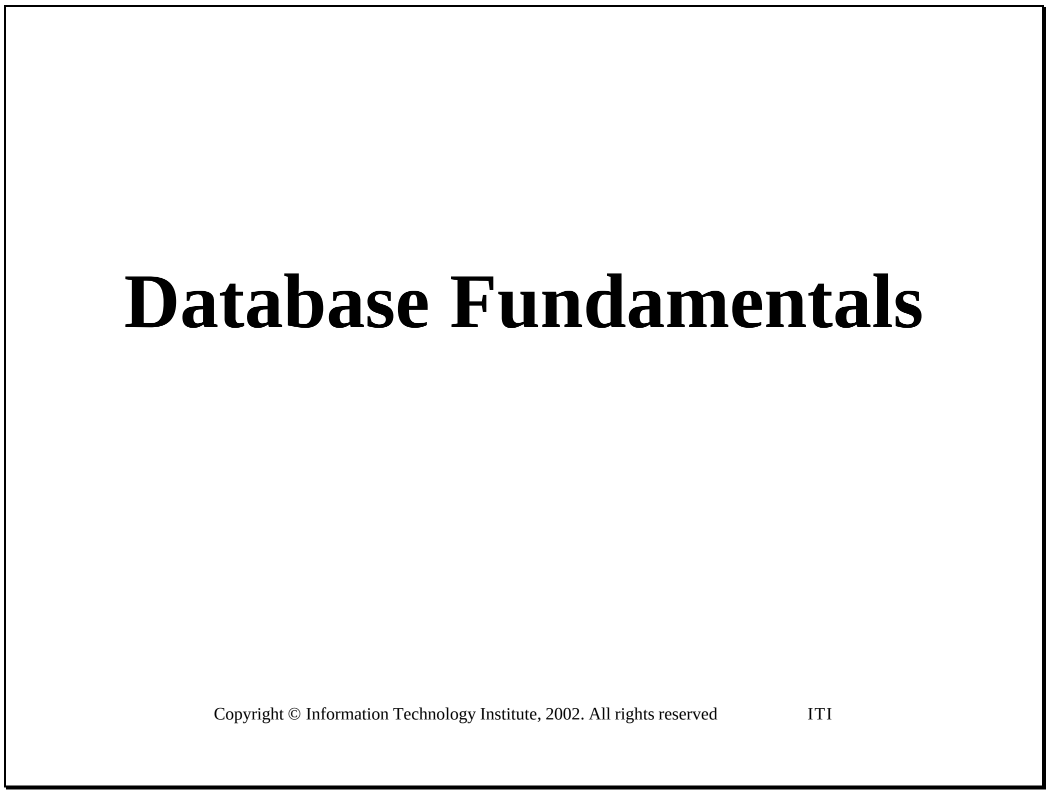
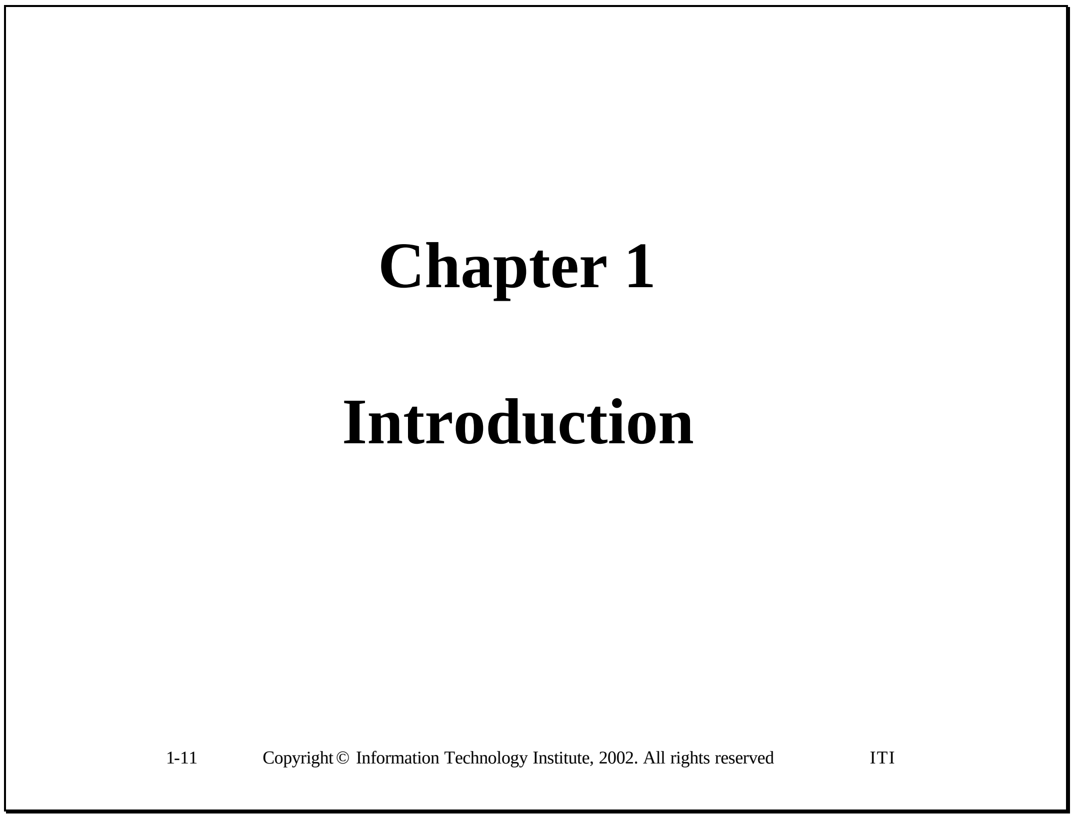


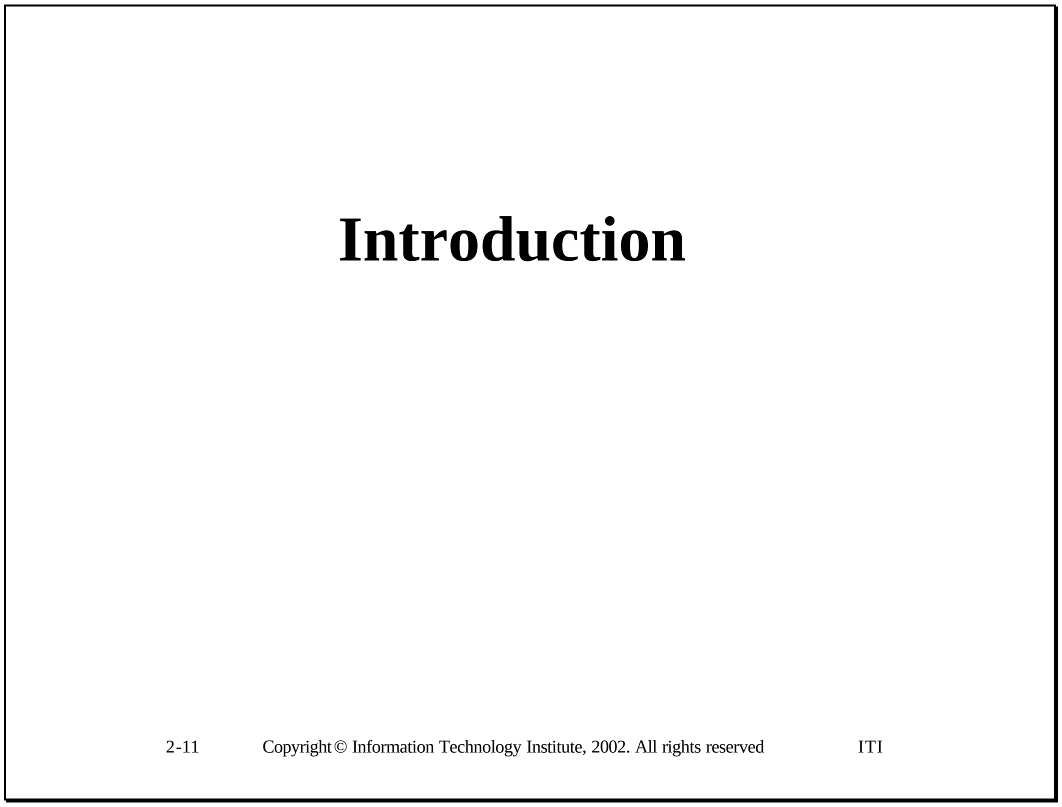
# Database Fundamentals



# Chapter 1

# Introduction

1-11 ITI



## Introduction

The following material is a general introduction to the database fundamentals.
Our world is based on a combination of different business processes and the relationships between them. Database design is an art which is used to achieve many objectives for any organization.

**For example**

Automating and computerizing the manual paperwork could be an objective while fulfilling management requests efficiently and quickly could be another kind of objectives. To clarify more. We shall consider the following question taking restaurants as our business case.

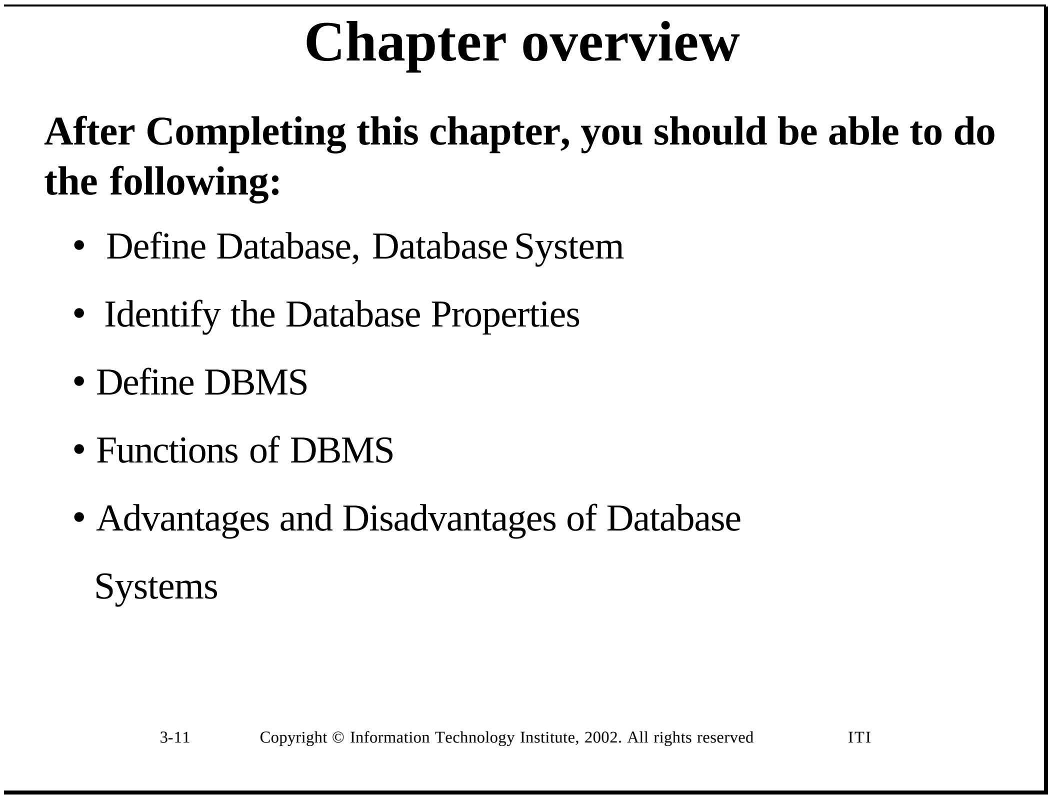
Question: How could you know if the customer prefers beans or mashed potatoes with his fried chicken?

Answering any ad hoc question requested by managers is another kind of objectives.
For example; how could you know if your customers prefer BEANS or mashed potatoes with their fried chicken?

Creating a "real life" information system for such an example requires taking into consideration different dependencies between people, companies, business events, boss demands and even psychology.

Such dependencies are known as BUSINESS RULES in database.

In order to be able to implement such systems and satisfy the business requirements, you need to enter the world of the database, and know how to design a database, and then how to access this base based on predetermined questions or even a run-time question.

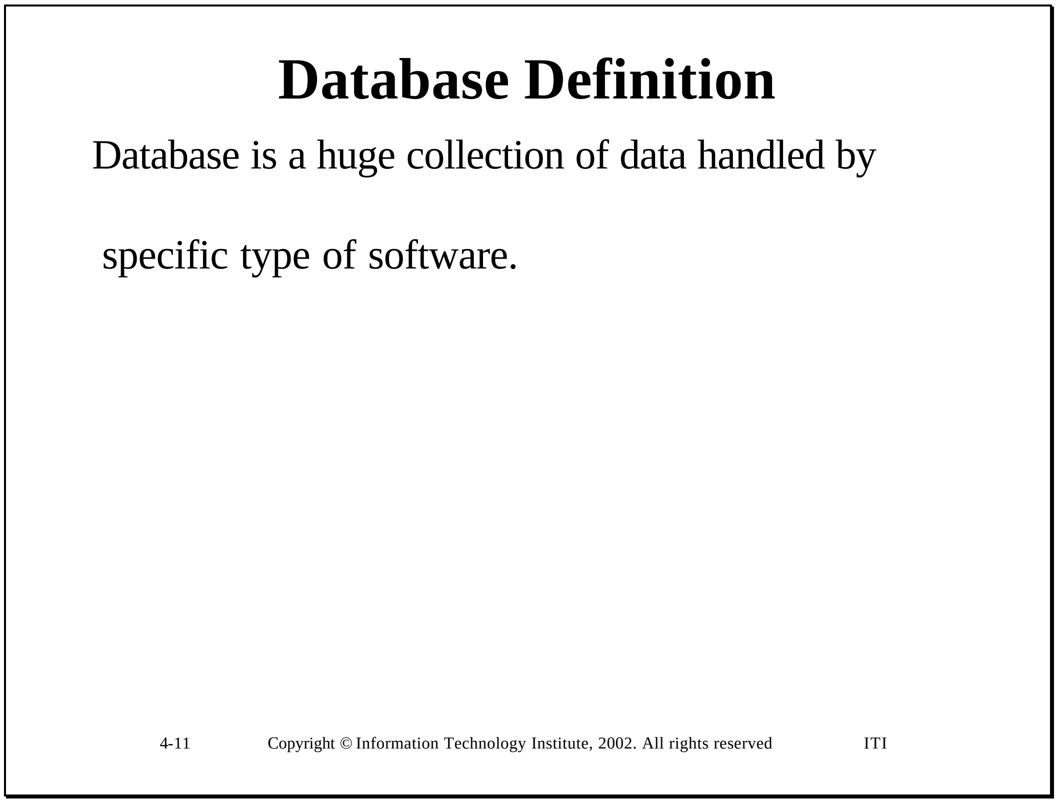


## Chapter overview

**After Completing this chapter, you should be able to do the following:**

- Define Database, Database System
- Identify the Database Properties
- Define DBMS
- Functions of DBMS
- Advantages and Disadvantages of Database Systems

This lesson describes the basic concepts necessary for a good understanding of database.



## Database Definition

Database is a huge collection of data handled by

specific type of software.

4-11 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Define the Database

The term database is a composition of "data" and "base". "Data" are facts of the real world, which are supplied by a "base".

The word "data base" was first used in the early 1960's. It was later written as "data-base" and recently as "database".

File is another term that is used to mean a collection of facts of the real world. However, a file is an accumulation of data with the same structure. On the other hand, a database is an integration of different kinds of data.

A database is a logical collection of **non-redundant**, **shareable** data that is used by different application systems of some given **enterprise.** This data represents some aspect of the real world.

**Redundant** data is the duplicated data that adds no further information. Eliminating such data is the simplest way to avoid inconsistency.

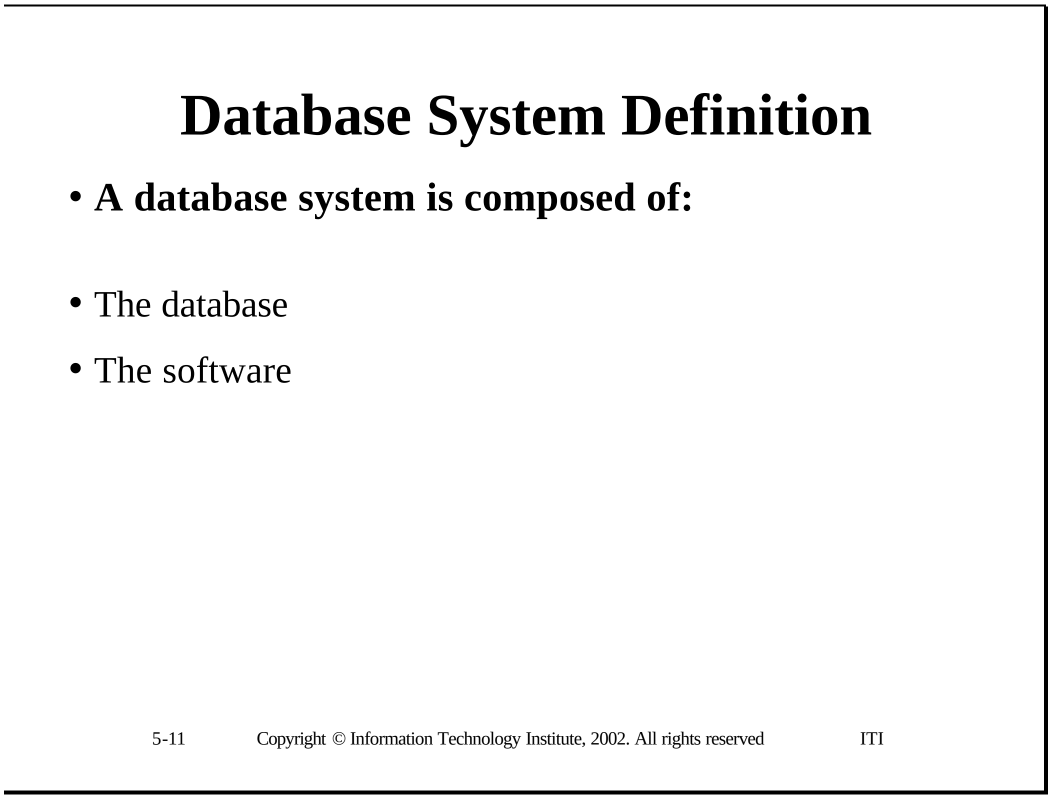
**Shareable** data is the data that can be accessed by several applications.

An **enterprise** might be a single individual – with a small private database – or a complete corporation – with large shared data – or anything in between, for example:

- Banking System
- Hospital Information System

- Library System

This concept of databases is highly related to computers development. Currently, computers are being considered as information processing machines rather than machines that calculate. As a result, the demand to use computers to store increasing data of the real world and utilize it has increased with ease. For Database management systems (DBMS) were used for multi purpose use rather than file systems because they are more powerful. A definition of database systems and its functions are explained in the following.



## Database System Definition

- **A database system is composed of:**
- The database
- The software

5-11 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### What is a database system?

A database system is a combination of the database and the software together.

The user of the system has the ability to perform a variety of operations on the database files, including:

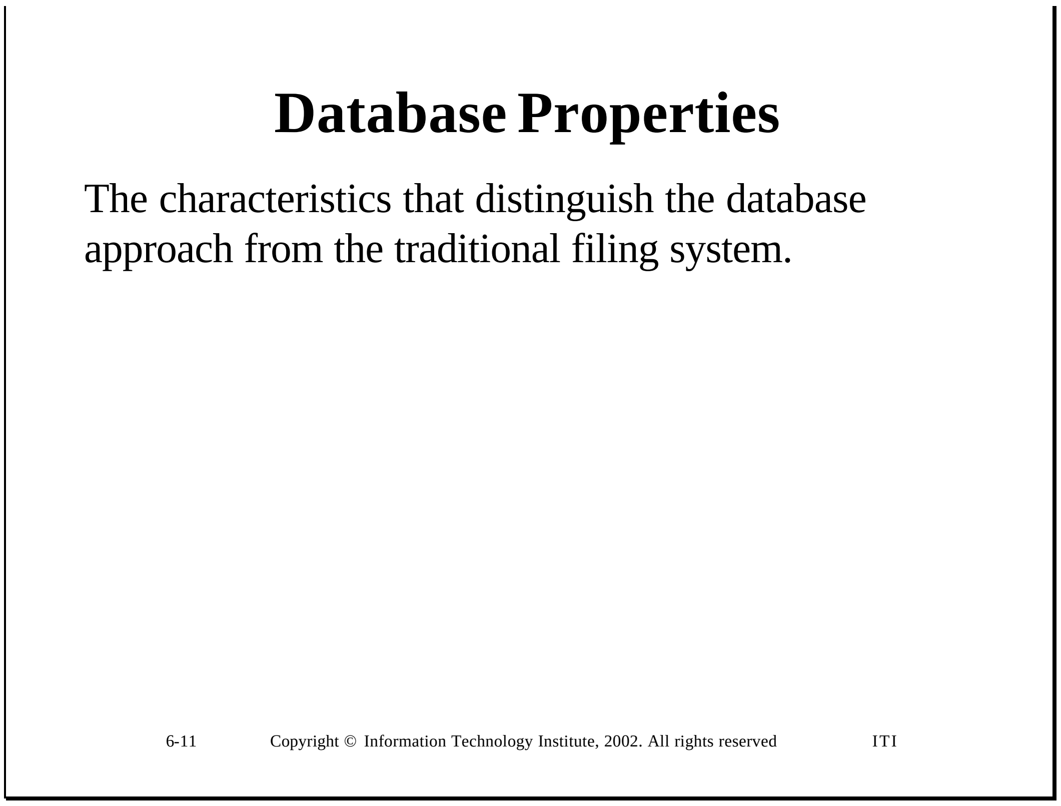
- Adding new data into existing files.
- Retrieving data from existing files.
- Deleting data from existing files.
- Editing of existing data.

So, the overall purpose of database system is to:

- Store enormous amounts of data efficiently.
- Multi-purpose utilization of data.
- Effective and easy utilization of data.
- Maintaining the information easily.

**Note:**

Some books distinguish between data and information, using data to refer to the values actually stored in the database, and using information to refer to the meaning of those values as understood.



## Database Properties

The characteristics that distinguish the database approach from the traditional filing system.

6-11 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Database Properties

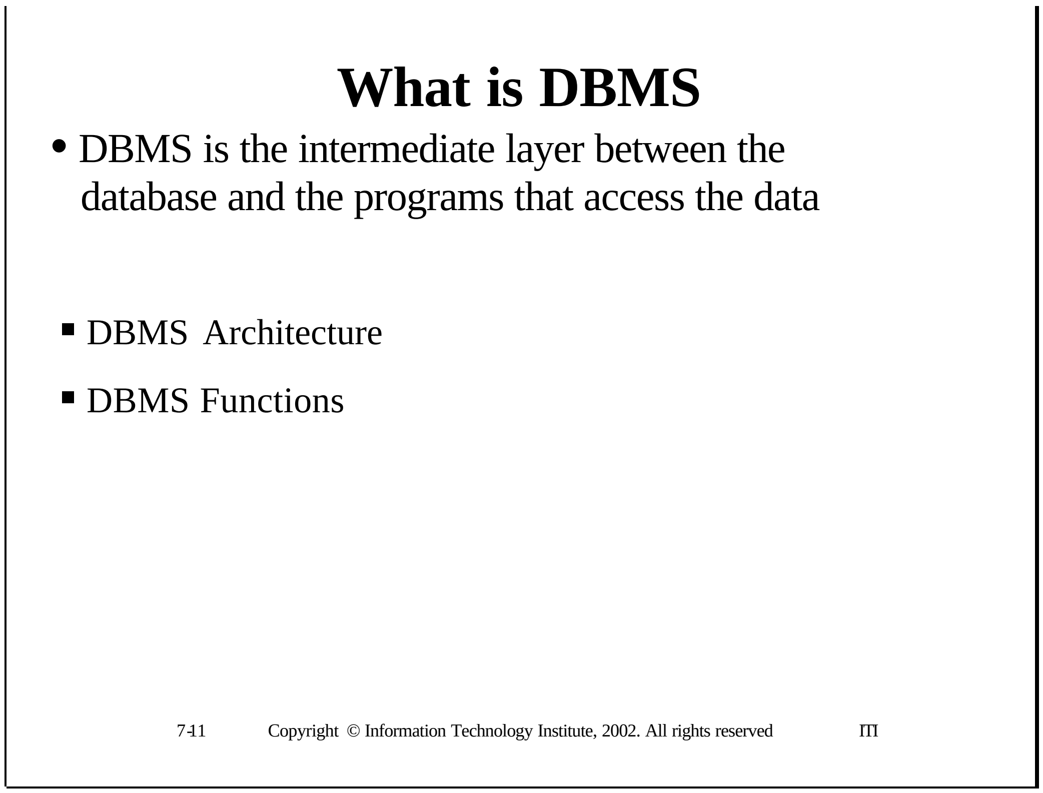
A database has the following properties:

- Redundancy can be reduced.
- It contains a logically coherent collection of data with some inherent meaning.
- It allows sharing the data by concurrent users.
- Security restrictions can be applied.
- Data integrity can be maintained.

A database can be of any size and of varying complexity. For example, a list of names and addresses of students in a certain year may consist only of a few hundred records, each with a simple structure. On the other hand, the card catalogue of a large library may contain half a million cards stored under different categories; by primary author's last name, by subject, by book title, with each category organized in alphabetic order.

A database also provides insulation between programs and data. In traditional file processing, the structure of data files is embedded in the program; so any changes in the structure of the file may affect the program itself.

DBMS access programs, on the other hand, do not require such changes, as the data structure is already saved in the database catalog.



## What is DBMS

- DBMS is the intermediate layer between the database and the programs that access the data

  - DBMS Architecture
  - DBMS Functions

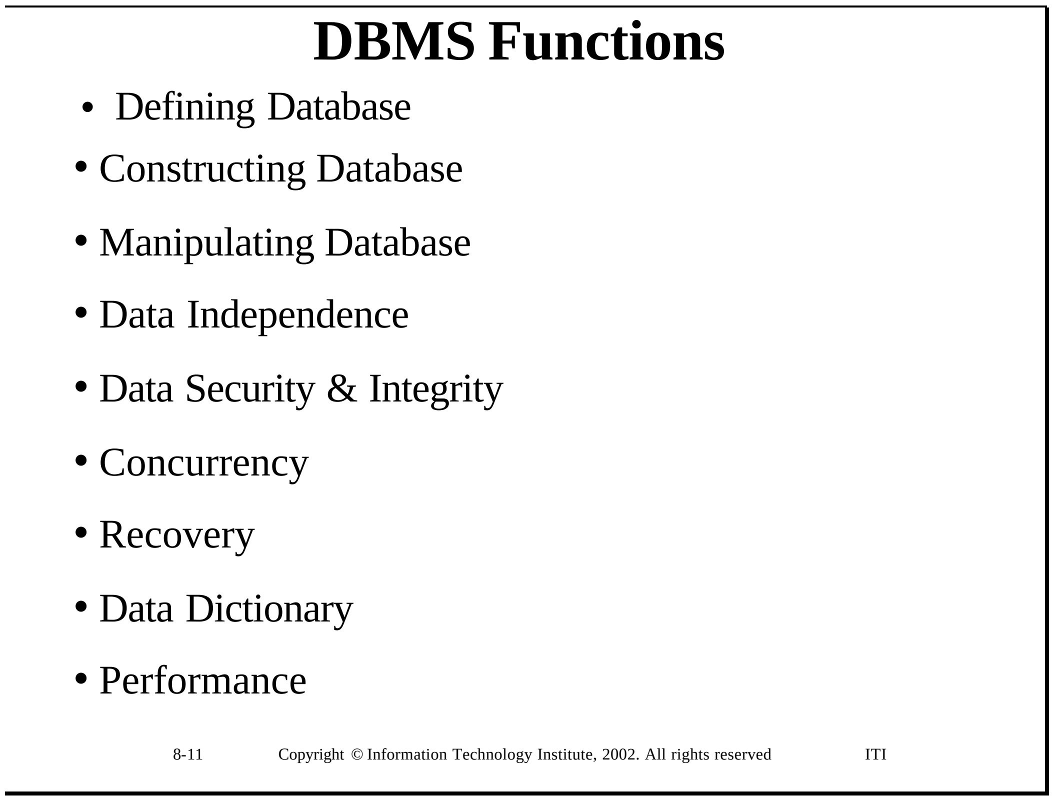
### DBMS Architecture

Database management system is a layer of software between the physical database itself and the users of the system.

DBMS enables users to create and maintain a database, and handles all requests from users to access the database, such as:

- Adding or removing tables.
- Retrieving and updating data.

The DBMS is a general-purpose software system that facilitates defining, constructing, and manipulating databases for various applications. Examples of DBMSs are Microsoft SQL Server, Oracle…etc.



## DBMS Functions

- Defining Database
- Constructing Database
- Manipulating Database
- Data Independence
- Data Security & Integrity
- Concurrency
- Recovery
- Data Dictionary
- Performance

### DBMS Functions

**Defining the database** involves specifying the data types, structures, and constraints for the data to be stored in the database.

**Constructing the database** is the process of storing the data it self on some storage medium that is controlled by the DBMS.

**Manipulating a database** includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini-world, and generating reports from the data.

**Data Independence** is used to reduce side effects on programs when data changes are not only in contents but also in structures.

**Data Security and Integrity** is achieved through monitoring user requests and reject any attempts to violate the security and integrity checks defined by the database administrator.

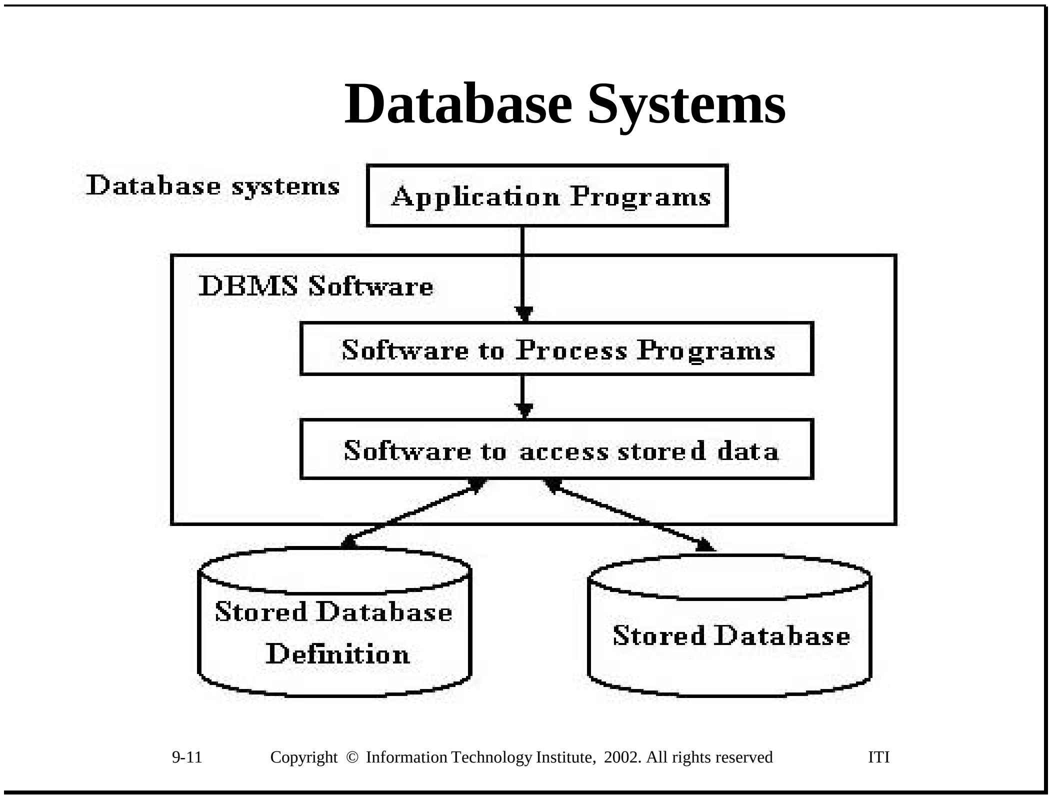
**Concurrency** keeps consistency of data when updated by multiple users or user programs at the same time.

**Recovery** is needed in case of database destruction as a result of hardware failures, software and human errors or accidents.

**Data Dictionary** is considered as a system database. The content of the data dictionary can be regarded as data about data (that is called metadata).

**Performance** DBMS should provide all of the functions identified above as efficiently as possible

**User Language Interface** Application programs need some means of interfacing with the database. This is achieved by adding commands designed for database manipulation to the procedural high-level language. In this type of interface the high level language is called a host language and the additional database commands are referred to as a data manipulation language. A DBMS may allow access to the same database via different host languages.



## Database Systems

Database systems
Application Programs
DBMS Software
Software to Process Programs
Software to access stored data
Stored Database Definition
Stored Database

9-11 Copyright © Information Technology Institute, 2002. All rights reserved ITI

The graph shows the role of DBMS as an intermediate layer between the application program side and the database side.

**Example:**

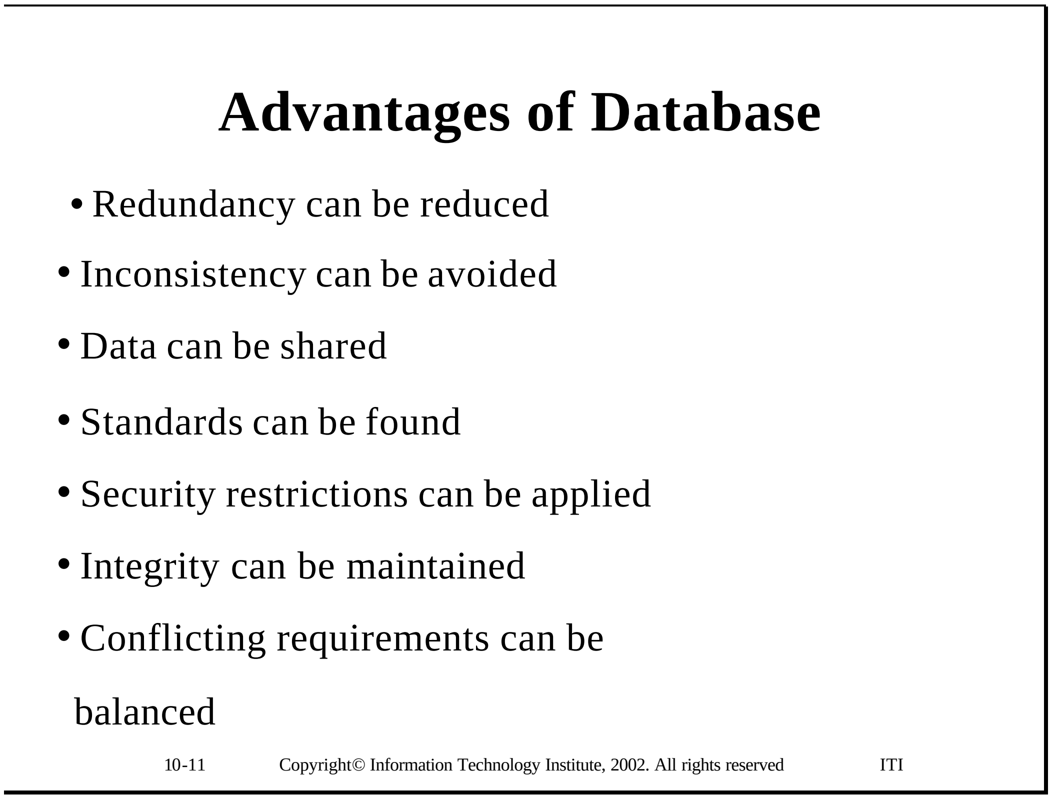
The user types the following command

SELECT Customer-no, Customer-name FROM CUSTOMER;
He wants to retrieve the customer number "Customer-no" and the Customer name "Customer-name" from "CUSTOMER" table.

The following events take place:

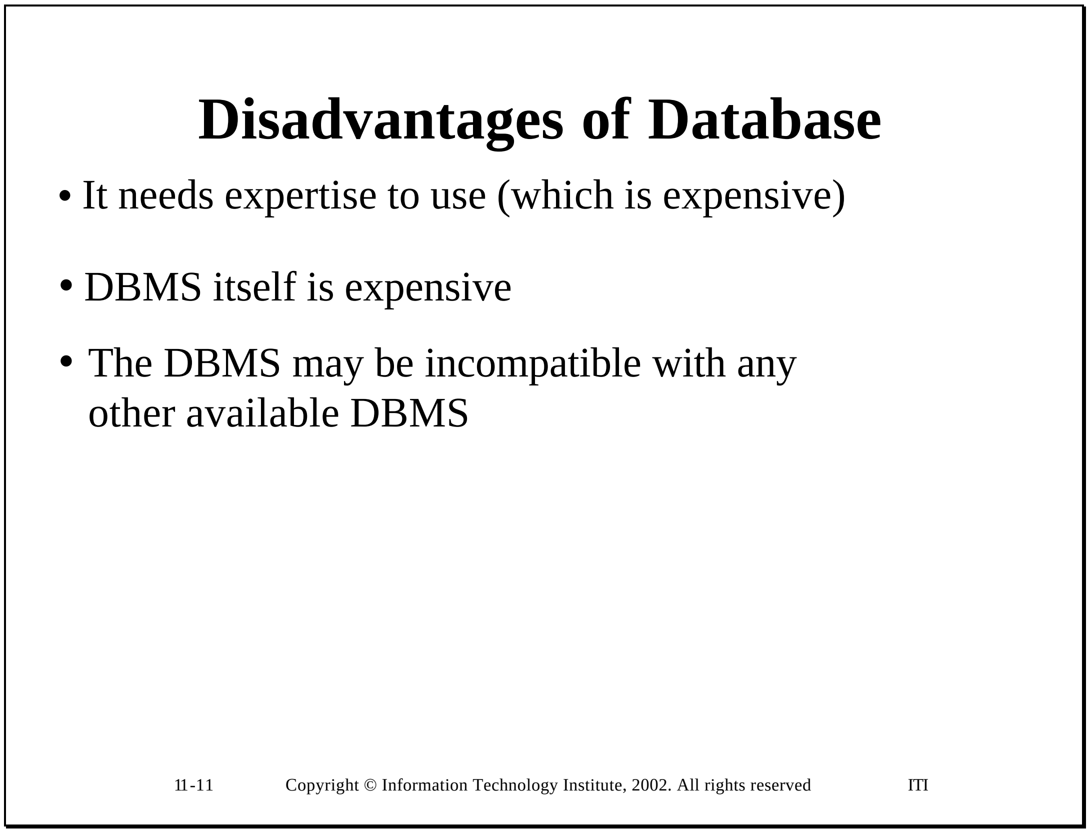
- The user types the instruction
- DBMS intercepts the request and analyzes it
- DBMS recognizes that the query accesses the table "CUSTOMER" and needs only two fields from it
- DBMS determines from the data dictionary the physical database descriptions and the disk files containing the necessary records
- DBMS transfers control to the operating system for reading the records
  Operating system retrieves the necessary data from the disk files, transfers them to the system buffers, and transfers control to the DBMS
- DBMS transfers data from system buffers to the user's work area
- The DBMS displays the customer list on the user's terminal

A sequence of events similar to the ones above takes place when the user wants to update the database instead of retrieving data. The command entered by the user first reads the designated records, then modifies the data in the user's work area, and finally, issues an instruction to the DBMS to write the modified data back into the database. The DBMS then issues the appropriate WRITE command to the operating system for implementing the modified function.



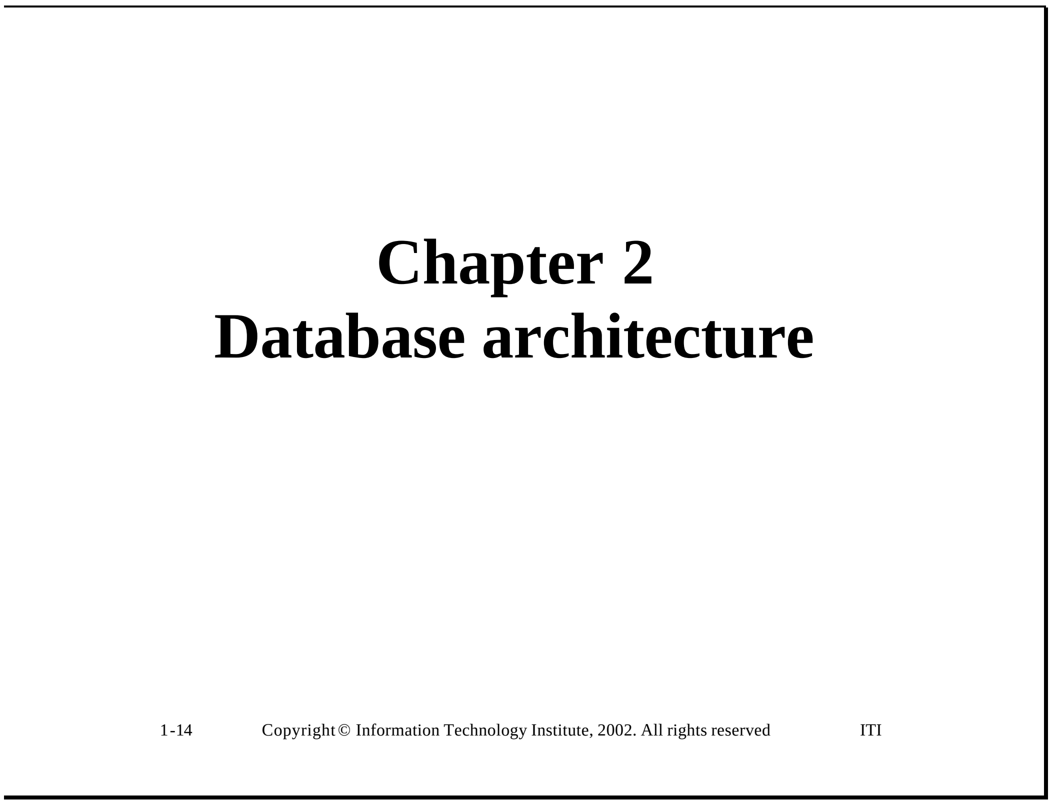
## Advantages of Database

- Redundancy can be reduced
- Inconsistency can be avoided
- Data can be shared
- Standards can be found
- Security restrictions can be applied
- Integrity can be maintained
- Conflicting requirements can be balanced



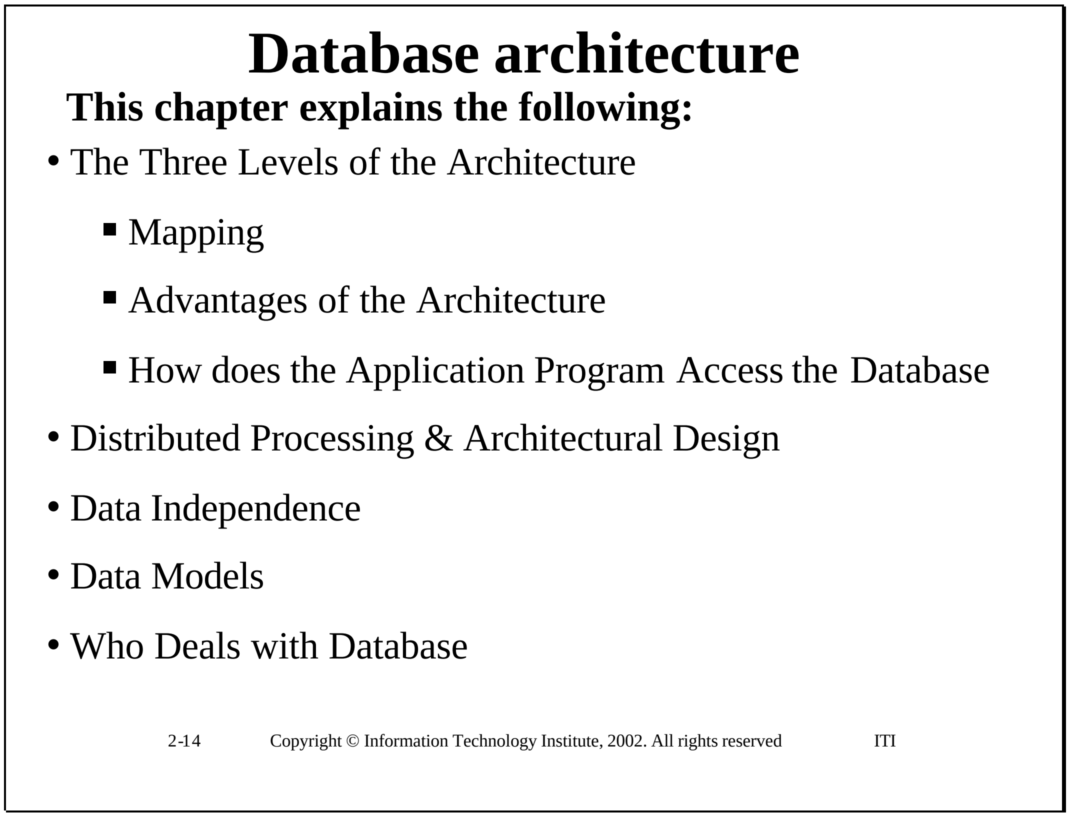
## Disadvantages of Database

- It needs expertise to use (which is expensive)
- DBMS itself is expensive
- The DBMS may be incompatible with any other available DBMS



# Chapter 2
# Database architecture

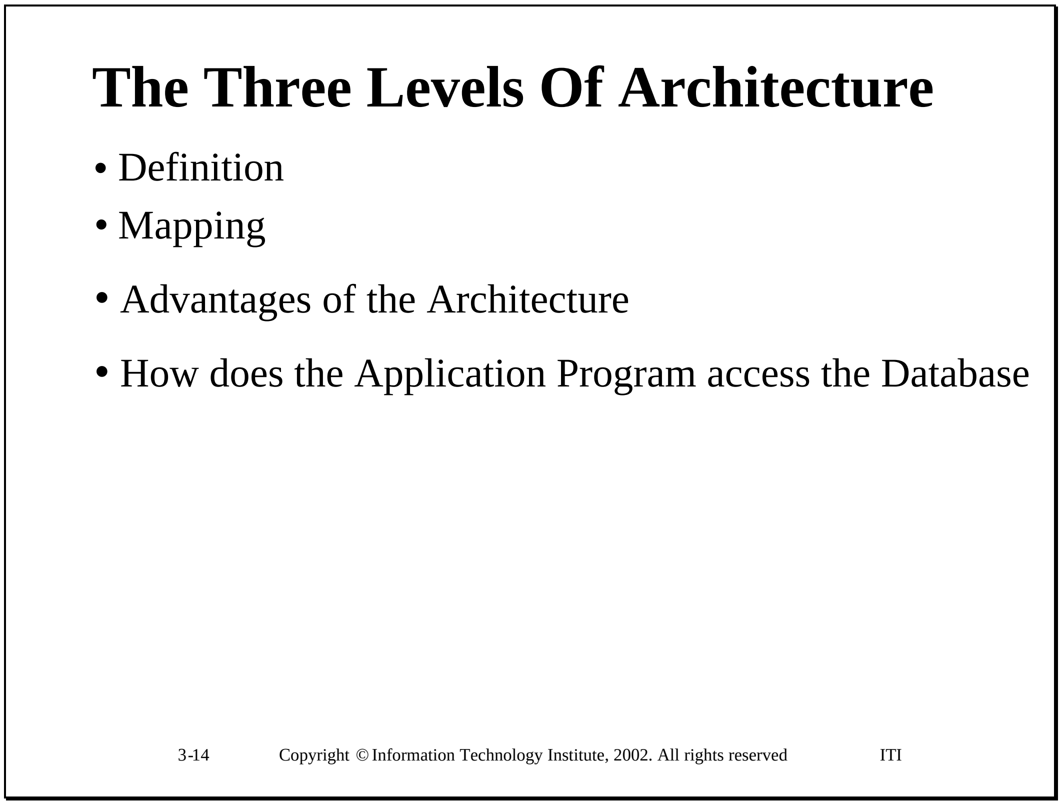
1-14 ITI



## Database architecture

**This chapter explains the following:**

- The Three Levels of the Architecture
  - Mapping
  - Advantages of the Architecture
  - How does the Application Program Access the Database
- Distributed Processing & Architectural Design
- Data Independence
- Data Models
- Who Deals with Database



## The Three Levels Of Architecture

- Definition
- Mapping
- Advantages of the Architecture
- How does the Application Program access the Database

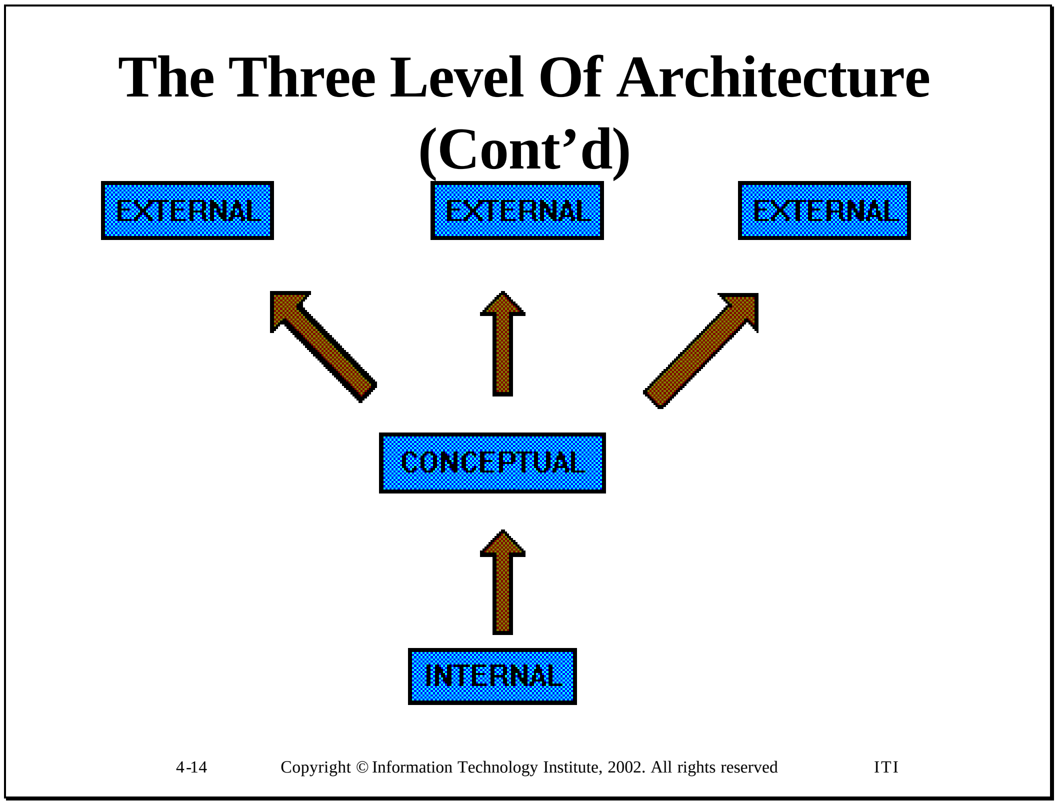
Presenting the database architecture will help provide a framework on which general database concepts can be described and the structure of a specific database system can be explained.

