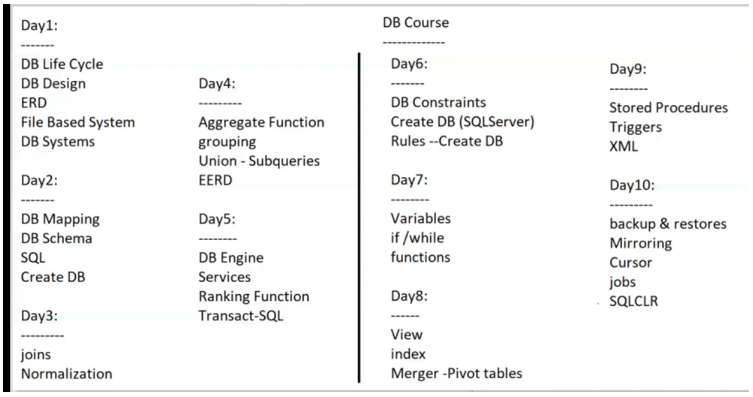
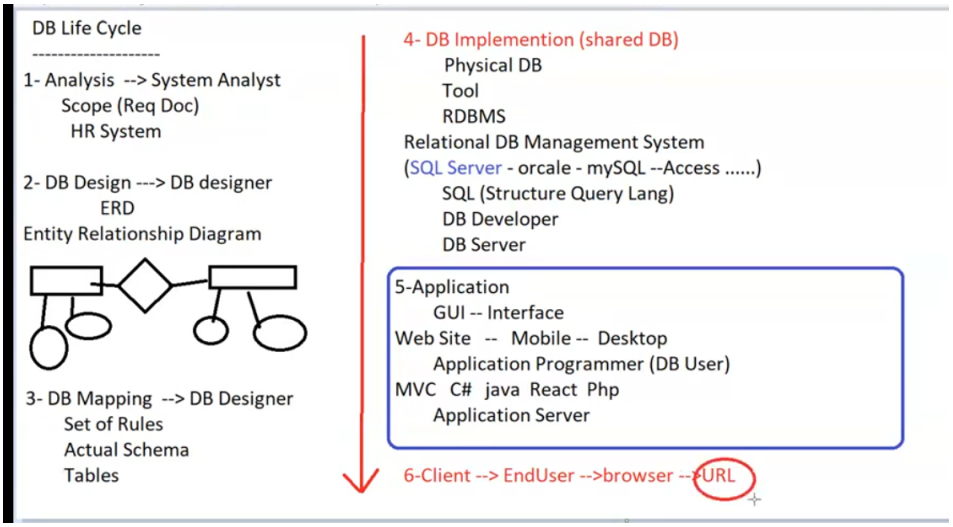
SQL introduction, DB Design, ERD, File Based System, DB system (Day 1)



DB life cycle

- we must understand client requirements to start working in database this called system analysis.
- DB design ⇒ transfer requirements to shape called ERD(Entity relationship diagram) , the one who do this called DB designer.
- Entity is the object who i should store data about it inside the system.

  for example: school⇒(courses, students, teachers...etc.).
- DB mapping ⇒ set of roles provides actual schema , Tables. who do this called DB designer.
- DB implementation (physical DB): collection of tools⇒RDBMS(SQL server ,oracle ,MySQL) ⇒ DB developer.
- After downloading sql server ,your computer became DB server.
- Any DB must be centralized and shared.
- If you have two different DB on different servers this called inconsistent DB.
- Application (GUI): interact with database.



File based system

- Delimited file: inside it(Amr,1,2,Nasr....).
- Fixed width file: read and write based on number of bites.

Disadvantages of read and write from files in database

1- Difficaluty of search.

2- Low performance.

3- Separated copies (every developer take copy and the others can not see the updates).

4- No relationships.

5- No DB integrity.

6- DB redundancy.

7- Long development time.

8- Security & permissions.

9- Constrains and rules.

10- No data quality every data received as text.

11- Manual backup & restore.

12- No standard.

13- Different integrations.
>Database system⇒ tables & relationships : database, DBMS, DB system.

Advantages of DB System

1- one standard.

2- metadata (data about data : discretion of data) and data.

3- columns has data types.

4- primary key.

5- foregin key.

6- centralized DB.

strong entity: department, weak entity: entity that became not interest after deleting member for example entity family.

Disadvantages of DB System

- It needs expertise to use
- DBMS itself is expensive
- The DBMS may be incompatible with any other available DBMS

Data Base Users

1. **Database Administrator (DBA)**:

   The DBA is responsible for the overall management of the database. They handle tasks such as database security, performance tuning, backup and recovery, and user access management. The DBA ensures that the database operates efficiently and that data is secure and available.

2. **System Analysts**:

   System Analysts work on analyzing and designing systems that use databases. They gather and define requirements for database applications, helping ensure that the system will meet the needs of its users and integrate well with other IT infrastructure.

3. **Database Designer**:

   A Database Designer focuses on creating the logical and physical structure of the database. They are responsible for defining the schema, tables, relationships, constraints, and indexes based on the requirements provided by system analysts and stakeholders.

4. **Database Developer**:

   Database Developers are responsible for writing and optimizing the SQL queries that interact with the database. They build the database applications, stored procedures, functions, triggers, and other programming objects to ensure that data can be accessed and manipulated efficiently.

