Chapter 1: Introduction

Mrs. Sunita M Dol (Aher),
Assistant Professor,
Computer Science and Engineering Department,
Walchand Institute of Technology, Solapur, Maharashtra

Introduction

- A **Database Management System (DBMS)** is a
  - collection of interrelated data and
  - a set of programs to access those data.
- The primary **goal of a DBMS** is to provide a way to store and retrieve database information that is both convenient and efficient
- **Management of data** involves both
  - Defining structures for storage of information and
  - Providing mechanisms for the manipulation of information

Database-System Applications

- **Enterprise Information**
  - **Sales:** For customer, product, and purchase information.
  - **Accounting:** For payments, receipts, account balances, assets and other accounting information.
  - **Human resources:** For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
  - **Manufacturing:** For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
  - **Online retailers:** For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

Database-System Applications

- **Banking and Finance**
  - **Banking:** For customer information, accounts, loans, and banking transactions.
  - **Credit card transactions**: For purchases on credit cards and generation of monthly statements.
  - **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- **Universities:** For student information, course registrations, and grades
- **Airlines:** For reservations and schedule information.
- **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Purpose of Database Systems

- Before DBMS came along, organization usually stored information in the file processing system.
- **Example :** Consider part of a **savings-bank enterprise** that keeps information about all customers and savings accounts. The system has a number of application programs that manipulate the files, including
  - A program to debit or credit an account
  - A program to add a new account
  - A program to find the balance of an account
  - A program to generate monthly statements
- New application programs are added to the system as the need arises.
  - For example, suppose that the savings bank decides to offer checking accounts.
    - As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank.
    - It may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts.

Purpose of Database Systems

- This typical **file-processing system** is supported by a conventional operating system.
- **Disadvantages** of this typical file processing system are:
  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation
  - Integrity problems
  - Atomicity problems
  - Concurrent-access anomalies
  - Security problems

Purpose of Database Systems

- **Disadvantages** of this typical file processing system are:
  - **Data redundancy and inconsistency**
    - The programs may be written in several programming languages.
    - The same information may be duplicated in several places or files.
      - **For example** , the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records.
    - The redundancy of data leads to higher storage and access cost.
    - The various copies of the same data may no longer agree
      - **For example** , a changed customer address may be reflected in savings-account records but not elsewhere in the system.

Purpose of Database Systems

- **Disadvantages** of this typical file processing system are:
  - **Difficulty in accessing data**
    - The conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use and need to write a new program to carry out each new task
      - **Example** Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it.
  - **Data isolation**
    - Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Purpose of Database Systems

- **Disadvantages** of this typical file processing system are:
  - **Integrity problems**
    - The data values stored in the database must satisfy certain types of consistency constraints.
    - Hard to add new constraints or change existing ones
      - **Example -** The balance of a bank account may never fall below a prescribed amount (say, $25).
  - **Atomicity problems**
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all

Purpose of Database Systems

- **Disadvantages** of this typical file processing system are:
  - **Concurrent-access anomalies**
    - Concurrent access needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
  - **Security problems**
    - Hard to provide user access to some, but not all, data
    - Example- in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about the various bank employees.

