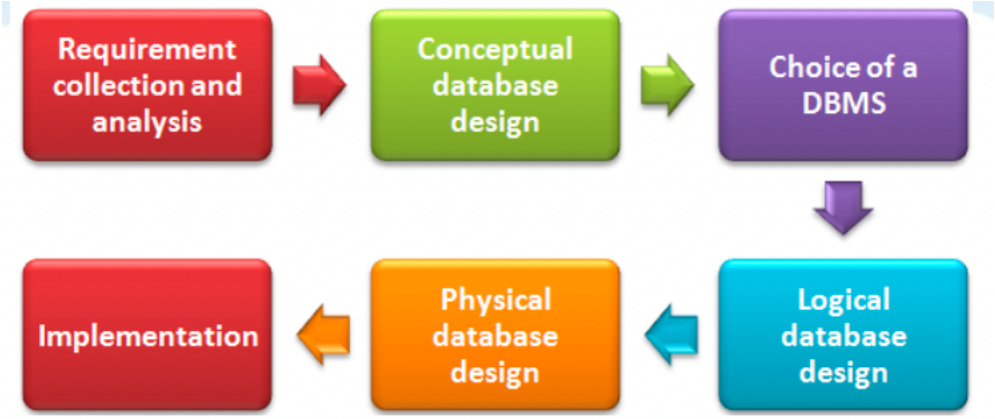
Unit-2: Database Design, Architecture and Model

Overview of Database Design Process and View of Data

Overview of Database Design Process

The process of designing the general structure of the database:



- A high-level data model provides the database designer with a conceptual framework in which to specify the data requirements of the database users, and how the database will be structured to fulfill these requirements.
- The **initial phase** of database design, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The **outcome** of this phase is a **specification of user requirements**.
- Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a **conceptual schema of the database**.
- The schema developed at this **conceptual-design phase** provides a detailed overview of the enterprise. The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. The designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.
- In terms of the relational model, the conceptual-design process involves decisions on what attributes we want to capture in the database and how to group these attributes to form the various tables.
- A fully developed conceptual schema indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data.
- Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.
- The process of moving from an abstract data model to the implementation of the database proceeds in **two final design phases**.

- **Logical Design –** Deciding on the database schema. Database design requires that we find a"good" collection of relation schemas. The designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?

Physical Design – Deciding on the physical layout of the database

Database Design for a University Organization:

- To illustrate the design process, let us examine how a database for a university could be designed. The initial specification of user requirements may be based on interviews with the database users, and on the designer's own analysis of the organization.
- The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the major characteristics of the university.

✓ The university is organized into departments. Each department is identified by a unique name (dept name), is located in a particular building, and has a budget.

✓ Each department has a list of courses it offers. Each course has associated

with it a course id, title, dept name, and credits.

✓ Instructors are identified by their unique ID. Each instructor has name,

associated department (dept name), and salary.

✓ Students are identified by their unique ID. Each student has a name, an associated major department (dept name), and tot cred (total credit hours the student earned thus far).

✓ The university maintains a list of classrooms, specifying the name of the building,

room number, and room capacity.

✓ The university maintains a list of all classes (sections) taught. Each section is identified by a course id, sec id, year, and semester, and has associated with it a semester, year, building, room number, and time slot id (the time slot when the class meets).

✓ The department has a list of teaching assignments specifying, for each instructor,

the sections the instructor is teaching.

✓ The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

View of Data

- A database system is a collection of interrelated data and a set of programs that allow usersto access and modify these data.
- A major purpose of a database system is to provide userswith an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.
- For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database.
- Since many database-system users are not computer trained, developers hide the complexity from users through **several levels of abstraction**, to simplify users' interactions with the system:

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

Data abstraction in database system is a mechanism to hide complexity of database. It allows database system to provide abstract view to database user. It hides how data are actually stored and maintained in database. Data abstraction simplifies user's interactions with the system.

There are three levels of data abstraction.

Three levels of data abstraction

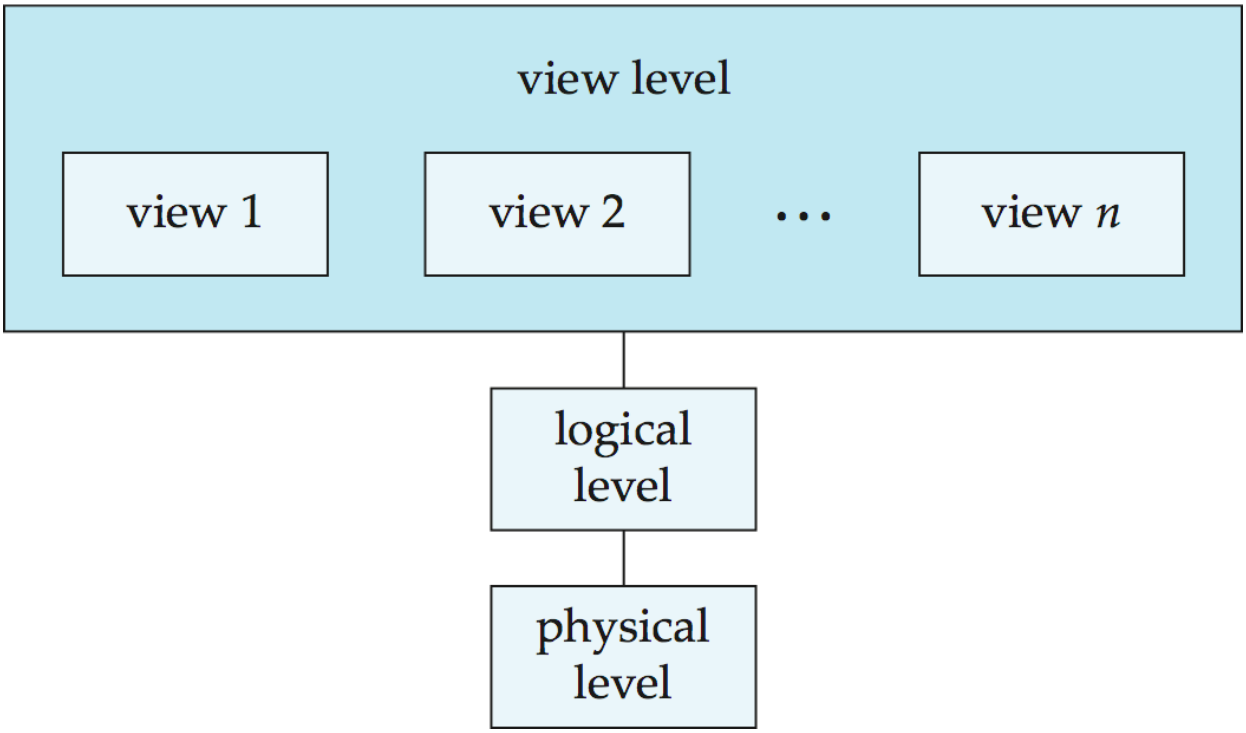


Figure 1: The three levels of data abstraction.

- **Physical level**

  The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

- **Logical level**

  The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction

- **View level**

  The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Example

View level

- View result
- View student information

Logical level: entire database schema

- Courses (CourseNo, CourseName, Credits, Dept)
- Student (StudentID, Lname, Fname, Level, Major)
- Grade (StudentID, CourseNo, Mark)

Physical level:

How these tables are stored, how many bytes it required, etc

Instances and Schemas

Database change over time as information is inserted, deleted and updated. The collection of information stored in database at a particular moment called an instance of the database. The overall design of the database is called database schema. Schemas are change infrequently, if at all.

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database

- Example: The database consists of information about a set of customers and accounts and the relationship between them
- Analogous to type information of a variable in a program
- **Physical schema**: database design at the physical level
- **Logical schema**: database design at the logical level

❖ **Instance** – the actual content of the database at a particular point in time

- Analogous to the value of a variable

✍ *The collection of information stored in the database at a particular moment is called an instance of the database*

✍ *The overall design of the database which is not expected to change frequently is called database schema*.

Level Database Architecture and Data Independence

Level Database Architecture (three-schema architecture)

The goal of the three-schema architecture (Level Database Architecture), illustrated in Figure 2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

- Internal Level
- Conceptual Level
- External Level

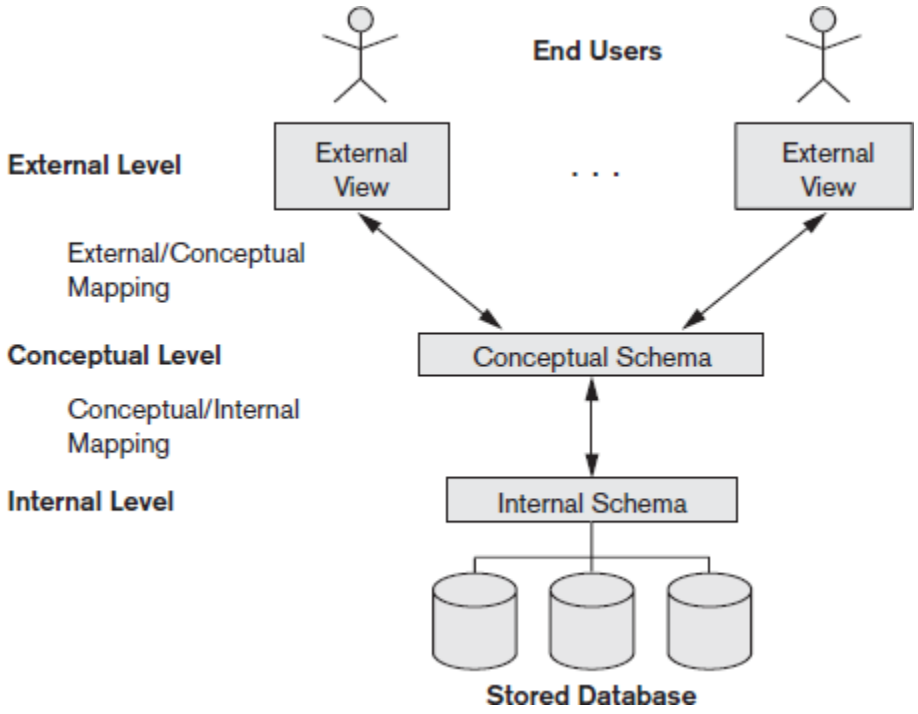


Figure 2: The three-schema Architecture

1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **Instances and Schemas**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.
3. The **external or view level** includes a number of **external schemas or user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

Data Independence

Data independence is an ability to modify a schema definition at one level of database system without affecting scheme definition in next higher level.

There are two types of data independence.

- **Physical data independence**
  - The ability to modify the physical schema without changing the logical schema
  - It is an ability to modify the physical schema without causing application programs to be rewritten Modification at this level usually required for performance improvement reason.
  - It is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
  - Some examples of changes under physical data independence:
    - Using a new storage device like Hard Drive or Magnetic Tapes
    - Modifying the file organization technique in the database
    - Switching to different data structures
    - Change of location of database from say C drive to D Drive

    Due to physical data independence, any of the above changes will not affect the conceptual layer.

- **Logical data independence**
  - It is an ability to modify the conceptual/logical scheme without causing application programs to be rewritten. Logical schema needs to be modified if we require to modify logical structure of database. Logical data independence is harder to achieve since application programs are usually dependent on logical structure of the data.
  - It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected
  - Some examples of changes under logical data independence are:

- Add/Modify/Delete a new attribute, entity or relationship is possible without the rewriting of existing application programs.
- Merging two records into one.
- Breaking an existing record into two or more records.

Due to logical data independence, any of the above changes will not affect the external layer.

Database Languages

The languages which are used to interact with database management systems is called database languages. A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates. In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language (DDL)**. The DDL is also used to specify additional properties of the data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition language**. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints. The DDL provides facilities to specify such constraints.

DDL statements are converted into equivalent low level statements by DDL interpreter. DDL is used by the database administrators or database designers to specify the conceptual schema of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). Results of DDL statements are recorded in special file called data dictionary. Structured query language supports following DDL statements:

- Create statement
- Drop statement
- Alter statement

The DDL provides the facilities to define: Database schema, Database tables, Integrity constraints, views, Security and Authorization, Modify the Schema etc. Example of a DDL statement is given below:

```
CREATE TABLE account (
     account-number CHAR(10),
     balance INTEGER,
     branch CHAR(10)
)
```

Execution of the above DDL statement creates the account table and it updates a special set of tables called the data dictionary. Data dictionary is a special file that stored metadata. Data bout data is called metadata. More accurately we can say that information about conceptual database schema is stored in data dictionary. The DDL provides the facilities to define:

- Database scheme
- Database tables
- Integrity constraints
- Domain constraints
    - Referential integrity (references constraint in SQL)
    - Assertions
    - Triggers
    - Views
- Security and Authorization
- Modify the Scheme

The common DDL Commands in SQL are: CREATE, ALTER, DROP

Data-Manipulation Language

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- **C**reate new information in the database
- **R**ead information from the database
- **U**pdate information in the database
- **D**elete information in the databaseThere are basically two types:

- **Procedural DMLs:** They require a user to specify what data are needed and how to get those data.
- **Declarative DMLs** (also referred to as **nonprocedural DMLs**): They require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than procedural DMLs. However, since a user does not have to specify how to get data, the database system has to figure out an efficient means of accessing data. The DML component of SQL is nonprocedural.

A **query** is statement requesting the retrieval of information. Special set of DML which only use to retrieve information from database called **query language**.

Example of SQL query is given below:

```
SELECT * FROM account

WHERE balance <1000
```

Execution of this statement retrieves the records of all accounts in which balance is below 1000.

QBE

- QBE stands for Query By Example
- If we talk about normal queries we fire on the database they should be correct and in a well-defined structure which means they should follow a proper syntax if the syntax or query is wrong definitely we will get an error and due to that our application or calculation definitely going to stop. So to overcome this problem QBE was introduced. It was developed in 1970 by Moshe Zloof at IBM.

- The Query By Example provides a simple interface for a user to enter queries. Instead of writing an entire SQL command, the user can just fill in blanks or select items to define the query she wants to perform.
- QBE is offered with most database programs, though the interface is often different between applications. For example, Microsoft Access has a QBE interface known as "Query Design View" that is completely graphical. The phpMyAdmin application used with MySQL, offers a Web-based interface where users can select a query operator and fill in blanks with search terms. Whatever QBE implementation is provided with a program, the purpose is the same – to make it easier to run database queries and to avoid the frustrations of SQL errors.
- Points about QBE:
    - ♦ Supported by most of the database programs.
    - ♦ It is a Graphical Query Language.
    - ♦ Created in parallel to SQL development.

Two –tier and Three-tier Application Architecture

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines, on which remote database users work, and server machines, on which the database system runs. Database applications are usually partitioned into two or three parts.

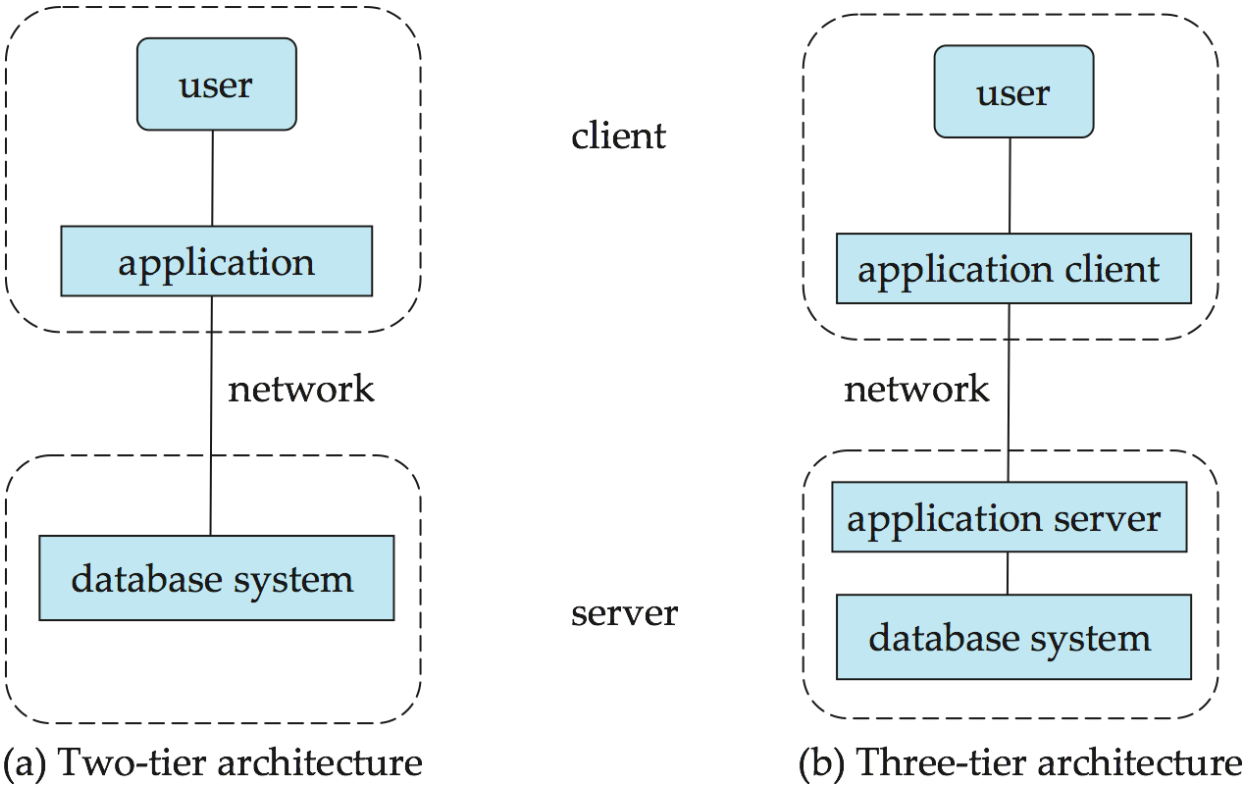


Figure 3: Two-tier and Three-tier Architectures

In a **two-tier architecture**, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

In contrast, in **a three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed

across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

Database System Architecture (Overall System Structure)

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components.

