Introduction to Database

Data

Data is a collection of distinct pieces of information, particularly information that has been organized in some specific way for use in analysis or making decisions.

Information

Information is the data after processing, organizing, structuring or presenting in a given context so as to make it useful.

Data Base (DB)

A database is a set of data that has a regular structure and that is organized in such a way that a computer can easily find the desired information.
A database can generally be looked at as being a collection of records, each of which contains one or more fields (i.e., pieces of data) about some entity (i.e., object), such as a person, organization, city, product, work of art, chemical.

For example, the fields for a database that is about people who work for a specific company might include the name, employee identification number, address, telephone number, date employment started, position and salary for each worker.

Characteristics

Self-describing nature of a database system الوصف الذاتي لنظام قواعد البيانات

A database system is referred to as *self-describing* because it not only contains the database itself, but also metadata which defines and describes the data and relationships between tables in the database.
This separation of data and information about the data makes a database system totally different from the traditional file-based system in which the data definition is part of the application programs.

Insulation between program and data العزل بين البيانات والبرمجيات

In the file-based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well.

Support for multiple views of data تدعم رؤية متعددة للبيانات

A database supports multiple views of data. A *view* is a subset of the database, which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or group of users.

Sharing of data and multiuser system مشاركة البيانات ضمن انظمة متعددة

Current database systems are designed for multiple users. That is, they allow many users to access the same database at the same time. This access is achieved through features called ***concurrency control strategies.*** These strategies ensure that the data accessed are always correct and that data integrity is maintained.

Control of data redundancy ضبط الوفرة في البيانات

In the database approach, ideally, each data item is stored in only one place in the database. In some cases, data redundancy still exists to improve system performance.

Data sharing مشاركة البيانات

The integration of all the data, for an organization, within a database system has many advantages. **First**, it allows for data sharing among employees and others who have access to the system. **Second**, it gives users the ability to generate more information from a given amount of data than would be possible without the integration.

Enforcement of integrity constraints تطبيق شروط التكامل

Database management systems must provide the ability to define and enforce certain constraints to ensure that users enter valid information and maintain data integrity.
A *database constraint* is a restriction or rule that dictates what can be entered or edited in a table such as a postal code using a certain format or adding a valid city in the City field.

Restriction of unauthorized access تقييد الوصول غير المصرح به

Not all users of a database system will have the same accessing privileges امتيازات. For example, one user might have read-only access (i.e., the ability to read a file but not make changes), while another might have read and write privileges

For this reason, a database management system should provide a security subsystem to create and control different types of user accounts and restrict unauthorized access وصول غير مصرح به.

Data independence استقلالية البيانات

Another advantage of a database management system is how it allows for data independence. In other words, the system data descriptions or data describing data (metadata) are separated from the application programs. This is possible because changes to the data structure are handled by the database management system and are not embedded in the program itself.

Transaction processing المعاملات التجارية

A database management system must include concurrency control subsystems. This feature ensures that data remains consistent and valid during transaction processing even if several users update the same information.

Provision for multiple views of data تقديم رؤى متعددة للبيانات

By its very nature, a DBMS permits many users to have access to its database either individually or simultaneously. It is not important for users to be aware of how and where the data they access is stored.

Backup and recovery facilities تسهيل عمليات النسخ الاحتياطي والاسترداد

Backup and recovery are methods that allow you to protect your data from loss.

Key Terms

concurrency control strategies: features of a database that allow several users access to the same data item at the same time

data type: determines the sort of data permitted in a field, for example numbers only

data uniqueness: ensures that no duplicates are entered

database constraint: a restriction that determines what is allowed to be entered or edited in a table

metadata: defines and describes the data and relationships between tables in the database

read and write privileges: the ability to both read and modify a file

read-only access: the ability to read a file but not make changes

self-describing: a database system is referred to as self-describing because it not only contains the database itself, but also metadata which defines and describes the data and relationships between tables in the database

view: a subset of the database

Advantages

- Reduced data redundancy
- Reduced updating errors and increased consistency تناسق.
- Greater data integrity and independence from applications programs.
- Improved data access to users through use of host and query languages.
- Improved data security.
- Reduced data entry, storage, and retrieval costs.
- Facilitated development of new applications program.

Disadvantages

- Database systems are complex, difficult, and time-consuming to design.
- Substantial hardware and software start-up costs.
- Damage to database affects virtually all applications programs.
- If a user wants to change the structure of a file, all the programs that access that file might need to be changed as well.
- Extensive conversion costs in moving form a file-based system to a database system.
- Initial training required for all programmers and users.

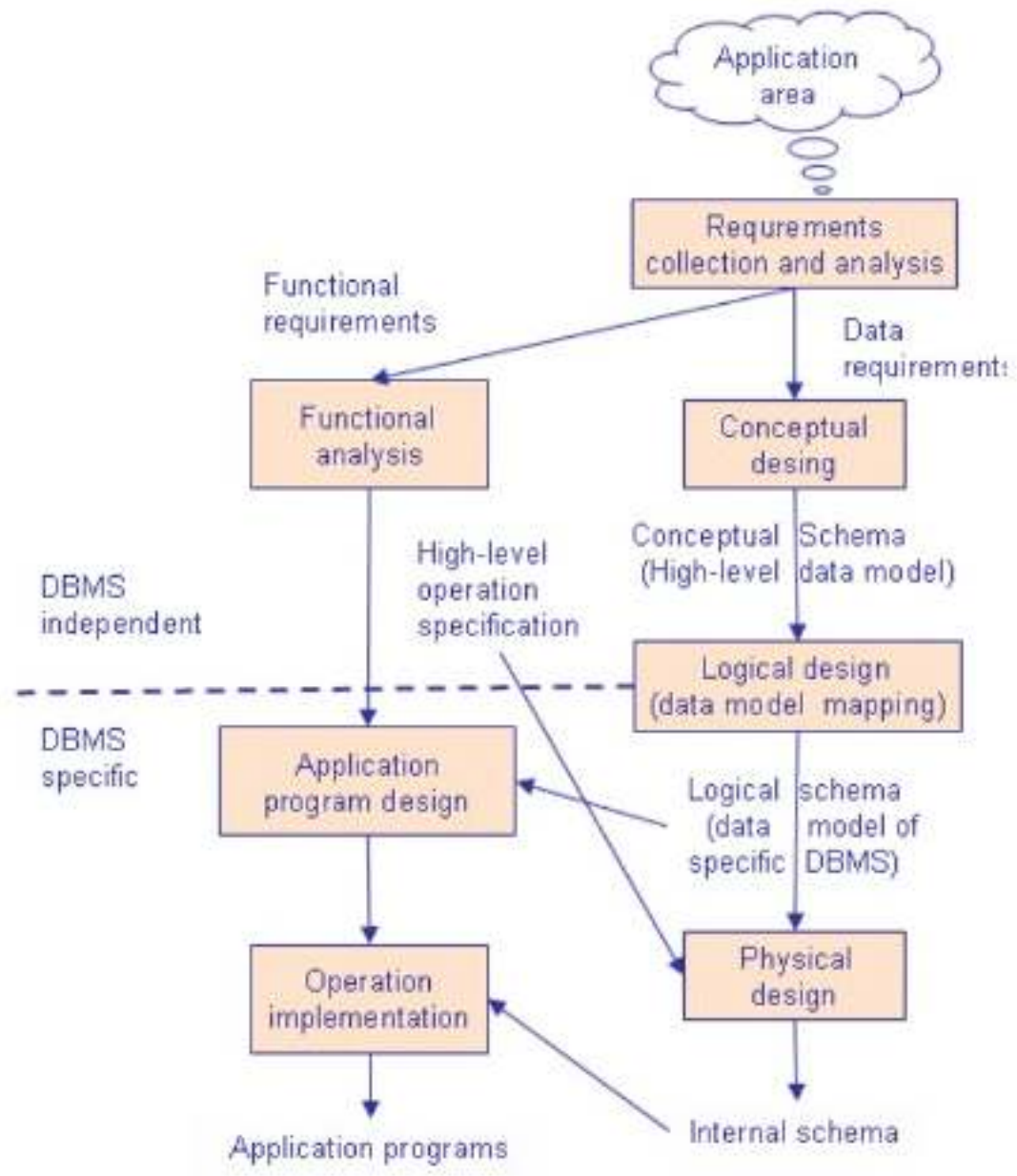
Relational Data Base (RDB)

A relational database is a way of organizing data such that it appears to the user to be stored in a series of interrelated tables. Relational databases became the dominant type for high performance applications because of their efficiency, ease of use, and ability to perform a variety of useful tasks that had not been originally envisioned.

Data Base Management System (DBMS)

A database management system (DBMS) is software that has been created to allow the efficient use and management of databases, including ensuring that data is consistent and correct and facilitating it's updating. For small, single user databases, all functions are often managed by a single program; for larger and multi-user databases, multiple programs are usually involved and a client-server architecture is generally employed.

Main phases of database design



Note that some phases are database management system independent and some are dependent.

Phase 1: The requirements collection and analysis phase

Produce both data requirements and functional requirements.

The idea is to design first the database without thinking about the actual database system - just to concentrate on the data.

- **Data requirements** are used as a **source of database design**. The data requirements should be specified in as **detailed and complete form** as possible.

- **Functional requirements** of the application. These consist of **user-defined operations** that will be applied to the database (retrievals and updates). The functional requirements are used as a **source of application software design**. Of course some functions may produce also needs for database design.

✓ The description of the data used or generated.
✓ The details how the data is to be used or generated.
✓ Any additional requirements for the new database system.

Phase 2: Conceptual Database Design التصميم النظري

Once all the requirements have been collected and analyzed, the **next step** is to create a conceptual schema مخطط for the database.
The purpose of the conceptual design phase ?
Is to build a conceptual model **based upon** the previously identified requirements, but **closer to** the final physical model النموذج الفعلي.

The result of this phase ?
Is an ***Entity-Relationship* model/Diagram** (**ERD**).
Describes:

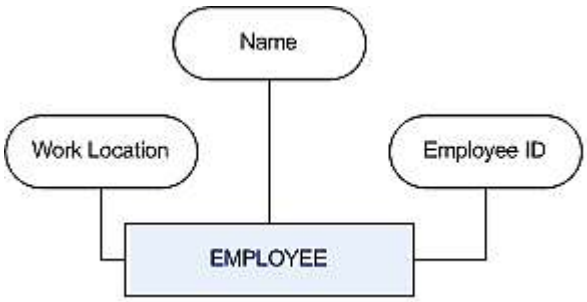
- How different entities (objects, items) are related to each other.
- What attributes (features) each entity has.
- Includes the definitions of all the concepts (entities, attributes) of the application area.

Entities

An entity is an object that exists. It doesn't have to do anything; it just has to exist. In database, an entity can be a single thing, person, place, or object. Data can be stored about such entities. A design tool that allows database administrators to view the relationships between several entities is called the entity relationship diagram (ERD).
Each entity has a unique ID called **Primary Key**.

