UNIT I

ENTITY-RELATIONSHIP(ER) MODEL:

- ER modeling: A graphical technique for understanding and organizing the data independent of the actual database implementation
- Entity: Anything that may have an independent existence and about which we intend to collect data. Also known as Entity type.
- Entity instance: a particular member of the entity type e.g. a particular student
- Attributes: Properties/characteristics that describe entities
- Relationships: Associations between entities

Attributes

- The set of possible values for an attribute is called the domain of the attribute

Example:

  - The domain of attribute marital status is just the four values: single, married, divorced, widowed
  - The domain of the attribute month is the twelve values ranging from January to December

- Key attribute: The attribute (or combination of attributes) that is unique for every entity instance
  - E.g the account number of an account, the employee id of an employee etc.
- If the key consists of two or more attributes in combination, it is called a composite key

Simple Vs composite attribute

- Simple attribute: cannot be divided into simpler components
  E.g age of an employee
- Composite attribute: can be split into components
  E.g Date of joining of the employee.
  – Can be split into day, month and year

Single Vs Multi-valued Attributes

- Single valued : can take on only a single value for each entity instance
  E.g. age of employee. There can be only one value for this
- Multi-valued:can take many values
  E.g. skill set of employee

Stored Vs Derived attribute

- Stored Attribute:Attribute that need to be stored permanently.
- E.g. name of an employee
- Derived Attribute: Attribute that can be calculated based on other attributes
- E.g. : years of service of employee can be calculated from date of joining and Current date

Regular Vs. Weak entity type

- Regular Entity: Entity that has its own key attribute.

E.g.: Employee, student, customer, policy holder etc.

- Weak entity: Entity that depends on other entity for its existence and doesn't have key attribute of its own

E.g.: spouse of employee

RELATIONSHIPS

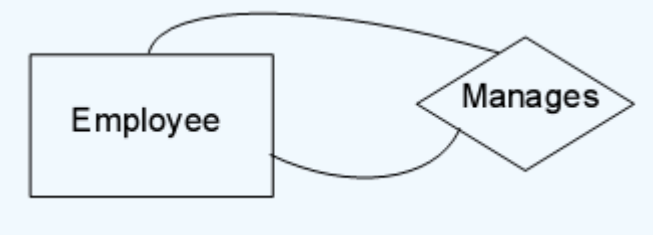
•A relationship type between two entity types defines the set of all associations between these entity types

• Each instance of the relationship between members of these entity types is called a Relationship instance

Degree of a Relationship

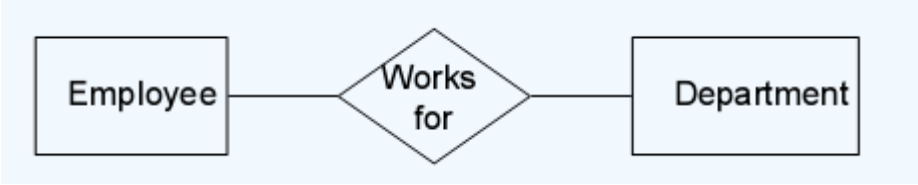
• Degree: the number of entity types involved

•One: Unary Relationship



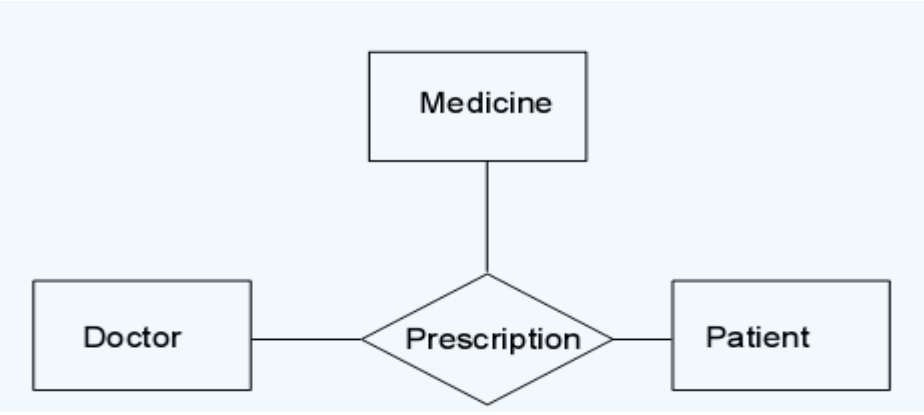
- ✓ A unary relationship is represented as a diamond which connects one entity to itself as a loop.
- ✓ The relationship above means, some instances of employee manage other instances of Employee.
- ✓ E.g.: employee manager-of employee is unary

•Two: Binary Relationship



- ✓ A relationship between two entity types
- ✓ E.g.: employee works-for department is binary

• Three: Ternary Relationship



- ✓ customer purchase item, shop keeper is a ternary relationship

Cardinality

• Relationships can have different connectivity

– one-to-one (1:1)

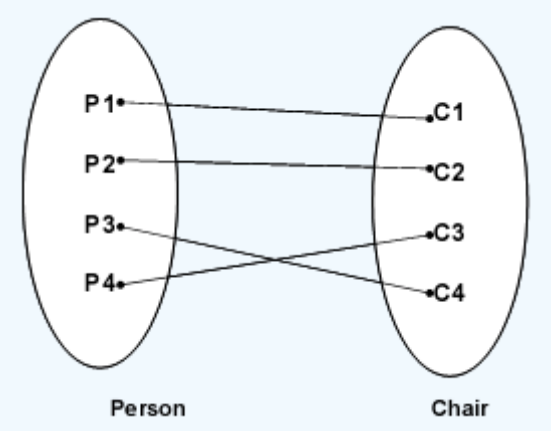
– one-to-many (1:N)

– many-to- One (M:1)
– many-to-many (M:N)

E.g.: Employee head-of department (1:1).
Lecturer offers course (1:n) assuming a course is taught by a single lecturer Student enrolls course (m:n).