The main aim of this architecture is to separate the user applications from the physical database.

### The Three Levels of the Architecture

It is a DBMS architecture needed to provide a framework for defining database structures. This model has been proposed by ANSI/SPARC. Not all DBMS(s) fit neatly into this architecture, but most do. Mappings exist between the following three levels:

| | |
|---|---|
| **External level** | (individual users view) |
| **Conceptual level** | (community user view) |
| **Internal level** | (storage) |



## The Three Level Of Architecture (Cont'd)

4-14 Copyright © Information Technology Institute, 2002. All rights reserved ITI

- **External Level**

  This level includes a number of external schemas or user views. Each view or schema presents the part of the database that a particular user is interested in and hides the rest of the database.

At this level each user (application programmer or terminal user) will deploy his preferred language which will be used as "data sublanguage".

For instance, for the application programmer, the language could be COBOL or Visual Basic, while for the end user, the language will be either a query language, some kind of forms, or menu-driven, tailored to that user's requirements.

In this level, the external schema describes the local views of the database required by an application program, if several programs require identical local views they may share the same external schema.
Properties of the data such as the format of data items or the sequence in which data is seen may be specified by the external schema, but it can not override any of the constraints imposed by the conceptual schema.

- **Conceptual Level**

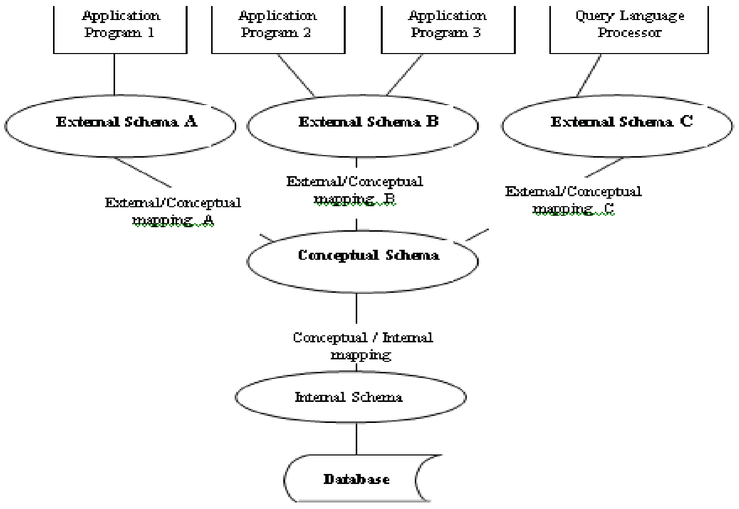
  The conceptual level is essentially a representation of the entire information content of the database in a form abstracted from physical storage. It may be quite different or similar to external views held by a particular user. It is data as it really is, rather than as users are forced to see it - it is multiple occurrences of multiple types of conceptual records.

  The conceptual level is a description of all the data which is to be stored in the database, including the constraints which are applied to the data.

- **Internal Level**

  This is a low level representation of the entire database. It defines the various types of stored records, indexes existing, methods of representing stored fields, physical sequences of the stored records, storage media, and so on.
  Programs accessing this level directly (i.e. utility programs) are risky since they bypass the security and integrity checks that the DBMS program normally imposes.

## Mapping Process



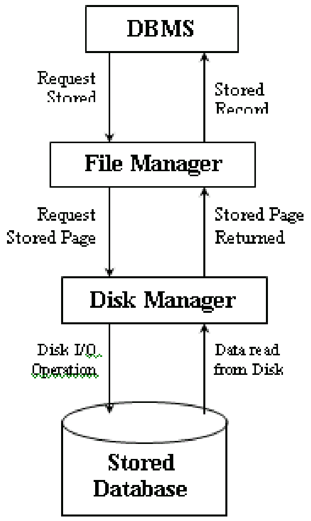
5-14 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Mapping

The above figure shows the relationships between the three types of scheme.
**Mapping** is the process of transforming requests and results between levels.
The database software is responsible for mapping between the three types of schemes. It must be capable of checking the schemes for consistency, and must use information in the schema via the conceptual schema.

The DBMS transforms a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for finally processing over the database.
If the request is to retrieve from the database, then the data extracted must be reformatted before presenting it to match the user's external view.
This process may sometimes be time-consuming, so some DBMS(s) do not support external views.

## Mapping Example 1



**What is going on when locating a specific piece of data in the database?**

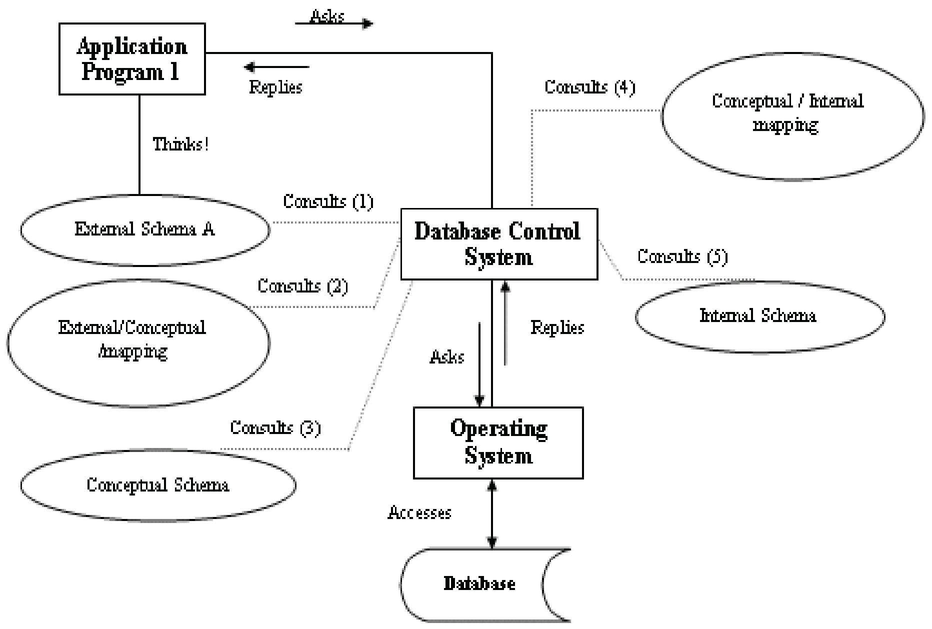
- The **DBMS** decides what stored record is required, and asks the **File Manager** to retrieve that record.
- The **File Manager** in turn decides what page contains the desired record and asks the **Disk Manager** to retrieve the page. The page is the unit of I/O. It is the amount of data transferred between the disk and main storage in a single disk access. Typical page sizes are 1K, 2K, or 4K bytes.
- The **Disk Manager** determines the physical location of the desired page on the disk, and issues the necessary disk I/O operation.

## How does the Application Program Access the Database?

The following steps illustrate the process

- A user issues an access request, using a data sub-language such as SQL. The DBMS receives analyses that request.
- The DBMS maps the request, in turn, from the external schema for the user, to the corresponding external/conceptual mapping, to the conceptual schema, to the conceptual/internal mapping, and to the storage structure definition.
- The DBMS executes the necessary operations on the stored database.

## Mapping Example 2



7-14 Copyright © Information Technology Institute, 2002. All rights reserved ITI

The above figure shows the way in which an application program accesses the database via the database control system. The number in parentheses shows the sequence, in which the schemes and mappings are cons ulted.

From this architecture point of view, the functions of the DBMS used include:

- **Data Definition**

The DBMS accepts data definitions (external schemes, conceptual schema, internal schema, and all mapping) in source code and converts them to the appropr iate object code.

- **Data Manipulation**

The DBMS is responsible for handling requests from the user to retrieve, and update existing data in the database, or to add new data to the database. The DBMS also include a data manipulation language (DML) processor component. A DML request may be "**Planned**"; where the need for the request was foreseen in advance of the time the request is actually executed. Or the request could be "**unplanned**", where the need was not seen in advance, but arose due to an unexpected event.

- **Data Security and Integrity**
  The DBMS monitors user requests and reject any
  attempts to violate the security and integrity checks defined by DBA.
- **Data Recovery and Concurrency** DBMS must enforce certain recovery and concurrency controls.
- **Data Dictionary** DBMS provides a data dictionary or metadata (data about data) function.
- **Performance** DBMS should provide all of the functions identified above as efficiently as possible.

## Advantages of the Three-Layer Architecture

The three-level architecture has many advantages regarding program/data independence:

- Changes in conceptual schema, for example addition of a data item, will not affect any existing application program.
- Changes in internal schema, such as changing block size or providing additional pointers, will not affect any existing application.
- Changes in external schema will not affect other application programs that do not use the external schema in question.

## Distributed Processing

A well-designed system architecture can highly affect the system performance.

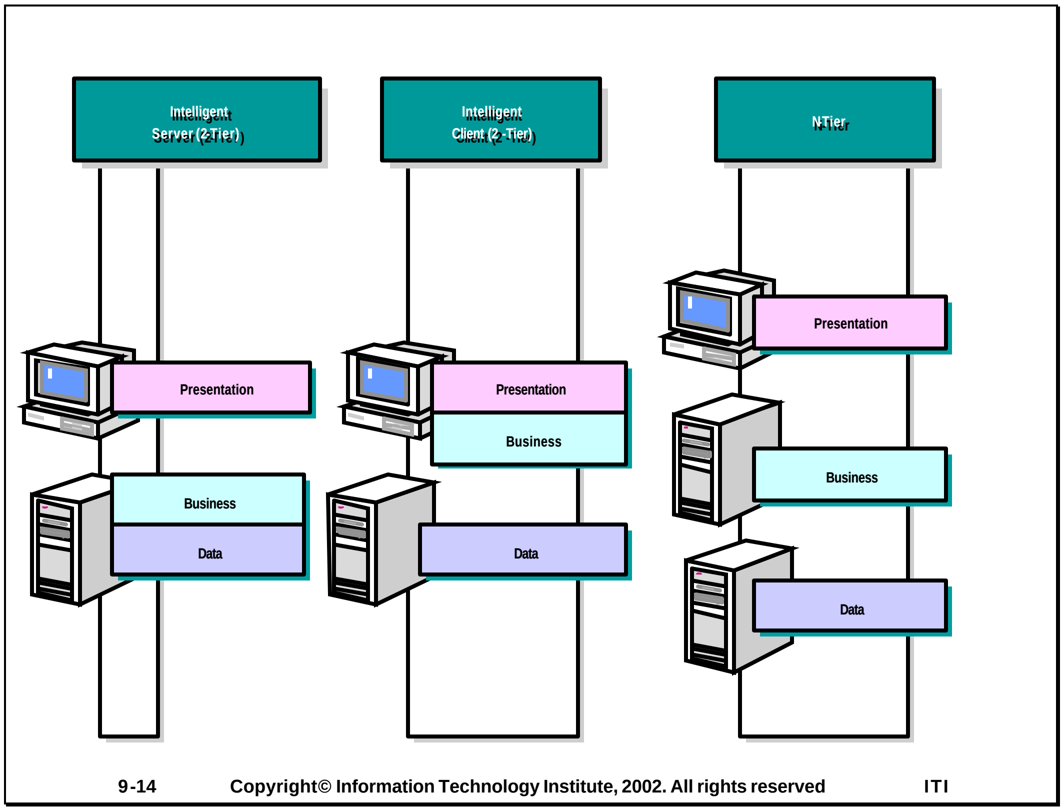
### Distributed Processing and Architectural Design

Planning a database design requires knowledge of the business requirements that will be modeled. The selected architecture affects how the software application is developed, deployed, and managed.
There are several application architectures that can be used to implement what is called "client/server" applications.
A database system can be divided into three logical layers, which can physically reside on one or more servers. These layers are:

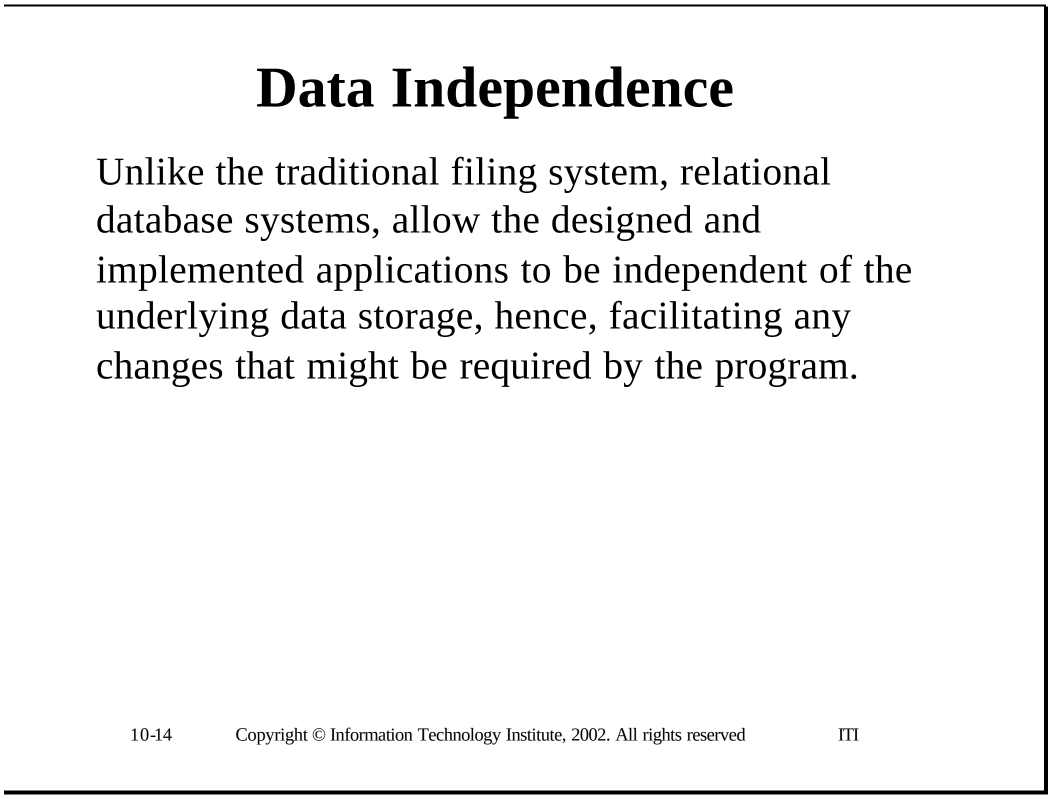
**Data:** Includes database definition, data integrity logic and all other objects that are associated with the data (The role of the DBMS).

**Business:** Includes the application logic and business rules. Functions like ensuring that the employee's salary does not exceed LE 5000 and has a positive value, calculating the net salary of the employee, etc. can be included.
This layer can reside on the client (application program), on the server (DBMS), be distributed on both of them, or reside on a separate layer.

**Presentation:** The way of presenting the data to the users and interfacing with the database. This is usually implemented on the client (application program).



This figure shows different models that can be used.
In the ‘Intelligent Server’ model, both of data and business logic reside on the server side, and the client side is responsible for the interface.
In the ‘intelligent client’ model, the data resides on the server, while the interface and the business modules are loca ted on the client.
The business layer can also be distributed on the client and the server machines or it can be installed on a separate machine.



## Data Independence

Unlike the traditional filing system, relational database systems, allow the designed and implemented applications to be independent of the underlying data storage, hence, facilitating any changes that might be required by the program.

### Data Independence

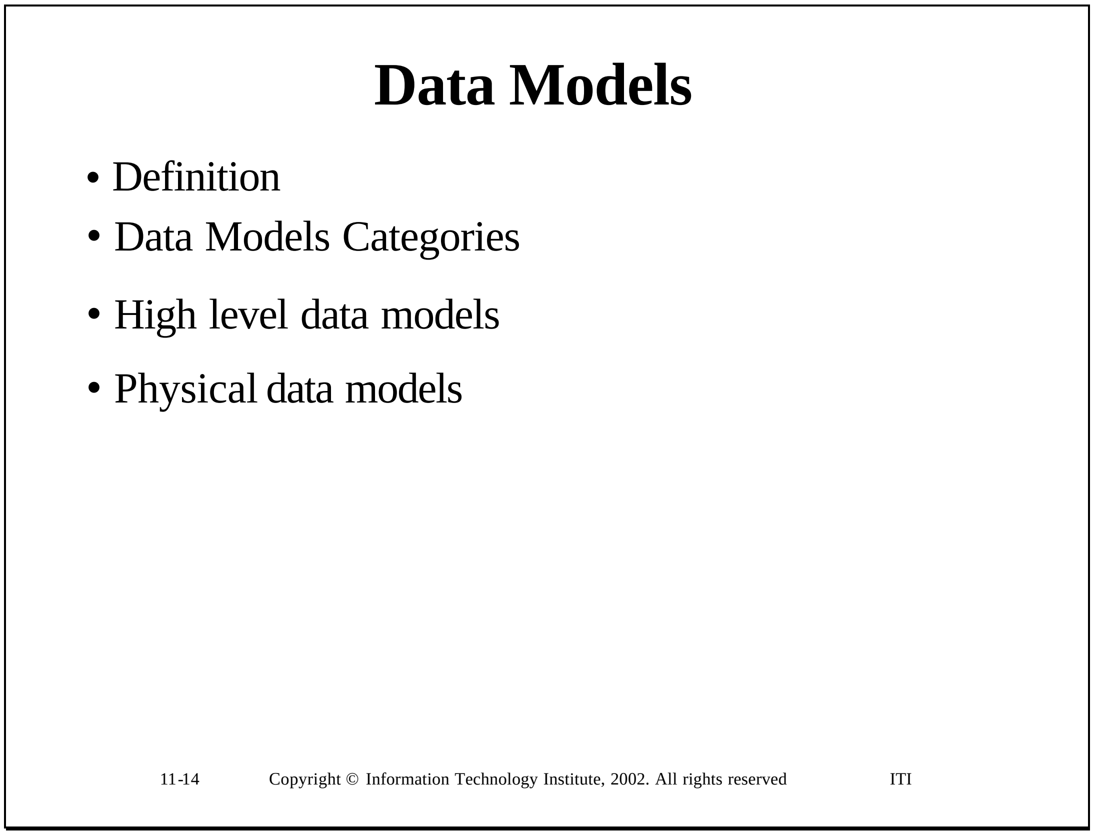
One of the advantages of a database system is that different users can access the database for data retrieval and report generation without worrying about the structure or access strategy of the data storage. For example, to generate a customer list from customers table, the user does not have to know if the field ‘customer-no’ is a key or not. In addition, the form of the query will not change if ‘customer-no’ is made a key, so the access strategy is invisible to the user.

On the other hand, if the report is generated using a file management system, the program must know how the file is structured. Moreover, knowledge of the data organization and the access techniques is built into the application logic and code.

For example, consider an application that was process an employee file, and assume that, for performance reasons, the file was stored indexed on the “employee number” field. The (data-dependent) application will typically be aware of the fact that the index exists, and aware also of the file sequence as defined by that index. The internal structure of the application will be built around these facts.
In this example, the application is data dependent, because it is impossible to change the storage structure (how the data is physically stored) or access technique (how it is accessed) without affecting the application.

In a database system, on the other hand, any application is independent of the storage structure and access strategy. This feature is referred to as **data independence**.

Using the three schema architecture, data independence can be defined as the ability to change the schema at one level of a database system without having to change the schema at the other levels.



## Data Models

- Definition
- Data Models Categories
- High level data models
- Physical data models

### Data Models

**A model** is a pictorial representation of reality.

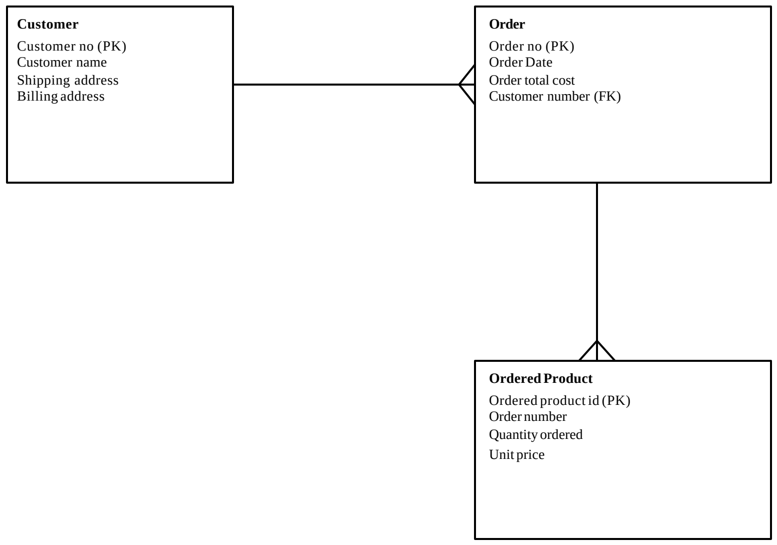
**Data modeling** is a technique for organizing and documenting a system’s data, a collection of concepts that can be used to describe the structure and logical organization of the database. Data modeling is sometimes called database modeling because a data model is usually implemented as a database. It is sometimes referred to as information modeling.

**Categories of data models**

- High Level (Conceptual) Data Models provide concepts that are close to the way many users perceive data; entities, attributes, and relationships. An example of such models is the Entity-Relationship model (ER).

- Physical Data Models describes how data is stored in the computer and the access path needed to access and search for data.

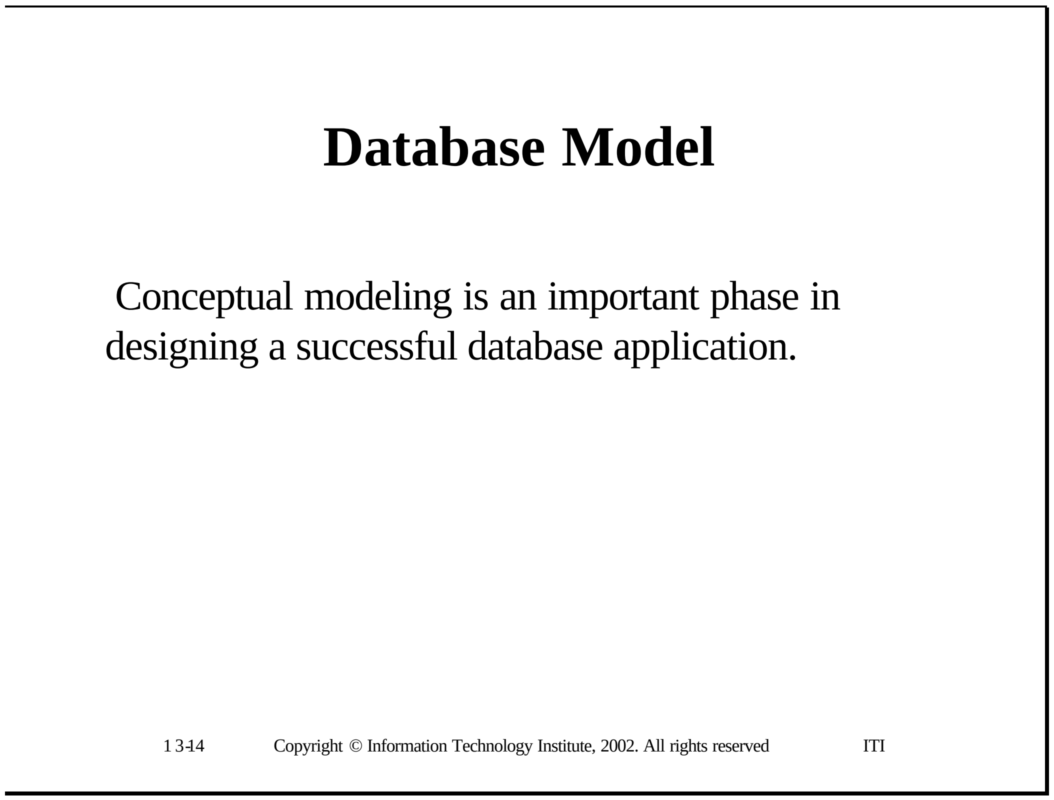
## Example



The above diagram shows you an example of a simple data model called an entity relationship diagram or ERD, even if you don't understand yet what we mean by entity relationship diagram, the diagram makes the following business assertions:

- We need to store data about customers, orders and the ordered products.
- The value of customer number uniquely identifies one and only one customer. The value of order number uniquely identifies one and only one order.
- For a customer we need to know the customer name, shipping address and billing address.
- A customer can order zero, one or more orders.
- An order sold can have or more ordered products. Thus, an order must contain at least one ordered product.

After knowing more about ER model, you will be able to read data models and construct them



## Database Model

Conceptual modeling is an important phase in designing a successful database application.

### Database Model

As mentioned before, a data model is defined as a set of guidelines for representing the logical organization of data in the database.

Given an enterprise with a defined collection of data, the mapping of this data onto the real DBMS is done based on a data model. Various architectures exist for databases and various models have been proposed including the:

- Relational model
- Network model
- Hierarchical model

Next is a short insight into the network and hierarchical models. The hierarchical model and the network model are both defined by a process of abstraction from implemented systems. The relational model is discussed in more details in the next lesson.

- **Hierarchical model overview**

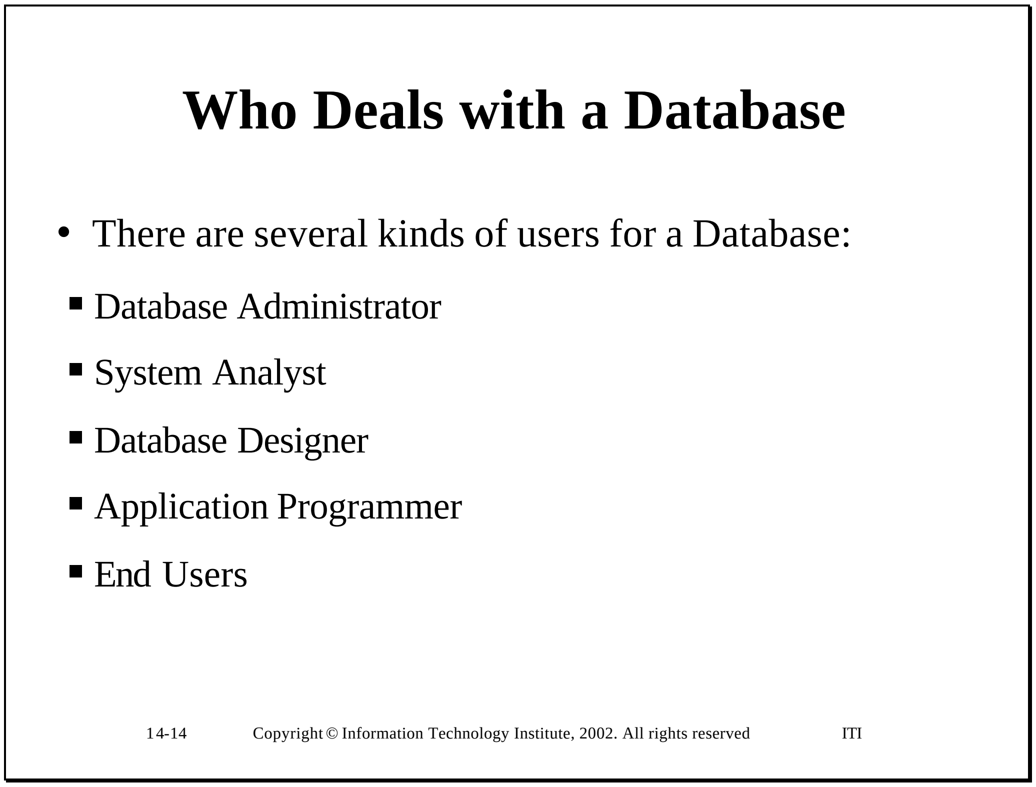
A hierarchical data structure consists of an ordered set of trees - or more precisely, an ordered set consisting of multiple occurrences of a single type of tree.
There are general rules for data manipulation and data integrity contained within the general schema for a hierarchical system. These however differ from system to system (e.g. the IMS system contains its own specific IMS data manipulation section … etc).

- **Network model overview**

  A network data structure can be regarded as an extended form of the hierarchical data structure. The main difference between the two is that in a hierarchical structure, a child record has exactly one parent whereas in a network structure, a child record can have any number of parents (possibly even zero).

  A network database consists of two data sets, a set of records and a set of links, where the record types are made up of fields in the usual way.

  Networks are complicated data structures. Operators on network databases are complex, functioning on individual records, and not sets of records. Increased complexity does not imply increased functionality but on the contrary, the network model is not more powerful than the relational model. A network-based DBMS can however provide good performance because of its lack of abstraction and its closeness to the storage structure used, though this is at the expense of good user programming. The network model can also incorporate certain integrity rules.



## Who Deals with a Database

- There are several kinds of users for a Database:
  - Database Administrator
  - System Analyst
  - Database Designer
  - Application Programmer
  - End Users

- **Database Administrator**

Is the person responsible for putting and implementing the suitable strategy and policy for the whole database, in addition to providing the necessary technical support.
The functions of the DBA include defining the internal and conceptual schema, defining security and integrity checks, defining backup and recovery procedures, monitoring performance and responding to changing requirements.

- **System Analyst**

Is the person responsible for determining the requirements of end users, especially naïve and parametric end users, and also develops specifications for canned transactions that meet these requirements.

- **Data Designer**

Designers are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store this data. These tasks are undertaken before the database is actually implemented.

- **Application Programmer**

Is the person who implements these specifications as programs with a high-level language to build canned transaction. Then tests, debugs, documents, and maintains these transactions.

- **End Users**

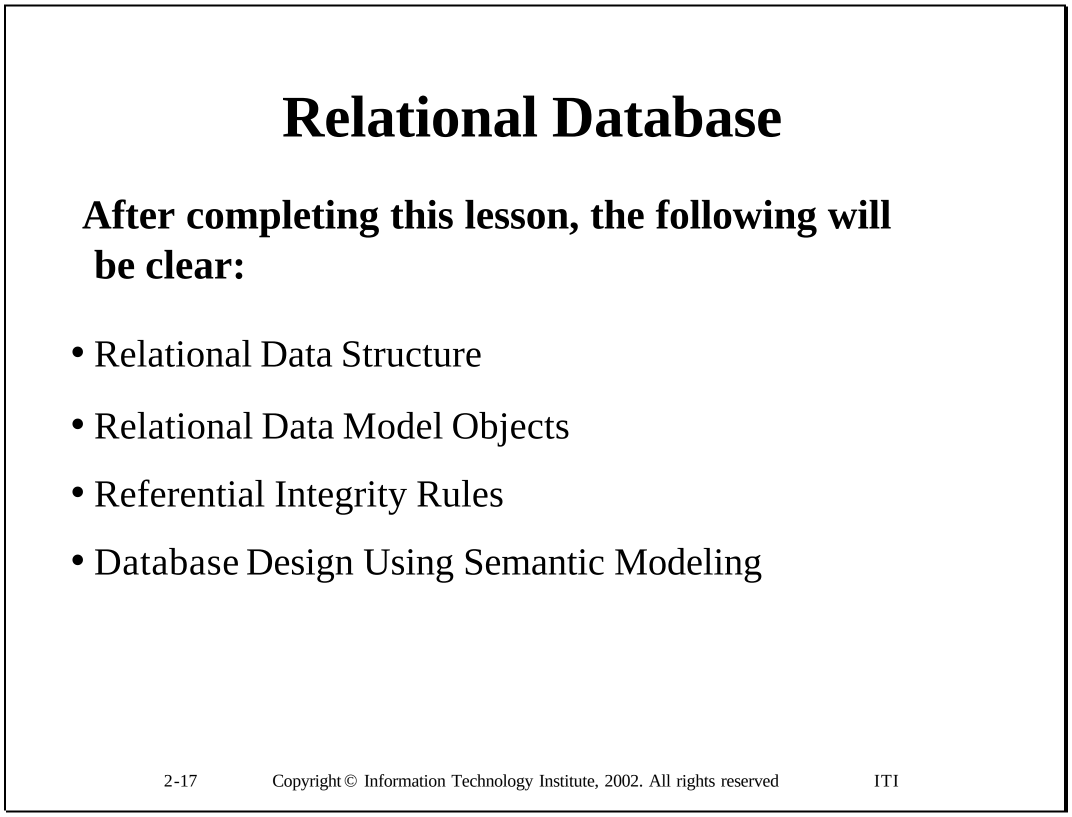
These are the people whose jobs require access to the database for querying, updating and generating reports.

▪ **Casual end users** occasionally access the database, but they may need different information each time, such as managers.
▪ **Naïve and parametric end users**: Their main job is to query and update the database using types of queries and updates called canned transactions, such as bank tellers and reservation clerks.

▪ **Sophisticated end users**: They are familiar with the facilities of DBMS, such as engineers, scientists and business analysts.

▪ **Stand-alone users:** They are online users who use an online application, through menus, forms, and query languages.

# Chapter 3
# Relational Database

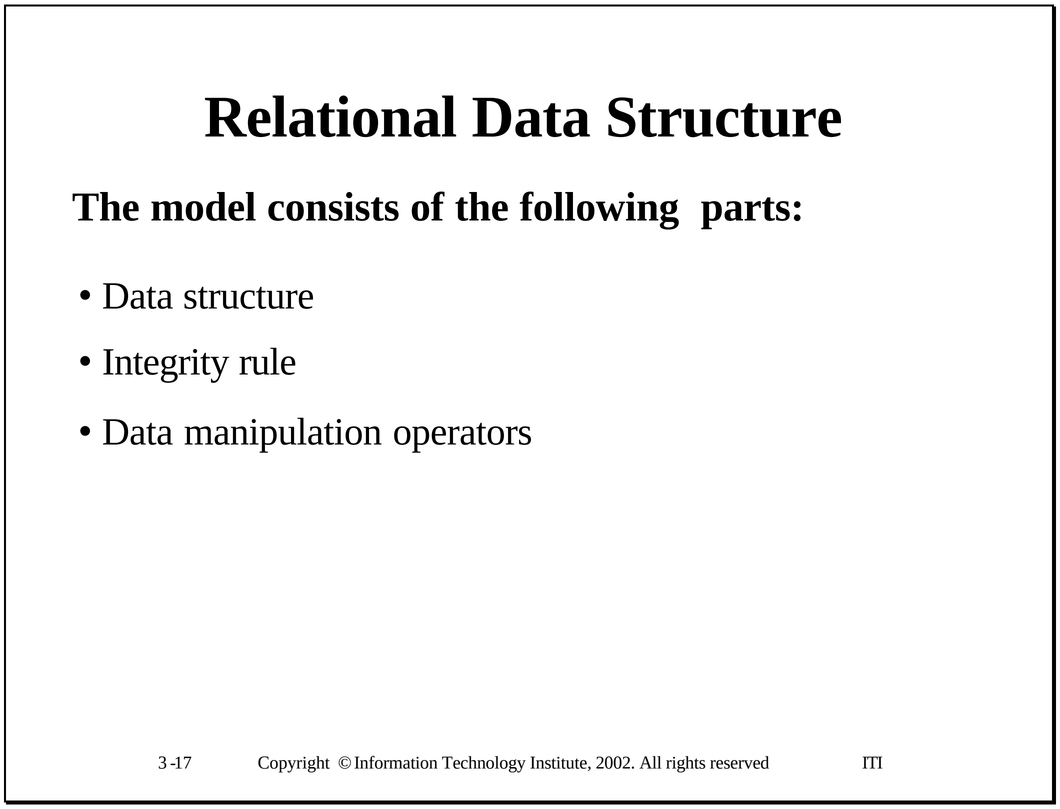


## Relational Database

**After completing this lesson, the following will be clear:**

- Relational Data Structure
- Relational Data Model Objects
- Referential Integrity Rules
- Database Design Using Semantic Modeling

2-17 Copyright © Information Technology Institute, 2002. All rights reserved ITI



## Relational Data Structure

**The model consists of the following parts:**

- Data structure
- Integrity rule
- Data manipulation operators

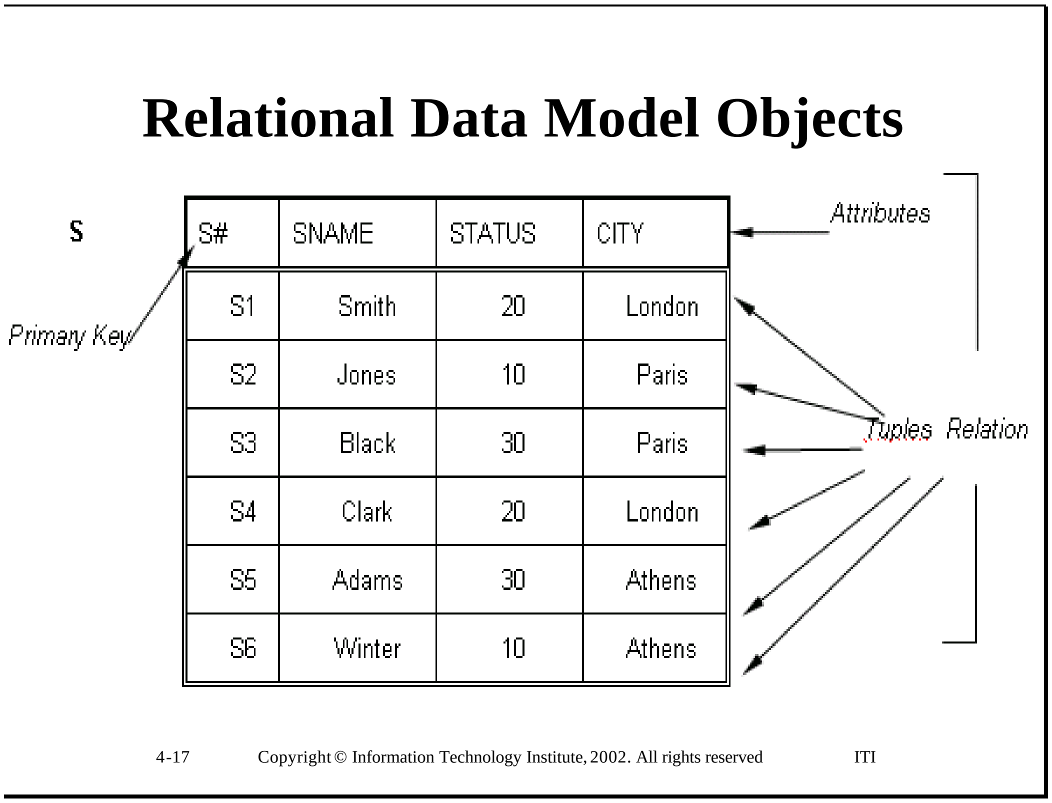
3 -17 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Relational Data Structure

The relational data model basically consists of three parts:

- Data structures called tables or relations.
- Integrity rules on allowable values and combinations of values in tables.
- Data manipulation operators which can be either the relational algebra or the relational calculus plus an assignment operator.

It is important to note that the relational model is a paper based model and that not all of its features are implemented in relational database systems.



## Relational Data Model Objects

| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Jones | 10 | Paris |
| S3 | Black | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |
| S6 | Winter | 10 | Athens |

## Relational Data Model Objects

In the relational database model, the following terms must be defined:

**A relation:** It corresponds to what is normally called a table. It is a collection of attributes describing data.

**A tuple:** It corresponds to a row of such a table.

**An attribute:** It corresponds to a column of such a table.

**A primary key:** It is unique identifier for the table. It may be one attribute or a combination of attributes.

**A domain:** It is a pool of values, from which one or more attributes draw their actual values.

## There are also some other terms concerning the relation:

• **Null Value**

An attribute value may be with null value that is not yet known or not applicable. The null value will be represented by blank. Null values cannot be used in the **primary key** field.

• **Redundant & Duplicated Data**

**Redundant data:** is the data that can be deleted without information being lost.

**Duplicated data:** is Present when an attribute has two or more identical values.
To eliminate data redundancy, the table can be split as shown

| Supplier # | Part # | Part- description |
|---|---|---|
| S2 | P1 | Bolt |
| S7 | P6 | Bolt |
| S2 | P4 | Nut |
| S5 | P1 | Bolt |

| Supplier # | Part # |
|---|---|
| S2 | P1 |
| S7 | P6 |
| S2 | P4 |
| S5 | P1 |

Part-1

| Part # | Part- description |
|---|---|
| P1 | Bolt |
| P6 | Bolt |
| P4 | Nut |

- **Repeating Groups**

In the following table: the attribute Part# is an example of a repeating group

The information of suppliers and parts are illustrated below:

| Supplier# | Supplier-name | Part# |
|---|---|---|
| S5 | Wells | P1 |
| S2 | Heath | P1, P4 |
| S7 | Barron | P6 |
| S9 | Edwards | P8, P2, P6 |

| Supplier# | Part-description | Part# |
|---|---|---|
| S5 | wood | P1 |
| S2 | wood | P1<br>P4 |
| S7 | glass | P6 |
| S9 | fiber | P8<br>P2<br>P6 |

To eliminate the repeating groups. we use normalization or split the table into two.

The supplier can be considered as a separate entity (no existence of part number with it), and the part is considered as another separate entity. Information about which supplier supplies which parts is stored in a separate entity called Supp-Part.