The storage manager is important because databases typically require a large amount of storage space. Since the main memory of computers cannot store such volume of information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.

The Query Processor is important because it helps the database system simplify and facilitate access to data. It allows database users to obtain good performance without being burdened with understanding the physical-level details of the implementation of the system. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

1. Storage Manager

The storage manager is a component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving and updating data in the database.

The storage manager components include:

- **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

- **Transaction manager**, which ensures that the database remains in a consistent state despite system failures, and that concurrent transaction executions proceed without conflicting.
- **File Manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
- **Buffer Manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation:

- **Data files**, which store the database itself.
- **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**, which provide fast access to data items.

2. The Query Processor

The query processor components include:

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low level instructions that the query evaluation engine understands.
  A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.
- **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

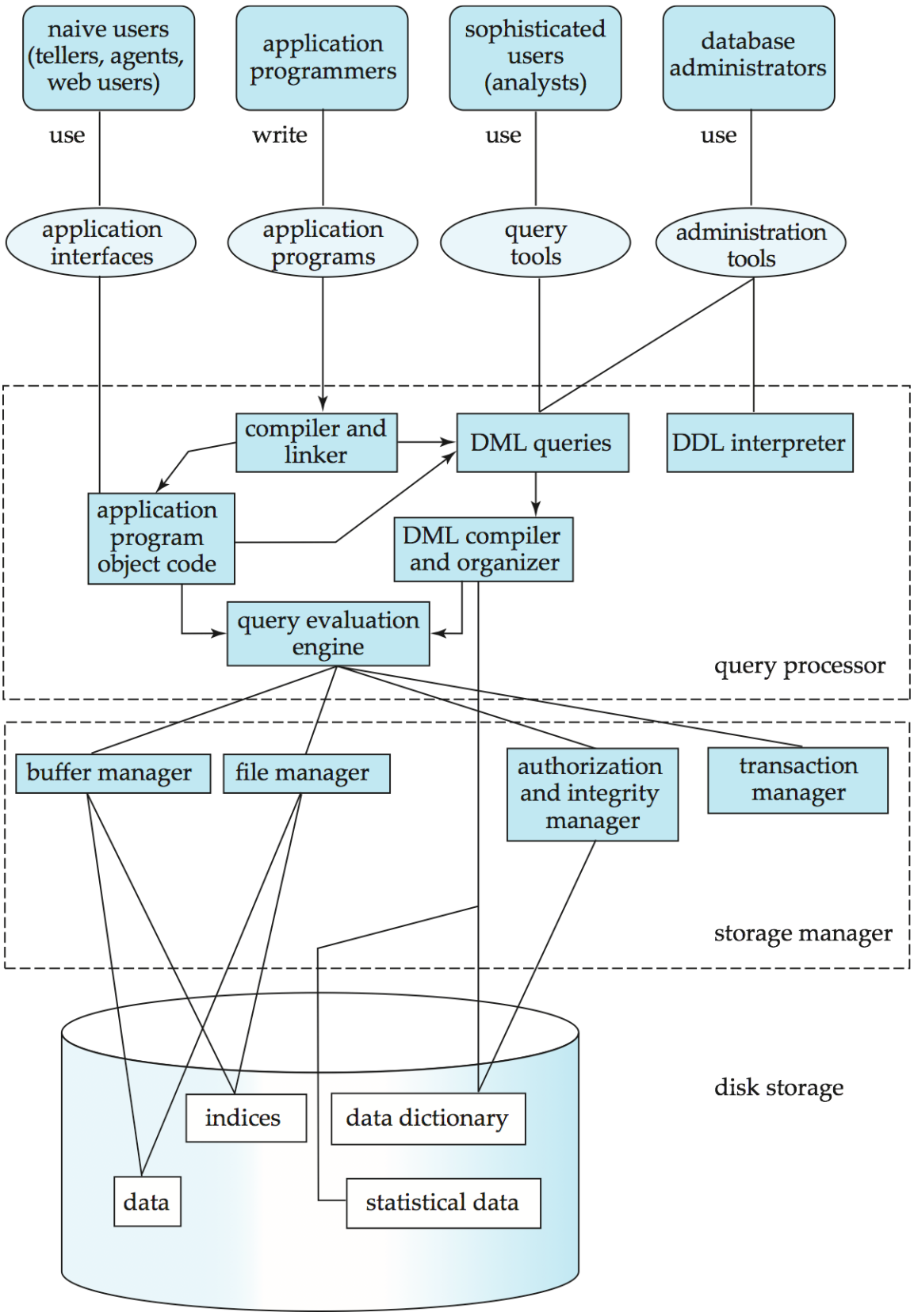


Figure 4: Database System Structure

Data Models

Data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels. Some of the data models are as follows:

Entity-Relationship Data Model (E-R Model)

The entity-relationship (E-R) data model is a high-level data model that uses a collection of basic objects, called entities, and relationships among these objects. An entity is a thing or object in the real world that is distinguishable from other objects. Entities are described in a database by a set of attributes. A relationship is an association among several entities. E-R data model is best used for the conceptual design of a database. Overall logical structure of a database can be expressed graphically by E-R diagram. The basic components of this diagram are:

- Rectangles (represent entity sets)
- Ellipses (represent attributes)
- Diamonds (represent relationship sets among entity sets)
- Lines (link attributes to entity sets and entity sets to relationship sets)

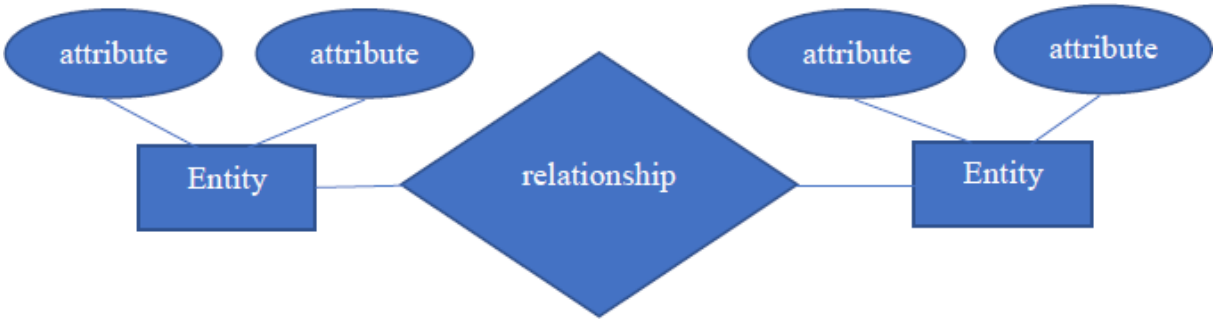


Figure 5: ER Diagram Concept

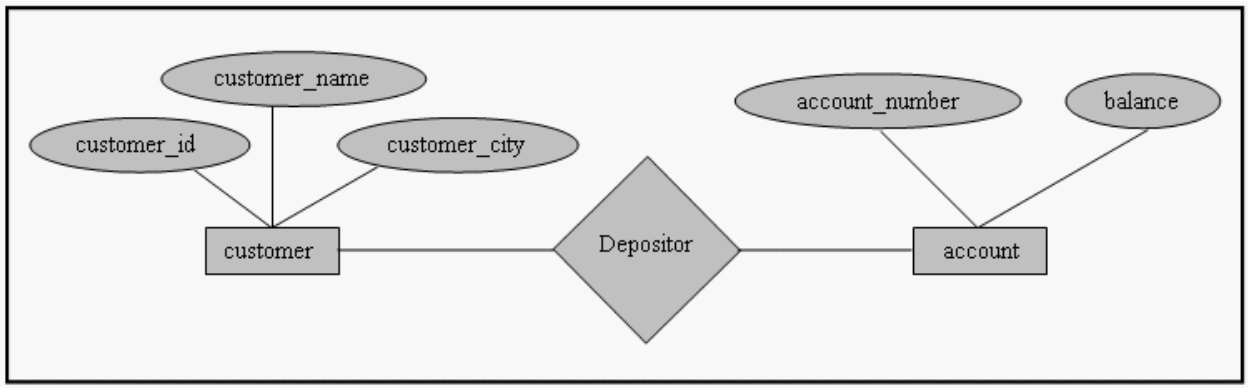


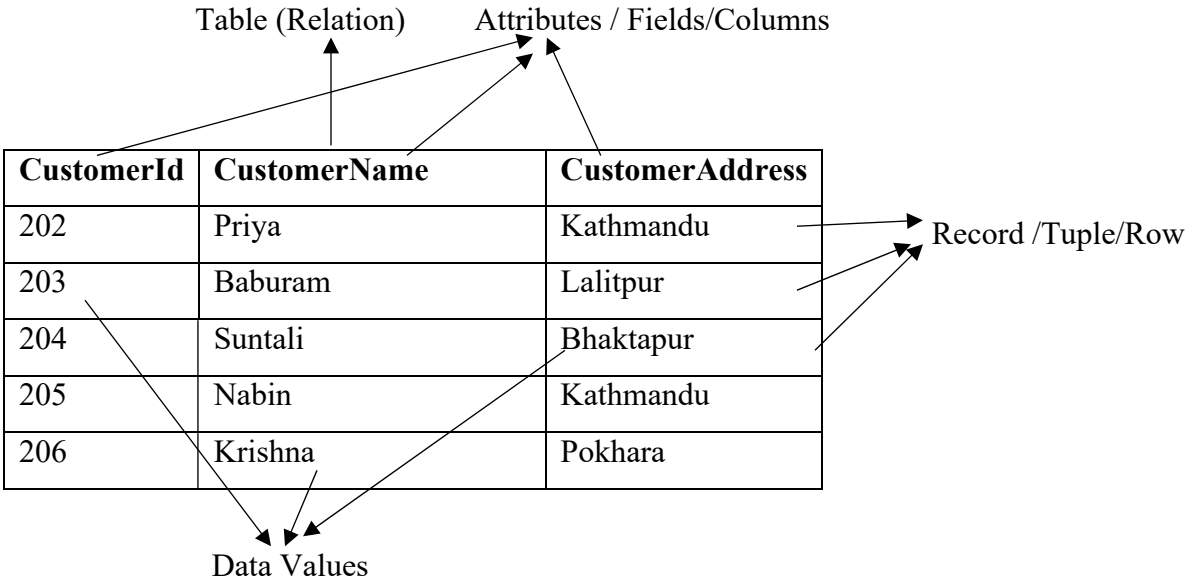
Figure 6: An Example of ER data Model

Beside entities and relationship among them, E-R model has a capability to enforce constraints, mapping cardinalities which tell no. of entities to which another entity can be associated via relationship set. If each account must belong to only one customer, E-R model can express it.

Relational Data Model

Relational model describes database design by a collection of tables (relations). It represents both data and their relationships among those data. Each table consist number of columns (attributes) with unique names.

It is the most widely used data model. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are known as relations. Each table corresponds to an entity set or relationship set, and each row represents an instance of that entity set or relationship set. Relationships link rows from two tables by embedding row identifiers (keys) from one table as attribute values in the other table. Structured Query Language (SQL) is used to manipulate data stored in tables.



CustomerId	CustomerName	CustomerAddress
202	Priya	Kathmandu
203	Baburam	Lalitpur
204	Suntali	Bhaktapur
205	Nabin	Kathmandu
206	Krishna	Pokhara

Object-based Data Model

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model. Object-oriented data model is an extension to E-R model with notions of encapsulation, methods (functions), and object identity. It defines a database in terms of objects, classes, and methods. The object-relational data model combines features of the object-oriented data model and relational data model.

Example:

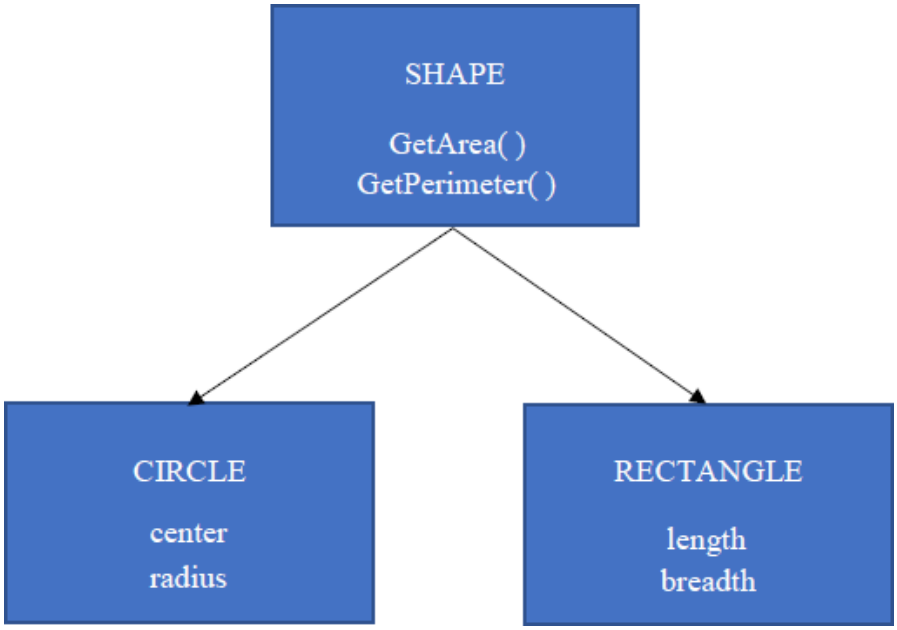


Figure 7: An Example of Object based data Model

Shape, Circle, and Rectangle are all objects in this model.

- Circle has the attributes center and radius.
- Rectangle has the attributes length and breadth.
- The objects Circle and Rectangle inherit from the object Shape.

Hierarchical Data Model

Hierarchical model is the record-based representational or implementation data model thus represents data by a set of records but records are organized in hierarchical or order structure and database is a collection of such disjoint trees. The nodes of the tree represent record types. Hierarchical tree consists one root record type along with zero or more occurrences of its dependent subtree and each dependent subtree is again hierarchical. A parent record can have several children, but a child can have only one parent. It, therefore, represents only one-to-one and one-to-many relationships.

Example:

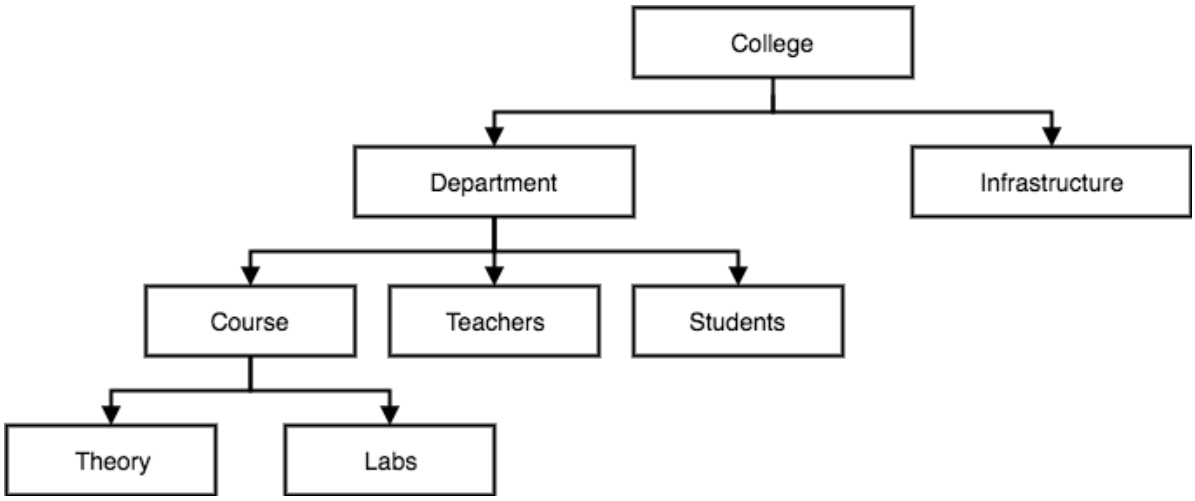


Figure 8: An Example of Hierarchical Data Model

Network Data Model

A network data model is an extension of the hierarchical database structure. It is also record-based representational data model where data are represented by the set of records and relationships among data are represented by links. Network model is more flexible than hierarchical data model. It describes data and relations between data by using graph rather than tree like structure. Thus unlike hierarchical data model, network data model is able to represent many-to-many relationships also.