5. **Application Programmers**:

   Application Programmers write the code for applications that interface with the database. They focus on building the front-end or middleware that communicates with the database, ensuring that the right data is presented to the end-users or passed between systems.

6. **BI & Big Data Specialist (Data Scientist)**:

   BI (Business Intelligence) Specialists and Data Scientists focus on extracting insights from the data stored in databases. They work with large datasets, applying analytical and statistical methods to uncover trends, patterns, and actionable intelligence. They often use tools and technologies for data mining, machine learning, and reporting.

7. **End Users**:

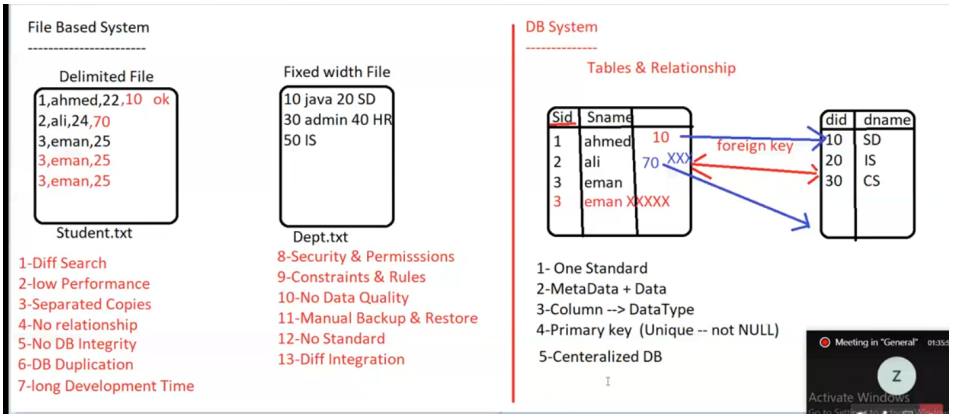
   End Users are the individuals who interact with the database through applications. They perform various tasks, such as querying, updating, and reporting data, depending on their role in the organization. They are typically shielded from the complexities of the database structure and focus on using the data to meet their needs.

➔ Basic Definitions

Database: A collection of related data.

Database Management System (DBMS): A software package/ system to facilitate the creation and maintenance of a computerized database.(model introduced in 1970 IBM but RDBMS appears in1980)

Database System: The DBMS software together with the data itself. Sometimes, the applications are also included. (Software + Database)



Entity Relationship Diagram Concepts

⇒ Entity-Relationship Diagram (ERD)

identifies information required by the business by displaying the relevant entities and the relationships between them.

Basic constructs of the ER model:

⇒ Entity

An **entity** is any object, person, place, event, or concept about which data is stored in a database. Entities represent real-world things or concepts and are usually represented as tables in a database.

- **Examples**:
  - A **student** in a school database.
  - A **product** in an e-commerce database.
  - An **employee** in a company's HR database.

In ER (Entity-Relationship) modeling, entities are typically depicted as rectangles.

- **Type**: Strong Entity & Weak Entity

▼ **Strong Entity Vs Weak Entity**

1. Strong Entity:

- A **strong entity** (or regular entity) is an independent entity that can be uniquely identified by its own attributes, without needing any other entity.
- It has a **primary key** that uniquely identifies each instance of the entity.
- Strong entities typically represent real-world objects that exist independently, such as a **person**, **product**, or **department**.
- They are represented by a **single rectangle** in Entity-Relationship (ER) diagrams.

Example:

- **Employee**:
  - Attributes: `EmployeeID`, `Name`, `Position`.
  - The `EmployeeID` is the primary key and uniquely identifies each employee.

2. Weak Entity:

- A **weak entity** is an entity that cannot be uniquely identified by its own attributes alone. It relies on a **strong entity** for its identification.
- Weak entities have a **partial key** (also called a **discriminator**), but they depend on the **primary key** of a related strong entity to form a complete primary key.

- They often represent objects or concepts that exist only in the context of a strong entity, such as **dependents of an employee** or **line items in an invoice**.
- Weak entities are represented by a **double rectangle** in ER diagrams, and their relationship to the strong entity is depicted by a **double diamond**.

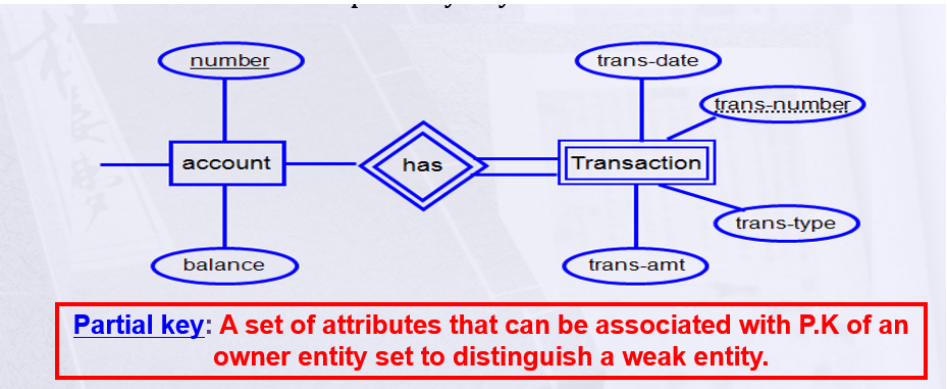
Example:

- **Dependent** (for an employee):
  - Attributes: `DependentName`, `Relation`.
  - The `Dependent` entity cannot be identified by `DependentName` alone. It requires the `EmployeeID` of the related **Employee** entity to form a complete identifier, such as a composite key: `(EmployeeID, DependentName)`.

