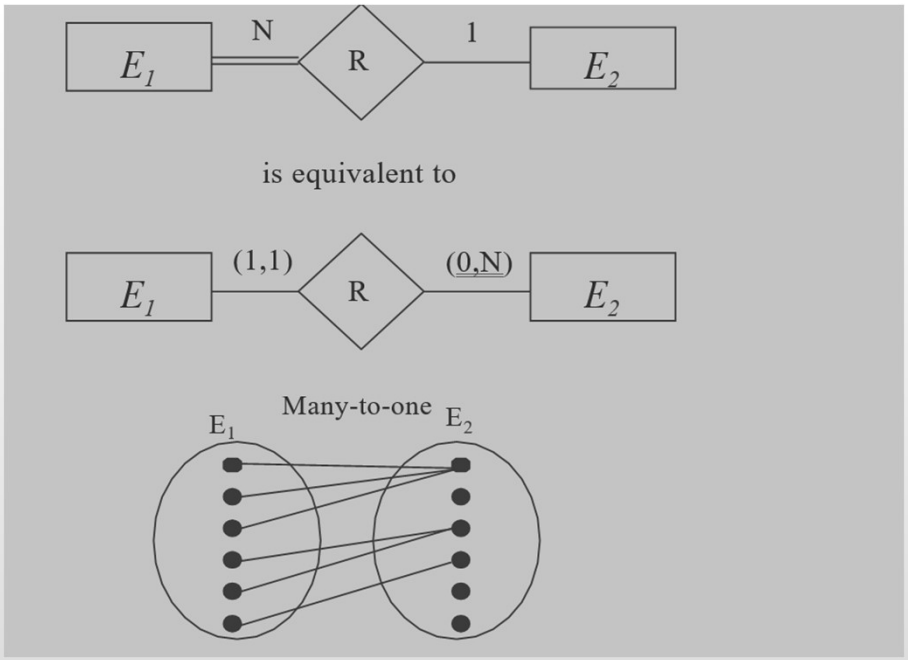


STRUCTURAL CONSTRAINTS

- Cardinality Ratio and Participation Constraints are together called *Structural Constraints*.
- They are called *constraints* as the *data* must satisfy them to be consistent with the requirements.
- *Min-Max notation*: pair of numbers (*m,n*) placed on the line connecting an entity to the relationship.
- *m*: the minimum number of times a particular entity *must appear* in the relationship tuples at any point of time
  - 0 – partial participation
  - ≥ 1 – total participation

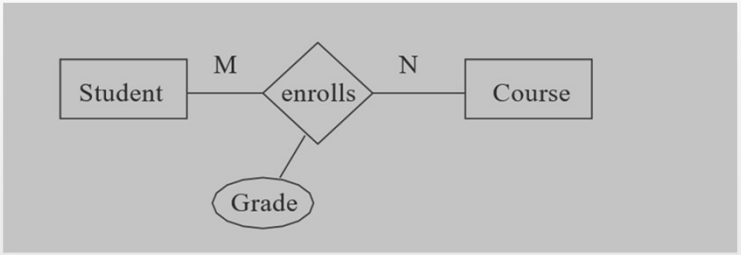
- *n*: similarly, the maximum number of times a particular entity
- *can appear* in the relationship tuples at any point of time

COMPARING THE NOTATIONS



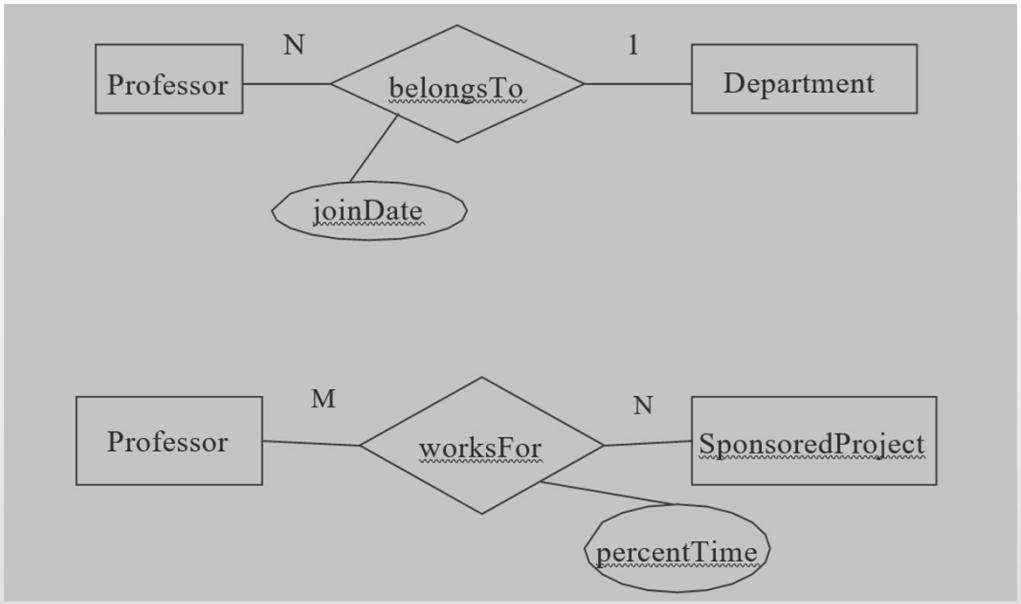
ATTRIBUTES FOR RELATIONSHIP TYPES

- Relationship types can also have attributes.
- properties of the association of entities.



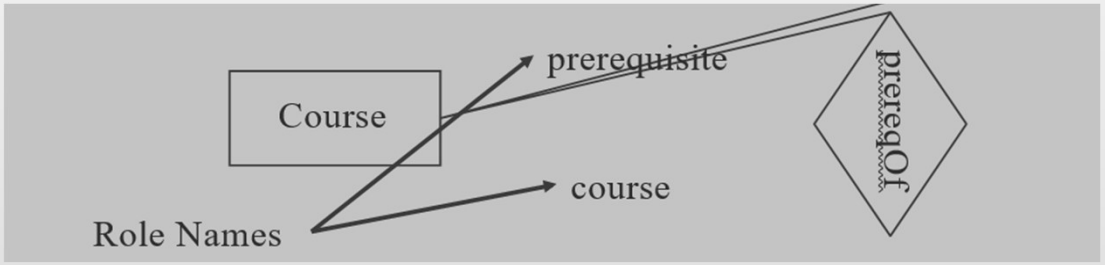
- *grade* gives the letter grade (S,A,B, etc.) earned by the student for a course.
  - neither an attribute of *student* nor that of *course*.

ATTRIBUTES FOR RELATIONSHIP TYPES – MORE EXAMPLES



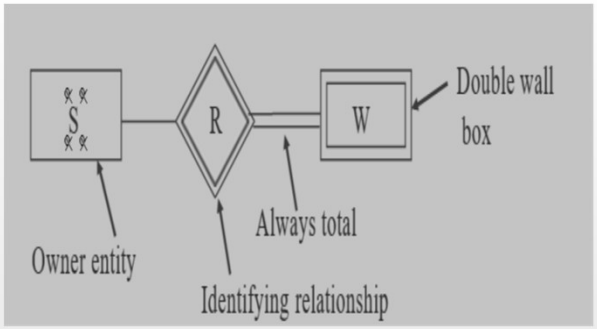
RECURSIVE RELATIONSHIPS AND ROLE NAMES

- Recursive relationship: An entity set relating to itself gives rise to a *recursive* relationship
- E.g., the relationship *prereqOf* is an example of a recursive relationship on the entity *Course*
- Role Names – used to specify the exact role in which the entity participates in the relationships
  - Essential in case of recursive relationships
  - Can be optionally specified in non-recursive cases



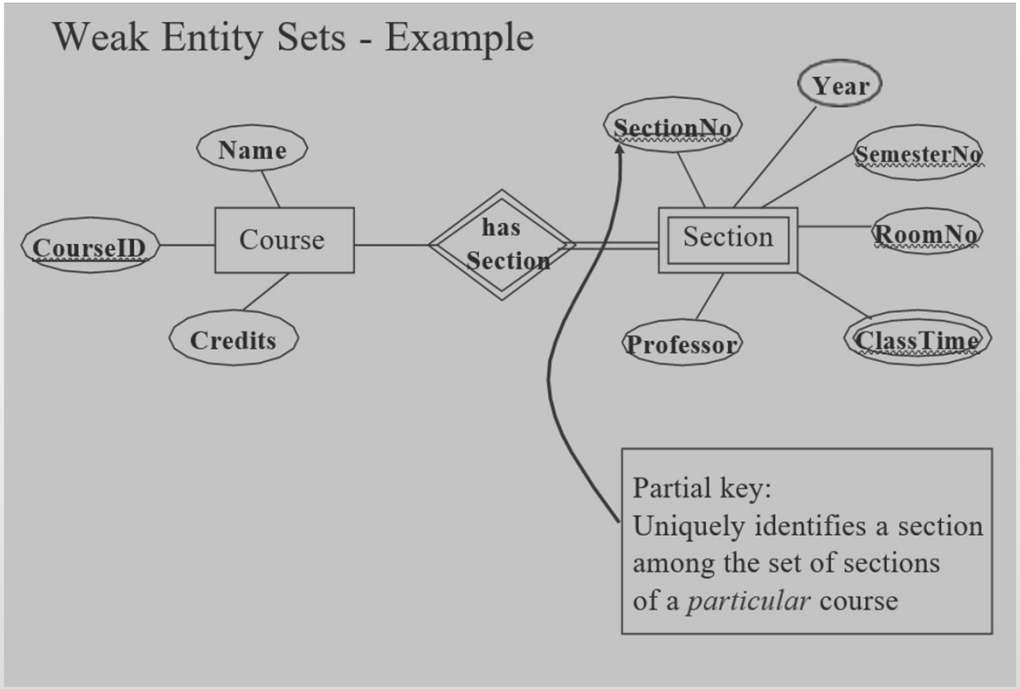
WEAK ENTITY SETS

- Weak Entity Set: An entity set whose members owe their
  - existence to some entity in a *strong entity set.*
- weak entities are not of independent existence.
- each weak entity is associated with some entity of the