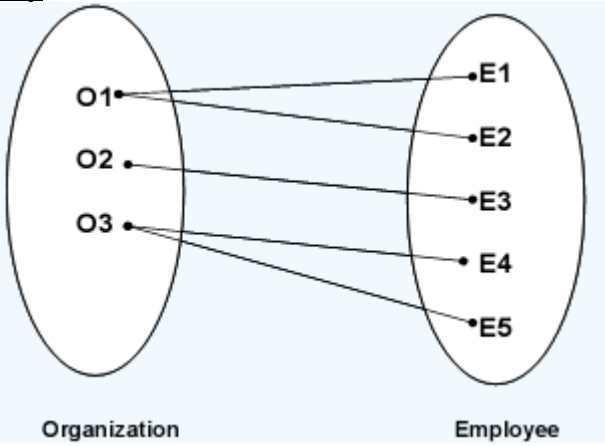
The minimum and maximum values of this connectivity is called the cardinality of the relationship.

Cardinality – One – To – One



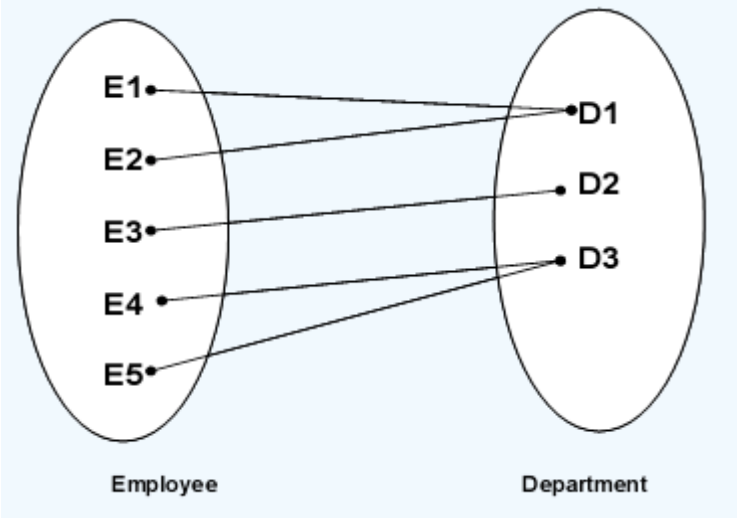
One instance of entity type Person is related to one instance of the entity type Chair.

Cardinality – One –to- Many



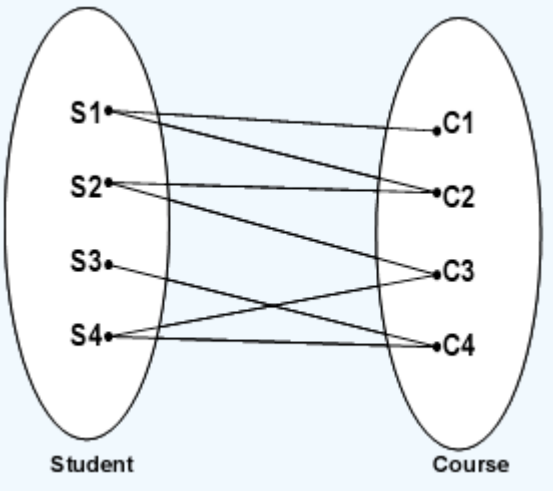
One instance of entity type Organization is related to multiple instances of entity type Employee.

Cardinality – Many-to-One



Reverse of the One to Many relationship

Cardinality – Many-to-Many



Multiple instances of one Entity are related to multiple instances of another Entity.

Relationship Participation

• Total: Every entity instance must be connected through the relationship to another instance of the other participating entity types
• Partial: All instances need not participate
E.g.: Employee Head-of Department
Employee: partial
Department: total
All employees will not be head-of some department. So only few instances of employee entity participate in the above relationship. But each department will be headed by some employee. So department entity's participation is total and employee entity's participation is partial in the above relationship.

COMPONENTS OF DBMS

Database Users

Users are differentiated by the way they expect to interact with the system

- Application programmers
- Sophisticated users
- Naïve users
- Database Administrator
- Specialized users etc,.

- **Application programmers:**

– Professionals who write application programs and using these application programs they interact with the database system

- **Sophisticated users :**

– These user interact with the database system without writing programs, But they submit queries to retrieve the information

- **Specialized users:**

– Who write specialized database applications to interact with the database system.

- **Naïve users:**

– Interacts with the database system by invoking some application programs that have been written previously by application programmers

Eg : people accessing database over the web

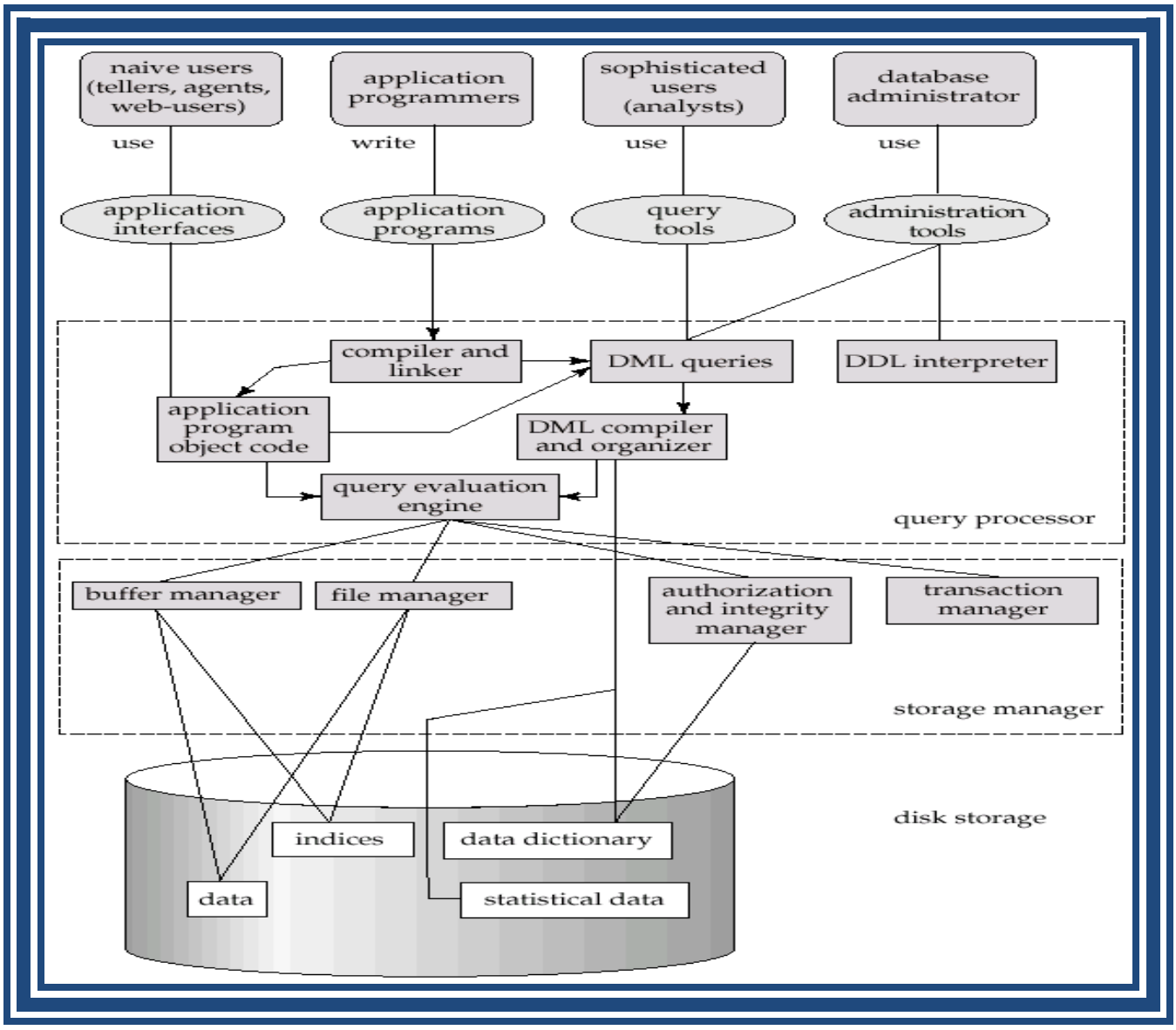
Database Administrator:

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
  - Schema definition
  - Access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Monitoring performance

Storage Manager

- The Storage Manager include these following components/modules
  - Authorization Manager
  - Transaction Manager
  - File Manager
  - Buffer Manager
- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - interaction with the file manager
  - efficient storing, retrieving and updating of data
    - Authorization Manager
      - Checks whether the user is an authorized person or not
      - Test the satisfaction of integrity constraints
    - Transaction Manager
      - Responsible for concurrent transaction execution

It ensures that the database remains in a consistent state despite of the system failure



Query Processor

- It is also a collection of components and used to interprets the queries which is submitted by the user.
- The query processor includes these following components
    - DDL interpreter
    - DML Compiler
    - Query evaluation engine
- DDL interpreter

  -Interprets DDL statements and records the definition in the data dictionary
- DML Compiler
    - Translate the DML Statements into an evaluation plan that contain low level instructions and the query evaluation engine can understand these instruction
    - Query can be translated into many alternative evaluation plans
    - Query optimization –picks up the lowest cost evaluation plan from the alternatives

PURPOSE OF DATABASE SYSTEM

- In the early days, ***File-Processing system*** is used to store records. It uses various files for storing the records.
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Hard to add new constraints or change existing ones
  - Atomicity problem
    - Failures may leave database in an inconsistent state with partial updates carried out
    - E.g. transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access anomalies
    - Concurrent accessed needed for performance
  - Security problems
- Database systems offer solutions to all the above problems

RELATIONAL ALGEBRA:

- It is a procedural language

Basic Operations

- select
- project
- union
- set difference
- Cartesian product
- rename

Select Operation

- It is used to select tuples from a relations
- Notation: $\sigma_p(r)$

  p is called the selection predicate
- Defined as:

  $\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$
- Each term is one of:

  <attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

- Example

  $\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}(account)$

Project Operation

- It is used to select certain columns from the relation.

- Notation:

  $\prod_{A1, A2, \ldots, Ak} (r)$

  Where A_1, A_2 are attributing names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- E.g. To eliminate the *branch-name* attribute of *account* relation

  $\prod_{account\text{-}number,\ balance} (account)$

Union Operation

- For $r \cup s$ to be valid.

  1. *r, s* must have the *same arity* (same number of attributes)

  2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- E.g. to find all customers with either an account or a loan
- $\prod_{customer\text{-}name} (depositor) \cup \prod_{customer\text{-}name} (borrower)$

Set Difference Operation

- It is used to find tuples that are in one relation but are not in another relation.
- Notation $r - s$
- Defined as:

  $r - s = \{t \mid t \in r \text{ and } t \notin s\}$
- Set differences must be taken between *compatible* relations.
  - *r* and *s* must have the *same arity*

Cartesian-Product Operation

- It is used to combine information from two relations.
- Notation $r \times s$
- Defined as:

  $r \times s = \{t\, q \mid t \in r \text{ and } q \in s\}$

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.

Example:

$\rho_x (E)$

returns the expression *E* under the name *X*

FUNCTIONAL DEPENDENCY:

Functional dependency

In a given relation R, X and Y are attributes. Attribute Y is **functionally dependent** on attribute X if each value of X determines **EXACTLY ONE** value of Y, which is represented as X -> Y (X can be composite in nature).

• We say here "x determines y" or "y is functionally dependent on x" X→Y does not imply Y→X

• If the value of an attribute "Marks" is known then the value of an attribute "Grade" is determined since Marks→ Grade

Full functional dependency

X and Y are attributes X Functionally determines Y Note: Subset of X should not functionally determine Y

Marks are fully functionally dependent on STUDENT# COURSE# and **not on sub set of** STUDENT# COURSE#. This means Marks can not be determined either by STUDENT# **OR** COURSE# alone. It can be determined only using STUDENT# **AND** COURSE# together. Hence Marks is fully functionally dependent on STUDENT# COURSE#.

Partial functional dependency

X and Y are attributes. Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.

Course Name, IName, Room# are partially dependent on composite attributes STUDENT# COURSE# because COURSE# alone defines the Course Name, IName, Room#.

Transitive functional dependency

X Y and Z are three attributes. X -> Y, Y-> Z => X -> Z

Room# depends on IName and in turn IName depends on COURSE#. Hence Room# transitively depends on COURSE#.

1NF:

A relation schema is in 1NF:

– If and only if all the attributes of the relation R are atomic in nature.

– Atomic: The smallest levels to which data may be broken down and remain meaningful

2NF:

A Relation is said to be in Second Normal Form if and only if:

– It is in the First normal form, and

– No partial dependency exists between non-key attributes and key attributes.