Supplier

| Supplier# | Supplier-name |
|---|---|
| S5 | Wells |
| S2<br>S2 | Heath<br>Heath |
| S7 | Barron |
| S9<br>S9<br>S9 | Edwards<br>Edwards<br>Edwards |

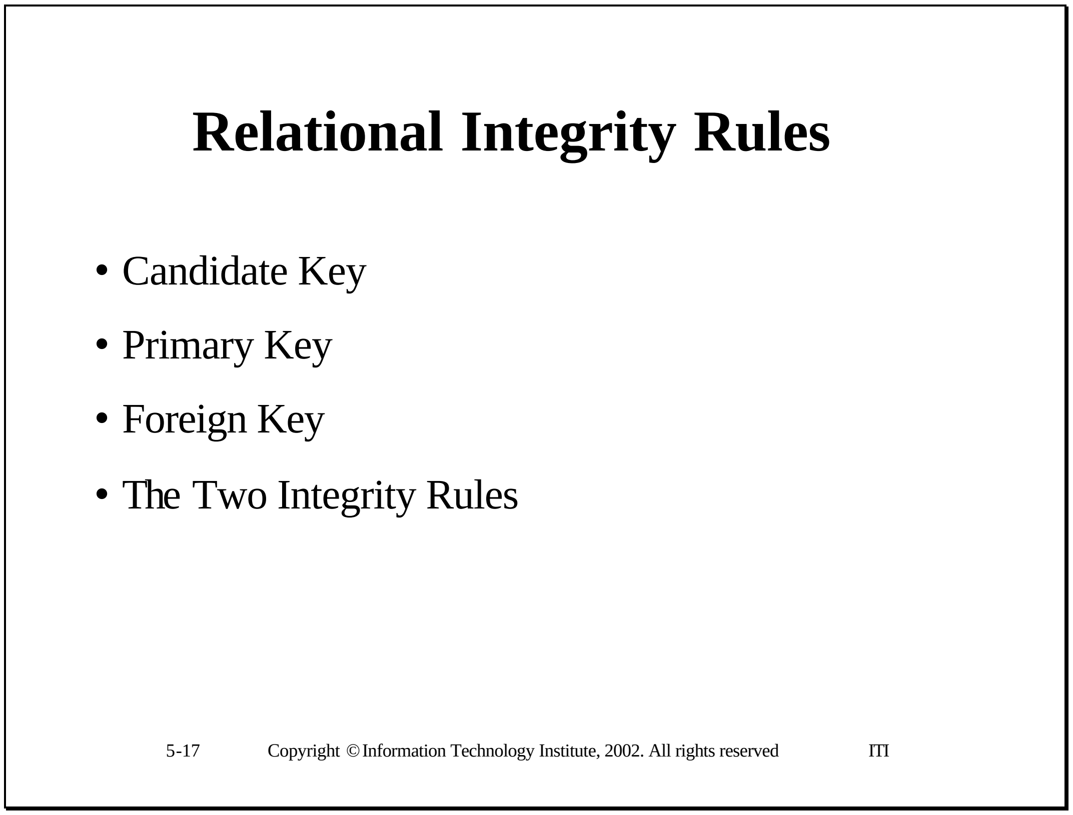
Supplier/Part

| Supplier# | Part# |
|---|---|
| S5 | P1 |
| S2<br>S2 | P1<br>P4 |
| S7 | P6 |
| S9<br>S9<br>S9 | P8<br>P2<br>P6 |

Part

| Part# | Part-name |
|---|---|
| P1 | wood |
| P2<br>P3 | xxx<br>yyyy |
| P4 | zzzz |

More details about normalization and table splitting will be discussed later.



## Relational Integrity Rules

- Candidate Key
- Primary Key
- Foreign Key
- The Two Integrity Rules

5-17 Copyright © Information Technology Institute, 2002. All rights reserved ITI

- **Candidate Key**

Any relation has at least one candidate key. Each candidate key K of a relation R has the two following properties:

**Uniqueness:** At any given time, no two tuples of **R** have the same value for ***K***.

**Minimality:** If ***K*** is a composite, then no component of ***K*** can be eliminated without destroying the uniqueness property

- **Primary key**

Each relation has only one primary key. It is the chosen candidate key.

- **Foreign key**

Is an attribute (or attribute combination) in one relation R2 whose values are required to match those of the primary of some relation

**Example:**

This is a primary key R1: S(**s#**, sname, status, scity).

This is a foreign key R2: SP(**S#**, P#).

Note that 'S#' is primary key in the relation R1 and foreign key in the relation R2.

## • The Two Integrity Rules

### ▪ Entity Integrity

No attribute participating in the primary key of a base relation is allowed to accept null values.

### ▪ Referential Integrity

If base relation R2 includes a foreign key **FK** matching the primary key **PK** of some base relation R1, then every value of **FK** in R2 must either equal a value of primary key **Pk** in some tuple of R1, or be wholly null.

**For DELETE and UPDATE commands,** there are three possibilities:

**CASCADES:** where the foreign key in the case matching is deleted/updated.

**RESTRICTED:** only primary keys with no matching foreign keys can be deleted/updated.

**NULLIFIES:** all matching foreign keys are set to null, and delete/update then takes place in the other relation.

**FOREIGN KEY** clause - syntax:

```
FOREIGN KEY  (foreign-key IDENTIFIES target
NULLS      [NOT] ALLOWED
DELETE    OF  target effect
UPDATE   OF  target-Primary-key effect)
```

Where:

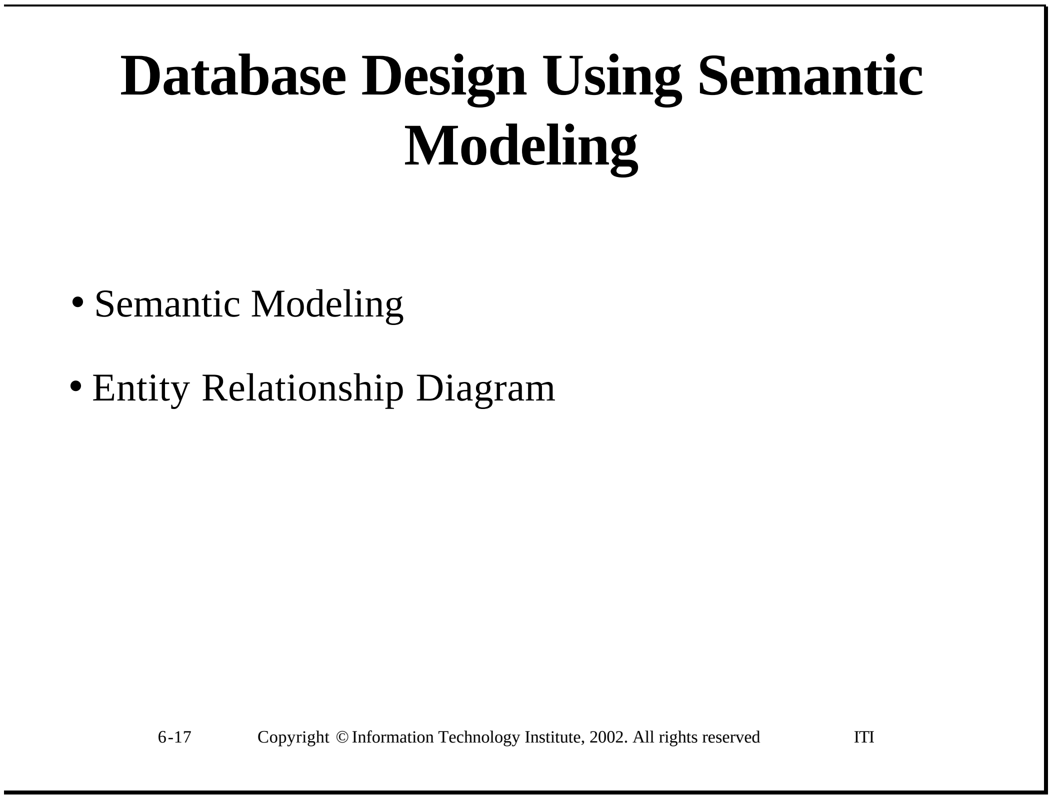
- ♦ "foreign-key" is the same as the Primary key in the PRIMARY KEY.
- ♦ "target" is a relation name.
- ♦ "Target-Primary-key" specifies the "Primary-key" for "target", and

  "effect" is CASCADES or RESTRICTED or NULLIFIES.

**Example:**

```
CREATE   TABLE   SP
(S# ..., p#, ..., QTY ...)
PRIMARY   KEY (S#, P#)
```

```
FOREIGN   KEY (S# IDENTIFIES S
                 NULL  NOT ALLOWED
                 DELETE   OF  S RESTRICTED
                 UPDATE   OF  S.S# CASCADES)
  FOREIGN   KEY (P# IDENTIFIES P
                  NULL  NOT ALLOWED
                  DELETE   OF  P RESTRICTED
                  UPDATE   OF  P.P# RESTRICTED);
```



## Database Design Using Semantic Modeling

- Semantic Modeling
- Entity Relationship Diagram

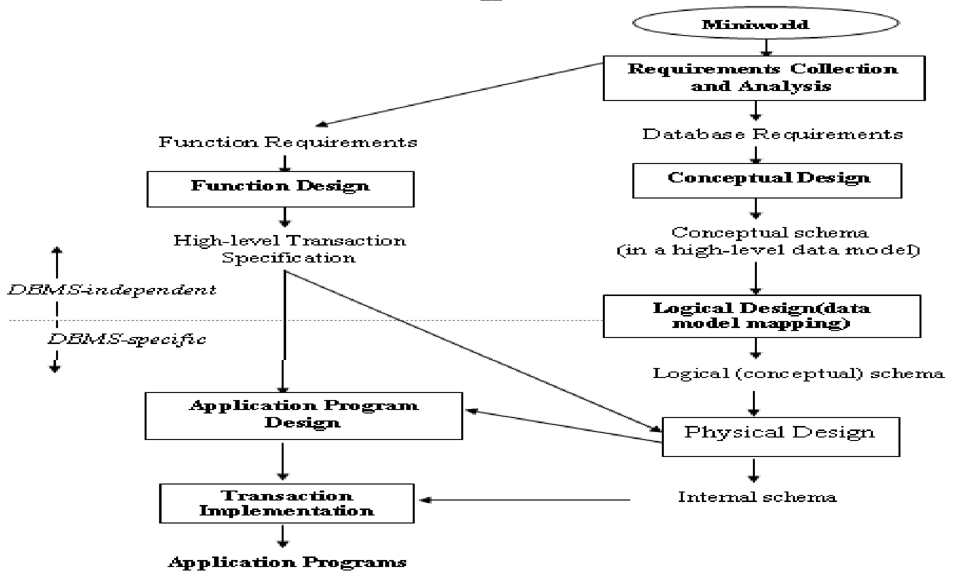
- **Semantic Modeling**

Semantic modeling is sometimes called data modeling, ER modeling or Entity modeling.

- **Entity Relationship Diagram**

Entity relationship diagram is a way to represent logical database structure in a pictorial, simple and readily understood way.
It is a ***top-down*** methodology used for designing the database, starting at real-world constructs (entities) and finishing at low level abstractions (database).

## Database Design Process


Miniworld
Requirements Collection and Analysis
Function Requirements
Database Requirements
Function Design
Conceptual Design
High-level Transaction Specification
Conceptual schema (in a high-level data model)
DBMS-independent
DBMS-specific
Logical Design(data model mapping)
Logical (conceptual) schema
Application Program Design
Physical Design
Transaction Implementation
Internal schema
Application Programs

*Phases of Database Design*

7-17 Copyright © Information Technology Institute, 2002. All rights reserved ITI

**The figure shows a simplified description of the database design process.**

- **Requirement collection and analysis:** during which the designers interview potential database users to understand and document their data requirements. At the same time, information is gathered about the **functional requirements** of the application. These consist of the user-defined operations (or transactions) that should be performed by the database (such as retrieval and update operations). The result of this step is a set of written user's requirements.

- **Conceptual database design:** Once all the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database, using high-level conceptual data model. **The conceptual schema** is a description of all the data requirements of the users. It specifies the logical data content of the database and the constraints, which apply to the data. These are expressed using the concepts provided by the high-level data model.

- After the conceptual schema has been designed, the basic data model operations can be used to specify **high-level transactions** corresponding to the user-defined operations.

- **Logical database design** or **data model mapping:** this step involves the actual implementation of the database, using a commercial DBMS. Its result is a database schema in the specific language of the DBMS.

- **Physical database design**: this is the last phase, during which the internal storage structures and file organizations for the database are specified. Application programs are often designed and implemented in parallel.

  After meeting with customers and gathering all their requirements, deriving a schema from such requirements is a very important process. We will now discuss the basic concepts involved in the construction of an ER model schema.

- **Entities**

  An entity is defined as “a thing which can be distinctly identified“. It is a distinguished real world object that is to be represented in the database; each entity has attributes or properties.

  **Example** :

- The entity lecture has the properties place and time.
- The entity employee has the properties name, address, date of birth and telephone.

**An entity instance** is a single occurrence of an entity.

**Example** :

The entity student may have multiple instances: Mary, Mark, Susan, and so forth.

In data modeling, we don’t concern ourselves with individual students because we recognize that each student is described by similar pieces of data.

In an ER diagram an entity is represented by a box.

An entity can be classified into:

- **Weak entity** whose existence is dependent on some other entity. It cannot exist if the other entity does not also exist. A weak entity is represented by a double-lined box.

  **Example** :

- The branches of a company, can be considered as weak entity, as it depends on the entity of the company. It is considered weak because it cannot exist without the existence of the company entity.
- Relative entity depends highly on the person entity, so you will never find such entity without the existence of the person entity itself.

- **Regular entity** is an entity whose existence is dependant on any other entities. It is represented by a box in the ER diagram.

- **Attributes**

Entities have properties called attributes. An attribute is a descriptive property or characteristic of an entity. Synonyms include element, property, and field.

**Attributes can have several properties:**

**Simple or composite**:

**Simple (atomic)** attributes are those that can't be divisible. **Composite** attributes can be divided into smaller subparts, representing more basic attributes. For example, the Address attribute may be subdivided into street address, city, state and Zip. Street address may be divided into Number, street name, and apartment number. A simple attribute is represented by an ellipse.

**Single or multi-valued:**

Most attributes have a single value for a particular entity; such attributes are called **single-valued**. For example, Age is a single-valued attribute of a person. In some cases an attribute can have a set of values for the same entity. For example a Color attribute for a car, or a College Degree attribute for a person. These attributes are called **multi-valued**. An attribute is represented by a double ellipse if it is multi-valued.

**Base or derived:**

In some cases, two or more attribute values are related; for example the Age and Birth-Date attributes of a person. For a particular person entity, the value of Age can be determined from the current date and the value of that person's Birth-date. The Age attribute is called a **derived** attribute.

**Missing (Null Attributes):**

Some attributes may not have a value, or may not apply to a certain instance. For example, College Degree for a person may have the value NULL as that person does not have any college degree.

**Default** value for an attribute is the value that will be recorded if not specified by the user.

**Key** is an attribute**,** or a group of attributes, that assumes a unique value for each entity instance. In ER diagrams, key attributes are underlined in their ellipses.

- **Candidate key** is a “candidate to become the primary identifier” of instances of an entity.

- **Primary key** is the candidate key that will most commonly be used to uniquely identify a single entity instance.

**For example,** For the above entity “employee”, the employee number, name, age, birth-date, address, telephone number…etc. can be considered as attributes.

The employee number can be considered as a key, as each employee has his own unique number.

The **Data type** for an attribute defines what class of data can be stored in that attribute. It is similar to declaring types for variables in most programming languages.

- **Relationships**

A relationship is an association between two or more entities. More than one relationship can exist between any two entities. The entities involved in a given relationship are said to be the participants in that relationship.

**For example:**

There is a relationship between departments and employees, representing the fact that a given department employs a given set of employees.

In an ER diagram a relationship is represented by a diamond.

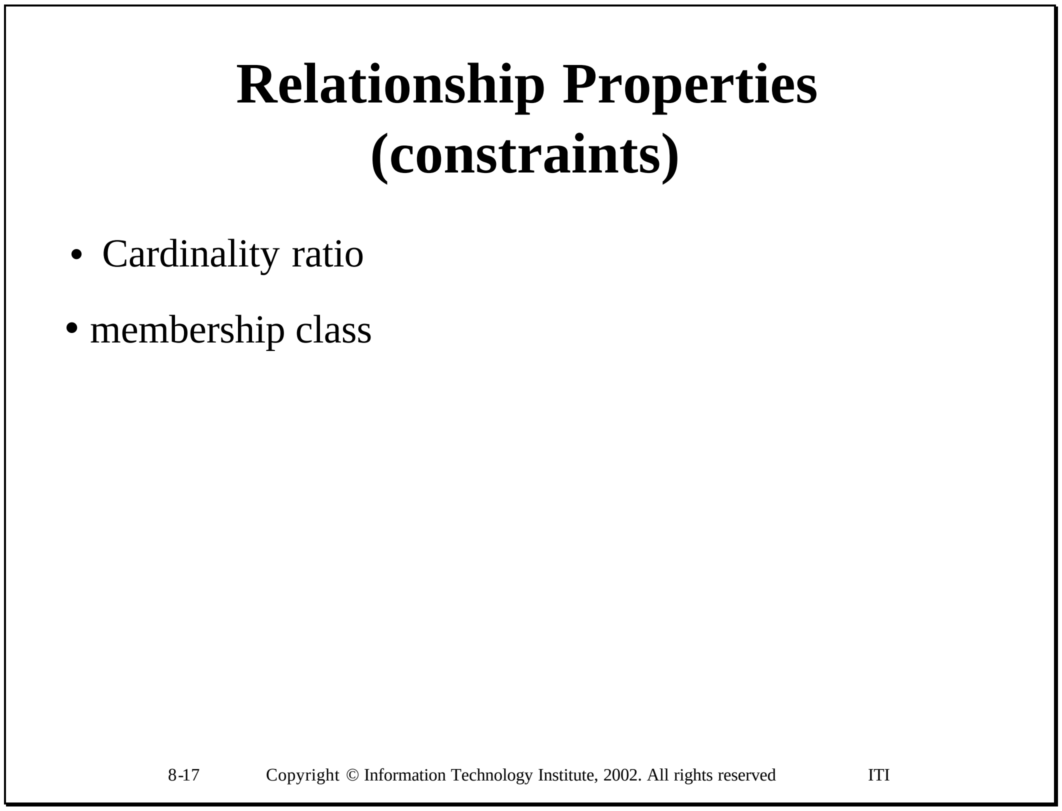
**Constraints on Relationships**

Relationships have certain constraints that limit the possible combination of entities participating in relationship instances.

Constraints are determined from the mini-world situation that the relationship represents.

In the ER model there are two main types of constraints:

- Cardinality ratio
- Membership class



## Relationship Properties (constraints)

- Cardinality ratio
- membership class

8-17 Copyright © Information Technology Institute, 2002. All rights reserved ITI

· **The Cardinality ratio Constraint**

Specifies the number of relationships that an entity participates in Common cardinality ratios for a relationship between two entities are:

▪ **1:1 (one to one)**

**Example:**

The relation between car and employee.

An employee uses at most one car.

A car is used by at most one employee.

▪ **1:M (one to many)**

**Example:**

An employee uses at most one car.

A car can be used by several (**many**) employees.

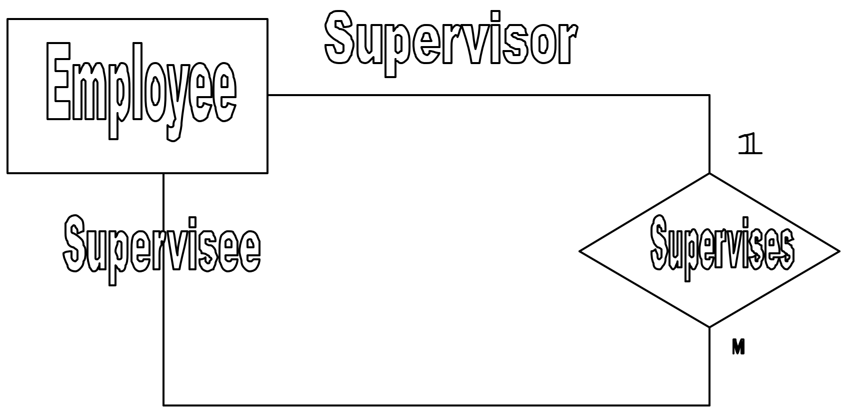
- **M:M (many to many)**

**Example:**

An employee uses several (**many**) cars.

A car can be used by several (**many**) employees.

## Recursive Relation



- **Recursive relationshipss**

A relationship in which the same entity participates more than once is called a recursive relationship.

The figure shows an example. The supervision relationship relates an employee to a supervisor where both employee and supervisor entities are members of the same employee entity type.

Thus the employee entity type participates twice in the supervision relationship, once in the role of supervisor and once in the role of supervisee.

- **The membership class constraint**

It specifies whether the existence of an entity depends on being related to another entity.

The membership class (participation) of an entity E in a relationship R is **obligatory** (Total Participation) if every E occurrence must participate in an R relationship occurrence.

The membership class of an entity E in a relationship R is **non-obligatory** (Partial Participation) if an E occurrence can exist without participating in an R relationship.

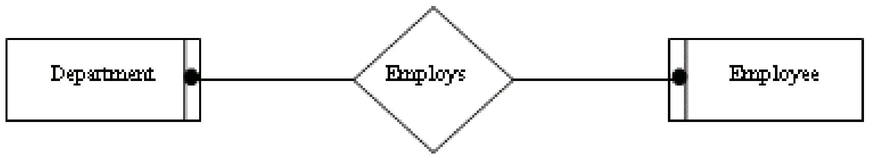
**Example:**

Suppose the enterprise rules for an employee's relationship between Department and Employee are:

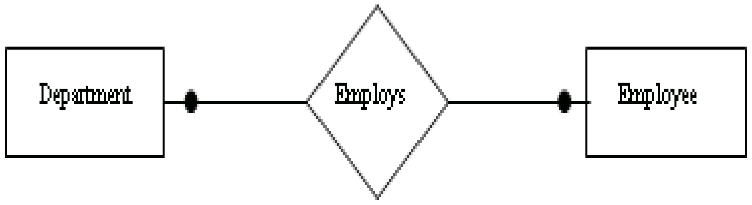
- Every employee must be employed within a department
- A department may exist even if it has no employees

We will say that membership of Employee in the Employs relationship is **obligatory**. Membership of department in the Employs relationship is **non-obligatory**.

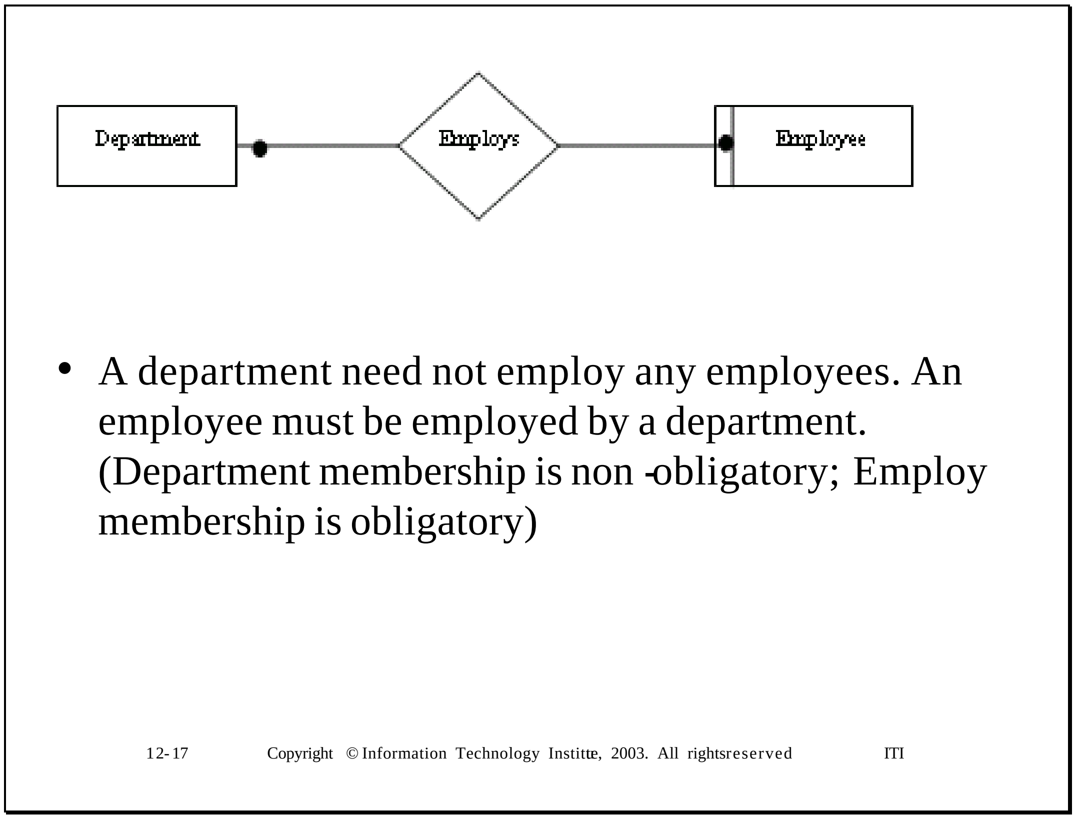
For a relationship between two entity types, there are then four possible combinations of membership class:

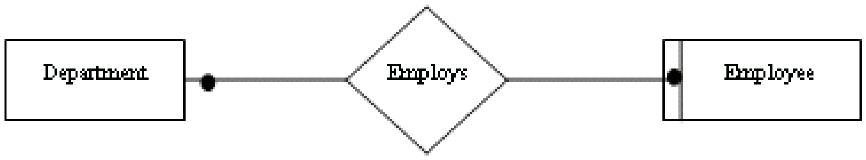


- A department must employ at least one employee. An employee must be employed by a department, (Department membership is obligatory Employee membership is obligatory)

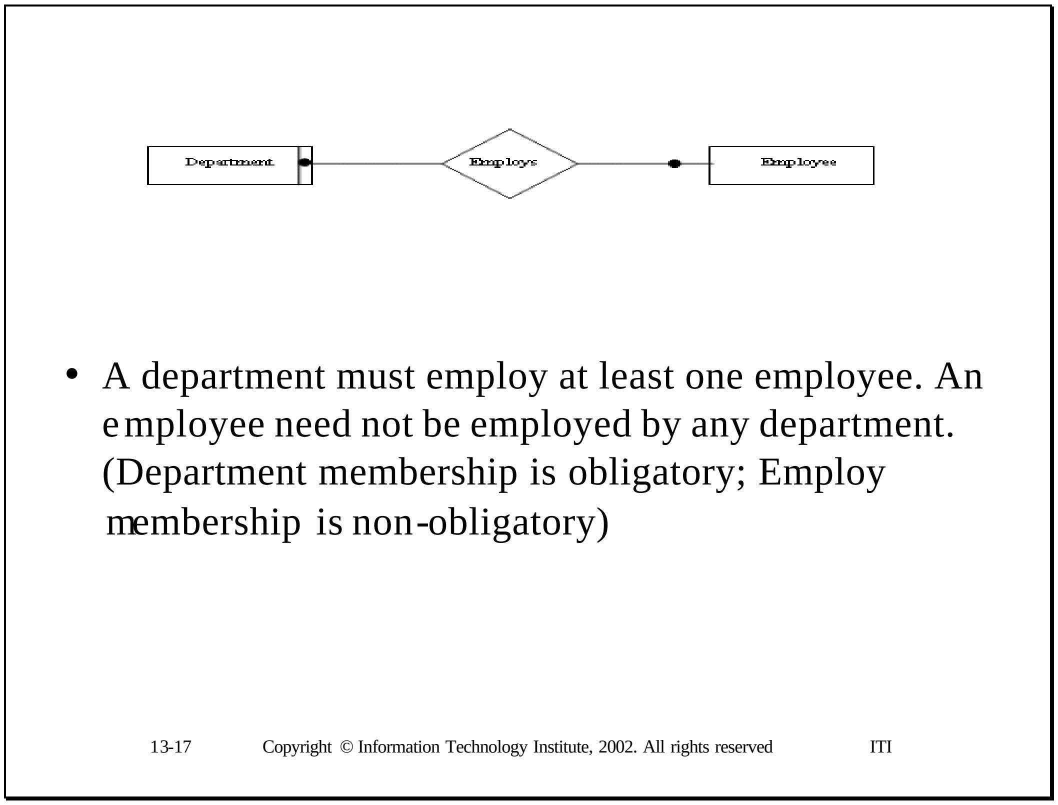


- A department need not employ any employees. An employee need not be employed by any department. (Department membership is non-obligatory; Employ membership is non-obligatory)



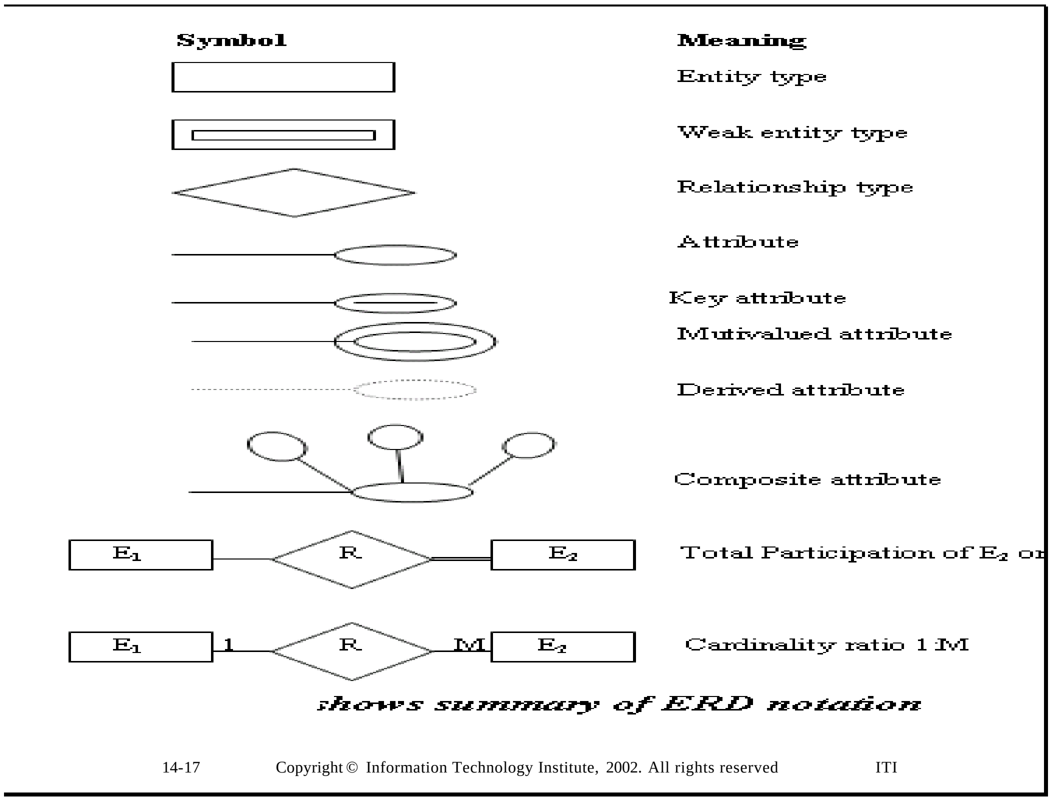


- A department need not employ any employees. An employee must be employed by a department. (Department membership is non -obligatory; Employ membership is obligatory)

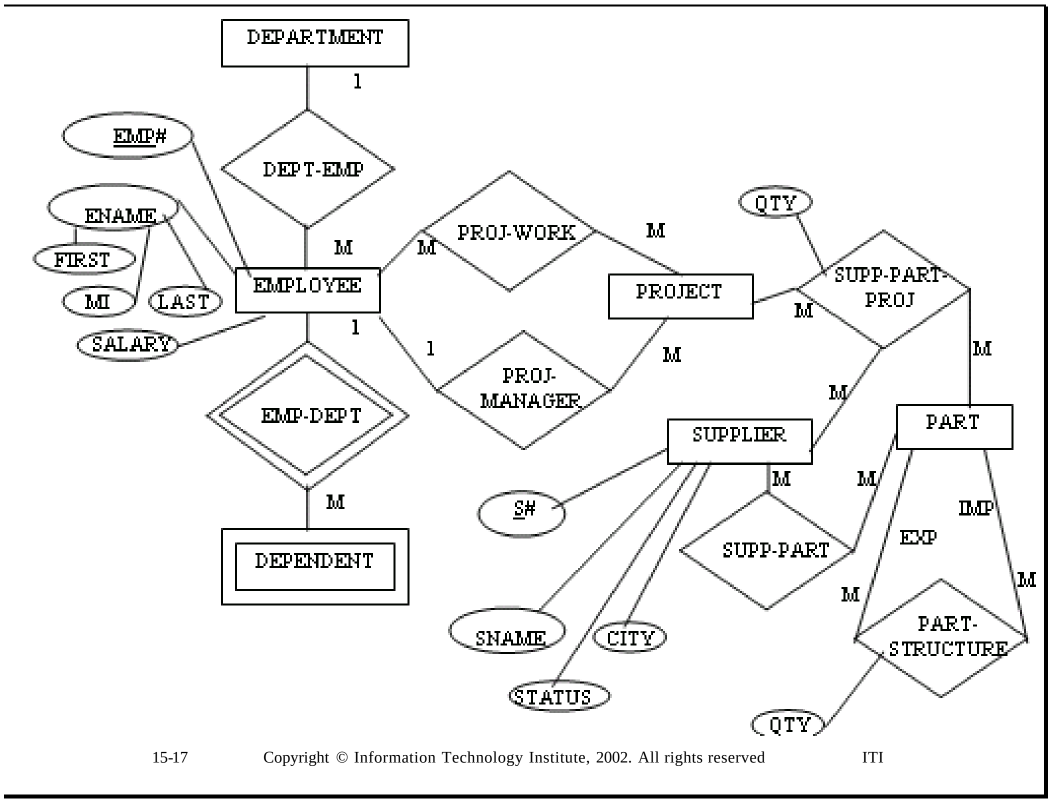

Department
Employs
Employee

- A department must employ at least one employee. An employee need not be employed by any department. (Department membership is obligatory; Employ membership is non-obligatory)

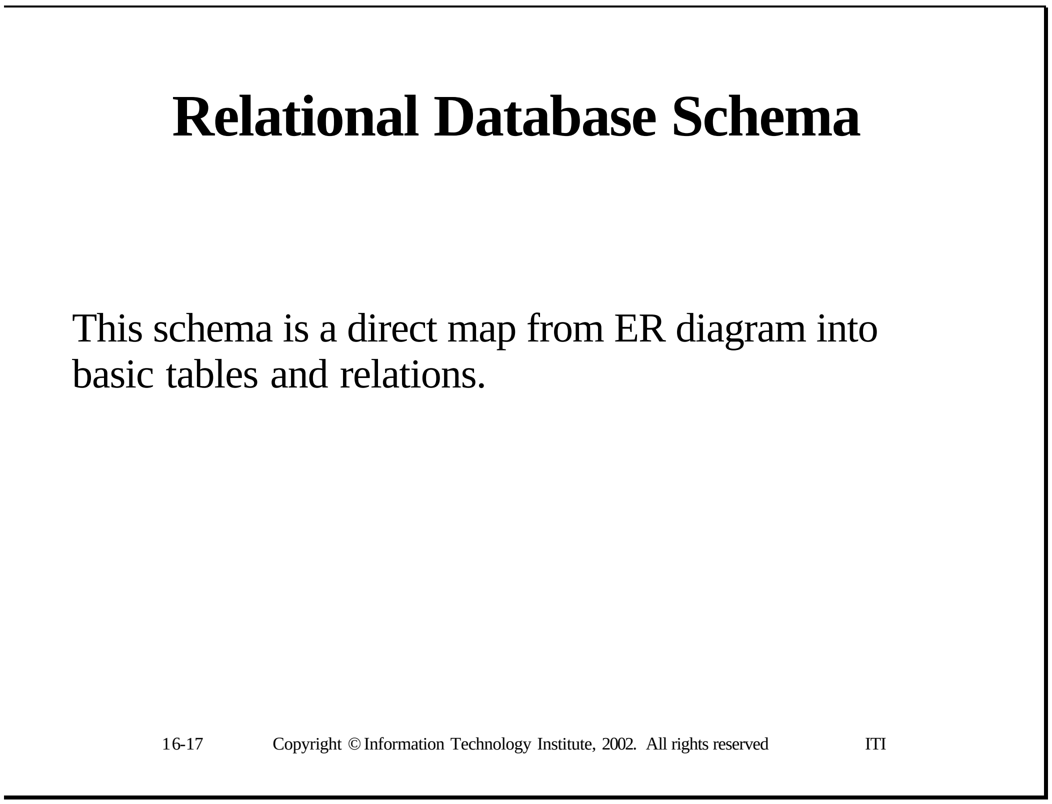
13-17 Copyright © Information Technology Institute, 2002. All rights reserved ITI



The figure shows a summary of ERD notation.



The figure shows an example for a complete E/R diagram for a company database.



## Relational Database Schema

This schema is a direct map from ER diagram into basic tables and relations.

### Relational Database Schema

The relational schema is a direct map from the ER diagram into basic tables with defined columns and relations

For example the employee entity will be translated into the following table:

| <u>**ID**</u> | Phone | Name | Birth_Date | Address |
|---|---|---|---|---|

The table contains the Id, Phone, Name, Birth_date and address columns with a primary key on the column ID.
The following is a list of the terms used in the relational database structure:

- **Table**

  A table in a relational system consists of a row of a column headings, together with zero or more rows of data values (different numbers of data rows at different times).A data value is stored in the intersection of a row and column.

- **Constraints**

  Constraints are some rules and steps that can be done in order to guarantee the correctness and integrity of the data stored.

Such rules can be divided into **declarative** and **procedural** constraints.

- **Declarative Constraints** include all constraints built within the definition of the table. For example primary key, foreign key…etc.

- **Procedural Constraints** can be categorized into stored procedures and triggers.

Stored procedures and triggers can be simply considered as business logic implemented on the server side, and handled by the DBMS used.

## Entity Integrity & Referential Integrity in Relational Model

Entity integrity constraint state that no primary key value can be null, because the primary key value is used to identify tuples in a relation.

Referential integrity constraint is specified between 2 relations and is used to maintain the consistency among tuples of 2 relations.

To define referential integrity more formally, we first review the concept of a foreign key.

The condition for a foreign key is:

- The attributes in FK have the same domain as the primary key attributes.
- A value of foreign key occurs as the value of PK or is null.

### Update Operations and Dealing with Constraint Violations:

The operations of the relational model can be categorized into retrievals and updates.

There are three basic update operations: (1) insert, (2) modify, (3) delete.

Whenever update operations are applied, the integrity constraints specified should not be violated.

### Insert Operation

The insert operation might violate any of constraints discussed before

- Key constraint can be violated if the key value already exists.
- Entity constraint can be violated if the primary key value is null.
- Referential integrity can be violated if the value of any foreign key doesn't exist in the referenced relation.

If an insertion violates one or more constraint, the default option is to reject the insertion process; in this case, it would be useful if the DBMS could explain to the user why the insertion was rejected. Another option is to attempt to correct the reason for rejecting the insertion. This option is not usually used with insertion.

**Delete Operation**

The delete operation might violate only referential integrity, if the tuple being deleted is referenced by the foreign keys from other tuples in the database.

Three options are available if a deletion operation causes a violation

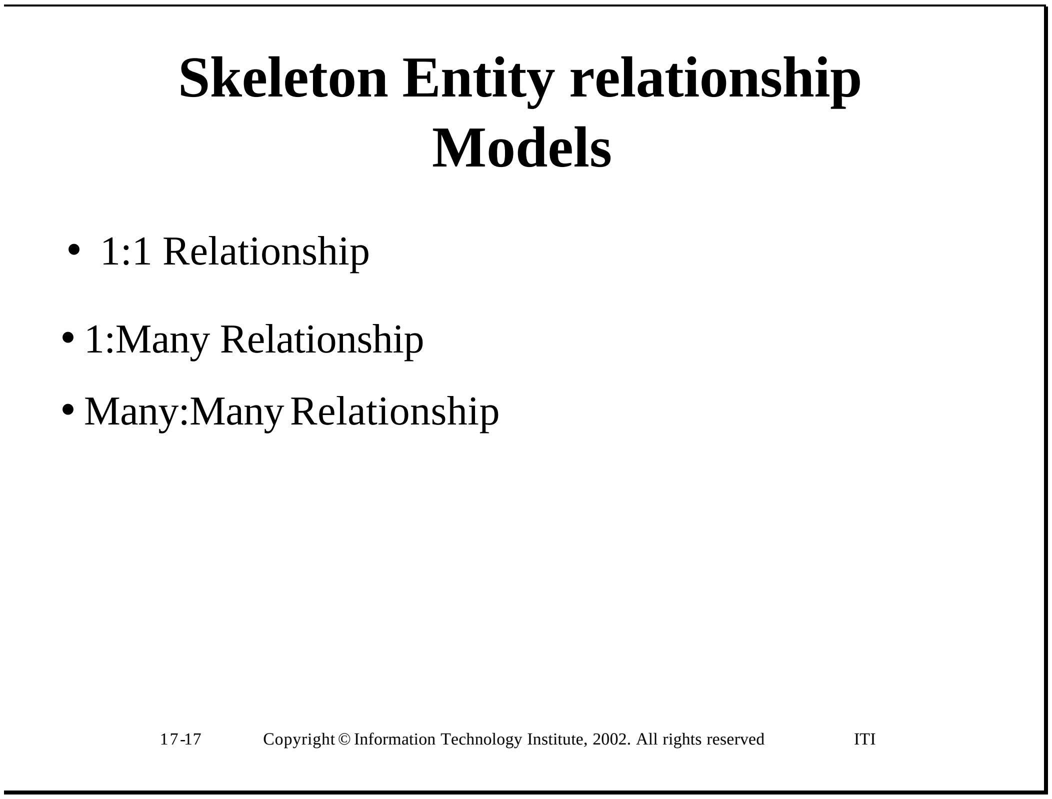
- Rejecting the deletion.
- Attempt to cascade or propagate the deletion by deleting tuples that reference the tuple that is being deleted.
- Modify the referencing attribute values that cause the violation to null or an existing value.

**Update operation**

Updating an attribute that is neither a PK nor a FK usually causes no problem, the DBMS need only to check to confirm the new value is of a correct data type.

Modifying a PK value is similar to deleting one tuple and inserting another in its place, because we use the PK to identify the tuple, the issues discussed earlier under both insert and delete come into play.

If the FK is modified, The DBMS must make sure that the new value refers to an existing tuple in the referenced relation or is null.



## Skeleton Entity relationship Models

- 1:1 Relationship
- 1:Many Relationship
- Many:Many Relationship

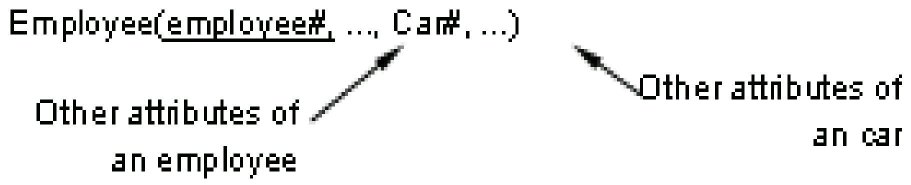
The combination of an ER type diagram and its associated set of fully-normalized table types is called an ER model. It will be convenient to start by building a skeleton ER model step by step.

- **Representation of 1:1 Relationships:**

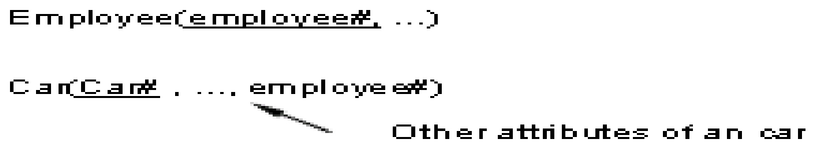
Suppose company cars are assigned to company employees on a 1:1 basis, that is no car is shared between employees, and no employee can use of more than one car. Employees and cars are identified by employee# and car#, respectively. This situation will be modeled by a 1:1 relationship Uses between the entity types Employees and Car.

For a 1:1 relationship the rules of building skeleton table are as follows:

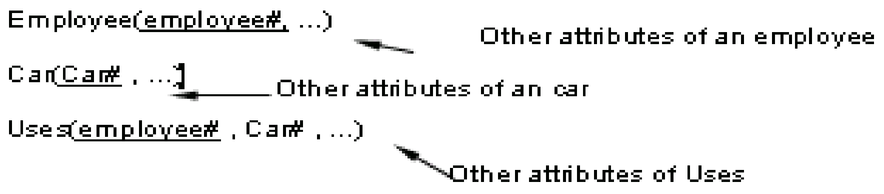
▪ If membership is obligatory for both entity types, put all attributes into a single table type. The identifier of this table is, any of the identifiers of the two entities. Suppose every employee has a company car and every company car is used by an employee.


Employee(employee#, ..., Car#, ...)
Other attributes of an employee
Other attributes of an car

- If membership is obligatory for only one entity types, define two table types, one for each entity. Post the identifier of the non-obligatory entity into the obligatory entity's table type. Suppose every company car is used by an employee, but not every employee has a company car.



- If membership is non-obligatory for both entity types, define three table types, one for each entity and one for the relationship between them. The identifier for the relationship table can be the identifier of either entity tables. Suppose employees do not necessarily have company cars, and cars are not necessarily used by employees.


Employee(employee#, ...)
Other attributes of an employee
Car(Car# , ...)
Other attributes of an car
Uses(employee# , Car# , ...)
Other attributes of Uses

- **Representation of 1: Many Relationships**

Suppose a patient cannot be assigned to more than one hospital ward, while a hospital ward may contain many patients. Hospital wards and patients are identified by ward-name and patient#, respectively. This situation will be modeled by a 1:m relationship Contains between the entity types Ward and Patient.

For a 1:m relationship the rules of building skeleton table are as follows:

- If membership of the many entity type is obligatory, define two table types, one for each entity. Post the identifier of the 1 entity into the many entity's table type. Suppose every Patient must belong to a ward and not every ward contains a patient.

Ward (ward-name ...)

Patient (patient#, ..., ward-name)

Post identifier

- If membership of the many entity type is non-obligatory, define three table types, one for each entity and one for the relationship. The identifier of the relationship table is the identifier of the 1 entity table. Suppose some Patients do not belong to a ward and not every ward contains a patient.

Ward (ward-name ...)

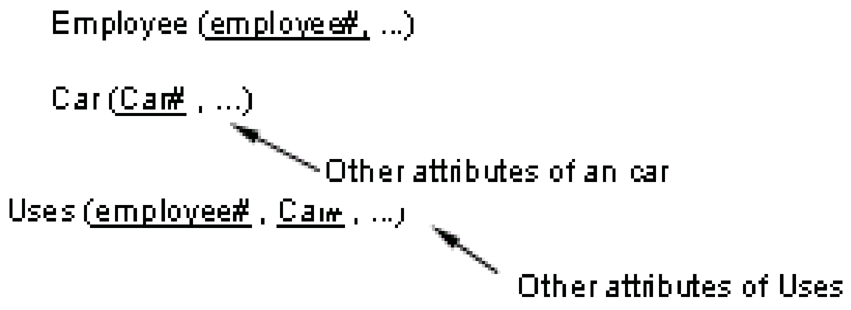
Patient (patient#, ...)

Contains (patient#, ward-name, ...)