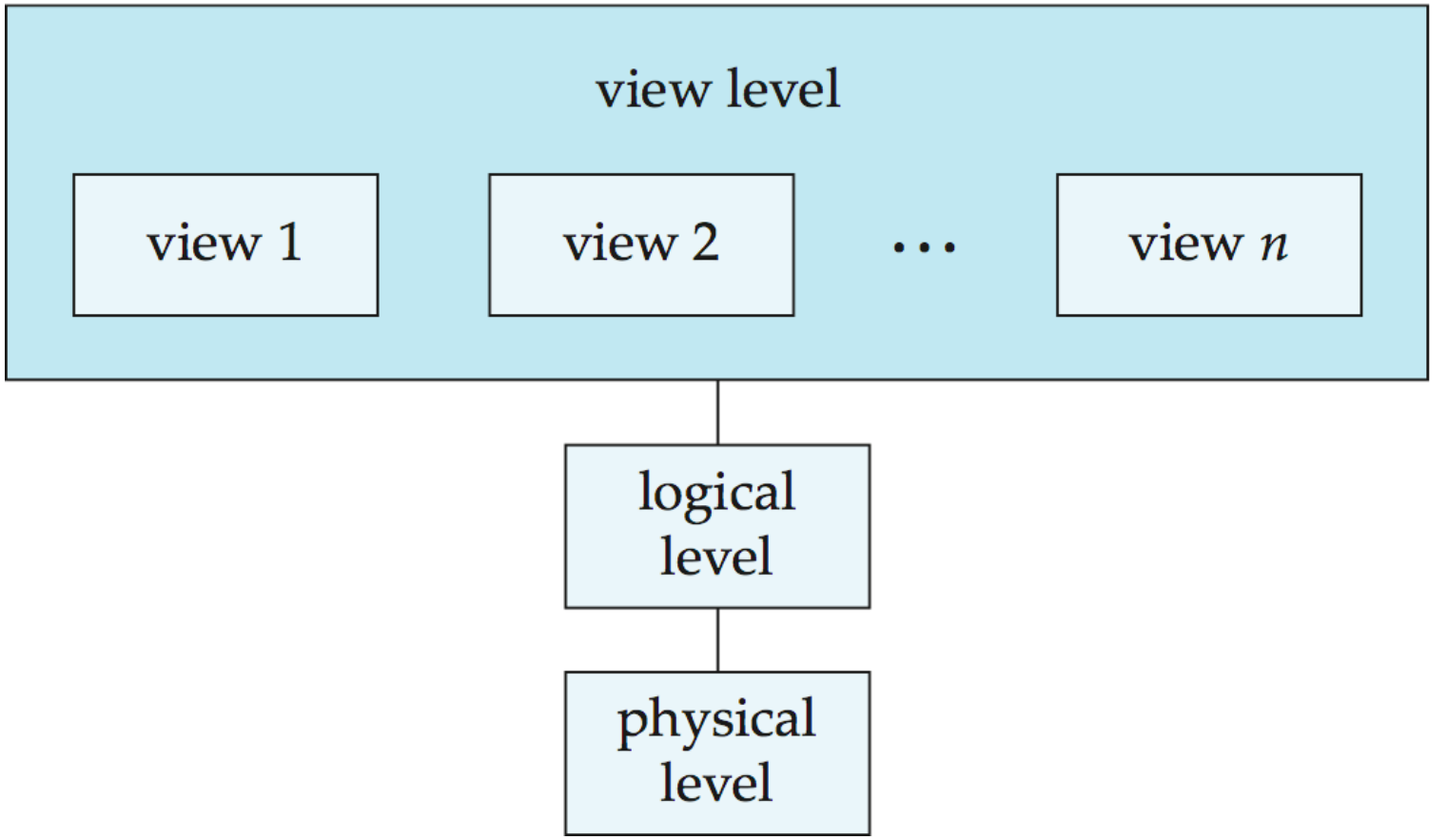
View of Data

- **Levels of Abstraction**
  - **Physical level:** The lowest level of abstraction describes how the data are actually stored.
  - **Logical level:** It describes what data are stored in the database, and what relationships exist among those data.
  - **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.
  - For example

```
type customer = record
                    customer-id : string;
                    customer-name : string;
                    customer-street : string;
                    customer-city : string;
                end;
```

View of Data

- **Levels of Abstraction**



View of Data

- **Instances and Scheme**
  - **Schema**: The overall design of the database is called the database schema. A database schema corresponds to the variable declarations in a program.
  - **Physical Schema**: The physical schema describes the database design at the physical level.
  - **Logical Schema**: The logical schema describes the database design at the logical level. It is analogous to type information of a variable in a program
    Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
  - **Instance**: The collection of information stored in the database at a particular moment is called an instance of the database. The values of the variables in a program at a point in time correspond to an instance of a database schema.

View of Data

- **Instances and Scheme**
  - **Physical Data Independence**: The ability to modify the physical schema without changing the logical schema
    - Applications depend on the logical schema
    - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Data Models

- A data model is a collection of conceptual tools for describing
  - data,
  - data relationships,
  - data semantics, and
  - consistency constraints
- Different Data Models
  - Relational Model
  - Entity-Relationship Model
  - Object-Based Data Model
  - Semistructured Data Model
  - Network data model
  - Hierarchical data model

Data Models

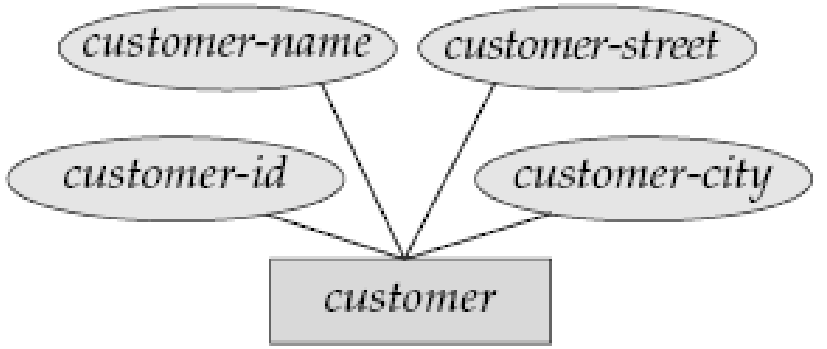
- **Different Data Models**
  - **Relational Model**
    - The relational model uses a collection of tables to represent both data and the relationships among those data.
    - Each table has multiple columns, and each column has a unique name.
    - Figure presents a sample relational database comprising one tables which show details of bank customers,