Examples: Give Entities and attributes for:

- Students in IT College.
- Library.
- Patient in doctor clinic.
- Teachers in primary school.
- Store goods.
- Supper center.
- Restaurant.



Entity Attributes

An **attribute** defines the **information** about the entity that needs to be stored. If the entity is an employee, attributes could include name, employee ID, and work location.
An entity will have zero or more attributes, and each of those attributes apply only to that entity.

Domain

Domain of an entity describes the possible values of attributes. In the entity, each attribute will have only one value, which could be blank or it could be a number, text, a date, or a time. Here are examples of entity types and domains:

- Name: Jane Doe
- Employee ID: 123456
- Work Location: RO, ME, Floor 2

Here's an example figure of an entity.

Primary Key

The **key** is the unique identifier that identifies the instance of entity. A key is also a domain because it will have values. These values are unique to each record.

A key isn't always required, but it should be! In our example, a unique key value ensures that the Employee entity cannot have duplicate Social Security Numbers or Employee IDs.

Relationships

Defines the interaction between entities (one-one, one-many, and many-many).

One-to-Many Relationships

A one-to-many relationship is the most common type of relationship.

Examples:

- The publishers and titles tables have a one-to-many relationship. each publisher produces many titles, but each title comes from only one publisher.
- Mother may have more than one son. But the son have just one mother.
- A teacher may teach zero or more classes, while a class is taught by one.
- A single customer can purchase multiple orders, but a single order could not be linked to multiple customers.
- Bank agent may have more than bank account but the bank account belongs to just one agent.
- Person may have many phone number but the phone number belong to just one person.

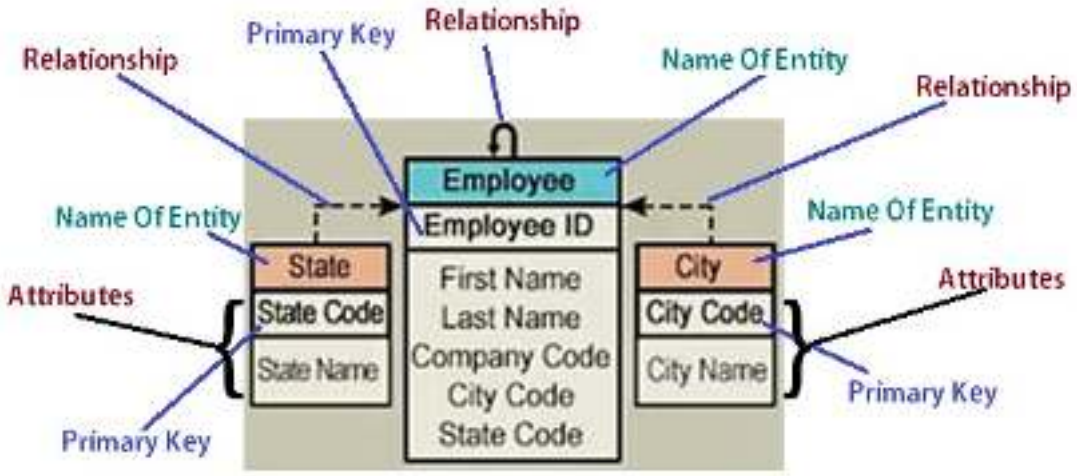
Many-to-Many Relationships

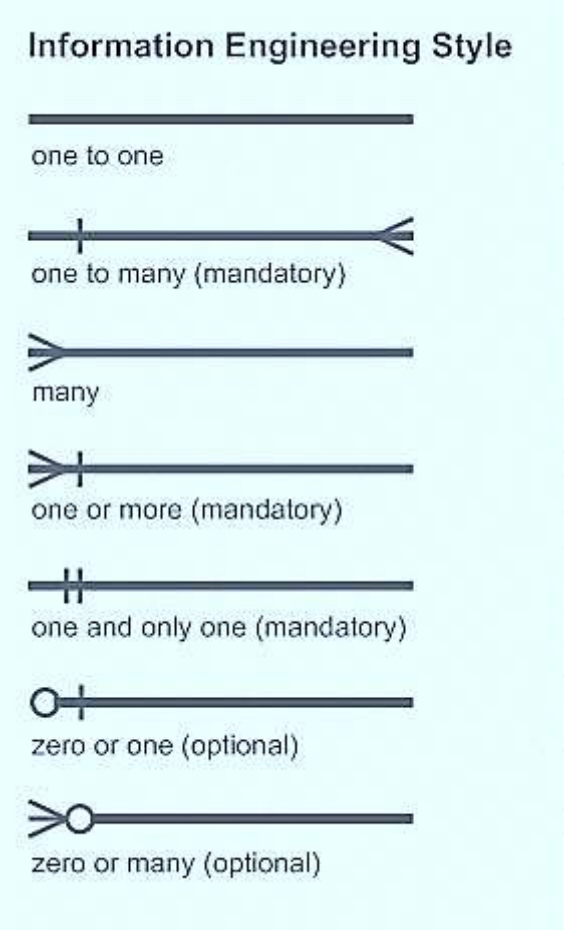
Examples:

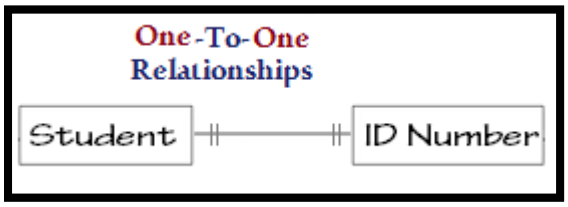
- The authors may have many books titles. And the book may have many authors.
- The person may belong to many groups and the group may have many person.
- Student may belong to many glasses and the glass have many students.

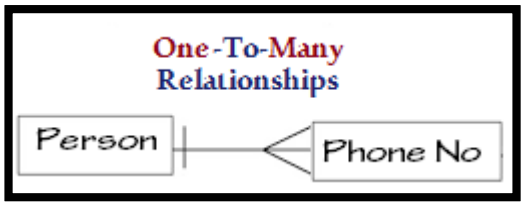
One-to-One Relationships

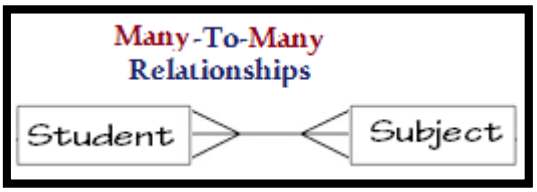
- Any city have one postal code and the postal code belong to just one city.
- The student have one ID number and each ID number belong to just one student.

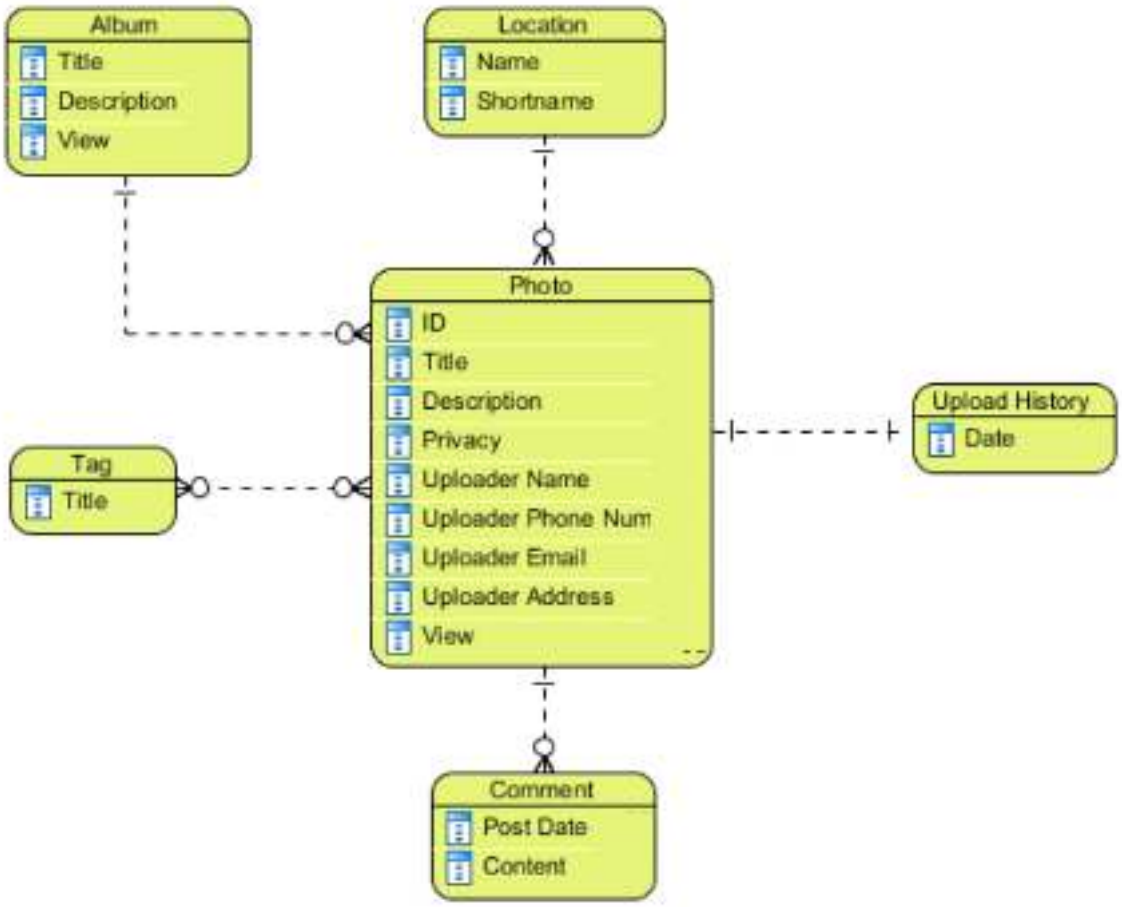

Relationship
Primary Key
Relationship
Name Of Entity
Relationship
Name Of Entity
Employee
Employee ID
First Name
Last Name
Company Code
City Code
State Code
Name Of Entity
State
State Code
State Name
Attributes
Primary Key
City
City Code
City Name
Attributes
Primary Key


Information Engineering Style
one to one
one to many (mandatory)
many
one or more (mandatory)
one and only one (mandatory)
zero or one (optional)
zero or many (optional)