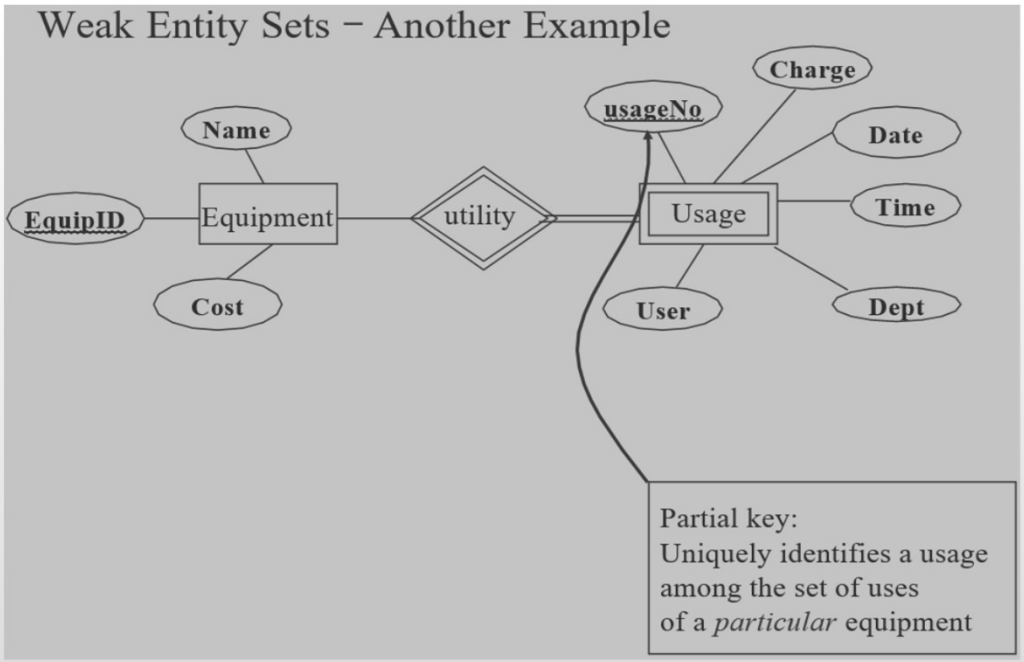
- *Owner* entity set through a special relationship
- Weak entity set may not have a key attribute
- The weak entity type always has total participation(existence dependency) in a relationship because the weak entity type can not be identified without an owner identity.
- A weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two.

EXAMPLES…



A popular course may have several sections each taught by a different professor and having its own class room and meeting times

EXAMPLES…



Institute has many pieces of equipment and we like to keep track of their utilization. Keeping track of the usage is relevant only when the equipment exists!

COMPLETE EXAMPLE FOR E/R SCHEMA: SPECIFICATIONS (1/2)

- In an educational institute, there are several departments and each student belongs to one of them. Each department has a unique department number, a name, a location, phone number and is headed by a professor. Professors have a unique employee Id, name and a phone number. A professor works for exactly one department.
- We like to keep track of the following details regarding students: name, unique roll number, sex, phone number, date of birth, age and one or more email addresses. Students have a local address consisting of the hostel name and the room number. They also have home address consisting of house number, street, city and PIN. It is assumed that all students reside in the hostels.

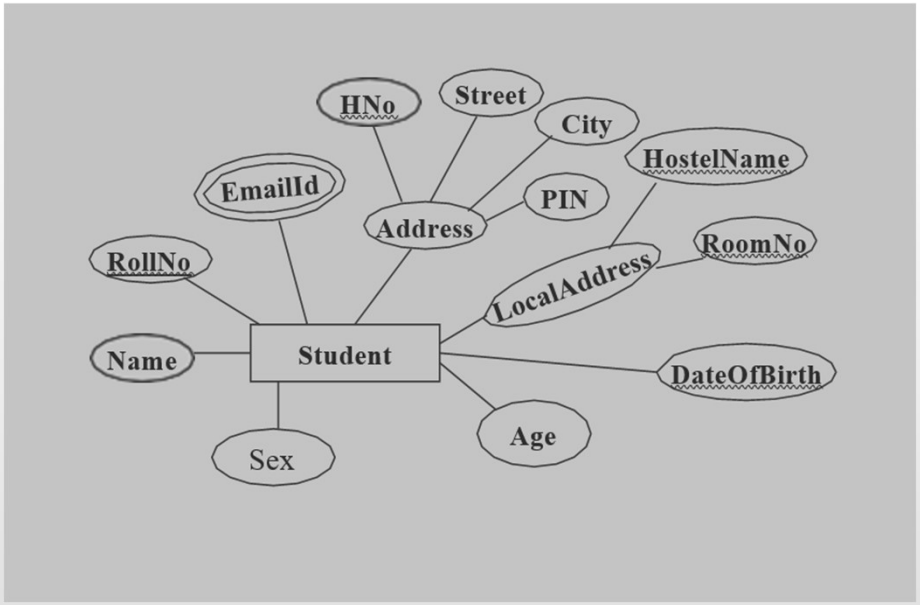
COMPLETE EXAMPLE FOR E/R SCHEMA: SPECIFICATIONS (2/2)

- A course taught in a semester of the year is called a *section.* There can be several sections of the same course in a semester; these are identified by the *section number*. Each section is taught by a professor and has its own timings and a room to meet. Students enroll for several sections in a semester.

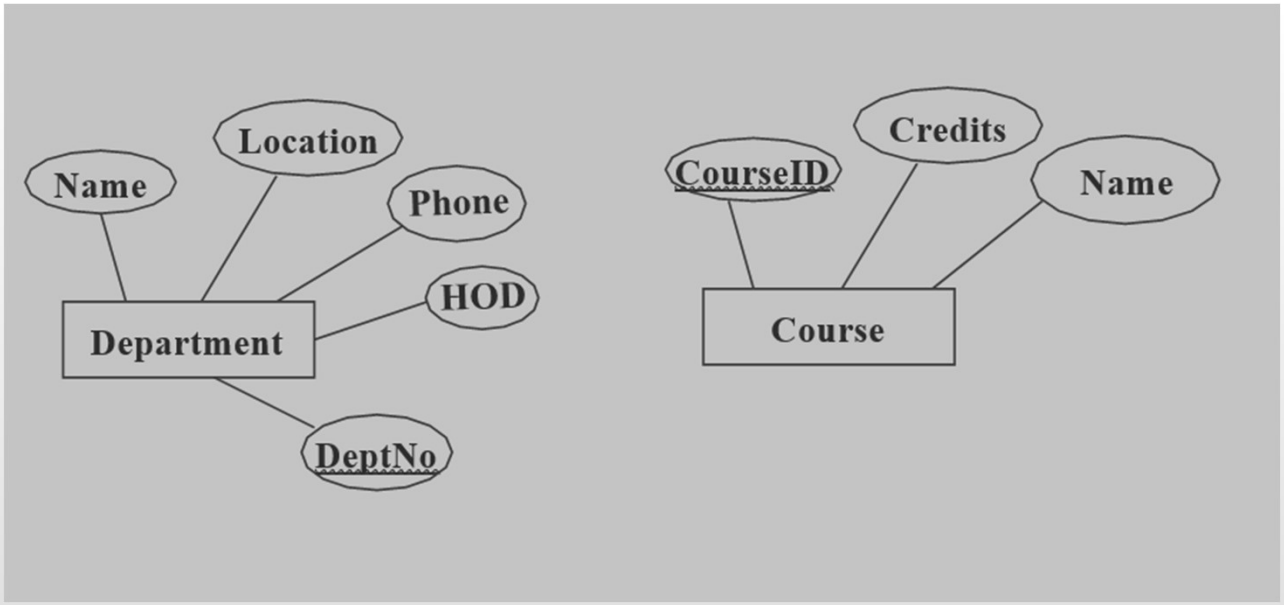
- Each course has a name, number of credits and the department that offers it. A course may have other courses as pre-requisites i.e, courses to be completed before it can be enrolled in.