• An attribute of a relation R that belongs to any key of R is said to be a prime attribute and that which doesn't is a non-prime attribute. To make a table 2NF compliant, we have to remove all the partial dependencies

Note: - All partial dependencies are eliminated

3 NF:

A relation R is said to be in the Third Normal Form (3NF) if and only if

− It is in 2NF and

− No transitive dependency exists between non-key attributes and key attributes.

• STUDENT# and COURSE# are the key attributes.

• All other attributes, except grade are non-partially, non-transitively dependent on key attributes.

• Student#, Course# - > Marks

• Marks -> Grade

Note : - All transitive dependencies are eliminated

Boyce-Codd Normal form – BCNF

A relation is said to be in Boyce Codd Normal Form (BCNF)

- if and only if all the determinants are candidate keys.BCNF relation is a strong 3NF, but not every 3NF relation is BCNF.

Fourth NF:

Fourth normal form (**4NF**) is a normal form used in database normalization. Introduced by Ronald Fagin in 1977, 4NF is the next level of normalization after Boyce–Codd normal form (BCNF). Whereas the second, third, and Boyce–Codd normal forms are concerned with functional dependencies, 4NF is concerned with a more general type of dependency known as a multivalued dependency. A table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies $X \twoheadrightarrow Y$, X is a super key—that is, X is either a candidate key or a superset thereof.

Multivalued dependencies

If the column headings in a relational database table are divided into three disjoint groupings X, Y, and Z, then, in the context of a particular row, we can refer to the data beneath each group of headings as x, y, and z respectively. A multivalued dependency $X \twoheadrightarrow Y$ signifies that if we choose any x actually occurring in the table (call this choice x_c), and compile a list of all the $x_c yz$ combinations that occur in the table, we will find that x_c is associated with the same y entries regardless of z.A **trivial multivalued dependency** $X \twoheadrightarrow Y$ is one where either Y is a subset of X, or X and Y together form the whole set of attributes of the relation.A functional dependency is a special case of multivalued dependency. In a functional dependency $X \rightarrow Y$, every x determines *exactly one* y, never more than one.

Pizza Delivery Permutations

Restaurant	Pizza Variety	Delivery Area
A1 Pizza	Thick Crust	Springfield
A1 Pizza	Thick Crust	Shelbyville
A1 Pizza	Thick Crust	Capital City
A1 Pizza	Stuffed Crust	Springfield
A1 Pizza	Stuffed Crust	Shelbyville
A1 Pizza	Stuffed Crust	Capital City
Elite Pizza	Thin Crust	Capital City
Elite Pizza	Stuffed Crust	Capital City
Vincenzo's Pizza	Thick Crust	Springfield
Vincenzo's Pizza	Thick Crust	Shelbyville
Vincenzo's Pizza	Thin Crust	Springfield
Vincenzo's Pizza	Thin Crust	Shelbyville

Varieties By Restaurant

Restaurant	Pizza Variety
A1 Pizza	Thick Crust
A1 Pizza	Stuffed Crust
Elite Pizza	Thin Crust
Elite Pizza	Stuffed Crust
Vincenzo's Pizza	Thick Crust
Vincenzo's Pizza	Thin Crust

Delivery Areas By Restaurant

Restaurant	Delivery Area
A1 Pizza	Springfield
A1 Pizza	Shelbyville
A1 Pizza	Capital City
Elite Pizza	Capital City
Vincenzo's Pizza	Springfield
Vincenzo's Pizza	Shelbyville

FIFTH NORMAL FORM

Fifth normal form (**5NF**), also known as **Project-join normal form** (**PJ/NF**) is a level of database normalization designed to reduce redundancy in relational databases recording multi-valued facts by isolating semantically related multiple relationships. A table is said to be in the 5NF if and only if every join dependency in it is implied by the candidate keys.

A join dependency *{A, B, … Z} on R is implied by the candidate key(s) of R if and only if each of A, B, …, Z is a super key for R

Travelling Salesman Product Availability By Brand

Travelling Salesman	Brand	Product Type
Jack Schneider	Acme	Vacuum Cleaner
Jack Schneider	Acme	Breadbox
Willy Loman	Robusto	Pruning Shears
Willy Loman	Robusto	Vacuum Cleaner
Willy Loman	Robusto	Breadbox
Willy Loman	Robusto	Umbrella Stand
Louis Ferguson	Robusto	Vacuum Cleaner
Louis Ferguson	Robusto	Telescope
Louis Ferguson	Acme	Vacuum Cleaner
Louis Ferguson	Acme	Lava Lamp
Louis Ferguson	Nimbus	Tie Rack

Product Types By Travelling Salesman

Travelling Salesman	Product Type
Jack Schneider	Vacuum Cleaner
Jack Schneider	Breadbox
Willy Loman	Pruning Shears
Willy Loman	Vacuum Cleaner
Willy Loman	Breadbox
Willy Loman	Umbrella Stand
Louis Ferguson	Telescope
Louis Ferguson	Vacuum Cleaner
Louis Ferguson	Lava Lamp
Louis Ferguson	Tie Rack

Brands By Travelling Salesman

Travelling Salesman	Brand
Jack Schneider	Acme
Willy Loman	Robusto
Louis Ferguson	Robusto
Louis Ferguson	Acme
Louis Ferguson	Nimbus

Product Types By Brand

Brand	Product Type
Acme	Vacuum Cleaner
Acme	Breadbox
Acme	Lava Lamp
Robusto	Pruning Shears
Robusto	Vacuum Cleaner
Robusto	Breadbox
Robusto	Umbrella Stand
Robusto	Telescope
Nimbus	Tie Rack

DOMAIN/KEY NORMAL FORM (DKNF)

Domain/key normal form (DKNF) is a normal form used in database normalization which requires that the database contains no constraints other than domain constraints and key constraints.

DENORMALIZATION

Demoralization is the process of attempting to optimize the performance of a database by adding redundant data or by grouping data. In some cases, demoralizations helps cover up the inefficiencies inherent in relational database software. A relational normalized database imposes a heavy access load over physical storage of data even if it is well tuned for high performance.

DATA MODELS:

A data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.

- It is underlying structure of database
- It is collection of conceptual tolls for describing data, relationship and constraints.

Categories:

- Entity-Relationship model
- Relational model
- Object based data model
- Semi structured data model

1) **Relational model**

It uses a collection of tables to represent the data and the relationships among those data. Each table contains many columns and each has a unique name. It is the widely used data model and most of the database system is based on relational model.