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

Data Models

- **Different Data Models**
  - **Entity-Relationship Model**
    - The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects.
    - Entities are described in a database by a set of **attributes**.
    - A **relationship** is an association among several entities.
    - The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

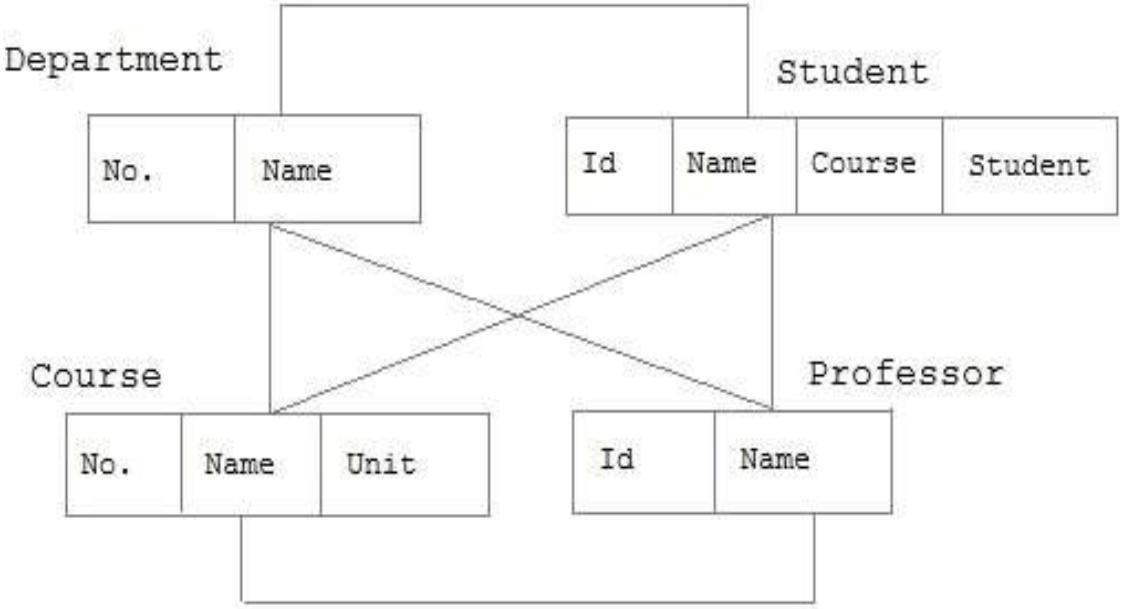


Data Models

- **Different Data Models**
  - **Object-Based Data Model**
    - Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
    - This led to the development of an object-oriented data model with notions of encapsulation, methods (functions), and object identity.
    - The object-relational data model combines features of the object-oriented data model and relational data model.
  - **Semistructured Data Model**
    - The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes.
    - The Extensible Markup Language (XML) is widely used to represent semistructured data.

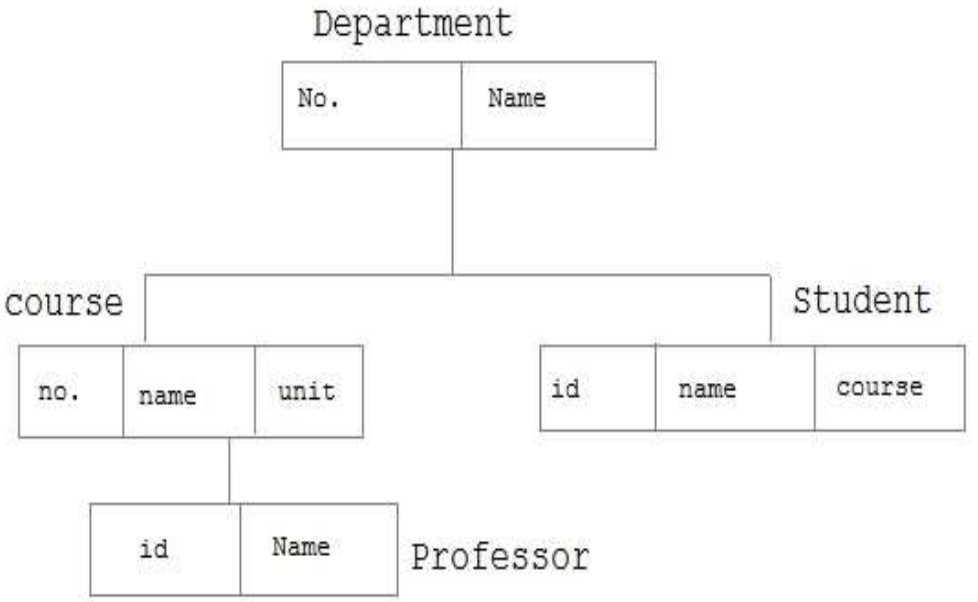
Data Models

- **Different Data Models**
  - **Network data model**
    - Entities are organised in a graph, in which some entities can be accessed through several path



Data Models

- **Different Data Models**
  - **Hierarchical data model**
    - Each entity has only one parent but can have several children .
    - At the top of hierarchy there is only one entity which is called Root.