Key Differences:

Feature	Strong Entity	Weak Entity
Independence	Can exist independently and be uniquely identified by its own attributes.	Depends on a strong entity for identification.
Primary Key	Has a primary key (unique identifier) of its own.	Does not have a primary key; has a partial key and relies on the strong entity's primary key.
Representation in ER Diagram	Single rectangle.	Double rectangle (to indicate its dependence).
Example	Employee, Product, Course.	Dependent, Invoice Line Item.

In summary, a **strong entity** exists independently and has its own unique identifier, while a **weak entity** depends on a strong entity for identification and cannot exist without it.



⇒ Attributes

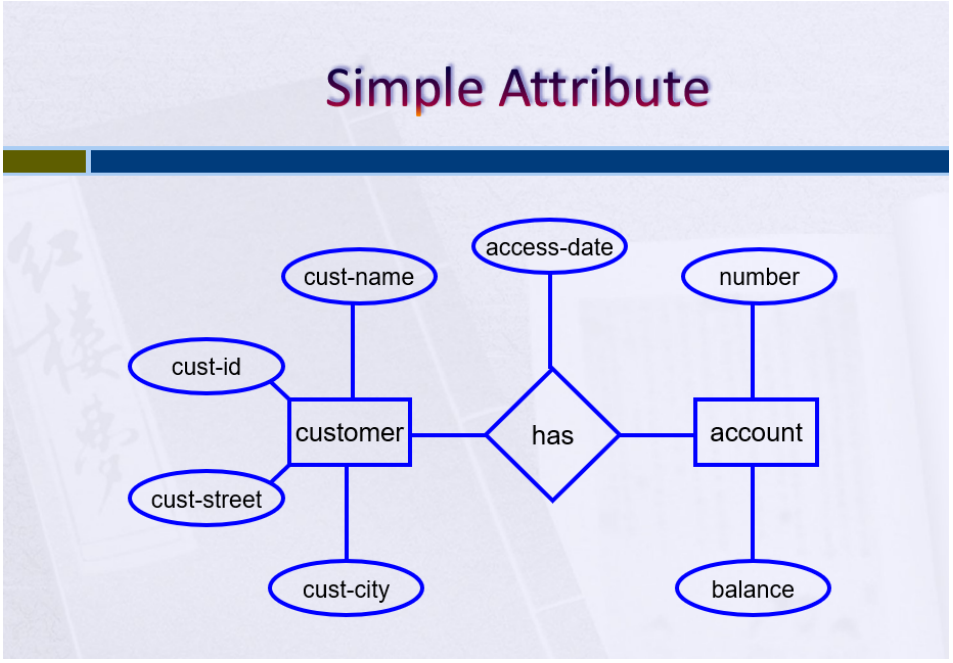
Attributes are the properties or characteristics that describe an entity. Each entity has a set of attributes that store specific data about the entity. In a relational database, attributes are represented as fields or columns in a table.

- **Examples**:
  - For the **student** entity: `StudentID`, `Name`, `DateOfBirth`, `Major`.
  - For the **product** entity: `ProductID`, `ProductName`, `Price`, `Category`.
  - For the **employee** entity: `EmployeeID`, `Name`, `Position`, `HireDate`.

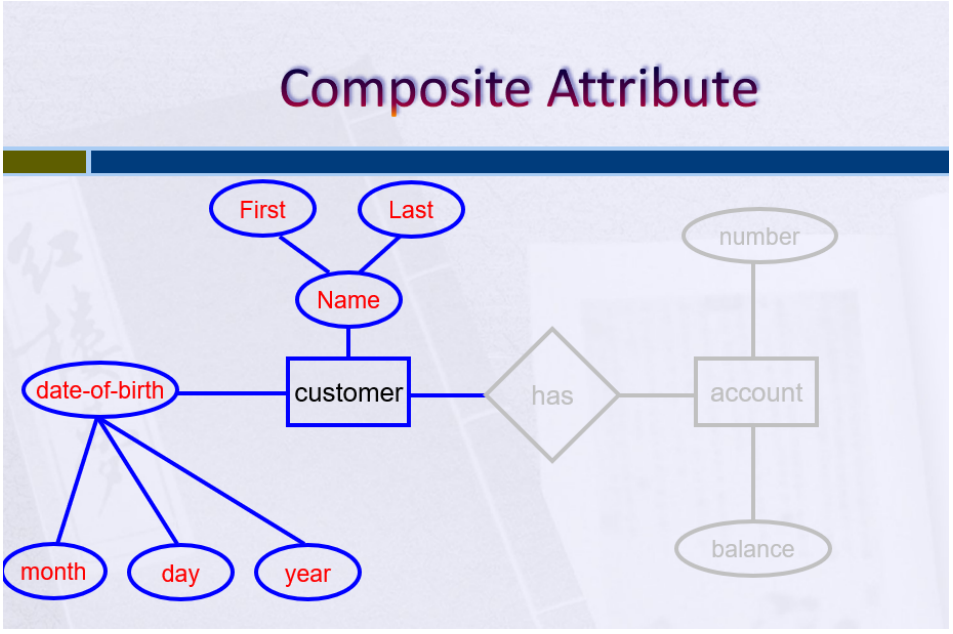
Attributes can have different types, such as integers, strings, dates, etc.