2) **Object based data model**

Object based data model can be seen as extending the E-R model with notion of encapsulation, methods and object indentify.

3) **Semi structured data model**

- Specification of data where individual data item of same type may have different sets of attributes
- Sometimes called schema less or self-describing
- XML is widely used to represent this data model

4) **Entity-Relationship model**

The entity-relationship (ER) data model allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships. It is widely used to develop an initial database design.

LOGICAL DATABASE DESIGN: RELATIONAL DBMS

CONVERTING STRONG ENTITY TYPES

- Each entity type becomes a table
- Each single-valued attribute becomes a column
- Derived attributes are ignored
- Composite attributes are represented by components
- Multi-valued attributes are represented by a separate table
- The key attribute of the entiry type becomes the primary key of the table

ENTITY EXAMPLE

- Address is a composite attribute
- Years of service is a derived attribute (can be calculated from date of joining and current date)
- Skill set is a multi-valued attribute
- The relational Schema

Employee (E#, Name, Door_No, Street, City, Pincode, Date_Of_Joining)
Emp_Skillset(E#, Skillset)

CONVERTING WEAK ENTITY TYPES

- Weak entity types are converted into a table of their own, with the primary key of the strong entity acting as a foreign key in the table
- This foreign key along with the key of the weak entity form the composite primary key of this table
- The Relational Schema Employee (E# ,…….)

Dependant (Employee, Dependant_ID, Name, Address)

- Dependant is a weak entity. Dependant doesn't mean anything to the problem without the information on for which employee the person is a dependant.

CONVERTING RELATIONSHIPS

• The way relationships are represented depends on the cardinality and the degree of the relationship

• The possible cardinalities are:

1:1, 1:M, N:M

• The degrees are:

Unary
Binary
Ternary

Binary 1:1

• **Case 1:** Combination of participation types

The primary key of the partial participant will become the foreign key of the total participant .

Employee(E#, Name,…)
Department (Dept#, Name…,Head)

• Case 2: Uniform participation types

The primary key of either of the participants can become a foreign key in the other

Employee (E#,name…)
Chair(item#, model, location, used_by)
(or)
Employee (E#, Name….Sits_on)
Chair (item#,….)

Binary 1:N

The primary key of the relation on the "1" side of the relationship becomes a foreign key in the relation on the "N" side

Eg:Teacher (ID, Name, Telephone, …)
Subject (Code, Name, …, Teacher)

Binary M:N

• A new table is created to represent the relationship
• Contains two foreign keys – one from each of the participants in the relationship
• The primary key of the new table is the combination of the two foreign keys

Student (Sid#,Title…) Course(C#,Cname,…)
Enrolls (Sid#, C#)

Unary 1:1

• Consider employees who are also a couple
• The primary key field itself will become foreign key in the same table Employee(E#, Name,… Spouse)

Unary 1:N

• The primary key field itself will become foreign key in the same table
• Same as unary 1:1

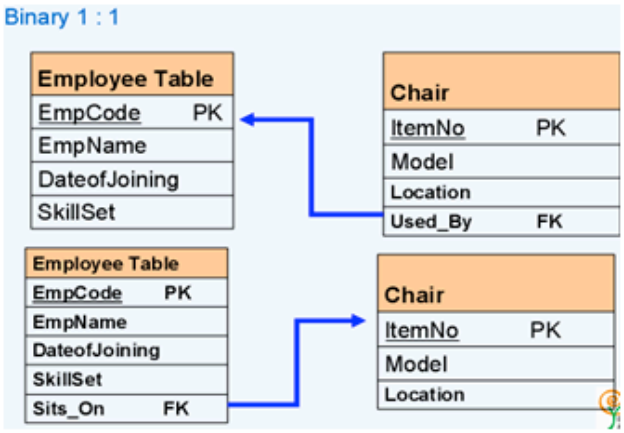
Unary M:N

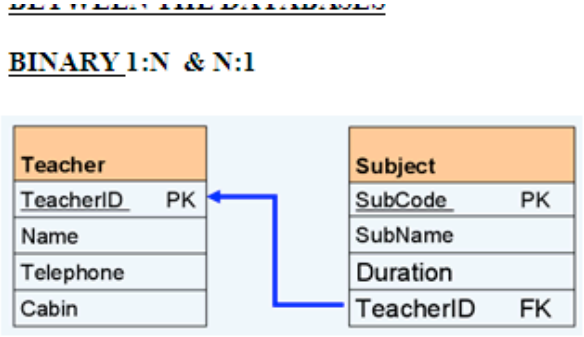
• There will be two resulting tables. One to represent the entity and another to represent the M:N relationship as follows

Employee(E#, Name,…)
Guaranty(Guarantor, beneficiary)

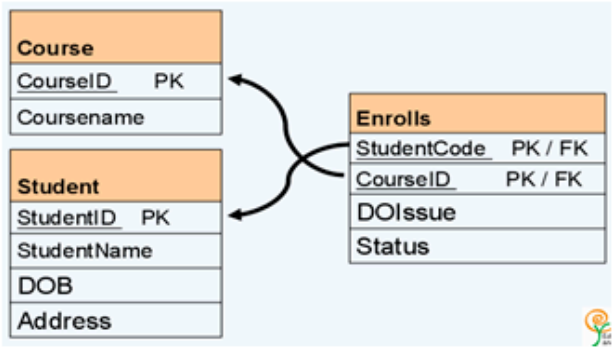
TERNARY RELATIONSHIP

- Represented by a new table
- The new table contains three foreign keys – one from each of the participating Entities
- The primary key of the new table is the combination of all three foreign keys
- Prescription (Doctor#, Patient #, Medicine_Name)

