Database and SQL

Database and SQL: 2 credits (30 hours); L, T, P, SPW

1. Relational database conception principles

- Functional dependence
- Algorithms and normalization
- Normal forms
- Integrity constraints (static, dynamic, etc)

2. SQL language

3. Database administration

- Physical implementation of the data
- Structure of the file and index
- Control of concurrent access
- Breakdown resistance
- Security and protection of data
- Parameter setting, start, stop, save, restoration
- Distributed database, distributed processing
- Auditing, optimization

INTRODUCTION

The database is the collection of interrelated data in an organized manner. While hearing the term database, you might be tempted to ask why database? From the definition given above "database being the collection of data", there is always a need to collect data from people i.e. to have information about a particular group of people for example customers, employees, students, orders and so on. In addition what you should also understand that a database cannot manage itself. Since it's just an act of collecting data and not the management of that data; as it concerns the management of data, it's up to the hands of what we called DBMS (Database Management system), DBMS is just a piece of software that manages the creation, updating, deleting and management of data.

Before the term computerized database, humans had their ways of collecting data from the user's i.e through papers. We called that a paper base way of collecting data. With the paper-based data collection method, there were a lot of disadvantages since to query data was not an easy task to do. So when computers were introduced, it, therefore, gave rise to an electronic way of collecting data.

Still with this, two methods of collecting data were introduced i.e the flat file method and the RDBMS method. With the flat file, all data is placed in a single table. This, therefore, created other concerns like the redundancy of data.

What is Data?

Data is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed. For example: When you visit any website, they might store you IP address, that is data, in return they might add a cookie in your browser, marking sure that you visited the website, that is data, your name, it's data, your age, it's data.

Data becomes information when it is processed, turning it into something meaningful. Like, based on the cookie data saved on user's browser, if a website can analyse that generally men of age 20-25 visit us more, that is information, derived from the data collected.

What is a Database?

A Database is a collection of related data organized in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, storing data.

During early computer days, data was collected and stored on tapes, which were mostly write-only, which means once data is stored on it, it can never be written again. They were slow and bulky, and soon computer scientists realized that they needed a better solution to this problem.

What is DBMS?

A DBMS is a software that allows creation, definition and manipulation of database, allowing users to store, process and analyze data easily. DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.

DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.

- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID(Atomicity, Consistency, Isolation, Durability) properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS

Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.

Easily Maintenance: It can be easily maintainable due to the centralized nature of the database system.

Reduce time: It reduces development time and maintenance need.

Backup: It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.

multiple user interface: It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- Complexity: Database system creates additional complexity and requirements.

- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

Users

- **Database Administrators:** Database Administrator or DBA is the one who manages the complete database management system. DBA takes care of the security of the DBMS, it's availability, managing the license keys, managing user accounts and access etc.
- **Application Programmer or Software Developer:** This user group is involved in developing and desiging the parts of DBMS.
- **End User:** These days all the modern applications, web or mobile, store user data. How do you think they do it? Yes, applications are programmed in such a way that they collect user data and store the data on DBMS systems running on their server. End users are the one who store, retrieve, update and delete data.

I. : Database Model

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

Types of Data Model

High-level Conceptual Data Models

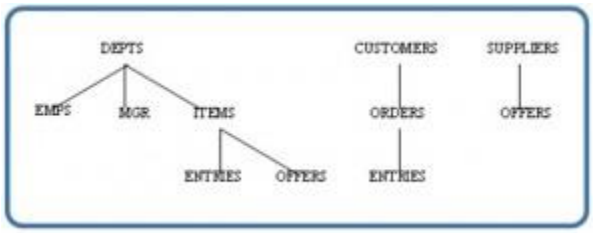
High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model, which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee’s name, address and birthdate. A relationship represents an association among entities; for example, an employee works on many projects. A relationship exists between the employee and each project.

Record-based Logical Data Models

Record-based logical data models provide concepts users can understand but are not too far from the way data is stored in the computer. Three well-known data models of this type are relational data models, network data models and hierarchical data models.