## • Representation of Many:Many Relationships

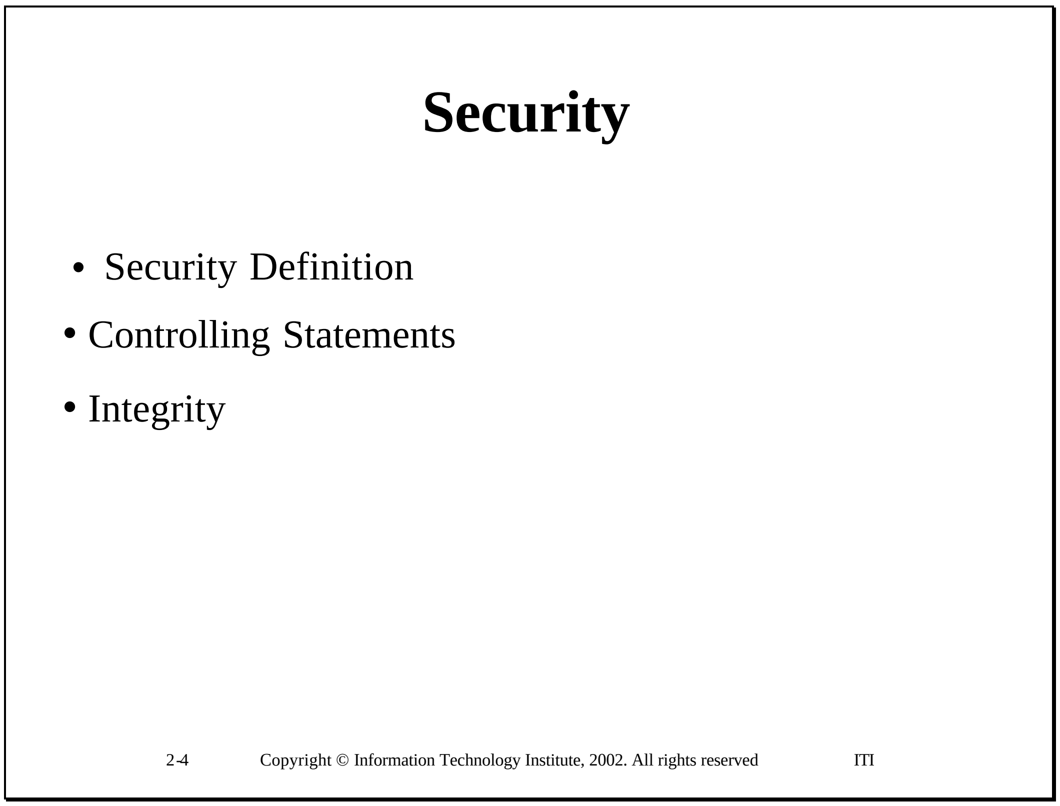
**For a n:m relationship the rules of building skeleton table are as follows:**

▪ Regardless of membership class, define three table types, one for each entity and one for the relationship. The identifier of the relationship table is the combination of both the two entities' identifier. Suppose many employees can use many company cars, and cars are used by many employees.


Employee (employee#, ...)
Car (Car# , ...)
Other attributes of an car
Uses (employee# , Car# , ...)
Other attributes of Uses

# Chapter 4
Security

1-4 ITI



## Security

- Security Definition
- Controlling Statements
- Integrity

2-4 Copyright © Information Technology Institute, 2002. All rights reserved ITI

Security involves ensuring that users are allowed to do the operations they are trying to do.

There are many aspects to the security problem, among them are the following:

- **Legal, social, and ethical aspects** (for example, does the person making the request have a legal right to the requested information?)

- **Physical control** (for example, should the computer or terminal room be locked or otherwise guarded?)

- **Policy questions** (for example, how does the enterprise owning the system decide who should be allowed access to what?)

- **Operational problems** (for example, if a password scheme is used, how are the password themselves kept secret? How often are they changed? )

- **Hardware controls** (for example, does the CPU provide any security features, such as storage protection keys or a privileged operation mode?)

- **Operating system** security (for example, does the underlying operating system erase the contents of storage and data files when they are finished with?)

- **Issues** that are the specific concern of the database system itself (for example, does the database system have a concept of data ownership?)

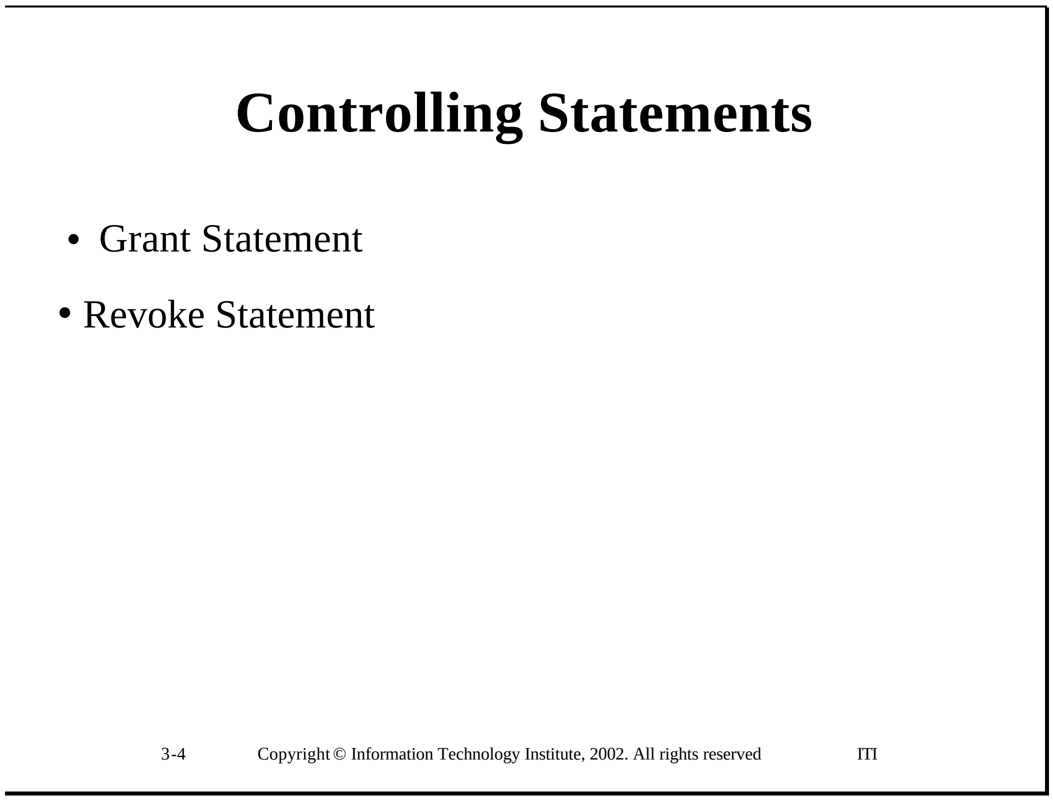
For a “data object”, each user will have different access rights or authorities.

In order for the DBMS to be able to perform this function properly:

- The authorities must be saved in the catalog, in the form of authorization constraints. The authorities must be made known to system by means of the **GRANT** and **REVOKE** statements which will be explained later.

- There must be a means of checking a given access request against the applicable authorization constraints.

- The system must be able to recognize which particular user is initiating a particular request. For that reason, when users logon to the system, they are required to enter, not only their user ID, but also a password.



## Controlling Statements

- Grant Statement
- Revoke Statement

3-4 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### GRANT and REVOKE statements:

GRANT and REVOKE statements represent the principal user interface to what is usually called the authorization subsystem.

In order to be able to perform any operation in SQL, the user must hold the appropriate authority for that operation; otherwise, the operation will be rejected with an appropriate error message or exception code.

Granting rights is done by means of the GRANT statement.

**For example**

GRANT SELECT ON TABLE S TO CHARLEY; this statement will make Charely able to perform select statement on the table S.

GRANT SELECT, UPDATE (STATUS, CITY) ON TABLE S TO JUDY, JACK, JOHN;

GRANT ALL ON TABLE S, P, SP TO FRED, MARY;

GRANT SELECT ON TABLE P TO PUBLIC;

**PUBLIC** is a special keyword, standing for “all user of the system”.

GRANT INDEX ON TABLE S TO PHIL;

In general, the rights that apply to tables or views are as follows:

| | TABLE | INDEX | Comment |
|---|---|---|---|
| SELECT | v | | |
| UPDATE | v | | can be column specific |
| DELETE | v | | |
| INSERT | v | | |
| ALTER | v | | right to execute ALTER TABLE on the table |
| INDEX | | v | right to execute CREATE INDEX on the table |

If user U1 grants some authority to some other user U2, user U1 can subsequently revoke that authority form user U2. Here are some examples:
REVOKE SELECT ON TABLE S FROM CHARLEY ;

REVOKE UPDATE ON TABLE S FROM JOHN ;

REVOKE INSERT, DELETE ON TABLE SP FROM NANCY, JACK;

REVOKE All ON TABLE S, P, SP FROM SAM ;

REVOKE SELECT ON TABLE S FROM CHARLEY ;

REVOKE UPDATE ON TABLE S FROM JOHN ;

REVOKE INSERT, DELETE ON TABLE SP FROM NANCY, JACK;

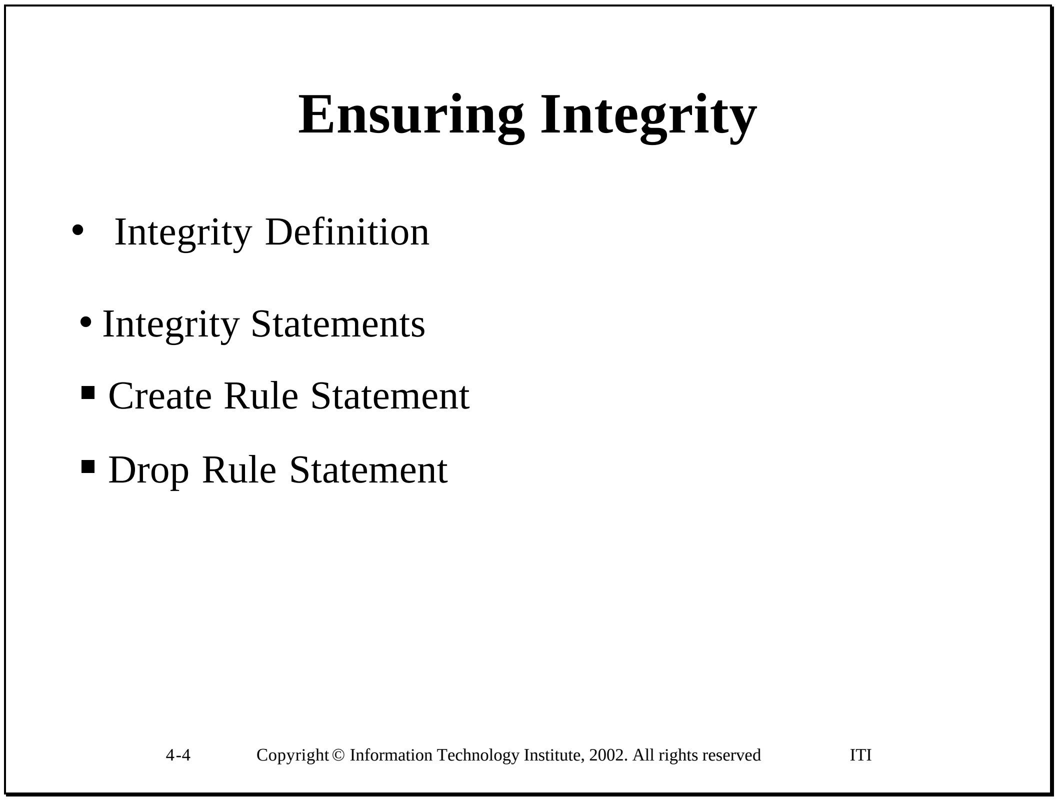
REVOKE All ON TABLE S, P, SP FROM SAM ;

If user U1 has the right to grant some authority A to another user U2, then, user U2 in turn has the right to grant some authority A to another user U3, and so on... Here are some examples:

**User U1**: GRANT SELECT ON TABLE S TO U2 WITH GRANT OPTION ;

**User U2**: GRANT SELECT ON TABLE S TO U3 WITH GRANT OPTION ;
**User U3**: GRANT SELECT ON TABLE S TO U4 WITH GRANT OPTION ;

If user U1 now issues, REVOKE SELECT ON TABLE S TO U2; then the revocation will cascade, that is U2's GRANT to U3 and U3's GRANT to U4 will also be revoked automatically.
These statements will also be re-explained another rime in the part of SQL statements.



## Ensuring Integrity

- Integrity Definition
- Integrity Statements
  - Create Rule Statement
  - Drop Rule Statement

### Integrity

The term “integrity” refers to the accuracy or correctness of the data in the database.

There are two statements used to perform the integrity rules

- Create Integrity Rule
- Drop Integrity Rule

**Create Integrity Rule**

- The following rule, ensure that the Status values must be positive

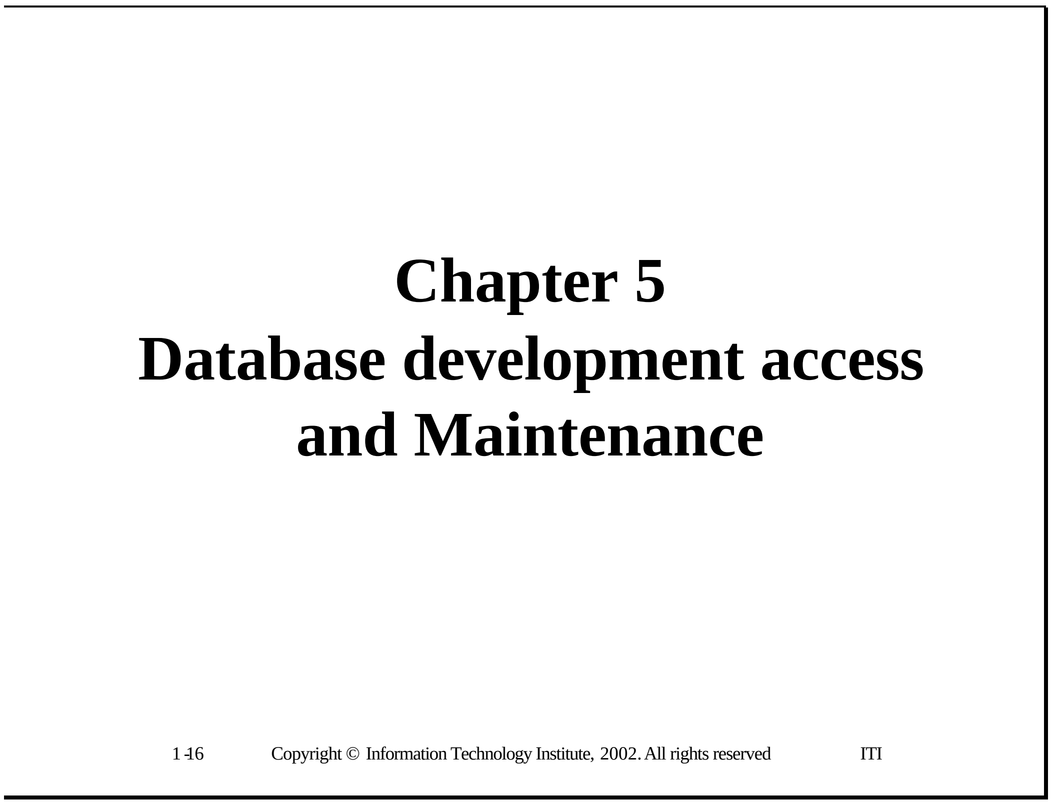
```
CREATE INTEGRITY RULE R1
ON INSERT S.status,
UPDATE S.status:
CHECK FORALL S (S.status>0)
ELSE REJECT;
```

- The following rule ensure that the Status value, must never decrease

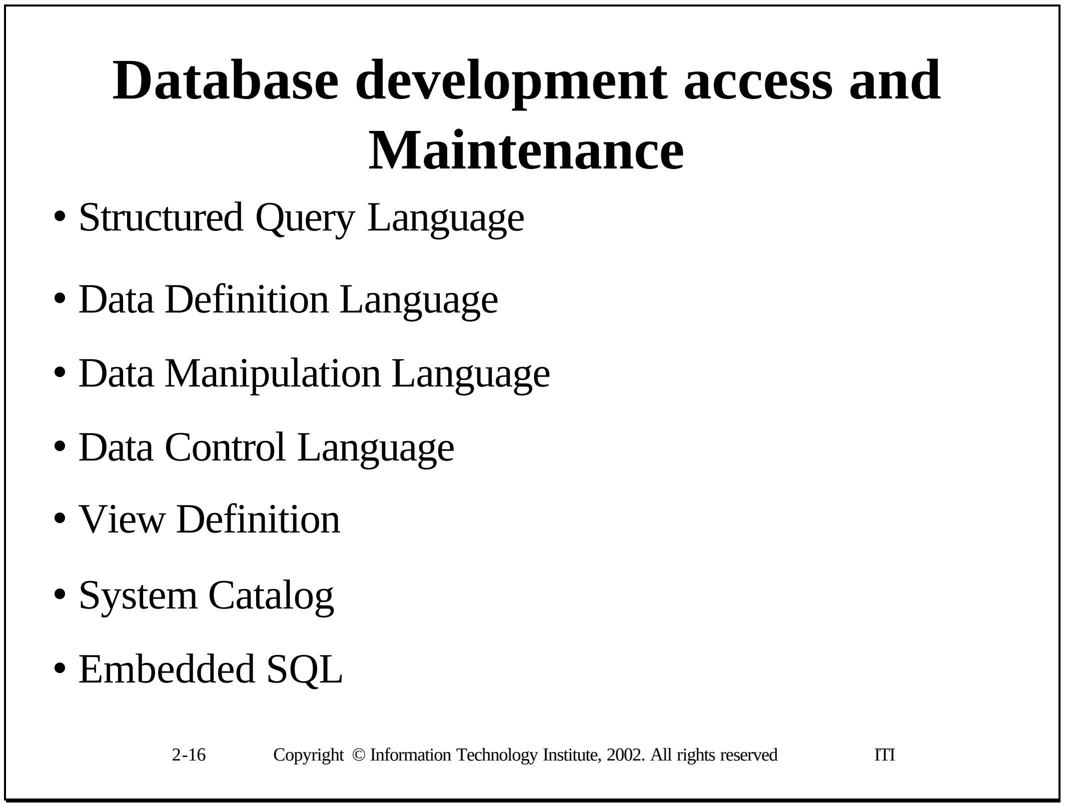
```
CREATE INTEGRITY RULE R1
BEFORE UPDATE OF S.Status FROM NEW_Status

CHECK NEW_Status > S.Status
```

**Drop Integrity Rule**

Drop INTEGRITY RULE R1

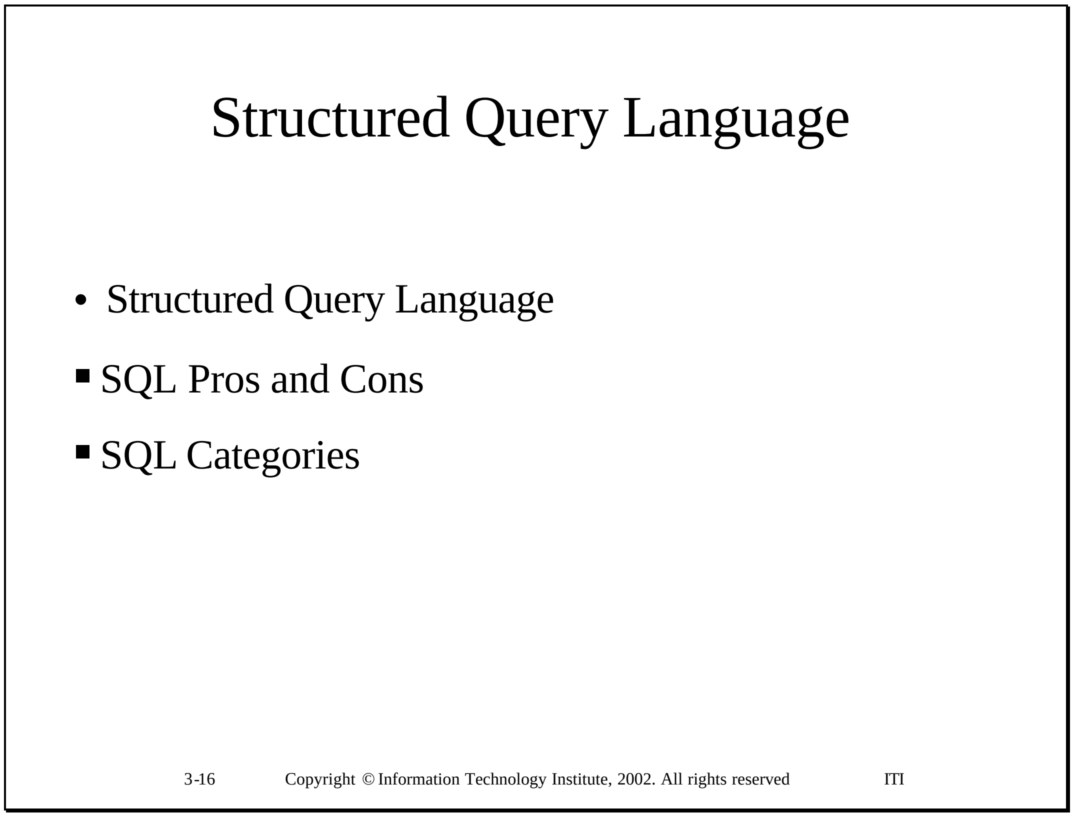


# Chapter 5 Database development access and Maintenance



## Database development access and Maintenance

- Structured Query Language
- Data Definition Language
- Data Manipulation Language
- Data Control Language
- View Definition
- System Catalog
- Embedded SQL



## Structured Query Language

- Structured Query Language
- ▪ SQL Pros and Cons
- ▪ SQL Categories

3-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

SQL was originally spelled SEQUEL. ***SQL*** is an abbreviation of "***Structured Query Language***". It was first defined by Chamberlain and others at the IBM Research Laboratory in San Jose, California. It is both an interactive query language and a database programming language. A prototype of this language was implemented under the name System R.

SQL enables you to select, insert, modify, and delete the information in a database; perform system security functions and set user permissions on tables and databases; handle online transaction processing within an application; create stored procedures which leads to reduction in application coding; and data transfer between different databases.

In 1982, The American National Institute (ANSI) chartered its database committee X32H to develop a proposal for a standard relational language, which was finally ratified by ANSI in 1986. And in 1987, the ANSI standard was also accepted as an international standard by the International Organization for Standardization (ISO).

Other vendors began to develop their own SQL-based products.

**Pros and Cons of having a standard.**

First, we should perhaps briefly consider the question of whether having an SQL standard is beneficial.

**Following are the advantages:**

- Reduced training costs: Application developers can move from one environment to another without the need for expensive retraining.
- Application portability: Applications especially those developed by third-party software vendors-can run unchanged in a variety of different hardware and software environments.
- Applications can be developed on one platform (e.g., a workstation) and then run on another (e.g., a mainframe).
- Application lifetime: Standard languages are assured of a reasonably long lifetime also (other things being equal)
- Intersystem communication: Different systems can communicate more easily With one another.

**There are, however, some disadvantages:**

- A standard can stifle creativity: Implementers might effectively be prevented From providing "the best" solution to some problem because the standard already prescribe some alternative, less satisfactory, solutions to that problem.
- The language is filled with numerous restrictions, and annoying special rules. Regardless of the drawbacks identified above, the situation is that the standard exists, vendors are supporting it, and customers are demanding such support. Hence in this part and the following we will introduce some of the SQL components.

**SQL Language can be categorized into:**

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

**Note** : in the following examples we shall use the S, P, and SP relations that are as follows:

**S**

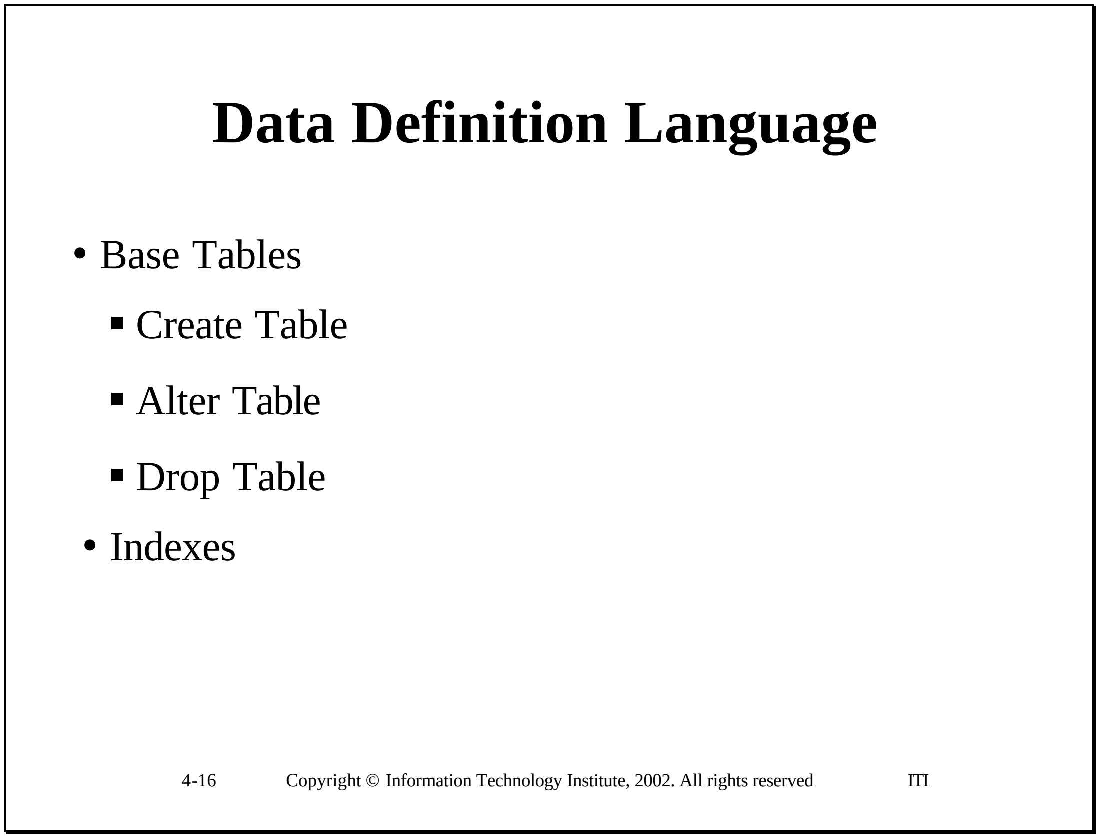
| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

**P**

| P# | PNAME | COLOR | WEIGHT | CITY |
|---|---|---|---|---|
| P1 | Nut | Red | 12 | London |
| P2 | Bolt | Green | 17 | Paris |
| P3 | Screw | Blue | 17 | Rome |
| P4 | Screw | Red | 14 | London |
| P5 | Cam | Blue | 12 | Paris |
| P6 | Cog | Red | 19 | London |

**SP**

| S# | P# | QTY |
|---|---|---|
| S1 | P1 | 300 |
| S1 | P2 | 200 |
| S1 | P3 | 400 |
| S1 | P4 | 200 |
| S1 | P5 | 100 |
| S1 | P6 | 100 |
| S2 | P1 | 300 |
| S2 | P2 | 400 |
| S3 | P2 | 200 |
| S4 | P2 | 200 |
| S4 | P4 | 300 |
| S4 | P5 | 400 |



## Data Definition Language

- Base Tables
  - ▪ Create Table
  - ▪ Alter Table
  - ▪ Drop Table
- Indexes

4-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

DDL statements are used to define and manage the objects in the database.

The principal DDL statements are as follows:

*CREATE TABLE* *CREATE INDEX* *CREATE VIEW*

*ALTER TABLE*

*DROP TABLE* *DROP INDEX* *DROP VIEW*

We defer all view commands after the DML explanation.

### • Base Tables

The base tables are the relations, while the view tables are the views of the users to some relations.

#### ▪ CREATE TABLE:

The general format of that statement is as follows:

***CREATE TABLE*** *base-table-name (column-definition[, column-definition]...*

*[, primary-key-definition] [, foreign-key-definition [, foreign-key-definition ] ... ] );*

Where a "column-definition" in turn, takes the form:

*column-name data-type[Not NULL]*

**A. Data Types**

SQL supports the followings data types:

• **Numeric data:**

| | |
|---|---|
| INTEGER | Signed full word binary integer. |
| SMALLINT | Signed half word binary integer. |
| DECIMAL(*p[,q]*) | Signed packed decimal number, *p* digits and sign, with assumed decimal point q digits from the right. |
| FLOAT(*p*) | Signed floating point number of *p* binary digits precision. |

• **String data**

| | |
|---|---|
| CHARACTER(n) | Fixed length string of exactly n 8-bit characters. |
| VARCHAR(n) | Varying length string of up to n 8-bit characters. |
| GRAPHIC(n) | Fixed length string of exactly n 16-bit characters. |
| VARGRAPHIC(n) | Varying length string of up to n 16-bit characters. |

• **Date/item data:**

| | |
|---|---|
| DATE | date (yyyymmdd) |
| TIME | time (hhmmss) |
| TIMESTAMP | "timestamp" (combination of date and time, accurate to the nearest microsecond) |

Various abbreviations and alternative appalling (e.g., CHAR for CHARACTER, DEC for DECIMAL, INT for INTEGER).

- **NULL VALUES**

  SQL supports the concept of a null data value. Any column can contain null values unless those that have the definition at CREATE TABLE, explicitly specifies NOT NULL. NULL is a special value that is used to represent "value unknown" or " value inapplicable".

**Example:** use the following statement to create a table S, that has the attributes: (S#, SNAME, STATUS, CITY)

```
CREATE TABLE S (S#     CHAR(5)  NOT NULL,
                SNAME   CHAR(20) NOT NULL,
 STATUS   SMALL INT,
CITY        CHAR(15)
PRIMARY KEY (S#) );
```

- **Alter Table**

  ALTER TABLE is used at any time to add a new column at the right of an existing base table. The general format of that statement is as follows:

  **ALTER TABLE** base-table-name **ADD** column-name data-type;

  **Example:**

  ALTER TABLE S ADD DISCOUNT SMALLINT;

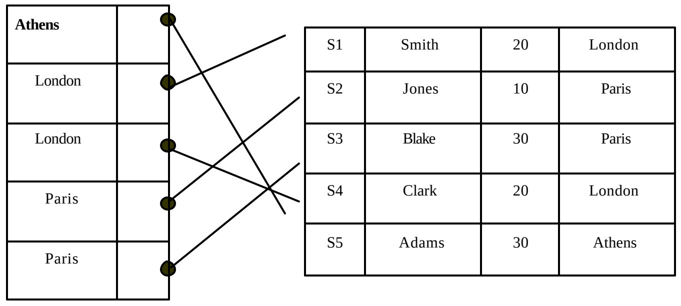
- **Drop Table**

  It is also possible to destroy an existing base table at any time. The general format of that statement is as follows:

  *DROP TABLE base-table-name;*

  The specified base table is removed from the system. All indexes and views defined on that base table are automatically dropped also.

## Index



|  |  |  |  |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

file (index) Supplier file (data)

## Indexes

Index is used to accelerate the process of searching the database.

There are two functions used for INDEXES: *CREATE INDEX* and *DROP INDEX*.

In order to understand the index, consider the above figure.

Assume that the query "find all suppliers in city C" (where C is a parameter) is an important function which is frequently executed and which is therefore required to perform efficiently and quickly. Given such requirement, the DBA might choose the stored representation shown in the above figure. In that representation, there are two stored files, a supplier file and a city file; the city file, which is stored in city sequence (because city is the primary key), includes pointers (RIDs) into the supplier file. To find all suppliers in London (say), the DBMS now has two possible strategies:

- Search the entire supplier file, looking for all records with city value equal to London.

- Search the city file for the London entries, and for each such entry follow the pointer to the corresponding record in the supplier file.

If the ratio of London suppliers to others is small, the second of these strategies is likely to be more efficient than the first, because of the following: the DBMS is aware of the physical sequencing of the city file (it can stop its search of that file as soon as it finds a city that comes later than London in alphabetic order), and even if it did have to search the entire city file, that search would still probably require fewer I/O's overall, because the city file is physically smaller than the supplier file (because the records are smaller).

In this example, the city file is the **index for** the supplier file; equivalently, the supplier file is indexed by the city file.

An index is a special kind of stored file. In which each entry (i.e., record) consists of precisely two values, a data value and a pointer (RID);the data value is a value for a certain field of the indexed file, and the pointer identifies a record of that file that has that value for that field. The rele vant field of the indexed file is called the indexed field, or sometimes the index key.

The fundamental advantage of an index is that it speeds up retrieval. However, the disadvantage is that it slows down updates.

For example, every time a new stored record is added to the indexed file, a new entry will also have to be added to the index.

▪ **Creating Index**

***CREATE INDEX*** has the following format:

***CREATE [UNIQUE] INDEX*** *index-name*
***ON*** *base-table-name (Column-name [order][, Column-name [order]...)*
**[CLUSTER];**

The term ***CLUSTER*** means that the logically related records are physically closed together on the disk. The optional ***CLUSTER*** specification means that this is a clustering index. A given base table can have at most one ***CLUSTER*** index.

The optional *order* may be ***ASC*** (ascending) or ***DESC*** (descending). The default value is ***ASC***.

**Example** :
Create an index called X on table T in which entries in that index are ordered by ascending R-value within descending Q-value within ascending P-value.

*CREATE INDEX 'X' ON T (P, Q DESC,R);*

The ***UNIQUE*** option in ***CREATE INDEX*** specifies that no two records in the indexed base table will be allowed to take on the same value for the indexed field.

**Example:**

to create an index called XS on table S which is a unique index.

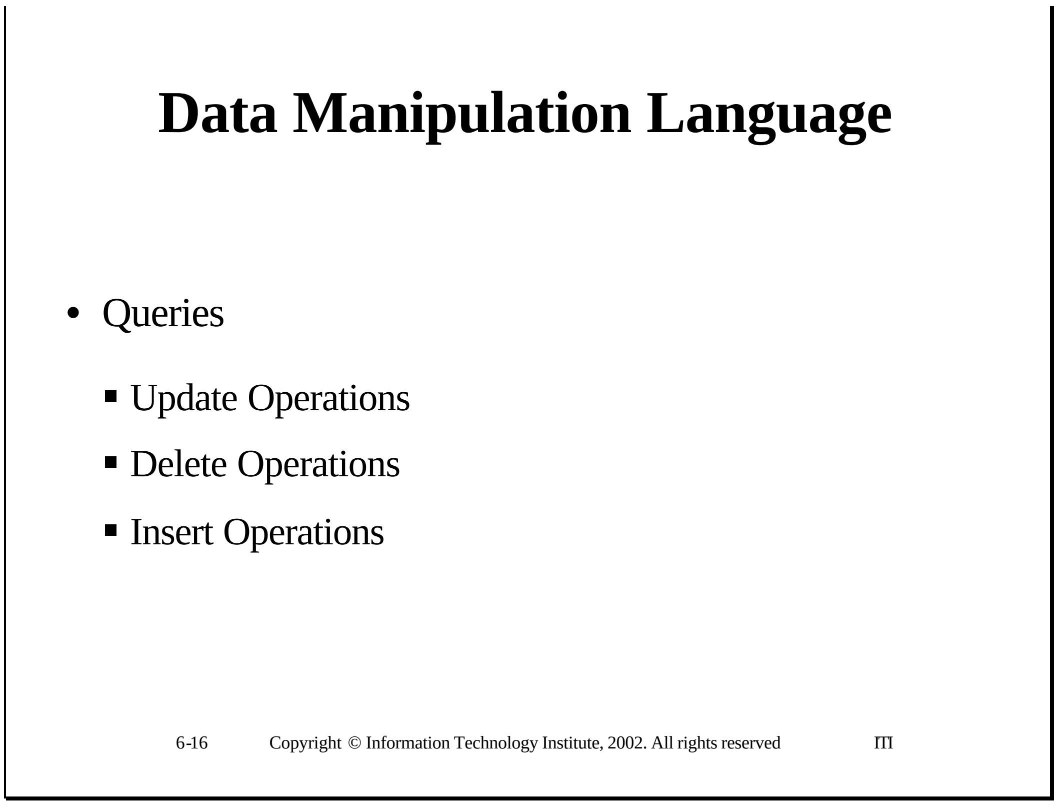
*CREATE UNIQUE INDEX XS ON S (S#);*

For the following example: ***CREATE INDEX*** *XS* ***ON*** *S (CITY);*

***UNIQUE*** has not been specified in this case because multiple suppliers can be located in the same city.

- **The statement to drop an index is:**

  ***DROP INDEX*** *index-name*

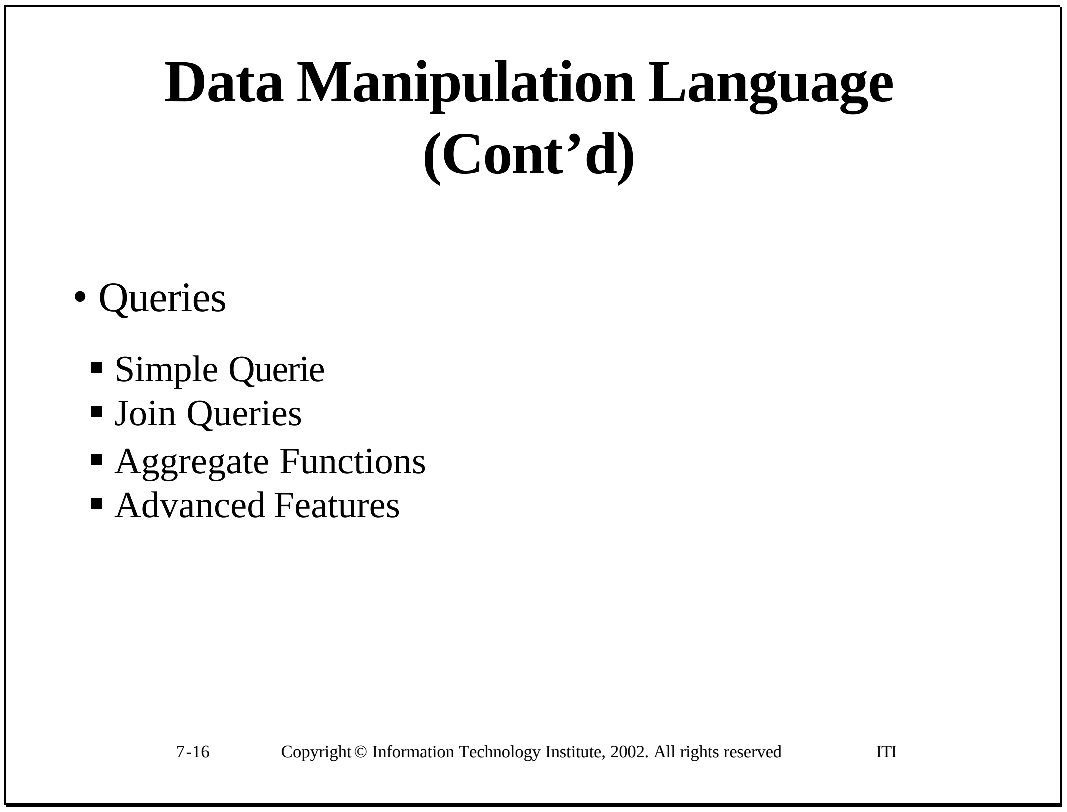


## Data Manipulation Language

- Queries
  - Update Operations
  - Delete Operations
  - Insert Operations

6-16 ITI

SQL provides four Data Manipulation Language (DML) statements SELECT, UPDATE, DELETE, and INSERT.



## Data Manipulation Language (Cont’d)

- Queries
  - Simple Querie
  - Join Queries
  - Aggregate Functions
  - Advanced Features

7-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Queries

**Simple Queries**

- Get supplier numbers and status for suppliers in Paris

```
 SELECT   S#, STATUS
FORM     S
WHERE    CITY = 'Paris';
```

The general form of the SELECT statement is as follows:

```
SELECT [DISTINCT] field(s)
FROM table(s)
[WHERE predicate]
[GROUP BY field(s)]
[HAVING predicate]
[ORDER BY field(s)];
```

Some examples are illustrated as follows:

- **Simple retrieval:**

♦ Get part numbers for all parts supplied

```
SELECT  P#
FROM    S
```

In this form, the **duplication** is allowed.

- Get part numbers for all parts supplied and eliminate **duplicates** from the result

  **Note** : If the data stored contains duplicate rows, the duplication will not be eliminated unless you explicitly request it to do via the keyword "DISTINCT" as in the example.

```
SELECT DISTINCT P#
FROM   SP;
```

- Get full details of all suppliers

```
SELECT *
FROM   S;
```

- For all Parts, get the number and the weight of the parts in grams. (Part weights are given in table P in pounds)

```
 Select P.P#, 'Weight in grams=', P.WEIGHT*454

  From P;
```

- **Table Aliasing:** in the following example, we create alias 's1' for the table 'skill'

```
Select s1.skill, s1.description
FROM skill s1
Where s1.Skill <> 'programming'
```

- **Quantified retrieval:**

♦ Get supplier number for suppliers in Paris with status > 20

```
    SELECT  S#
     FORM    S
```

```
WHERE    CITY = ' Paris'
AND        STATUS > 20;
```

The condition (Predicate following WHERE) may be: =, ~ =, >, ~ >, > =, <, ~ <, and <=, where ~ represents not. These operations are used for comparison. The Boolean operators AND, OR and NOT.

♦ ORDERING: "Get supplier numbers and status for supplier in Paris, in descending order of status"

```
SELECT S#, STATUS
FROM    S
WHERE  CITY = 'Paris'
ORDER  BY STATUS DESC;
```

Ordering may be specified in the same manner as in CREATE INDEX

- **Join Queries**

  Join Queries is the ability to retrieve data from more than one table, it is one of the most powerful features of relational systems.

  The following examples illustrate different types of joins.

  - **Simple Equijoin**

| S# | SNAME | CITY |
|---|---|---|
| S1 | Smith | London |
| S2 | Jones | Paris |
| S3 | Blake | Athens |

| P# | Pname | CITY |
|---|---|---|
| P1 | Nut | London |
| P2 | Bolt | Rome |
| P3 | Screw | Rome |

To get all the information about the supplier and the product sharing the same city, we will use the following statement:

```
SELECT Supplier.*, Product.* FROM Supplier, Product
WHERE Supplier.CITY = Product.CITY;
```

**The result will be:**

| S# | SNAME | CITY | P# | Pname | CITY |
|---|---|---|---|---|---|
| S1 | Smith | London | P1 | Nut | London |

- **Greater-than join**

  Get all combinations of supplier and products information such that the supplier city follows the product city in alphabetical order.

  SELECT Supplier.*, Product.* FROM Supplier, Product

  WHERE Supplier.CITY > Product.CITY;

**The result will be:**

| S# | SNAME | CITY | P# | Pname | CITY |
|---|---|---|---|---|---|
| S2 | Jones | Paris | P1 | Nut | London |
| S2 | Jones | Paris | P2 | Bolt | Rome |
| S2 | Jones | Paris | P3 | Screw | Rome |

- **Self Join**

Get all pairs of products such that they have same city

SELECT First.P#, Second.P#

FROM Product first, Product second

Where First.CITY = Second.city

AND First.P# < Second.P#;

**The result will be:**

| P# | P# |
|---|---|
| P2 | P3 |

This query involves a join of a table Product with itself, as explained in the following.

Assume that we had two separate copies of table Product, the “first“ copy and the “second” copy, then the logic of the query is as follows:
We need to be able to examine all possible pairs of products, one from the first copy of the Product and one from the second, and to retrieve the two product numbers from such a pair of rows if and only if the city values are equal.

We therefore, need to be able to reference two product rows at the same time, in order to distinguish between the two references, we introduce arbitrary range variables “first” and “second”, each of which “ranges over” table Product.

**Note** : The purpose of the condition first.P# < second.P# is to eliminate pairs of product numbers of the form ( x,x) and to guarantee that the pairs (x,y) and (y,x) will not both appear.

▪ **Join of more than one table**

In this type we are going to collect data from more than two tables.

**Examples**

♦ Get all combinations of the supplier and the part tables.

```
SELECT S.*, P.*
FROM   S, P
WHERE S.CITY = P.CITY;
```

This is an EQUIJOIN operation.

**Result:**

| S# | SNAME | STATUS | S.CITY | P# | PNAME | COLOR | WEIGHT | P.CITY |
|---|---|---|---|---|---|---|---|---|
| S1 | Smith | 20 | London | P1 | Nut | Red | 12 | London |
| S1 | Smith | 20 | London | P4 | Screw | Red | 14 | London |
| S1 | Smith | 20 | London | P6 | Cog | Red | 19 | London |
| S2 | Jones | 10 | Paris | P2 | Bolt | Green | 17 | Paris |
| S2 | Jones | 10 | Paris | P5 | Cam | Blue | 12 | Paris |
| S3 | Blake | 30 | Paris | P2 | Bolt | Green | 17 | Paris |
| S3 | Blake | 30 | Paris | P5 | Cam | Blue | 12 | Paris |
| S4 | Clark | 20 | London | P1 | Nut | Red | 12 | London |
| S4 | Clark | 20 | London | P4 | Screw | Red | 14 | London |
| S4 | Clark | 20 | London | P6 | Cog | Red | 19 | London |

In this operation the two columns S.CITY and P.CITY are in the result table and they are equal.

Assume that the relation S has the following attributes S#, SNAME, STATUS, CITY, and P has the following attributes P#, PNAME, COLOR, WEIGHT, CITY.

To eliminate one of the CITY columns, the SELECT statement is written as:

```
SELECT S.*, P.P#, P.PNAME, P.COLOR, P.WEIGHT
FROM    S, P
WHERE S.CITY = P.CITY;
```

♦ Get all combinations of the supplier and part information such that the supplier city follows the part city in alphabetical order

```
SELECT S.*, P.*
FROM S, P
WHERE S.CITY > P.CITY;
```

**Result :**

| S# | SNAME | STATUS | S.CITY | P# | PNAME | COLOR | WEIGHT | P.CITY |
|---|---|---|---|---|---|---|---|---|
| S2 | Jones | 10 | Paris | P1 | Nut | Red | 12 | London |
| S2 | Jones | 10 | Paris | P4 | Screw | Red | 14 | London |
| S2 | Jones | 10 | Paris | P6 | Cog | Red | 19 | London |
| S3 | Blake | 30 | Paris | P1 | Nut | Red | 12 | London |
| S3 | Blake | 30 | Paris | P4 | Screw | Red | 14 | London |
| S3 | Blake | 30 | Paris | P6 | Cog | Red | 19 | London |

♦ Get all combinations of the supplier information and part information where the supplier and part concerned are calculated, but omitting supplier with status 20

```
SELECT S.*, P.*
FROM  S, P
WHERE S.CITY = P.CITY
AND S.STATUS ~ = 20;
```

**Result :**

| S# | SNAME | STATUS | S.CITY | P# | PNAME | COLOR | WEIGHT | P.CITY |
|---|---|---|---|---|---|---|---|---|
| S2 | Jones | 10 | Paris | P2 | Bolt | Green | 17 | Paris |
| S2 | Jones | 10 | Paris | P5 | Cam | Blue | 12 | Paris |
| S3 | Blake | 30 | Paris | P2 | Bolt | Green | 17 | Paris |
| S3 | Blake | 30 | Paris | P5 | Cam | Blue | 12 | Paris |

- Get all pairs of cities names such that a supplier located in the first city supplies a part stored in the second city;

```
SELECT  DISTINCT   S.CITY, P.CITY
FROM    S, SP, P
WHERE  S.S# = SP.S#
AND       SP.P# = P.P#;
```

- **Aggregate Functions**

The select statement as described so far is still inadequate for many practical problems.

