• Flat file system:
  – One or more data files accessed via dedicated programs

• Typical large organization (e.g., university) has many departments, each with specific needs
  – Each department has own set of data
  – Each department has own set of apps for processing data
  – Data stored in one or more data files accessed by app programs which define
    * Record/field structure (object/attribute structure)
    – Apps written in high-level language
    – Generally, significant overlap in data stored in various departments

• This approach leads to many problems:
  1. Data redundancy
     – Data stored multiple places
     – Ramifications:
       (a) Wasted resources
         * Wasted disk space
         * Wasted effort
         * Both result in wasted money
       (b) Inconsistent data (as a result of updates)
         * Doubtful if a datum that occurs in multiple files will be updated simultaneously
           · Files will be inconsistent for a time interval
         * Clerical errors may result in permanent inconsistencies
  2. Concurrency control
     – Have many independent programs with uncontrolled access to same data
  3. Interdependence of data and programs
     – Data structure of DB embedded in app programs
     – Based on structures defined using data types inherent in host language
     – Ramifications:
       (a) Difficulty in modifying DB/apps
         * If modify record structure or field data type, all apps must be modified accordingly
       (b) Difficulty sharing data (among departments)
         * Most likely incompatible record formats in different locations, so cannot share files easily
       (c) Difficulty extracting new info
         * If want to extract info not anticipated at design time, either
           i. Extract manually using results from existing software
           ii. Create new app
     – The interdependence tends to promote a proliferation of apps
       * This tends to result in much ad hoc code
       * This in turn tends to result in problems related to
         (a) Data integrity
           · Checks that data meet certain constraints
         (b) Security
           · Limiting users to authorized data
         (c) Concurrency control
           · Limiting number of users accessing a given piece of data at any one time

• These problems motivated the development of DBMSs
• Conference on Data Systems Languages (CODASYL)
  – Purpose was to establish standards for DBs
  – 1967 - created Database Task Group (DBTG)
  – DBTG charged with generating standards for environment for creation of DB and manipulation of data
  – 1969 - DBTG initial report
• Codd
  – 1970 - seminal paper proposing relational model
  – Proposed 8 services that should be supported by any full DBMS:
    1. Data storage, retrieval, and update
       * Primary purpose of DBMS
       * Physical details of data storage should be hidden from user
    2. User-accessible catalog
       * Stores metadata - data about data
       * Stores info pertaining to all aspects of DB design, usage, and maintenance
    3. Transaction support
       * Transaction: an atomic operation on the DB
         · Consists of a read and/or write
       * Transaction executes in its entirety, or not at all
       * Insures consistency of DB in case of DBMS failure
    4. Concurrency control
    5. Recovery
       * Enables DB to be returned to a consistent state in case of failure
    6. Restriction of unauthorized access
    7. Support for data communication
       * Provide for remote access of DB
    8. Integrity support
       * Insure data is correct and consistent
• DBTG
  – 1971 - formal proposal
  – DBMS should consist of 3 components:
    1. Network schema
       * Describes logical organization of DB
       * Includes
         (a) DB name
         (b) Structure of each record type
         (c) Data types of each record field
    2. Subschema
       * Describes DB as seen by users
    3. Data management language
       * Used to define structure of data
       * Used to manipulate data
       * Proposed 3 sub-languages:
         (a) Schema data definition language (DDL)
            · Defines schema
            · Used by DBA
(b) Subschema DDL
   · Defines parts of DB required by apps

(c) Data manipulation language
   · For manipulation of data
   · Used by anyone querying DB
   - Not adopted by ANSI

• ANSI Standards Planning and Requirements Committee (SPARC)
  - 1975 - proposed 3-level architecture with data dictionary
  - Based on IBM (Codd) proposals
  - Reflected need for independent layer between implementational and application levels
  - Purpose of 3-level architecture:
    * Users should be able to access same data, but with customized view
    * Should be able to change one view without affecting others
    * Physical data storage should be invisible to user
    * Changes to physical storage should not affect user view
    * Changes to physical storage should not affect internal structure of DB
    * Changes to internal structure should not affect user view

• ANSI-SPARC proposal did not become formal standard
  - Is basis for modern DBMS architectures

Intro: Data Models

• Data model:
  - Integrated collection of concepts for describing data, relationships between data, and constraints on data in an organization (Connelly and Begg)
  - Schemas represented in terms of a data model

• Data models are used at each level of the DBMS

• Models do not need to be the same across DBMS levels

• Data models are categorized in terms of their degree of abstraction

1. High-level/conceptual/object-based models
   - Highest degree of abstraction
   - Describe DB in terms of
     (a) Entities
        * Concepts to be represented
     (b) Attributes
        * Characteristics of the entities
     (c) Relationships
        * Associations among entities
   - Example paradigms:
     (a) Entity-relation (ER) model
     (b) Object-oriented
     (c) Semantic
     (d) Functional

2. Representational/Implementational/record-based models
   - Describe DB in terms of logical records
   - Correspond closely to way data represented physically, while hiding the details
Applicable to external and conceptual DBMS levels
Better than OO for representing structure
Poorer than OO for representing constraints
Example paradigms:
(a) Relational model
   * Represent DB as tables
(b) Network (legacy)
   * Represent DB as collections of records
   * Represent relations as sets of records
   * Graph-based representation
(c) Hierarchical (legacy)
   * Same representation as Network, but record may only have 1 parent
   * Tree-based representation
Relational preferred because the other 2 require knowledge of physical representation

3. Physical models
   Describe DB at physical level
   Example paradigms:
   (a) Unifying
   (b) Frame memory