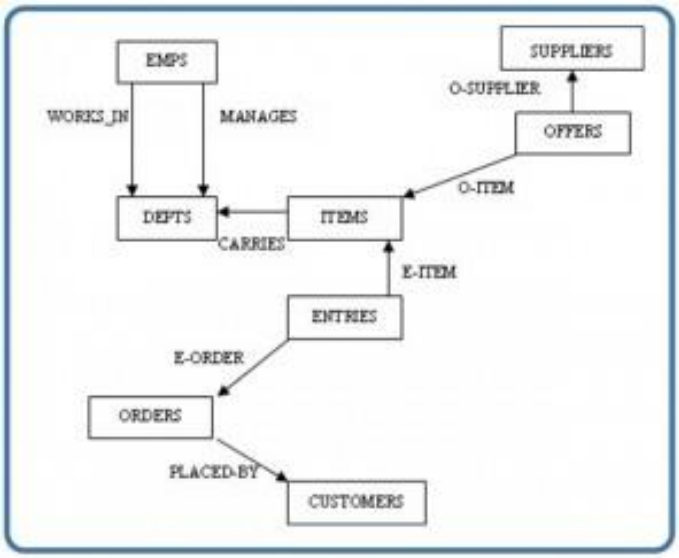
Hierarchical Model:

The hierarchical model represents data as a hierarchical tree structure. Each branch of the hierarchy represents a number of related records. The figure below shows this schema in hierarchical model notation.



Network Model

The network model represents data as record types. This model also represents a limited type of one to many relationships called a set type, as shown in the Figure below.



Relational Model

The relational model represents data as relations, or tables. For example, in the membership system at Science World, each membership has many members. The membership identifier, expiry date and address information are fields in the membership. The members are individuals such as Mickey, Minnie, Mighty, Door, Tom, King, Man and Moose. Each record is said to be an instance of the membership table.

A. Questions

2. What is a data model?

3. What is a high-level conceptual data model?
4. What is an entity? An attribute? A relationship?
5. List and briefly describe the common record-based logical data models.

A. The Relational Data Model

The relational data model was introduced by C. F. Codd in 1970. Currently, it is the most widely used data model.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language called s*tructured query language (SQL)*
- Almost all modern commercial database management systems

Fundamental Concepts in the Relational Data Model

i. Relation

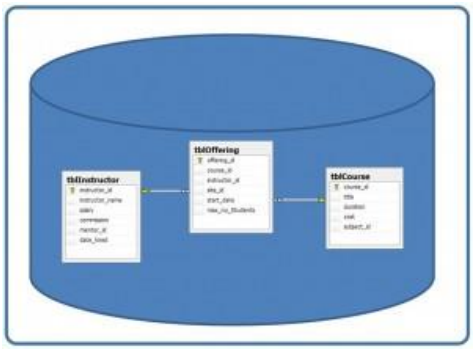
A *relation,* also known as a *table* or *file,* is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A *row*, or record, is also known as a *tuple*. The column in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.

The steps below outline the logic between a relation and its domains.

1. Given n domains are denoted by D1, D2, … Dn
2. And r is a relation defined on these domains
3. Then $r \subseteq D1 \times D2 \times \ldots \times Dn$

Table

A database is composed of multiple tables and each table holds the data. The Figure below shows a database that contains three tables.



Column

A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization.

The principal storage units are called *columns* or *fields* or *attributes*. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.

Domain

A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

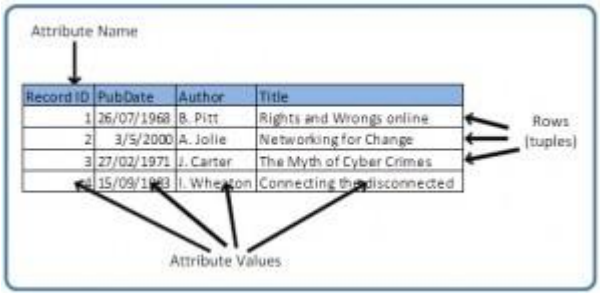
- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column.

Records

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form. Records allow us to do this. *Records* contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project.



Degree

The *degree* is the number of attributes in a table.

Properties of a Table

- A table has a name that is distinct from all other tables in the database.
- There are no duplicate rows; each row is distinct.
- Entries in columns are atomic. The table does not contain repeating groups or multivalued attributes.
- Entries from columns are from the same domain based on their data type including:
    - number (numeric, integer, float, smallint,…)
    - character (string)
    - date
    - logical (true or false)
- Operations combining different data types are disallowed.
- Each attribute has a distinct name.
- The sequence of columns is insignificant.
- The sequence of rows is insignificant.

Questions:

- Using correct terminology, identify and describe all the components in The table below
- What is the possible domain for field EmpJobCode?
- How many records are shown?
- How many attributes are shown?
- List the properties of a table.