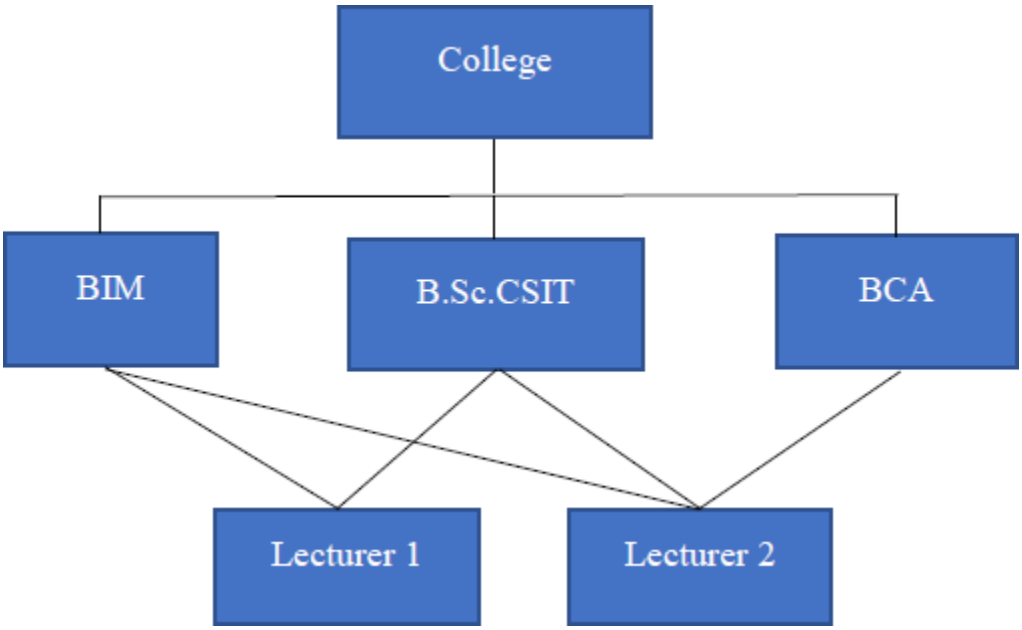


Figure 9: An Example of Network Data Model

Entity-Relationship Model and E-R Diagram in Detail

- Entity-relationship model describes data involves in real world in terms of object and their relationships. It is widely used for initial database design. It describes overall structure of database. E-R model is in fact, semantic data model which describes the meaning of data. It has a capability to map the meanings and interactions of real world objects on to the conceptual schema.
- The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a database can be modeled as a collection of entities, and relationship among entities.
- The ER data mode was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database.
- The ER data model employs three basic concepts:
    - entity sets,
    - relationship sets,
    - attributes.

- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

Entities and Entity Sets

An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. For example, each student in a class is an entity.

  - Example: specific person, company, event, plant
- Entities can be described by a set of properties called attributes. For example: customer_id, customer_name, customer_address are attributes for entity customer. Similarly, course_id, course_name are attributes for entity course.

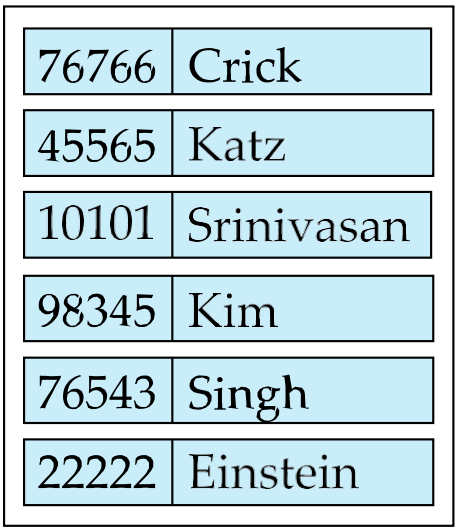
An **entity set** is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set instructor. Similarly, the entity set student might represent the set of all students in the university.

- Entities of an entity set has same set of attributes.
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.

  Example: **instructor=(ID,name,salary)**

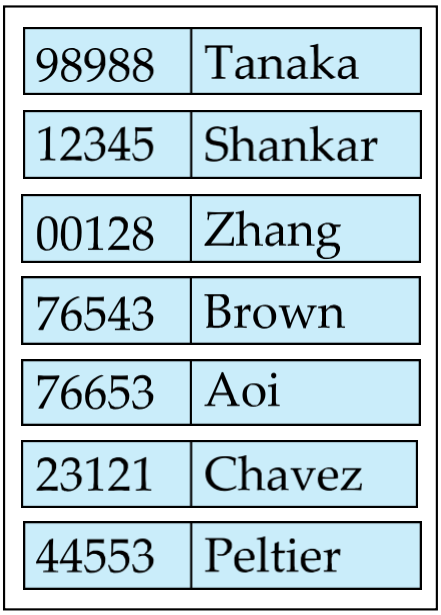
  course= (course_id, title, credits)

Entity Sets -- instructor and student



76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor



98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Relationship and Relationship Sets

- A **relationship** is an association among several entities. For example, ―Bipin teaches Rekha, here teaches is the association between entities Bipin and Rekha
- Example: we can define a relationship **advisor** that associates instructor **Katz** with student **Shankar**. This relationship specifies that Katz is an advisor to student Shankar.
- A relationship set is a set of relationship of same type. Formally, it is a mathematical relation on n ≥ 2 (possibly nondistinct) entity sets. If E1, E2, . . . , En are entity sets, then a relationship set R is a subset of {(e1, e2, . . . , en) | e1 ∈ E1, e2 ∈ E2, . . . , en ∈ En} where (e1, e2, . . . , en) is a relationship.
- In another way, we can say that association between entity sets is called relationship set. For example, ―Teacher teaches Student‖, here **teaches** is the association between entity sets "Teacher" and " Student"
- Example: we define the relationship set **advisor** to denote the associations between students and the instructors who act as their advisors. Pictorially, we draw a line between related entities.

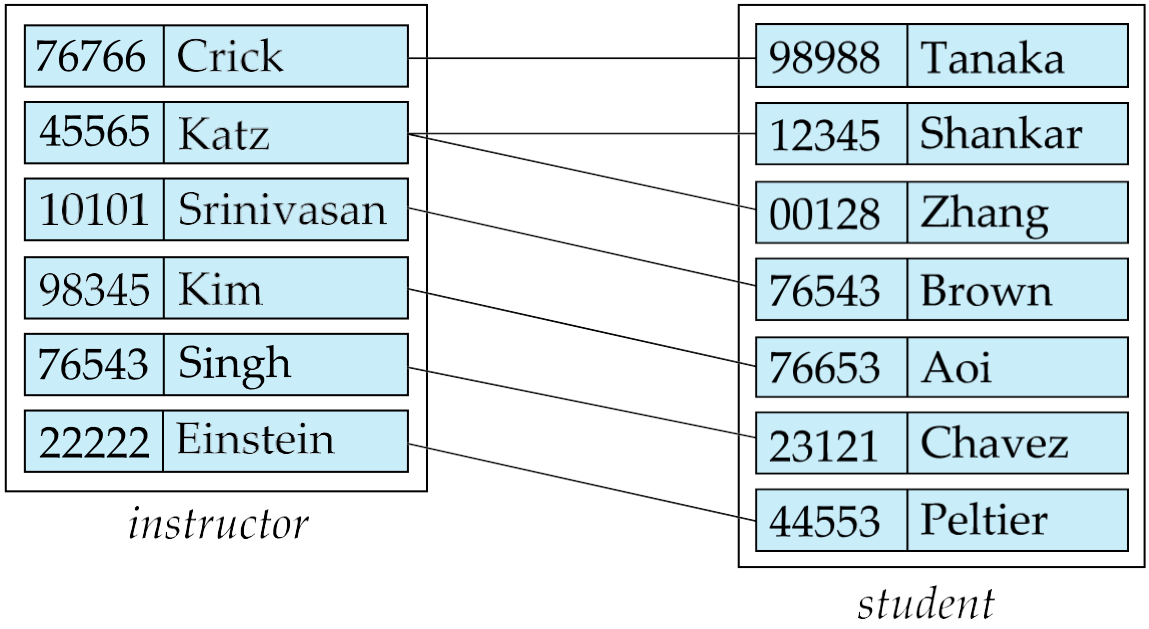


Figure: Relationship set *advisor*.

- An attribute can also be associated with a relationship set i.e. a relationship may also have attributes called **descriptive attributes.** For instance, the **advisor** relationship set between entity sets **instructor** and **student** may have the attribute date which tracks when the student started being associated with the advisor
- We could associate the attribute date with that relationship to specify the date when an instructor became the advisor of a student. The advisor relationship among the entities corresponding to instructor Katz and student Shankar has the value "10 June 2007" for attribute date, which means that Katz became Shankar's advisor on 10 June 2007.

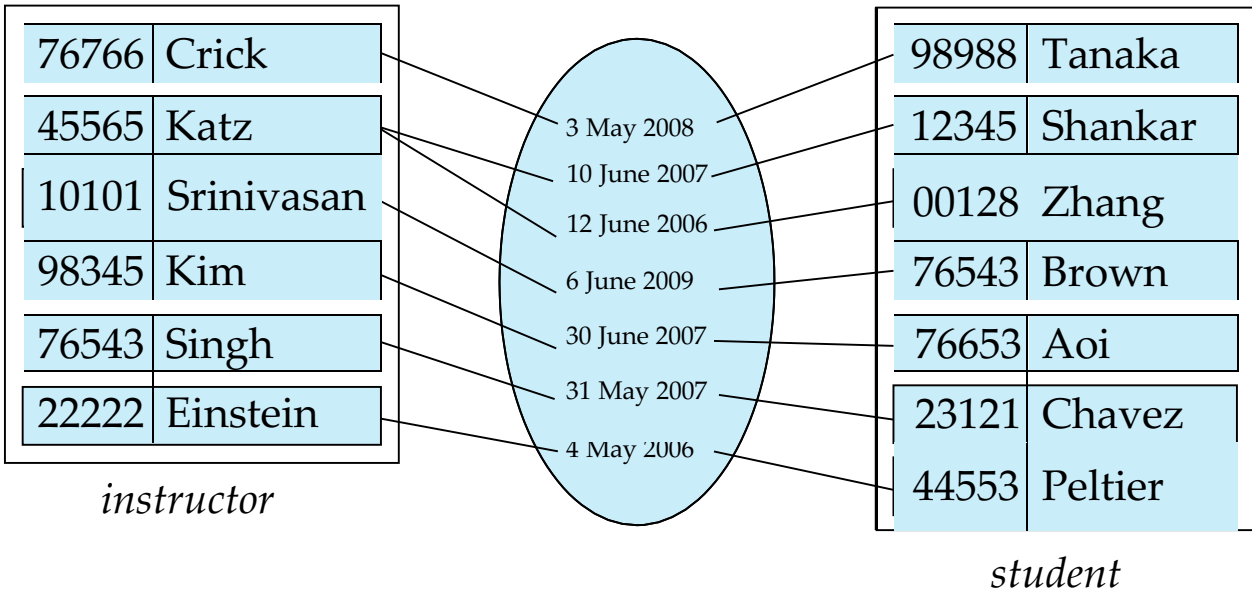


Figure: *date* as attribute of the *advisor* relationship set.

Degree of a Relationship Set

- The number of entity sets that participate in a relationship set is the degree of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.
- Binary relationship: involve two entity sets (or degree two).
    - most relationship sets in a database system are binary
- Relationships between more than two entity sets are rare. Most relationships are binary.
    - Example: *students* work on research *projects* under the guidance of an *instructor*.
    - relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Attributes

- An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.
- In simple, attribute is descriptive property of entity set. Set of attributes describes entity set. For example

customer = (customer-id, customer-name, customer-city)

account=(account_number, balance)

loan = (loan_number, amount)

- The set of permitted values for an attribute called domain of that attribute. For example
  - set of all text strings of certain length for customer_name is domain of that attribute.
- Formally, an attribute is a function which maps an entity set into a domain. Every entity is described by a set of (attribute, data value) pairs. There is one pair for each attribute of the entity set.

  For example: particular customer entity in customer entity set can describe by the set of pairs :

  (customer_id, c01),

  (customer_name, Hari) and,

  (customer_city,Kathmandu).

Types of Attributes

❖ **Simple and Composite attributes**

- Attribute which cannot be divided into subparts (i.e. into other attributes) called **simple attribute**.
- For example, **customer_id** in customer entity set is simple attribute, since it cannot be divided into sub attributes.
- Attribute that can further divide into subparts called **composite attribute**. For example, **customer_name** in customer entity set is composite attribute since it can be divided into sub attributes: **customer_fname, customer_mname** and **customer_lname**. Composite attributes helps to group related attributes, which makes modeling clearer.

❖ **Single-valued and Multivalued attributes**

- Attribute that can take only one value in every entry called **singled-valued attribute.** For example, attribute **customer_name** in customer entity set is single-valued attribute since it can not contain more than one customer name in any entry.
- An attribute that can take more than one values in any entry called multivalued attribute. For example, in a customer entity set attribute **customer_phonenumber**

is multivalued attribute since customer may have zero or one or several phone number.

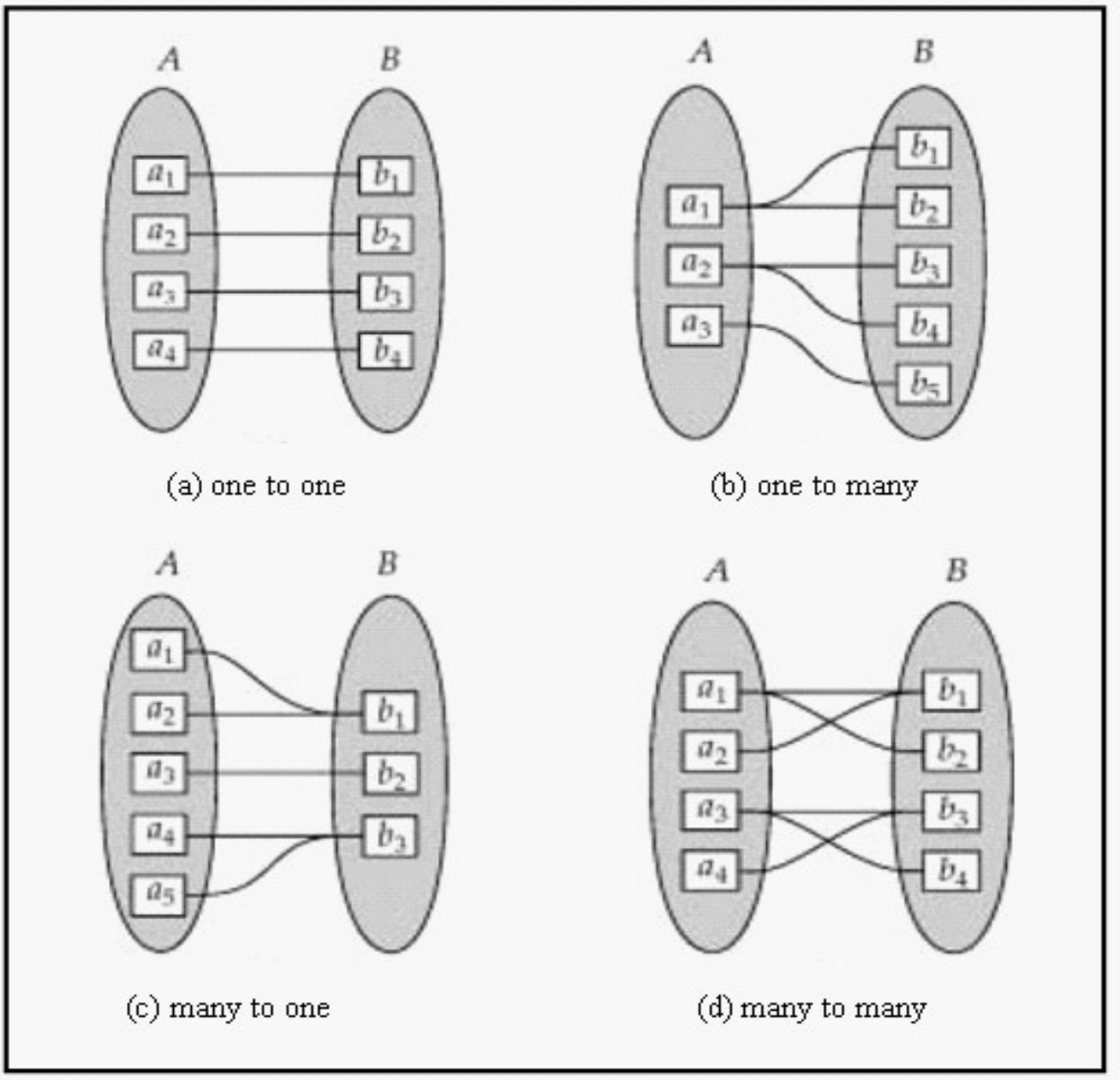
- **Stored and Derived attributes**
  - Attribute whose values can be derived from the values of other related attributes or entities called **derived attribute**.
  - And the attribute whose value is stored in database is called **stored attribute**.
  - For example, in customer entity set, attribute **age** is derived attribute if customer entity set has attribute **date_of_birth**. We can derive **age** of customer from **date_of_birth** and **current_date**. Here the attribute **date_of_birth** is **stored attribute** and the attribute **age** is **derived attribute**. The value of derived attribute is not stored, it is computed when required.