- **Type**:

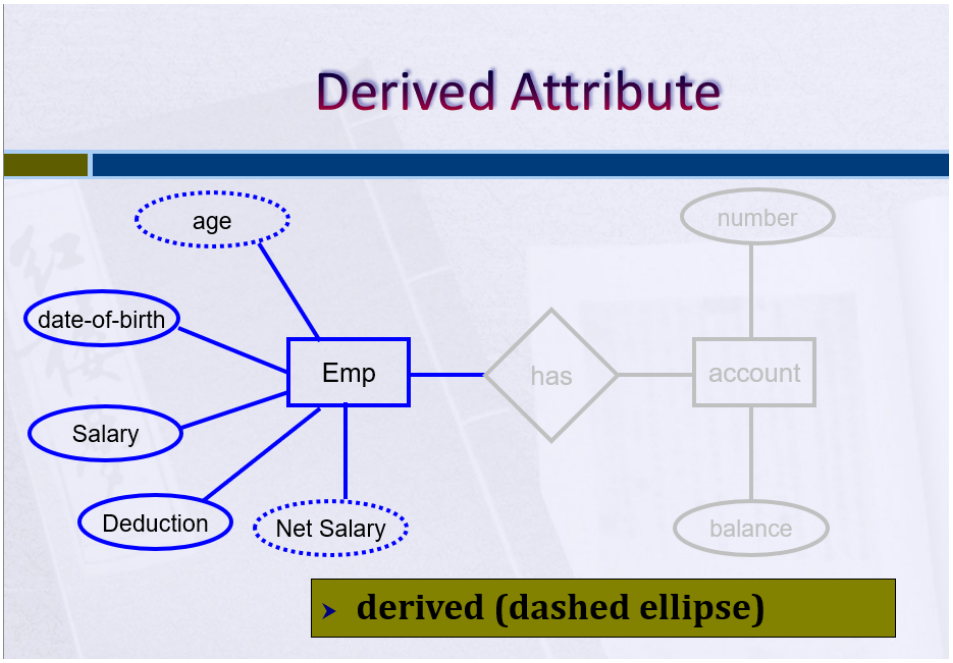
1. simple: city



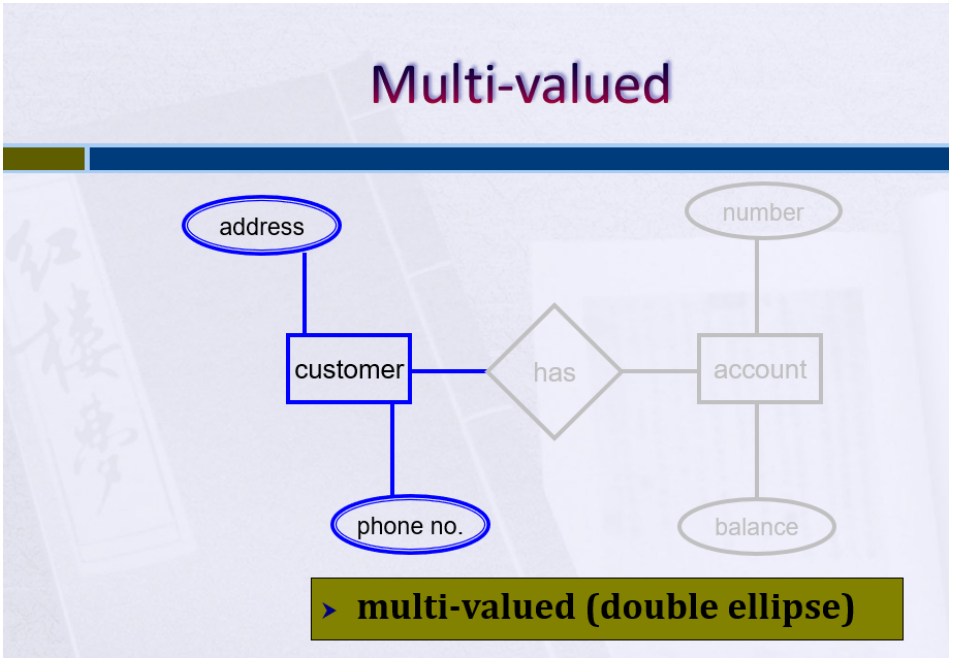
1. composite: name (first, last name)