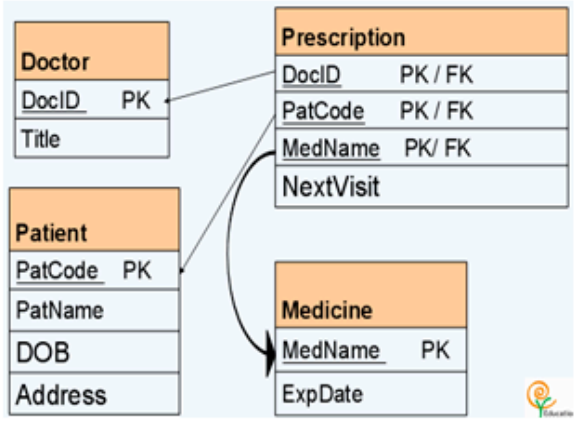




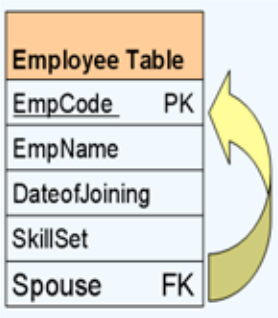
BINARY N:M



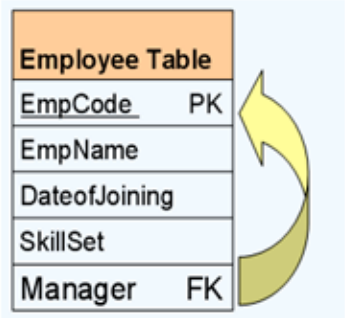
TERNARY



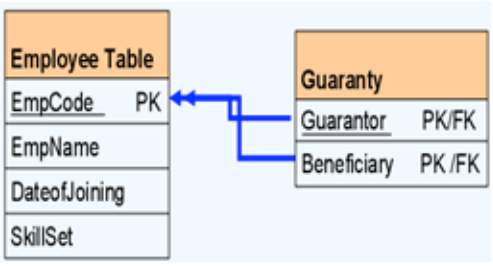
UNARY 1:1



UNARY 1:N &N:1



UNARY N:M



CODD'S RULE:

Rule 0: The Foundation rule

For a system to qualify as a RELATIONAL DATABASE MANAGEMENT system, that system must use its RELATIONAL facilities to MANAGE the DATABASE.

Rule 1: Information rule

This rule states that all information (data), which is stored in the database, must be a value of some table cell. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

This rule states the NULL values in the database must be given a systematic treatment. As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active online catalog

This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application. If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View updating rule

This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level insert, update and delete rule

This rule states the database must employ support high-level insertion, updation and deletion. This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records

Rule 8: Physical data independence

This rule states that the application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application

Rule 9: Logical data independence

This rule states that the logical data must be independent of its user's view (application).

Any change in logical data must not imply any change in the application using it.

For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rule to apply.

Rule 10: Integrity independence

This rule states that the database must be independent of the application using it.

All its integrity constraints can be independently modified without the need of any change in the application.

This rule makes database independent of the front-end application and its interface.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations.

User must also see that data is located at one site only.

This rule has been proven as a foundation of distributed database systems

Rule 12: Non-subversion rule

This rule states that if a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

ENHANCED ENTITY-RELATIONSHIP MODEL

One entity type might be a subtype of another--very similar to subclasses in OO programming

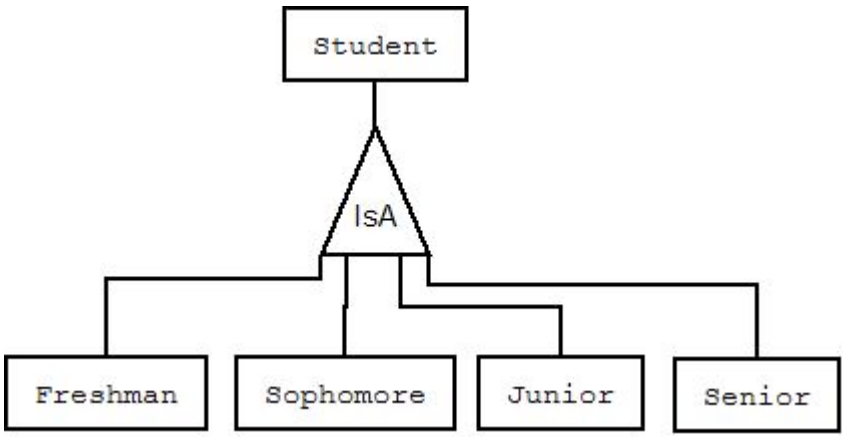
- **Freshman** is a subtype of **Student**

A relationship exists between a **Freshman** entity and the corresponding **Student** entity

- e.g., **Freshman** John is related to **Student** John

This relationship is called ***IsA***

- Freshman IsA Student, an eagle IsA bird
- The two entities related by IsA are always descriptions of the same real-world object
- Typically used in databases to be implemented as Object Oriented Models.
- The upper entity type is the more abstract entity type (super type) from which the lower entities inherit its attributes



Properties of IsA

Inheritance - All attributes of the supertype apply to the subtype.

- E.g., GPA attribute of **Student** applies to **Freshman**
- The subtype inherits all attributes of its supertype.
- The key of the supertype is also the key of the subtype