For example, even a query as simple as "How many suppliers are there?" Can not be expressed using only the constructs introduced until now.

SQL therefore provides a number of special aggregate functions to enhance its

basic retrieval power.

Aggregate functions include:

| | |
|---|---|
| COUNT | Count the number of values in a column. |
| SUM | Compute the sum of the values in a column. |
| AVG | Average of the values in a column. |
| MAX | Get the maximum value in a column. |
| MIN | Get the minimum value in a column. |

Here, are some examples about the aggregate function:

- Get the total number of suppliers.

```
SELECT COUNT ( * )
FROM   S;
```

**Result:**

5

- Get the total number of suppliers who are currently supplying parts.

```
SELECT COUNT (DISTINCT S#)
FROM   SP;
```

**Result :**

4

**GROUP BY:**

It is used to logically rearrange the table represented by the FROM clause into partitions or groups. Get the part number and the total shipment quantity for part.

```
▪     SELECT      P#, SUM (QTY)
 FROM         SP
 GROUP BY    P#;
```

**HAVING:**

The HAVING clause sets conditions on the GROUP BY clause similar to the way WHERE interacts with SELECT.

▪ Get part numbers for all parts supplied by more than one supplier.

```
SELECT       P#
FROM         SP
GROUP BY     P#
HAVING       COUNT (*) > 1;
```

- **Advanced Features**

▪ **Retrieval using LIKE:**

In general, a 'LIKE predicate' takes the form:

Column-name LIKE character-string-constant of the type( CHAR or VARCHAR).

The ' – ' character stands for any single character.

The ' * ' character stands for any sequence of n characters, where n may be zero.

All the other characters simply stand for themselves.

♦ Get the parts whose names begin with the letter C.

```
SELECT       P.*
FROM         P
WHERE        P.NAME   LIKE 'C%';
```

▪ **Retrieval Involving Subquery**

A subquery is a SELECT statement nested inside a SELECT, INSERT, UPDATE or DELETE statement or inside another query.

Use subqueries to break down a complex query into a series of logical steps and, to solve a problem with a single statement.

Subqueries are useful when your query relies on the results of another query.

- Get supplier names for suppliers who supply part P2.

```
SELECT      SNAME
FROM        S
WHERE       S# IN (SELECT   S# FROM   SP  WHERE       P# = ‘P2’)
```

- Get supplier names for suppliers who supply at least one red part.

```
SELECT   SNAME
FROM     S
WHERE    S# IN (SELECT   S#  FROM    SP WHERE P#   IN (SELECT  P#
 FROM    P  WHERE   COLOR = ‘Red’));
```

- Get supplier numbers for suppliers who are located in the same city as supplier S !

```
SELECT       S#
FROM         S
WHERE        CITY = (SELECT   CITY
                     FROM    S
                     WHERE  S# = ‘S2’);
```

- Get supplier numbers for suppliers with status value less than the current maximum status value in the S table.

```
SELECT       S#
FROM         S
WHERE        STATUS < (SELECT MAX (STATUS)
                       FROM    S);
```

- Get supplier names for suppliers who supply part P2.

```
SELECT       SNAME
FROM         S
WHERE        EXISTS  (SELECT  *
              FROM    SP
```

```
                WHERE   S# = S.S#
                AND     P# = 'P2') ;
```

♦ Get supplier names for suppliers who do not supply part P2.

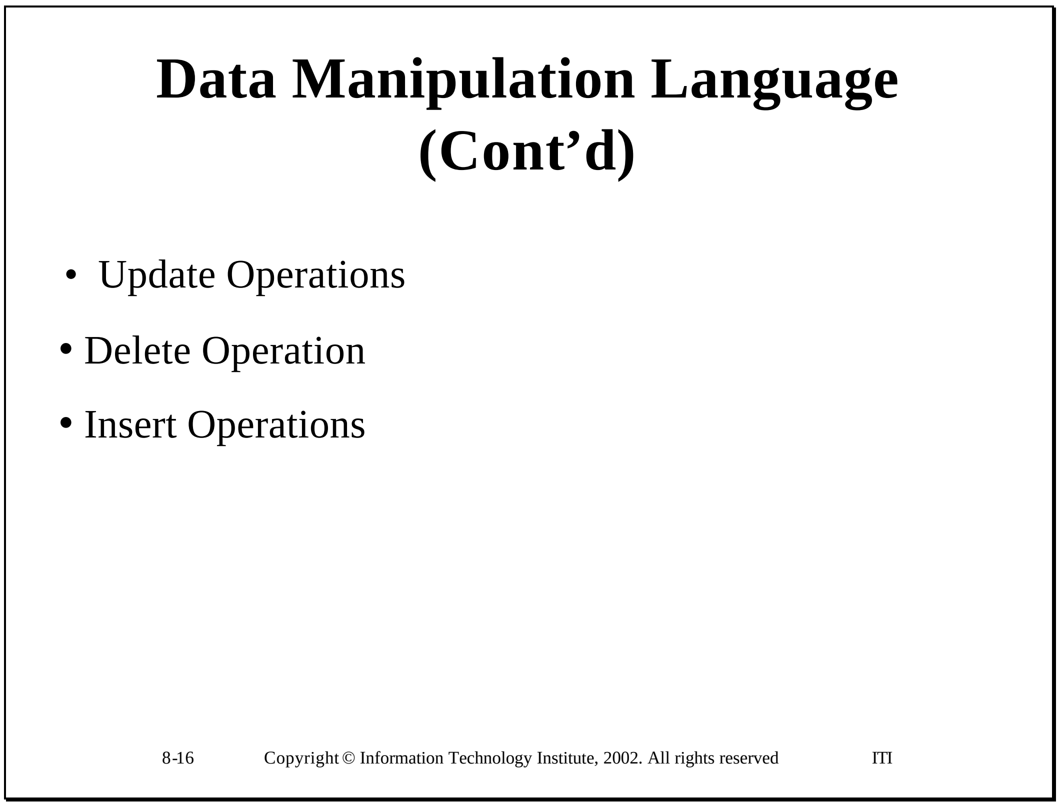
```
SELECT      SNAME
FROM        S
WHERE       NOT EXISTS  (SELECT  *
                         FROM    SP
                         WHERE   S# = S.S#
                         AND     P# = 'P2') ;
```

- **Retrieval Involving UNION:**

Combines the results of two or more queries into a single result set consisting of all the rows belonging to all queries in the union, taking into consideration, that the **number of selected columns** and its **data types** must be the same in all the queries in the union

▪ Get part number of parts that either weight more than 16 pounds or supplied by supplier S2 or both.

```
SELECT      P#
FROM        P
WHERE       WEIGHT > 16
UNION
SELECT      P#
FROM        SP
WHERE       S# = 'S2';
```



## Data Manipulation Language (Cont'd)

- Update Operations
- Delete Operation
- Insert Operations

8-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Update Operations

The update statement has the following format:

```
UPDATE table
    SET       field = expression [,field = expression] …
    [WHERE  predicate];
```

**Examples:**

- Change the color of part P2 to yellow, increase its weight by 5, and set its city to NULL.

```
UPDATE        P
SET             COLOR    = 'Yellow'
              WEIGHT     = WEIGHT + 5
              CITY       = NULL
WHERE          P# = 'P2';
```

- Change the shipment quantity to zero for all suppliers in London.

```
UPDATE        SP
SET           QTY = 0
WHERE         'London' = (SELECT CITY
                          FROM   S
                          WHERE S.S# = SP.S#);
```

- Change the supplier number for supplier S2 to S9.

  ```
  UPDATE        S
  SET           S# = 'S9'
  WHERE         S# = 'S2';
  ```

**Delete Operation**

The delete statement has the following format:

```
DELETE
FROM    table
[WHERE  predicate];
```

All records in "table" that satisfy "predicate" are deleted.

**Examples:**

- Delete supplier S1.

  ```
   DELETE
  FROM    S
  WHERE   S# = 'S1';
  ```

- Delete all suppliers in Madrid.

  ```
  DELETE
  FROM    S
  WHERE   CITY = 'Madrid';
  ```

- Delete all shipments for suppliers in London.

  ```
  DELETE
  FROM    SP
  WHERE   'London'= (SELECT    CITY
  ```

```
                    FROM     S
                    WHERE    S.S# = SP.S#)
```

**Insert Operation**

The insert statement has the following formats:

```
INSERT
INTO        table [(field [,field] ...)]
VALUES      (constant [, constant], ...)];
```

or

```
INSERT
INTO        table [( field [,field] ...)]
SELECT ...  FROM ... WHERE ... ;
```

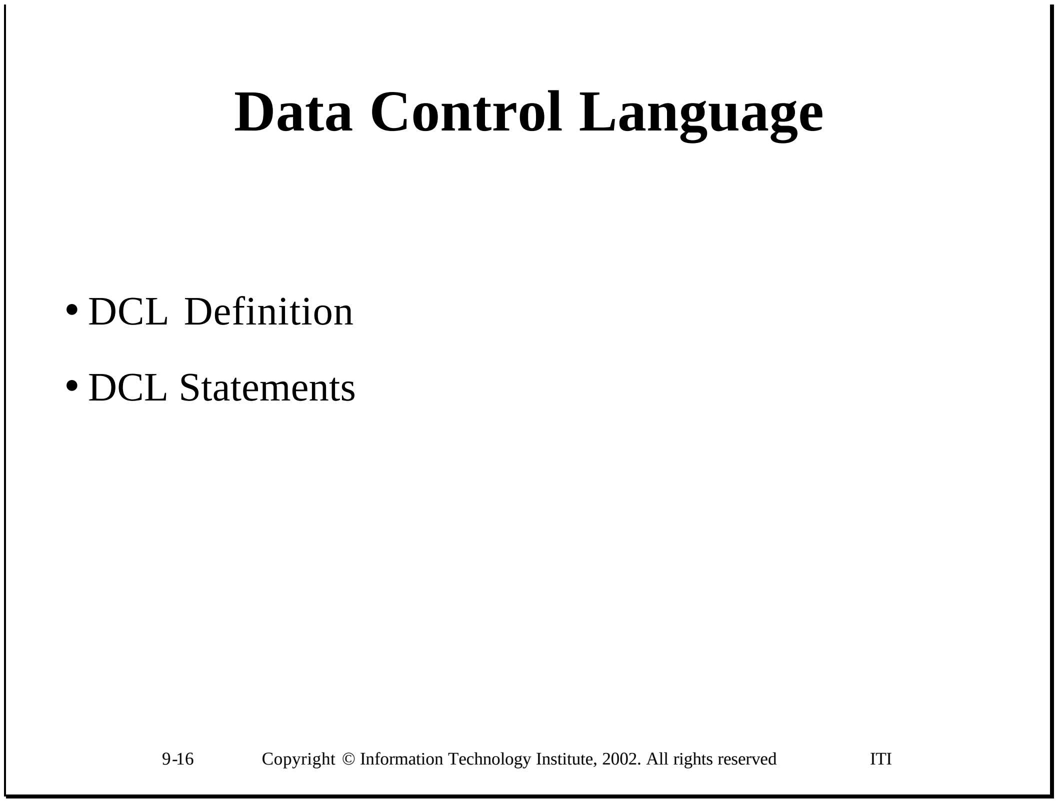
**Examples:**

- Add part P7 on table P that has Athens as its city, and 24 for the weight. The name and color at present are unknown.

```
INSERT
INTO       P (P#, CITY, WEIGHT)
VALUES   ('P7', 'Athens', 24);
```

- INSERT into table the result from another SELECT statement

```
INSERT INTO worker (name, age, userName)
SELECT name, SYSDATE-birthDate, 'OPS$'||name
 FROM candidate    WHERE evaluation > 78;
```



## Data Control Language

- DCL Definition
- DCL Statements

9-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Data Control Language (DCL)

Data control language is used to authorize who can retrieve or modify data for security reasons. Security, in this case, refers to the protection of data against unauthorized access. This function is performed by the SQL statements **GRANT** and **REVOKE.**

**Examples:**

- Grant the right of selection on the 'employees' table to the user 'Mary'
  GRANT SELECT ON TABLE employees TO Mary;

- Grant all rights to Fred and Mary.
  GRANT ALL ON TABLE customer TO Fred, Mary;

If user U1 grants some authority to another user U2, U1 can subsequently **revoke** that authority from U2.

Revoking authority is done by means of the **REVOKE** statement.

**Examples:**

- Prevent the user ‘Charley’ from selecting the ‘Customer’ table.
  REVOKE SELECT ON TABLE customer FROM Charley;

- REVOKE UPDATE ON TABLE customer FROM John;

- REVOKE SELECT ON TABLE customer FROM Charley;

- REVOKE ALL ON TABLE customer FROM Sam;

If user U1 has the right to grant some authority A to another user U2, then U1 also has the right to grant the same authority to user U2 “with the GRANT option”. Passing the GRANT option along from U1 to U2 in this way means that U2 in turn gets the right to grant the authority to some third user U3, and also gets the right to pass the GRANT option along to U3.

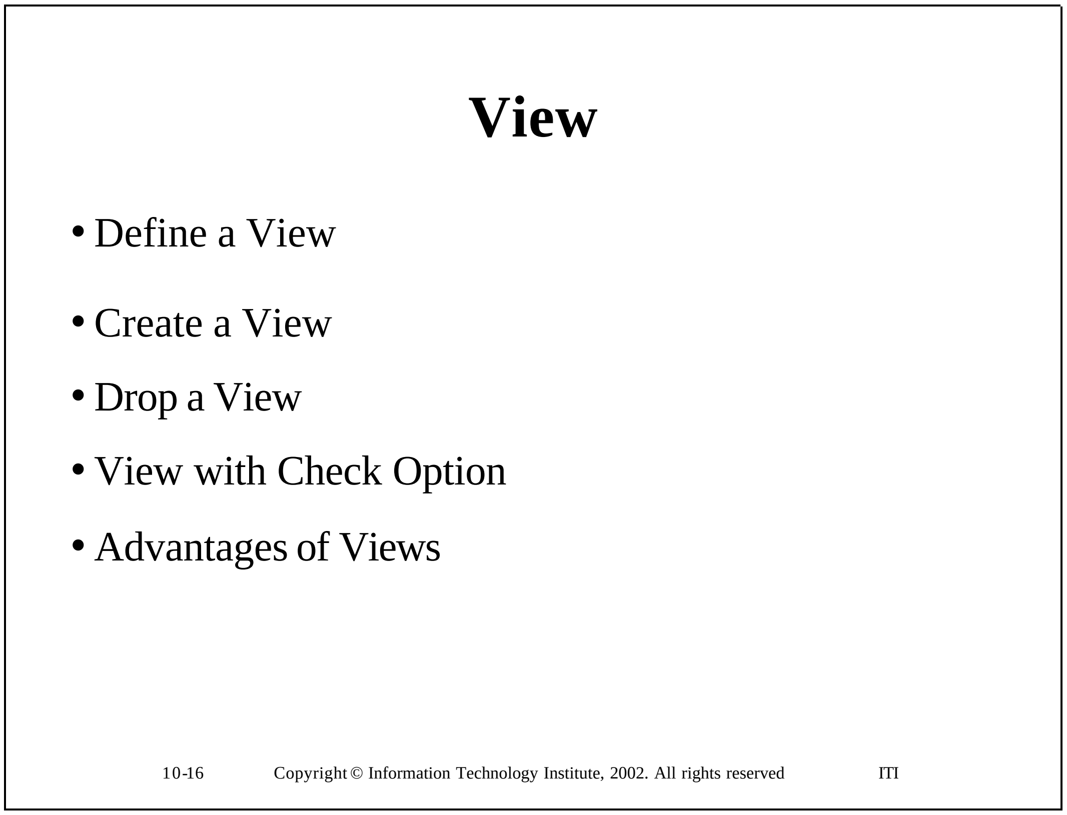
**Examples:**

User U1 says:
GRANT SELECT ON TABLE S TO U2 WITH GRANT OPTION;

User U2 says:
GRANT SELECT ON TABLE S TO U3 WITH GRANT OPTION;

User U3 says:
GRANT SELECT ON TABLE S TO U4 WITH GRANT OPTION;

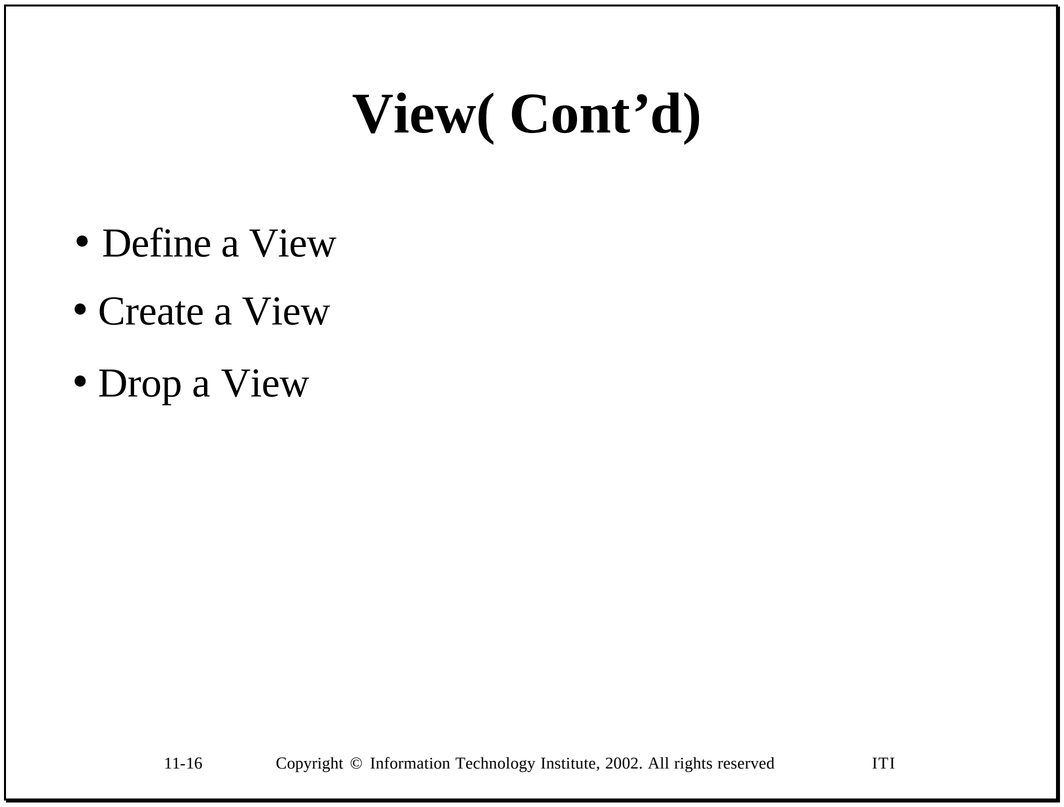
If user U1 now issues:
REVOKE SELECT ON TABLE S FROM U2;
Then the revoking effect will cascade. That is, U2’s GRANT to U3 and U3’s GRANT to U4 will also be automatically revoked.



## View

- Define a View
- Create a View
- Drop a View
- View with Check Option
- Advantages of Views

10-16 ITI



## View( Cont'd)

- Define a View
- Create a View
- Drop a View

11-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Definition

A view is a virtual table that presents the data from one or more tables to the user in an alternative way. The view does not actually exist, but appears to the user as if it does.

### CREATE VIEW

The general format of that operation is as follows:

```
CREATE VIEW view-name [(column-name [, column-name]...)]
AS subquery;
```

**Examples:**

- Create view 'REDPARTS' that has the following fields: part #, part name, weight and city.

```
CREATE VIEW REDPARTS ( P#, PNAME, WT, CITY)
 AS     SELECT P#, PNAME, WEIGHT, CITY)
      FROM   P
      WHERE  COLOR = 'Red' ;
```

**Note** : When the above "create view" is executed, the subquery following the AS is not executed; instead, it is simply saved in the catalog. However, to the user, it appears as if there was a real table in the database called REDPARTS.

Views can simply be accessed as if they were tables:

Select * from REDPARTS;

- Create view CITYPAIRS that has distinct pairs supplier cities and city parts.

```
CREATE    VIEW CITYPAIRS (SCITY, PCITY)
 AS      SELECT   DISTINCT  S.CITY, P.CITY
 FROM    S, SP, P
 WHERE  S.S# = SP.S#
AND       SP.P# = P.P#;
```

**DROP VIEW**

The general format of this operation is as follows:

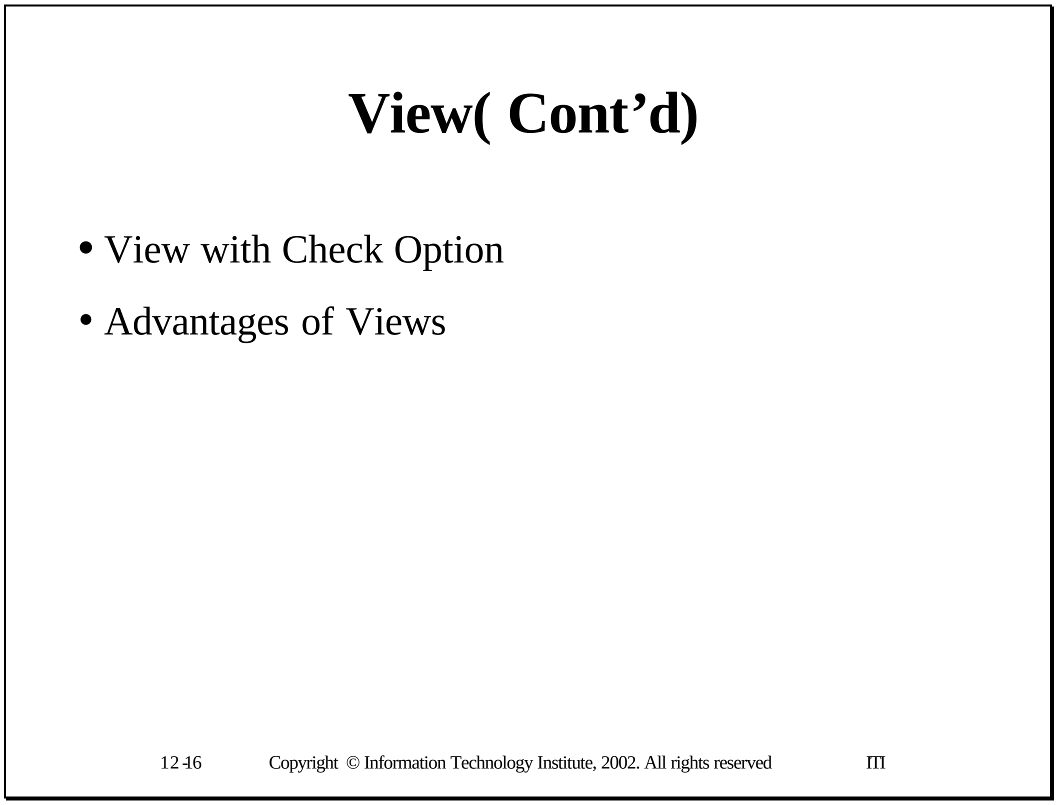
DROP VIEW view-name;

The specified view is dropped (its definition is removed from the catalog). Any view defined in terms of that view is automatically dropped too.

**Example:**

DROP VIEW REDPARTS;

The virtual table that had the "RECPARTS" properties no longer exists.



## View( Cont’d)

- View with Check Option
- Advantages of Views

12-16 ITI

### Check option

**Example:**

```
CREATE VIEW suppliers
AS
SELECT *
FROM suppliers
WHERE status > 15
WITH CHECK OPTION;
```

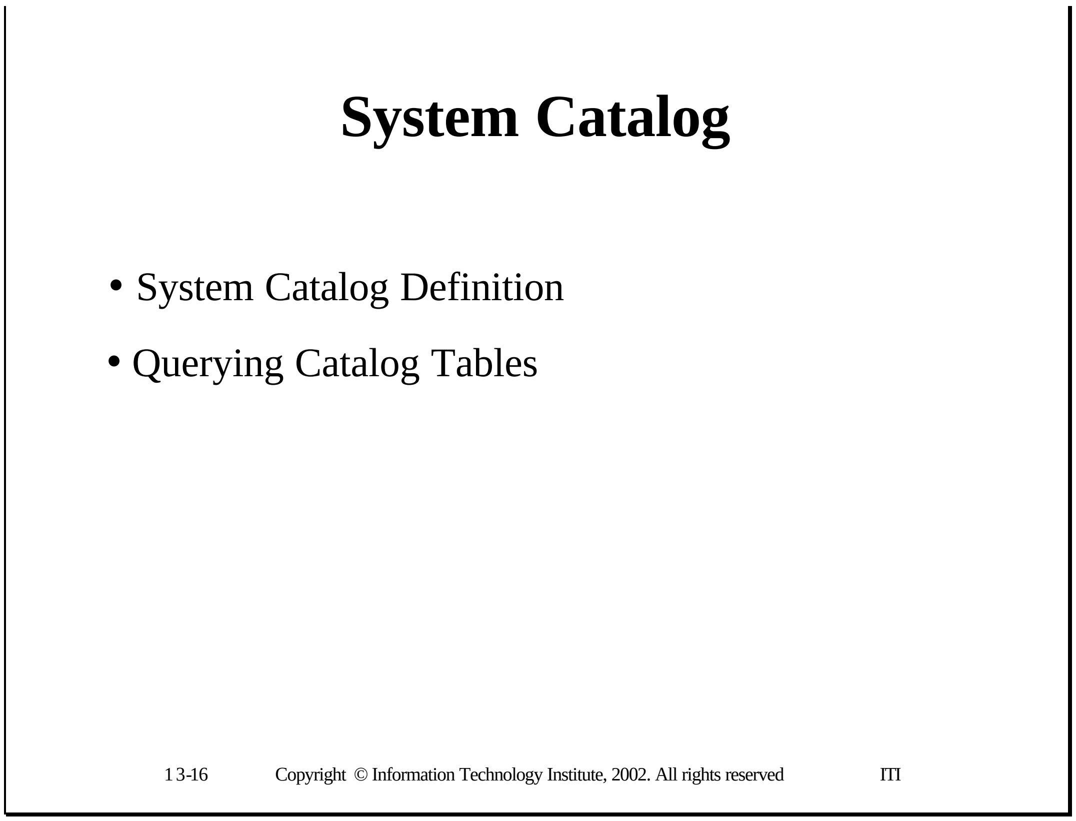
The clause “with check option” indicates that UPDATE and INSERT operations carried out on the view should be checked to ensure that every updated or inserted row still satisfies the view-defining condition (status > 15 in the example).

Once created, views can be treated exactly like normal tables:

- New records can be inserted into the view.
- Existing records can be deleted from the view.
- Records in the view can be updated.

## Advantages of Views

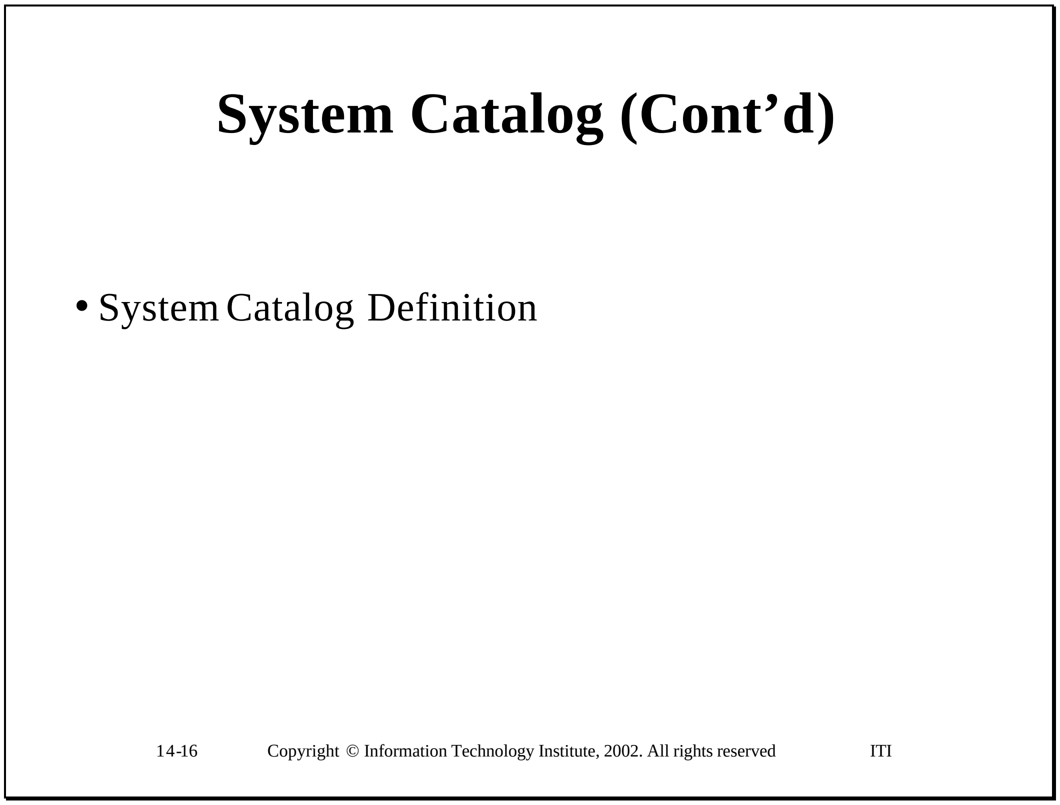
- They allow the same data to be seen by different users in different ways.
- They allow users to focus on the data of concern to them and to ignore the rest.
- They provide security for the rest of the data that is hidden.



## System Catalog

- System Catalog Definition
- Querying Catalog Tables

13-16 ITI



## System Catalog (Cont'd)

- System Catalog Definition

14-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Definition

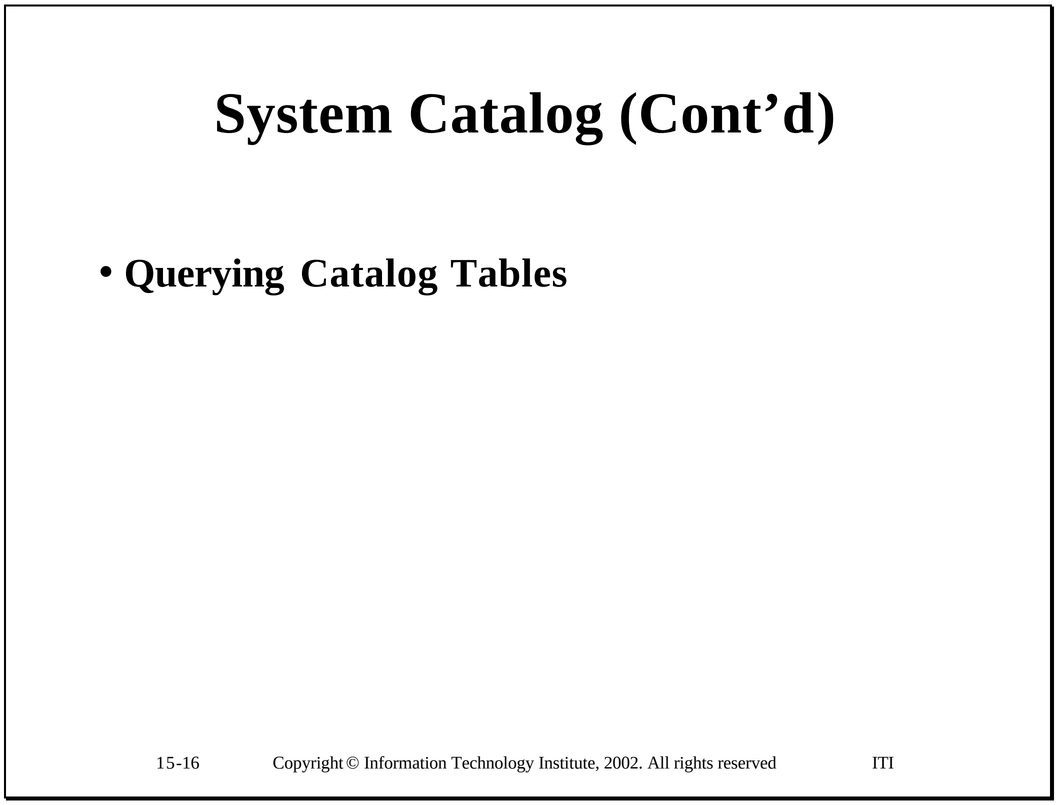
The system catalog is a system database that contains information about the objects contained in the database. This kind of data is sometimes called metadata (data about data). The system catalog carries information about the names of tables, columns names and their data types, indexes … etc.

The system catalog has the following tables:

**SYSTABLES:** This table contains a row for each table (base table or view) in the entire system. It stores the table name, the name of the user that created it, the number of columns in the table, and other information.

**SYSCOLUMNS:** This table contains a row for each column of every table in the entire system. It stores the column name, its table name, its data type and other information.

**SYSINDEXES:** This table contains a row for each index in the entire system. It stores the index name, its table name, its user creator and so on.



## System Catalog (Cont’d)

- **Querying Catalog Tables**

### Querying Catalog Tables

As mentioned, the catalog system consists of tables just like the ordinary user tables, so it can be accessed by means of the SQL statements explained.

**Examples:**

- Find out all the tables that contain an S# column.

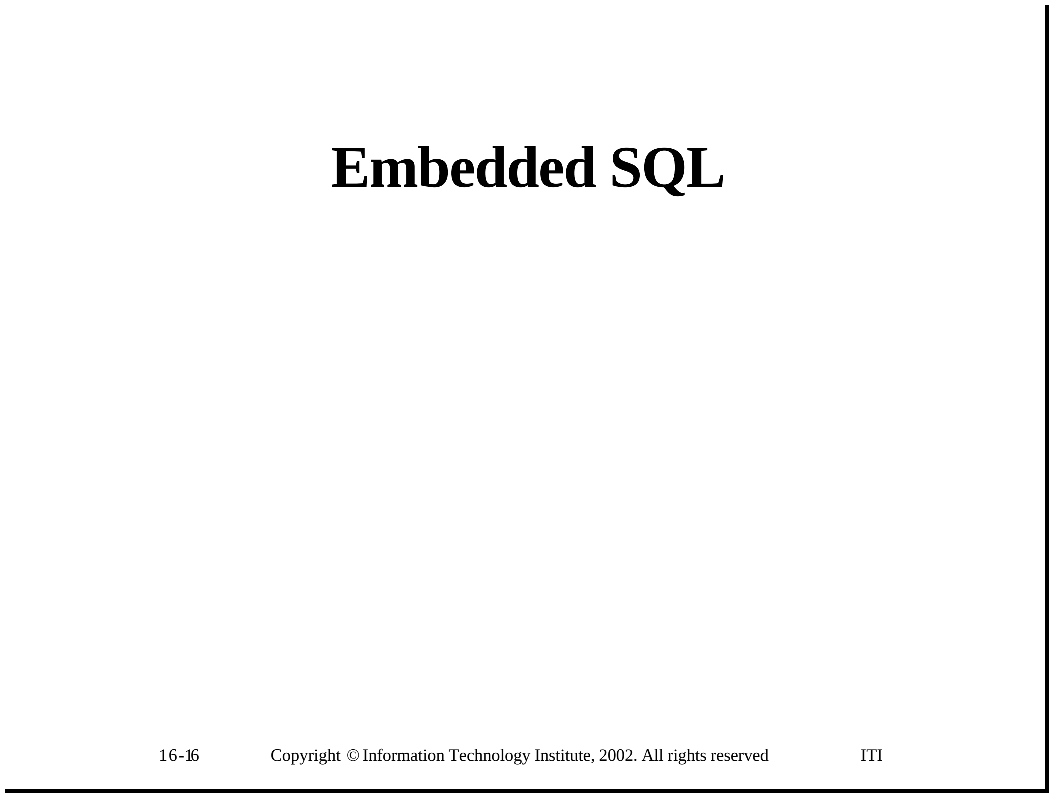
```
SELECT  TBNAME
FROM          SYSCOLUMNS
WHERE   NAME = ‘S#’;
```

- List the columns of the table S.

```
SELECT  NAME
              FROM              SYSCOLUMNS
                 WHERE        TBNAME = ‘S’;
```

3. Find out the number of tables that is owned by the user ‘MANAGER’.

```
SELECT    COUNT (*)
FROM            SYSTABLES
WHERE     CREATOR = ‘MANAGER’
```



## Embedded SQL

16-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

Embedded SQL is a dual mode principle, where any SQL statement that could be executed online at the terminal and could also be used within an application program. Accordingly, the explained SQL statements could be used in the application program by embedding them into the host language code. In addition, SQL statements could be used inside visual basic program, rather than using visual basic script.

- Embedded SQL statements are prefixed by EXEC SQL, so that they can easily be distinguished from statements of the host language.

- An executable SQL statement can appear wherever an executable statement in the host language can appear.

- SQL statements can include references to host variables; such references are prefixed with a colon.

- Any table used in the program should be declared by means of an EXEC SQL DECLARE statement.

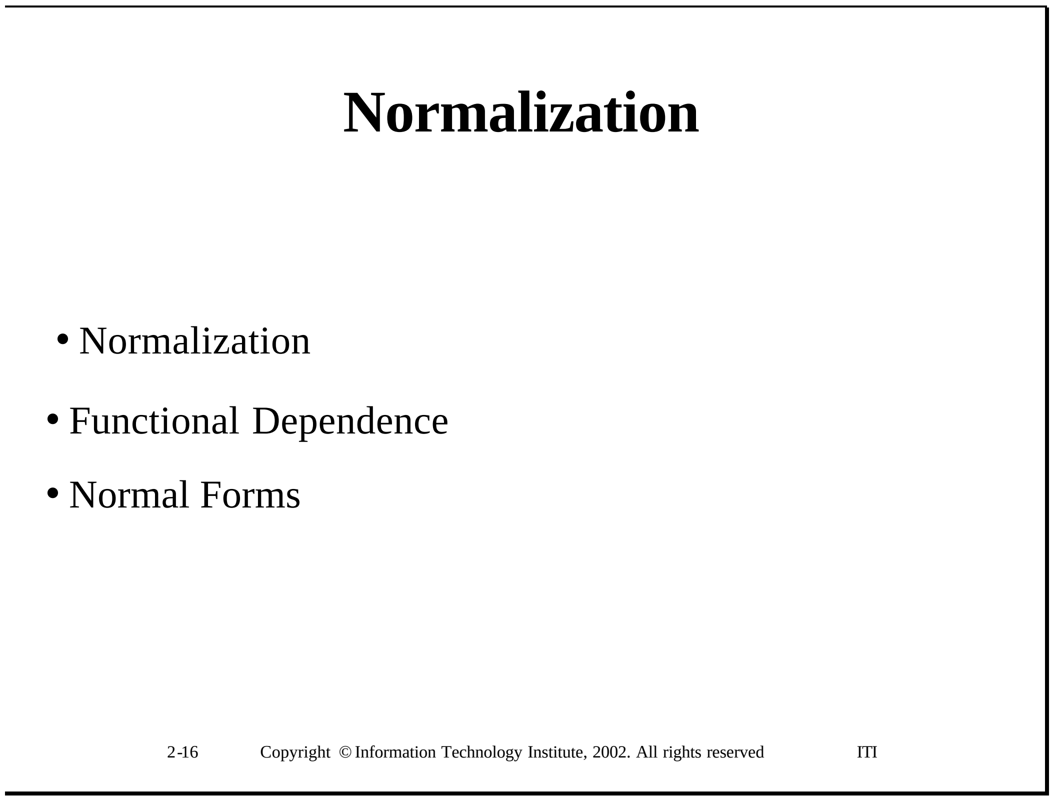
- After any SQL statement has been executed, feedback information is returned to the program in the area called the SQL communication area (SQLCA). The SQL communication area is included in the program by means of an EXEC SQL

INCLUDE SQLCA statement.

- Host variables must have a data type compatible with that of the SQL fields to which they will be compared or assigned to or from.

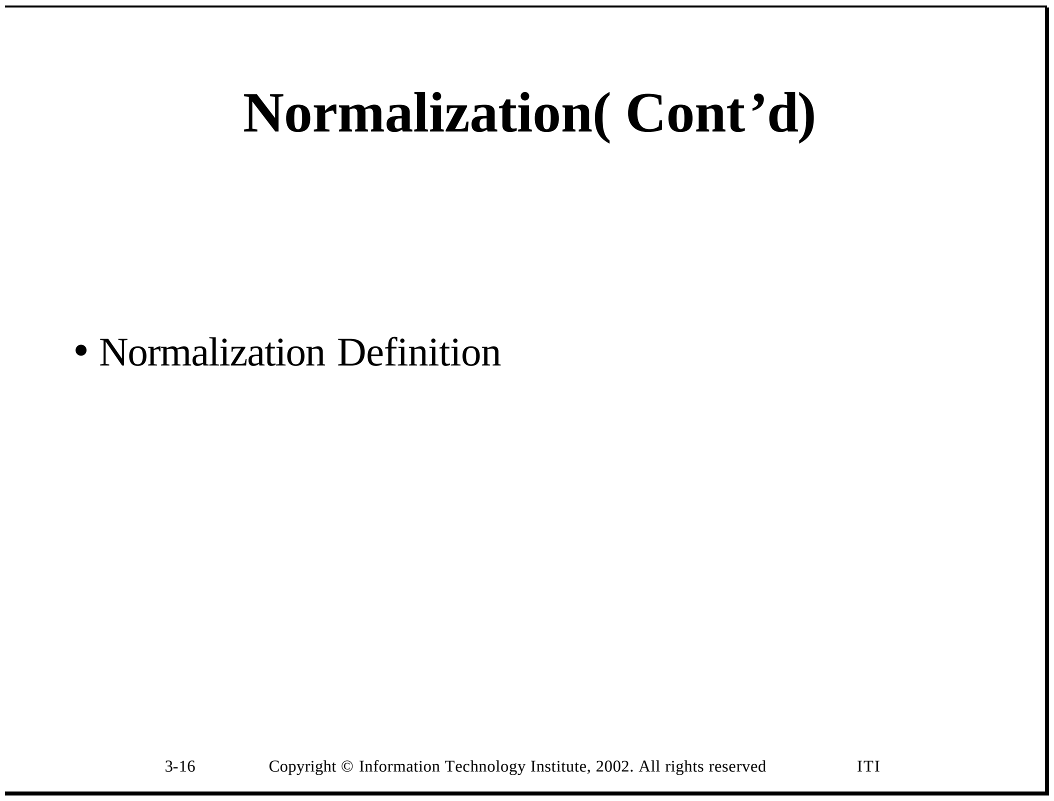
- Host variables and database fields can have the same name.

# Chapter 6
# Normalization



## Normalization

- Normalization
- Functional Dependence
- Normal Forms



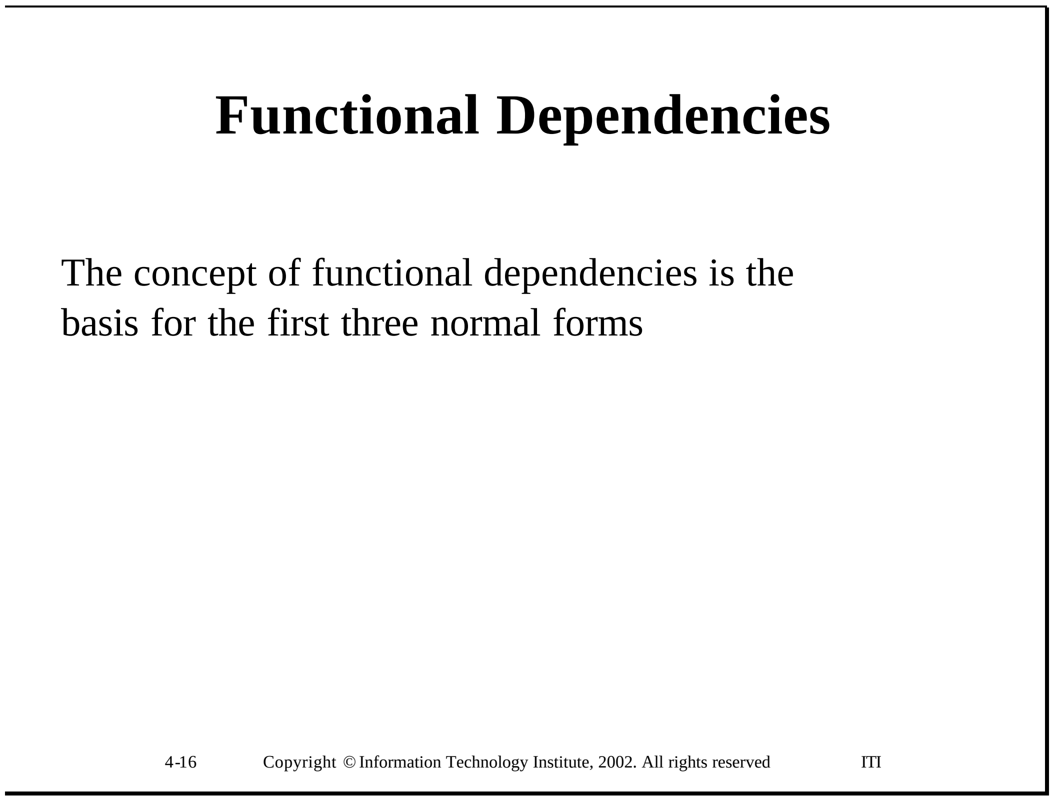
## Normalization( Cont'd)

- Normalization Definition

Normalization is widely used as a guide in designing relational databases. It is essentially a two-step process that puts data into a tabular form by removing repeating groups and duplicates from the relational tables.

The term *fully normalized* is used to describe a collection of tables that are structured so that they do not contain redundant data. In most cases, a well-normalized table is also fully normalized, but there are some situations in which further normalization is desired.

Normalization theory is based on the concepts of normal forms. A relational table is said to be of a particular normal form if it satisfies a certain set of constraints. There are currently five normal forms that have been defined. In this section, we will cover the first three normal forms that were defined by E. F. Codd.



## Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms

### Basic Concepts

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are

- **Mutually independent:** no non-key column is dependent upon any combination of the other columns.
- **Fully dependent upon the primary key.**

The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF. In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of *functional dependencies* and lossless decomposition.