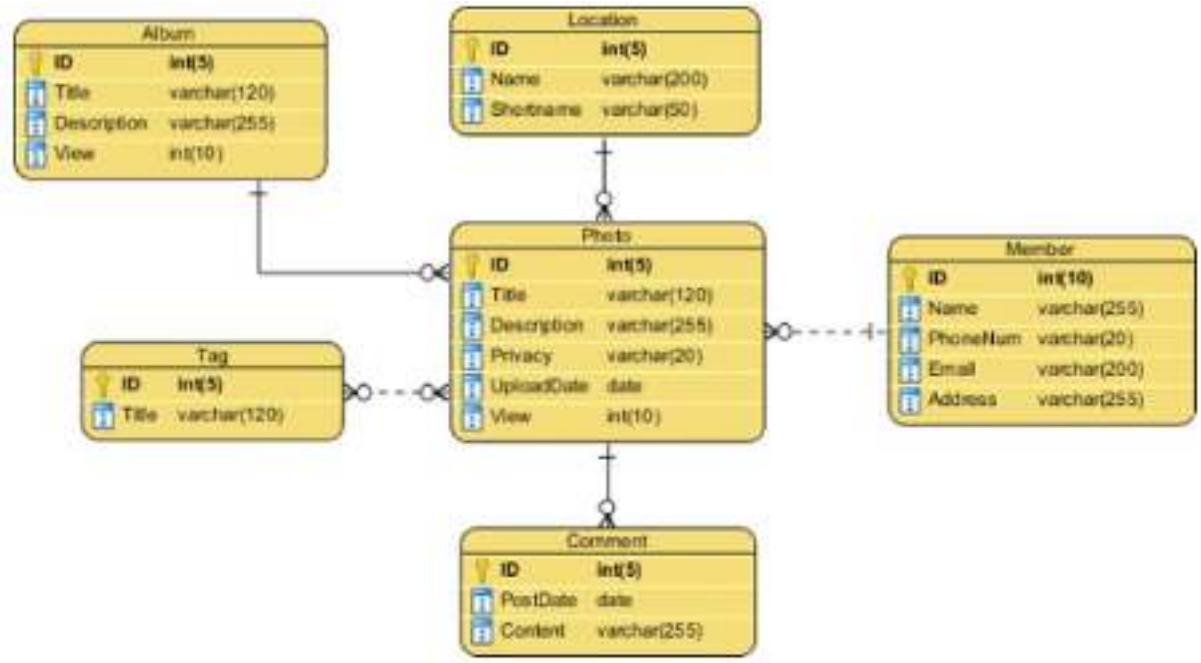


Conceptual ERD example

Phase 3: Logical Design

You can create the logical design using a pen and paper, or you can use a design tool. By beginning with the logical design, you focus on the information requirements **without** getting bogged down الغوص immediately with implementation detail.

It is more complex than conceptual model in that column types are set.

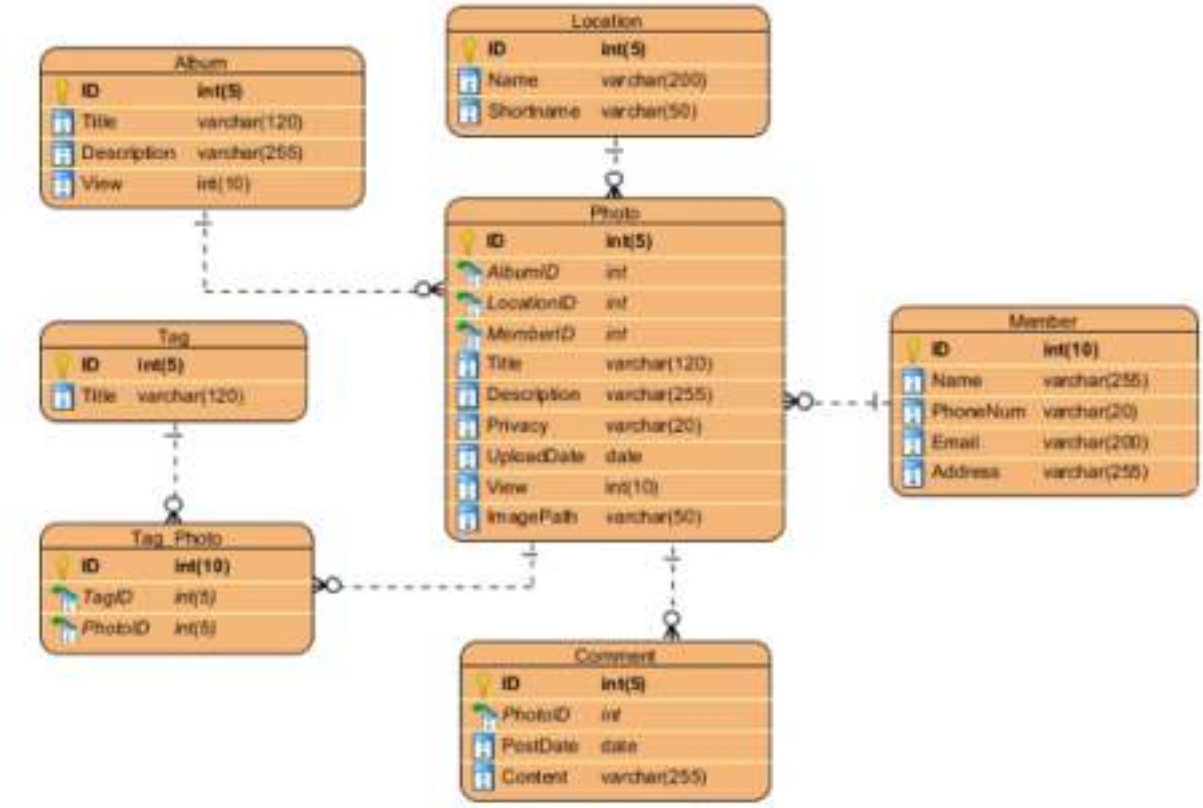


Logical ERD example

Phase 4: Physical Design

The goal of the last phase of database design, physical design, is to implement the database. At this phase one must know which database management system (DBMS) is used.

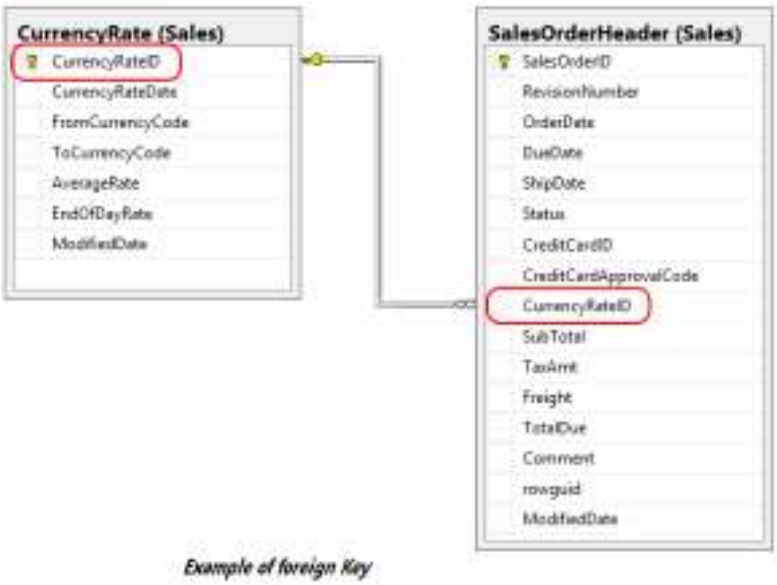
Physical ERD represents the actual design of a relational database. It represents how data should be structured and related in a specific DBMS so it is important to consider the restriction of the DBMS you use when you are designing a physical ERD. Besides, database designers may also add primary keys, foreign keys and constraints to the design.



Physical ERD example

Foreign Keys

A foreign key is a set of one or more columns in a table that refers to the primary key in another table. There isn't any special code, configurations, or table definitions you need to place to officially "designate" a foreign key.



Example of foreign Key

Logical	Physical
Entity	Table
Relationship	Foreign Key
Attribute	Column
Unique Identifier	Primary Key

Constructing an ER model

Before beginning to draw the ER model, read the requirements specification carefully. Document any assumptions you need to make.

1. Identify entities - list all potential entity types. These are the object of interest in the system. It is better to put too many entities in at this stage and they discard them later if necessary.
2. Remove duplicate entities - Ensure that they really separate entity types or just two names for the same thing.

    - Also do not include the system as an entity type
    - E.g. if modelling a library, the entity types might be books, borrowers, etc.
    - The library is the system, thus should not be an entity type.

3. List the attributes of each entity (all properties to describe the entity which are relevant to the application).

    - Ensure that the entity types are really needed.
    - Are any of them just attributes of another entity type?
    - If so keep them as attributes and cross them off the entity list.
    - Do not have attributes of one entity as attributes of another entity!

4. Mark the primary keys.

    - Which attributes uniquely identify instances of that entity type?
    - This may not be possible for some weak entities.

5. Define the relationships

    - Examine each entity type to see its relationship to the others.

6. Describe the cardinality and optionality of the relationships

    - Examine the constraints between participating entities.

7. Remove redundant relationships

    - Examine the ER model for redundant relationships.

ER modelling is an iterative process, so draw several versions, refining each one until you are happy with it. Note that there is no one right answer to the problem, but some solutions are better than others!

Attributes Type

Single valued Attributes: An attribute that has a single value for a particular entity. For example, age of an employee entity.

Multi valued Attributes: An attributes that may have multiple values for the same entity.

Contact hobbies

contactid	firstname	lastname	hobbies
1639	George	Barnes	reading
5629	Susan	Noble	hiking, movies
3388	Erwin	Star	hockey, skiing
5772	Alice	Buck	
1911	Frank	Borders	photography, travel, art
4848	Hanna	Diedrich	gourmet cooking

Hobbies

contactid	hobby
1639	reading
5629	hiking
5629	movies
3388	hockey
3388	skiing
1911	photography
1911	travel
1911	art
4848	gourmet cooking

Compound /Composite Attribute: Attribute can be subdivided into two or more other Attribute. For Example, Name can be divided into First name, Middle name and Last name.

Derived Attribute: Attributes derived from other stored attribute. For example age from Date of Birth and Today's date.

Stored Attribute: An attribute, which cannot be derived from other attribute, is known as stored attribute. For example, Birthdate of employee.

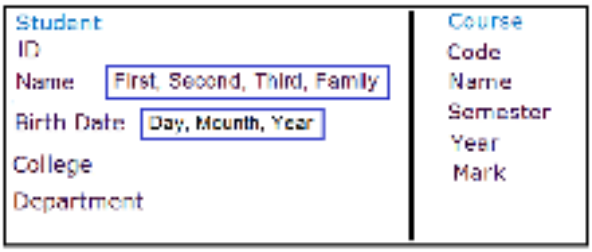
Key Attribute: represents primary key. (Main characteristics of an entity).

Optional Attribute/Null Value Attribute: An optional attribute may not have a value in it and can be left blank. For example, In a STUDENT email address is an optional attribute.