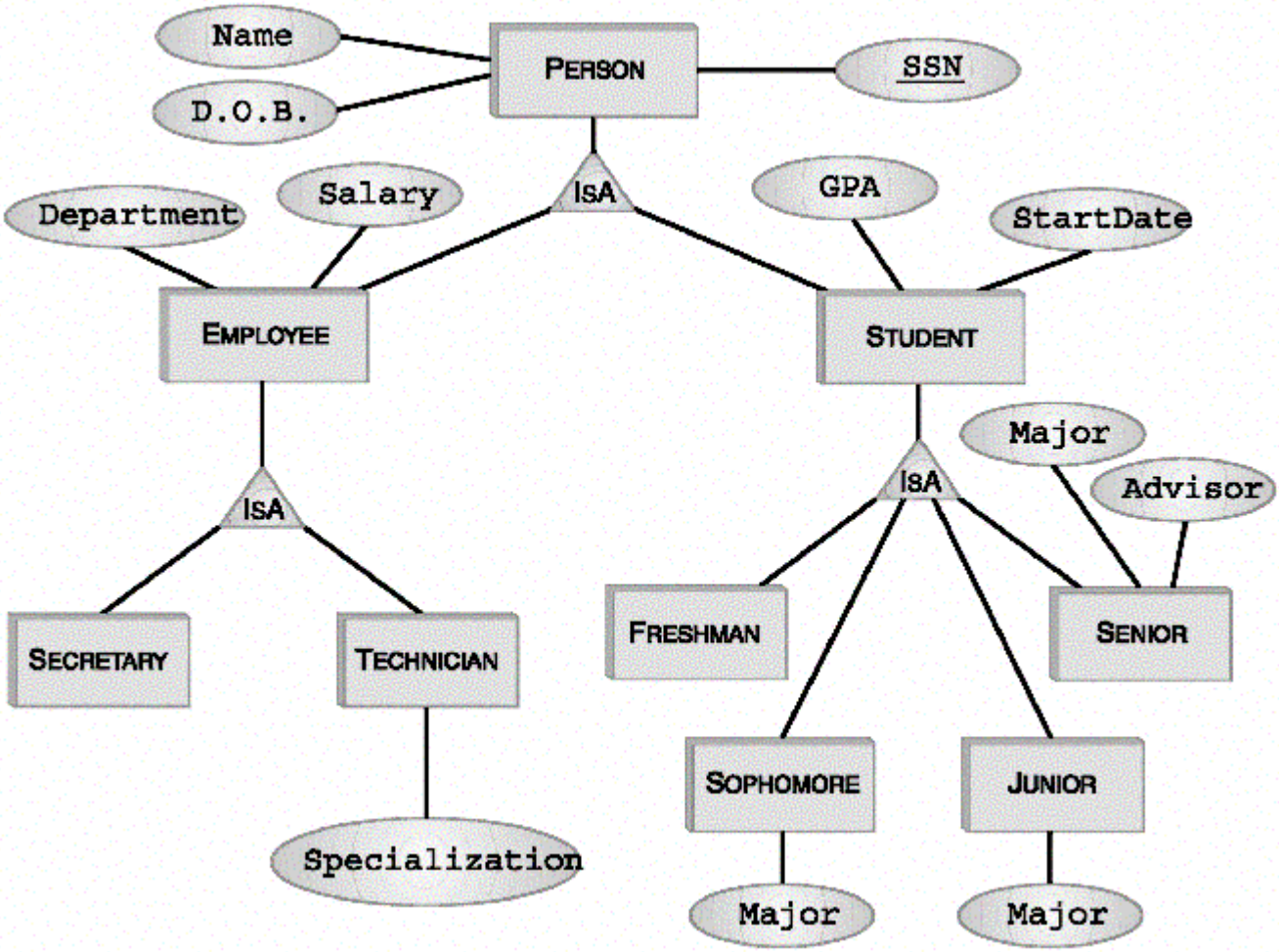
Transitivity - This property creates a hierarchy of IsA relationships

- **Student** is subtype of **Person**,
  Freshman is subtype of **Student**,
  therefore **Freshman** is also a subtype of **Person**

Advantage: Used to create a more concise and readable E-R diagram. It best maps to object oriented approaches either to databases or related applications.

- Attributes common to different entity sets need not be repeated
- They can be grouped in one place as attributes of the supertype
- Attributes of (sibling) subtypes are likely to be different (and should be for this to be very useful)

Example of IsA



CHARACTERISTICS OF DATABASE:

- Self-describing nature of a database system: A DBMS **catalog** stores the description of the database. The description is called **meta-data**). This allows the DBMS software to work with different databases.
- Insulation between programs and data: Called **program-data independence**. Allows changing data storage structures and operations without having to change the DBMS access programs.
- Data Abstraction: A **data model** is used to hide storage details and present the users with a conceptual view of the database.
- Support of multiple views of the data: Each user may see a different view of the database, which describes only the data of interest to that user.
- Sharing of data and multiuser transaction processing : allowing a set of concurrent users to retrieve and to update the database. Concurrency control within the DBMS guarantees that each **transaction** is correctly executed or completely aborted. OLTP (Online Transaction Processing) is a major part of database applications.

FILE ORGANIZATION

- The database is stored as a collection of ***files***.
- Each file is a sequence of ***records***.
- A **record** is a sequence of fields.
- Classifications of records
    - **Fixed length record**
    - **Variable length record**

FIXED-LENGTH RECORDS

- Fixed length record approach:
    - assume record size is fixed
    - each file has records of one particular type only
    - different files are used for different relations

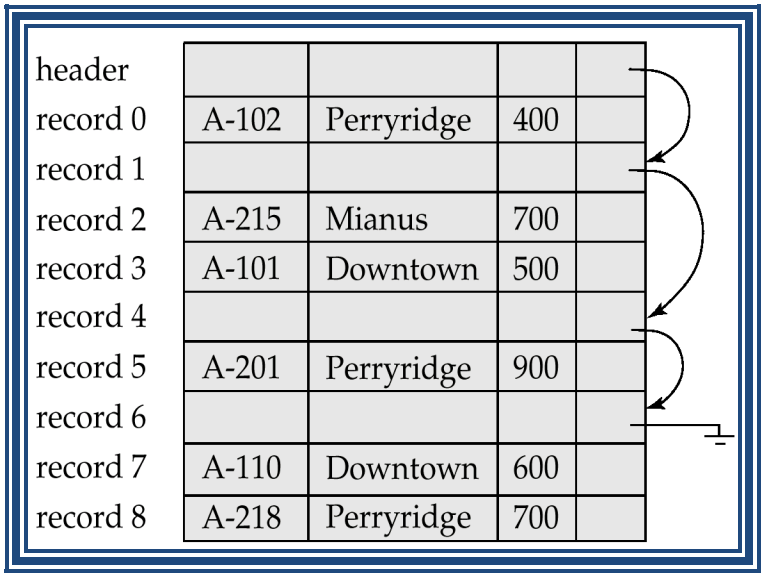
This case is easiest to implement.