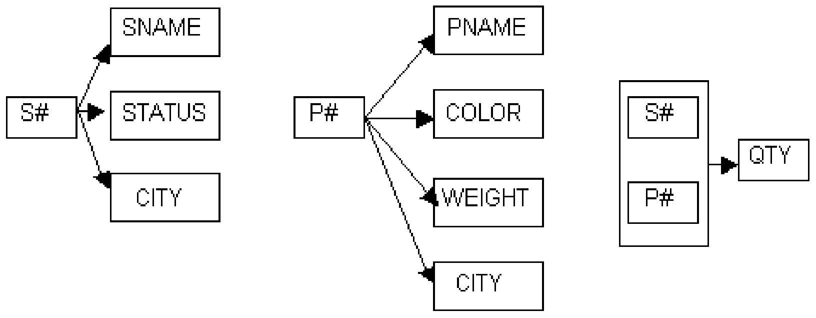
### Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms. A column, Y, of the relational table R is said to be functionally dependent upon column X of R if and only if each value of X in R has associated with it precisely one value of Y at any given time. X and Y may be composite. Saying that column Y is functionally dependent upon X is the same as saying that the values of column X identify the values of column Y. If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

A shorthand notation for describing a functional dependency is:
R.x —> R.y

This can be read as: “In the relational table named R, column x functionally determines (identifies) column y.”

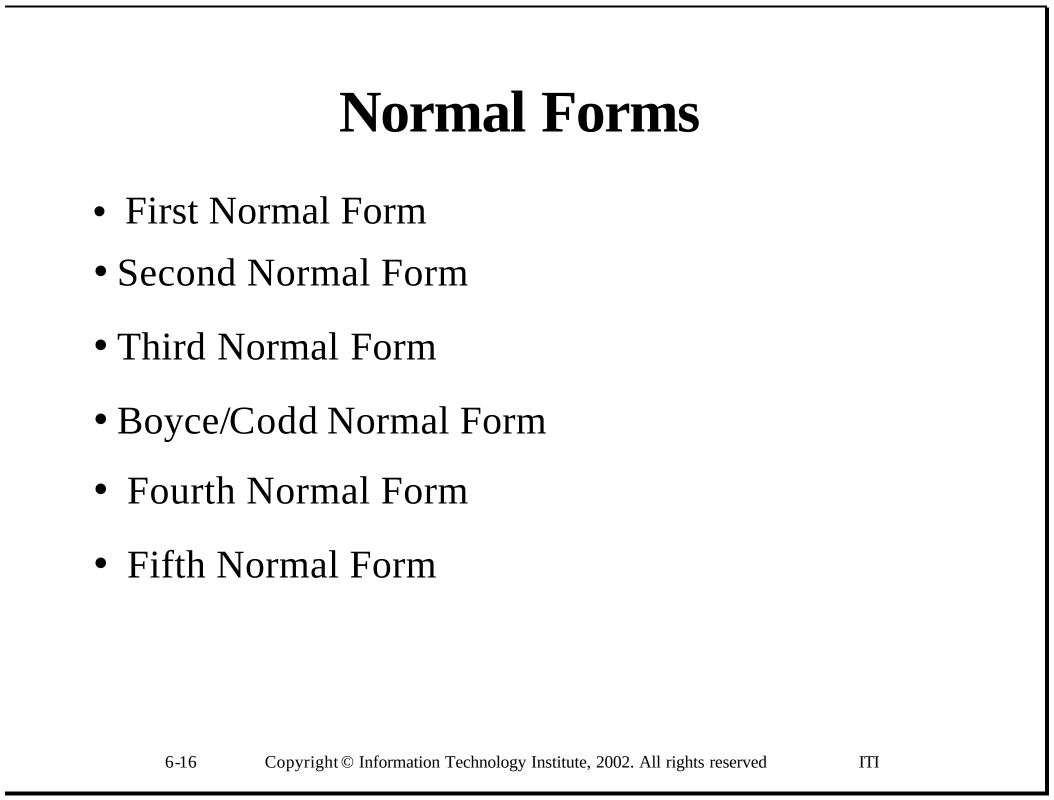
## Functional Dependency Example



**Example:**
In the figure, attributes SNAME, Status, and CITY of relation S are each functionally dependent on attribute S# of relation S because, given a particular value for S#, there exists exactly one corresponding value for each SNAME, Status and CITY. This can be represented as:
S.S#----→ S.(SNAME, Status, CITY)

The same can be applied on relation P.
Also for the relation of Supplier-Product, the attribute QTY depends on both the supplier and the product which was ordered.

**Full functional dependency** applies to tables with composite keys. Column Y in relational table R is fully functional dependent on X of R if it is functionally dependent on X and not functionally dependent upon any subset of X. Full functional dependence means that when a primary key is composite, made of two or more columns, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

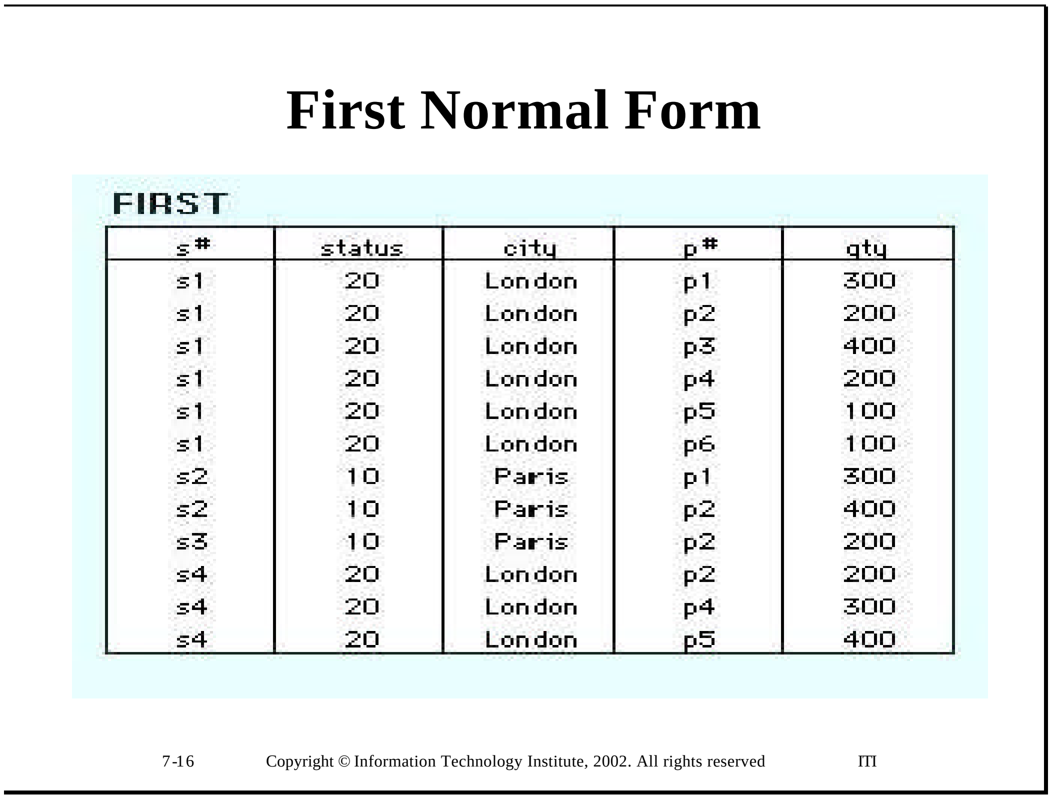


## Normal Forms

- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce/Codd Normal Form
- Fourth Normal Form
- Fifth Normal Form

6-16 ITI

The existing Normal Forms are first, second, third, Boyce/Codd, fourth, and fifth Normal Forms.



## First Normal Form

FIRST

| s# | status | city | p# | qty |
|---|---|---|---|---|
| s1 | 20 | London | p1 | 300 |
| s1 | 20 | London | p2 | 200 |
| s1 | 20 | London | p3 | 400 |
| s1 | 20 | London | p4 | 200 |
| s1 | 20 | London | p5 | 100 |
| s1 | 20 | London | p6 | 100 |
| s2 | 10 | Paris | p1 | 300 |
| s2 | 10 | Paris | p2 | 400 |
| s3 | 10 | Paris | p2 | 200 |
| s4 | 20 | London | p2 | 200 |
| s4 | 20 | London | p4 | 300 |
| s4 | 20 | London | p5 | 400 |

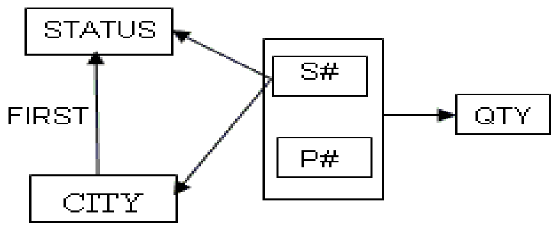
**First Normal Form (1NF):**

A relation *R* is in first normal form (*1NF)*, if and only if all underlying domains contain atomic values only (i.e. The relation 'First' is in the *1NF*, as each domain contains simple values).

The *1NF* has disadvantages regarding modifications. The following are some of the problems that may occur with each of the basic operations:

- **Inserting**: S5 cannot be inserted until it supplies some part.
- **Deleting**: Deleting S3, p2 causes a loss of information about S3.
- **Updating**: CITY values for a given supplier are redundant, and thus all occurrences should be updated. If supplier S1 moved from London to New York, then six rows would have to be updated with this new information

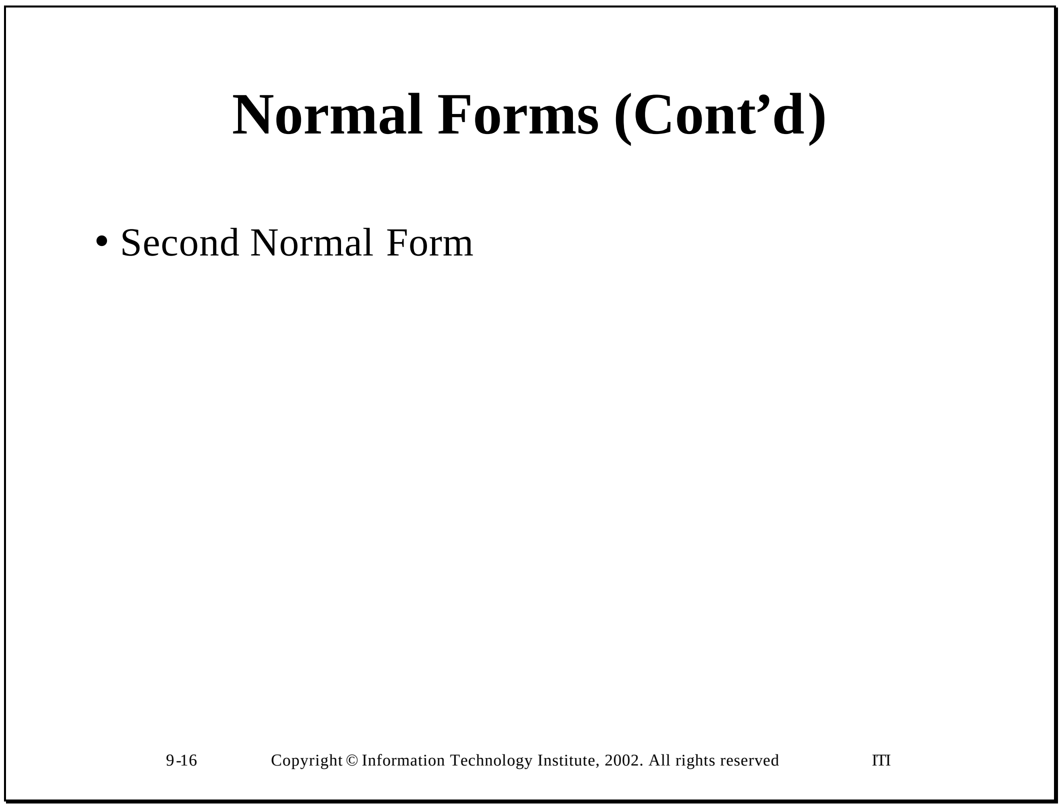
## Functional Dependency of Relation First



*Functional Dependencies in the relation FIRST*

**For example**:

Let the relation *FIRST(S#, STATUS, CITY, P#, QTY)* where attribute is functionally dependent on the composite attribute. *(*This can be written as *S.CITY* ® *S.STATUS. (S#,P#)* is a primary key of *FIRST*. *STATUS, CITY* are not a FFD on the primary key. The figure shows the dependency diagram of this example



## Normal Forms (Cont'd)

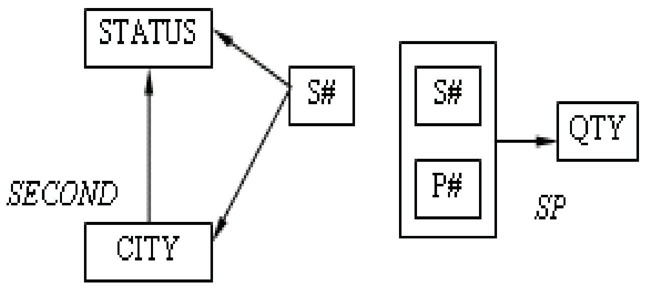
- Second Normal Form

9-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Second Normal Form (2NF)

A relation *R* is in second normal form (*2NF)*, if and only if it is in *1NF* and every non-key attribute is fully dependent on the primary key.

## Example



*Functional Dependencies in the relation SECOND & SP*

**For example**:

Replace relation *FIRST* by two relations: *SECOND (S#, STATUS, CITY)* and *SP(S#, P#, QTY).* The figure shows the dependency diagram in the relation SECOND and SP:

The following is a sample of tabulation for SECOND and SP:

**SECOND**

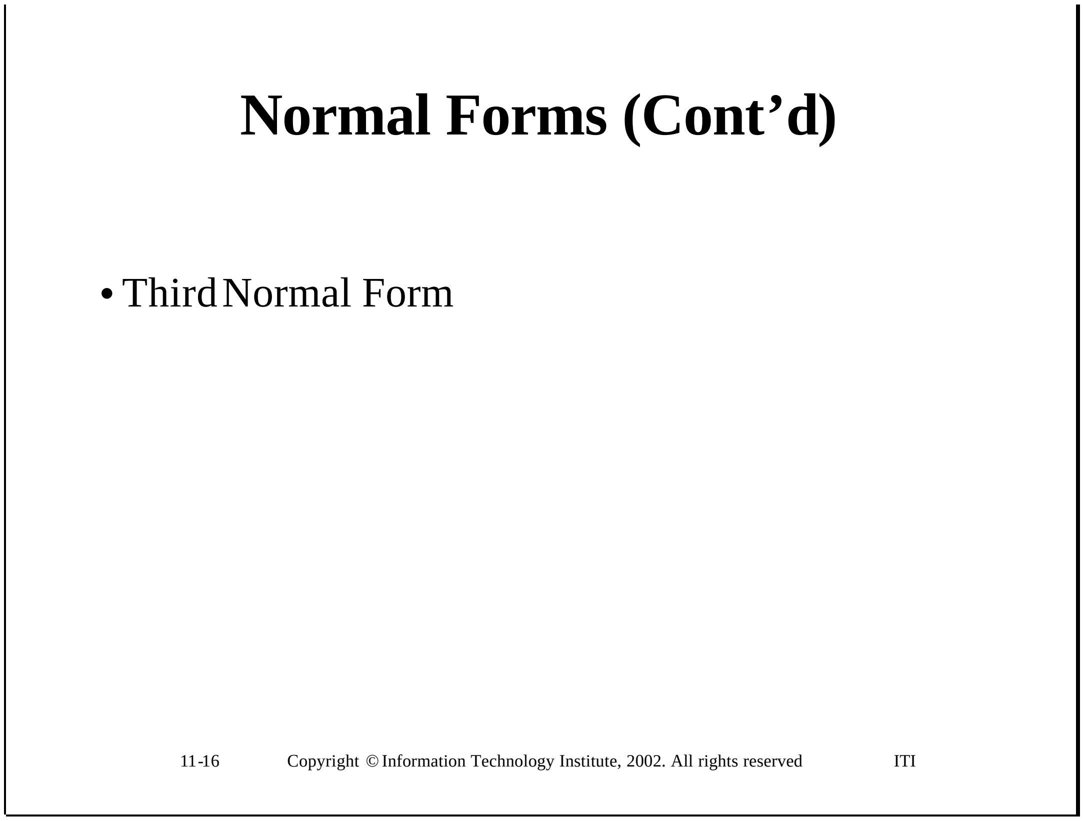
| S# | Status | CITY |
|---|---|---|
| S1 | 20 | Cairo |
| S2 | 10 | Banha |
| S3 | 10 | Banha |
| S4 | 20 | Cairo |
| S5 | 30 | Tanta |

**SP**

| S# | P# | QTY |
|---|---|---|
| S1 | P1 | 300 |
| S1 | P2 | 200 |
| S1 | P3 | 400 |
| S1 | P4 | 200 |
| S1 | P5 | 100 |
| S1 | P6 | 100 |
| S2 | P1 | 300 |
| S2 | P2 | 400 |
| S3 | P2 | 200 |
| S4 | P5 | 200 |
| S4 | P4 | 300 |

The problems that may have occurred in the case of *1NF* are solved for each of the basic operations:

- **Inserting**: S5 can be inserted even if it does not supply any part.
- **Deleting**: Deleting S3, p2 do not lose information about S3.
- **Updating**: Redundancy in CITY is eliminated.



## Normal Forms (Cont'd)

- Third Normal Form

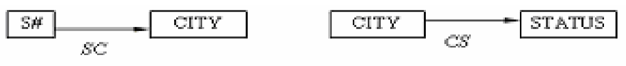
11-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Third Normal Form (3NF)

A relation *R* is in third normal form (*3NF)*, if and only if it is in *2NF* and every non-key attribute is non transitively dependent on the primary key.
In the previous example, the SECOND relation might still cause problems on applying basic operations; the dependency of STATUS on S#, is transitive (via CITY). The following update anomalies may take place:

- **Inserting**: A city cannot have a certain status, until there is some supplier located in that city.
- **Deleting**: Deleting a certain city, will destroy the information for the supplier and information for the status of the City (e.g. Deleting S5 tuple from SECOND).
- **Updating**: The status for a given city is duplicated in SECOND numerous times.

The solution for these problems is to replace the original relation by two projections SC and CS, as shown in the following figure. SC and CS are now in 3NF.



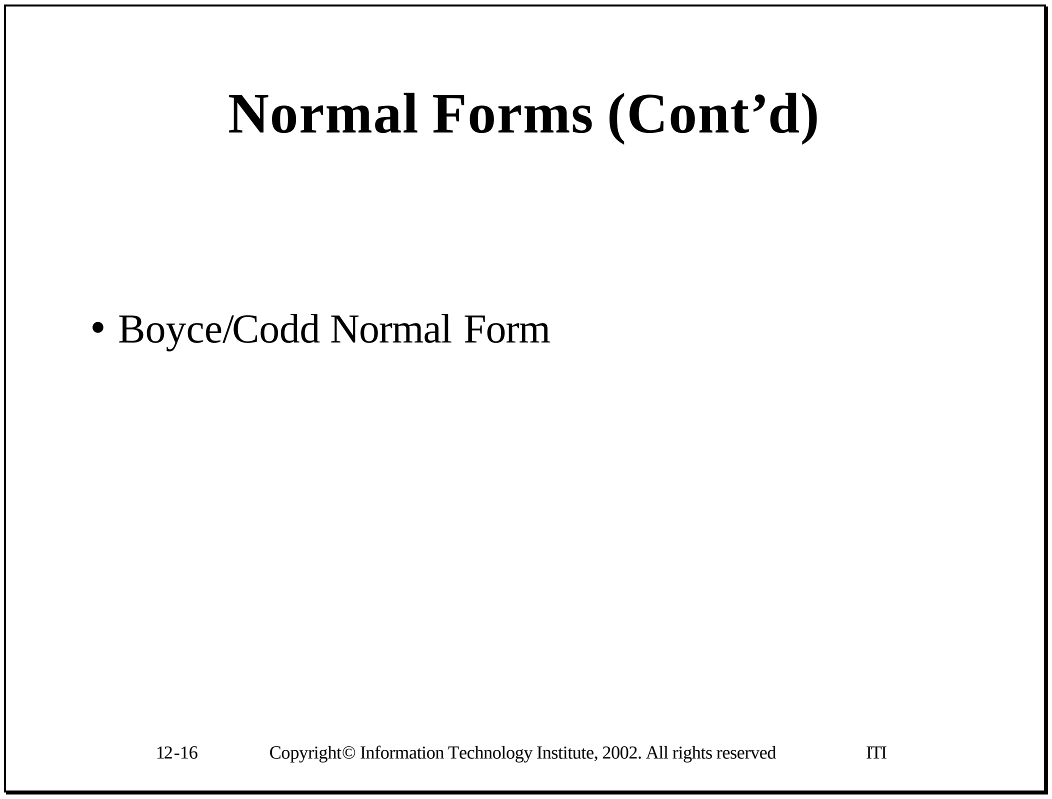
*Functional Dependencies in the relation SC & CS*

**SC**

| **S#** | **CITY** |
|---|---|
| S1 | Cairo |
| S2 | Banha |
| S3 | Banha |
| S4 | Cairo |
| S5 | Tanta |

**CS**

| **CITY** | **STATUS** |
|---|---|
| Cairo | 20 |
| Banha | 10 |
| Tanta | 30 |



## Normal Forms (Cont’d)

• Boyce/Codd Normal Form

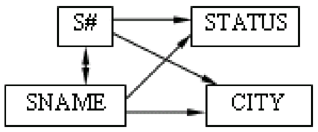
**Boyce/Codd Normal Form (BCNF):**
Originally, *1NF*, *2NF*, and *3NF* were defined by Codd. These normal forms do not appropriately handle a relation with the following properties:

- A relation with multiple candidate keys.
- A relation with composite candidate keys.
- A relation with overlapping candidate keys.

A determinant is any attribute on which some other attribute is (fully) functionally dependent. A relation *R* is in the ***Boyce/codd* normal form *(BCNF*),** if and only if every determinant is a candidate key and only candidate keys are determinants.

**Examples:**

- **Non-overlapping candidate key**
  Consider the example of the supplier relation S, S# and SNAME are candidate keys, where S (S#, SNAME, STATUS, CITY) with the following FD diagram:
  (NOTE: S is in BCNF).



- **Overlapping Candidate Key**

  Assume that in SSP, (S#, P#) and (SNAME, P#) are its candidate key, where

  SSP (S#, SNAME, P#, QTY). SSP is not in BCNF. To solve this problem, break

  the relation SSP down into two projections, as shown:

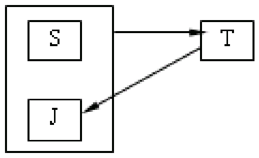
  SS (S#, SNAME) and SP(S#, P#, QTY).

  Both of SS and SP are in the BCNF.

Assume that we have an SJT (S, J, T) relation, where S is the student, J is the subject and T is the teacher. The FD diagram is as shown below. Assume that:

- For each subject, each student is taught by only one teacher.
- Each teacher teaches only one subject (but each subject is taught by several

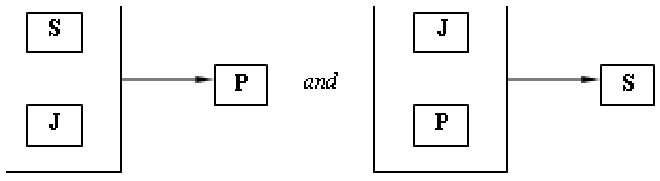
  teachers).



So, there are two overlapping candidate keys, namely the combination (S, J) and (S, T). SJT is not in BCNF. To solve this problem, break the relation SJT down into two projections, ST and TJ.
Both of ST and TJ are in the BCNF.

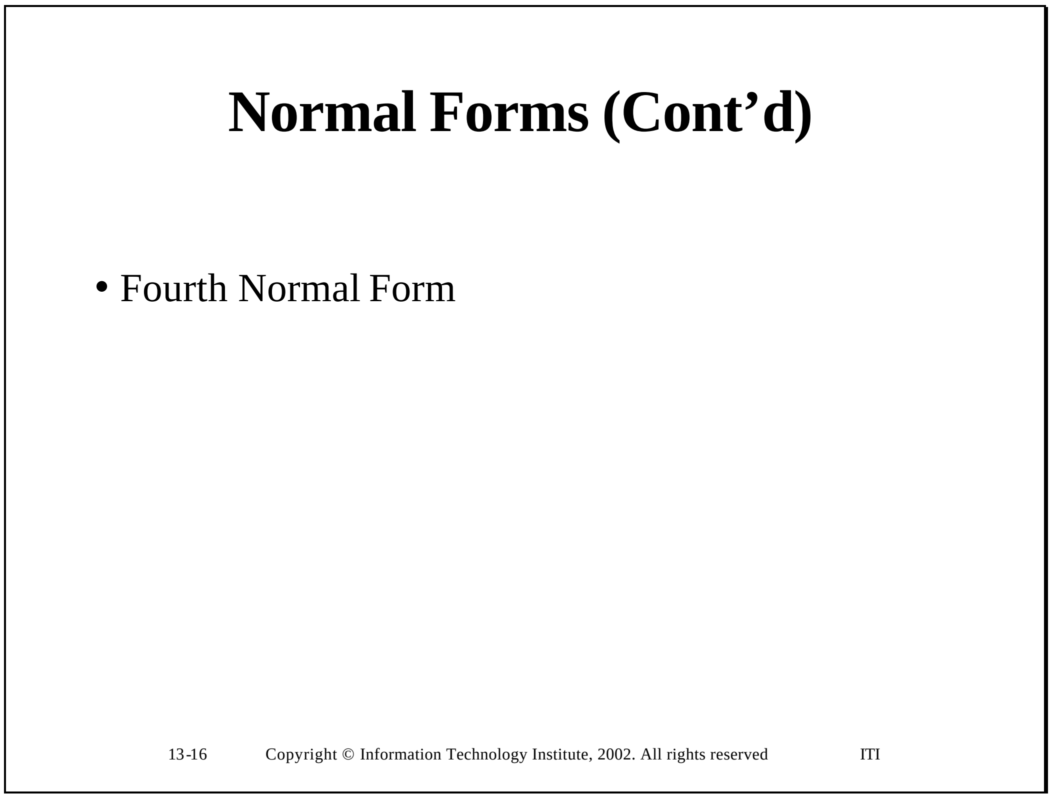
Assume that in an EXAM (S, J, P), S is the student, J is the subject and P is the position. The FD diagram is as shown below. Assume that:

NO two students can obtain the same position in the same subject.



So, there are two overlapping candidate keys, namely the combination (S, J) and (J, P). SJP is in BCNF.

So, Overlapping candidate keys do not necessarily lead to problems in every case.



## Normal Forms (Cont'd)

- Fourth Normal Form

**Fourth Normal Form (4NF):**

Given a relation *R* with attributes *A*, *B* and *C*, and the multi-valued dependence *(MVD): R.A* -->–> *R.B*, one value of A determines many values of B if and only if the set of *B*-values matching a given *A-value/C-value* pair in *R* depends only on the *A*-value and is independent on the *C-value*. As usual *A, B* and *C* may be composite.

A relation *R* with attributes *A, B,* and *C* can be non loss decomposed into its two projections *R1(A , B)* and *R2(A , C)* if and only if the *MVDs,* A -->-->B½C hold in R.

A relation R is in fourth normal form (*4NF*), if and only if whenever there exists an *MVD* in *R*, say *A* -->--> *B*, then all attributes of *R* are also functionally dependent on *A*. In other words, the only dependencies (*FDs and MVDs*) in *R* are of the form *K*–>*X*. Equivalently, *R* is in *4NF* if and only if it is in *BCNF* and all *MVD*s in *R* are in fact *FD*s.

## Normal Forms (Cont’d)

- Fifth Normal Form

**Fifth Normal Form (5NF):**

A relation *R* satisfies the join dependency *(JD) *(X, Y,..., Z),* if and only if R is equal to the join of its projections on X,Y,..., Z, where X,Y,..., Z are subsets of the set of attributes of *R.*

*R (A,B,C)* satisfies the *JD*(AB, AC)* if and only if it satisfies the pair of *MVDs A* --->---> *B½C*

A relation R is in the fifth normal form 5NF, also called projection- join normal form (PJNF), if and only if every dependency in *R* is a consequence of the candidate keys of *R.*

The overall process can be summarized informally in the following set of rules:

- Take the projection of the original *1NF* relation to eliminate any non-full functional dependencies. This step will produce a collection of *2NF* relations.
- Take the projection of those *2NF* relations to eliminate any transitive functional dependencies. This step will produce a collection of *3NF* relations.
- Take the projection of those *3NF* relations to eliminate any remaining functional dependencies in which the determinant is not a candidate key. This step will produce a collection of *BCNF* relations.

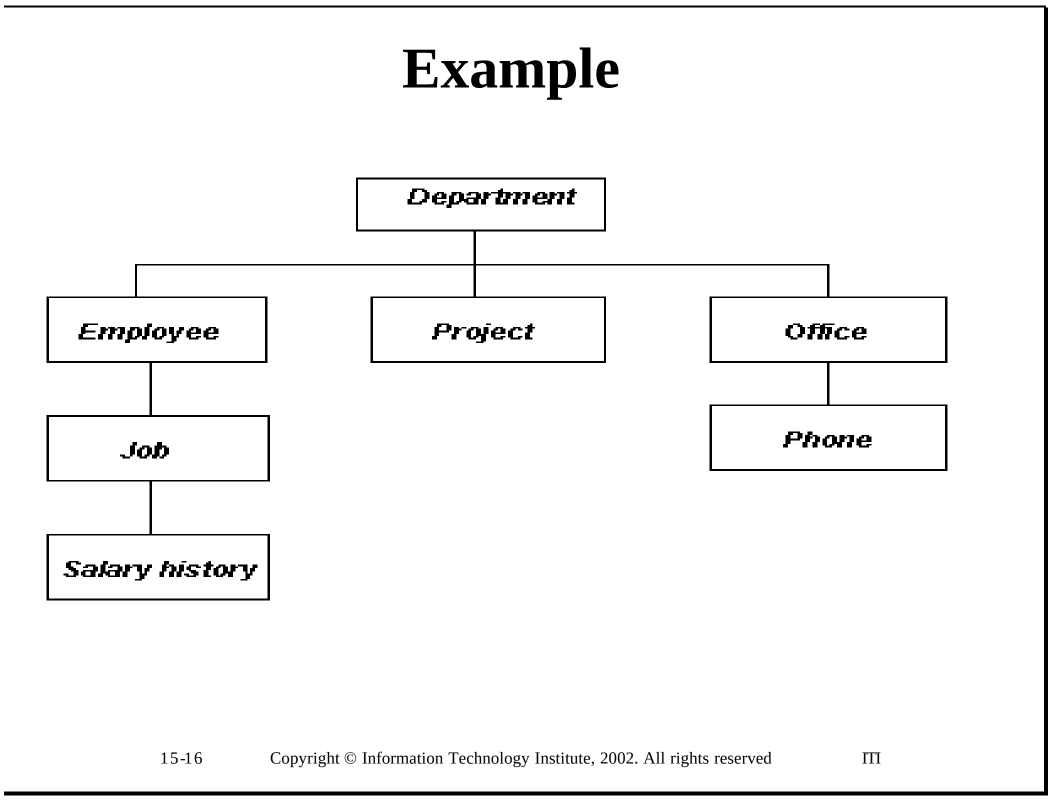
**Note:** Steps 1-3 can be condensed into the single guideline "Take projections of the original to eliminate all FDs in which the determinant is not a candidate key".

- Take the projection of those *BCNF* relations to eliminate any multi-valued dependencies that are not functional dependencies. This step will produce a collection of *4NF* relations.

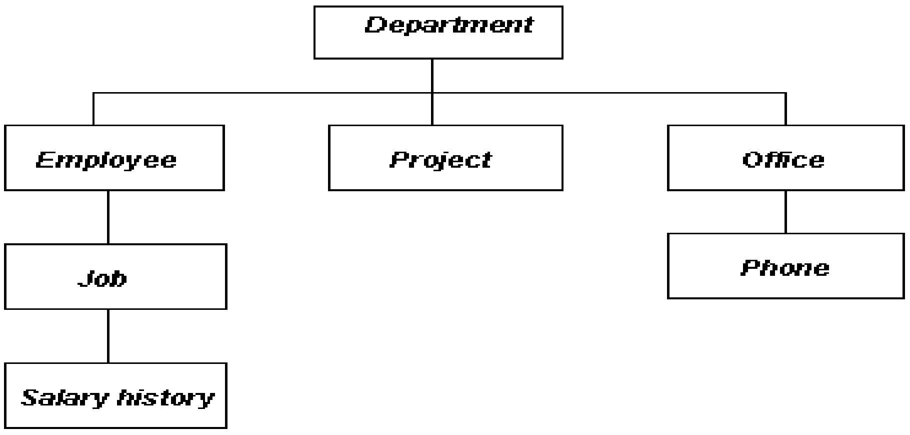
**Note:** In practice, it is useful to eliminate such MVDs before applying steps 1-3 above.

- Take the projection of those *4NF* relations to eliminate any join dependencies that are not implied by the candidate keys, if you can find any. This step will produce a collection of *5NF* relations.

We shall now show an example to illustrate the use of normalization in database design.



## Example



15-16 Copyright © Information Technology Institute, 2002. All rights reserved ITI

**Example**

A company has a set of departments.

Each department has a set of employees, a set of projects and a set of offices.

Each employee has a job history and each job has a salary history.

Each office has a set of phones.

The database has the following information:

- Each department has:
  Dept# (unique), budget, the department manager's employee#(unique).

- Each employee has:
  Empl# (unique), Emp-name, Address, current project#, office#, phone#, title of each job the employee has held, plus date and salary for each distinct salary received in that job.

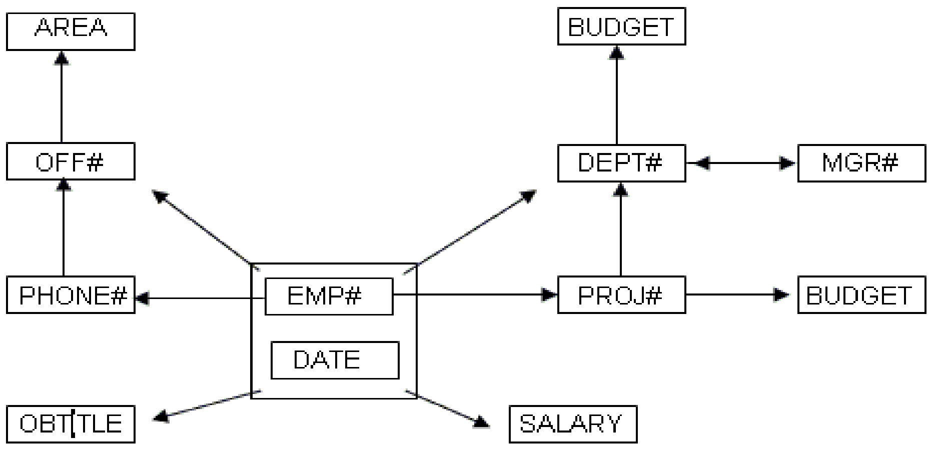
- Each project has:
  Project# (unique), and budget.

- Each office has:
  Office# (unique), area in sq. feet, phone numbers in that office (unique).

**Assumptions:**

- No employee is the manager of more than one department at a time.
- No employee works in more than one department at a time.
- No employee works in more than one project at a time.
- No employee has more than one office ata time.
- No employee has more than one phone at a time.
- No employee has more than one job at a time.
- No project is assigned to more than one department at a time.
- No office is assigned to more than one department at a time.

## Step 0



**Step 0:**

DEPT0 (DEPT#, DBUDGET, MGR#, XEMP0, XPROJ0, XOFFICE0)

(EMP#, Emp-name, Address, PROJ#, OFF#, PHONE#, XJOB0)

(JOBTITLE, XSALHIST0)

(DATE, SALARY)

(PROJ#, PBUDGET)

(OFF#, AREA, XPHONE0)

(PHONE#)

**Step 1:**

Change DEPT0 into 1NF

DEPT1 (DEPT#, DBUDGET, MGR#)

EMP1 (DEPT#, EMP#, Emp-name, Address, PROJ#, OFF#, PHONE#)

JOB1 (DEPT#, EMP#, JOBTITLE)

SALHIST1 (DEPT#, EMP#, JOBTITLE, DATE, SALARY)

PROJ1 (DEPT# , PROJ #, PBUDGET)

OFFICE1 (DEPT# , OFF#, AREA)

PHONE1 (DEPT# , OFF#, PHONE# )

**Step 2:**

Change DEPT1 into 2NF

- DEPT1 is also in 2NF.
- EMP1 is also in 2NF; consider EMP# as the primary key as DEPT# is actually redundant in the primary key for this relation (Where EMP# ® DEPT#).
- JOB1: DEPT# is not required as a primary key in this relation.

  EMP# -------→ DEPT# but DEPT# is not FD on (EMP#, JOBTITLE) (i.e the primary key), so JOB1 is not in 2NF. Replace it by:

  JOB2A (EMP#, JOBTITLE), which is in 2NF.

  JOB2B (EMP#, DEPT#), which is a projection of EMP1, so we can simply reject it.

- SALHIST1, as in JOB1, is replaced by:

  SALHIST2: (EMP#, JOBTITLE, DATE, SALARY) and JOB2B.

  But JOB2A is a projection of SALHIST2, the relation JOB2A is rejected.

- PROJ1as in EMP1, is in 2NF if we consider DEPT# as a non-key attribute.
- OFFICE1 is in 2NF after considering DEPT# as a non-key attribute.
- PHONE1, as in SALHIST1 and JOB1, is replaced by two 2NF relations:

  PHONE2 (PHONE#, OFF#) – PHONE# and OFF# may exist without being assigned to any person, so we can't consider PHONE2 as a projection of EMP1.

  (DEPT#, OFF#) which is a projection of OFFICE1 and is rejected.

So, the 2NF relations are:

DEPT2 (DEPT#, DBUDGET, MGR#)

EMP2 (EMP#, DEPT#, Emp-name, Address, PROJ#, OFF#, PHONE#)

SALHIST2 (EMP#, DATE, JOBTITL, SALARY)

PROJ2 (PROJ #, DEPT#, PBUDGET)

OFFICE2 (OFF#, DEPT#, AREA)

PHONE2 (PHONE#, OFF#)

**Step 3:**

Eliminating transitive dependencies will reduce the 2NF relations to 3NF.

EMP2 is the only relation that needs to be normalized to relations in 3NF (OFF# and DEP# are both transitively dependent on EMP#; OFF# via PHONE#, and DEPT# via PROJ# and OFF# (and hence via PHONE#)).

The 3NF relations corresponding to EMP2 are

EMP3(EMP#, PROJ#, PHONE#)

X(PHONE#, OFF#)

Y(PROJ #, DEPT# )

Z(OFF#, DEPT#)

X, Y and Z are all projections of existing relations, so the 3NF relations finally are:

DEPT3( DEPT#, DBUDGET, MGR# )

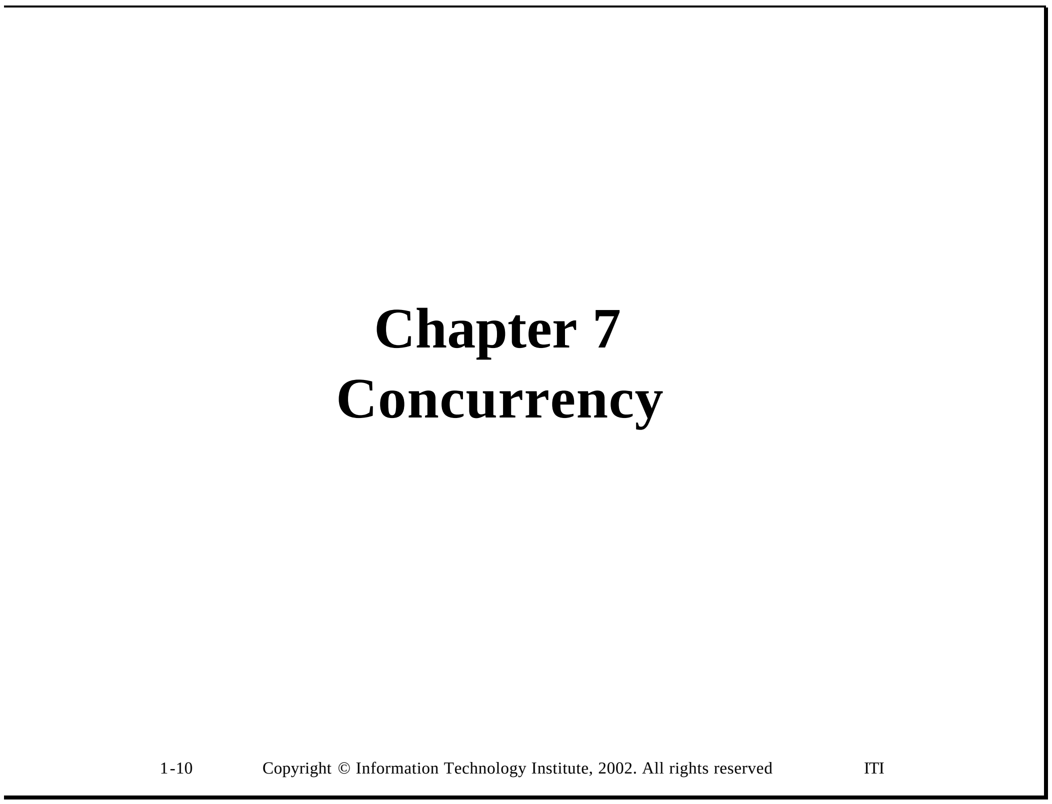
EMP3( EMP#, PROJ#, Emp-name, Address, PHONE#)

SALHIST3 (EMP#, DATE, JOBTITL , SALARY )

PROJ3(PROJ #, DEPT# , PBUDGET )

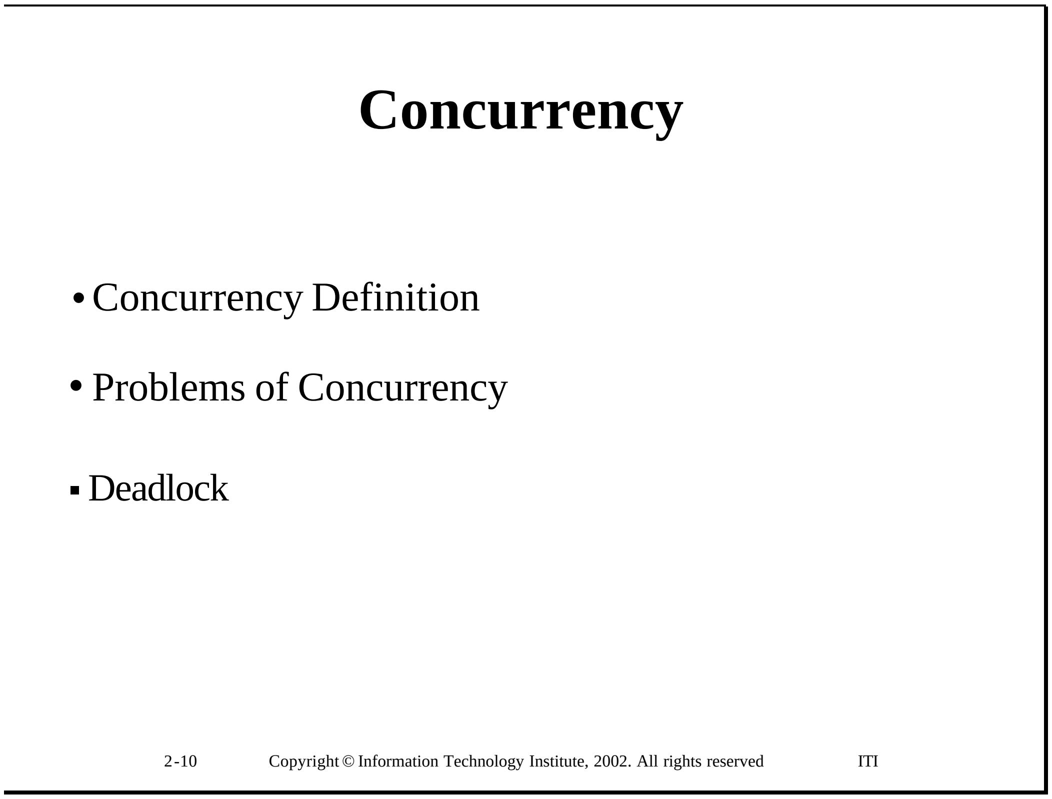
OFFICE3( OFF#, DEPT#, AREA)

PHONE3( PHONE#, OFF#)



# Chapter 7 Concurrency

1-10 ITI



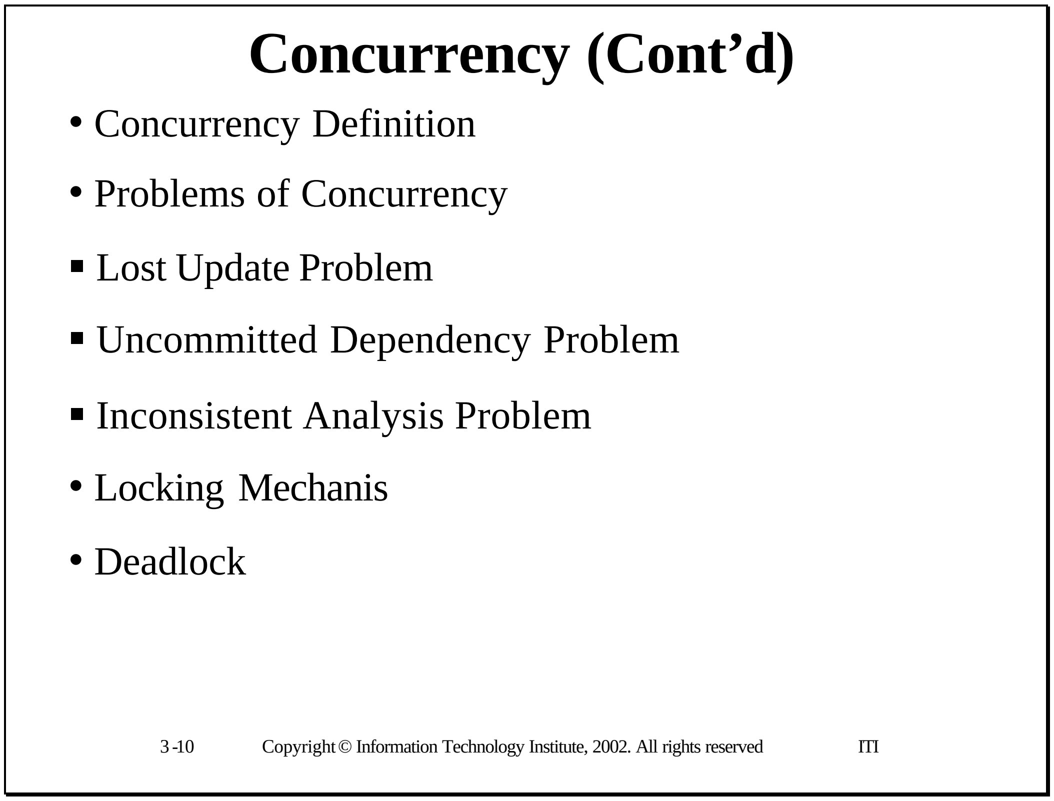
## Concurrency

- Concurrency Definition
- Problems of Concurrency
- Deadlock

2-10 Copyright © Information Technology Institute, 2002. All rights reserved ITI

First, we can define the term transaction. **Transaction** is a logical unit of work, consisting of multiple statements. If this logical unit of work has been successfully completed, the database is in a consistent state and all updates made by that unit of work can now be **committed** or, in other words, made permanent.