Database Languages

- **Data-Manipulation Language:**
  - It enables users to access or manipulate data as organized by the appropriate data model.
  - The types of access are:
    - Retrieval of information stored in the database
    - Insertion of new information into the database
    - Deletion of information from the database
    - Modification of information stored in the database
  - There are basically two types:
    - **Procedural DMLs**: It require a user to specify what data are needed and how to get those data.
    - **Declarative or nonprocedural DMLs**: It require a user to specify what data are needed without specifying how to get those data.
  - A query is a statement requesting the retrieval of information.
  - The portion of a DML that involves information retrieval is called a query language.

Database Languages

- **Data-Definition Language:**
  - **S**pecification of a database schema by a set of definitions expressed by a special language is called a data-definition language (DDL).
  - Specification of the storage structure and access methods used by the database system by a set of statements in a special type of DDL is called a **data storage and definition language**.
  - The data values stored in the database must satisfy certain consistency constraints.
  - So the database systems implement integrity constraints
    - **Domain Constraints**
    - **Referential Integrity**
    - **Assertions**
    - **Authorization**

Database Languages

- **Data-Definition Language:**
  - **Integrity Constraints**
    - **Domain Constraints:** A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
    - **Referential Integrity:** There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).
    - **Assertions:** An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions.

Database Languages

- **Data-Definition Language:**
  - **Integrity Constraints**
    - **Authorization:** We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database.

      These differentiations are expressed in terms of authorization, the most common being:

      - **Read authorization**, which allows reading, but not modification, of data;
      - **Insert authorization**, which allows insertion of new data, but not modification of existing data;
      - **Update authorization**, which allows modification, but not deletion, of data; and
      - **Delete authorization**, which allows deletion of data.

      We may assign the user all, none, or a combination of these types of authorization.

Relational Database

- **Table**
  - Each table has multiple columns and each column has a unique name. Tables are also known as relations.
  - Figure presents a sample relational database comprising three tables: One show details of bank customers, the second shows accounts, and the third shows which accounts belong to which customers.

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Relational Database

- **Data Manipulation Language**
  - The SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table.
    - **Example 1:** SQL language finds the name of the customer whose customer-id is 192-83-7465:

      select *customer.customer-name*

      from *customer*

      where *customer.customer-id* = 192-83-7465

      The query specifies that those rows *from* the table *customer where* the *customer-id* is 192-83-7465 must be retrieved, and the *customer-name* attribute of these rows must be displayed. If the query were run on the table, the name Johnson would be displayed.

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

Relational Database

- **Data Manipulation Language**
  - Queries may involve information from more than one table.
    - For instance, the following query finds the balance of all accounts owned by the customer with customer_id 192-83-7465.

      select *account.balance*
      from *depositor*, *account*
      where *depositor.customer-id* = 192-83-7465 **and**
      depositor.account-number = account.account-number

      The system would find that the two accounts numbered A-101 and A-201 are owned by customer 192-83-7465 and would print out the balances of the two accounts, namely 500 and 900.

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Relational Database

- **Data Definition Language**
  - SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc.
    - **Example 1:** the following statement in the SQL language defines the *account* table:

      create table *account*
      (*account-number* **char**(10),
      balance **integer**)

      Execution of the above DDL statement creates the *account* table. In addition, it updates a special set of tables called the data dictionary or data directory. A data dictionary contains metadata—that is, data about data.

Relational Database

- **Database Access from Application Programs**
  - There are two ways to do this:
    - By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database and retrieve the results.
      - The Open Database Connectivity (**ODBC**) standard for use with the C language
      - The Java Database Connectivity (**JDBC**) standard for the Java language.
    - By extending the host language syntax to embed DML calls within the host language program. The DML precompiler, converts the DML statements to normal procedure calls in the host language.

Database Design

- **Design Process**
  - The initial phase of database design is to characterize fully the data needs of the prospective database users. The outcome of this phase is a specification of user requirements.
  - Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this **conceptual-design** phase provides a detailed overview of the enterprise.