Constraints in E-R Model

- E-R model has a capability to enforce constraints. Two most important type of constraints in E-R model are-
  - **Mapping Cardinalities (Cardinality ratio),**
  - **Participation Constraints**

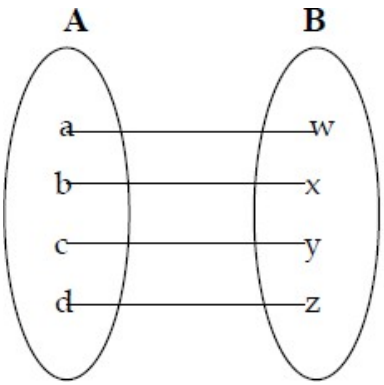
Mapping Cardinality Constraints

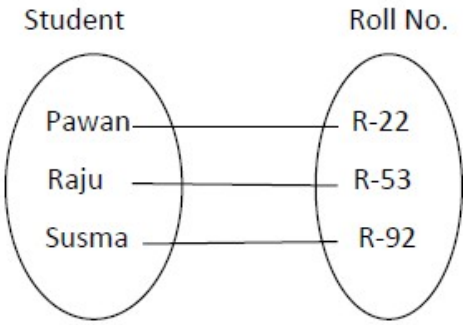
- Express the number of entities to which another entity can be associated via a relationship set. It is most useful in describing binary relationship sets.
- Mapping Cardinalities describes no. of entities to which another entity can be associated via relationship set.
  - Mapping cardinalities are most useful in describing binary relationship sets but it can also describe relationship sets that involve more than two entity sets.
- For binary relationship set between entity set A and B mapping cardinality must one of the following.
  a) **One to one**
  b) **One to many**
  c) **Many to one**
  d) **Many to many**



One to one relationship

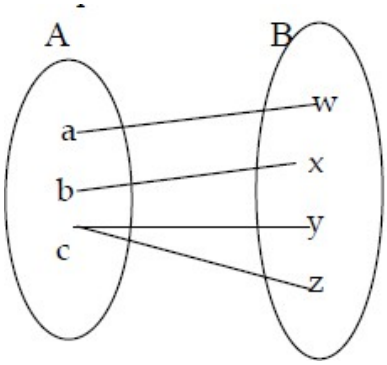
- ✓ If every entity in set A is associated with at most one entity in set B and vice-versa then the relationship is called one-to-one relationship. The following figure shows one to one mapping cardinality between entity sets A and B

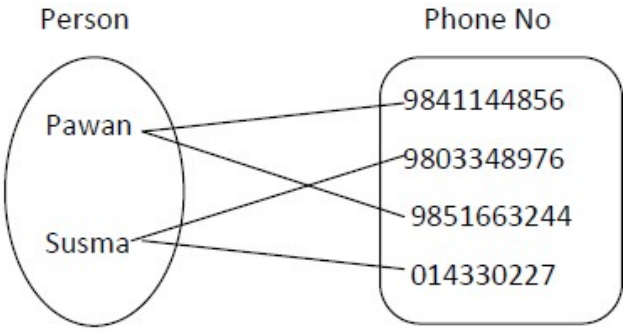




One-to-Many Relationship

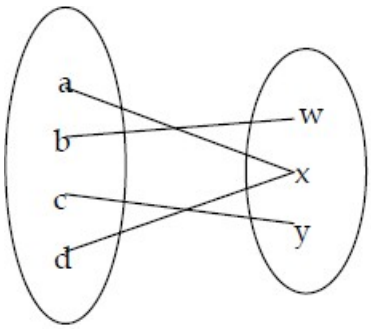
- ✓ If an entity in set A can be associated with any number (zero or more) of entities in set B but every entity in set B can be associated with at most one entity in set A then it is called one-to-many relationship.

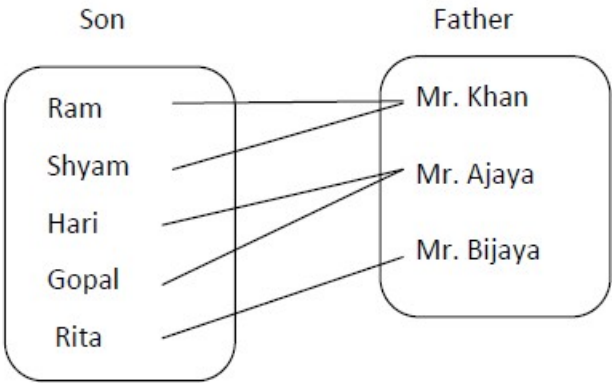




Many-to-One Relationship

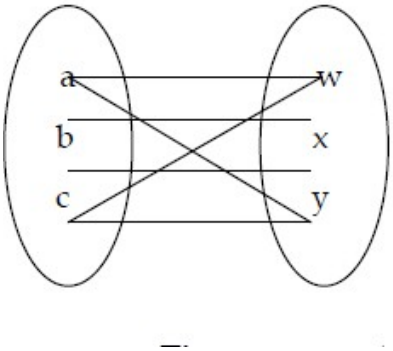
- ✓ If every entity in set A can be associated only one of entities in set B but an entity in B can be associated with any number of entities in set A, then it is called many-to-onerelationship.

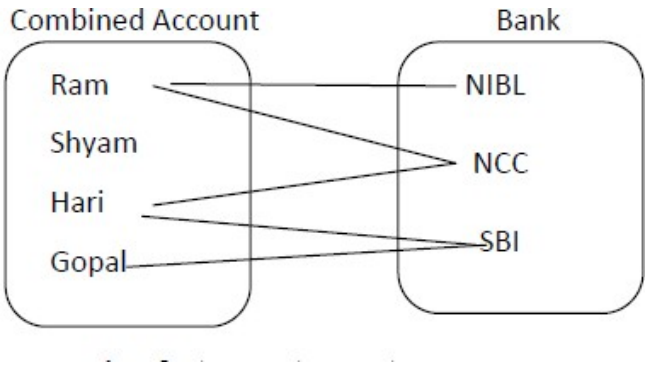




Many-to-Many Relationship

- ✓ If an entity in set A can be associated with any number of entities in set B and vice versa then it is called many-to-many relationship.





Note: In all fours types, some elements in A and B may not be mapped to any elements in the other set

Participation Constraints

- ➢ The participation of an entity set E in a relationship set R is said to be **total if every entity in E participates in at least one relationship in R**. **If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial**.

- **Total participation:** every entity in the entity set participates in at least one relationship in the relationship set. Example: Every student must have an associated instructor. Therefore the participation of *student* in the relationship set *advisor* is total.
- **Partial participation**: some entities may not participate in any relationship in the relationship set. Example: participation of *instructor* in *advisor* is partial
- The participation of *loan* in the relationship set *borrower* is total but *customer* entity set in *borrower* relationship set is partial since not all customers necessarily take loan from bank, customer may also those who are only account holder.

Keys

- Set of one or more attributes whose values are distinct for each individual entity in the entity set is called key, and its values can be used to identify each entity uniquely
- We must have a way to specify how entities within a given entity set are distinguished. Conceptually, individual entities are distinct; from a database perspective, however, the differences among them must be expressed in terms of their attributes.
- Therefore, the values of the attribute values of an entity must be such that they can uniquely identify the entity. In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.
- The concept of key is important to distinguish one entity from another and one relationship from another relationship. In fact, values of attributes distinguish one entity from another entity. To distinguish one entity from another entity in entity set there must exist attribute/s whose values must not duplicate in entity set. It ensures no two entities in an entity set can exist with same values for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other.
- Keys also help to identify relationships uniquely, and thus distinguish relationships from each other.
- There are different types of keys which are:
    - Super key

- Candidate key
- Primary key

Super Key

→ A **super key** is a set of one or more attributes which uniquely identifies an entity in entity set.

- For example: in customer entity set, single attribute **customer_id** is sufficient to uniquely identify one customer entity to another. So **customer_id** is a **superkey** in a customer entity set.
- Since combination of **customer_id** and **customer_name** can also uniquely identifies one customer entity to another. So combination of attributes **{customer_id,customer_name}** is also **superkey** in customer entity set.
- But single attribute **customer_name** can not be superkey in customer entity set because customer name only cannot uniquely identify one customer entity to another, there would be number of customers having same name.

→ If K is a super key and any superset of K is also super key.

Candidate Key

- The minimal superkey is called candidate key. That is, candidate key is a superkey but its proper subset is not superkey. For example: **customer_id** is a candidate key in customer entity set . Similarly **account_id** is a candidate key in account entity set.
- The candidate keys are defined as the set of keys that is minimal and can uniquely identify an entity.
- A **candidate key** of an entity set is a minimal super key. For example, **student-id** is candidate key of the entity set student but set of attributes **{student-id, name, program, semester, section}** is not candidate key of the entity set student because it has proper subset **{student-id, program, semester}**which is also key.
- **All candidate keys are super keys but vice versa is not true.**

Primary Key

- A **primary key** is a candidate key that is chosen by the database designer as the principle means of uniquely identifying entities within an entity set.

- There may exist several candidate keys, one of the candidate keys is selected to be the primary key.
- For example in employee entity set, **eid** can be best suited for the primary key. Rest of the attributes like **SSN**, **Passport_Number**, and **License_Number**, etc. are considered as a candidate key
- Primary keys provide a way to specify how entities and relations are distinguished

Primary Keys for Entity Sets and Relationship Sets.

Primary Key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
- No two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other
- The primary key of an entity set allows us to distinguish among the various entities of the set

Primary Key for Relationship Sets

- The primary key of an entity set allows us to distinguish among the various entities of the set. We need a similar mechanism to distinguish among the various relationships of a relationship set.
- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
    - Let *R* be a relationship set involving entity sets E1, E2, .. En
    - The primary key for R is consists of the union of the primary keys of entity sets E1, E2, ..En

      If the relationship set R has no attributes associated with it, then the set of attributes

primary-key(E1) ∪ ***primary-key(E2)*** ∪ ⋯ ∪ ***primary-key(En)***

describes an individual relationship in set R.

- If the relationship set *R* has attributes a1, a2, .., am associated with it, then the primary key of *R* also includes the attributes a1, a2, .., am

  If the relationship set R has attributes a1, a2, . . . , am associated with it, then

  the set of attributes

  primary-key(E1) ∪ ***primary-key(E2)*** ∪ ⋯ ∪ ***primary-key(En)*** ∪ ***{a1, a2, . . . , am}***

  describes an individual relationship in set R.

- In both of the above cases, the set of attributes **primary-key(E1)** ∪ **primary-key(E2)** ∪ ⋯ ∪ **primary-key(En)** forms a superkey for the relationship set.

- Example: relationship set "advisor".
  - The primary key consists of *inrsructor.ID* and s*tudent.ID*

- This indicates that primary keys of entity sets those involves in relationship set form super key in relationship set. For example: in relationship set depositor, **{customer_id, account_number}** are attributes and each attribute **customer_id** and **account_number** are primary key in set customer and account respectively.
  - If relationship set consist declarative attribute say **access_date** then it can also consider as attributes of relationship set depositor.
- The selection of primary key in relationship set depends up on mapping cardinality of that relationship set.
  - As an illustration, consider the entity sets **instructor** and **student**, and the relationship set **advisor**, with attribute **date**.
  - Suppose that the relationship set is many-to-many. Then the primary key of **advisor** consists of the union of the primary keys of **instructo**r and **student**

    So, in Many-to-Many relationships, the preceding union of the primary keys isa minimal superkey and is chosen as the primary key

- If the relationship is many-to-one from **student** to **instructor**—that is, each student can have have at most one advisor—then the primary key of **advisor** is simply the primary key of **student**.

  So,in Many-to-one relationships, the primary key of the "Many" side is a minimal superkey and is used as the primary key.

- However, if an instructor can advise only one student— that is, if the advisor relationship is many-to-one from **instructor** to **student**—then the primary key of **advisor** is simply the primary key of **instructor**.

  So, in One-to-Many relationships, the primary key of the "Many" side is a minimal superkey and is used as the primary key.

- For one-to-one relationships either candidate key can be used as the primary key. i.e .the primary key of **advisor** is primary key of **student** or primary key of **instructor**.

  So, in One-to-one relationships, the primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key.

- For non-binary relationship set where no cardinality constraints are defined then super key is only one possible candidate key. So it needs to be chosen as a primary key.

Task: Discuss about selection of primary key in depositor relationship of entity sets, customer and account. Assume that the primary keys of customer and account entity sets are customer_id and account_no respectively and descriptive attribute of depositor relationship set is access_date .

The Entity Relationship Diagram

- Once the entity types, relationships types, and their corresponding attributes have been identified, the next step is to graphically represent these components using entity-relationship (E-R) diagram.
- An E-R diagram is a specialized graphical tool that demonstrates the interrelationships among various entities of a database. It is used to represent the overall logical structure of the database.
- While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changed rarely; however, the instances in the entity and relationship sets change frequently.
- Thus, E-R diagrams are more useful in designing the database.
- E-R diagram focuses high level database design and hides low level details of database representation therefore it can be used to communicate with users of the system while collecting information
- Major components of E-R diagram are:

    - **Rectangles** - represent entity sets.
    - **Diamonds** - represent relationship sets.
    - **Lines** - link attributes to entity sets and entity sets to relationship sets.
    - **Ellipses** - represent attributes
    - **Double ellipses** - represent multivalued attributes
    - **Dashed ellipses**- denote derived attributes.
    - **Underline**- indicates primary key attributes
    - **Double Lines** - indicate total participation of an entity set in a relationship set.
    - **Double Rectangles** - represent weak entity sets.
    - **Double Diamonds** - represent identifying relationship set for weak entity set.

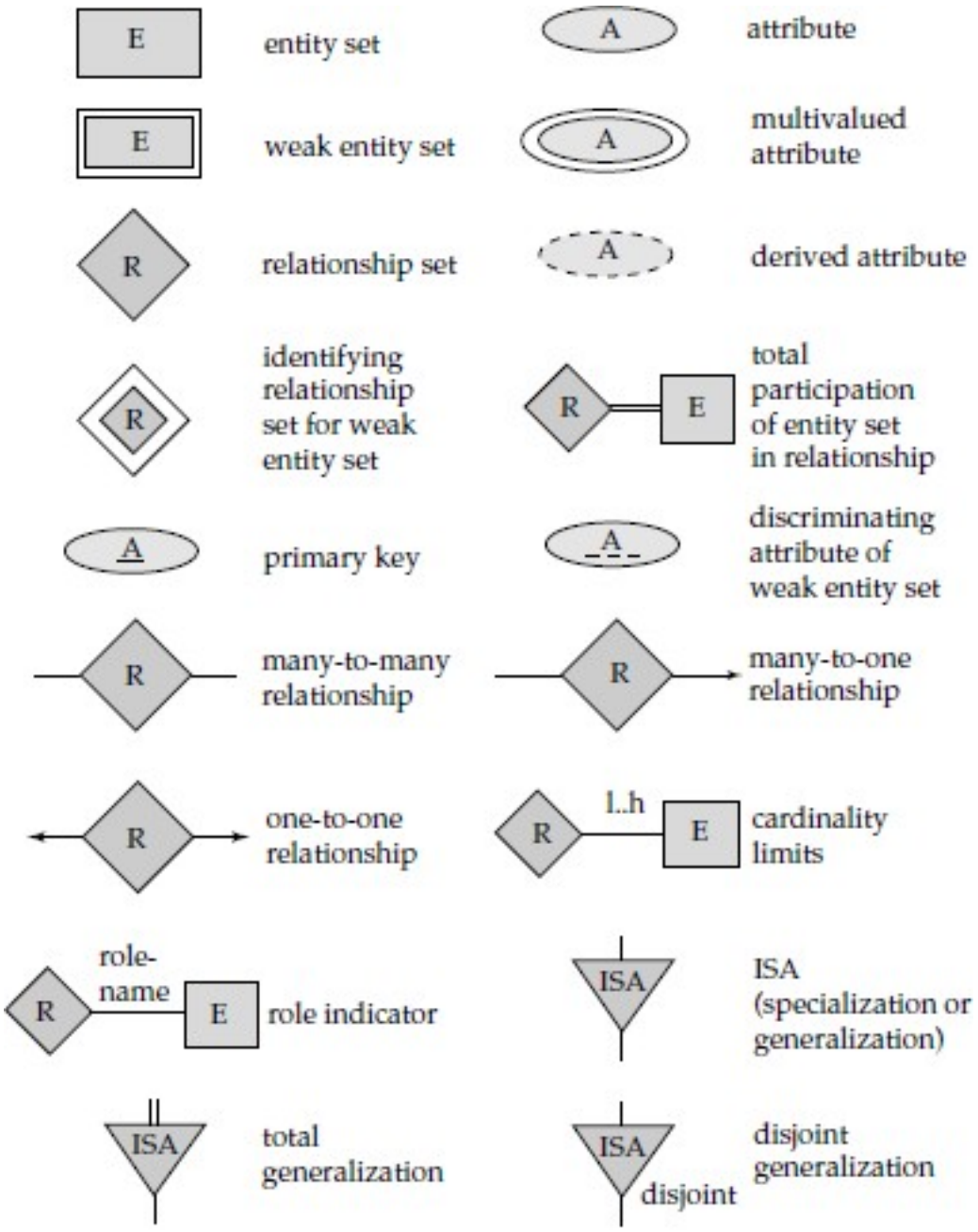


Figure Symbols used in the E-R notation.

Representation of relationship set in E-R Diagram

- The relationship set borrower having two entity sets customer and loan can be express as

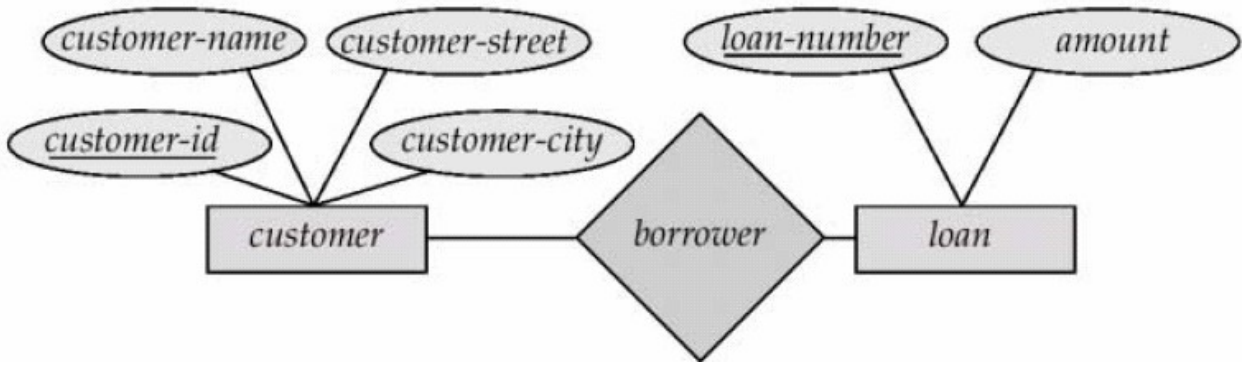


Figure: borrower relationship set in E-R Diagram

Representation of mapping cardinalities of relationship set in E-R Diagram

- The directed lines in E-R diagram are used to specify mapping cardinalities of relationship sets. **The directed line (→) tells "one," and an undirected line (—) tells "many" between the relationship set and the entity set.**

One-to-one relationship representation

☞ ***One to One (1:1)*** – "Student allotted a project" signifies a one-to-one relationship because only one instance of an entity is related with exactly one instance of another entity type.