- Professors also undertake research projects. These are sponsored by funding agencies and have a specific start date, end date and amount of money given. More than one professor can be involved in a project. Also a professor may be simultaneously working on several projects. A project has a unique *projectId.*

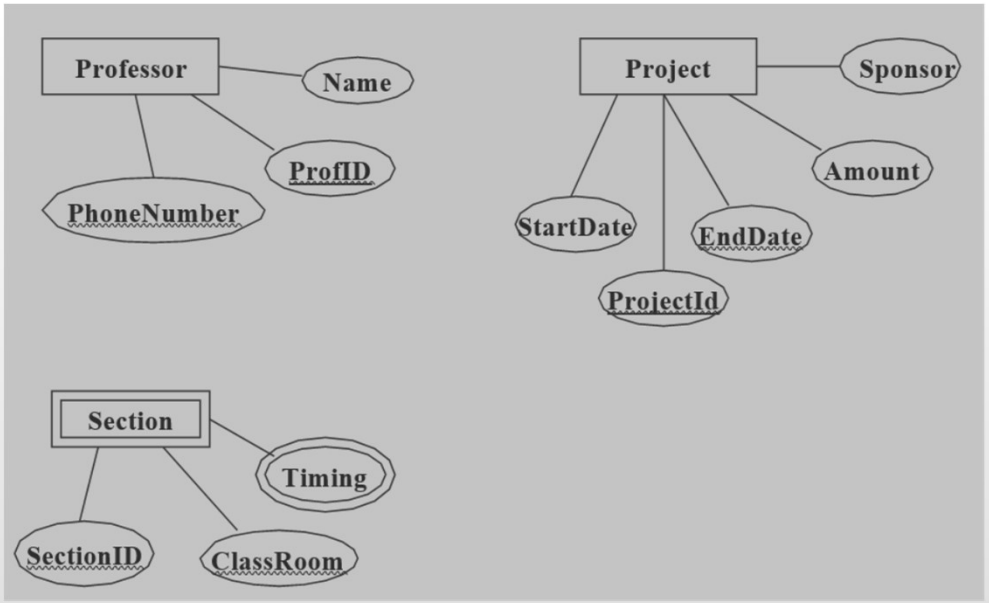
ENTITIES - STUDENT



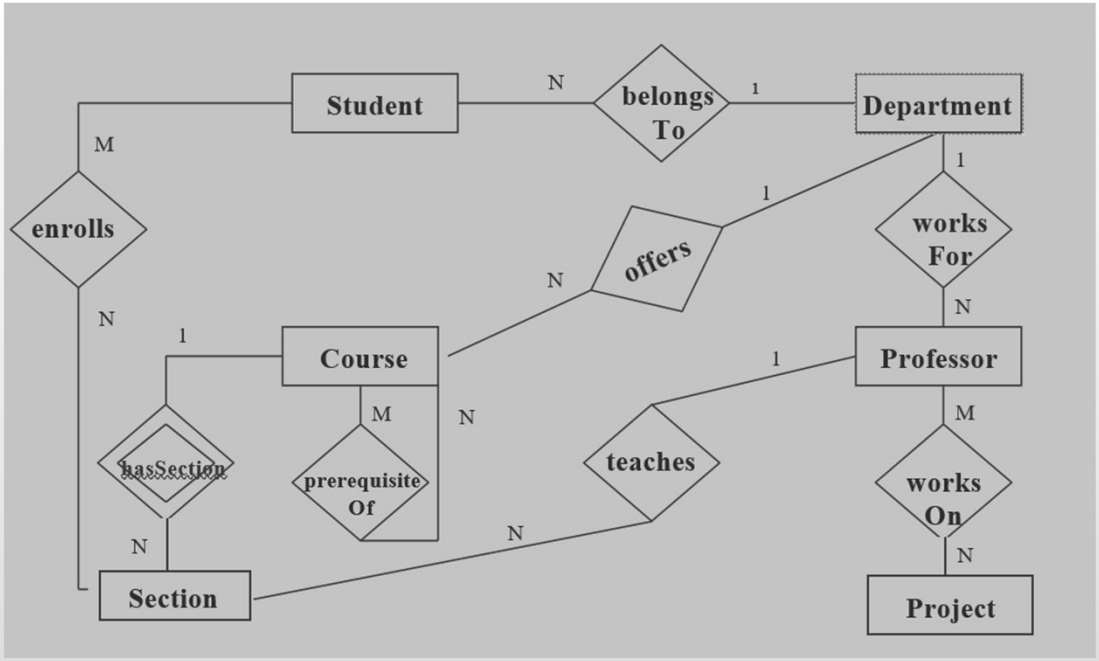
ENTITIES – DEPARTMENT AND COURSE



ENTITIES – PROFESSOR, PROJECT AND SECTIONS



E/R DIAGRAM SHOWING RELATIONSHIPS



DESIGN CHOICES: ATTRIBUTE VERSUS RELATIONSHIP

- **Should *offering department* be an attribute of a course or should we create a relationship between Course and Dept entities called, say, *offers* ?**

DESIGN CHOICES: ATTRIBUTE VERSUS RELATIONSHIP

- **Should *offering department* be an attribute of a course or should we create a relationship between Course and Dept entities called, say, *offers* ?**
  - Later approach is preferable when the necessary entity, in this case the Department, already exists.

DESIGN CHOICES: ATTRIBUTE VERSUS RELATIONSHIP

- **Should *class room* be an attribute of Section or should we create an entity called ClassRoom and have a relationship, say, meetsIn, connecting Section and ClassRoom?**

DESIGN CHOICES: ATTRIBUTE VERSUS RELATIONSHIP

- **Should *class room* be an attribute of Section or should we create an entity called ClassRoom and have a relationship, say, meetsIn, connecting Section and ClassRoom?**
  - In this case, the option of making classRoom as an attribute of Section is better as we do not want to give a lot of importance to class room and make it a an *entity*.

DESIGN CHOICES

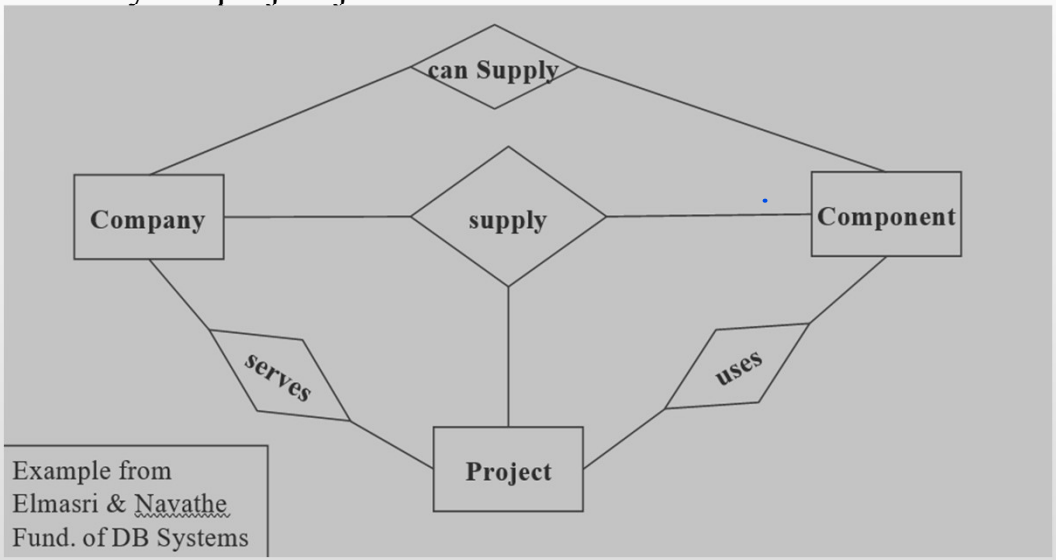
- Weak entity versus composite multi-valued attributes
  - **Note that a *section* could also be modeled as a composite multi- valued *attribute* of Course entity.**

DESIGN CHOICES

- Weak entity versus composite multi-valued attributes
- **Note that a *section* could also be modeled as a composite multi- valued *attribute* of Course entity.**
  - However, if so, *section* can not participate in relationships, such as, *enrolls* with Student entity.
- In general, if a thing, even though not of independent existence, participates in other relationships on its own, it is best captured as a *weak entity.*
  - If the above is not the case, composite multi-valued attribute may be enough.

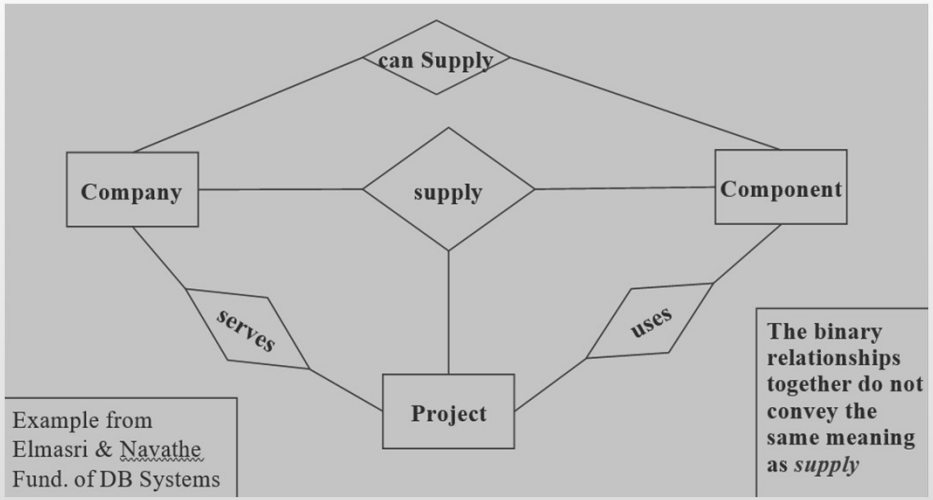
TERNARY RELATIONSHIPS

Relationship instance (c, p, j) in *supply* indicates that company c supplies a component p that is made use of by the project j



TERNARY RELATIONSHIP

- (c,p) in canSupply, (j,p) in uses, (c,j) in serves may not together imply (c,p,j) is in supply. Whereas the other way round is of course true.