Database Design

- **Design Process**
  - A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data.
  - The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.
    - **Logical Design** – Deciding on the database schema. Database design requires that we find a "good" collection of relation schemas.
      - Business decision – What attributes should we record in the database?
      - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
    - **Physical Design** – Deciding on the physical layout of the database

Database Design

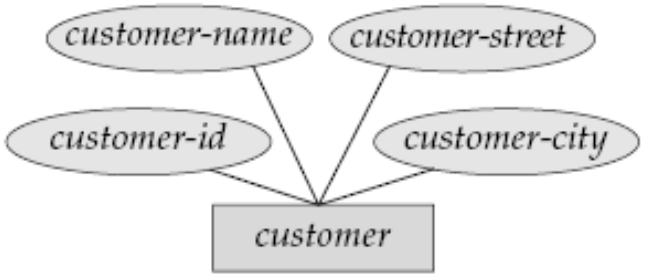
- **Database Design for Banking Enterprise**
  - The bank is organized into branches. Each branch is located in particular city and is identified by a unique name. The bank monitors the assets of each branch.
  - Bank customers are identified by their customer_id values. The bank store each customer's name and street and city where the customer lives. Customers may have account and can take out the loans. A customer may be associated with a particular banker who may act as a loan officer or personal banker for that customer.
  - Bank offers two types of account- saving and checking account. Accounts can be held by more than one customer and a customer can have more than one account. Each account is assigned unique account number. The bank maintains a record of each account' balance and most recent date on which the account was accessed each customer holding the account. Each saving account has an interest rate and overdrafts are recorded for each checking account.

Database Design

- **Database Design for Banking Enterprise**
  - The bank provides its customer with loans. A loan is originates at particular branch and can be held by one or more customers. A loan is identified by a unique loan number.
  - Bank employees are identified by their employee_id values. The bank administration stores the name and the telephone number of each employee, the name of employee's dependents and employee_id number of employee's manager. The bank also keep track of the employees' start date and thus the length of employment.

Database Design

- **Entity-Relationship Model**
  - The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects.
  - An entity is a “thing” or “object” in the real world that is distinguishable from other objects
  - Entities are described in a database by a set of **attributes**.
  - A **relationship** is an association among several entities.
  - The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.



Data Models

- **Entity-Relationship Model**
  - The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components
    - **Rectangles**, which represent entity sets
    - **Ellipses**, which represent attributes
    - **Diamonds**, which represent relationships among entity sets
    - **Lines**, which link attributes to entity sets and entity sets to relationships

Data Models

- **Entity-Relationship Model**

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

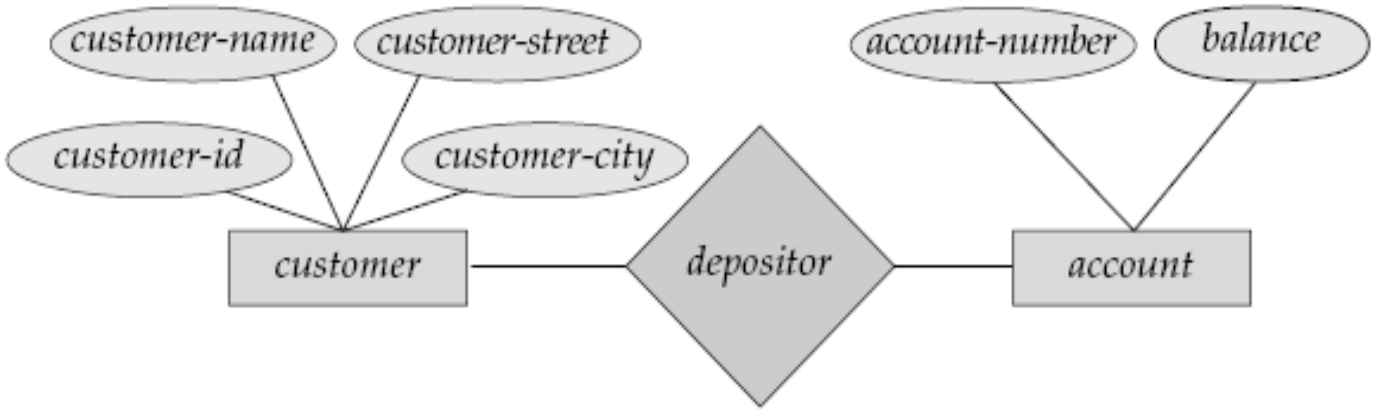
(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table



Database Design

- **Normalization**
  - The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy
  - Formalize what designs are bad, and test for them
  - The approach is to design the schemas that are in appropriate normal form.
  - To understand the need for normalization, consider the undesirable properties that a bad design may have are:
    - Repetition of information
    - Inability to represent certain information

Database Design

- **Normalization**
  - Consider the banking example with the following three tables: One show details of bank customers, the second shows accounts, and the third shows which accounts belong to which customers.