Figure: one to one relationship set in E-R Diagram

☞ This indicates a project is associated with at most one student via the relationship *Allotted* and a student is associated with at most one project via *Allotted*.

One to many relationship representation

- **One to Many (1: M)** – "A department recruits faculty" is a one-to-many relationship because a department can recruit more than one faculty, but a faculty member is related to only one department.

Figure: one to many relationship set in E-R Diagram

- This indicates a department can recruit more than one faculty, but a faculty member is related to only one department.

Many to one relationship representation

- **Many to One (M:1) –** "Many houses are owned by a person" is a many-to-one relationship because a person can own many houses but a particular house is owned only a person.

Figure: many to one relationship set in E-R Diagram

- This indicates a person can own many houses but a particular house is owned only a person.

Many to many relationship representation

- **Many to Many (M:N) –** "Author writes books" is a many-to-many relationship because an author can write many books and a book can be written by many authors.

Figure: many to many relationship set in E-R Diagram

- This indicates an author can write many books and a book can be written by many authors.

Representation of Composite, Multivalued, and Derived Attributes in E-R Diagram

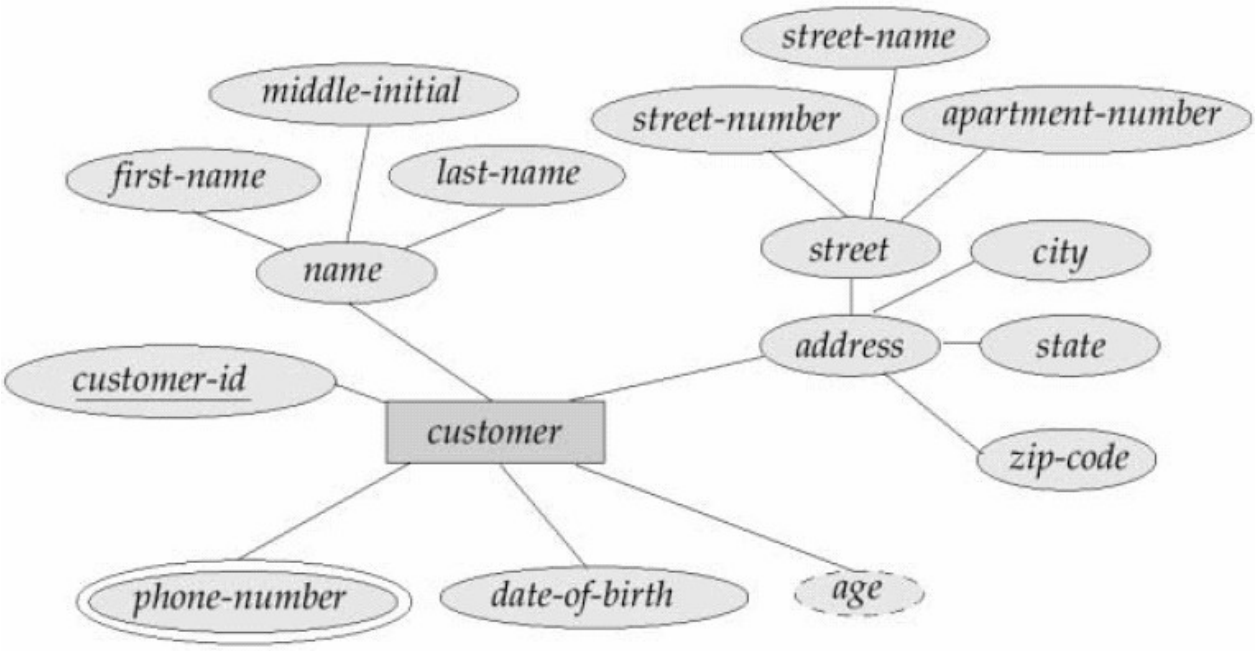


Figure: E-R Diagram with Composite, Multivalued, and Derived Attributes

- Here attribute "name" is composite attribute, "phone_number" is multivalued attribute and "age" is derived attribute.

Representing unary, binary, and n-ary relationships in E-R Diagram

Representing unary relationship (aka recursive relationship)

- **Unary Relationship:** If only one entity set participates in relationship more than it is called unary relationship.
- Here same entity set participates in relationship twice with different roles. Role names are specified above the link joining entity set and relationship set.
- This type of relationship set is sometimes called a **recursive relationship** set.
- In the example below, employee participates twice in relationship, once **as a manager** and again **as worker.**
- Here relationship set **"work-for"** is **recursive relationship set** and the labels **"manager"** and **"worker"** are roles.

- Roles in E-R diagrams are indicated by labeling the lines that connect diamonds to rectangles. Role labels are optional, and are used to clarify semantics of the relationship

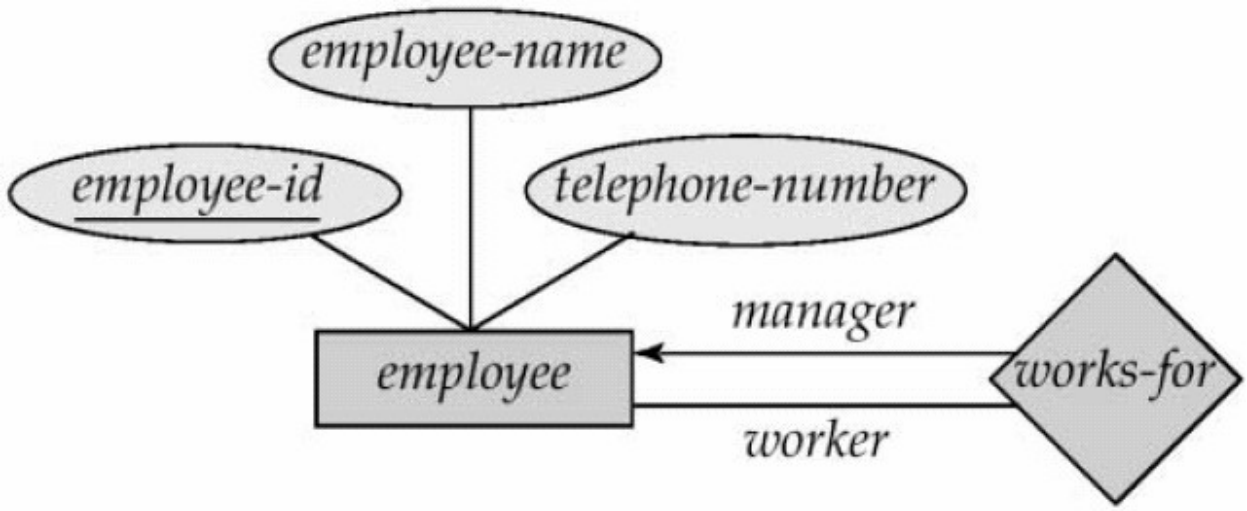


Figure: Roles in E-R Diagram

Representing binary relationship

- **Binary Relationship:** A Relationship set in which two entity sets participate is called binary relationships.
- In another way, we can say that relationship sets of degree 2 are called binary relationship. This is the most common type of relationship in database systems.
- For example, the —teaches— relationship given below has degree two and hence it is binary relationship

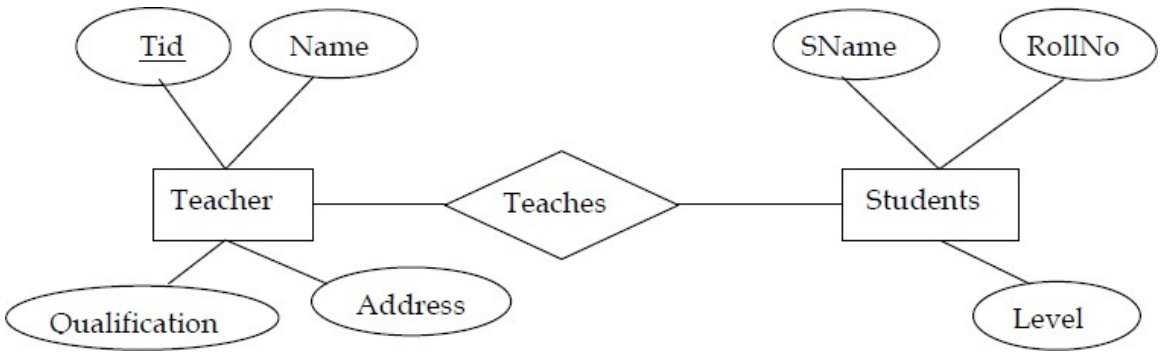


Figure: Binary relationship in E-R Diagram

Representing n- ary relationship

- Relationship set in which more than two entity sets involves is called **n-ary relationship**.
- **Ternary** and **quaternary** relationships are special cases of **n-ary** relationship.
- For example:

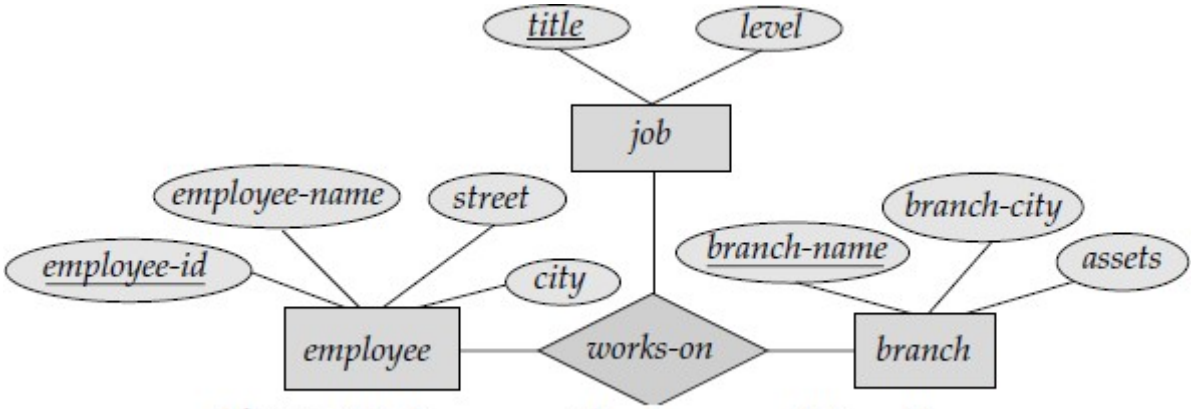


Figure E-R diagram with a ternary relationship.

Representation of participation constraints of entity set in relationship set

- In E-R diagram **total participation** of entity set in relationship set indicated by **double line** between that relationship set and entity set, **single line** indicates **partial participation**.

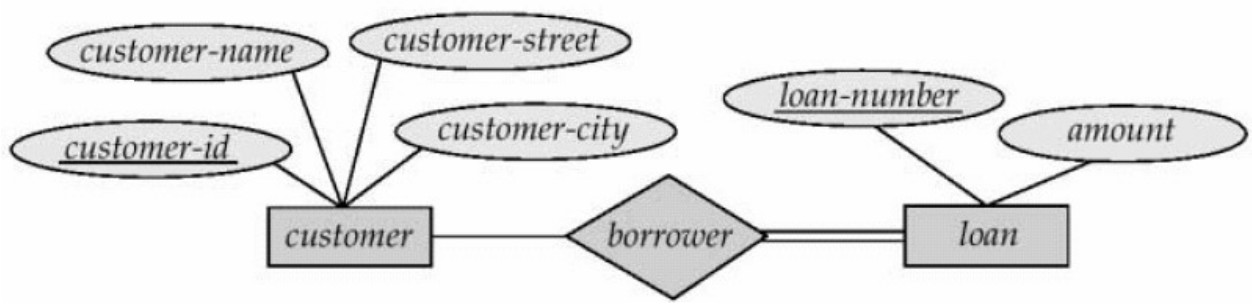


Figure: participation constraints in E-R Diagram

- This indicates participation of loan in borrower relationship is total and participation of customer in borrower relationship is partial. That is, every loan must have a customer associated to it via borrower but all customers may not associate to loan.

Notation for Expressing More Complex Constraints

- E-R diagram can also specify more complex constraints.
- It can specify cardinality limits. That is, it can specify how many no. of times each entity may participates in relationships in relationship set.
- An edge between an entity set and binary relationship can have an associated minimum and maximum cardinality in the form **l..h**, where **l** is the minimum and **h** is the maximum cardinality.
- A minimum value **1** indicates **total participation** of the entity set in the relationship set.
- A maximum value **1** indicates that the entity participates in at most one relationship, while maximum value ***** indicates no limit. That is label **1..*** on an edge is equivalent to double line.

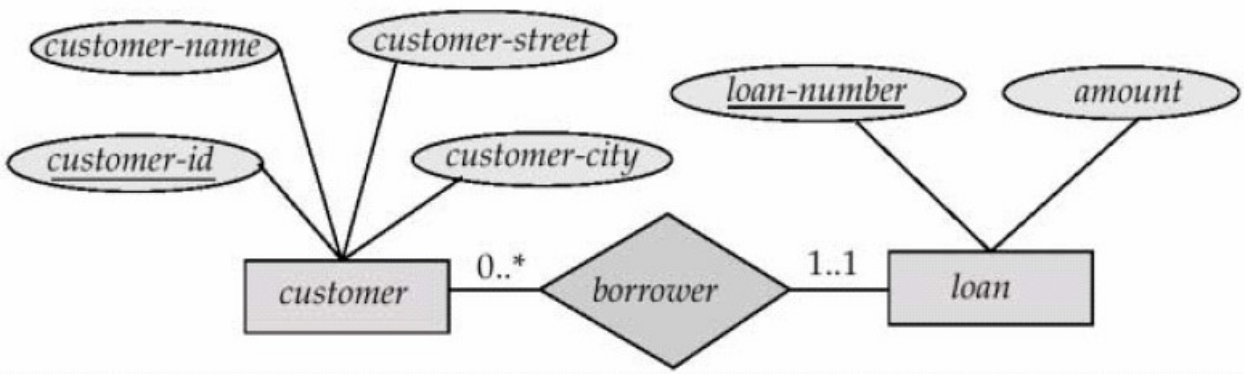


Figure: Cardinality limits on relationship sets

- Here, edge between **loan** and **borrower** has cardinality limit **1..1**. This indicates loan must have exactly one associated customer. The cardinality limit **0..*** on the edge from customer to borrower indicates that customer can have zero or more loans. Thus the relationship borrower is one to many from customer to loan and further the participation of loan in borrower is total.

Weak Entity Set vs. Strong Entity Set and their representation in E-R Diagram

- Entity set that does not have primary key known as **weak entity set** and entity set that has a primary key known as **strong entity set**.
- An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed as a **weak entity set.** An entity set that has a primary key is termed as **a strong entity set.**
- Let us consider entity set

  Payment= (payment_number,payment_date,payment_amount)

  - Here, payment numbers are typically sequence of numbers, starting from 1 and generated for each loan.
  - Thus although each payment entry is distinct, payments for different loan may share the same payment number.
  - Thus, this entity set does not have a primary key; **it is a weak entity set.**
- For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying or owner entity set**, using one of the key attribute of owner entity set.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
- An attribute of weak entity set that is used in combination with primary key of the strong entity set to identify the weak entity set uniquely is **called discriminator** of weak entity or **partial key.**
- The **primary key of weak entity set** can be composed by **combining primary key of identifying entity set and the weak entity set's discriminator**. For weak entity set payment primary key is **{loan_number,paymenr_number},** where *loan_number* is primary key of identifying entity set **loan**, and *payment_number* is discriminator of weak entity set **payment**.
- In E-R diagram,
  - **Weak entity set** is represented **by double rectangles**.
  - The **discriminator** of a weak entity set is represent by **underline attribute with a dashed line** and
  - **Double outlined diamond** represents **identifying relationship**.

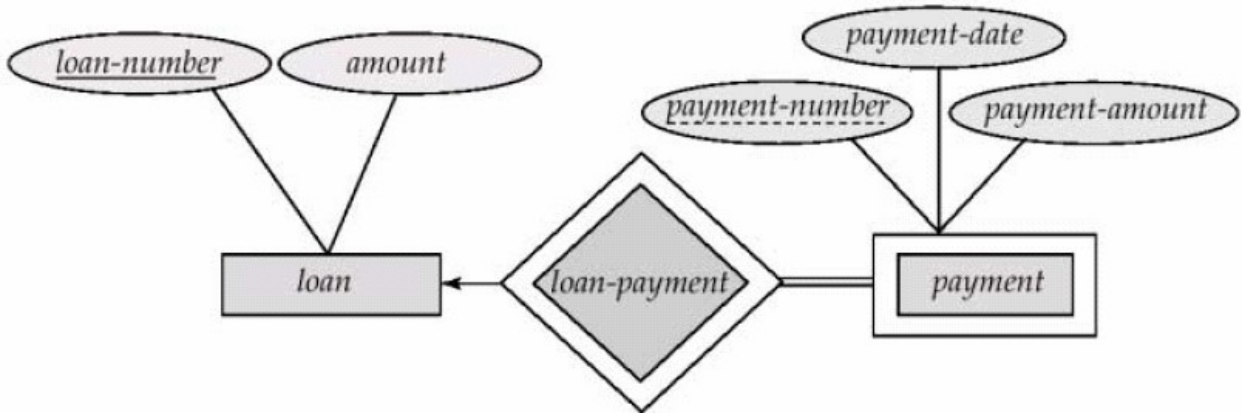


Figure: E-R diagram with weak entity set.

- In our example, **identifying entity set for weak entity set *payment* is *loan***, and a relationship loan-payment associates payment entities with their corresponding loan entities in **identifying relationship.**
- In the above figure*, payment-number* is **partial key** and *(loan-number, payment-number)* is **primary key** for payment entity set.

- An entity set that is not a weak entity set is termed a **strong entity set.**
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak entity set that it identifies.
- The relationship associating the weak entity set with the identifying entity set is called the identifying relationship

Assignment: Differentiate between strong and weak entity sets.