RELATIONAL DATA MODEL
&
NOTION OF KEYS

RELATIONAL DATA MODEL

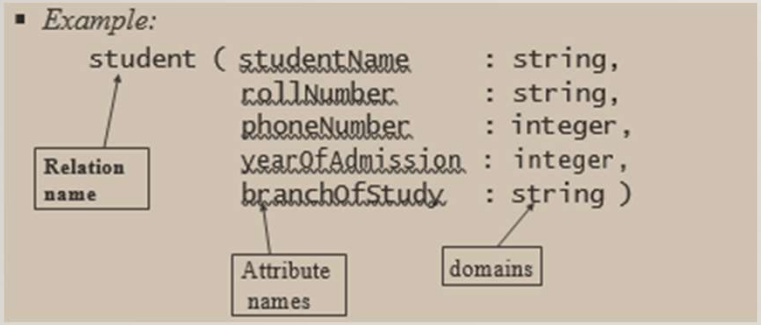
- Introduction
- **Proposed by Edgar F Codd (1923-2003) in the early seventies [Turing Award – 1981]**
- Most of the modern DBMS use the *relational* data model.
- Simple and elegant model with a mathematical basis.

Led to the development of a theory of data dependencies and database design.

- Relational algebra operations –
- crucial role in query optimization and execution.
- Laid the foundation for the development of
  - Tuple relational calculus and then
  - Database standard SQL

RELATION SCHEME

- Consists of relation name, and a set of attributes or field names or column names. Each attribute has an associated domain.
- *Example:*



- *Domain* – set of *atomic* (or *indivisible*) values – data type

Relation Instance

- A finite *set of tuples* constitute a relation instance.
- A tuple of the relation with scheme $R = (A_1, A_2, \ldots, A_m)$ is an ordered sequence of values $(v_1, v_2, \ldots, v_m)$ such that $v_i \in$ domain (A_i), $1 \leq i \leq m$

student

studentName	rollNumber	yearOf Admission	phoneNumber	branch Of Study
Ravi Teja	CS05B015	2005	9840110489	CS
Rajesh	CS04B125	2004	9840110490	CS

- No duplicate tuples (or rows) in a relation instance.
- We shall later see that in SQL, duplicate rows would be allowed in tables.

ATTRIBUTES

- Each entity is described by a set of attributes/properties that have associated values
- student entity
  - StudName – name of the student.
  - RollNumber – the roll number of the student.
  - Sex – the gender of the student etc.
- All entities in an Entity set/type have the same set of attributes. Chosen set of attributes – amount of detail in modeling.

ANOTHER RELATION EXAMPLE

- enrollment (studentName, rollNo, courseNo, sectionNo)

Student Name	rollNumber	courseNo	Section No
Rajesh	CS04B125	CS3200	2
Rajesh	CS04B135	CS3700	1
Suresh	CS04B130	CS3200	2

KEYS FOR A RELATION (1/2)

- **Key**: A set of attributes K, whose values uniquely identify a tuple in any instance. And none of the proper subsets of K has this property
- Example: {*rollNumber*} is a key for *student* relation.
- {*rollNumber, name*} – values can uniquely identify a tuple
  - but the set is not *minimal*
  - not a Key
- A key can not be determined from any particular instance data
  - it is an intrinsic property of a scheme
  - it can only be determined from the meaning of attributes

KEYS FOR A RELATION (2/2)

- A relation can have more than one key.
- Each of the keys is called a *candidate* key
  - Example: *book* (*isbnNo, authorName, title, publisher, year*) (Assumption : books have only one author)
  - *Keys*: {*isbnNo*}, {*authorName, title*}
- A relation has at least one key
- - the set of all attributes, in case no proper subset is a key.
- **Superkey**: A set of attributes that contains a key as a subset.
  - A key can also be defined as a *minimal superkey*
- **Primary Key**: One of the candidate keys chosen for indexing purposes (More details later…)

RELATIONAL DATABASE SCHEME AND INSTANCE

- **Relational database scheme**: *D* consist of a finite no. of relation schemes and a set *I* of integrity constraints.
- **Integrity constraints**: Necessary conditions to be satisfied by the data values in the relational instances so that the set of data values constitute a meaningful database
  - domain constraints
  - key constraints
  - referential integrity constraints
- **Database instance**: Collection of relational instances satisfying the integrity constraints.

DOMAIN AND KEY CONSTRAINTS

- **Domain Constraints**: Attributes have associated domains
- *Domain* – set of atomic data values of a specific type.
- *Constraint* – stipulates that the actual values of an attribute in any tuple must belong to the declared domain.
- **Key Constraint**: Relation scheme – associated keys Constraint – if K is supposed to be a key for scheme R,
  - *any* relation instance r on R should not have two tuples that have identical values for attributes in K.
  - Also, none of the key attributes can have null value.

FOREIGN KEYS

- Tuples in one relation, say $r_1(R_1)$, often need to refer to tuples in another relation, say r2(R2)
  - to capture relationships between entities
- Primary Key of R2 : $K = \{B1, B2, \ldots, Bj\}$
- A set of attributes $F = \{A1, A2, \ldots, Aj\}$ of R1 such that $dom(Ai) = dom(Bi)$, $1 \leq i \leq j$ and
  - whose values are used to refer to tuples in r_2
  - is called a *foreign key* in R1 *referring* to R2.

- R1, R2 can be the same scheme also.
- There can be more than one foreign key in a relation scheme

FOREIGN KEY – EXAMPLES (1/2)

Foreign key attribute *deptNo* of *course* relation refers to

Primary key attribute *deptID* of *department* relation

Course

courseId	name	credits	deptNo
CS635	ALGORITHMS	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	AERO DYNAMICS	3	3

Department

deptId	name	hod	phone
1	COMPUTER SCIENCE	CS01	22576235
2	ELECTRICAL ENGG	ES01	22576234
3	MECHANICAL ENGG	ME01	22576233

FOREIGN KEY – EXAMPLES(2/2)

- It is possible for a foreign key in a relation

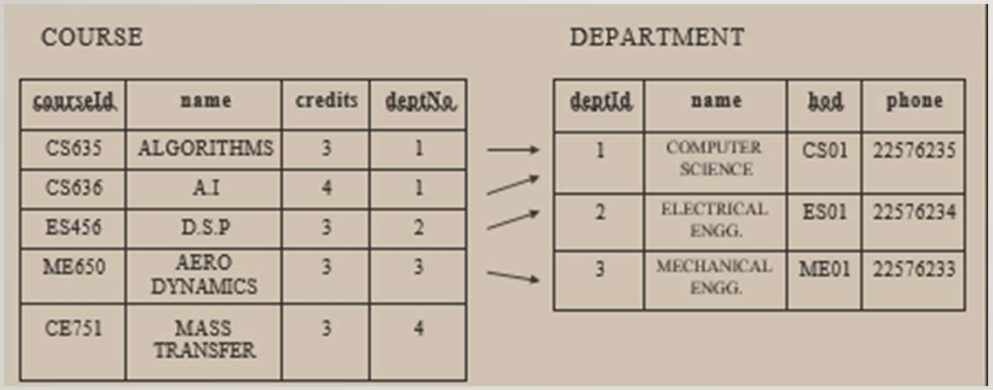
  to refer to the primary key of the relation itself An Example:

- univEmployee (empNo, name, sex, salary, dept, reportsTo)
- *reportsTo* is a foreign key referring to *empNo* of the same relation
-
  - Every employee in the university reports to some other employee for administrative purposes
    - - except the *vice-chancellor*, of course!

REFERENTIAL INTEGRITY CONSTRAINT (RIC)

- Let *F* be a foreign key in scheme R_1 referring to scheme R_2 and let *K* be the primary key of R_2.
- **RIC**: any relational instances r1on R1 and r2 on R2 must be s.t for any tuple *t* in r_1, either its *F*-attribute values are all *null* or they are identical to the *K*-attribute values of *some* tuple in r_2.
- RIC ensures that references to tuples in r_2 are for *currently existing* tuples.
  - That is, there are no *dangling* references.

REFERENTIAL INTEGRITY CONSTRAINT (RIC) - EXAMPLE



courseId	name	credits	deptNo
CS635	ALGORITHMS	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	AERO DYNAMICS	3	3
CE751	MASS TRANSFER	3	4

deptId	name	hod	phone
1	COMPUTER SCIENCE	CS01	22576235
2	ELECTRICAL ENGG.	ES01	22576234
3	MECHANICAL ENGG.	ME01	22576233

The new course refers to a non-existent department and thus violates the RIC

EXAMPLE RELATIONAL SCHEME

- student (rollNo, name, degree, year, sex, deptNo, advisor)
  - *degree* is the program (B Tech, M Tech, M S, Ph D etc) for which the student has joined.
  - *year* is the year of admission and
  - *advisor* is the EmpId of a faculty member identified as the student's advisor.
-
- department (deptId, name, hod, phone)
  - *phone* is that of the department's office.
-
- professor (empId, name, sex, startYear, deptNo, phone)
- *startYear* is the year when the faculty member has joined the department *deptNo*.