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Database Design

- **Normalization**
  - Suppose that instead of having two separate tables *account* and *depositor*, we have single table *depositor1* as shown below that combines the information from two tables *account* and *depositor.*
  - There are two rows in the depositor that contains the information about account A-201. The repetition of information in alternative design is undesirable.
  - Repeating information wastes space.
  - Suppose that we wish to change the account balance of A-201 from 700 to 750. This change must be reflected in two rows.

customer-id	*account-number*	*balance*
192-83-7465	A-101	500
192-83-7465	A-201	700
019-28-3746	A-215	400
677-89-9011	A-102	350
182-73-6091	A-305	900
321-12-3123	A-217	750
336-66-9999	A-222	700
019-28-3746	A-201	700

Database Design

- **Normalization**
  - Suppose that instead of having two separate tables *customer* and *depositor,* we have single table *customer1* as shown below that combines the information from two tables *customer* and *depositor.*
  - We cannot represent directly the information concerning the customer unless that customer has at least one account at the bank.
  - This is because rows in the customer table require the value for *account_number.* On solution to this problem is to introduce null values but the null values indicate that the value does not exists or is not known.

customer-id	*customer-name*	*customer-street*	*customer-city*	*account-number*	*balance*
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101	500
019-28-3746	Smith	4 North St.	Rye	A-215	700
677-89-9011	Hayes	3 Main St.	Harrison	A-102	400
182-73-6091	Turner	123 Putnam Ave.	Stamford	A-305	350
321-12-3123	Jones	100 Main St.	Harrison	A-217	750
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222	700
019-28-3746	Smith	72 North St.	Rye	A-201	900

Data Storage and Querying

- **Storage Manager**
  - The *storage manager* is the component of a database system 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 for
    - the interaction with the file manager.
    - for storing, retrieving, and updating data in the database.

Data Storage and Querying

- **Storage Manager**
  - The storage manager components include:
    - **Authorization and integrity manager**: tests for the satisfaction of integrity constraints and checks the authority of users to access data.
    - **Transaction manager**: ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
    - **File manager:** manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
    - **Buffer manager**: is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.

Data Storage and Querying

- **Storage Manager**
  - The storage manager implements several data structures as part of the physical system implementation:
    - **Data files**: store the database itself.
    - **Data dictionary:** stores metadata about the structure of the database, in particular the schema of the database.
    - **Indices:** can provide fast access to data items. For example, we could use an index to find the *instructor* record with a particular *ID*, or all *instructor* records with a particular *name*.
    - **Hashing:** is an alternative to indexing that is faster in some but not all cases.

Data Storage and Querying

- **The Query Processor**
  - The query processor components include:
    - **DDL interpreter**: interprets DDL statements and records the definitions in the data dictionary.
    - **DML compiler**: translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. The DML compiler performs query optimization.
    - **Query evaluation engine**: executes low-level instructions generated by the DML compiler.

Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- Consider the one account say A is debited and another account say B is credited.
- The all-or-none requirement is called **atomicity** e.g. both the credit and debit occur, or that neither occurs.
- In addition, it is essential that the execution of the funds transfer preserve the consistency of the database. That is, the value of the sum of the balances of *A* and *B* must be preserved. This correctness requirement is called **consistency**.

Transaction Management

- After the successful execution of a funds transfer, the new values of the balances of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called **durability**.
- A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction a unit of both atomicity and consistency.
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