Extended E-R (EER) Model

- The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a subclass and super class, along with the concepts of specialization and generalization.
- EER model is required because :
  - ✓ The ER modeling concepts are not sufficient for representing new database applications, which have more complex requirements than do the more traditional applications.
  - ✓ Additional semantic data modeling concepts must be used to represent these requirements as accurately and clearly as possible.
- EER model concepts includes
  - ✓ All modeling concepts of basic ER
  - ✓ Additional concepts:
    - Sub classes/super classes
    - specialization/generalization
    - categories (UNION types)
    - attribute and relationship inheritance
  - ✓ These are fundamental to conceptual modeling
- The additional EER concepts are used to model applications more completely and more accurately
  - ✓ EER includes some object-oriented concepts, such as inheritance

Subclass and superclass

The class that is derived from another class is called a subclass. The class from which a subclass derives is called the superclass. The following figure illustrates these two types of classes:

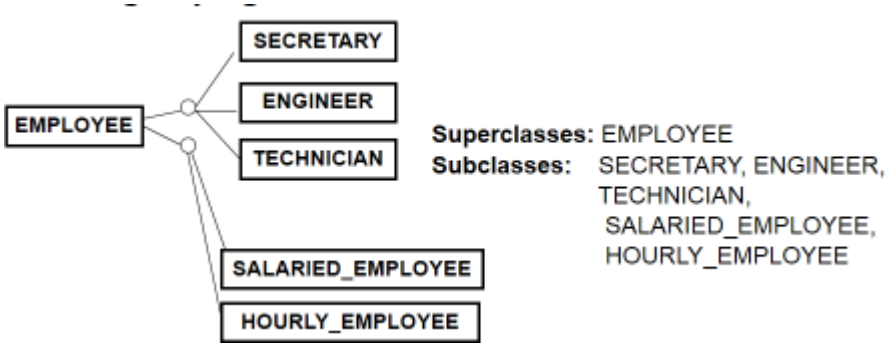


Figure: Superclass/subclass relationship

- ✓ An entity type may have additional meaningful sub-groupings of its entities. Example: EMPLOYEE may be further grouped into {SECRETARY, ENGINEER, TECHNICIAN, MANAGER, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE …}
- ✓ EER diagrams extend ER diagrams to represent these additional sub-groupings, called subclasses or subtypes. Each of these subgroups is called a subclass of the EMPLOYEE entity type.
- ✓ The EMPLOYEE entity type is called the superclass of each of these subclasses.
- ✓ The relationship between a superclass and any one of its subclasses is called a superclass/subclass or class/subclass or IS-A (IS-AN) relationship (e.g. e.g., EMPLOYEE/SECRETARY and EMPLOYEE/TECHNICIAN are two class/subclass relationships.
- ✓ Subclass entities have their own specific attributes. They also inherit all attributes and relationships of its superclass (subclasses can be considered as separate entity types).

Specialization and Generalization

Specialization

- ✓ The process of sub grouping within an entity set is called specialization.
- ✓ The process of defining a set of subclasses from a superclass is known as specialization.
- ✓ It follows top-down design approach.
- ✓ The set of subclasses that form a specialization is defined on the basis of some distinguishing characteristics of the entities in the superclass.

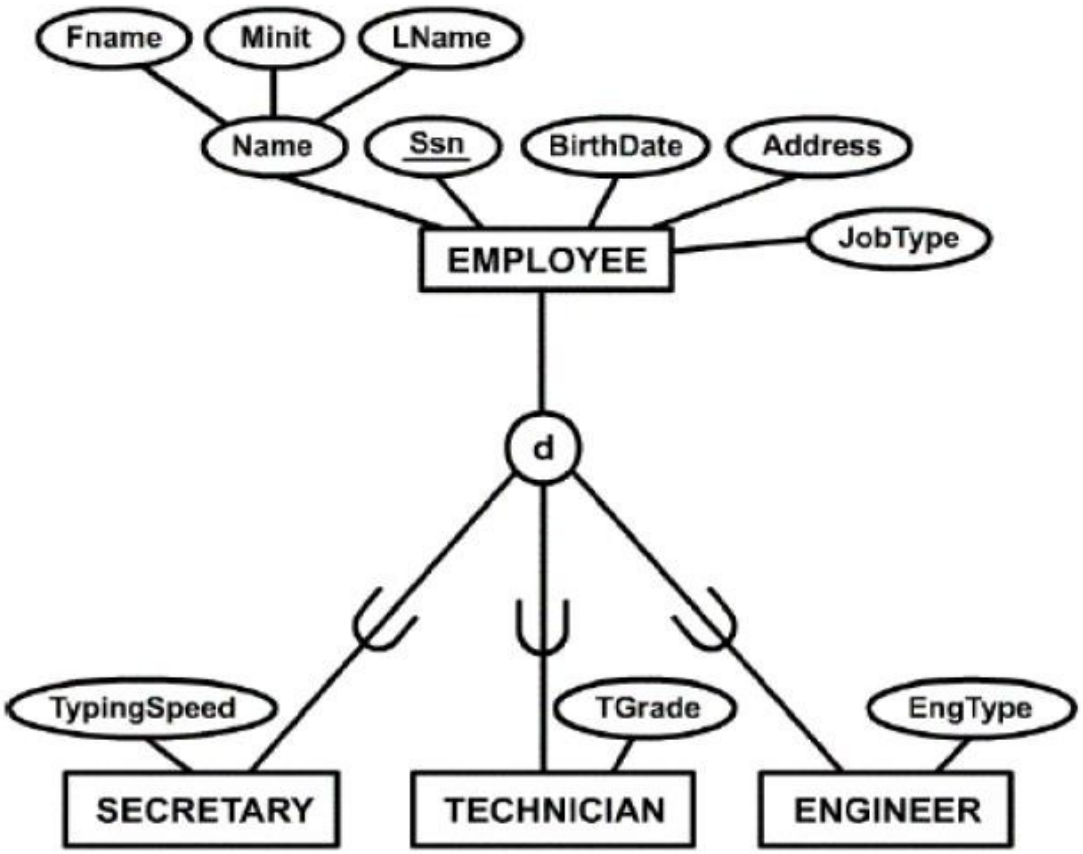
- ☞ {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based on the *job type* of each entity.

- ✓ The same entity type may have several specializations based on different distinguishing characteristics.
  - The EMPLOYEE entity type may have two specializations:
    - ❖ Based on the methods of pay:
      - {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}
    - ❖ Based on the type of job:
      - {SECRETARY, ENGINEER, TECHNICIAN}
- ✓ The specialization process allows us to do the following:
  - Define a set of subclass of an entity type
  - Establish additional specific attributes with each subclass
  - Establish additional specific relationship types between each subclass and other entity types or other subclasses

Following figure shows the Specialization of an Employee based on Job Type.

Notes:

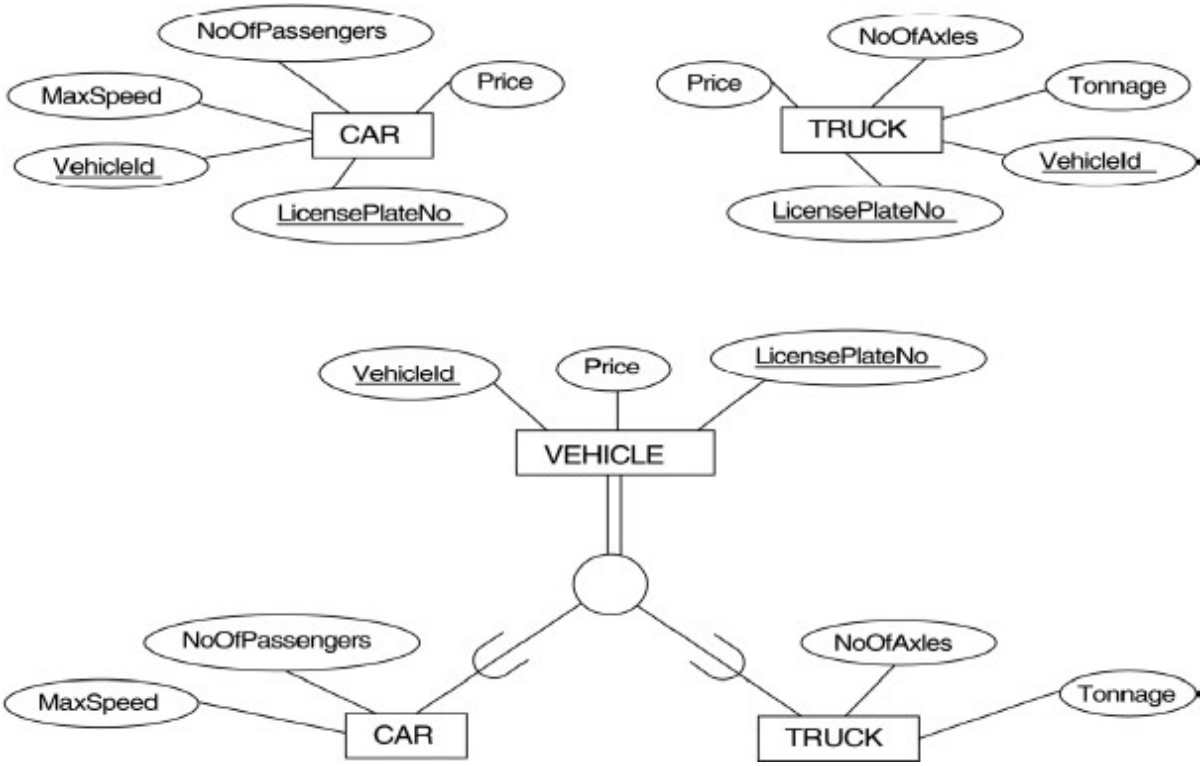
- ✓ In the following diagram the U-shaped symbol indicate that the subtype is a subset of the supertype.
- ✓ In EER diagram, disjoint-constraint is illustrated by placing the letter **d** inside the circle.
- ✓ In case overlap between subclasses is allowed, we place the letter **o** inside the circle. (**o** is default)



Generalization

- ✓ It is a bottom-up design process.
- ✓ Here, we combine a number of entity sets that share the same features into a higher-level entity set.
- ✓ The original classes become the subclass of the newly formed generalized superclass.
- ✓ The reason, a designer applies generalization is to emphasize the similarities among the entity sets and hide their differences.
- ✓ Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way. The terms specialization and generalization are used interchangeably.
- ✓ Example: CAR, TRUCK generalized into VEHICLE; both CAR, TRUCK become subclasses of the super class VEHICLE.
- ✓ We can view {CAR, TRUCK} as a specialization of VEHICLE

✓ Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK



Following figure is showing both specialization and generalization.

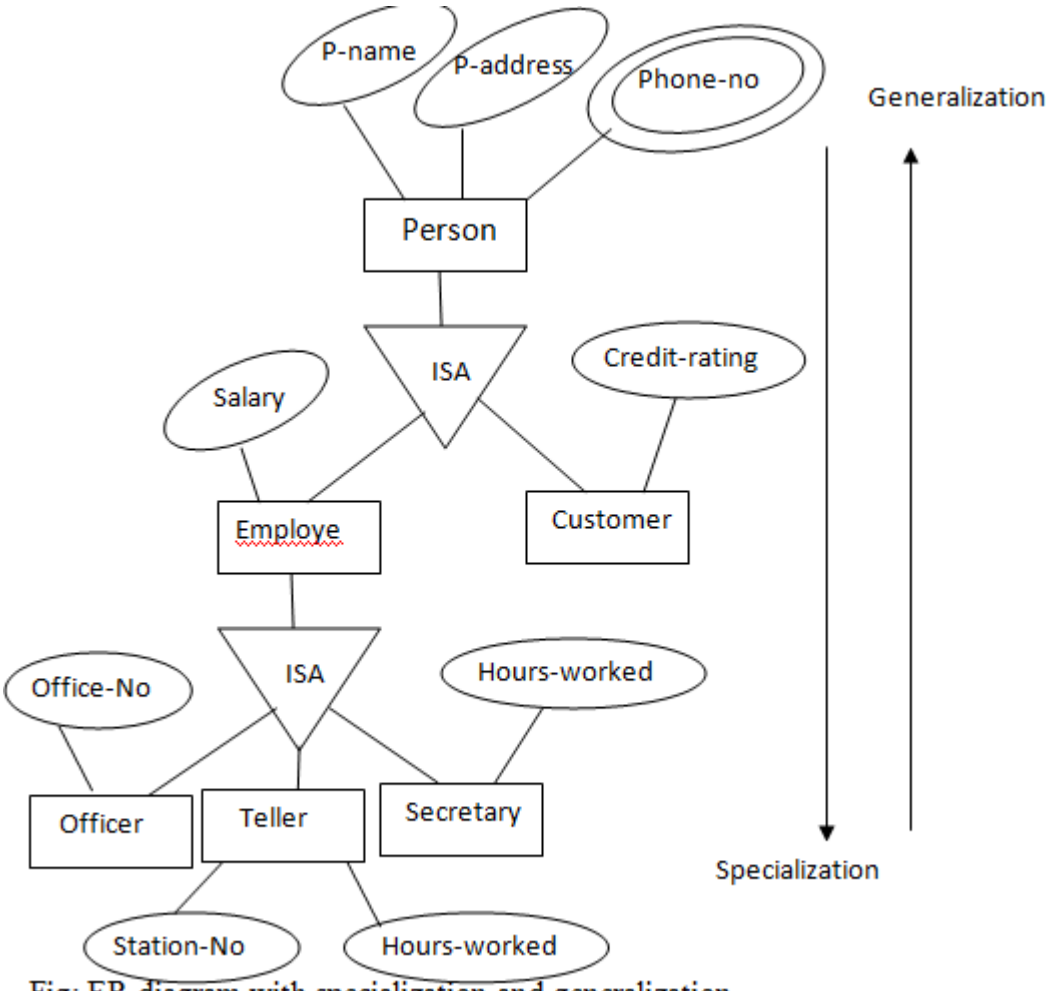


Fig: ER diagram with specialization and generalization

Constraints on Specialization and Generalization:

The constraints apply to a specialization/generalization may be categorized into three parts:

Part 1:

- Condition defined constraints
- User-defined constraints

Part 2:

- Disjoint Constraint
- Overlapping constraint

Part 3:

- Completeness Constraint or Participation constraint
  - ✓ Total generalization/Specialization
  - ✓ Partial generalization/Specialization

Condition defined constraints

If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses. Here, condition is a constraint that determines subclass members. For example, if the employee entity set has an attribute job-type, we can specify the condition of membership in the secretary subclass by a condition (job-type=”secretary”), which we call defining predicate of the subclass.

User-defined constraints

If no condition determines membership, the subclass is called user-defined. Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass. Membership in the subclass is specified individually for each entity in the superclass by the user.

Disjoint Constraint

It specifies that the subclasses of the specialization must be disjoint. Here an entity can be a member of at most one of the subclasses of the specialization and it is represented by **d** in EER diagram.

Overlapping constraint

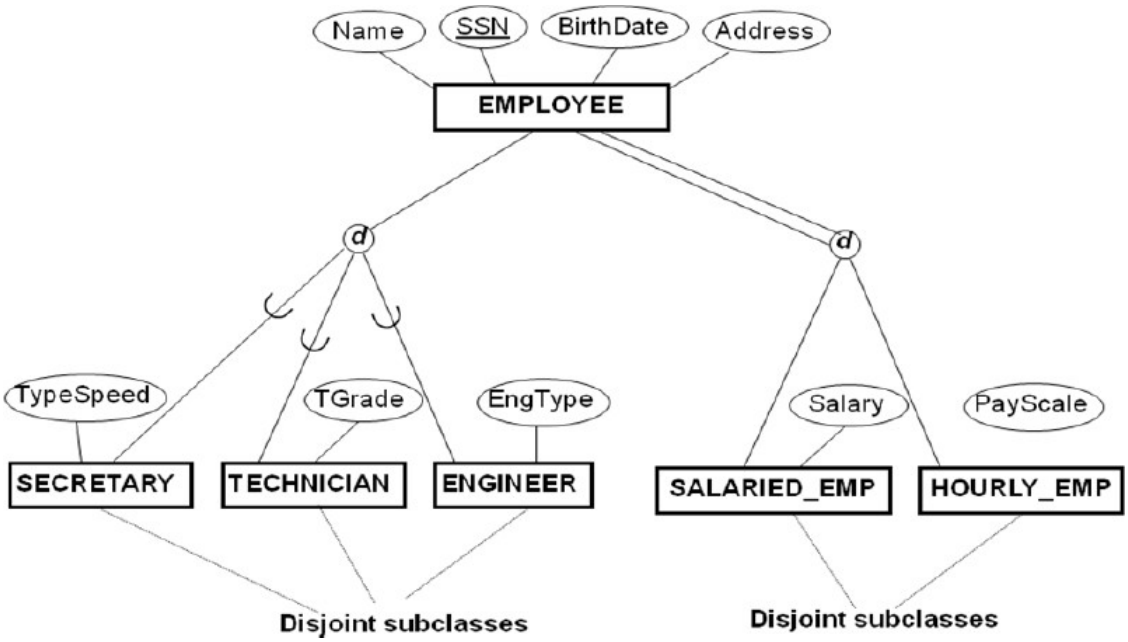
It specifies that the subclasses are not constrained to be disjoint, i.e., the same entity may be a member of more than one subclass of the specialization and it is represented by **o** in EER diagram.

Completeness Constraint

Total participation constraint →specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization. It is represented by double line in EER diagram.

Partial participation constraint → It allows every entity in the superclass may not belong to a

subclass and shown in EER diagrams by a single line. e.g. if some EMPLOYEE entities, (for example, sales representatives) do not belong to any of the subclasses {SECRETARY, ENGINEER, TECHNICIAN}, then the specialization is partial.