- Simple approach:
    - Record access is simple
        - Modification: do not allow records to cross block boundaries
- Deletion of record *I:*
  alternatives*:*
    - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
    - do not move records, but link all free records on a *free list*

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Free Lists

- Store the address of the first deleted record in the file header.
- Use this first record to store the address of the second deleted record, and so on

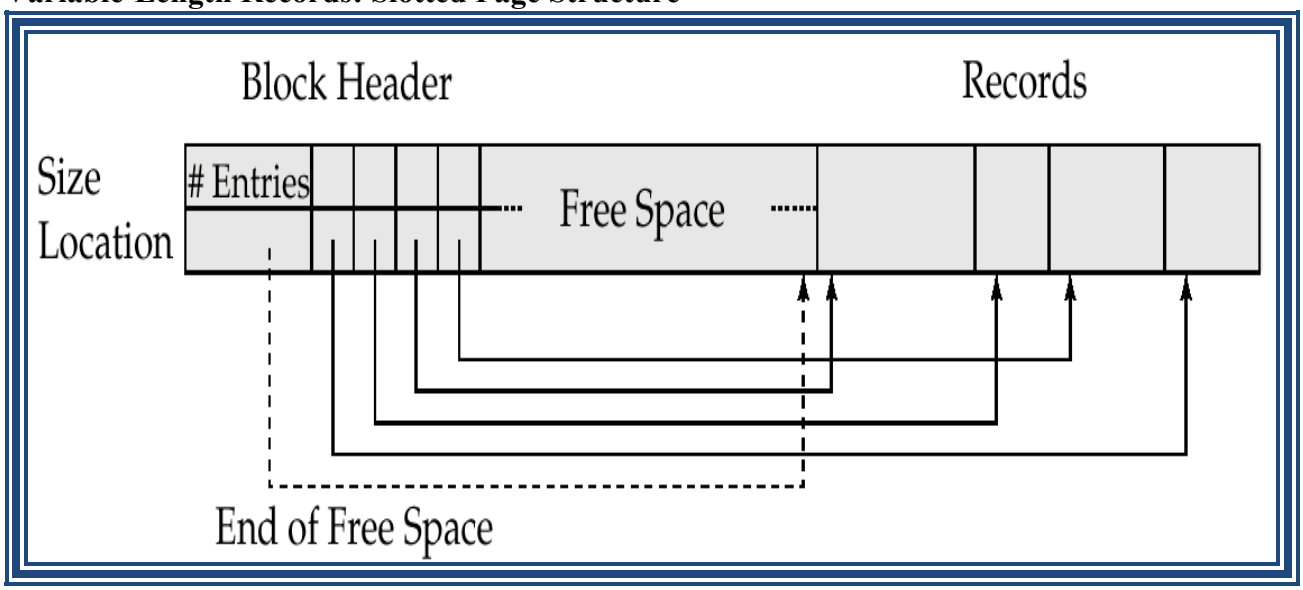


VARIABLE-LENGTH RECORDS

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields.
- Byte string representation
  - Attach an *end-of-record* (⊥) control character to the end of each record

Difficulty with deletion

Variable-Length Records: Slotted Page Structure

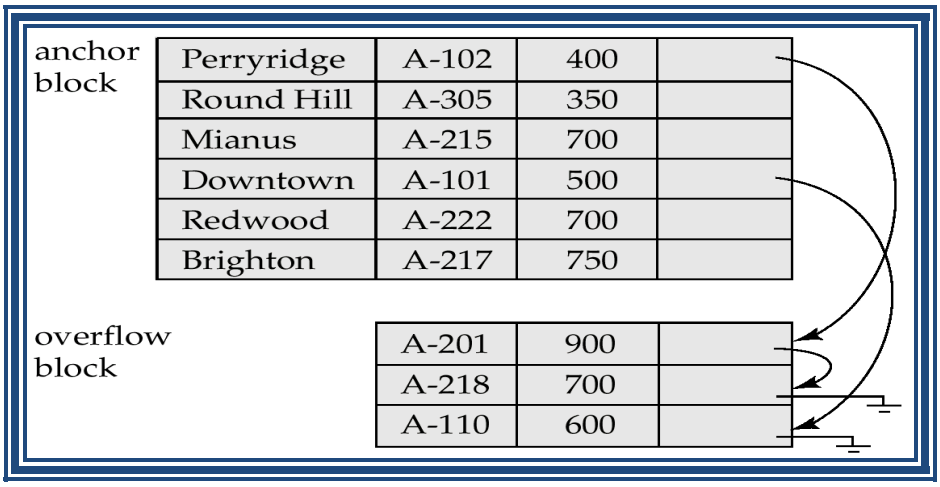


- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record

POINTER METHOD

- Pointer method
  - A variable-length record is represented by a list of fixed-length records, chained together via pointers.
  - Can be used even if the maximum record length is not known

- Disadvantage to pointer structure; space is wasted in all records except the first in a a chain.
- Solution is to allow two kinds of block in file:
  - Anchor block – contains the first records of chain
  - Overflow block – contains records other than those that are the first records of chains.

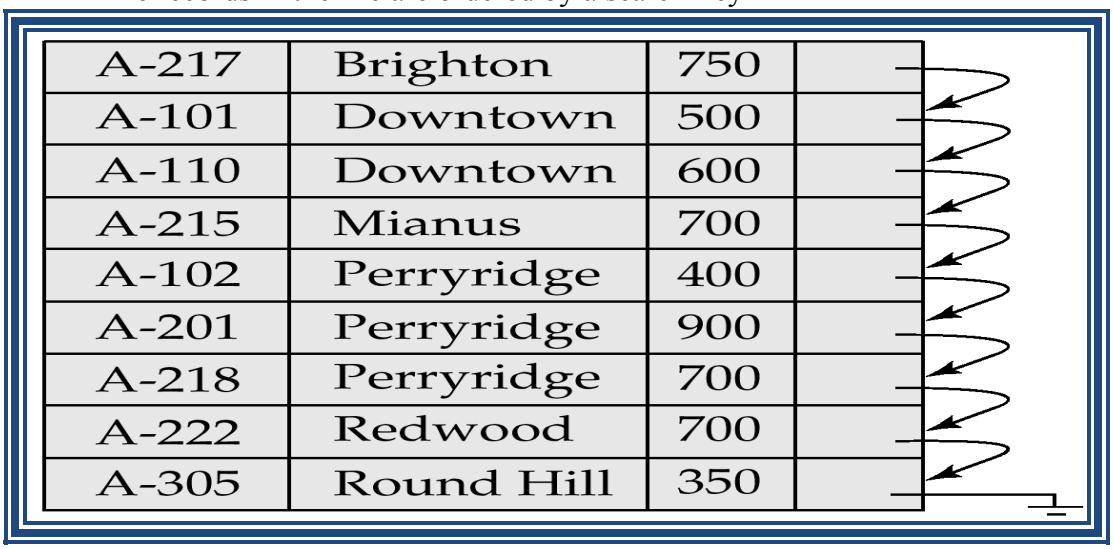


ORGANIZATION OF RECORDS IN FILES

- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Heap** – a record can be placed anywhere in the file where there is space
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there

- if no free space, insert the record in an overflow block
- In either case, pointer chain must be updated