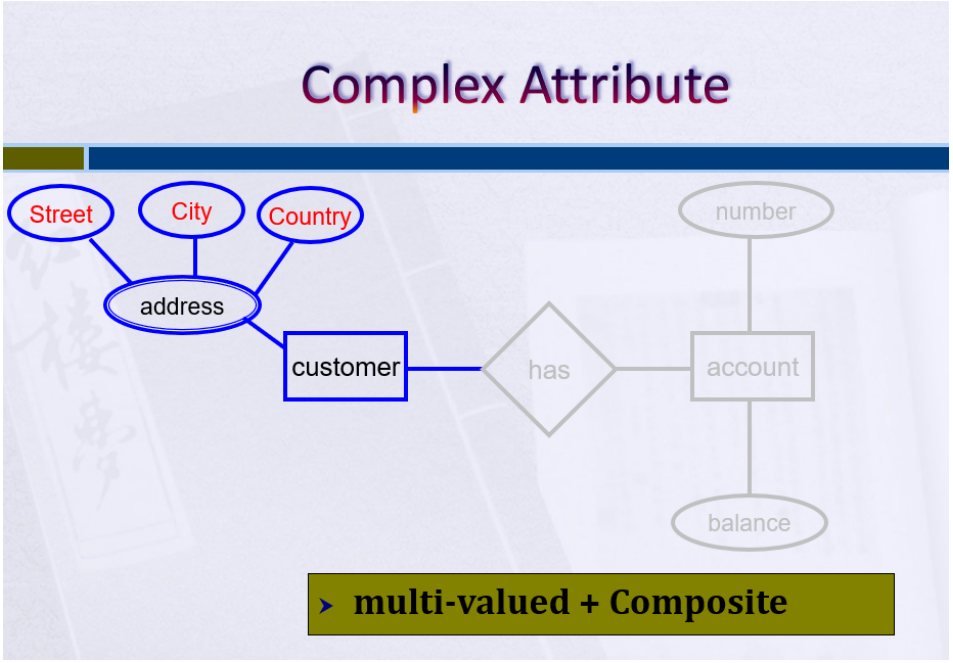
1. driven: can be driven from another attribute(age from birth date)



1. multi-valued: customer have two phone number



1. complex:(multi-valued + composite).



⇒ Relationships

Relationships define how entities are related to each other. In data modeling, relationships describe associations between two or more entities. They are crucial for organizing and retrieving data across multiple tables in a relational database.

- **Examples**:
  - A **student** enrolls in **courses** (a relationship between the student and course entities).
  - An **employee** works in a **department** (a relationship between employee and department entities).
  - A **customer** places an **order** (a relationship between customer and order entities).

Relation has three Properties:

⇒ Degree of relationship:

- unary (recursive): two instances in same entity :employee manage another employees
- binary two instances between two entities
- ternary two instances between three entities.

Property	**Unary Relationship**	**Binary Relationship**	**Ternary Relationship**
Definition	A relationship between instances of the same entity.	A relationship between two different entities.	A relationship involving three different entities.
Degree	1 (Involves a single entity).	2 (Involves two entities).	3 (Involves three entities).
Cardinality	Typically 1:1 or 1.	Can be 1:1, 1, or M.	Can involve complex cardinalities (often many-to-many).
Example	Employee managing another Employee (recursive relationship).	Employee working in a Department.	Supplier providing Products to a Customer.
Table Representation	One table, with a foreign key referencing the primary key of the same table.	Two tables, potentially with a junction table for Mrelationships. In One-to-One Relation: Use `foreign key` in one table to another. In One-to-Many Relation: Use `Junction table.`	Three tables with a junction table to handle the relationship.

Foreign Key	A foreign key refers to the primary key of the same table.	A foreign key in one table references the primary key of the other table.	The junction table contains foreign keys referencing all three entities.
Usage	Common in recursive structures (e.g., hierarchical or reporting relationships).	Most common type of relationship in database systems (e.g., employees, orders, products).	Used in more complex relationships involving three entities simultaneously (e.g., supply chain, project management).
Example Tables	Employee(EmployeeID, Name, ManagerID), where `ManagerID` refers to `EmployeeID`.	Employee(EmployeeID, Name, DepartmentID), Department(DepartmentID, DepartmentName).	Supplier(SupplierID, Name), Product(ProductID, Name), Customer(CustomerID, Name), and the junction table Supply(SupplierID, ProductID, CustomerID).
Example Table Structures	- **Employee Table**: Employee(EmployeeID, Name, ManagerID)	- **Employee Table**: Employee(EmployeeID, Name, DepartmentID) - **Department Table**: Department(DepartmentID, DepartmentName)	- **Supplier Table**: Supplier(SupplierID, SupplierName) - **Product Table**: Product(ProductID, ProductName) - **Customer Table**: Customer(CustomerID, CustomerName) - **Supply Table** (junction): Supply(SupplierID, ProductID, CustomerID)
Key Properties	- Involves only one entity. - Relationship is between instances of the same entity. - Uses a foreign key to link instances of the entity to itself.	- Involves two distinct entities. - Can be 1:1, 1, or Mdepending on the cardinality. - May require a junction table in Mrelationships.	- Involves three entities. - Uses a junction table that contains foreign keys for all three entities. - More complex to implement due to three-way involvement.

Summary:

- **Unary Relationships** are simple and involve self-referencing within one table.
- **Binary Relationships** are the most common, involving two tables and can vary in cardinality (1:1, 1, or M).
- **Ternary Relationships** are more complex, involving three tables and typically requiring a junction table to handle many-to-many-to-many relationships.

