1

From File Systems to Databases

Many Slides from Chapter 1 of

Database Systems: Design, Implementation, and Management, Fifth Edition, Rob and Coronel

1

Importance of DBMS

- Makes data management more efficient and effective
- Query language allows quick answers to *ad hoc* queries
- Provides better access to more and better-managed data
- Promotes integrated view of organization's operations
- Reduces the probability of inconsistent data

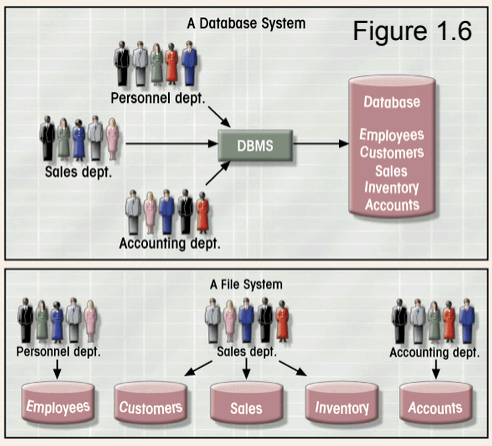
Historical Roots of Database

- First applications focused on clerical tasks
- Requests for information quickly followed
- File systems developed to address needs
  - Data organized according to expected use
  - Data Processing (DP) specialists computerized manual file systems

Database Systems

- Database consists of logically related data stored in a single repository
- Provides advantages over file system management approach
  - Eliminates inconsistency, data anomalies, data dependency, and structural dependency problems
  - Stores data structures, relationships, and access paths

Database vs. File Systems



Database System Types

- Single-user vs. Multiuser Database
  - Desktop
  - Workgroup
  - Enterprise
- Centralized vs. Distributed
- Use
  - Production or transactional
  - Decision support or data warehouse

DBMS Functions

- Data dictionary management
- Data storage management
- Data transformation and presentation
- Security management
- Multiuser access control
- Backup and recovery management
- Data integrity management
- Database language and application programming interfaces
- Database communication interfaces

Database Models

- Collection of logical constructs used to represent data structure and relationships within the database
  - Conceptual models: logical nature of data representation
  - Implementation models: emphasis on how the data are represented in the database

Database Models (con't.)

- Relationships in Conceptual Models
  - One-to-one (1:1)
  - One-to-many (1:M)
  - Many-to-many (M:N)
- Implementation Database Models
  - Hierarchical
  - Network
  - Relational

Hierarchical Database Model

- Logically represented by an upside down tree
  - Each parent can have many children
  - Each child has only one parent

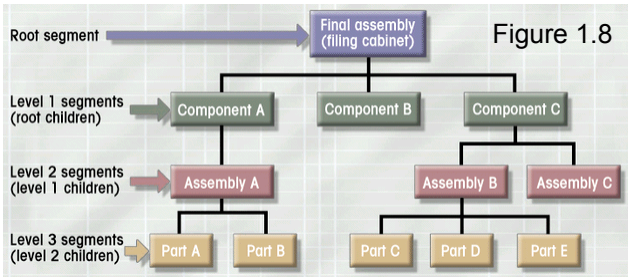


Figure 1.8

Network Database Model

- Each record can have multiple parents
  - Composed of sets
  - Each set has owner record and member record
  - Member may have several owners

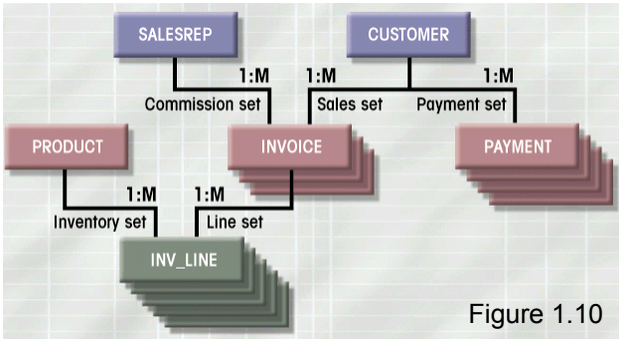


Figure 1.10

Relational Database Model (70s)

- Tables are a series of row/column intersections
- Logical Model … with great mathematical foundations—initially expected to be very inefficient
- Twenty years of R&D changed that dramatically: e.g. B+-trees & Optimizers have produced
  - Superior Performance with Data Independence
- Others Pluses: Standards & Strong vendors make RDBMS unbeatable
- Alternatives have resulted in extensions and improvements, rather than replacements.