Example1: IT - Student Data

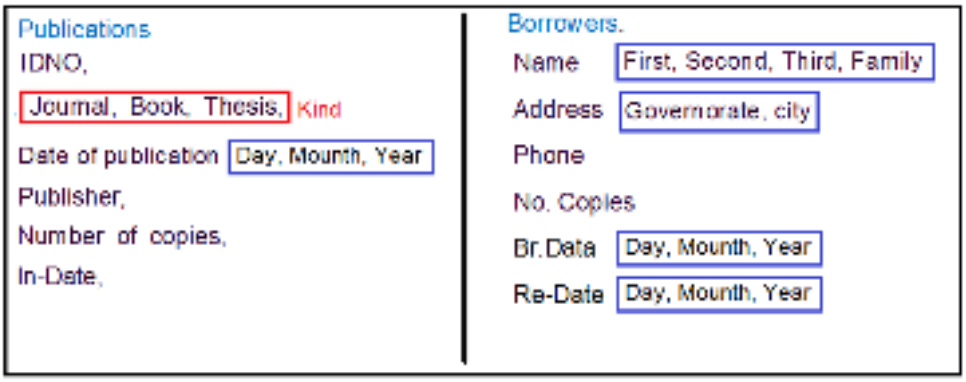
Data
ID, Name, Birth Date, College, Department, Marks.

Entities?
Attributes?



Example2: Library
Data
IDNO, Journal, Book, Thesis, Copy date, Publisher, Number of copies, In-Date, Borrower.

Entities?
Attributes?



ER Diagram

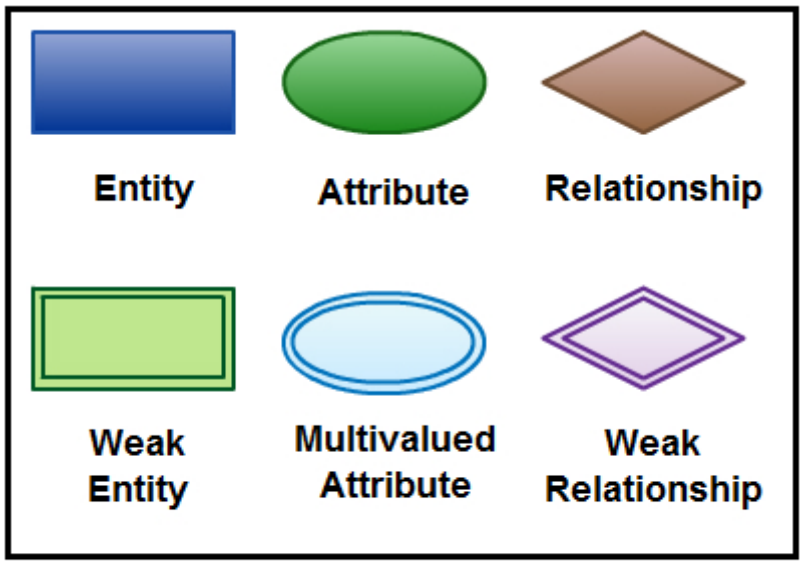
ER diagrams constitute a very useful framework for creating and manipulating databases.

1. Easy to understand and do not require a person to undergo extensive training to be able to work with it efficiently and accurately. So the designers can use ER diagrams to easily communicate with developers, customers, and end users, regardless of their IT proficiency.

2. Readily translatable into relational tables which can be used to quickly build databases.

3. May be applied in other contexts such as describing the different relationships and operations within an organization.

ER Diagram Symbols and Notations



Elements in ER diagrams

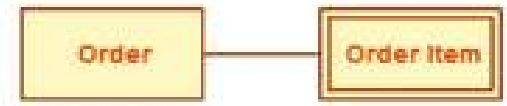
Entity

An entity can be a person, place, event, or object that is relevant to a given system. For example, a school system may include students, teachers, major courses, subjects, fees, and other items. Entities are represented in ER diagrams by a **rectangle** and named using **singular nouns**.

Weak Entity

A weak entity is an entity that **depends** on the **existence of another entity**. In more technical terms it can be defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key.

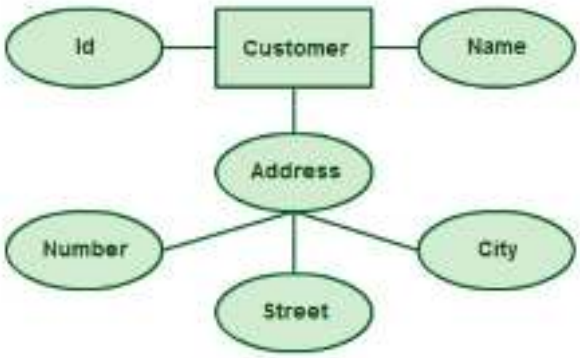
Example: Order item is a good example for this. The order item will be meaningless without an order so it depends on the existence of the order.



Weak Entity Example in ER diagram

Attribute

An attribute is a property, or characteristic of an entity, relationship, or another attribute. Attributes are represented by **oval shapes.**



Attributes in ER diagrams

An entity can have as many attributes as necessary.
Attributes can also have their own specific attributes (composite attributes).
For example, the attribute "customer address" can have the attributes number, street, city, and state.

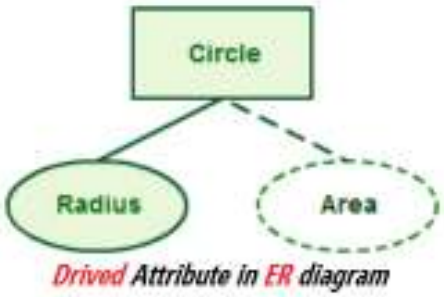
Multivalued Attribute

It is important to note that this is different to an attribute having its own attributes. For example, a teacher entity can have multiple subject values.

Example of Multivalued attribute

Derived Attribute

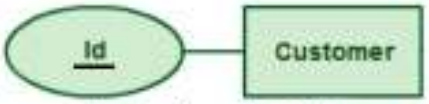
An attribute based on another attribute and derived from. For example, for a circle, the area can be derived from the radius.



Drived Attribute in ER diagram

Key Attribute

Key Attributes are represented by **oval shapes**, the name of key attributes must be **underline.**



Key Attributes in ER diagrams

Relationship

A relationship describes how entities interact. Relationships are represented by **diamond shapes** and are labeled using **verbs**.

Using Relationships in ER

For example, the entity "Carpenter" may be related to the entity "table" by the relationship "builds" or "makes".

Cardinality and Ordinality عدد وترتيب

These two further defines **relationships between entities** by placing the relationship in the context of numbers.

Cardinality in ER using UML notation

In an email system, for example, one account can have multiple contacts. The relationship, in this case, follows a "one to many" model. There are a number of notations used to present cardinality in ER diagrams. The following example uses UML to show cardinality.

How to Draw ER Diagrams

Below points show how to go about creating an ER diagram.

1. **Identify all the entities** in the system.
   An entity should **appear only once** in a particular diagram. Create rectangles for all entities and name them properly.

2. **Identify relationships** between entities.
   Connect them using a **line** and add a **diamond** in the middle describing the relationship.

3. **Add attributes** for entities.
   Give **meaningful attribute names** so they can be understood easily.

ER Diagram Best Practices

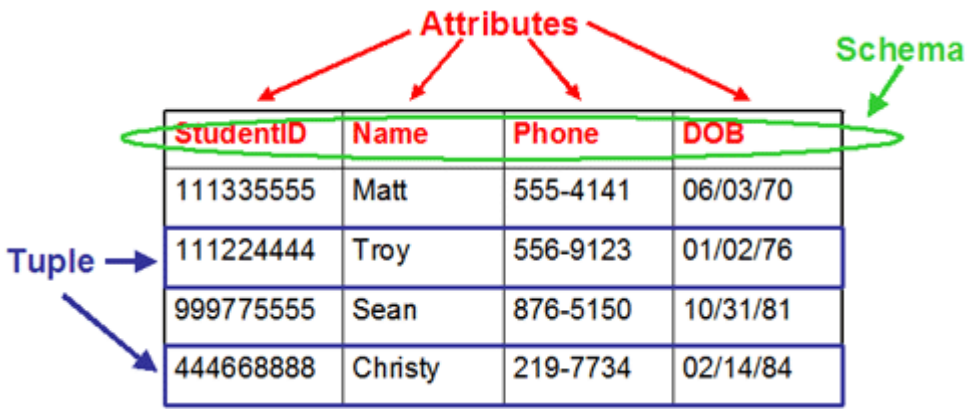
1. Provide a **precise and appropriate name** for each entity, attribute, and relationship in the diagram.

   - In naming entities, remember to use singular nouns.
   - Meanwhile attribute names must be meaningful, unique, system-independent, and easily understandable.

2. **Remove ambiguous**, redundant or unnecessary relationships between entities.
3. **Never connect** a relationship to another relationship.

Make **effective use of colors**. You can use colors to classify similar entities or to highlight key areas in your diagrams.

Terms



Example-1

Design the ER diagram corresponding to a relational database:

Person	(driver-id, name, address, age)
Car	(Car-No, year, model)
Accident	(report-number, driver-id, Car-No, damage-amount, location {city,town,street}, date)
Owns	(driver-id, Car-No)

Example-2

Design the ER diagram corresponding to a relational database:

Student

- Stud_no
- Age
- address (state, city, street)
- stud_name
- course_id
- hobby

Course

- course_id
- course_name

Subject

- course_id
- subj_id
- subject_name

Lecturer

- subj_id
- course_id
- Lecturer_name
- Lecturer_id

THE RELATIONAL ALGEBRA

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

- **Unary Relational Operations: SELECT and PROJECT**

1. The SELECT Operation

The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition. One can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition.

Syntax

$$\sigma_{<\text{Selection condition}>}(R)$$

- The σ would represent the SELECT command
- The <selection condition> would represent the condition for selection.
- The (R) would represent the Relation or the Table from which we are making a selection of the tuples.

Example: Results of select operation

σ (Dno=4 **AND** Salary>25000) OR (Dno=5 **AND** Salary>30000) (EMPLOYEE).

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

- Notice that the SELECT operation is commutative; that is,
  $\sigma_{cond_1}(\sigma_{cond_2}(R)) = \sigma_{cond_2}(\sigma_{cond_1}(R))$

  Hence, a sequence of SELECTs can be applied in any order.

- We can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,
  $\sigma_{cond_1}(\sigma_{cond_2}(\ldots(\sigma_{cond_n}(R))\ldots)) = \sigma_{cond_1 \text{ AND } cond_2 \text{ AND} \ldots \text{AND } cond_n}(R)$

- The SELECT operation chooses some of the rows from the table while discarding other rows.

2. The PROJECT Operation

If we are interested in only certain attributes of a relation, we use the PROJECT operation to project the relation over these attributes only. Therefore, the result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.

Syntax

$$\prod_{\text{<Attribute list>}}(R)$$

- $\prod$ would represent the ROJECT.
- <Attribute list> would represent the attributes (columns) we want from a relational.
- (R) would represent the relation or table we want to choose the attributes from.