⇒ Cardenality Constraint: `one-to-one`, `one-to-many`, `many-to-many`

Here's a comparison of these cardinality types based on several properties:

Property	One-to-One (1:1)	One-to-Many (1)	Many-to-Many (M)
Definition	One instance of entity A is associated with exactly one instance of entity B, and vice versa.	One instance of entity A is associated with many instances of entity B, but each instance of entity B is associated with only one instance of entity A.	Many instances of entity A can be associated with many instances of entity B, and vice versa.
Degree	2 (involves two entities).	2 (involves two entities).	2 (involves two entities).
Cardinality	1:1	1(or N:1)	M(many-to-many).
Example	A **person** can have one **passport**, and a passport can be issued to only one person.	A **department** can have many **employees**, but an employee belongs to only one department.	A **student** can enroll in many **courses**, and a course can have many students.
Table Representation	Two tables, with one having a foreign key referencing the other (or a shared primary key in some cases).	Two tables, with the "many" table containing a foreign key referencing the "one" table.	Three tables: two entity tables, plus a **junction table** to handle the many-to-many relationship.
Foreign Key	One of the tables has a foreign key pointing to the primary key of the other table, or they share the same primary key.	The "many" side of the relationship has a foreign key pointing to the primary key of the "one" side.	The junction table contains two foreign keys: one referencing each entity's primary key.
Usage	Used when each entity in the relationship can be linked to only one entity in the other table (e.g., unique relationships like one person to one passport).	Used when one entity can relate to multiple instances of another entity, but not vice versa (e.g., hierarchical relationships).	Used for relationships where many instances of one entity are related to many instances of another entity (e.g., enrollment, orders and products).

Example Tables	**Person(PersonID, Name) Passport(PassportID, PersonID)**	**Department(DepartmentID, DeptName) Employee(EmployeeID, Name, DepartmentID)**	**Student(StudentID, Name) Course(CourseID, CourseName) Enrollment(StudentID, CourseID)**
Example Table Structures	- **Person Table**: `PersonID (PK), Name` - **Passport Table**: `PassportID (PK), PersonID (FK)`	- **Department Table**: `DepartmentID (PK), DeptName` - **Employee Table**: `EmployeeID (PK), Name, DepartmentID (FK)`	- **Student Table**: `StudentID (PK), Name` - **Course Table**: `CourseID (PK), CourseName` - **Enrollment Table**: `StudentID (FK), CourseID (FK)`
Key Properties	- Each entity is linked to exactly one instance of the other entity. - Foreign key is either a part of the primary key or a separate field.	- One entity on the "one" side can have many related instances on the "many" side. - The "many" side contains the foreign key referencing the "one" side.	- Both entities have many possible associations with each other. - A junction table is used to track the associations between the two entities.

Summary:

- **One-to-One (1:1)** relationships link exactly one instance of one entity with one instance of another. It's common in cases of unique relationships.
- **One-to-Many (1)** relationships are used when one entity can be associated with multiple instances of another entity but not the other way around.
- **Many-to-Many (M)** relationships are the most complex, requiring a junction table to manage the associations between many instances of two entities.

⇒ Participation constraint:

- total: all rows inter the relation : all cars must(mandatory, one or more) assign to employee,
- partial: some of rows inter relation :employee may(optional) have car.

Property	**Total Participation**	**Partial Participation**
Definition	Every instance of the entity must participate in the relationship.	Some instances of the entity may participate in the relationship, but it is not mandatory for all.
Degree	2 (involves two entities).	2 (involves two entities).
Cardinality	Can be 1:1 or 1.	Can be 1:1, 1, or M.
Example	Every **student** must enroll in at least one **course**.	Some **employees** may not have a **manager**, but those that do are linked to one.
Table Representation	A foreign key is mandatory in the child table, referencing the parent table.	A foreign key may be present in the child table, but it can also be null, indicating that the relationship is optional.
Foreign Key	The child table contains a foreign key that must reference the parent table.	The child table may have a foreign key referencing the parent table, but it can be nullable.
Usage	Used to enforce that every instance of an entity is involved in a relationship, ensuring data integrity.	Used when some instances of an entity are optional in the relationship, allowing for more flexible data modeling.
Example Tables	**Student(StudentID, Name) Enrollment(CourseID, StudentID)**	**Employee(EmployeeID, Name, ManagerID) Manager(ManagerID, Name)**
Example Table Structures	- **Student Table**: `StudentID (PK), Name` - **Enrollment Table**: `CourseID (PK), StudentID (FK)` where every `StudentID` must exist in the `Student` table.	- **Employee Table**: `EmployeeID (PK), Name, ManagerID (FK)` where `ManagerID` can be null if the employee does not have a manager.
Key Properties	- All instances of the entity must be associated with instances of the related entity. - Ensures data integrity and completeness.	- Some instances may not participate in the relationship. - Provides

		flexibility in data modeling, allowing for optional relationships.

Summary:

- **Total Participation** ensures every instance of an entity must have a corresponding related instance, enforcing data integrity and completeness.
- **Partial Participation** allows for optional relationships where some instances may not participate, providing flexibility in data modeling.

> In summary:
>
> **Entities** are objects or concepts that have data stored about them.
>
> **Attributes** are specific pieces of information or characteristics of an entity.
>
> **Relationships** represent the associations between different entities.