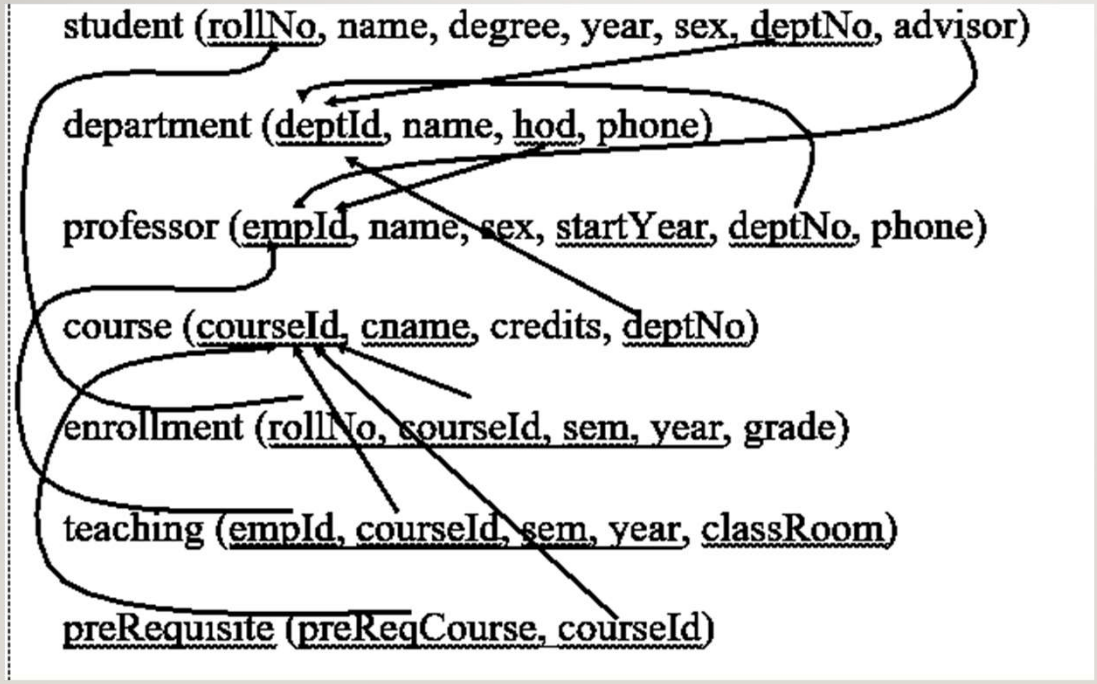
EXAMPLE RELATIONAL SCHEME

- course (courseId, cname, credits, deptNo)
  - *deptNo* indicates the department that offers the course.
- enrollment (rollNo, courseId, sem, year, grade)
  - *sem* can be either "odd" or "even" indicating the two semesters of an academic year.
  - The value of *grade* will be null for the current semester and non-null for past semesters.
- teaching (empId, courseId, sem, year, classRoom) preRequisite (preReqCourse, courseID)
  - Here, if (c1, c2) is a tuple, it indicates that c1 should be successfully completed before enrolling for c2.

EXAMPLE RELATIONAL SCHEME

- student (rollNo, name, degree, year, sex, deptNo, advisor)
- department (deptId, name, hod, phone)
- professor (empId, name, sex, startYear, deptNo, phone)
- course (courseId, cname, credits, deptNo)
- enrollment (rollNo, courseId, sem, year, grade)
- teaching (empId, courseId, sem, year, classRoom)
- preRequisite (preReqCourse, courseId)

EXAMPLE RELATIONAL SCHEME WITH RICS SHOWN



RELATIONAL ALGEBRA

- A set of operators (unary and binary) that take relation instances as arguments and return new relations.
- Gives a procedural method of specifying a retrieval query.
- Forms the core component of a relational query engine.
- SQL queries are internally translated into RA expressions.
- Provides a framework for query optimization.

- **RA operations**: *select* (σ), *project* (π), *cross product* ($'$),

  union ($\cup$), *intersection* ($\cap$), *difference* ($-$), *join* ($\bowtie$)

THE *SELECT* OPERATOR

- Unary operator.
- can be used to *select* those tuples of a relation that satisfy a given condition.
- *Notation*: $\sigma_{\theta}(r)$
  - σ : select operator (read as *sigma*)
  - θ : selection condition
  - r : relation name
- Result: a relation with the same schema as r consisting of the tuples in r that satisfy condition θ
- Select operation is commutative:

$$\sigma_{c1}(\sigma_{c2}(r)) = \sigma_{c2}(\sigma_{c1}(r))$$

SELECTION CONDITION

- *Select condition*:
  Basic condition or Composite condition
- *Basic condition*:
  Either A_i <compOp> A_j or A_i <compOp> c
- *Composite condition*:
  Basic conditions combined with logical operators AND, OR and NOT appropriately.
- Notation:

  <compOp> : one of $<, \leq, >, \geq, =, \neq$

  A_i, A_j : attributes in the scheme *R* of *r*

  c : constant of appropriate data type

EXAMPLES OF *SELECT* EXPRESSIONS

1. Obtain information about a professor with name "Giridhar"

$$\sigma_{\text{name} = \text{"Giridhar"}} (\text{professor})$$

2. Obtain information about professors who joined the university between 1980 and 1985, both inclusive

$$\sigma_{\text{startYear} \geq 1980 \wedge \text{startYear} \leq 1985} (\text{professor})$$

THE *PROJECT* OPERATOR

- Unary operator.
- Can be used to keep only the required attributes of a relation instance and throw away others.
- *Notation*: $\pi_{A_1 A_2 \ldots, A_k}(r)$ where $A_1, A_2, \ldots, A_k$ is a list L of desired attributes in the scheme of r.
- Result = $\{ (v_1, v_2, \ldots, v_k) \mid v_i \in \underline{dom}(A_i)$, $1 \leq \underline{i} \leq k$ and there is some tuple t in r *s.t* $\underline{t.A_1} = v_1, \underline{t.A_2} = v_2, \ldots, \underline{t.A_k} = v_k\}$
- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L

EXAMPLES OF *PROJECT* EXPRESSIONS

student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS03S001	Rajesh	M.S	2003	M	1	CS02
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	ES01
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

$\pi_{\text{rollNo, name}}$ (student)

rollNo	name
CS04S001	Mahesh
CS03S001	Rajesh
CS04M002	Piyush
ES04M001	Deepak
ME04M001	Lalitha
ME03M002	Mahesh

$\pi_{\text{name}} (\sigma_{\text{degree} = \text{"M.S"}} (\text{student}))$

name
Mahesh
Rajesh

Note: Mahesh is displayed only once because project operation results in a set.

SIZE OF *PROJECT* EXPRESSION RESULT

- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L
- What about the number of tuples in r_1?
- Two cases arise:
  - Projection List L contains some key of r_2
    - Then $|r_1| = |r_2|$
  - Projection List L does not contain any key of r_2
    - Then $|r_1| \leq |r_2|$

SET OPERATORS ON RELATIONS

- As relations are sets of tuples, set operations are applicable to them; but not in all cases.
- **Union Compatibility** : Consider two schemes R_1, R_2 where $R_1 = (A_1, A_2, \ldots, A_k)$; $R_2 = (B_1, B_2, \ldots, B_m)$
- R_1 and R_2 are called *union-compatible* if
  - $k = m$ and
  - $\underline{dom}(A_i) = \underline{dom}(B_i)$ for $1 \leq \underline{i} \leq k$
- **Set operations** – *union, intersection, difference*
  - Applicable to two relations if their schemes are union-compatible
- If $r_3 = r_1 \cup r_2$, scheme of r_3 is R_1 (as a convention)

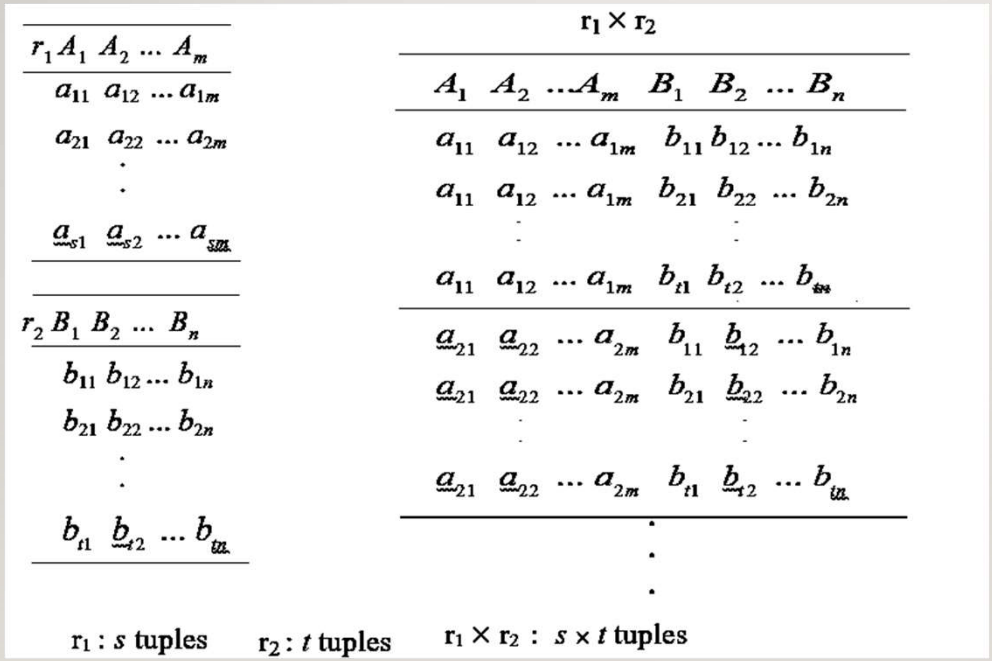
SET OPERATIONS

r_1 - relation with scheme R_1
r_2 - relation with scheme R_2 - union compatible with R_1

$r_1 \cup r_2 = \{t \mid t \in r_1 \text{ or } t \in r_2\}$
$r_1 \cap r_2 = \{t \mid t \in r_1 \text{ and } t \in r_2\}$
$r_1 - r_2 = \{t \mid t \in r_1 \text{ and } t \notin r_2\}$

By convention, in all the cases, the scheme of the result is that of the first operand i.e r_1.