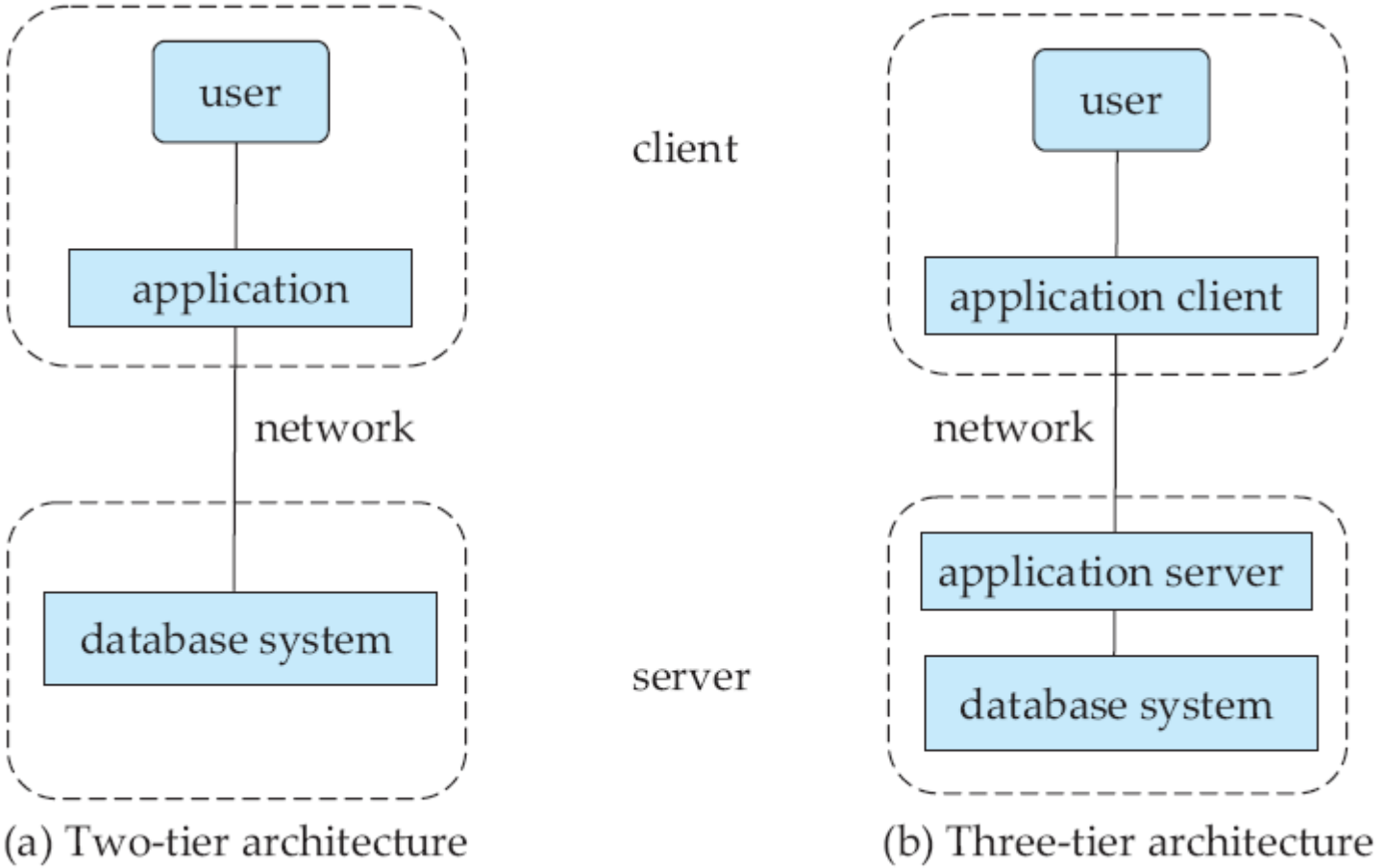
Database Architecture

- The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs.
- Database systems can be
  - centralized,
  - client-server, where one server machine executes work on behalf of multiple client machines.
  - parallel computer architectures.
  - Distributed databases which span multiple geographically separated machines

Database Architecture

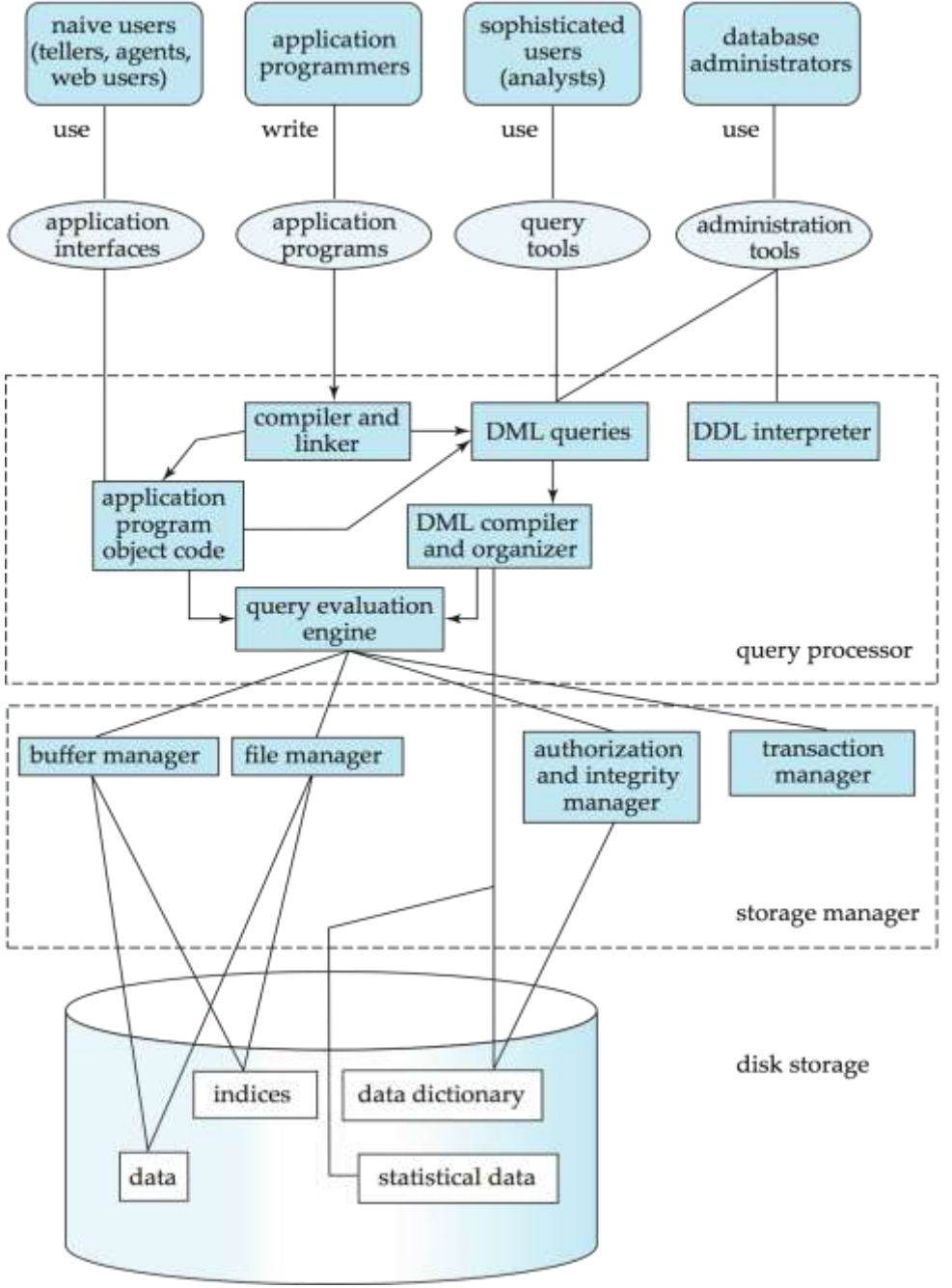
- **In a two-tier architecture,**
  - the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements.
  - Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.
- **In a three-tier architecture,**
  - the client machine acts as merely a front end and does not contain any direct database calls.
  - Instead, the client end communicates with an application server, usually through a forms interface.
  - The application server in turn communicates with a database system to access data.
  - The business logic of the application is embedded in the application server, instead of being distributed across multiple clients.