Example: Results of Project operation

Fig a. $\prod_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$.
Fig b. $\prod_{\text{Sex, Salary}}(\text{EMPLOYEE})$.

(a)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(b)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

- The PROJECT operation, on the other hand, selects certain columns from the table and discards the other columns.

Sequences of Operations and the RENAME Operation

In general, for most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations.

In the latter case, we must give names to the relations that hold the intermediate results.

For example:

To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression, also known as an **in-line expression**, as follows:

$\pi_{Fname, Lname, Salary}((\sigma_{Dno=5})(EMPLOYEE))$

Figure (**a**) shows the result of this in-line relational algebra expression.

Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, as follows:

$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$

$RESULT \leftarrow \pi_{Fname, Lname, Salary}(DEP5\_EMPS)$

It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.

We can also use this technique to rename the attributes in the intermediate and result relations. This can be useful in connection with more complex operations such as UNION and JOIN, as we shall see.

To rename the attributes in a relation, we simply list the new attribute names in parentheses, as in the following example:

$TEMP \leftarrow \sigma_{Dno=5}(EMPLOYEE)$

$R_{(First\_name, Last\_name, Salary)} \leftarrow \pi_{Fname, Lname, Salary}(TEMP)$

These two operations are illustrated in Figure (b).

If no renaming is applied:

- For the SELECT operation, the names of the attributes in the resulting relation are the same as those in the original relation and in the same order.
- For a PROJECT operation, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

- **Relational Algebra Operations from Set Theory**
  A. UNION, INTERSECTION, and MINUS

1. UNION Operation

UNION: The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

For example, to retrieve the Social Security numbers (Ssn) of all employees who either work in department 5 or directly supervise an employee (Super_Ssn) who works in department 5, we can use the UNION operation as follows:

DEP5_EMPS ← $\sigma_{\mathbf{Dno=5}}$(EMPLOYEE)
RESULT1 ← $\pi_{\mathbf{S}sn}$(DEP5_EMPS)
RESULT2(Ssn) ← $\pi_{\mathbf{Super\_S}sn}$(DEP5_EMPS)
RESULT ← RESULT1 ∪ RESULT2

- RESULT1 has the Ssn of all employees who work in department 5.
- RESULT2 has the Ssn of all employees who directly supervise an employee who works in department 5.

- The UNION operation produces the tuples that are in either RESULT1 or RESULT2 or both, while eliminating any duplicates.
  Thus, the Ssn value '333445555' appears only once in the result.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

2. INTERSECTION Operation

INTERSECTION: The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S.

3. MINUS Operation

SET DIFFERENCE (or MINUS): The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.

Examples:

(a) Two union-compatible relations.

(b) STUDENT ∪ INSTRUCTOR.

(c) STUDENT ∩ INSTRUCTOR.

(d) STUDENT – INSTRUCTOR.

(e) INSTRUCTOR – STUDENT.

(a)

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

B. The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

This set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).
CARTESIAN PRODUCT operation (CROSS PRODUCT or CROSS JOIN) is denoted by **X**.
In general, the CARTESIAN PRODUCT operation applied by itself is generally **meaningless**.
It is **mostly useful** when followed by a selection that matches values of attributes coming from the component relations.

For example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:

FEMALE_EMPS ← $\sigma_{Sex='F'}$ (EMPLOYEE)

EMPNAMES ← $\pi_{Fname, Lname, Ssn}$ (FEMALE_EMPS)

EMP_DEPENDENTS ← EMPNAMES **X** DEPENDENT

ACTUAL_DEPENDENTS ← $\sigma_{Ssn=Essn}$ (EMP_DEPENDENTS)

RESULT ← $\pi_{Fname, Lname, Dependent\_name}$ (ACTUAL_DEPENDENTS)

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

- **Binary Relational Operations: JOIN and DIVISION**

1. The JOIN Operation

A. Inner join

The JOIN operation, denoted by ⋈, is used to combine related tuples from two relations into single "longer" tuples.
This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

Syntax

$$(R1) \bowtie_{\text{<join condition>}} (R2)$$

Car

CarModel	CarPrice
CarA	20,000
CarB	30,000
CarC	50,000

Boat

BoatModel	BoatPrice
Boat1	10,000
Boat2	40,000
Boat3	60,000

$Car \bowtie_{CarPrice \geq BoatPrice} Boat$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

Suppose that we want to retrieve the name of the manager of each department.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple. We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows:

DEPT_MGR ← DEPARTMENT $\bowtie_{\mathbf{Mgr\_ssn=Ssn}}$ EMPLOYEE
RESULT ← $\prod_{\mathbf{Dname, Lname, Fname}}$ (DEPT_MGR)

The first operation is illustrated in Figure below. Note that Mgr_ssn is a foreign key of the DEPARTMENT relation that references Ssn, the primary key of the EMPLOYEE relation. This referential integrity constraint plays a role in having matching tuples in the referenced relation EMPLOYEE.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Hint: The JOIN operation can be specified as a CARTESIAN PRODUCT operation followed by a SELECT operation.

EMP_DEPENDENTS ← EMPNAMES **X** DEPENDENT
ACTUAL_DEPENDENTS ← $\sigma_{\mathbf{Ssn=Essn}}$ (EMP_DEPENDENTS)

These two operations can be replaced with a single JOIN operation as follows:

ACTUAL_DEPENDENTS ← EMPNAMES $\bowtie_{Ssn=Essn}$ DEPENDENT

Exercise:

Purchase both mobile, laptop but mobile price should be less than laptop price.

Mobile

Model	MPrice
Nokia	10
Samsung	20
Iphone	50

Laptop

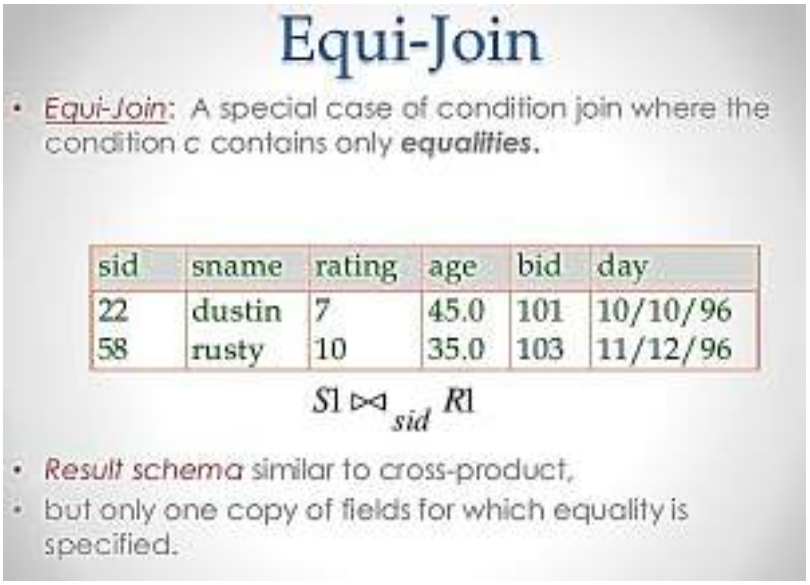
Model	LPrice
Dell	30
Acer	20
Asus	10

- **Variations of JOIN: The EQUIJOIN and NATURAL JOIN**

EQUIJOIN: The most common use of JOIN involves join conditions with equality comparisons only **=**.

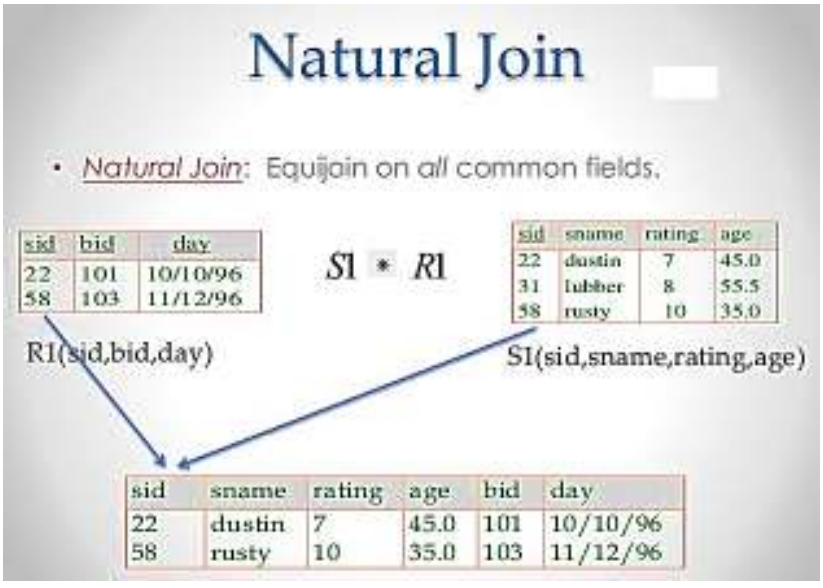
Notice: that in the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple.

For example, in Figure up, the values of the attributes Mgr_ssn and Ssn are identical in every tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes *requires the values to be identical* in every tuple in the result.



NATURAL JOIN (denote *): Was created to get rid of the second unnecessary attribute in an EQUIJOIN condition because this pair of attributes have identical values.

The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first.



Join attribute: *is the only attribute with the same name in both relations. So the attribute Dnum is called the for the NATURAL JOIN operation.*

Example: Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project.

In the following example

1. We rename the Dnumber attribute of DEPARTMENT to Dnum
2. Then we apply NATURAL JOIN:

PROJ_DEPT←PROJECT*ρ**(Dname, Dnum, Mgr_ssn, Mgr_start_date)**(DEPARTMENT)

The same query can be done in two steps by creating an intermediate table DEPT as follows:

DEPT ← ρ **(Dname, Dnum, Mgr_ssn, Mgr_start_date)**(DEPARTMENT)

PROJ_DEPT ← PROJECT * DEPT

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPT

Dname	Dnum	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

(a) PROJ_DEPT ⟵ PROJECT * DEPT

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

- If the attributes on which the natural join is specified already *have the same names in both relations*, renaming is unnecessary.

For example, to apply a natural join on the Dnumber attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write

DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

(b) DEPT_LOCS ⟵ DEPARTMANT * DEPT_LOCATIONS

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	DLocation
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

The resulting relation is shown in Figure (b), which combines each department with its locations and has one tuple for each location.

Notice:

- If no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples.
- If R has n_R tuples and S has n_S tuples, the result of a JOIN operation $R \bowtie_{\text{<join condition>}} S$ will have between zero and $n_R * n_S$ tuples.
- If there is no join condition, all combinations of tuples qualify and the JOIN degenerates into a CARTESIAN PRODUCT, also called CROSS PRODUCT or CROSS JOIN.