EMPLOYEE

EMPID	EMPLNAME	EMPINIT	EMPFNAME	EMPJOBCODE
123455	Friedman	A.	Robert	12
123456	Olanski	D.	Delbert	18
123457	Fontein	G.	Juliette	15
123458	Cruazona	X.	Maria	18

B: The Entity Relationship Data Model

The *entity relationship (ER) data model* has existed for over 35 years. It is well suited to data modeling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modelling is based on two concepts:

- Entities, defined as tables that hold specific information (data)
- *Relationships*, defined as the associations or interactions between entities

Entity, Entity Set and Entity Type

An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- An object with physical existence (e.g., a lecturer, a student, a car)
- An object with conceptual existence (e.g., a course, a job, a position)

Entities can be classified based on their strength. An entity is considered weak if its tables are existence dependent.

- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity
    - The Spouse table, in the COMPANY database, is a weak entity because its primary key is dependent on the Employee table. Without a corresponding employee record, the spouse record would not exist.

An entity is considered strong if it can exist apart from all of its related entities.

- Kernels are strong entities.
- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

Another term to know is *entity type* which defines a collection of similar entities.
An *entity set* is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. For example, in the fugure below, the entity type is EMPLOYEE.

Existence dependency

An entity's existence is dependent on the existence of the related entity. It is existence-dependent if it has a mandatory foreign key (i.e., a foreign key attribute that cannot be null). For example, in the COMPANY database, a Spouse entity is existence -dependent on the Employee entity.

Kinds of Entities

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities.

Independent entities

Independent entities, also referred to as kernels, are the backbone of the database. They are what other tables are based on. *Kernels* have the following characteristics:

- They are the building blocks of a database.
- The primary key may be simple or composite.
- The primary key is not a foreign key.

- They do not depend on another entity for their existence.

Dependent entities

Dependent entities, also referred to as *derived entities*, depend on other tables for their meaning. These entities have the following characteristics:

- Dependent entities are used to connect two kernels together.
- They are said to be existence dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.
- There are three options for the primary key:

1. Use a composite of foreign keys of associated tables if unique
2. Use a composite of foreign keys and a qualifying column
3. Create a new simple primary key

Characteristic entities

Characteristic entities provide more information about another table. These entities have the following characteristics:

- They represent multivalued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
- Options for primary key are as follows:

1. Use a composite of foreign key plus a qualifying column
2. Create a new simple primary key. In the COMPANY database, these might include:

- Employee (<u>EID,</u> Name, Address, Age, Salary) – EID is the simple primary key.
- EmployeePhone (<u>EID, Phone</u>) – EID is part of a composite primary key. Here, EID is also a foreign key.

Attributes

Each entity is described by a set of attributes (e.g., Employee = (Name, Address, Birthdate (Age), Salary).

Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the ERD.

An Attritube is being represented by an oval symbol.

Types of Attributes

There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

Simple attributes

Simple attributes are those drawn from the atomic value domains; they are also called *single-valued attributes*. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

Composite attributes

Composite attributes are those that consist of a hierarchy of attributes. Example Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}

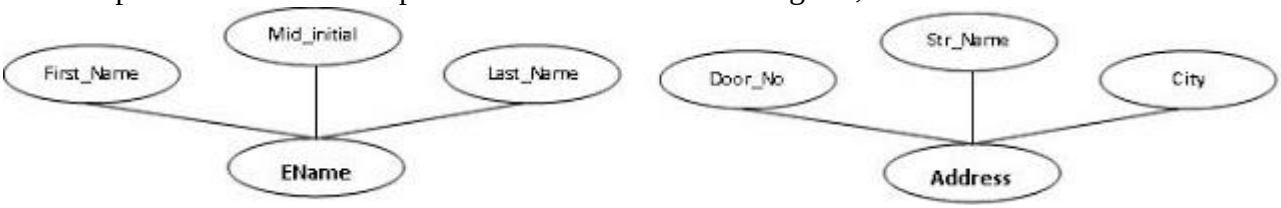
OR

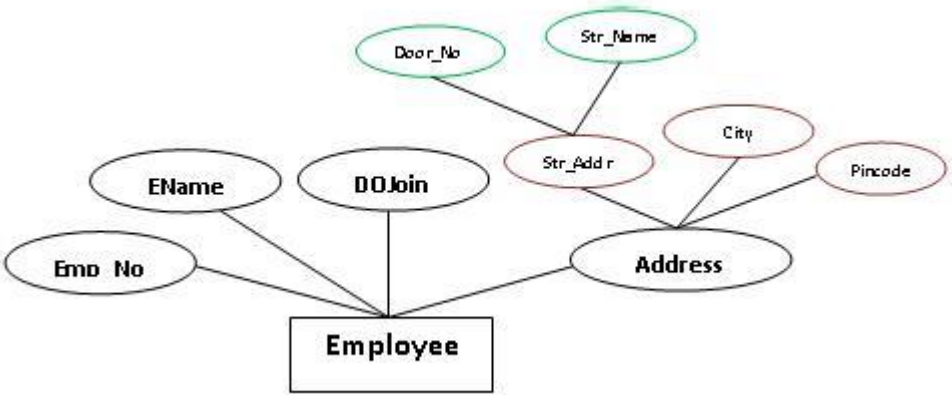
Composite attribute is an attribute where the values of that attribute can be further subdivided into meaningful sub-parts.