Database Architecture



(a) Two-tier architecture

(b) Three-tier architecture


naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

Database Users and Administrators

- People who work with a database can be categorized as database users or database administrators.
- Database Users and User Interfaces
  - **Naïve users**
  - **Application programmers**
  - **Sophisticated users**
  - **Specialized users**

Database Users and Administrators

- **Database Users and User Interfaces**
  - **Naïve users** are unsophisticated users who interact with the system by invoking one of the application programs.
    - For example, a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred and the account to which the money is to be transferred
  - **Application programmers** are computer professionals who write application programs. They can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Database Users and Administrators

- **Database Users and User Interfaces**
  - **Sophisticated users** interact with the system without writing programs. they form their requests using a database query language. They submit each such query to query processor whose function is to break down DML statements into statements that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.
  - **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Users and Administrators

- **Database Administrator**
  - A person who has central control of both data and program to access the data over the system is called a database administrator
  - The functions of a DBA include:
    - **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
    - **Storage structure and access-method definition**
    - **Schema and physical-organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Database Users and Administrators

- **Database Administrator**
  - The functions of a DBA include:
    - **Granting of authorization for data access**: By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
    - **Routine maintenance:** Examples of the database administrator's routine maintenance activities are:
      - Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
      - Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
      - Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Data Mining and Information Retrieval

- **Data mining (knowledge discovery from data)**
  - Extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) patterns or knowledge from huge amount of data.
  - Process of analyzing data from different perspectives and summarizing it into useful information .
  - Process of discovering interesting knowledge from large amount of data stored in database, data warehouse or other information repositories.
  - **Alternative names**
    - Knowledge discovery (mining) in databases (KDD), knowledge extraction, data/pattern analysis, data archeology, data dredging, information harvesting, business intelligence, etc.
  - **Data Mining Application:** Banking, Telecommunication industry, Retail Industry, DNA analysis etc.

Specialty Databases

- **Object-Based Data Models**
  - An object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.
  - Inheritance, object identity, and encapsulation (information hiding), with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modelling.
  - The object-oriented data model also supports a rich type system, including structured and collection types.
  - The major database vendors presently support the object-relational data model, a data model that combines features of the object-oriented data model and relational data model.

Specialty Databases

- **Semistructured Data Models**
  - Semistructured data models permit the specification of data where individual data items of the same type may have different sets of attributes
  - The XML language was initially designed as a way of adding markup information to text documents
  - XML provides a way to represent data that have nested structure, and furthermore allows a great deal of flexibility in structuring of data, which is important for certain kinds of non-traditional data.

References

- Database system concepts by Abraham Silberschatz, Henry F. Korth, S. Sudarshan (McGraw Hill International Edition) sixth edition.
- 2. Database system concepts by Abraham Silberschatz, Henry F. Korth, S. Sudarshan (McGraw Hill International Edition) fifth edition.
- http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/
- http://codex.cs.yale.edu/avi/db-book/db5/slide-dir/
- http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/