On the other hand, if something goes wrong, then the database might be in an inconsistent state, and all of the updates made by the logical unit so far must be **rolled back** or undone.



## Concurrency (Cont'd)

- Concurrency Definition
- Problems of Concurrency
  - Lost Update Problem
  - Uncommitted Dependency Problem
  - Inconsistent Analysis Problem
- Locking Mechanis
- Deadlock

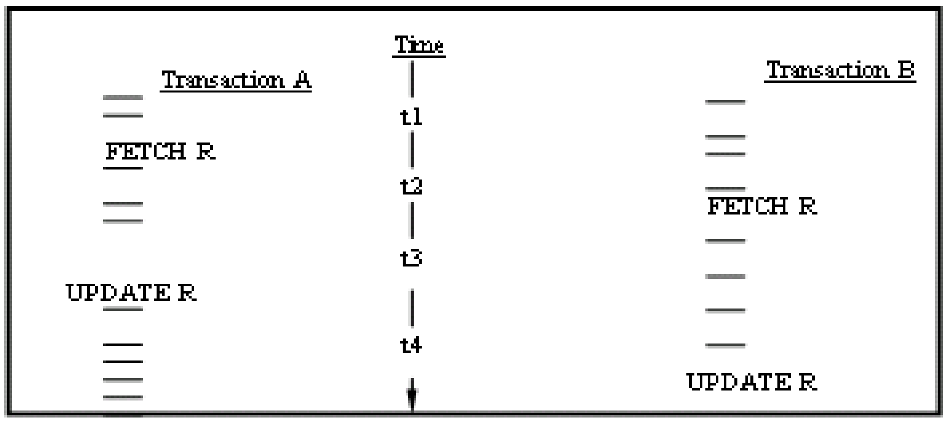
3 -10 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Concurrency

Most database management systems are multiple-user systems in which many users are allowed to access the same database at the same time. This feature requires some kind of concurrency control mechanism to ensure that the concurrent processes do not interfere with the operation of each other.

There are numerous problems that can arise if such mechanism is not applied:

## Lost Update Problem



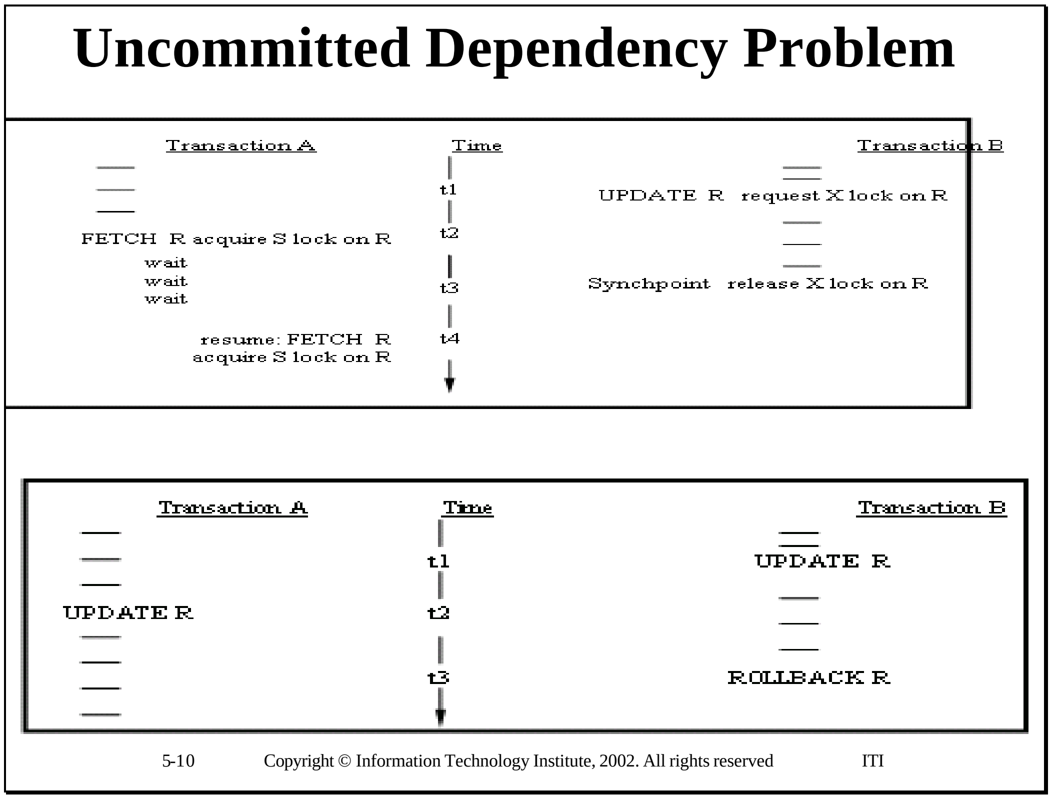
***Transaction A loses an update at time t4***

- **The lost update problem:**

Consider the case shown in the figure:

- Transaction A retrieves some record R at time t1.
- Transaction B retrieves that same record R at time t2.
- Transaction A updates the record at time t3.
- Transaction B updates the same record (on the basis of the values seen at time t2) at time t4.

Transaction A's update is *lost* at time t4, because transaction B overwrites it without even looking at it.



## Uncommitted Dependency Problem

| Transaction A | Time | Transaction B |
|---|---|---|
| — | | — |
| — | t1 | UPDATE R request X lock on R |
| — | | — |
| FETCH R acquire S lock on R | t2 | — |
| wait | | — |
| wait | t3 | Synchpoint release X lock on R |
| wait | | |
| resume: FETCH R acquire S lock on R | t4 | |

| Transaction A | Time | Transaction B |
|---|---|---|
| — | | — |
| — | t1 | UPDATE R |
| — | | — |
| UPDATE R | t2 | — |
| — | | — |
| — | t3 | ROLLBACK R |
| — | | |
| — | | |

5-10 Copyright © Information Technology Institute, 2002. All rights reserved ITI

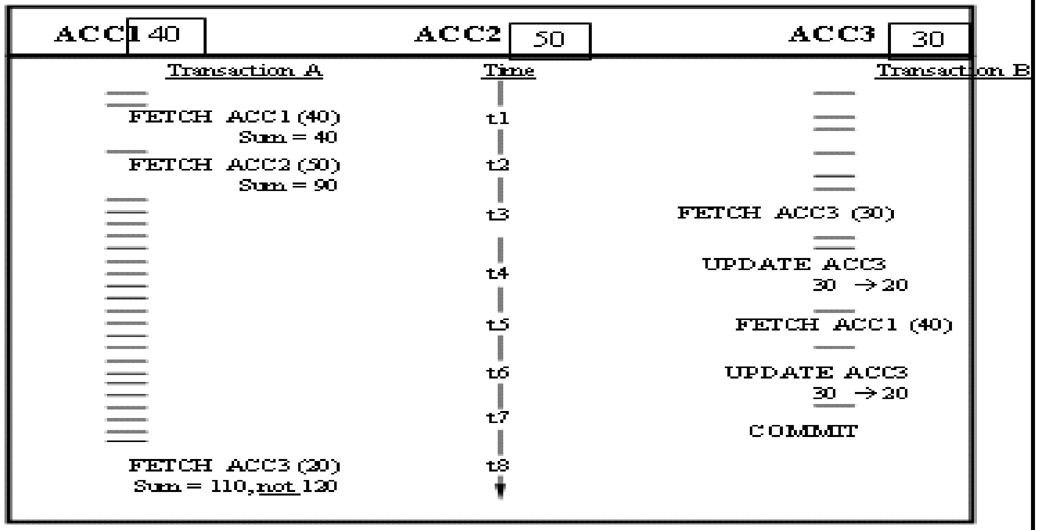
- **Uncommitted Dependency Problem**

  This problem arises if a transaction is allowed to retrieve a record that has been updated by another transaction but has not yet been committed.

  In the first figure, transaction **A** becomes dependent on an uncommitted change at time ***t2***.

In the second figure, transaction **A** updates an uncommitted change at time ***t2*** and loses that update at time ***t3***.

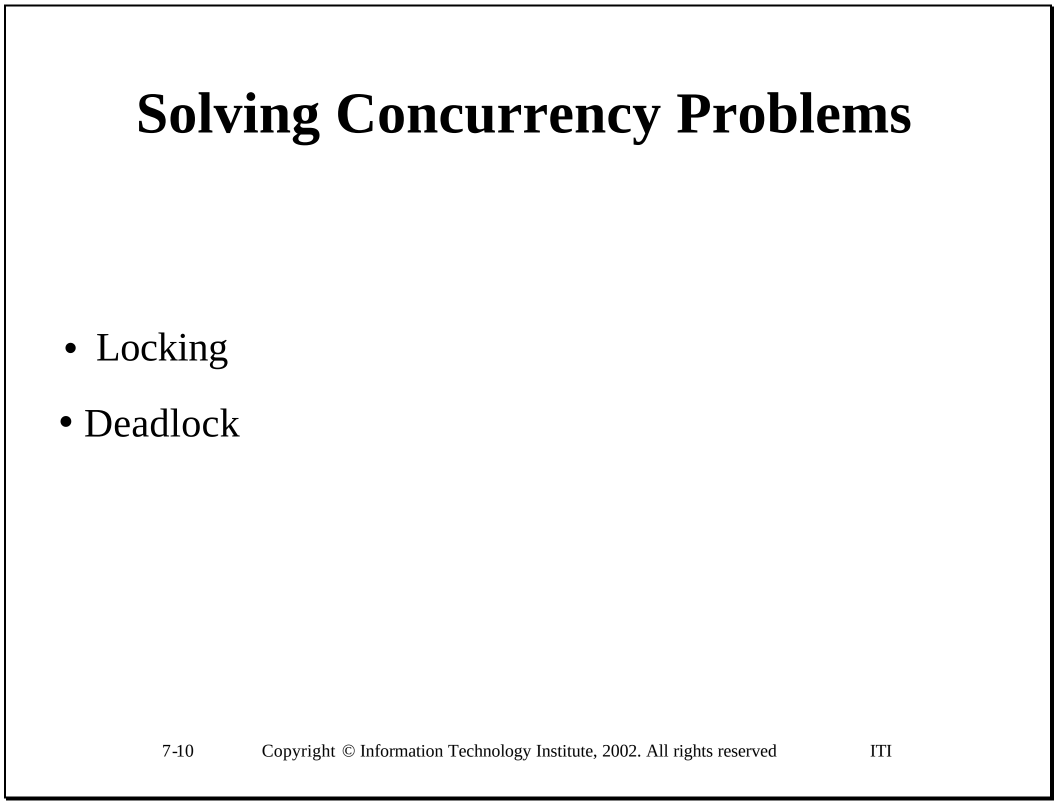
## Inconsistent Analysis Problem



- **Inconsistent Analysis Problem**

  This type of problem is illustrated in the figure. Transactions A and B are operating on account records. Transaction A is summing account balances, and transaction B is transferring an amount 10 from account 3 to account 1. The result produced by A (110) is obviously incorrect; if A writes that result back into the database, it would actually leave the database in an inconsistent state.

All of the previous problems can be solved using the Locking mechanism.



## Solving Concurrency Problems

- Locking
- Deadlock

### Locking

When a transaction needs to ensure that some object (record) -that it is currently using- will not change in some unexpected manner, it requires a lock on this object. A lock is a variable associated with a data item to describe its status with respect to possible operations that can be applied to it (read, update … etc).

There are two types of locking; **Exclusive locks** (**X** locks) or **Shared locks** (**S** locks).

If transaction **A** holds an **X** lock on record **R**, then a request from transaction **B** for a lock of either type on **R** will cause **B** to go into a wait state until **A**'s lock is released.

If transaction **A** holds an **S** lock on record **R**, then:

- A request from transaction **B** for an **X** lock on **R** will cause **B** to go into a wait state (and B will wait until A's lock is released).
- A request from transaction **B** for an **S** lock on **R** will be granted (that is, B will now also hold an S lock on R).

The following table summarizes the above points.

| - | X | S | - |
|---|---|---|---|
| X | N | N | Y |
| S | N | Y | Y |
| - | Y | Y | Y |

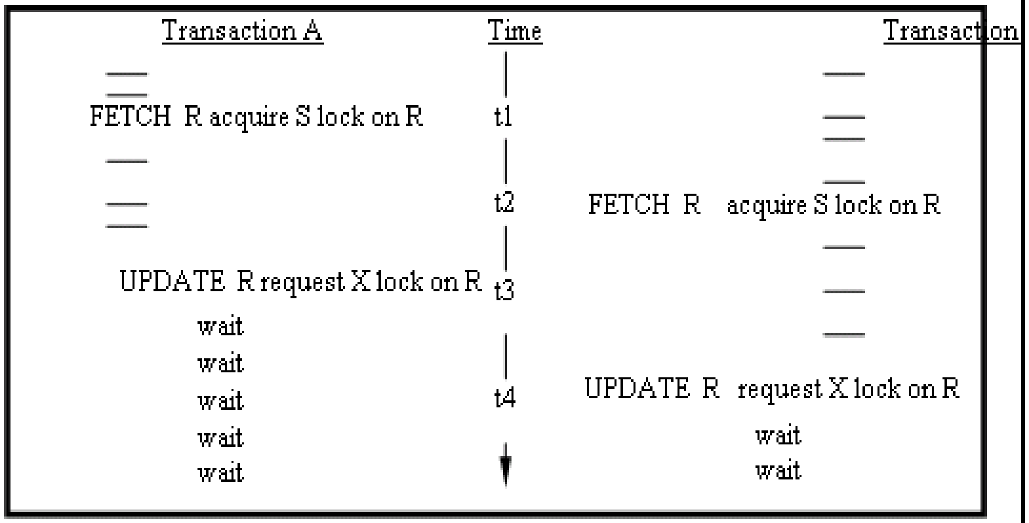
- When a transaction successfully retrieves a record, it automatically acquires an **S** lock on that record. When a transaction successfully updates a record, it automatically acquires an **X** lock on that record.
- **X** locks are held until the next synchpoint. Normally **S** locks are also held until the same time.

**Note** : A synchronization point (synchpoint) is established by executing either a COMMIT or a ROLLBACK operation and that is when:

- Updates made by the program since the previous Syncpoint are committed or undone (rolled-back).
- All records locks are released

This issue will be further discussed in the Recovery section.

## Solving Lost Update Problem


Transaction A
Time
Transaction
FETCH R acquire S lock on R
t1
t2
FETCH R acquire S lock on R
UPDATE R request X lock on R
t3
wait
wait
wait
t4
UPDATE R request X lock on R
wait
wait
wait
wait

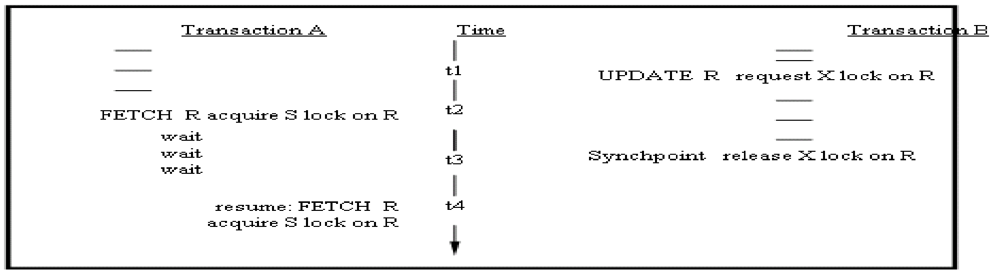
- **The Lost Update Problem:**

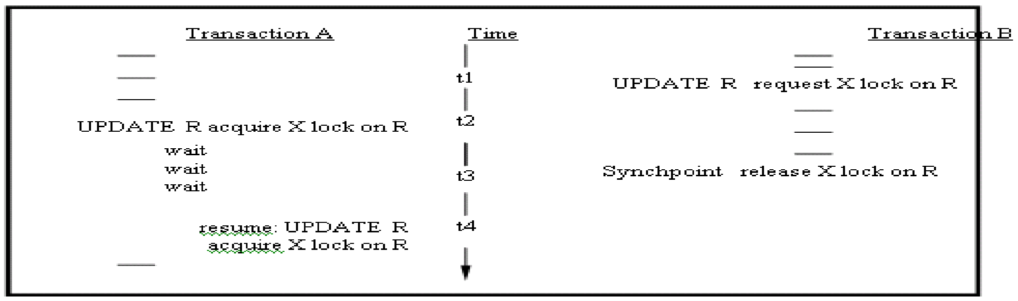
The figure is a modified version of the figure presented in the lost update section, it demonstrates the locking scheme explained previously. Although this technique solves the lost update problem, a new problem arises, namely the deadlock problem.

Transaction A's UPDATE at time t3 is not accepted, because it is an implicit request for an X lock on R, and such a request conflicts with the S lock already held by transaction B; so A goes into a wait state. B also goes into a wait state at time t4. The result is that both transactions are unable to proceed.

Even though the problem of lost updates is being solved, the system gets into a deadlock situation.

## Solving Uncommitted Dependency Problem


Transaction A
Time
Transaction B
t1
UPDATE R request X lock on R
FETCH R acquire S lock on R
t2
wait
wait
wait
t3
Synchpoint release X lock on R
resume: FETCH R
acquire S lock on R
t4


Transaction A
Time
Transaction B
t1
UPDATE R request X lock on R
UPDATE R acquire X lock on R
t2
wait
wait
wait
t3
Synchpoint release X lock on R
resume: UPDATE R
acquire X lock on R
t4

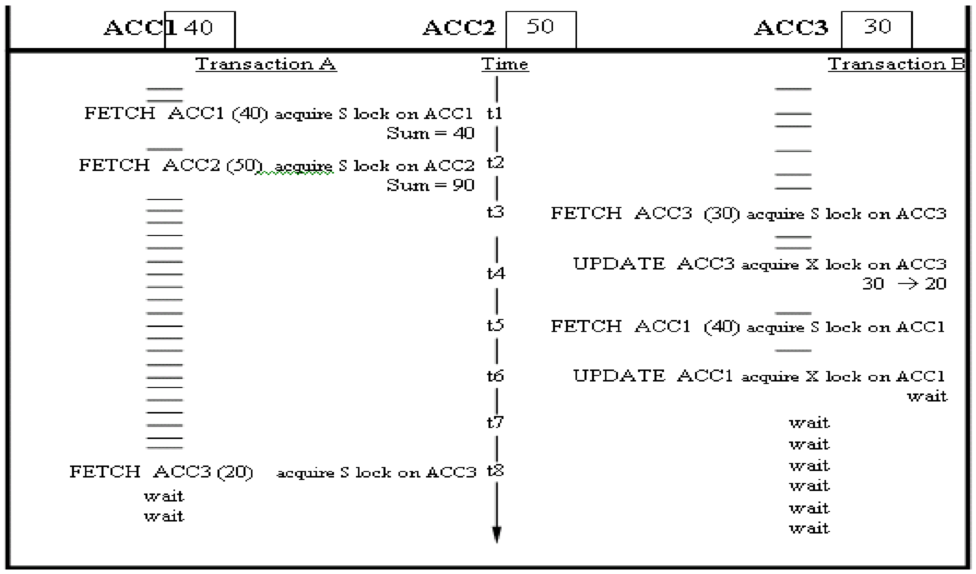
- **The Uncommitted Dependency Problem:**

The figures are modified versions of the two figures presented in the uncommitted dependency section using the locking scheme explained previously.

In the first figure, Transaction A is prevented from seeing an uncommitted change at time t2.

In the next figure, Transaction A is prevented from updating an uncommitted change at time t2.

## Solving Inconsistent Analysis Problem



ACC1 40 | ACC2 50 | ACC3 30

| Transaction A | Time | Transaction B |
|---|---|---|
| FETCH ACC1 (40) acquire S lock on ACC1 Sum = 40 | t1 | |
| FETCH ACC2 (50) acquire S lock on ACC2 Sum = 90 | t2 | |
| | t3 | FETCH ACC3 (30) acquire S lock on ACC3 |
| | t4 | UPDATE ACC3 acquire X lock on ACC3 30 → 20 |
| | t5 | FETCH ACC1 (40) acquire S lock on ACC1 |
| | t6 | UPDATE ACC1 acquire X lock on ACC1 wait |
| | t7 | wait |
| FETCH ACC3 (20) acquire S lock on ACC3 wait wait | t8 | wait wait wait wait wait |

- **The Inconsistent Analysis Problem:**

The figure is a modified version of the figure presented in the inconsistent analysis section using the locking scheme explained previously. Preventing both transactions from updating the records prevents inconsistent analysis, but a deadlock situation arises at time *t7*.

## Deadlock

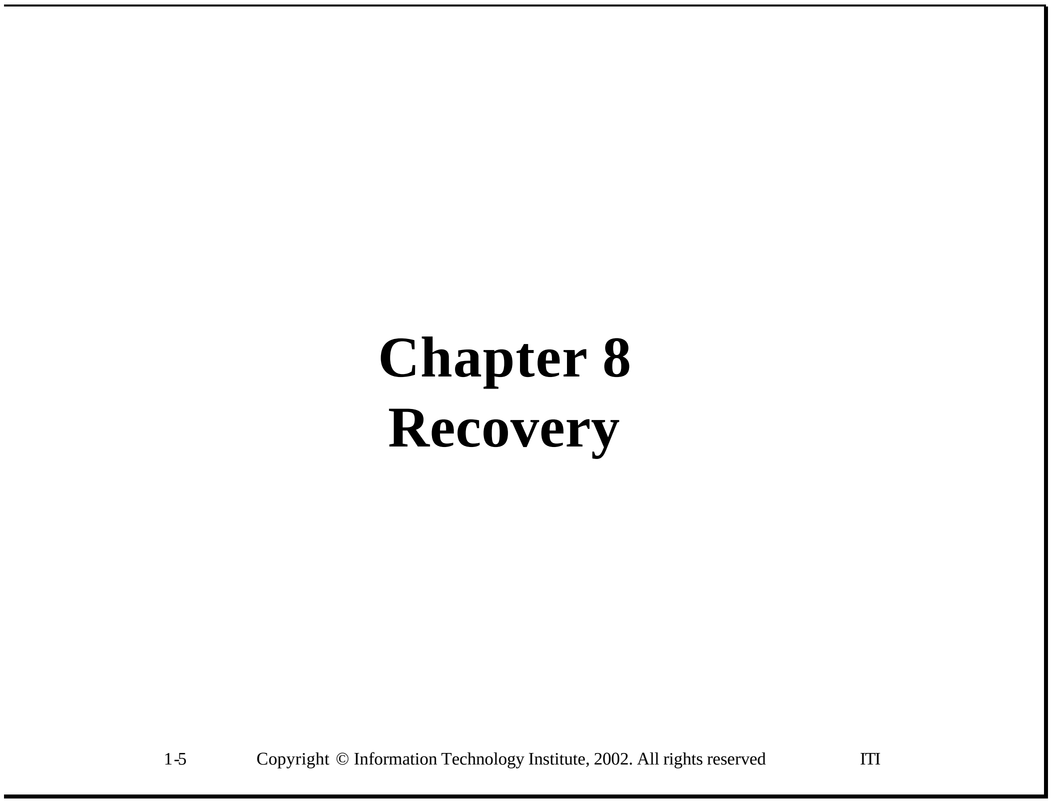
Deadlock is a situation in which two or more transactions are in a wait state at the same time.
Each transaction is waiting for one of the others to release a lock before it can continue running. The system should detect a deadlock situation and break it.

## Deadlock Prevention Protocol

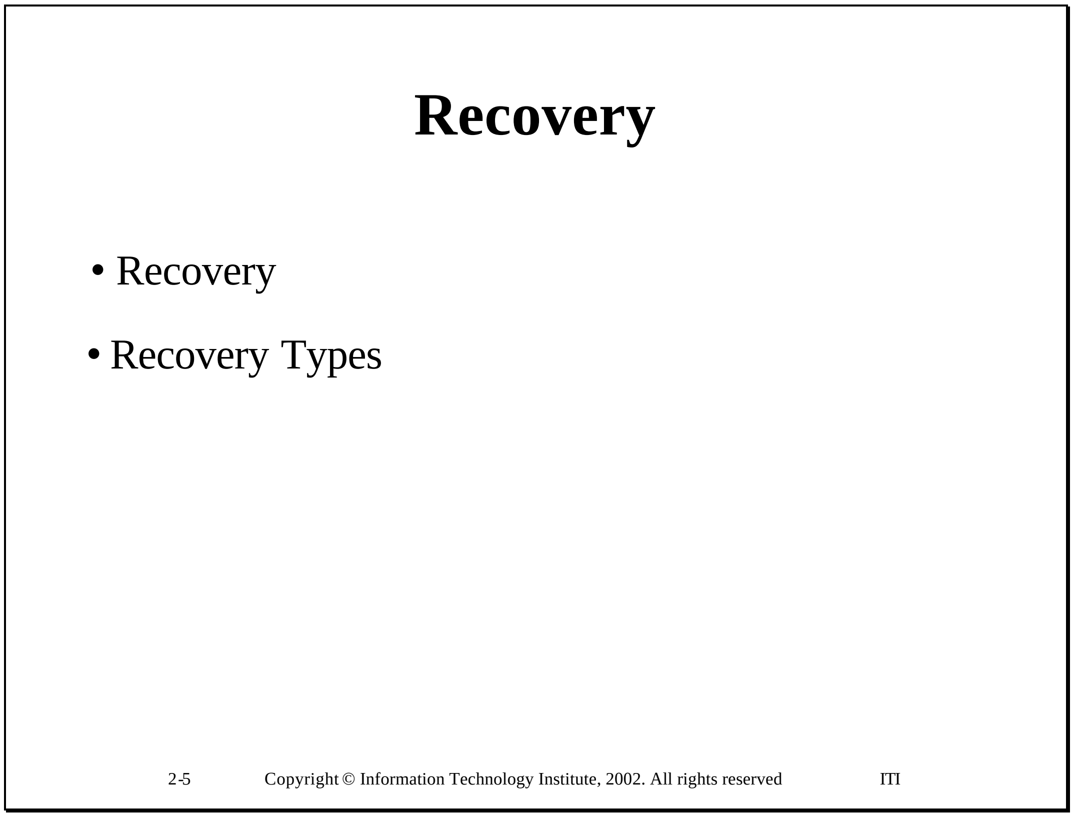
One way to prevent a deadlock is to use a deadlock prevention protocol. A number of deadlock prevention schemes have been proposed to make a decision about what to do with a transaction involved in a possible deadlock situation. Should the transaction be blocked and made to wait? Or should it be aborted? Or should the transaction preempt and another transaction aborted?
These techniques use the concept of transaction timestamp, which records the order in which the transactions are started. Hence, if transaction T1 starts before T2, then the timestamp of T1 < timestamp of T2 [TS (T1) < TS (T2)]. We can then decide which transaction to kill and which to continue based on the following schemes:

**Wait-die:** If TS (Ti) < TS (Tj), then Ti is older than Tj, so Ti is allowed to wait, otherwise, abort Ti (Ti dies) and restart it later with the same timestamp.

**Wound-wait:** If TS (Ti) < TS (Tj), then Ti is older than Tj. Tj is aborted (Ti wounds Tj) and restart later with the same timestamp, otherwise (Ti is younger than tj) Ti is allowed to wait.

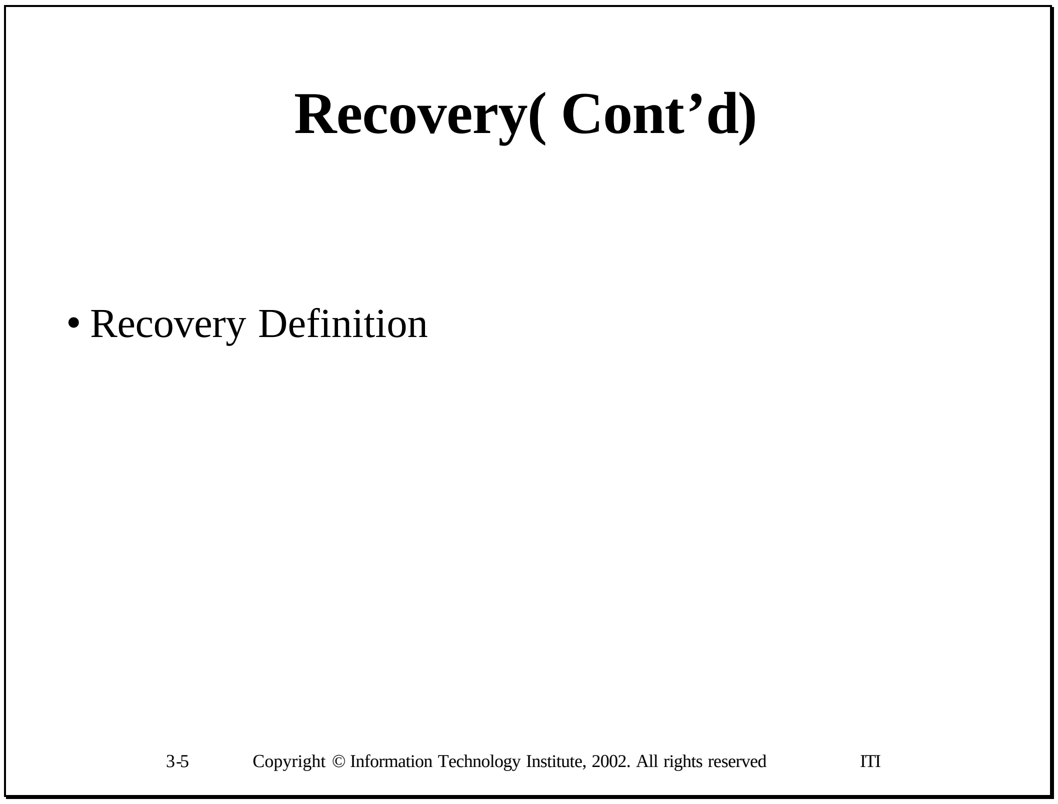


# Chapter 8
# Recovery



## Recovery

- Recovery
- Recovery Types



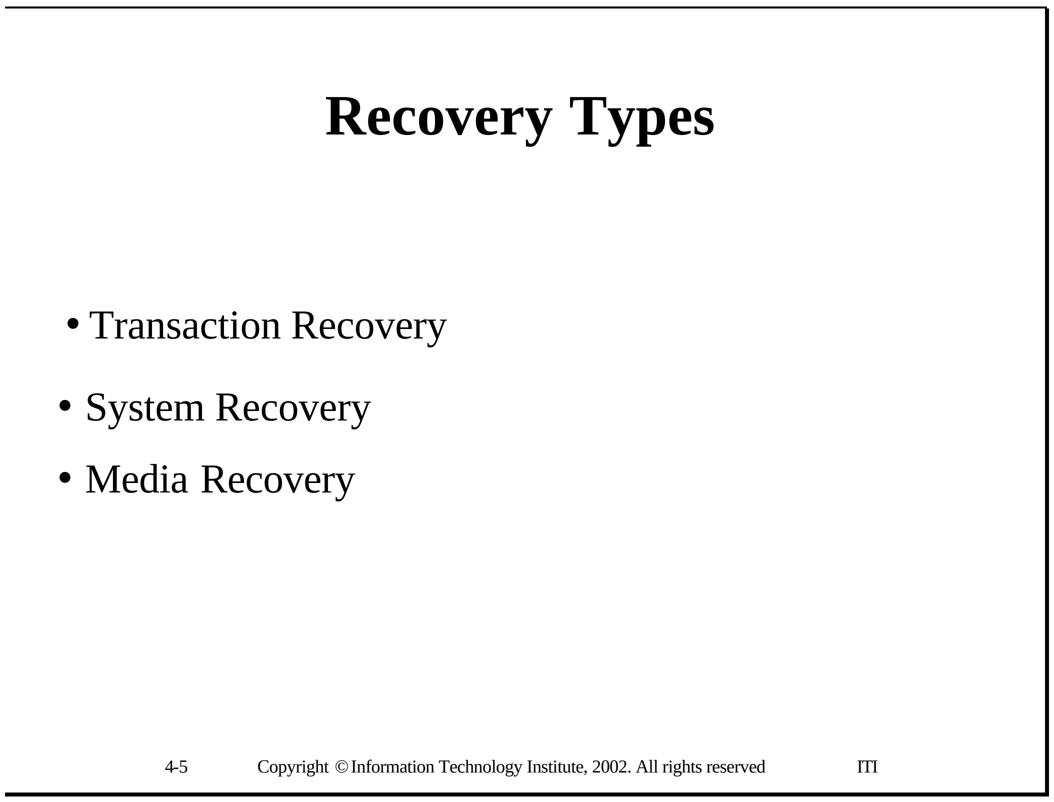
## Recovery( Cont'd)

- Recovery Definition

3-5 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Recovery Definition

Recovery is the process of returning the system into its previous healthy state. There are three types of recovery; transaction recovery, system recovery and media recovery.



## Recovery Types

- Transaction Recovery
- System Recovery
- Media Recovery

4-5 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Transaction Recovery

Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of the failure. To do this, the system must keep information about the changes that are being applied to data items by various transactions. This information is typically kept in the *system log file*.

Assume that a transaction that consists of two updates causes a system crash. Those updates should be both undone, so that the transaction is either executed in its entirety or is totally canceled.

The transaction manager is the system component that is responsible for maintaining this condition through the COMMIT and ROLLBACK operations:

- The COMMIT operation signals the successful end-of-transaction.
- The ROLLBACK operation signals unsuccessful end-of-transaction, so the transaction must be "rolled back" or “undone”.

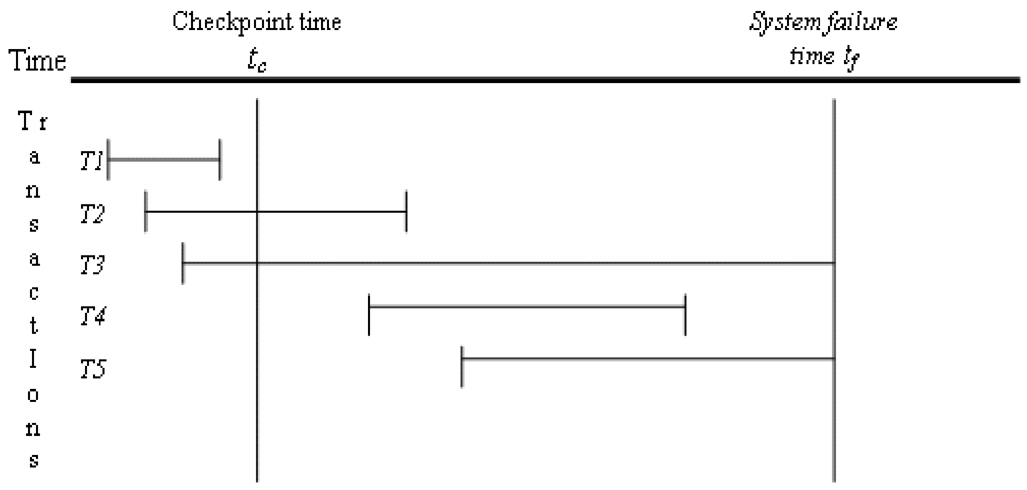
The system maintains a *log* with details of all update operations. The values of the updated objects before and after the update are saved in the corresponding log entry to be able to restore the updated object to its previous value.

**Synchronization Points:**
A Synchronization Point (Synchpoint) is established by executing either a *COMMIT* or a *ROLLBACK* operation. When a *Synchpoint* is established:

- All updates made by the pr ogram since the previous *Synchpoint* are committed (*COMMIT)* or undone *(ROLLBACK)*.
- All record locks are released.

**Note:** *COMMIT* and *ROLLBACK* terminate the transaction, not the program. *A transaction* is not only the logical unit of work but also the unit of recovery.

## System and Media Recovery



***Five transaction categories***

### System and Media Recovery

Global failures could occur in the system. Such failures fall into two categories:

- **System failures:**
  A system failure (such as a Power failure) affects all transactions cur rently in progress but does not physically damage the database. It is also known as soft crash.

When a system failure occurs, the content of main storage is lost. The transactions that were completed before the system failure *(COMMITED ones)* must be *redone* after restarting the system, while those transactions that were not completed must be *undone* (ROLLED BACK).

The system writes the database buffers out physically on the storage every duration of time. This is called the checkpoint. The figure above illustrates how this information is used:

- A system failure has occurred at time *tf*
- A most recent checkpoint prior to *tf* was taken at time tc

- Transactions of type *T1* were completed prior to time *tc*
- Transactions of type *T2* started prior to time *tc* and completed after time *tc* and before time *tf*
- Transactions of type *T3* started prior to time *tc* but were not complete by time *tf*
- Transactions of type *T4* started after time *tc* and completed before time *tf*
- Transactions of type *T5* started after time *tc* but were not complete by time *tf*

At restart time, the system goes through the following procedure in order to identify all transactions of type *T2 - T5*:

♦ Start with two lists of transactions, the UNDO list and the REDO list.Set the UNDO = list of ***all transactions given*** in the checkpoint record.
Set the REDO = ***empty***.

♦ Search forward through the log, starting from the checkpoint record

♦ If a *"begin transaction"*, log entry is found for transaction T, add T to the UNDO list.

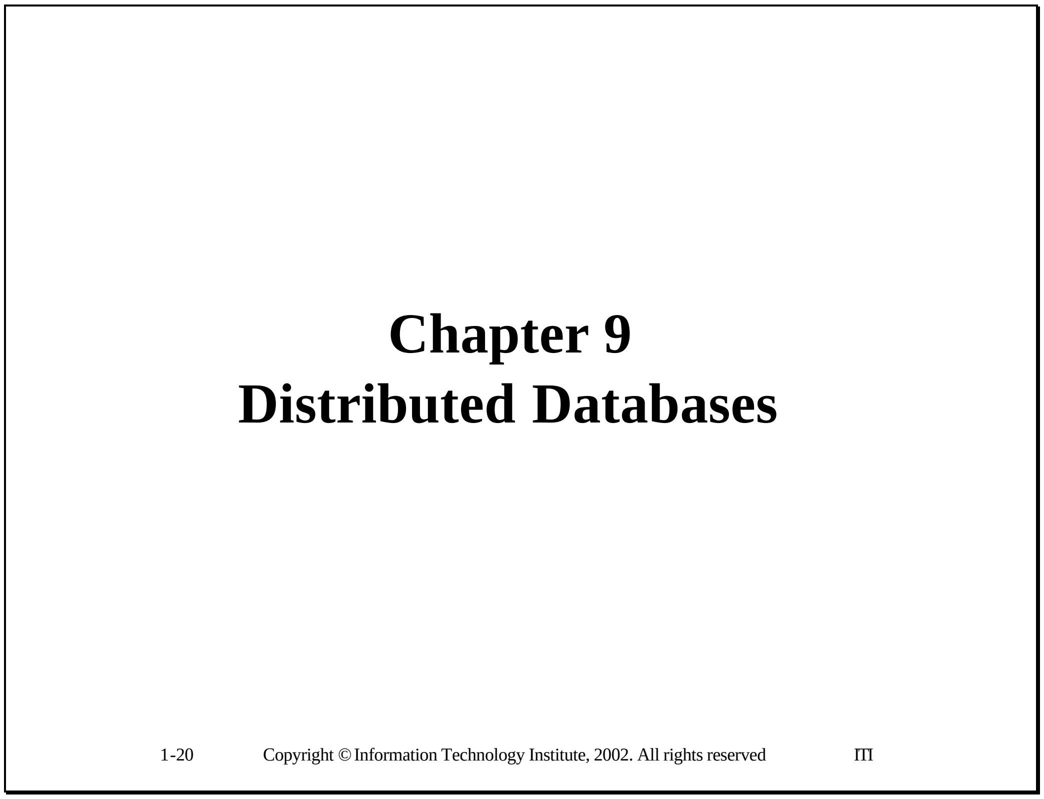
♦ If a *"Commit"* log entry is found for transaction T, move T from the UNDO list to the REDO list.

♦ When the end of the log is reached, the UNDO list identifies transactions of types *T3* and *T5*, while the REDO list identifies transaction of types *T2* and *T4*.

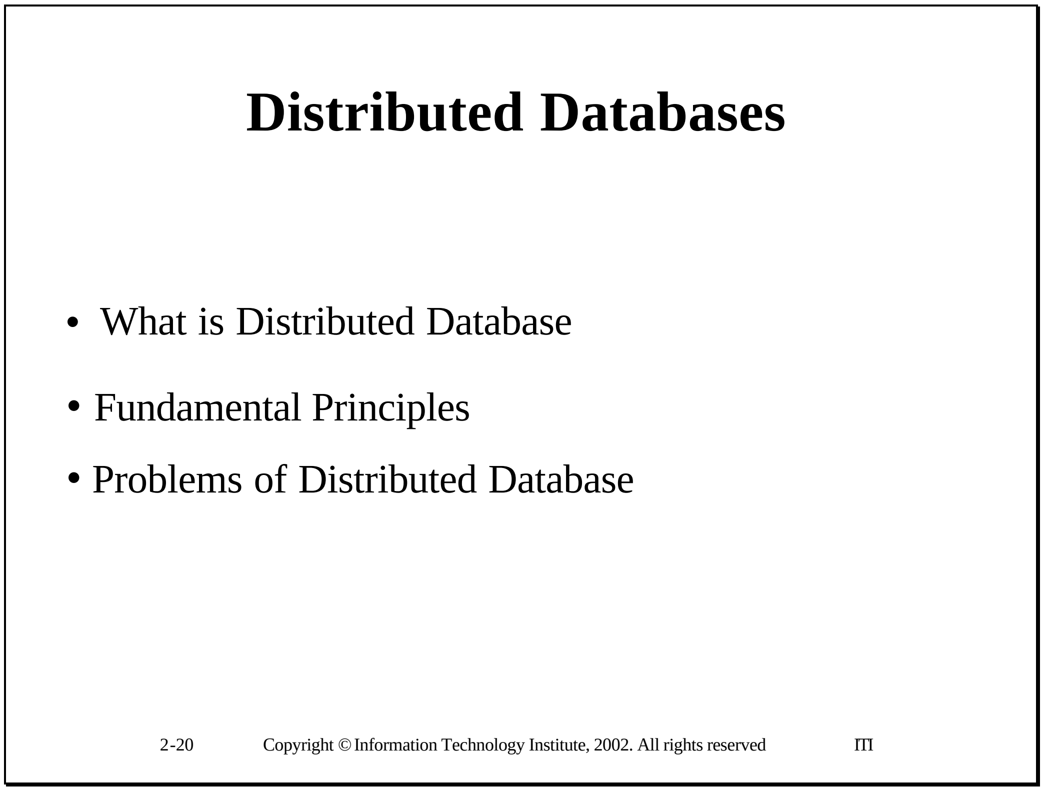
- **Media failures:**

A media failure (such as a head crash on the disk) does cause damage to the database, or to some portion of it, and affects at least those transactions currently using that portion. A media failure is also known as a hard crash.

Recovery from such a failure basically involves reloading the database from a *backup copy*, and then using the log to redo all transactions that were completed since that backup copy was taken.



# Chapter 9 Distributed Databases

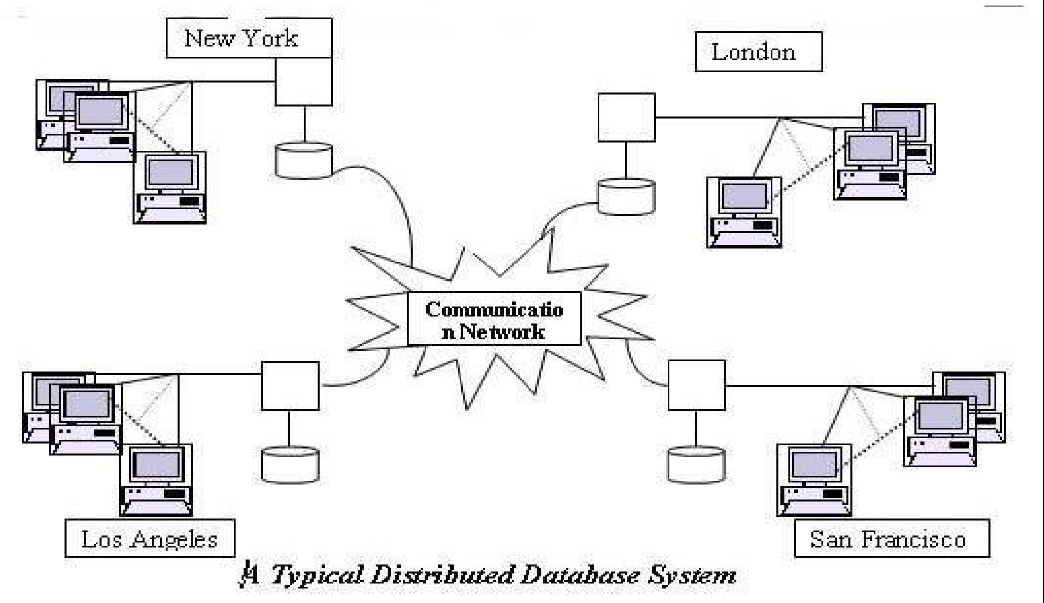


## Distributed Databases

- What is Distributed Database
- Fundamental Principles
- Problems of Distributed Database

2-20 ITI

## What is Distributed Database



A Typical Distributed Database System

A distributed database system consists of a group of database system sites, connected together via a communications network. A user at any site can access data anywhere in the network as if the data were all stored at the user's own site. The figure shows an example of a distributed database.