⇒ keys

> **Different Types of Keys:**
> Candidate Key
> Primary Key
> Foreign Key
> Composite Key
> Partial Key
> Alternate key
> Super Key

Key Type	Definition	Example Table Structure	Table Representation	Uniqueness	Nullability	Composition	Pur
Super Key	A set of one or more attributes that can uniquely identify a record in a table.	`Students (StudentID, Name, Email)`	`StudentID`	Unique (but may contain unnecessary attributes)	Can contain null values	Can be single or multiple attributes	Use ent
Candidate Key	A minimal super key; no subset can uniquely identify a record.	`Students (StudentID, Name, Email)`	`StudentID`	Unique	Cannot contain null values	Can be single or multiple attributes	Pot elig prir
Primary Key	A specific candidate key chosen to uniquely identify records in a table.	`Students (StudentID, Name, Email)`	`PRIMARY KEY (StudentID)`	Unique	Cannot contain null values	Can be single or multiple attributes	Enf inte ens rec uni

Foreign Key	An attribute that creates a relationship between two tables, referencing a primary key in another table.	`Enrollments (EnrollmentID, StudentID, CourseID)`	`FOREIGN KEY (StudentID) REFERENCES Students(StudentID)`	Not unique (may have duplicates)	Can contain null values	Usually a single attribute but can be composite	Enf refe inte bet rela
Composite Key	A key that consists of two or more attributes that together uniquely identify a record.	`Enrollments (StudentID, CourseID)`	`PRIMARY KEY (StudentID, CourseID)`	Unique	Cannot contain null values (if used as primary key)	Multiple attributes	Use uni ide wh attr suf
Partial Key	An attribute that can uniquely identify a subset of records but not the entire table (often used in weak entities).	`WeakEntity (EntityID, WeakAttribute)`	`PRIMARY KEY (EntityID, WeakAttribute)`	Unique for a subset	Can contain null values	Typically part of a composite key	Use ent ide rec
Alternate Key CT	Any candidate key that is not selected as the primary key.	`Students (StudentID, Name, Email)`	`UNIQUE (Email)`	Unique	Cannot contain null values	Can be single or multiple attributes	Pro alte me ide uni

Summary:

1-candidate key: some of primary keys.

2-composite: two attribute work as primary key if not found make an primary key from your self.

3-primary key: under the attribute there is a line .

4-foregin key

5- partial key ⇒ inside weak entity.

6-super key

7.alternative key

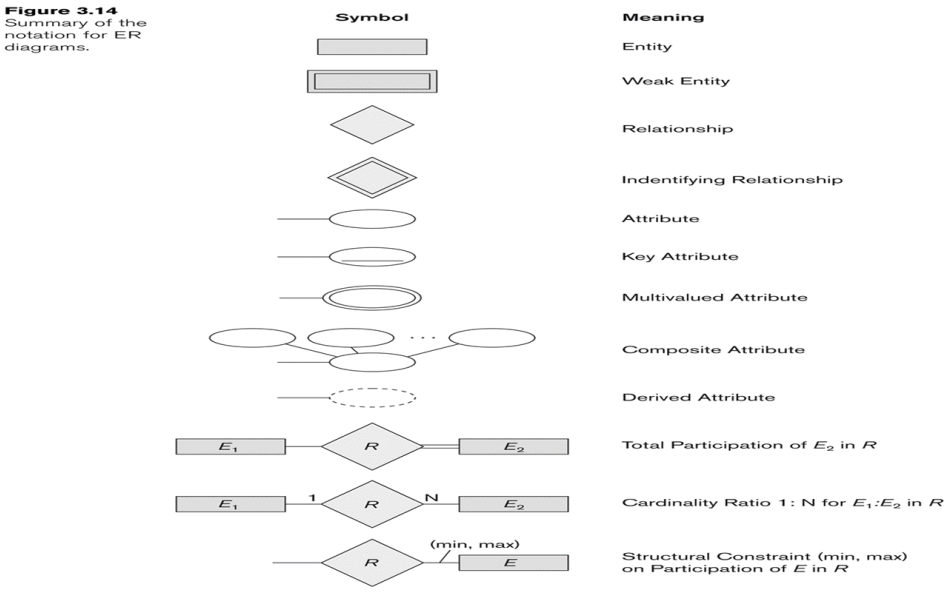


Figure 3.14
Summary of the notation for ER diagrams.