CROSS PRODUCT OPERATION



EXAMPLE QUERY USING CROSS PRODUCT

Obtain the list of professors (Id and Name) along with the *name* of their respective departments
- Info is present in two relations – professor, department

Schema

- profDetail (eId, pname, deptno) ← $\pi_{\text{empId, name, deptNo}}$ (professor)
- deptDetail (dId, dname) ← $\pi_{\text{deptId, name}}$ (department)
- profDept ← profDetail × deptDetail
- desiredProfDept ← $\sigma_{\text{deptno = dId}}$ (profDept)
- result ← $\pi_{\text{eId, pname, dname}}$ (desiredProfDept)

QUERY USING *CROSS PRODUCT* – USE OF RENAMING

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- profDetail (eId, pname, deptno) ← $\pi_{\text{empId, name, deptNo}}$ (professor)
  - this is a temporary relation to hold the intermediate result
  - "empId, name, deptNo" are being renamed as "eId, pname, deptno"
  - creating such relations helps us understand/formulate the query
  - we use "←" to indicate assignment operation.
- deptDetail (dId, dname) ← $\pi_{\text{deptId, name}}$ (department)
  - another temporary relation
- Renaming is necessary to ensure that the cross product has distinct attribute names.

USE OF RENAMING OPERATOR ρ

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- One can use the rename operator ρ and write the whole query as one big expression (as an alternative to using temporary relations)

$$\pi_{eId,\ pname,\ dname}\Big(\sigma_{deptno = dId}\big(\rho_{eId,\ pname,\ deptno}(\pi_{empId,\ name,\ deptNo}(professor)) \times \rho_{dId,\ dname}(\pi_{deptId,\ name}(department))\big)\Big)$$

- It is easier to understand and formulate the query with *meaningfully named* temporary relations as shown earlier.
- Students are encouraged to use temporary relations.

JOIN OPERATION

- ***Cross product*** : produces all combinations of tuples
  - often only certain combinations are meaningful
  - cross product is usually followed by selection

- ***Join*** : combines tuples from two relations provided they satisfy a specified condition (join condition)
  - equivalent to performing *cross product* followed by *selection*
  - a very useful operation

- Depending on the type of condition we have
  - *theta join*
  - *equi join*

THETA JOIN

- Let r_1 - relation with scheme $R_1 = (A_1, A_2, \ldots, A_m)$
  r_2 - relation with scheme $R_2 = (B_1, B_2, \ldots, B_n)$
  where w.l.o.g we assume $R_1 \cap R_2 = \phi$
- Notation for join expression : $r = r_1 \bowtie_\theta r_2$
- θ − the join condition - is of the form : $C_1 \wedge C_2 \wedge \ldots \wedge C_s$
  C_i is of the form : A_j <CompOp> B_k
  where <CompOp> is one of $\{ =, \neq, <, \leq, >, \geq \}$
- Scheme of the result relation r is:
  $(A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n)$

$$r = \{(a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n) \mid (a_1, a_2, \ldots, a_m) \in r_1, (b_1, b_2, \ldots, b_n) \in r_2 \text{ and } (a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n) \text{ satisfies } \theta\}$$

EXAMPLES..

Professor

empId	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

For each department, find its name and the name, sex and phone number of the head of the department

Department

deptId	name	hod	phone
1	Computer Science	CS01	22576235
2	Electrical Engg.	ES01	22576234
3	Mechanical Engg.	ME01	22576233

Courses

courseId	cname	credits	deptNo
CS635	Algorithms	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	Aero Dynamics	3	3

EXAMPLE

For each department, find its name and the name, sex and phone number of the head of the department.

prof (empId, p-name, sex, deptNo, prof-phone)

$$\leftarrow \pi_{\text{empId, name, sex, deptNo, phone}} (\text{professor})$$

$$\text{result} \leftarrow \pi_{\text{deptId, name, hod, p-name, sex, prof-phone}} (\text{department} \bowtie_{(\text{hod} = \text{empId})} \text{prof})$$

deptId	name	hod	p-name	sex	prof-phone
1	Computer Science	CS01	Giridher	M	22576235
2	Electrical Engg.	EE01	Rajiv Guptha	M	22576234
3	Mechanical Engg.	ME01	Tahir Nayyar	M	22576233

EQUI-JOIN AND NATURAL JOIN

- *Equi-join* : Equality is the only comparison operator used in the join condition
- *Natural join* : R_1, R_2 - have common attributes, say X_1, X_2, X_3
  - Join condition:

    $(R_1.X_1 = R_2.X_1) \wedge (R_1.X_2 = R_2.X_2) \wedge (R_1.X_3 = R_2.X_3)$
    - Values of common attributes should be equal
  - Schema for the result $Q = R_1 \cup (R_2 - \{X_1, X_2, X_3\})$
    - Only one copy of the common attributes is kept

- Notation for natural join : $r = r_1 * r_2$

EXAMPLE – EQUI-JOIN

Find courses offered by each department

$$\pi_{\underline{\text{deptId}},\ \text{name},\ \underline{\text{courseId}},\ \underline{\text{cname}},\ \text{credits}} \left(\text{Department} \bowtie_{(\underline{\text{deptId}} = \underline{\text{deptNo}})} \text{Courses} \right)$$

Example – Equi-join

deptId	name	courseId	cname	credits
1	Computer Science	CS635	Algorithms	3
1	Computer Science	CS636	A.I	4
2	Electrical Engg.	ES456	D.S.P	3
3	Mechanical Engg.	ME650	Aero Dynamics	3

NATURAL JOIN

Teaching

empId	courseId	sem	year	classRoom
CS01	CS635	1	2005	BSB361
CS02	CS636	1	2005	BSB632
ES01	ES456	2	2004	ESB650
ME01	ME650	1	2004	MSB331

To find the courses handled by each professor

Professor * Teaching

result

empId	name	sex	startYear	deptNo	phone	courseId	sem	year	classRoom
CS01	Giridhar	M	1984	1	22576345	CS635	1	2005	BSB361
CS02	Keshav Murthy	M	1989	1	22576346	CS636	1	2005	BSB632
ES01	Rajiv Guptha	M	1989	2	22576244	ES456	2	2004	ESB650
ME01	Tahir Nayyar	M	1999	3	22576243	ME650	1	2004	MSB331

DIVISION OPERATOR

- The necessary condition to determine $r \div s$ on instances r(R) and s(S) is $S \subset R$
- The relation $r \div s$ is a relation on schema $R - S$.

  A tuple t is in $r \div s$ if and only if

  1) t is in $\pi_{R-S}(r)$
  2) For every tuple t_s in s, there is t_r in r satisfying both
     a) $t_r[S] = t_s$
     b) $t_r[R - S] = t$

  // $t_r[S]$ – the sub-tuple of t_r consisting of values of attributes in S

- Another Definition $r = r_1 \div r_2$

  Division operator produces a relation R(X) that includes all tuples t[X] that appear in r_1 in combination with every tuple from r_2 where $R_1 = Z$ and $R_2 = Y$ and $Z = X \cup Y$

EXAMPLES..

R = (A, B, C, D), S = (A, B), X = (C, D)

x = r ÷ s

s

A	B
a_1	b_1
a_2	b_2

r

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_3	d_3
a_2	b_2	c_3	d_3

x

C	D
c_1	d_1
c_3	d_3

(c_2, d_2) is not present in the result of division as it does not appear in combination with all the tuples of s in r

QUERY USING DIVISION OPERATION

Find those students who have enrolled for *all* courses offered in the dept of Computer Science.

Step1: Get the course enrollment information for all students

$$studEnroll \leftarrow \pi_{rollNo,\ name,\ courseId}\ (student * enrollment)$$

Step2: Get the course Ids of all courses offered by CS dept

$$csCourse \leftarrow \pi_{courseId}(\sigma_{dname\ =\ \text{“Computer Science”}}(courses \bowtie_{deptId\ =\ deptNo} dept))$$

Result : $studEnroll \div csCourse$

```
-- Find students enrolled in all courses offered by the CSE department
SELECT student_id
FROM enrollment e
WHERE course_id IN (
    SELECT course_id
    FROM courses
    WHERE department = 'CSE'
)
GROUP BY student_id
HAVING COUNT(DISTINCT course_id) = (
    SELECT COUNT(DISTINCT course_id)
    FROM courses
    WHERE department = 'CSE'
);
```

EXAMPLES..

Suppose result of step 1
(we skip roll number for simplicity)

studEnroll

name	courseId
Mahesh	CS635
Mahesh	CS636
Rajesh	CS635
Pivush	CS636
Pivush	CS635
Deepak	ES456
Lalitha	ME650
Mahesh	ME650

result of step 2

csCourse

courseId
CS635
CS636

Let' s assume for a moment that student names are unique!

studEnroll ÷ csCourse

result

name
Mahesh
Pivush

COMPLETE SET OF OPERATORS

- Are all Relational Algebra operators essential ?

  Some operators can be realized through other operators

- What is the minimal set of operators ?
  - The operators $\{\sigma , \pi , \times , \cup, - \}$ constitute a *complete* set of operators
  - Necessary and sufficient set of operators.
  - Intersection – union and difference
  - Join – cross product followed by selection
  - Division – project, cross product and difference

- Retrieve the list of female PhD students
- Obtain the name and rollNo of all female BTech students
- Obtain the rollNo of students who never obtained an ‘E’ grade

Retrieve the list of female PhD students

$\sigma_{\text{degree} = \text{'phD'} \wedge \text{sex} = \text{'F'}}$ (student)

Obtain the name and rollNo of all female BTech students

$\pi_{\text{rollNo, name}} (\sigma_{\text{degree} = \text{'BTech'} \wedge \text{sex} = \text{'F'}}$ (student))

[redacted]

$\pi_{\text{rollNo}} (\sigma_{\text{grade} \neq \text{'E'}}$ (enrollment))

EXAMPLE QUERIES

Retrieve the list of female PhD students

$\sigma_{\text{degree} = \text{'phD'} \wedge \text{sex} = \text{'F'}}$ (student)

Obtain the name and rollNo of all female BTech students

$\pi_{\text{rollNo, name}} (\sigma_{\text{degree} = \text{'BTech'} \wedge \text{sex} = \text{'F'}}$ (student))

Obtain the rollNo of students who never obtained an 'E' grade

$\pi_{\text{rollNo}} (\sigma_{\text{grade} \neq \text{'E'}}$ (enrollment))

is incorrect!!