Note: Generalization usually is total because the super class is derived from the subclasses

The differences between the specialization and generalization

- The specialization process corresponds to a top-down conceptual refinement process during conceptual schema design.
  - we typically start with an entity type and then define subclasses of the entity type by successive specialization;
- The generalization process corresponds to a bottom-up conceptual synthesis.
  - we typically start with an entity type of subclasses and then define superclasses of the entity type by successive generalization.

UNION Types or Categories:

- It is possible that single superclass/subclass relationship has more than one super-class representing different entity types. In this case, the subclass will represent a collection of objects that is (a subset of) the UNION of distinct entity types; we call such a subclass a union type or a category.
- A category has two or more superclasses that may represent distinct entity types, whereas non-categorty superclass/subclass relationships always have a single superclass.
- Two categories in car registration database.

  – OWNER is a subclass of the union of PERSON, BANK, and COMPANY

  – REGISTRERED_VEHICLE is a subclass of the union of CAR and TRUCK

- An entity that is a member of OWNER must exist in only one of the superclass.
- Attribute inheritance works more selectively in the case of categories.
  - Example: Database for vehicle registration, vehicle owner can be a person, a bank (holding a lien on a vehicle) or a company.
  - Category (subclass) OWNER is a subset of the union of the three superclasses COMPANY, BANK, and PERSON
  - A category member must exist in at least one of its superclasses

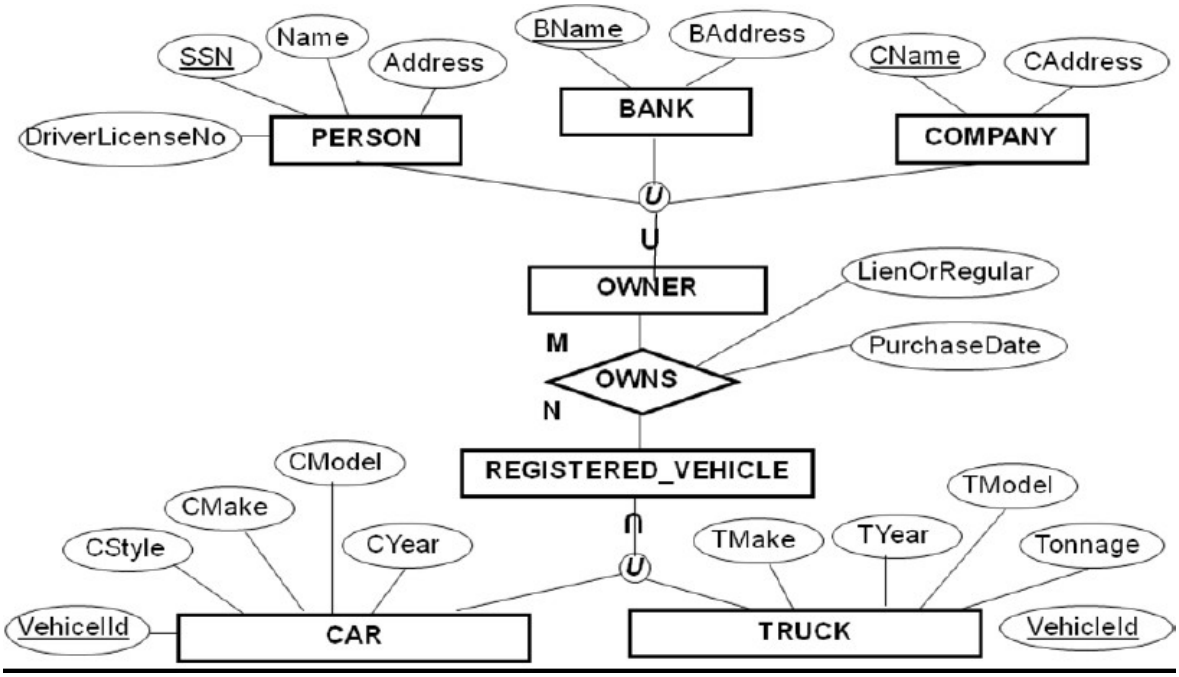


Figure: Two categories (union types): OWNER and REGISTERED_VEHICLE

Aggregation

- One of the limitations of E-R model is that it cannot express relationship among relationships. To illustrate this, let us consider quaternary relationship manages among employee, branch, job and manager. Its main job is to record managers who manages particular job/task perform by particular employee at particular branch.

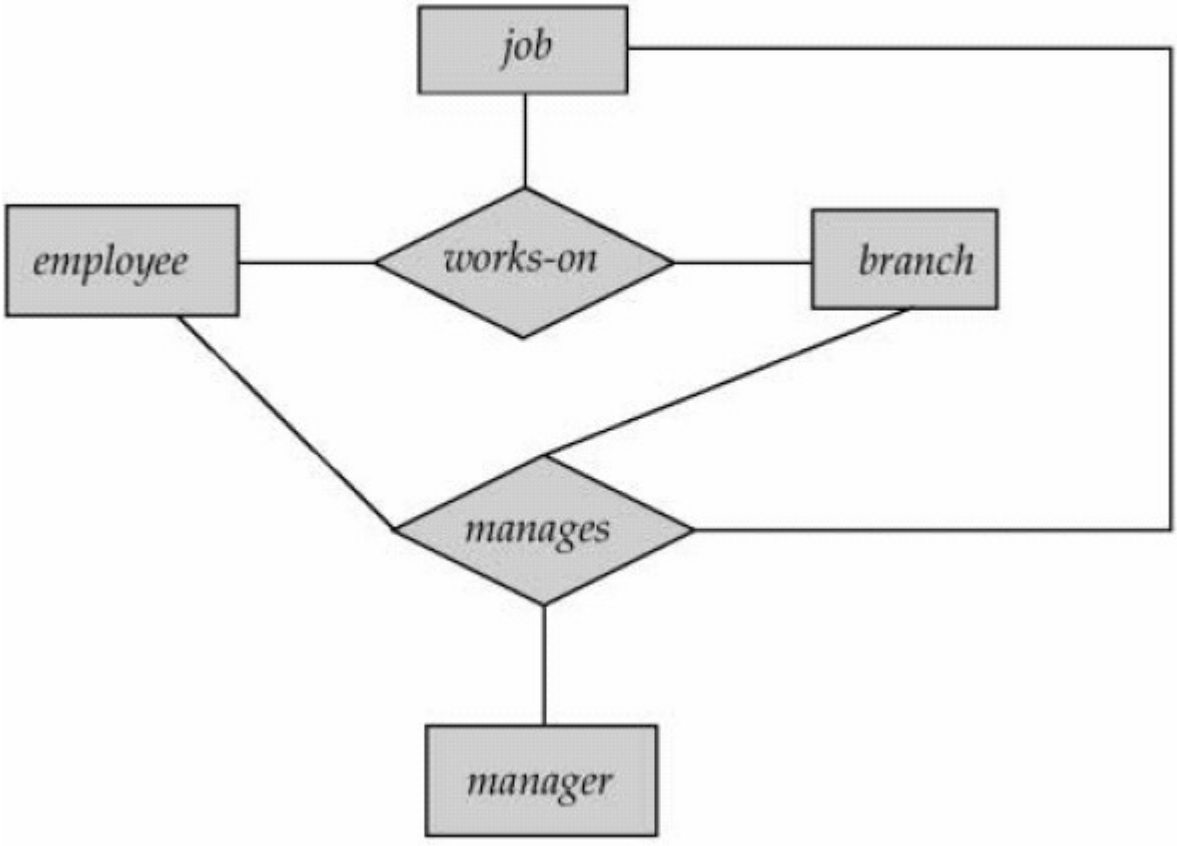


Figure: E-R Diagram with redundant relationship

This quaternary relationship is required since binary relationship between manager and employee can not represent required information. This E-R diagram is able to represent the required information but information are redundant since every employee, branch and job exist both relationship set "work-on" and "manages".

- Here aggregation is better to represent such information.
- Aggregation is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher-level entity sets and can participate in relationships.

- In our example, it treats relationship set work-on (including entity set employee, branch and job) as entity set. So now we can create binary relationship set "manages" between work-on and manager. This removes redundant information.

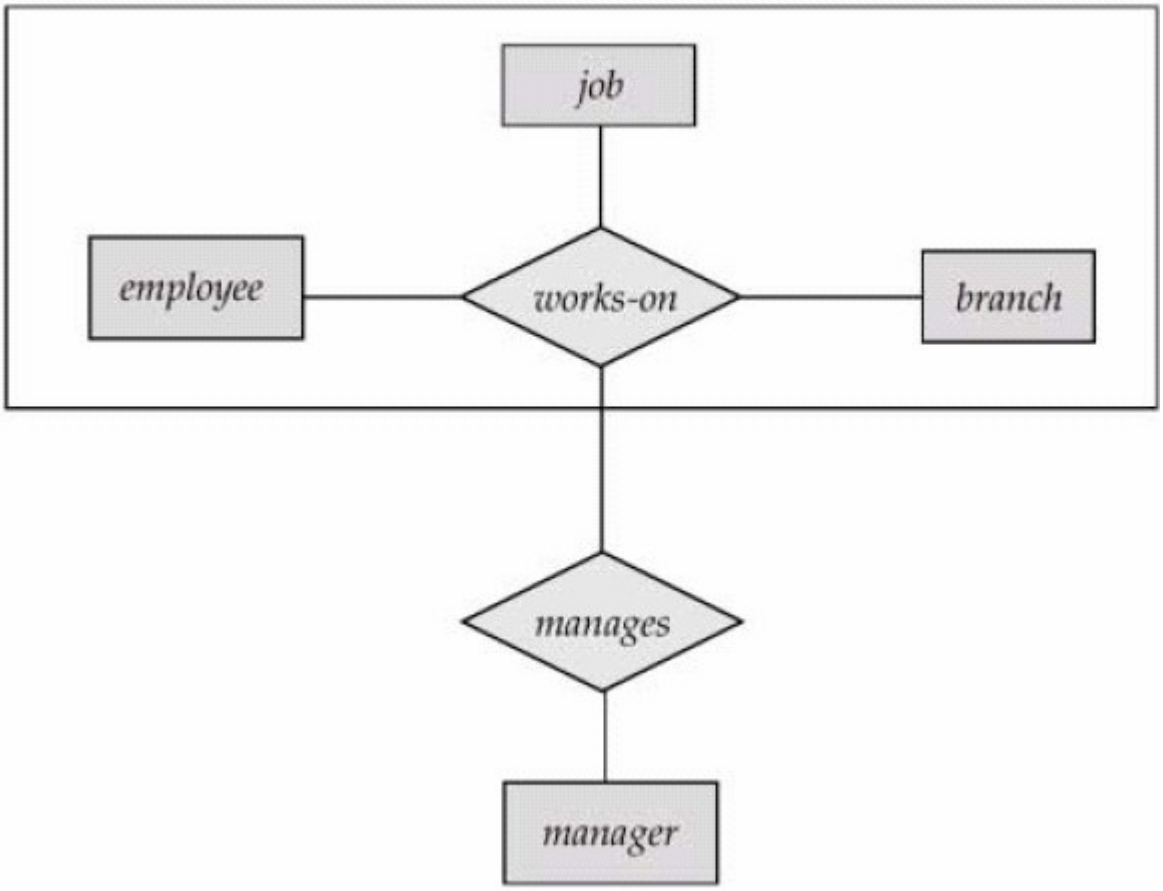


Figure: ER Diagram with aggregation

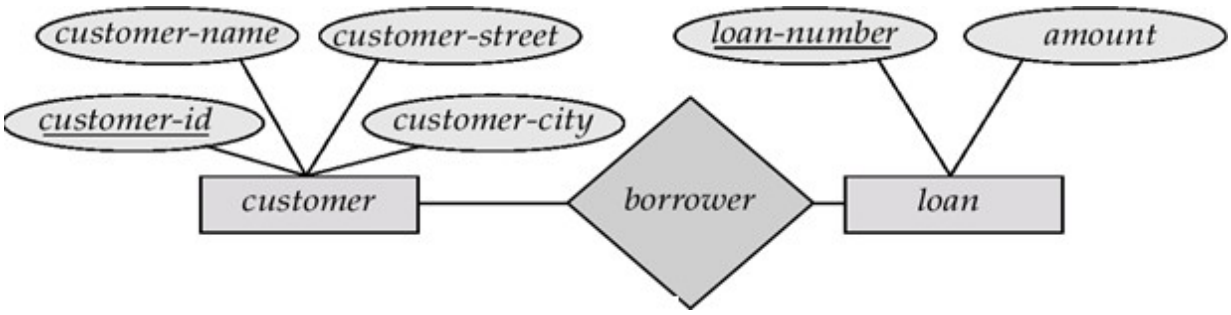
Reducing E-R Diagrams to Tables

- We can represent the E-R database schema by a set of tables.
- Each entity sets and each relationship sets in E-R schema can be represented by their corresponding tables.
- Each attributes of entity sets and relationship sets are map as columns of their corresponding tables.
- Similarly constraints specified in E-R diagram such as primary key, cardinality constraints etc are mapped to tables generated from E-R diagram.
- In fact, representing E-R schema into tables is converting E-R model of database into relational model.

- To reduce given ER diagram into table simply we create a table for each entity set and for each relationship sets. And that are assigned with the name of the corresponding entity set or relationship set as table name. Generally the number of attribute of an entity set or relationship set equal to the degree of a corresponding table (fields of a table).
- To reduce given ER diagram into tables normally we divide ER diagram into following sections:
    - **Strong entity sets**
    - **Weak entity sets**
    - **Relationship sets**
    - **Mapping of multivalued attributes**
    - **Mapping of Composite Attributes**
    - **Mapping of N-ary relationship types**

Tabular representation of Strong Entity Sets

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Choose one of the key attributes of E as the primary key for R. Consider an E-R diagram as given below



- For the tabular representation of the entity sets customer and loan of the given E-R diagram, we can represent the entity set customer by a table called customer, with four columns named customer-id, customer-name, customer-street and customer-city . Similarly entity set loan can be represented by table called loan with two columns namely, loan-number and amount.

customer

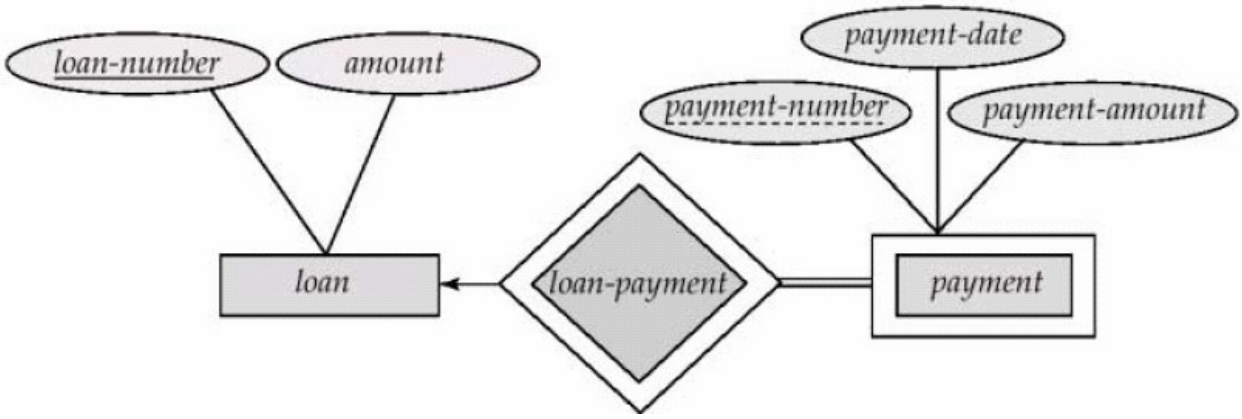
Customer-id	Customer-name	Customer-street	Customer-city
C-11	Bhupi	Balkhu	Kathmandu
C-12	Umesh	Katan	Mahendranagar
C-13	Aayan	Kuleshor	Kathmandu

loan

Loan-number	Amount
L-09	30000
L-07	60000
L-13	12000

Reducing weak entity sets into tables

- For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- In addition, include primary key(s) of owner entity type.
- The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
- To illustrates this consider the entity set payment in the following E-R diagram



payment

loan_number	payment_number	payment_date	Payment_amount
L12	5	2075-08-16	1000
L15	7	2075-09-15	2500
L15	8	2075-10-07	1500

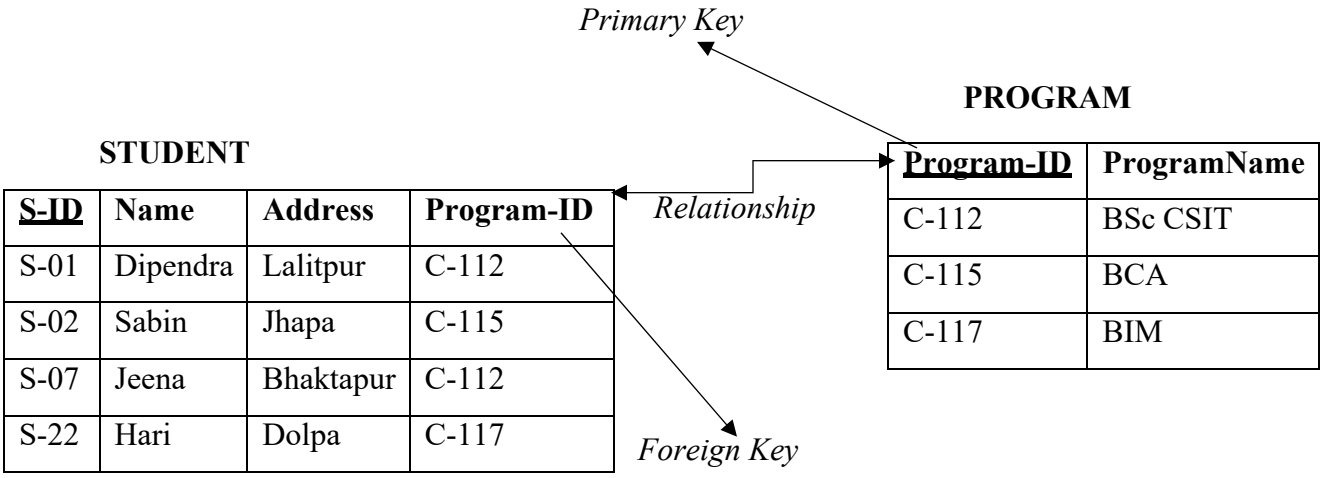
loan

loan_number	amount
L12	10000
L15	20000
L15	30000

Reducing Relationship sets into tables

What is foreign key??