LAB

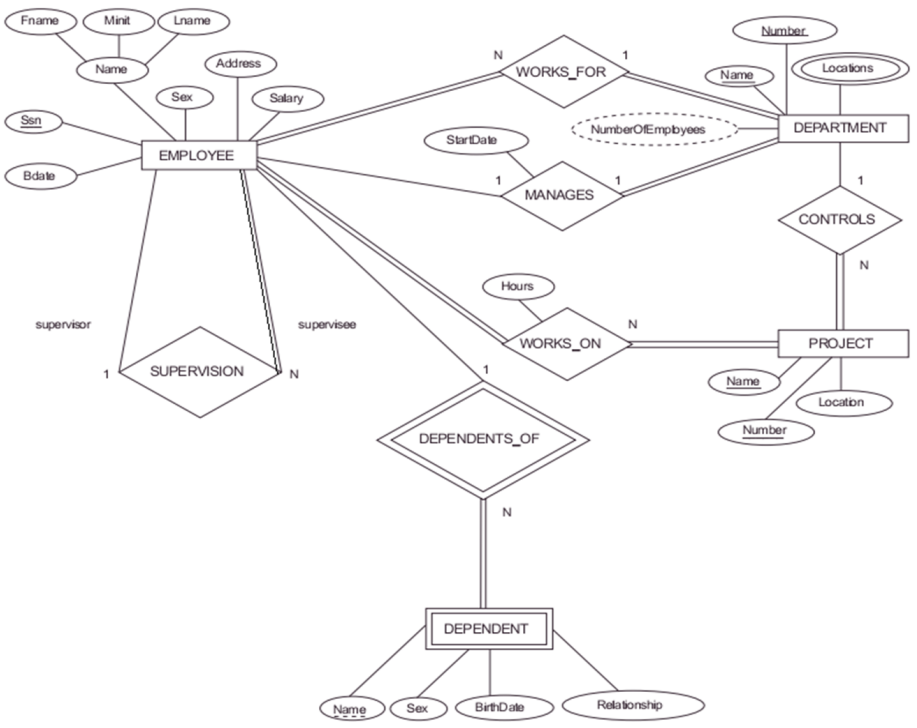
Case 1

A big company has decided to store information about its projects and employees in a database. The company has wisely chosen to hire you as a database designer. Prepare an E-R diagram for this Company according to The following Description:

- The company has a number of employees each employee has SSN, Birth Date, Gender and Name which represented as Fname and Lname.
- The company has a set of departments each department has a set of attributes .0DName, DNUM (unique) and locations.
- Employees work in several projects each project has Pname, PNumber as an identifier, Location and City.
- Each employee may have a set of dependent; each dependent has Dependent Name (unique), Gender, and Birth Date.

Note: if the employee left the company no needs to store his dependents info

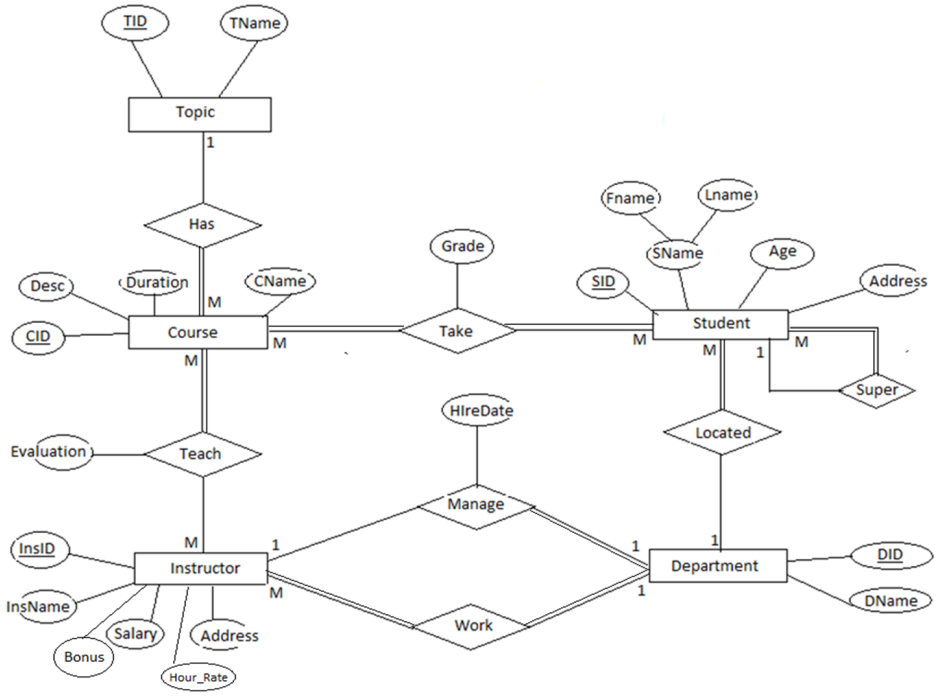
- For each Department, there is always one employee assigned to manage that Department and each manager has a hiring Date
- Department may have employees but employee must work on Only One department
- Each department may have a set of projects and each project must assigned to one department
- Employees work in several projects and each project has several employees and each employee has a number of working hours in each project
- Each employee has a supervisor



Case 2

ITI has decided to store information about its Students and curriculums in a database. ITI has wisely chosen to hire you as a database designer. Prepare an E-R diagram for ITI that describes its activities according to The following Description:

- ITI has a number of students in different departments(tracks), each student has St_id(unique),studentName(Fname,Lname),St_age,st_address
- Each department has dept_id(unique), dep_name
- Students takes many Courses, each course has crs_id, crs_name, crs_duration and crs_Description
- Student has a grade in each course
- Each department contains a set of instructor and each instructor in located in one department, each instructor has ins_id, ins_name, ins_salary, ins_hourRate, ins_bouns,ins_address
- For each Department, there is always one instructor assigned to manage that Department and each manager has a hiring Date
- Instructor may teach many courses and each course may be conducted by many instructors, each instructor has evaluation in each course
- Course is classified under one topic, each topic may have many courses, and each topic has top_name and top_id


TID
TName
Topic
1
Has
M
Desc
Duration
CName
CID
Course
M
M
Grade
Take
M
Fname
Lname
SName
Age
SID
Address
Student
M
1
M
Super
Evaluation
Teach
M
HIreDate
Manage
Located
InsID
Instructor
1
M
1
1
Department
DID
DName
InsName
Salary
Address
Bonus
Hour_Rate
Work
1