(what if some student gets E in one course and A in another?)

π_{rollNo} (student) $- \pi_{\text{rollNo}} (\sigma_{\text{grade} = \text{'E'}}$ (enrollment))

EXAMPLES…

Obtain the department Ids for departments with no lady professor

$$\pi_{\text{deptId}}(\text{dept}) - \pi_{\text{deptId}}(\sigma_{\text{sex} = \text{'F'}}(\text{professor}))$$

Obtain the rollNo of male students who have obtained at least one S grade

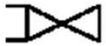
$$\pi_{\text{rollNo}}(\sigma_{\text{sex} = \text{'M'}}(\text{student})) \cap \pi_{\text{rollNo}}(\sigma_{\text{grade} = \text{'S'}}(\text{enrollment}))$$

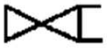
OUTER JOIN OPERATION (1/2)

- Theta join, equi-join, natural join are all called *inner joins* . The result of these operations contain only the matching tuples

- The set of operations called *outer joins* are used when all tuples in relation *r* or relation *s* or both in *r* and *s* have to be in result.

There are 3 kinds of outer joins:

Left outer join 

Right outer join 

Full outer join 

OUTER JOIN OPERATION (2/2)

Left outer join: $r \mathbin{⟕} s$

It keeps all tuples in the first, or left relation r in the result. For some tuple t in r, if no matching tuple is found in s then S-attributes of t are made null in the result.

Right outer join: $r \mathbin{⟖} s$

Same as above but tuples in the second relation are all kept in the result. If necessary, R-attributes are made null.

Full outer join: $r \mathbin{⟗} s$

All the tuples in both the relations r and s are in the result.

Instance Data for Examples

Student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS05S001	Anurish	M.S	2003	M	1	null
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	null
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

Professor

empId	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

LEFT OUTER JOIN

$$temp \leftarrow (student \ ⟕_{advisor = empId} \ professor)$$

$$\rho_{rollNo,\ name,\ advisor} (\pi_{rollNo,\ student.name,\ professor.name} (temp))$$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS05S001	Amrish	Null
CS04M002	Piyush	Giridhar
ES04M001	Deepak	Null
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

RIGHT OUTER JOIN

$$temp \leftarrow (student \bowtie_{advisor = empId} professor)$$

$$\rho_{rollNo,\ name,\ advisor} (\pi_{rollNo,\ student.name,\ professor.name} (temp))$$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Piyush	Giridhar
null	null	Keshav Murthy
null	null	Rajiv Guptha
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

FULL OUTER JOIN

$$temp \leftarrow (student \bowtie_{advisor = empId} professor)$$

$$\rho_{rollNo,\ name,\ advisor}\ (\pi_{rollNo,\ student\_name,\ professor\_name}\ (temp))$$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Piyush	Giridhar
CS05S001	Amrish	Null
null	null	Keshav Murthy
ES04M001	Deepak	Null
null	null	Rajiv Guptha
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

- Obtain the names, roll numbers of students who have got S grade in the CS3700 course offered in 2017 odd semester along with his/her advisor name.

```
SELECT
    s.name AS student_name,
    s.roll_number,
    a.name AS advisor_name
FROM
    students s
JOIN grades g ON s.roll_number = g.roll_number
JOIN courses c ON g.course_id = c.course_id
JOIN advisors a ON s.advisor_id = a.advisor_id
WHERE
    g.grade = 'S'
    AND c.course_id = 'CS3700'
    AND c.year = 2017
    AND c.semester = 'odd';
```

THE SQL STANDARD

SQL – STRUCTURED QUERY LANGUAGE

- An international standard (ANSI, ISO) that specifies how
  - a relational schema is created
  - data is inserted / updated in the relations
  - data is queried
  - transactions are started and stopped
  - programs access data in the relations
  - and a host of other things are done

  Every relational database management system (RDBMS) is required to support / implement the SQL standard.

  • RDBMS vendors may give additional features

  • Downside of using vendor-specific features - portability

SEQUEL

- developed by IBM in early 70's
- relational query language as part of System-R project at
- IBM San Jose Research Lab.
- the earliest version of SQL
- SQL evolution
  - SQL- 86/89
  - SQL- 92 - SQL2
  - SQL- 99/03 - SQL3 (includes object relational features) And the evolution continues .....
- Disclaimer: This module covers only important principles of SQL

COMPONENTS OF SQL STANDARD(1/2)

Data Definition Language (DDL)

- Specifies constructs for schema definition, relation definition, integrity constraints, views and schema modification.

Data Manipulation Language (DML)

- Specifies constructs for inserting, updating and querying the data in the relational instances (or tables).

Embedded SQL and Dynamic SQL

- Specifies how SQL commands can be embedded in a high-level host language such as C, C++ or Java for programmatic access to the data.

COMPONENTS OF SQL STANDARD(2/2)

- *Transaction Control*
- Specifies how transactions can be started / stopped, how a set of concurrently executing transactions can be managed.

- *Authorization*
- Specifies how to restrict a user / set of users to access only certain parts of data, perform only certain types of queries etc.

DATA DEFINITION IN SQL

Data Definition in SQL

Defining the schema of a relation

create table *r* (attributeDefinition-1, attributeDefinition-2,…,

name of the relation

attributeDefinition-n, [integrityConstraints-1],

[integrityConstraints-2]…, [integrityConstraints-m])

Attribute Definition –

attribute-name domain-type [NOT NULL] [DEFAULT v]

E.g.:

create table *example1* (A char(6) not null default "000000",

B int, C char (1) default "F");

DOMAIN TYPES IN SQL-92 (1/2)

Domain Types in SQL-92 (1/2)

- *Numeric data types*
  - integers of various sizes – INT, SMALLINT
  - real numbers of various precision – REAL, FLOAT, DOUBLE PRECISION
  - formatted numbers – DECIMAL (i, j) or NUMERIC (i, j)
    - i – total number of digits (precision)
    - j – number of digits after the decimal point (scale)
- *Character string data types*
  - fixed length – CHAR(n) – n: no. of characters
  - varying length – VARCHAR(n) – n: max.no. of characters
- *Bit string data types*
  - fixed length – BIT(n)
  - varying length – BIT VARYING(n)

DOMAIN TYPES IN SQL-92 (2/2)

Date data type

- DATE type has 10 position format – YYYY-MM-DD

Time data type

- TIME type has 8 position format – HH : MM : SS

Others

- There are several more data types whose details are available in SQL reference books

SPECIFYING INTEGRITY CONSTRAINTS IN SQL

Also called Table Constraints

Included in the definition of a table

Key constraints

PRIMARY KEY (A_1,A_2,…,A_k)

specifies that {A_1,A_2,…,A_k} is the primary key of the table

UNIQUE (B_1,B_2,…,B_k)

specifies that {B_1,B_2,…,B_k} is a candidate key for the table

There can be more than one UNIQUE constraint but only one PRIMARY KEY constraint for a table.

SPECIFYING REFERENTIAL INTEGRITY CONSTRAINTS

- FOREIGN KEY (A1) REFERENCES r2 (B1)
  - specifies that attribute A_1 of the table being defined, say r_1, is a
- *foreign key* referring to attribute B_1 of table r_2
-
  - recall that this means:
    - each value of column A_1 is either null or is one of the values appearing in column B1 of r2

SPECIFYING WHAT TO DO IF RIC VIOLATION OCCURS

- *RIC violation*
  - can occur if a referenced tuple is deleted or modified
  - action can be specified for each case using qualifiers ON DELETE or ON UPDATE
- *Actions*
  - three possibilities can be specified
    - SET NULL, SET DEFAULT, CASCADE
  - these are actions to be taken on the referencing tuple
  - SET NULL – foreign key attribute value to be set null
  - SET DEFAULT – foreign key attribute value to be set to its
    - default value
  - CASCADE – delete the referencing tuple if the referenced tuple is deleted or update the FK attribute if the referenced tuple is updated

TABLE DEFINITION EXAMPLE

```
create table students (
        rollNo char(8) not null,
        name varchar(15) not null,
        degree char(5),
        year  smallint,
        sex char not null,
        deptNo smallint,
        advisor char(6),
        primary key(rollNo),
        foreign key(deptNo) references
                                department(deptId)
                on delete set null on update cascade,
        foreign key(advisor) references
                                professor(empId)
                on delete set null on update cascade
 );
```

MODIFYING A DEFINED SCHEMA

- ALTER TABLE command can be used to modify a schema
- *Adding a new attribute*
- ALTER table student ADD address varchar(30);
- *Deleting an attribute*
  - need to specify what needs to be done about views or constraints that refer to the attribute being dropped
  - two possibilities
    - CASCADE – delete the views/constraints also RESTRICT – do not delete the attributes if there are some
      - views/constraints that refer to it.
  - ALTER TABLE student DROP degree RESTRICT Similarly, an entire table definition can be deleted

DATA MANIPULATION IN SQL

Basic query syntax

select $A_1, A_2, \ldots, A_m$ — a set of attributes from relations $R_1, \ldots, R_p$ that are required in the output table.

from $R_1, R_2, \ldots, R_p$ — the set of tables that contain the relevant tuples to answer the query.

where θ — a boolean predicate that specifies when a combined tuple of $R_1, \ldots, R_p$ contributes to the output.