R

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

S

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

R * S

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

B. Left Outer Join:

In left outer join all tuples of left relation remains part of the output. That tuples that have a matching tuple in the second relation do have the corresponding tuple from the second relation. However, for the tuples of the left relation, which do not have a matching record in the right tuple have NULL values against the attributes of the right relation.

Left outer join is equijoin plus the non-matching rows of the side relation having NULL against the attributes of right side relation.

COURSE

Bk-ID	Bk-Title	St-ID
B10001	Intro to Database Systems	S104
B10002	Programming Fundamentals	S101
B10003	Intro Data Structures	S101
B10004	Modern Operating Systems	S106
B10005	Computer Architecture	
B10006	Advanced Networks	S104

STUDENT

St-ID	St-Name
S101	Ali Tahir
S102	Farah Hasan
S104	Farah Naz
S106	Asmat Dar
S107	Liegal Ali

COURSE ⋈ STUDENT

Bk-ID	Bk-Title	St-ID	St-ID	St-Name
B10001	Intro to Database Systems	S104	S104	Farah Naz
B10002	Programming Fundamentals	S101	S101	Ali Tahir
B10003	Intro Data Structures	S101	S101	Ali Tahir
B10004	Modern Operating Systems	S106	S106	Asmat Dar
B10006	Advanced Networks	S104	S104	Farah Naz
B10005	Computer Architecture	NULL	NULL	NULL

C. Right Outer Join:

In right outer all tuples of right relation remain part of the output relation, whereas, on the left side the tuples, which do not match with the right relation, are left as NULL. It means that right outer join will always have all the tuples of right relation and those of left relation which are not matched are left an NULL.

COURSE ⋈ STUDENT

Bk-ID	Bk-Title	St-ID	St-ID	St-Name
B10001	Intro to Database Systems	S104	S104	Farah Naz
B10002	Programming Fundamentals	S101	S101	Ali Tahir
B10003	Intro Data Structures	S101	S101	Ali Tahir
B10004	Modern Operating Systems	S103	S103	Farah Hasan
B10005	Advanced Networks	S104	S104	Farah Naz
NULL	NULL	NULL	S106	Asmat Dar
NULL	NULL	NULL	S107	Liaqat Ali

As we can see, a single JOIN operation is used to combine data from two relations so that related information can be presented in a single table. These operations are also known as **inner joins**, to distinguish them from a different join variation called **outer joins**.

Informally, an *inner join* is a type of match and combine operation defined formally as a combination of CARTESIAN PRODUCT and SELECTION. Note that sometimes a join may be specified between a relation and itself. The NATURAL JOIN or EQUIJOIN operation can also be specified among multiple tables, leading to an *n-way join*. For example, consider the following three-way join:

$$((\text{PROJECT} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

This combines each project tuple with its controlling department tuple into a single tuple, and then combines that tuple with an employee tuple that is the department manager. The net result is a consolidated relation in which each tuple contains this project-department-manager combined information.

Precedence of relational operators:

1. [σ, π, ρ] (Highest)
2. [X, ⋈]
3. ∩
4. [∪, —] (Lowest)

Student

Id	Name	Suburb
1108	Robert	Kew
3936	Glen	Bundoora
8507	Norman	Bundoora
8452	Mary	Balwyn

Takes

SID	SNO
1108	21
1108	23
8507	23
8507	29

Enrol

SID	Course
3936	101
1108	113
8507	101

Course

No	Name	Dept
113	BCS	CSCE
101	MCS	CSCE

Subject

No	Name	Dept
21	Systems	CSCE
23	Database	CSCE
29	VB	CSCE
18	Algebra	Maths

Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\langle\text{selection condition}\rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\langle\text{attribute list}\rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle\text{join condition}\rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle\text{join condition}\rangle} R_2$, OR $R_1 \bowtie_{(\langle\text{join attributes 1}\rangle), (\langle\text{join attributes 2}\rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\langle\text{join condition}\rangle} R_2$, OR $R_1 *_{(\langle\text{join attributes 1}\rangle), (\langle\text{join attributes 2}\rangle)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$

Files and Records

A **file** is a *sequence* of records. In many cases, all records in a file are of the same record type.

Records and Record Types

Data is usually stored in the form of **records**. Each record consists of a collection of related data **values** or **items**, where each value is formed of one or more bytes and corresponds to a particular **field** of the record. Records usually describe entities and their attributes.

For example, an EMPLOYEE record represents an employee entity, and each field value in the record specifies some attribute of that employee, such as Name, Birth_date, Salary, or Supervisor.

A collection of field names and their corresponding data types constitutes a **record type** or **record format** definition.

A **data type**, associated with each field, specifies the types of values a field can take. The data type of a field is usually one of the standard data types used in programming.

These include:

- Numeric (integer, long integer, or floating point)
- String of characters (fixed-length or varying)
- Boolean (having 0 and 1 or TRUE and FALSE values only)
- Sometimes specially coded **date** and **time** data types.

The number of bytes required for each data type is fixed for a given computer system.

- Integer may require 4 bytes
- Long integer 8 bytes
- Real number 4 bytes
- Boolean 1 byte
- Date 10 bytes (assuming a format of YYYY-MM-DD)
- Fixed-length string of *k* characters *k* bytes.
- Variable-length strings may require as many bytes as there are characters in each field value.

For example, an EMPLOYEE record type may be defined—using the C programming language notation—as the following structure:

```
struct employee {
                char        name[30];
                char        ssn    [9];
                int         salary;
                int         job_code;
                char        department [20];
              } ;
```

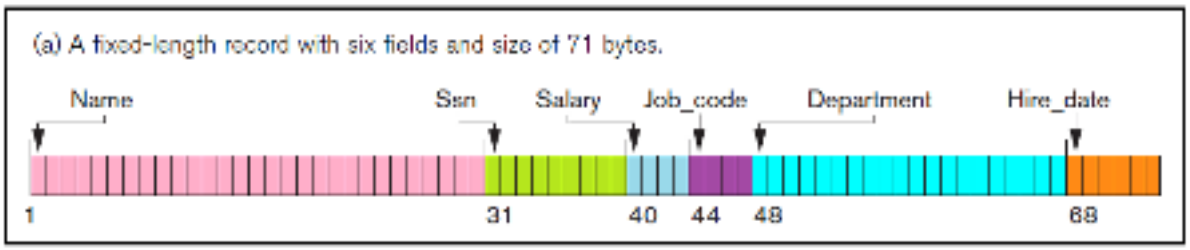
Fixed-Length Records, & Variable-Length Records

- If every record in the **file** has exactly the same size (in bytes), the file is said to be made up of **fixed-length records**.
- If different records in the **file** have different sizes, the file is said to be made up of **variable-length records**.

A file may have variable-length records for several reasons:

1. The file records are of the same record type, but one or more of the fields:
   - Of varying size (**variable-length fields**).
     For example, the Name field of EMPLOYEE can be a variable-length field.
   - May have multiple values for individual records; such a field is called a **repeating field** and a group of values for the field is often called a **repeating group**.
   - **Optional**; that is, they may have values for some but not all of the file records (**optional fields**).
2. The file contains records of different record types and hence of varying size (**Mixed file**).

The fixed-length EMPLOYEE records in Figure (a) have a record size of 71 bytes. Every record has the same fields, and field lengths are fixed, so the system can identify the starting byte position of each field relative to the starting position of the record.



Notice that it is possible to represent a file that logically should have variable-length records as a fixed-length records file.

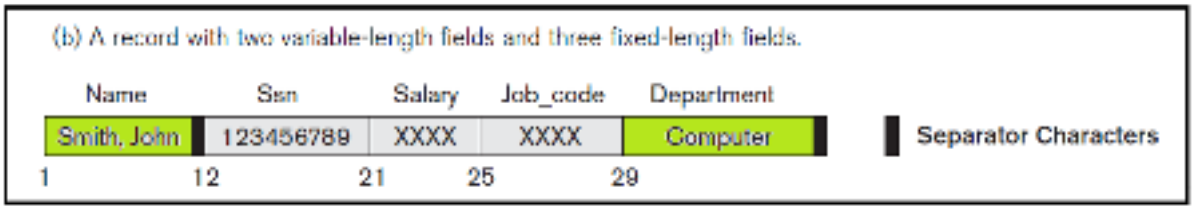
- Optional fields, we could have every field included in every file record but store a special NULL value if no value exists for that field.

- Repeating field, we could allocate as many spaces in each record as the maximum possible number of occurrences of the field. In either case, space is wasted when certain records do not have values for all the physical spaces provided in each record.

Formatting records of a file of variable-length records (Other options)

For variable-length fields, each record has a value for each field, but we do not know the exact length of some field values. To determine the bytes within a particular record that represent each field:

- We can use special **separator** characters (such as **?** or **%** or **$**)—which do not appear in any field value—to **terminate** variable-length fields, as shown in Figure (b).
- Or we can store the length in bytes of the field in the record, **preceding** the field value.

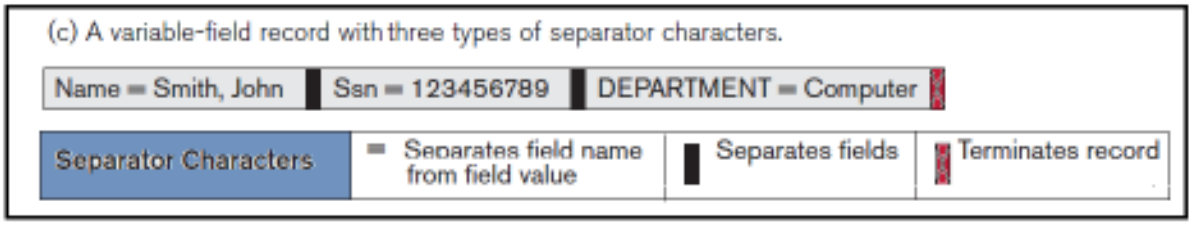


Formatted a file of records with optional fields:

If the total number of fields for the record type is large, but the number of fields that actually appear in a typical record is small:

- We can include in each record a sequence of <field-name, field-value> pairs rather than just the field values. Figure (c).

Three types of separator characters are used in Figure (c), although we could use the same separator character for the first two purposes—

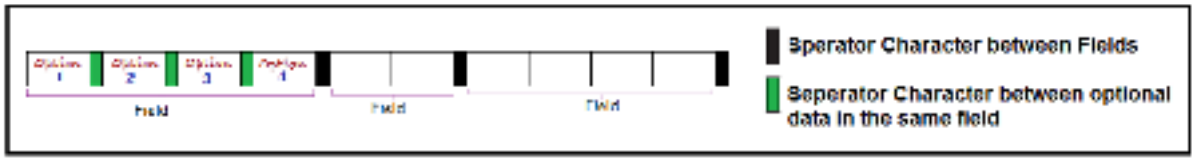


separating the ❶field name from the field value, ❷separating one field from the next field and ❸the end (terminate) of record.