Typical examples for composite attribute are;

- Name – may be stored as first name, last name, middle initial
- Address – may be stored as door_no, street_name, area_name, city, pincode etc.
- DOB – may be stored as date, month and year and so on.

The composite attributes are represented as follows in ER diagram;





Multivalued attributes

Attribute which may expect one or more values for every entity is called multi-valued attribute. For example, an organization while storing employee details may accept more than one phone number per employee.

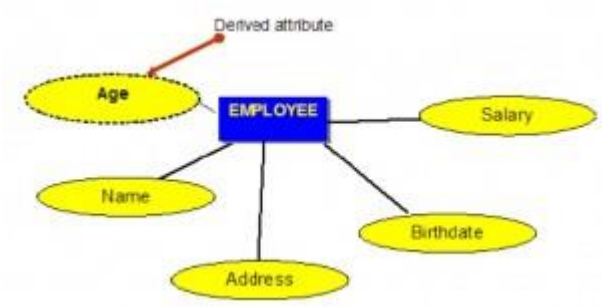
* In ER diagram, multi-valued attribute is represented with a double ellipse.

Derived attributes

Derived attributes are attributes that contain values calculated from other attributes. Example:

Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.

An attribute whose value can be derived from another attribute of the same table or from a set of entities is called Derived attribute.



Keys

An important constraint on an entity is the key. The *key* is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

Types of Keys

There are several types of keys. These are described below.

Candidate key

A candidate key is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

From our COMPANY database example, if the entity is Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID), possible candidate keys are:

EID, SIN

First Name and Last Name – assuming there is no one else in the company with the same name

Last Name and DepartmentID – assuming two people with the same last name don't work in the same department

Composite key

A *composite key* is composed of two or more attributes, but it must be minimal. Using the example from the candidate key section, possible composite keys are:

- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and Department ID – assuming two people with the same last name don't work in the same department

Primary key

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.

A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.

In the following example, EID is the primary key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Secondary key

A secondary key is an attribute used strictly for retrieval purposes (can be composite), for example: Phone and Last Name.

Alternate key

Alternate keys are all candidate keys not chosen as the primary key.

Foreign key

A foreign key (FK) is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type.

In the COMPANY database example below, DepartmentID is the foreign key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Nulls

A *null* is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

- No data entry
- Not permitted in the primary key
- Should be avoided in other attributes
- Can represent
  - An unknown attribute value

- A known, but missing, attribute value
- A "not applicable" condition

- Can create problems when functions such as COUNT, AVERAGE and SUM are used
- Can create logical problems when relational tables are linked

NOTE: The result of a comparison operation is null when either argument is null. The result of an arithmetic operation is null when either argument is null (except functions that ignore nulls).

Example of how null can be used

Salary_tbl

emp#	jobName	salary	commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

To begin, find all employees (emp#) in Sales (under the jobName column) whose salary plus commission are greater than 30,000.

- SELECT emp# FROM Salary_tbl
- WHERE jobName = Sales AND
- (commission + salary) > 30,000 –> E10 and E12

This result does not include E13 because of the null value in the commission column. To ensure that the row with the null value is included, we need to look at the individual fields. By adding commission and salary for employee E13, the result will be a null value. The solution is shown below.

- SELECT emp# FROM Salary_tbl
- WHERE jobName = Sales AND
- (commission > 30000 OR
- salary > 30000 OR
- (commission + salary) > 30,000 –>E10 and E12 and E13

Relationships

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. Company database examples include:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, Date)

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

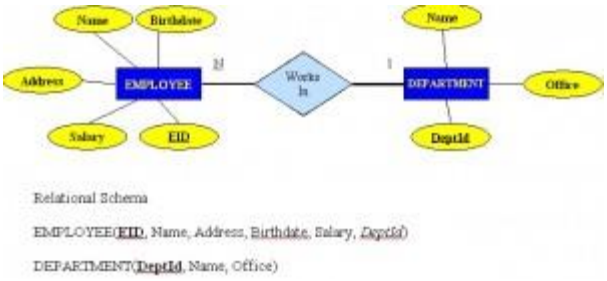
- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime…)

Types of Relationships

Below are descriptions of the various types of relationships.