Relational DBMS Evolution

- ER model
- Recursion
- O-O
- OLAP and decision support
- The web and XML, …

Entity Relationship Database Model (Peter Chen '76)

- Complements the relational data model concepts
- Represented in an entity relationship diagram (ERD)
- Based on entities, attributes, and relationships

Figure 1.13

Entity Relationship Database Model

- **Advantages**
  - **Exceptional conceptual simplicity**
  - **Visual representation**
  - **Effective communication tool**
  - **Integrated with the relational database model**
- **Disadvantages**
  - **No data manipulation language**
- **ER Model became a powerful design tool in DB design!**

Recursion in SQL (80s)

- SQL:1999 permits recursive view definition
- Example: find all employee-manager pairs, where the employee reports to the manager directly or indirectly (that is manager's manager, manager's manager's manager, etc.)

```
with recursive empl (employee_name, manager_name ) as (
        select employee_name, manager_name
        from   manager
    union
        select manager.employee_name, empl.manager_name
        from   manager, empl
        where manager.manager_name = empl.employe_name)
select *
from   empl
```

This example view, *empl,* is called the *transitive closure* of the *manager* relation

The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *manager* with itself
    - This can give only a fixed number of levels of managers
    - Given a program we can construct a database with a greater number of levels of managers on which the program will not work
- Computing transitive closure
  - The next slide shows a *manager* relation
  - Each step of the iterative process constructs an extended version of *empl* from its recursive definition.
  - The final result is called the *fixed point* of the recursive view definition.
- Recursive views are required to be *monotonic.* That is, if we add tuples to *manger* the view contains all of the tuples it contained before, plus possibly more

Example of Fixed-Point Computation

employee_name	*manager_name*
Alon	Barinsky
Barinsky	Estovar
Corbin	Duarte
Duarte	Jones
Estovar	Jones
Jones	Klinger
Rensal	Klinger

Iteration number	*Tuples in empl*
0	
1	(Duarte), (Estovar)
2	(Duarte), (Estovar), (Barinsky), (Corbin)
3	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)
4	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)

Object-Relational Data Models

- Small Talk and C++ made OO very popular (90s)
- OODBMS featuring persistent types gained some popularity
- Vendors Extended the relational data model by including object orientation and constructs to deal with added data types (ORDBMS--SQL 2003).
- ORDBMS:
  - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
  - Upward compatibility with existing relational languages.

Complex Data Types

- Motivation:
  - Permit non-atomic domains (atomic ≡ indivisible)
  - Example of non-atomic domain: set of integers,or set of tuples
  - Allows more intuitive modeling for applications with complex data
- Intuitive definition:
  - allow relations whenever we allow atomic (scalar) values — relations within relations
  - Retains mathematical foundation of relational model
  - Violates first normal form.

Structured Types and Inheritance in SQL

- Structured types can be declared and used in SQL

```
create type Name as
    (firstname      varchar(20),
     lastname       varchar(20))
     final

create type Address as
    (street         varchar(20),
     city           varchar(20),
     zipcode        varchar(20))
     not final
```

  - Note: **final** and **not final** indicate whether subtypes can be created
- Structured types can be used to create tables with composite attributes

```
create table customer (
    name        Name,
    address     Address,
    dateOfBirth date)
```

- Dot notation used to reference components: *name.firstname*

Structured Types (cont.)

- User-defined row types

```
create type CustomerType as (
    name Name,
    address Address,
    dateOfBirth date)
```

- Can then create a table whose rows are a user-defined type

```
create table customer of CustomerType
```

Methods

- Can add a method declaration with a structured type.

  method *ageOnDate* (*onDate* **date**)

  returns interval year

- Method body is given separately.

  create instance method *ageOnDate* (*onDate* **date**)

  returns interval year

  for *CustomerType*

  begin

  return *onDate* - **self**.*dateOfBirth*;

  end

- We can now find the age of each customer:

  select *name.lastname, ageOnDate* (**current_date**)

  from *customer*

Inheritance

- Suppose that we have the following type definition for people:

  create type *Person*
  (*name* **varchar**(20),
  address **varchar**(20))

- Using inheritance to define the student and teacher types

  create type *Student*
  under *Person*
  (*degree* **varchar**(20),
  department **varchar**(20))
  create type *Teacher*
  under *Person*
  (*salary* **integer**,
  department **varchar**(20))

- Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

Object-Identity and Reference Types

- Define a type *Department* with a field *name* and a field *head* which is a reference to the type *Person,* with table *people* as scope:

  create type *Department* (
  name **varchar** (20),
  head **ref** (*Person*) **scope** *people*)

- We can then create a table *departments* as follows

  create table *departments* **of** *Department*

- We can omit the declaration **scope** people from the type declaration and instead make an addition to the **create table** statement:

  create table *departments* **of** *Department*
  (*head* **with options scope** *people*)

Path Expressions

- Find the names and addresses of the heads of all departments:

  select *head–>name*, *head–>address*
  from *departments*

- An expression such as “head–>name” is called a **path expression**
- Path expressions help avoid explicit joins
  - If department head were not a reference, a join of *departments* with *people* would be required to get at the address
  - Makes expressing the query much easier for the user

1

Database Models and the Internet

- JDBC
- XML, Xpath, Xquery