Formatting A repeating field

Needs two separators:

- One separator character to separate the repeating values of the field
- Another separator character to indicate termination of the field.



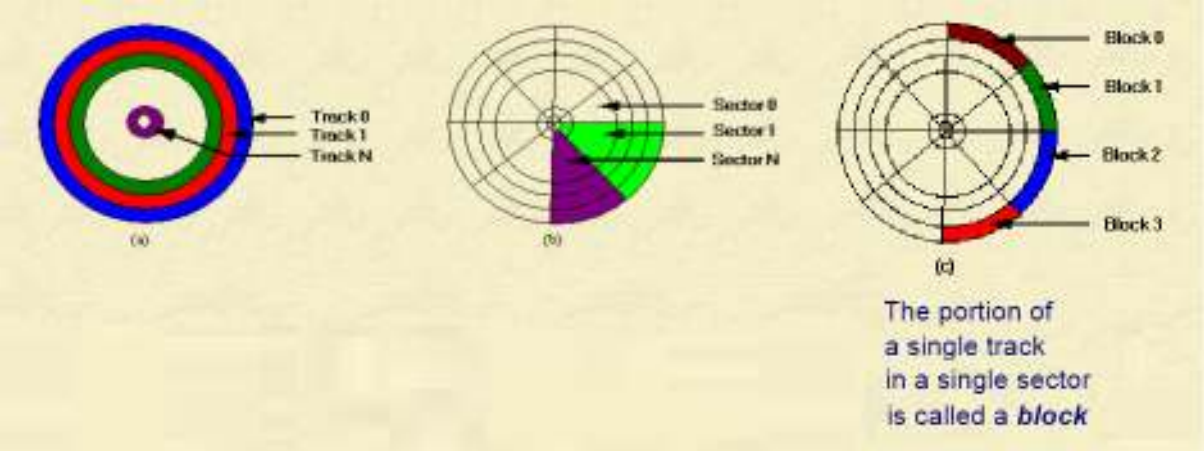
Formatting file that includes records of different types

Each record is preceded by a **record type** indicator. Programs that process files of variable-length records need to be more complex than those for fixed-length records, where the starting position and size of each field are known and fixed.

Organizing records in the file

Record Blocking and Spanned *vs* Unspanned Records

The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory.



When the block size is larger than the record size
→Each block will contain numerous records.
When the block size is smaller than the record size
→Some files may have unusually large records that cannot fit in one block.

UnSpanned organization:

- If records are not allowed to cross block boundaries.
- May have some unused space in each block.

Spanned organization:

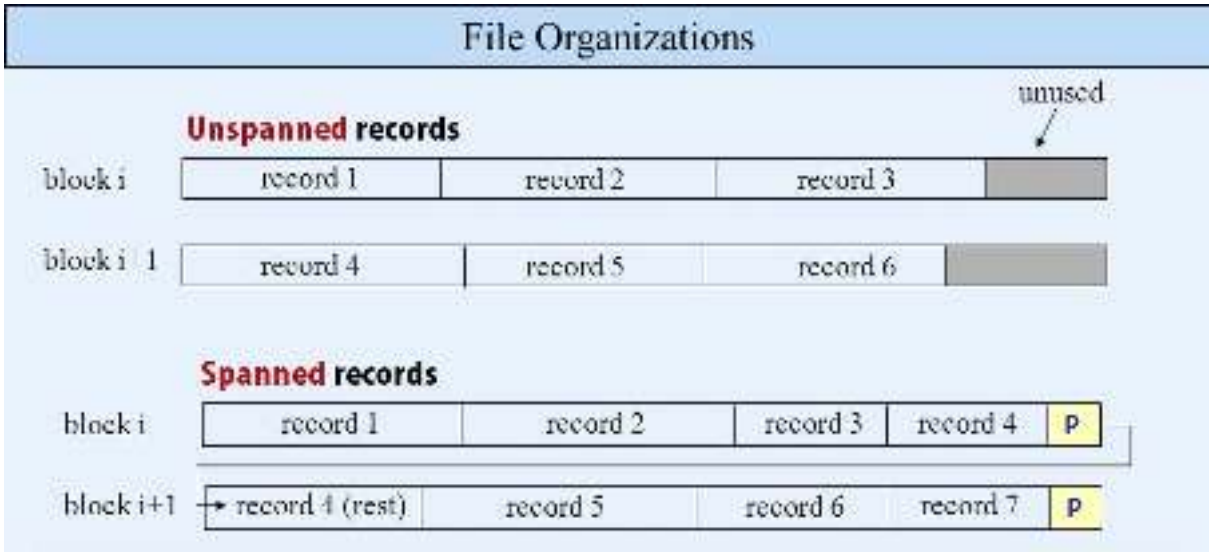
- If records are allowed to cross block boundaries
- To utilize this unused space (in case of unspanned) we can store part of a record on one block and the rest on another.
- A **Pointer** at the end of the first block points to the block containing the rest of the record.

For fixed-length records

Use **Unspanned** organization when block size larger than record size because it makes each record start at a known location in the block, **simplifying record processing**.

For variable-length records

Either a spanned or an unspanned organization can be used.
If the average **record** is large or **larger than block**, it is advantageous to use **spanning** to reduce the lost space in each block.
When using spanned organization, each block may store a different number of records.
Remainder of the record in this case it is not the next consecutive block on disk.



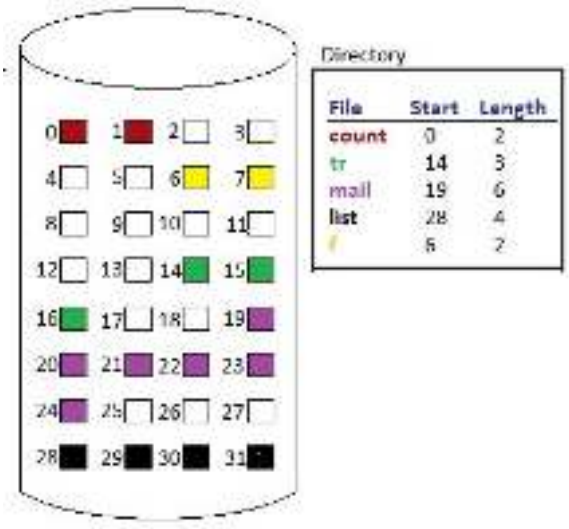
Allocating File Blocks on Disk

There are several standard techniques for allocating the blocks of a file on disk.

Contiguous allocation

The file blocks are allocated to **consecutive disk blocks**.

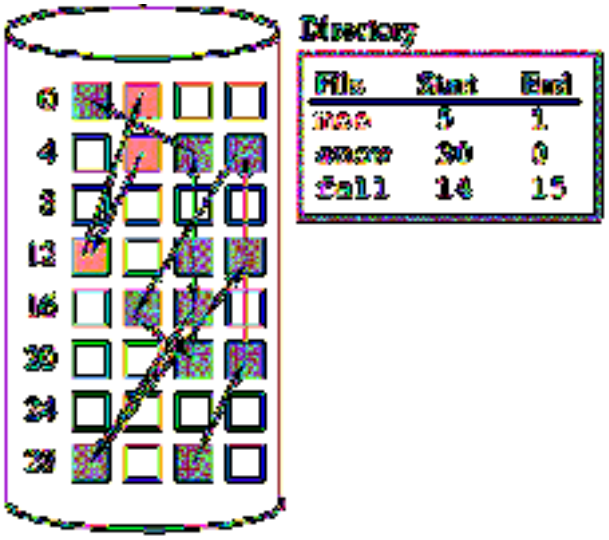
- This makes **reading** the whole file **very fast** using double buffering.
- But it makes **expanding** the file **difficult**.
- Only **starting** block and **length** of file in blocks are needed to work with the file



Linked allocation

Each file block **contains a pointer** to the next file block. A combination of the two allocates clusters of consecutive disk blocks, and the clusters are linked.

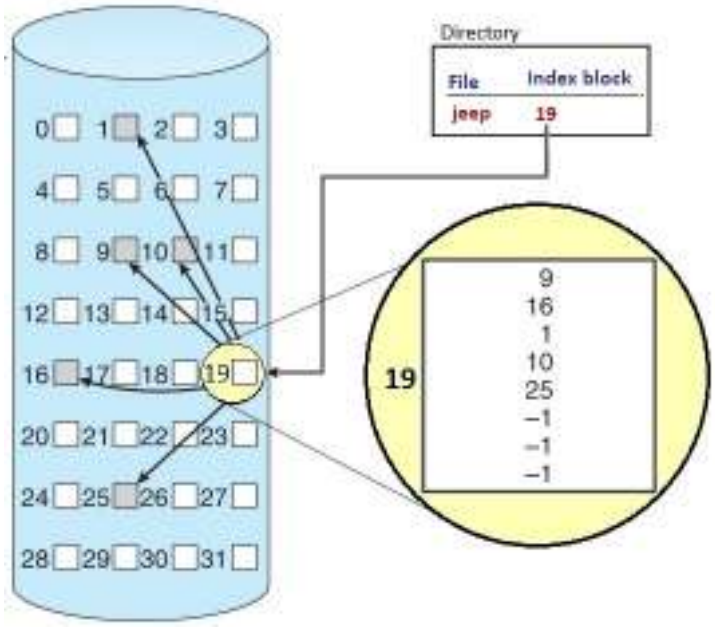
- This makes it **easy** to **expand** the file.
- But makes it **slow** to **read** the **whole file** (no direct access).



Indexed allocation

Bringing all the **pointers together**. Where one or more **index blocks** contain pointers to the actual file blocks. It is also common to use combinations of these techniques.

- Much more **effective** for **direct access**.
- **Inefficient** for **small files** (both access and space).



File Headers

A file header or file descriptor contains information about a file that is needed by the system programs that access the file records.

The header includes information to determine the disk addresses as well as record format descriptions:

Fixed-length unspanned records

- Field lengths
- Order of fields within a record

Variable-length records.

- Field type
- Separator characters
- Record type codes

To search for a record on disk:

1. One or more blocks are copied into main memory buffers.
2. Programs then search for the desired record or records within the buffers, using the information in the file header.

3. If the address of the block that contains the desired record is not known, the search programs must do a linear search through the file blocks.
4. Each file block is copied into a buffer and searched until the record is located or all the file blocks have been searched unsuccessfully.

This can be very time-consuming for a large file. The goal of a good file organization is to locate the block that contains a desired record with a minimal number of block transfers.

Files of Unordered Records (Heap Files)

In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a heap file.

We can use either spanned or unspanned organization and it may be used with either fixed-length or variable-length records.

Modifying a variable-length record

1. Deleting the old record
2. Inserting a modified record

Because the modified record may not fit in its old space on disk.

Inserting a new record

Is very efficient.

1. The last disk block of the file is copied into a buffer.
2. The new record is added.
3. The block is then rewritten back to disk.
4. The address of the last file block is kept in the file header.