One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. Figure 8.7 shows the relationship of one of these employees to the department.



One to one (1:1) relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

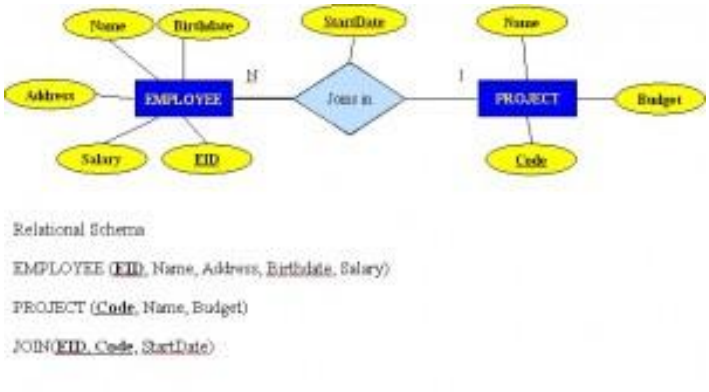
Many to many (M:N) relationships

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.
- The composite entity table must contain at least the primary keys of the original tables.

- The linking table contains multiple occurrences of the foreign key values.
- Additional attributes may be assigned as needed.
- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

The diagram above shows another aspect of the M:N relationship where an employee has different start dates for different projects. Therefore, we need a JOIN table that contains the EID, Code and StartDate.



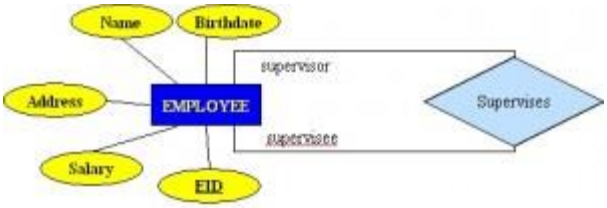
Example of mapping an M:N binary relationship type

- For each M:N binary relationship, identify two relations.
- A and B represent two entity types participating in R.
- Create a new relation S to represent R.
- S needs to contain the PKs of A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table R can be the PK.
- The combination of the primary keys (A and B) will make the primary key of S.

Unary relationship (recursive)

A unary relationship, also called recursive, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See Figure 8.9 for an example.

For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.



Relational Schema

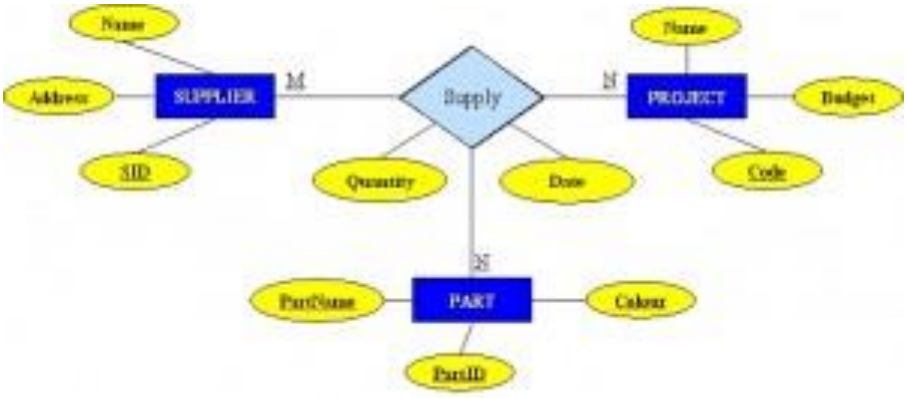
EMPLOYEE (EID, Name, Address, Birthdate, Salary, *Super-EID*)

Ternary Relationships

A *ternary relationship* is a relationship type that involves many to many relationships between three tables.

Refer to Figure 8.10 for an example of mapping a ternary relationship type. Note *n-ary* means multiple tables in a relationship. (Remember, N = many.)

- For each n-ary (> 2) relationship, create a new relation to represent the relationship.
- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an n-ary relationship, all the participating entities hold a **many** side.



Relational Schema

SUPPLIER (SID, Name, Address)
PROJECT(Code, Name, Budget)
PART(PartID, PartName, Colour)
Supply(SID, Code, PartID, Quantity, Date)