- A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce relationship between two relations (table). A set of attributes that references primary key of another table is called foreign key. For example, if a student enrolls in program then program-id (primary key of relation program) can be used as foreign key in student relation.
- A FOREIGN KEY in one table points to a PRIMARY KEY in another table



PROGRAM

Program-ID	ProgramName
C-112	BSc CSIT
C-115	BCA
C-117	BIM

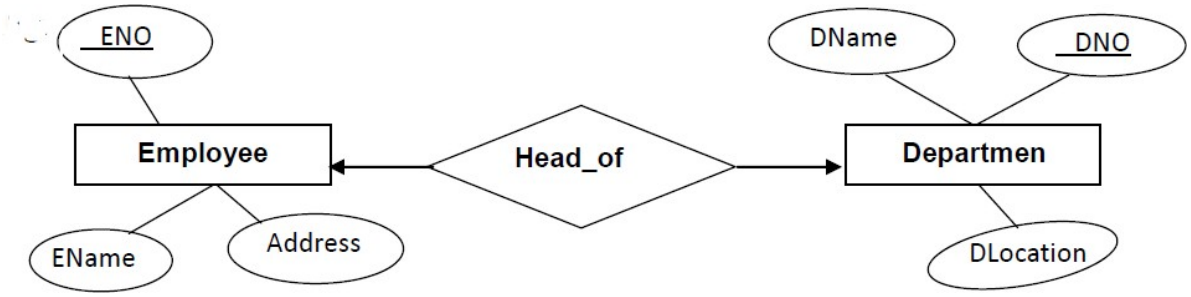
STUDENT

S-ID	Name	Address	Program-ID
S-01	Dipendra	Lalitpur	C-112
S-02	Sabin	Jhapa	C-115
S-07	Jeena	Bhaktapur	C-112
S-22	Hari	Dolpa	C-117

Figure: Primary Key and Foreign Key

Mapping of Binary 1:1 relation types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations-S, say- and include a foreign key in S which is the primary key of T. It is better to choose an entity type with *total participation* in R in the role of S.



Employee

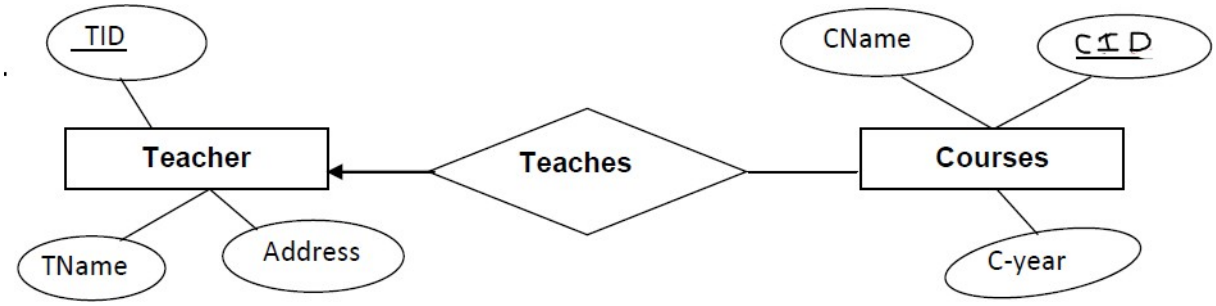
ENO	***EName***	***Address***	***DNO***
1	Bhupi	Ktm	D1
2	Aayan	Dang	D2
3	Anju	Palpa	D3
4	Umesh	Mugu	D4
5	Ramesh	Rolpa	D5

Department

DNO	***DName***	***DLocation***
D1	Maths	Ktm
D2	Computer	Kanchanpur
D3	Nepali	Palpa
D4	Physics	Chitwon
D5	Biology	Rampur

Mapping of Binary 1: N relation types

For each regular binary 1: N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.



Courses

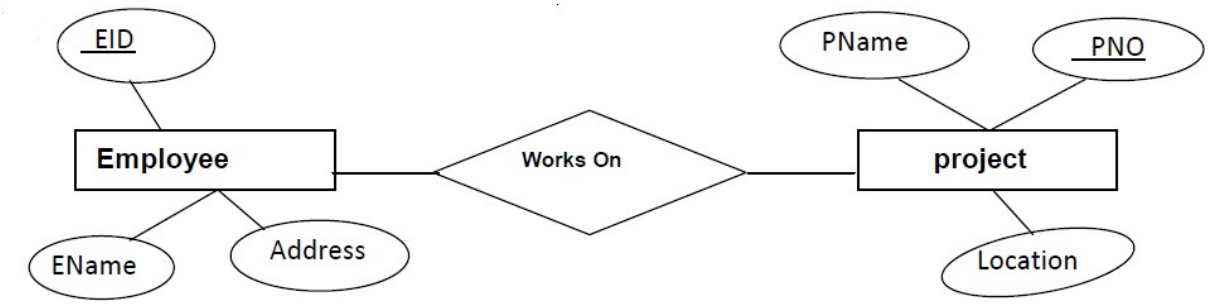
CID	***CName***	***C-year***	***TID***
1	TOC	2011-1-1	T1
2	DAA	2012-1-1	T2
3	SAD	2011-6-1	T1
4	Java	2010-1-1	T3
5	C++	2012-6-1	T2

Teacher

TID	***TName***	***Address***
T1	Deepak	Ktm
T2	Bhupi	Kanchanpur
T3	Arjun	Pokhara

Mapping of Binary M: N relationship types

- For each regular binary M:N relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.



Employee

EID	*EName*	*Address*
1	Bhupi	Ktm
2	Aayan	Dang
3	Anju	Palpa
4	Umesh	Mugu

Works

EID	*PNO*
1	P1
2	P2
3	P3
1	P2
2	P1
4	P2

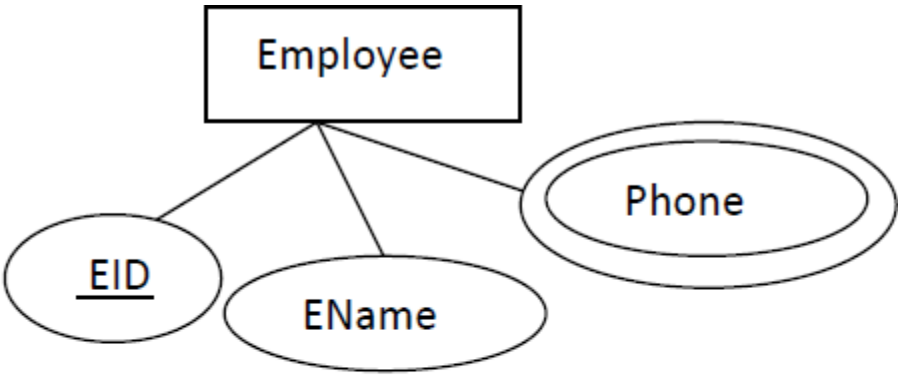
Project

PNO	*PName*	*Location*
P1	Kulekhani	Ktm
P2	Chameliya	Darchulla
P3	Melamchi	Kabhre

Mapping of multivalued attributes

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

Example: The relation Employee_phone is created. The attribute phone represents the multivalued attribute. The primary key of R is the combination of {EID, phone}.

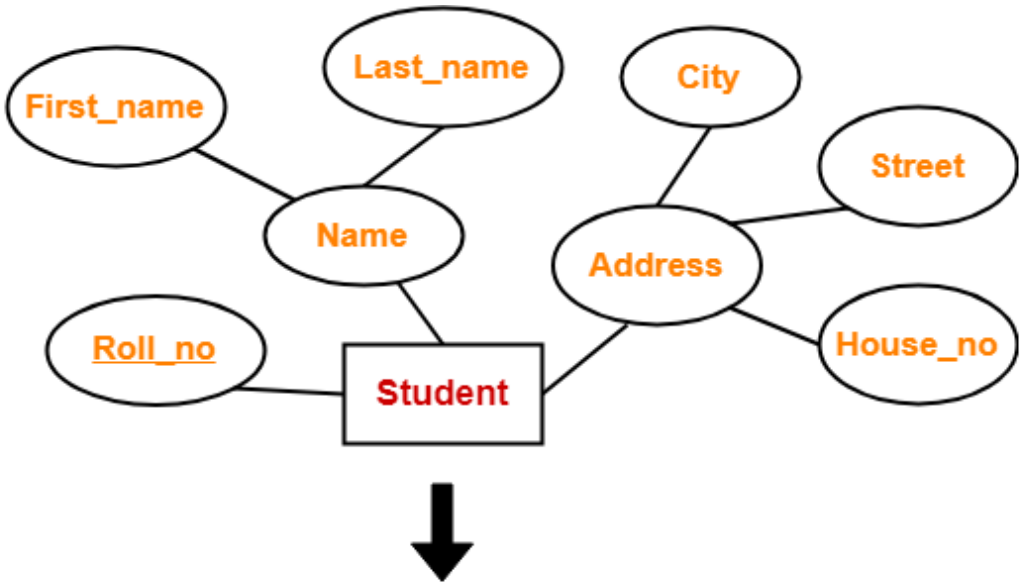


Employee_phone

EID	*EName*	*Phone*
1	Bhupi	9849148483
1	Bhupi	9813334505
2	Umesh	9803334505

Mapping of Composite Attributes

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.



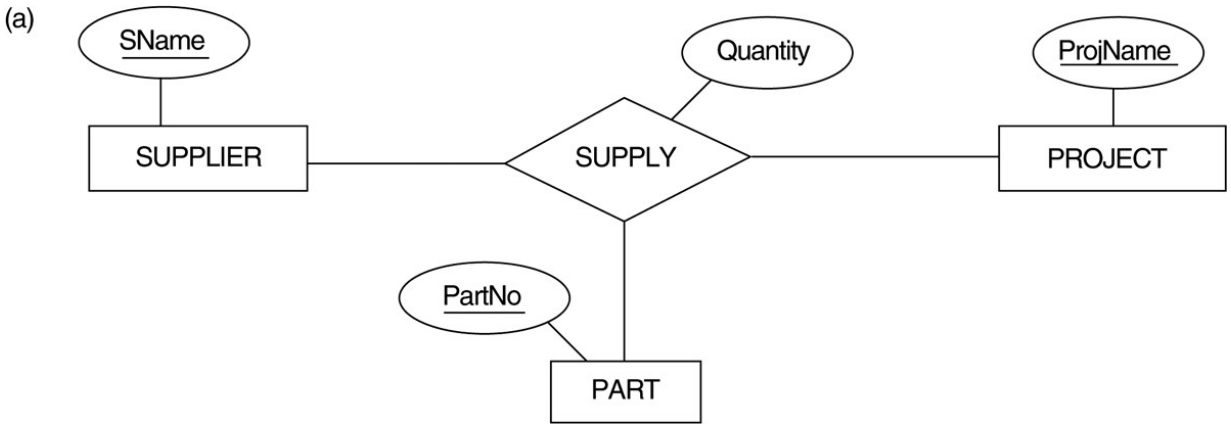
Roll_no	First_name	Last_name	House_no	Street	City

Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

Mapping of N-ary relationship types

- For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

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



SUPPLIER

SNAME	. . .

PROJECT

PROJNAME	. . .

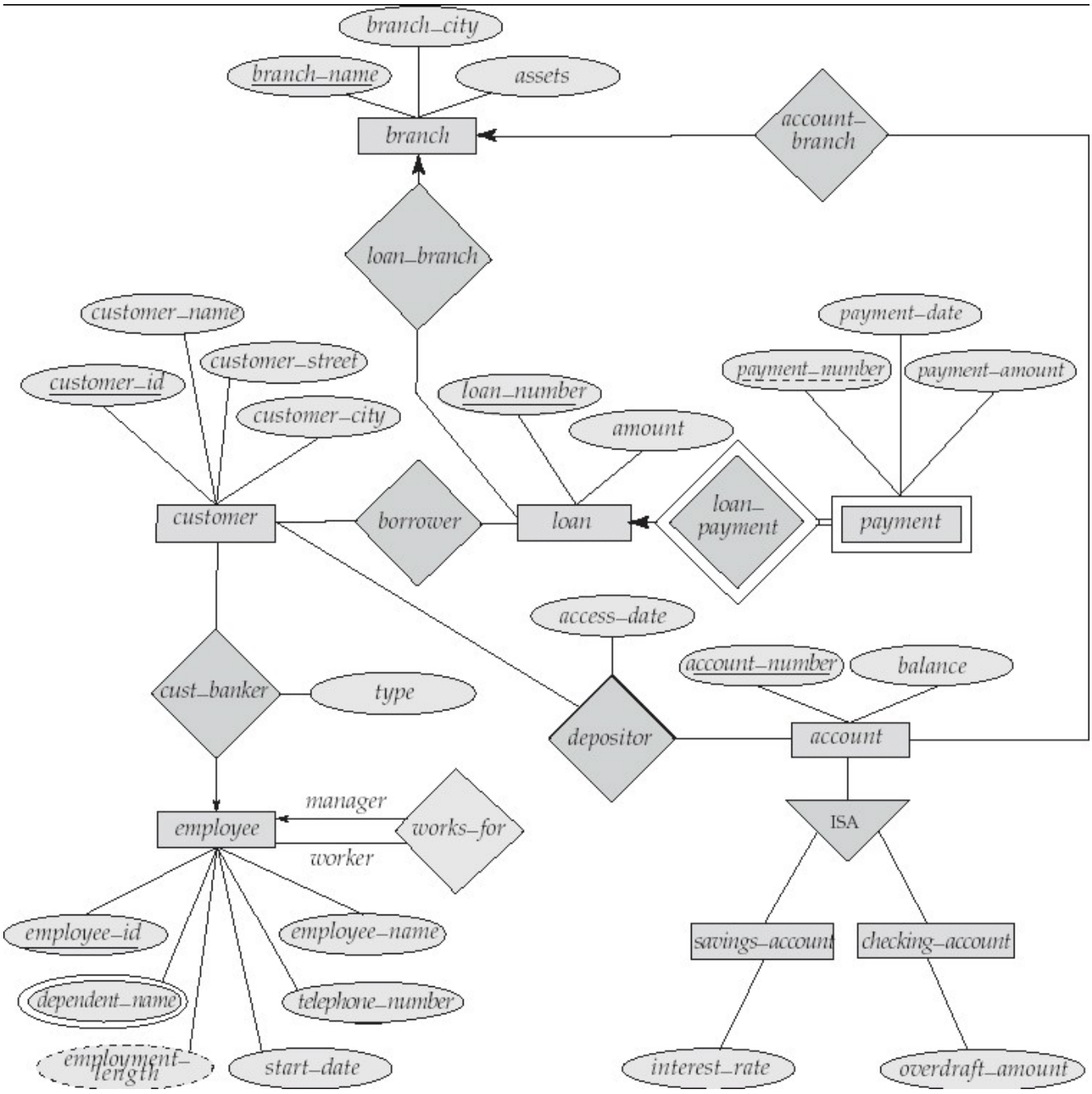
PART

PARTNO	. . .

SUPPLY

SNAME	PROJNAME	PARTNO	QUANTITY

E-R Diagram for a Banking Enterprise



Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents

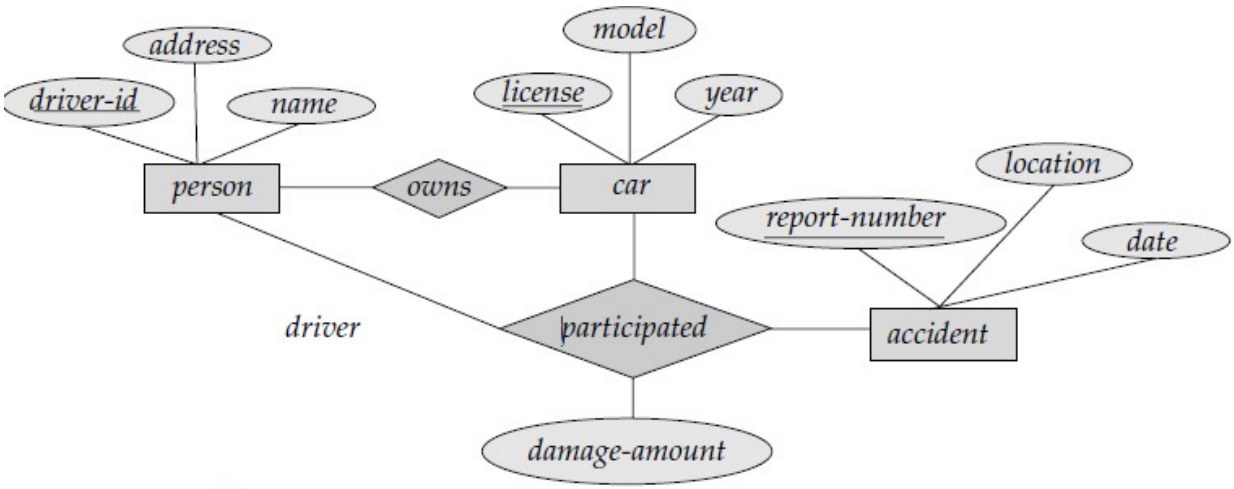


Figure E-R diagram for a Car-insurance company.