Deleting a record

1. A program must first find the record block.
2. Copy the block into a buffer.
3. Delete the record from the buffer.
4. Finally rewrite the block back to the disk.

Disadvantage

This leaves unused space in the disk block. Deleting a large number of records in this way results in wasted storage space.

Files of Ordered Records (Sorted Files)

We can physically order the records of a file on disk based on the values of one of their fields—called the ordering field. This leads to an ordered or sequential file.

Ordered records advantages over unordered files:

1. Reading the records in order becomes efficient because no sorting is required.

2. Finding the next record from the current one usually requires no additional block accesses because the next record is in the same block as the current one (unless the current record is the last one in the block).
3. Using a search condition based on the value of an ordering key field results in faster access when the binary search technique.

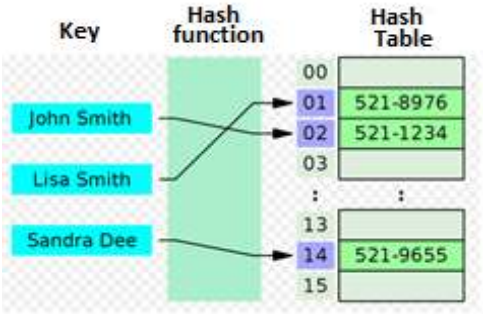
Hashing Techniques تقنية التجزئة

Hash File organization method is the one where data is stored at the data blocks whose address is generated by using hash function.

Provides very fast access to records under certain search conditions. This organization is usually called a hash file.

A **hash table** is a collection of items which are stored in such a way as to make it easy to find them later. Each position of the hash table, often called a **slot**, can hold an item and is named by an integer value starting at 0.

The search condition must be an equality condition on a single field, called the hash field (hash key).



Key → H(K) → Address

The idea behind hashing

1. Provide a function h, called a hash function which is applied to the hash field value of a record and yields the address of the disk block in which the record is stored.
2. A search for the record within the block can be carried out in a main memory buffer.
3. For most records, we need only a single-block access to retrieve that record.

Hash Function

Hash function can be simple mathematical function to any complex mathematical function. There are many type of hashing function Direct, mathematical like mod, sin, etc.

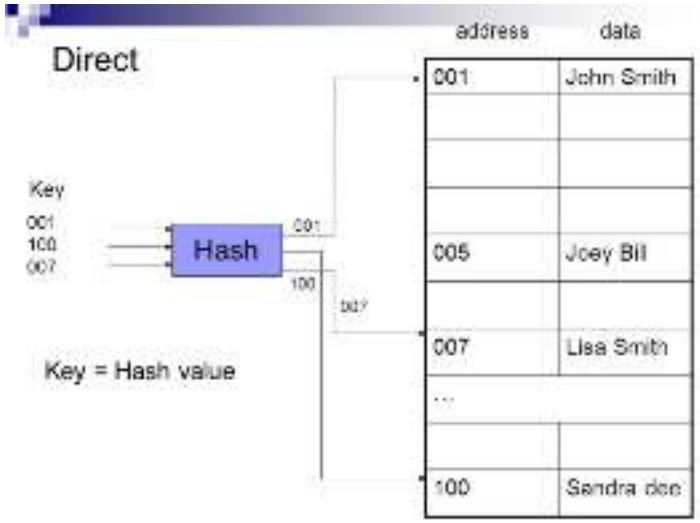
- **Direct Hashing**: the hash function can use any of the column value (Field) to generate the address. Most of the time, hash function uses

primary key to generate the hash index – address of the data block. A key is address without any algorithm manipulation.

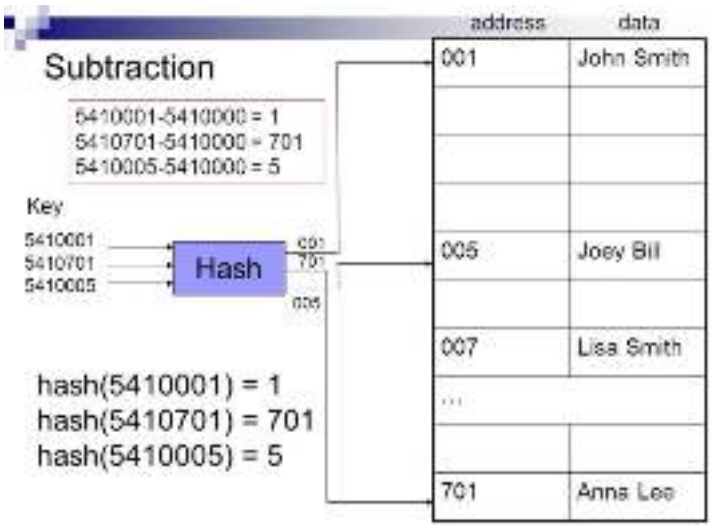
Key → Address

We can even consider primary key itself as address of the data block. That means each row will be stored at the data block whose address will be same as primary key.

1. No collision
2. Good for small data
3. Not suitable to large data



- **Subtraction Hashing**: would be some subtraction value.
  Key → Address
  1. No collision
  2. Good for small data
  3. Not suitable to large data

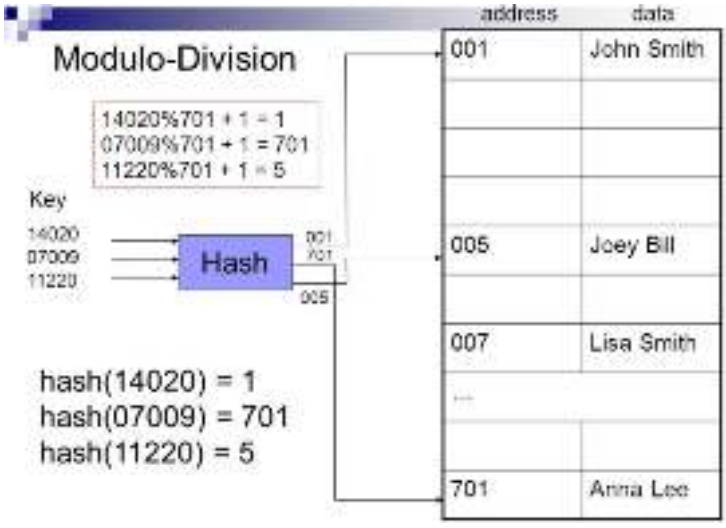


- **Modulo Division Hashing**: this hash function can also be simple mathematical function like mod.
  Key → H(K) →Address

  H (K) → Key modulo size

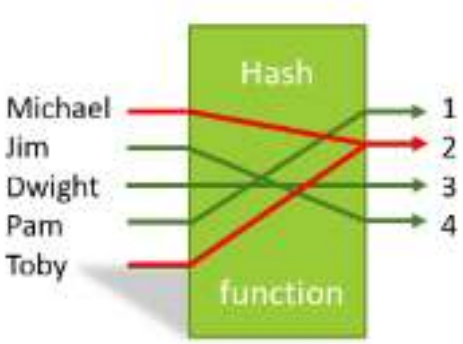
  Imagine we have hash function as mod (5) to determine the address of the data block. So what happens to the above case? It applies mod (5) on primary keys and generates 3,3,1,4 and 2 respectively and the records are stored in those data block addresses.

  1. Collision
  2. Suitable to large data



Hash collisions

When different input values lead to the same output hash value, this is known as a **hash collision**. Consider the following, simplified hash function:



Hash tables deal with collisions in one of two ways.

Collisions can be reduced with a selection of a good hash function but it is hard to do.

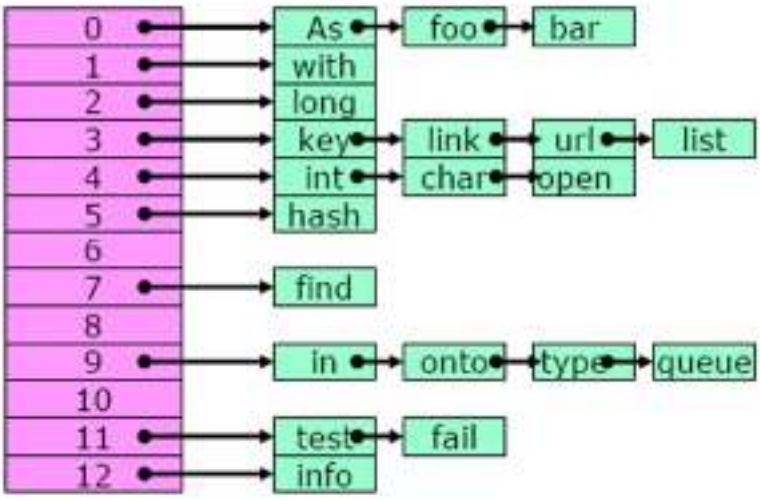
Collision Resolving strategies

Few Collision Resolution ideas

1. Separate chaining
2. Some Open addressing techniques
   - Linear Probing
   - Quadratic Probing

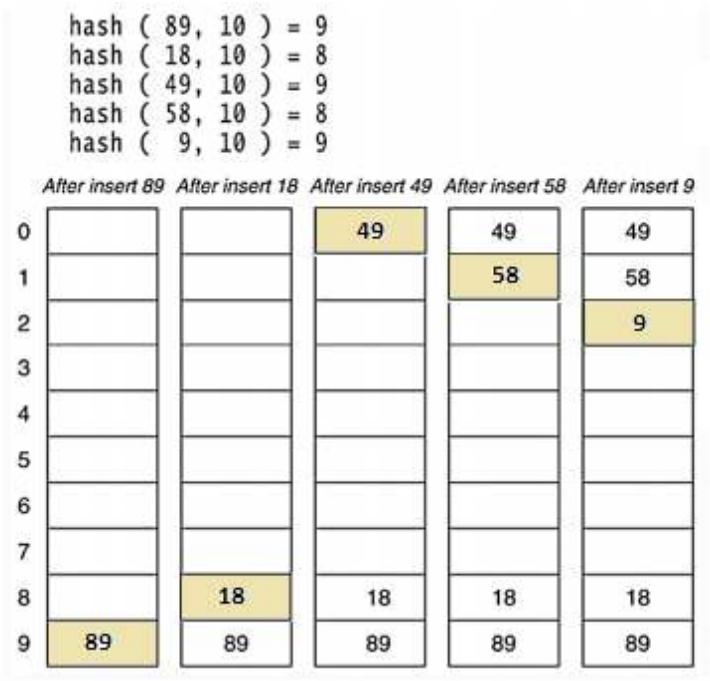
Option 1: Separate chaining

Collisions can be resolved by creating a list of keys that map to the same value



Option 2: Linear Probing

- Table remains a simple array of size N
- On insert(x)
    1. Compute f(x) mod N
    2. If the cell is full
    3. Find another by sequentially searching for the next available slot.



---------- بالتوفيق ----------