Equivalent to:

$$\pi_{A_1, A_2, \ldots, A_n}(\sigma_\theta(R_1 \times R_2 \times \ldots \times R_p))$$

Assuming that each attribute name appears exactly once in the table.

MEANING OF THE BASIC QUERY BLOCK

The *cross-product M* of the tables in the from clause would be considered.

- Tuples in *M* that satisfy the condition θ are *selected*.
  - For each such tuple, values for the attributes A1,A2,….,Am (mentioned in the select clause) are *projected*.
- This is a conceptual description
  - - in practice more efficient methods are employed for evaluation.
- The word *select* in SQL should not be confused with select operation of relational algebra.

SQL QUERY RESULT

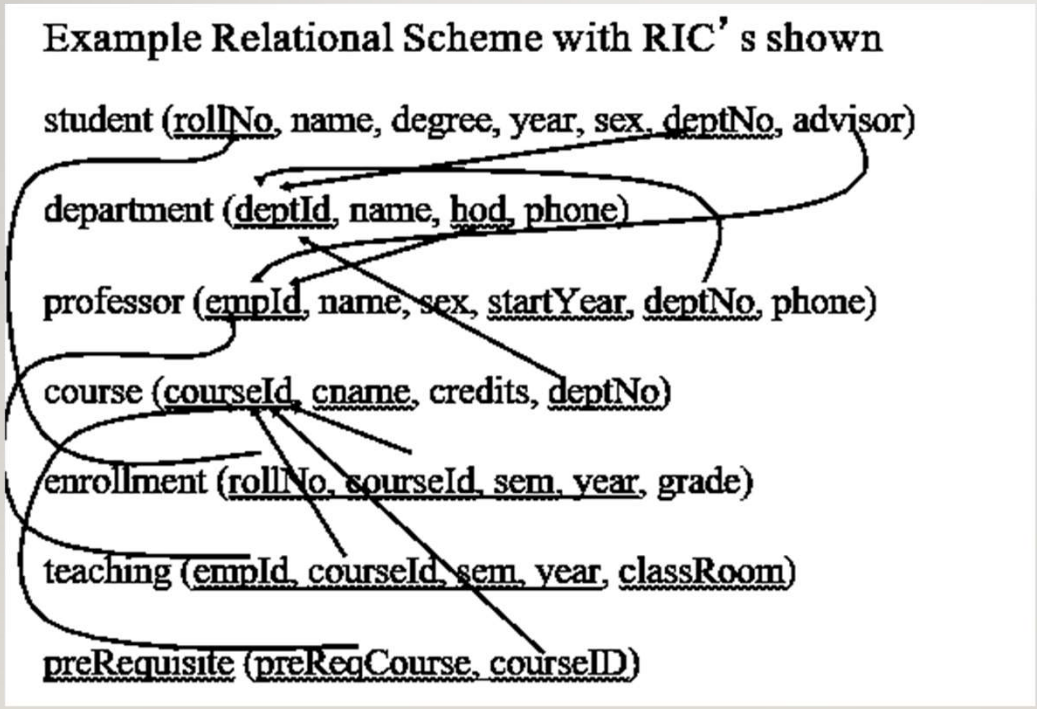
The result of any SQL query

- a table with *select* clause attributes as column names.
- duplicate rows may be present.
  - differs from the definition of a relation.
- duplicate rows can be eliminated by specifying DISTINCT keyword in the *select* clause, if necessary.

```
SELECT DISTINCT name
FROM student WHERE ...
```

- duplicate rows are essential while computing aggregate functions (average, sum etc).
- removing duplicate rows involves additional effort and is done only when necessary.

Example Relational Scheme with RIC's shown



EXAMPLE QUERIES INVOLVING A SINGLE TABLE

Get the rollNo, name of all women students in the dept no. 5.

```
select rollNo, name
from student
where sex = 'F' and deptNo = 5;
```

Get the employee Id, name and phone number of professors in the CS dept (deptNo = 3) who have joined after 1999.

```
select empId, name, phone
from professor
where deptNo = 3 and startYear > 1999;
```

EXAMPLES INVOLVING TWO OR MORE RELATIONS (1/2)

Get the rollNo, name of students in the CSE dept (deptNo = 3) along with their advisor's name and phone number.

```
select rollNo, s.name, f.name as advisorName,
                phone as advisorPhone
from student as s, professor as f
where s.advisor = f.empId and
          s.deptNo = 3;
```

attribute renaming in the output

table aliases are used to disambiguate the common attributes

table aliases are required if an attribute name appears in more than one table.
Also when *same* relation appears twice in the from clause.

EXAMPLES INVOLVING TWO OR MORE RELATIONS (2/2)

Get the names, employee ID's, phone numbers of professors in CSE dept who joined before 1995.

```
select empId, f.name, f.phone
from professor as f, department as d
where f.deptNo = d.deptId and
      d.name = 'CSE' and
      f.startYear < 1995
```

NESTED QUERIES OR SUBQUERIES

While dealing with certain complex queries

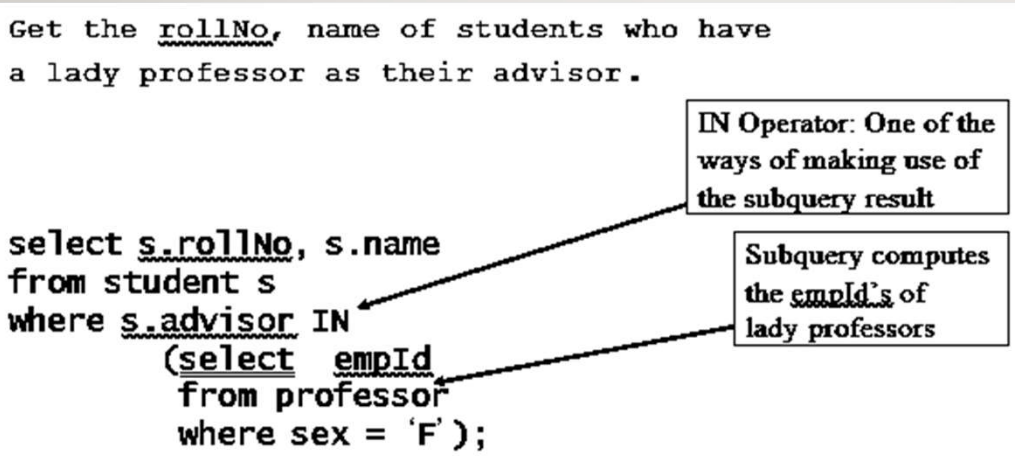
- beneficial to specify part of the computation as a separate query & make use of its result to formulate the main query.
- such queries – nested / subqueries.

Using subqueries

- makes the main query easy to understand / formulate
- sometimes makes it more efficient also
    - sub query result can be computed once and used many times.
        - not the case with all subqueries.

NESTED QUERY EXAMPLE

Get the rollNo, name of students who have a lady professor as their advisor.




```
select s.rollNo, s.name
from student s
where s.advisor IN
        (select  empId
         from professor
         where sex = 'F');
```

NOT IN can be used in the above query to get details of students who don't have a lady professor as their advisor.

SET COMPARISON OPERATORS

SQL supports several operators to deal with subquery results or in general with collection of tuples.

Combination of { =, <, ≤, ≥, >, <> } with keywords { ANY, ALL } can be used as set comparison operators.

Get the empId, name of the senior-most Professor(s):

```
select p.empId, p.name
from professors p
where p.startYear <= ALL ( select distinct startYear
                           from professor );
```

SEMANTICS OF SET COMPARISON OPERATORS

op is one of <, ≤, >, ≥, =, <>

- *v op* ANY *S*
  true if for some member *x* of *S*, *v* op *x* is true
  false if for no member *x* of *S*, *v* op *x* is true

S is a subquery

- *v op* ALL *S*
  true if for every member *x* of *S*, *v* op *x* is true
  false if for some member *x* of *S*, *v* op *x* is not true

- IN is equivalent to = ANY
  NOT IN is equivalent to
  <> ALL

- *v* is normally a single attribute, but while using IN or NOT IN it can be a tuple of attributes

CORRELATED AND UNCORRELATED NESTED QUERIES

If the nested query result is independent of the current tuple being examined in the outer query,
nested query is called *uncorrelated*,
otherwise, nested query is called *correlated*.

Uncorrelated nested query

- nested query needs to be computed only once.

Correlated nested query

- nested query needs to be re-computed for each row examined in the outer query.

EXAMPLE OF A CORRELATED SUBQUERY

Get the roll number and name of students whose gender is same as their advisor's.

```
select s.rollNo, s.name
from student s
where s.sex = ALL ( select f.sex
                    from professor f
                    where f.empId = s.advisor );
```

THE *EXISTS* OPERATOR

Using *EXISTS*, we can check if a subquery result is non-empty

EXISTS(S) is *true* if *S* has at least one tuple / member
is *false* if *S* contain no tuples

```
Get the employee Id and name of professors
who advise at least one women student.

select f.empId, f.name
from professors f
where EXISTS ( select s.rollNo
               from student s
               where s.advisor = f.empId and
                     s.sex = 'F' );
```

a correlated subquery

THE NOT EXISTS OPERATOR

Obtain the department Id and name of departments that do not offer any 4 credit courses.

a correlated subquery

```
select d.deptId, d.name
from department d
where NOT EXISTS ( select courseId
                   from course c
                   where c.deptNo = d.deptId and
                         c.credits = 4 );
```

Queries with *existentially* quantified predicates can be easily specified using *EXISTS* operator.

Queries with *universally* quantified predicates can only be specified after translating them to use *existential* quantifiers.