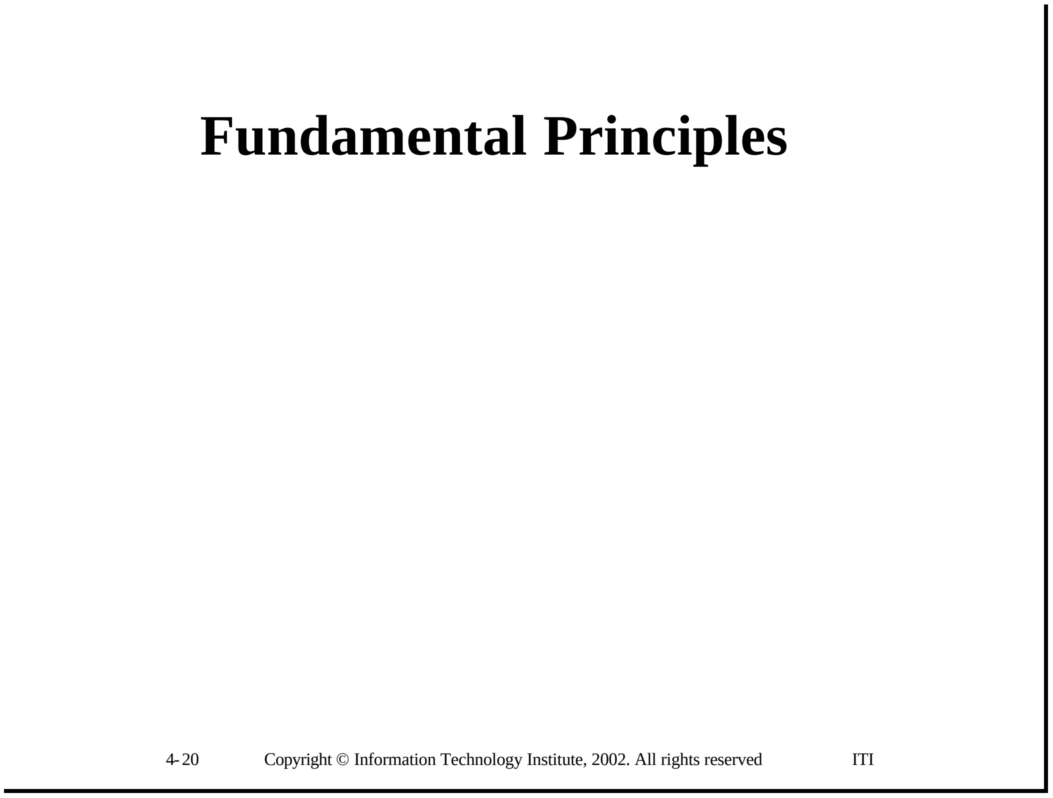
Accordingly, the distributed database system can be regarded as a kind of partnership among the individual local DBMSs at the individual local sites. A new software component at each site provides the necessary communication functions. It is the combination of these new components together with the existing DBMS that constitutes the Distributed Database Management System, DDBMS.

Full support for distributed database enables a single application to operate on data that is spread across a variety of different databases, managed by a variety of different DBMSs, running on a variety of different machines, supported by a variety of different operating systems, and connected together by a variety of different communication networks.

**Why Distributed Database is Important?**

- A distributed system enables the structure of the database to mirror the structure of the enterprise.
- Distributed database provides the ability to replicate the data from one site to another.
- A distributed database breaks down big, unmanageable problem into smaller pieces.

The main disadvantage of distributed database systems is that distributed systems are complex and more costly.



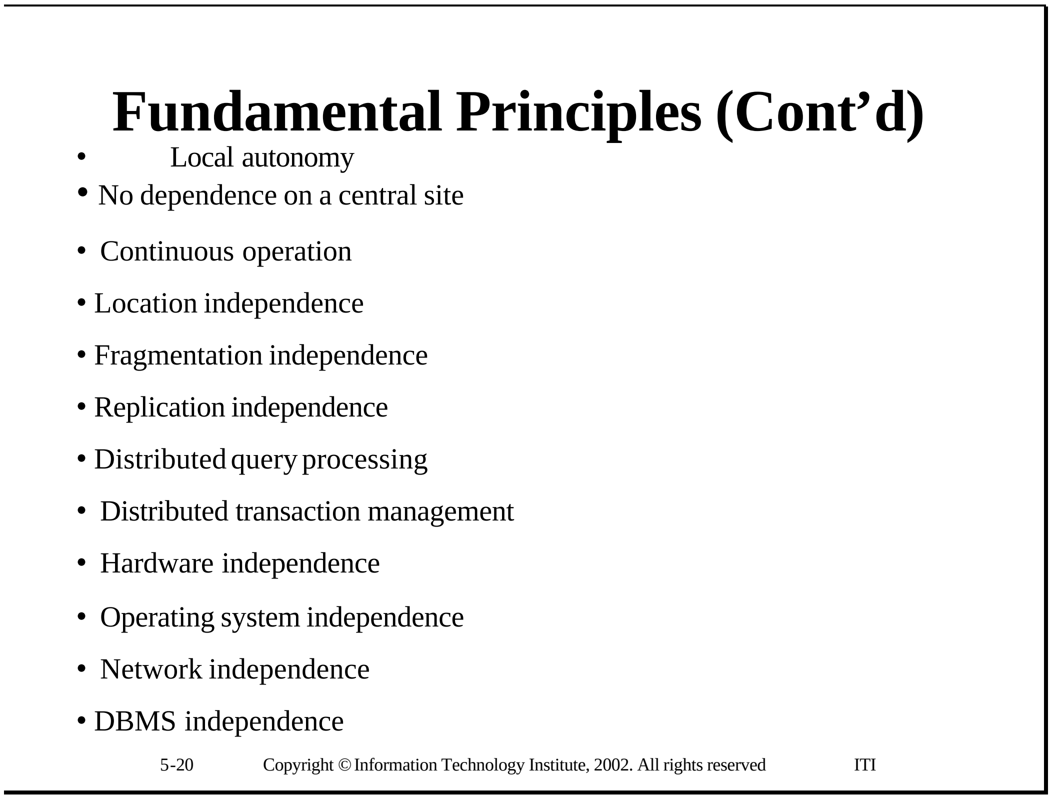
## Fundamental Principles

4-20 ITI

### Fundamental Principles

A more point of any distributed database system is that a user will view, the “distributed system exactly like a non-distributed system”.

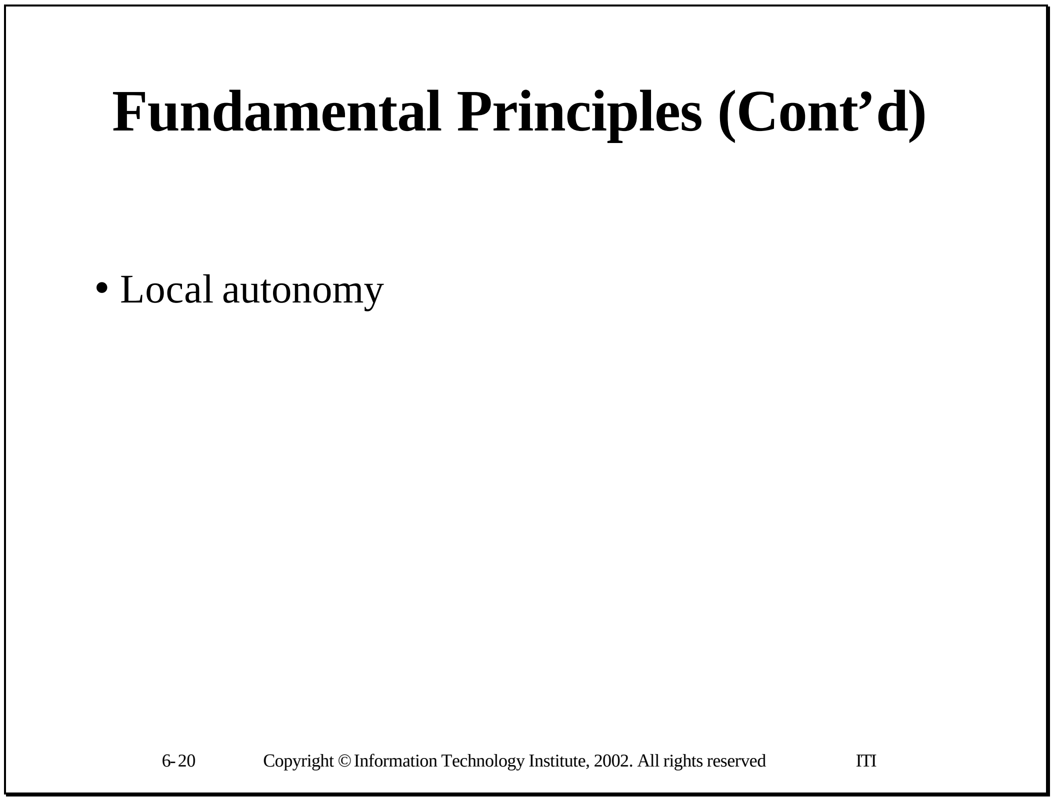
Other fundamental principles for distributed databases are mentioned in the following.



## Fundamental Principles (Cont’d)

- Local autonomy
- No dependence on a central site
- Continuous operation
- Location independence
- Fragmentation independence
- Replication independence
- Distributed query processing
- Distributed transaction management
- Hardware independence
- Operating system independence
- Network independence
- DBMS independence

5-20 ITI



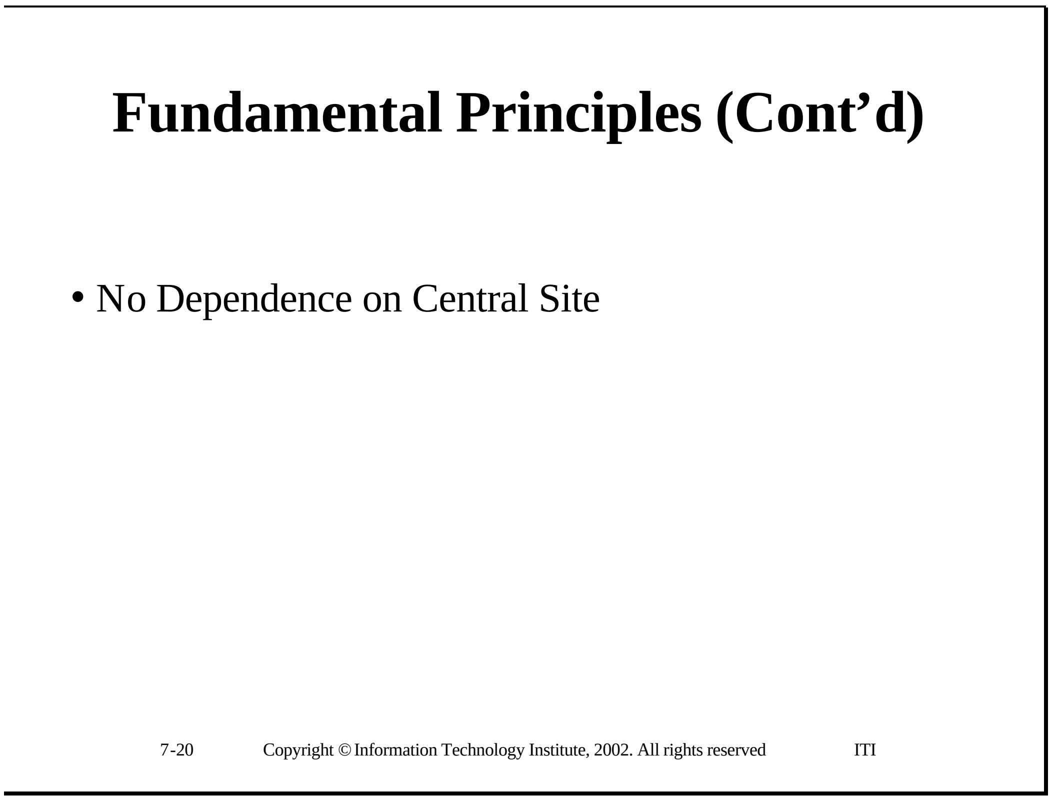
## Fundamental Principles (Cont'd)

- Local autonomy

### Local autonomy

Local autonomy means that all operations at a given site are controlled by that site. A site X for example should not depend on another site Y for its successful functioning.

Local autonomy also implies that local data is owned and managed locally. Data belonging to some local database, may be accessible from some remote site. Issues like security, integrity, and storage representation of local data still remain under the control of the local site.



## Fundamental Principles (Cont’d)

- No Dependence on Central Site

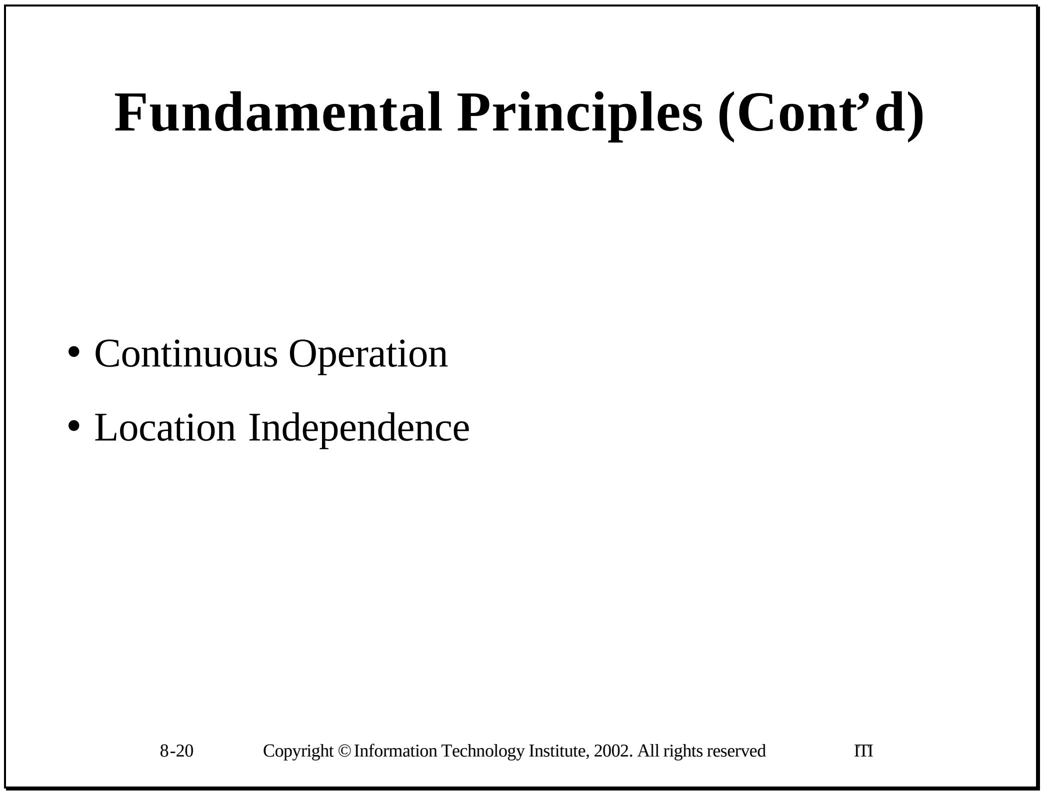
7-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

**No dependence on a central site:**

Local autonomy implies that all sites must be treated as equals; there is no reliance on a central site for any central service, such as centralized query processing or centralized transaction management that would make the entire system dependent on a single site.

Dependence on a central site has the following disadvantages:

- A bottleneck may occur at the central site.
- The system would be unsafe; the whole system could be down if a failure occurs in the central site.



## Fundamental Principles (Cont'd)

- Continuous Operation
- Location Independence

8-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

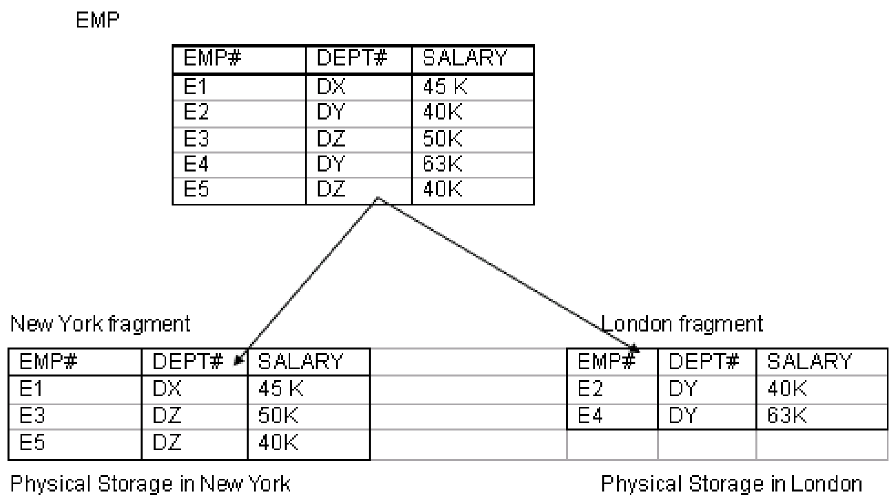
### Continuous operation

In a distributed system, just as in a non-distributed one, there should be no need for any planned system outage. To perform some functions, such as adding a new site, or upgrading the DBMS at an existing site to a new release.

### Location independence

Users do not have to know where data is physically stored, the system should be able to behave as if the data was stored at the users' local site. The system, thus, allows data to migrate from site to site. Such migratability is desirable because it allows data to be moved around the network in response to changing performance requirements.

## Fragmentation Independence



EMP

| EMP# | DEPT# | SALARY |
|---|---|---|
| E1 | DX | 45 K |
| E2 | DY | 40K |
| E3 | DZ | 50K |
| E4 | DY | 63K |
| E5 | DZ | 40K |

New York fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E1 | DX | 45 K |
| E3 | DZ | 50K |
| E5 | DZ | 40K |

London fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E2 | DY | 40K |
| E4 | DY | 63K |

**Fig. 6.2 An Example of Fragmentation**

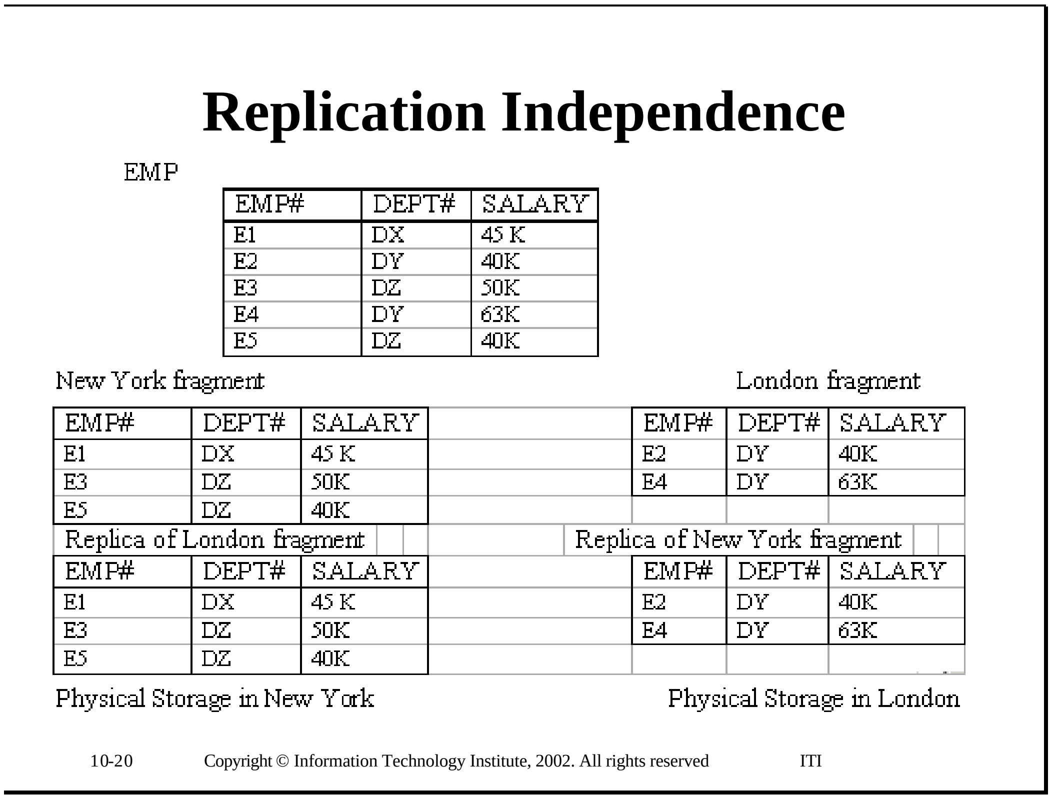
### Fragmentation independence:

A system supports data fragmentation if a given relation can be divided up into fragments for physical storage purposes. Fragmentation is desirable for performance reasons. Data can be stored at the location where it is most frequently used, so that most operations are purely local, and network traffic is reduced.
The figure shows an example of fragmentation.

A system that supports data fragmentation should also support fragmentation independence, i.e. users should be able to use the system as if the data was in fact not fragmented at all.

There are two basic types of fragmentation, horizontal and vertical. These correspond to the relational operations of restriction and projection respectively.

In general, a fragment can be any arbitrary sub-relation that is derivable from the original relation via restriction and projection operations. Reconstructing the original relation from the fragments is done via suitable join and union operations.



## Replication Independence

EMP

| EMP# | DEPT# | SALARY |
|---|---|---|
| E1 | DX | 45 K |
| E2 | DY | 40K |
| E3 | DZ | 50K |
| E4 | DY | 63K |
| E5 | DZ | 40K |

New York fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E1 | DX | 45 K |
| E3 | DZ | 50K |
| E5 | DZ | 40K |

London fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E2 | DY | 40K |
| E4 | DY | 63K |

Replica of London fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E1 | DX | 45 K |
| E3 | DZ | 50K |
| E5 | DZ | 40K |

Replica of New York fragment

| EMP# | DEPT# | SALARY |
|---|---|---|
| E2 | DY | 40K |
| E4 | DY | 63K |

Physical Storage in New York

Physical Storage in London

10-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

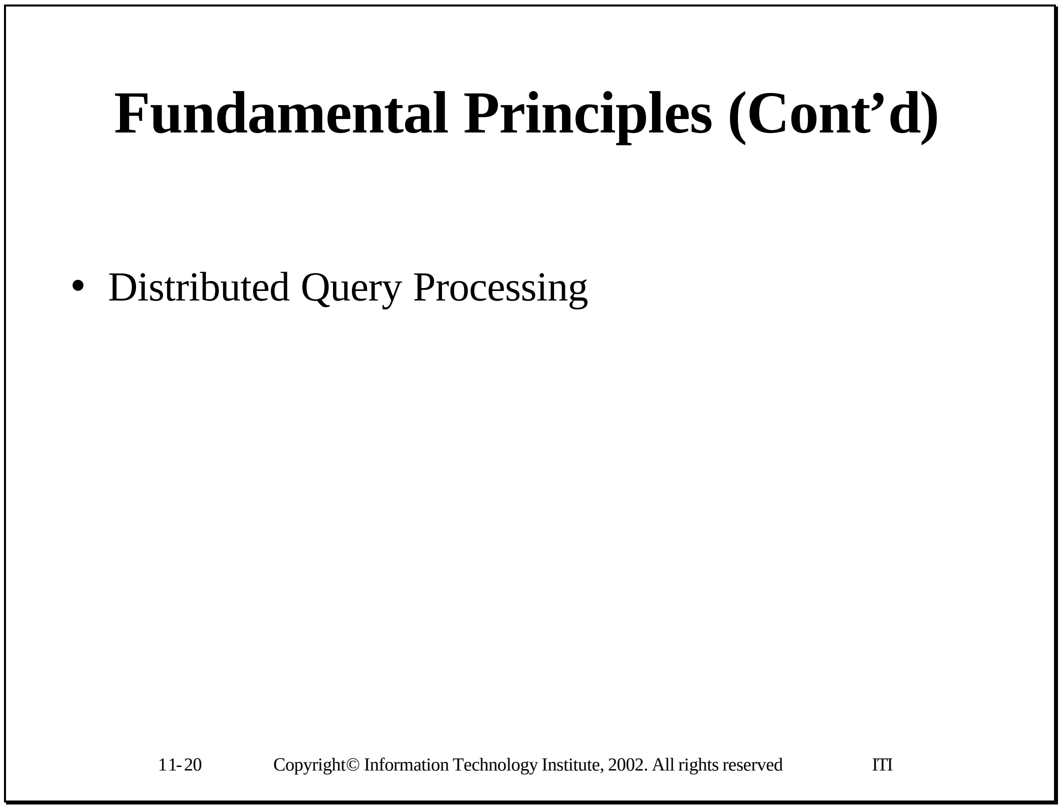
### Replication independence

A system supports data replication if several distinct copies or replicas of a given relation can be stored at distinct sites on the physical level. Replication is desirable for two reasons:

- It leads to better performance
- It causes better availability.

The major drawback of replication is the need to update all copies of a replicated object after any modification.

A system that supports data replication should also support replication independence, i.e. users should be able to use the system as if the data was in fact not replicated at all.



## Fundamental Principles (Cont’d)

- Distributed Query Processing

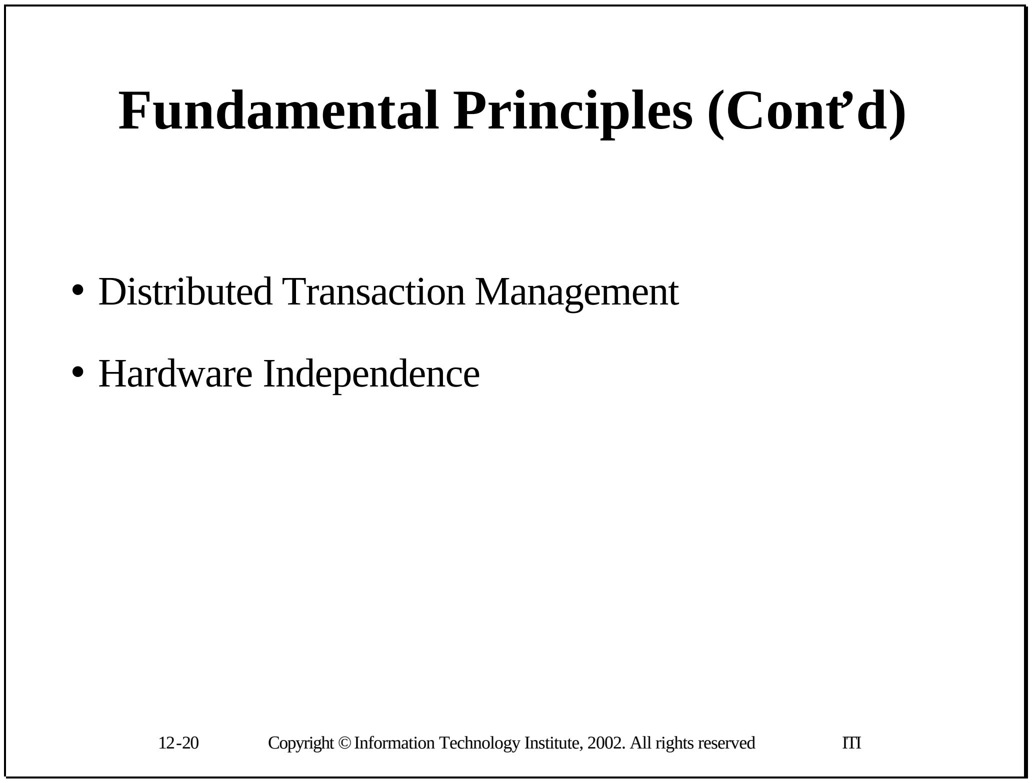
11-20 Copyright© Information Technology Institute, 2002. All rights reserved ITI

### Distributed query processing

Consider the query “find London suppliers of red parts. ” Assume that the user is at the New York site and the data is at the London site. Assume also that there are n supplier records that satisfy the request. If the system is relational, the query will basically involve two messages (one to send the request from New York to London, and one to return the result set of n record from London to New York). If, on the other hand, the system is record-at-a-time, the query will basically involve 2n messages (n from New York to London requesting the next record and n from London to New York to return that next record).

The example illustrates that a relational system is likely to outperform a non-relational one, so this is another reason why distributed systems are always relational.

Optimization is even more important in distributed systems than in centralized systems. In a query that involves multiple sites, there are many possible ways of moving data around the network in order to satisfy the request, and it is crucially important that an efficient strategy be followed.


## Fundamental Principles (Cont'd)

- Distributed Transaction Management
- Hardware Independence

12-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

**Distributed transaction management:**

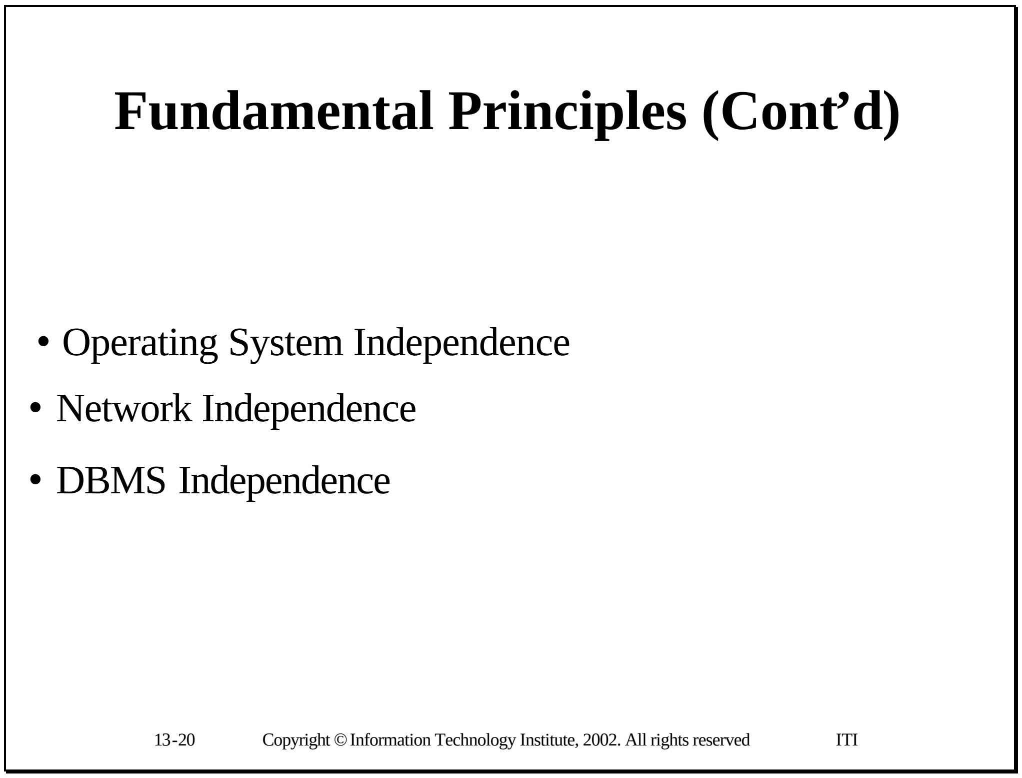
There are two main aspects to transaction management: recovery control and concurrency control. First it is necessary to introduce the new term "agent", it is the process performed on behalf of a given transaction at a given site.

**Recovery control:** In order to ensure that a given transaction is atomic (all-or-nothing) in the distributed environment, the system must ensure that the set of agents for that transaction either all commit in unison or all roll back in unison. This effect can be achieved by means of the two-phase commit protocol.

**Concurrency control:** Concurrency control in a distributed system will almost certainly be based on locking, just as it is in non-distributed systems.

## Hardware independence:

Real world computer installations typically involve a number of different machines. The data on all of those systems needs to be integrated and presented to the user as a "single-system image". Thus it is necessary to be able to run the same DBMS on different hardware systems that participate as equal partners in a distributed system.



## Fundamental Principles (Cont’d)

- Operating System Independence
- Network Independence
- DBMS Independence

13-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

## Operating system independence

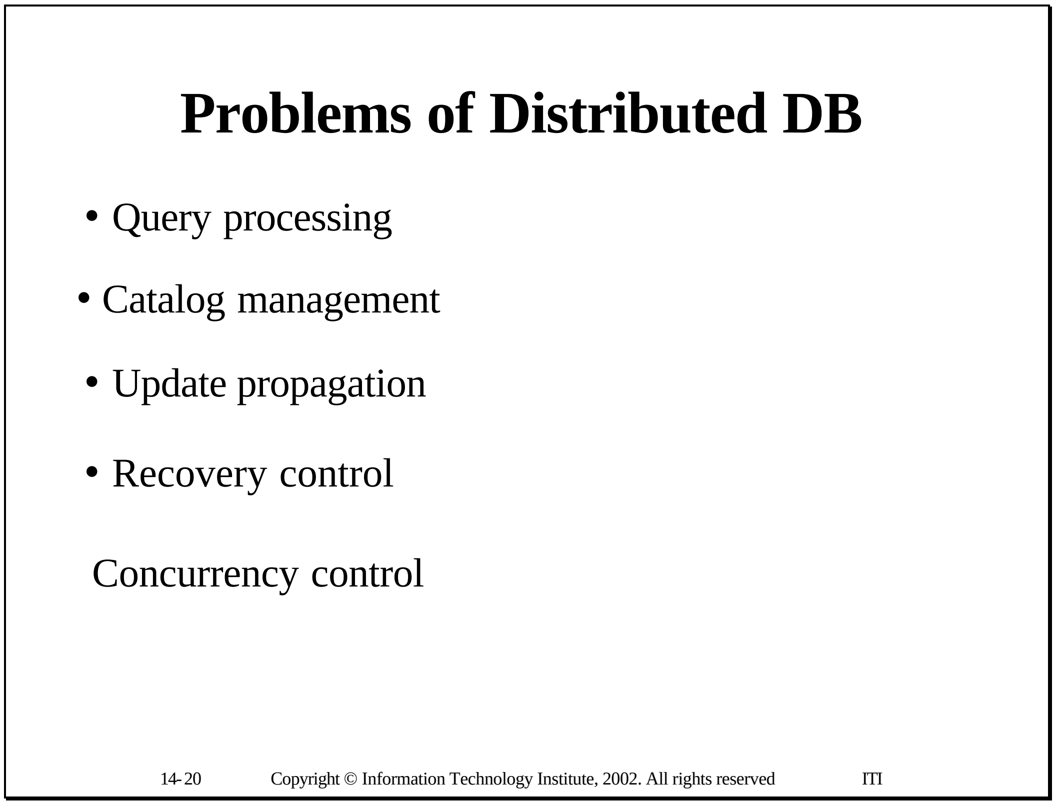
It is obviously desirable, to run the same DBMS on different hardware systems, and also to run it on different operating systems, whether on the same hardware or not.

### Network independence

If the system was able to support multiple sites, with different hardware and different operating systems, it would also be required to support a variety of different communication networks.

### DBMS independence

It might be possible for the distributed system to be heterogeneous, at least to some degree. Real-world computer installations typically run not only on different machines and different operating systems; they oftenly run different DBMS systems which participates somehow in the distributed system.



## Problems of Distributed DB

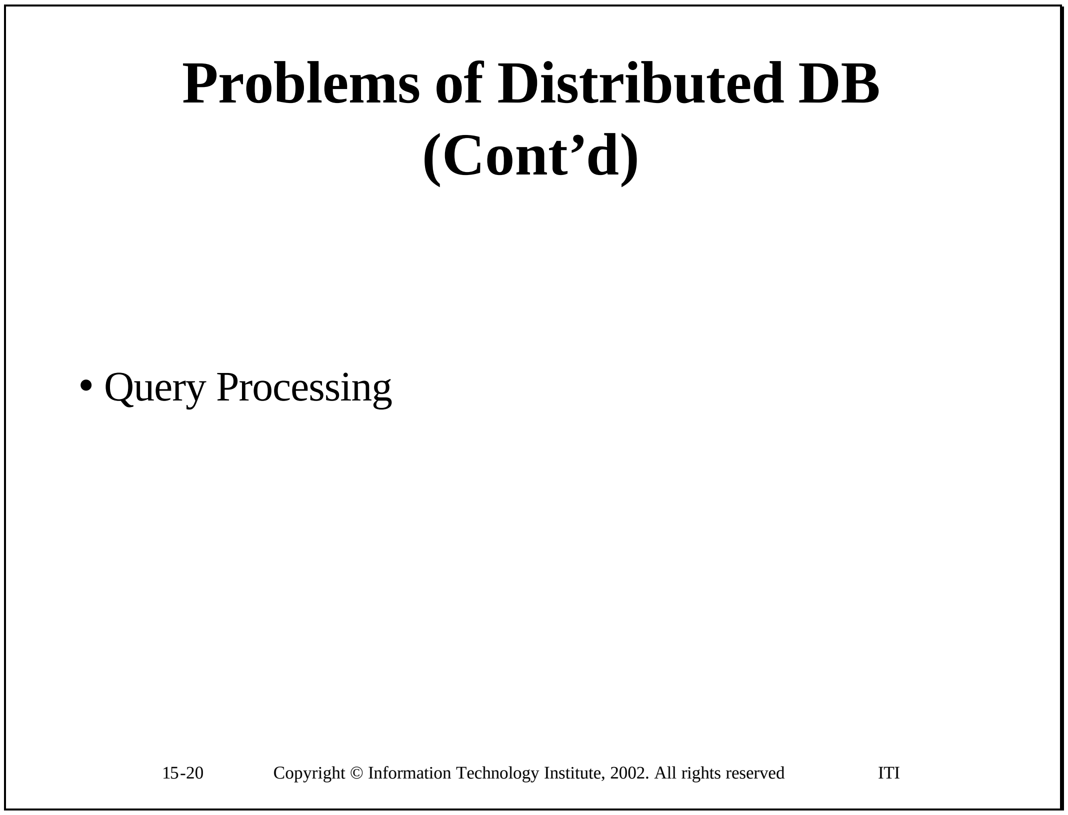
- Query processing
- Catalog management
- Update propagation
- Recovery control

Concurrency control

14-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

One of the objectives of distributed database systems is to minimize the number and volume of messages. Some of the issues involved are:

- Query processing
- Catalog management
- Update propagation
- Recovery control
- Concurrency control

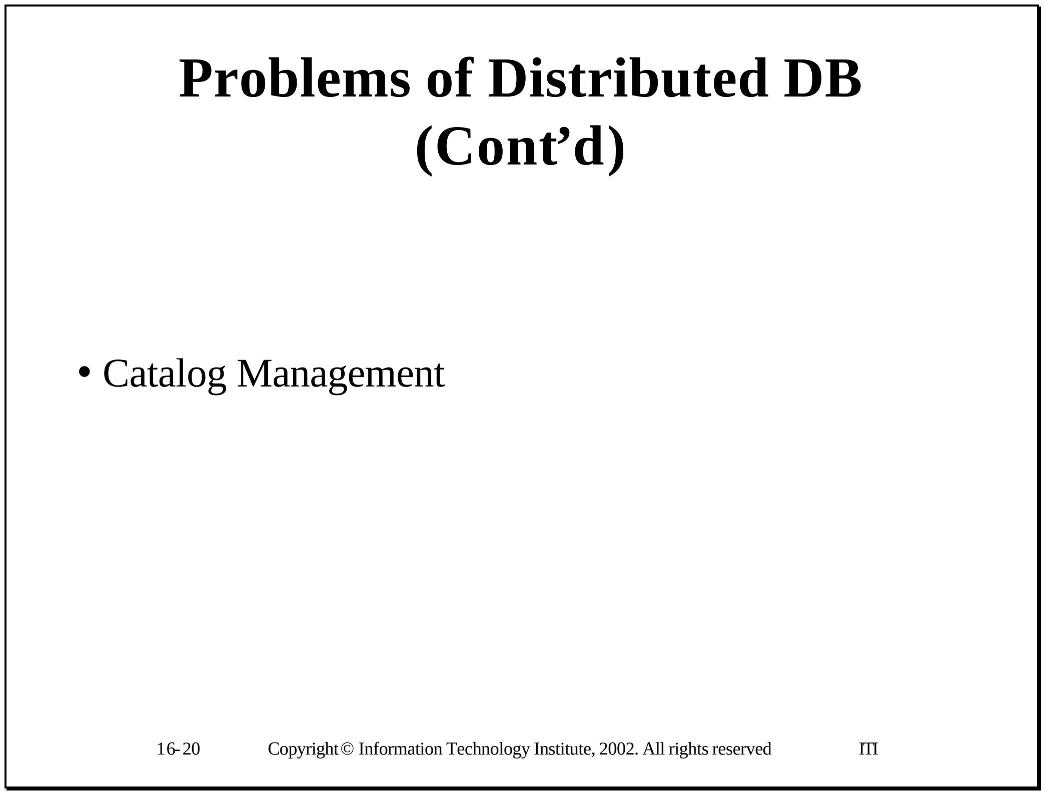


## Problems of Distributed DB (Cont'd)

- Query Processing

### Query Processing

The objectives of minimizing network traffic imply that the query optimization process itself needs to be distributed as well as the query execution process. In other words, the process will typically consist of a global optimization step, followed by local optimization steps at each affected site.



## Problems of Distributed DB (Cont'd)

• Catalog Management

### Catalog Management

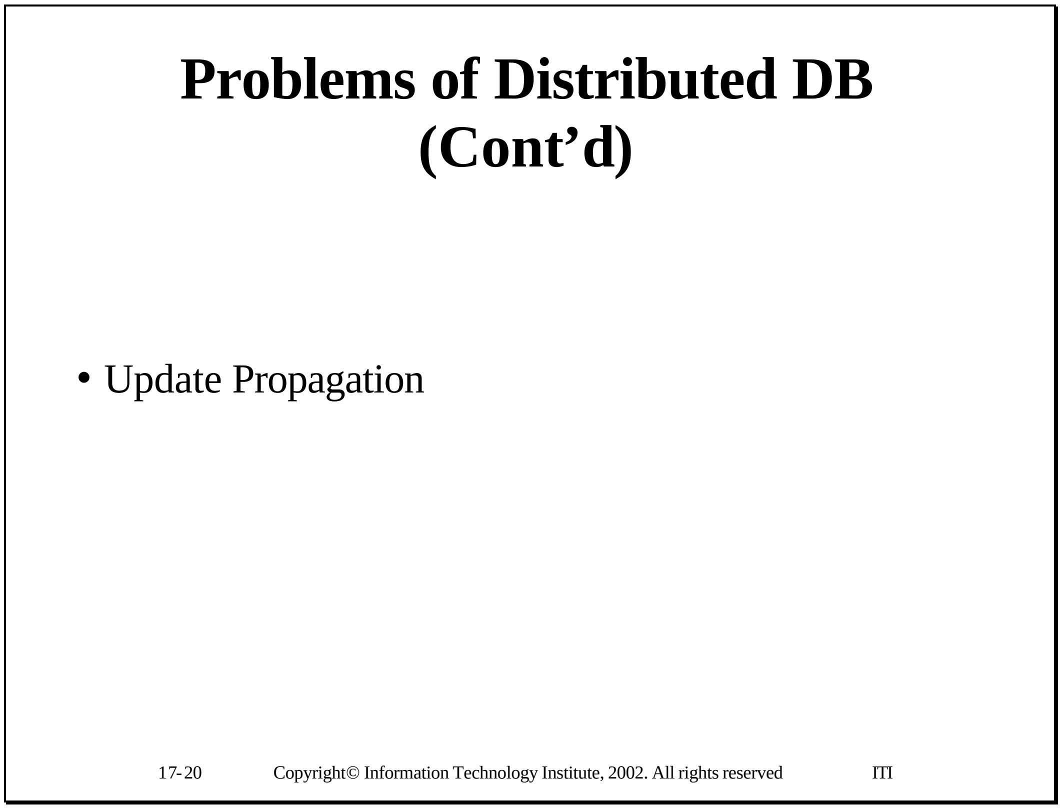
In a distributed system, the system catalog will include in addition to the usual catalog data describing the relations, indexes, users, etc., the necessary control information to enable the system to provide the desired location, fragmentation, and replication independence. The question now is: Where and how should the catalog itself be stored? Here are some of the alternatives:

- Centralized: The entire catalog is stored exactly once, at a single central site. This approach violates the objective of "no reliance on a central site".

- Fully replicated: The total catalog is stored in its entirety at every site. This approach suffers from a severe loss of autonomy, in that every catalog update has to be propagated to every site.

- Partitioned: Each site maintains its own catalog for objects stored at that site. The total catalog is the union of all of those local catalogs. This approach makes non-local operations very expensive.
- Combination of Centralized and Partitioned: Each site maintains its own local catalog, in addition, a single central site maintains a unified copy of all of those

local catalogs. This approach is more efficient than Partitioned catalog, but violates the “no reliance on a central site” objective again.



## Problems of Distributed DB (Cont’d)

• Update Propagation

17-20 Copyright© Information Technology Institute, 2002. All rights reserved ITI

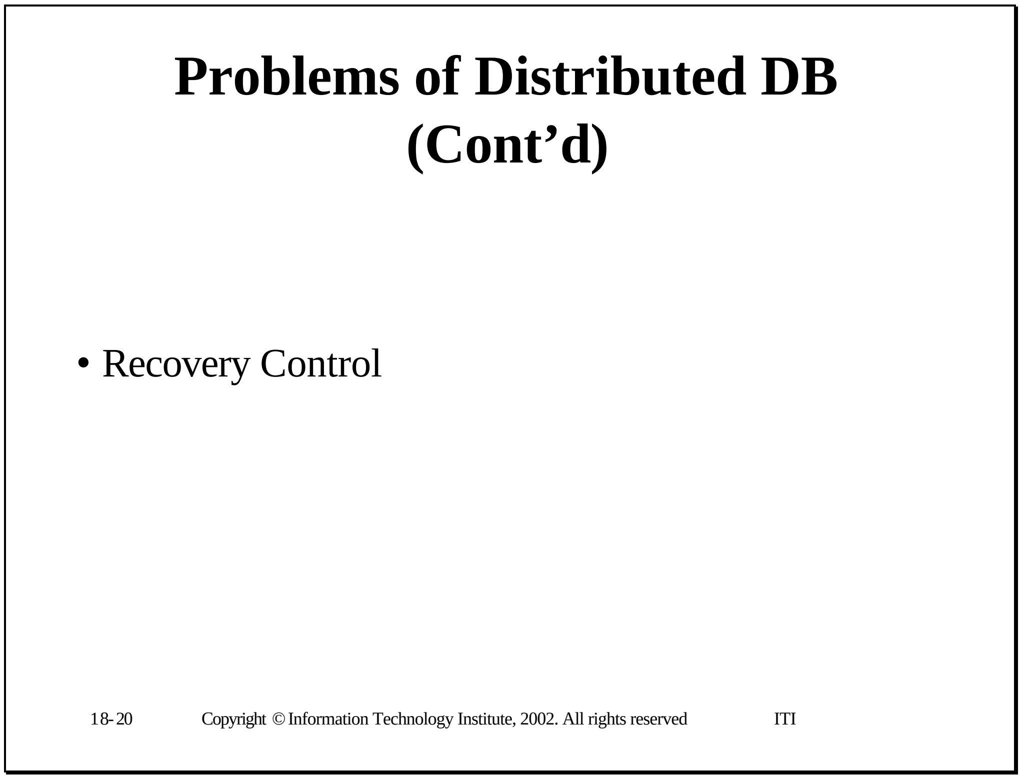
### Update Propagation

An update to any given logical object must be propagated to all stored copies of that object. A difficulty that arises immediately is that some sites holding a copy of the object might be unavailable at the time of the update. The obvious strategy of propagating updates immediately to all copies might thus be unacceptable, because it implies that the update will fail if any one of those copies is currently unavailable.

A common scheme for dealing with this problem is the so-called “primary copy” scheme, which works as follows:

One copy of each replicated object is designated as a primary copy. The others are all secondary copies. Primary copies of different objects are at different sites.

Update operations are assumed to be complete as soon as the primary copy has been updated. The site holding that copy is then responsible for propagating the update to the secondary copies at some other time.



## Problems of Distributed DB (Cont'd)

• Recovery Control

18-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

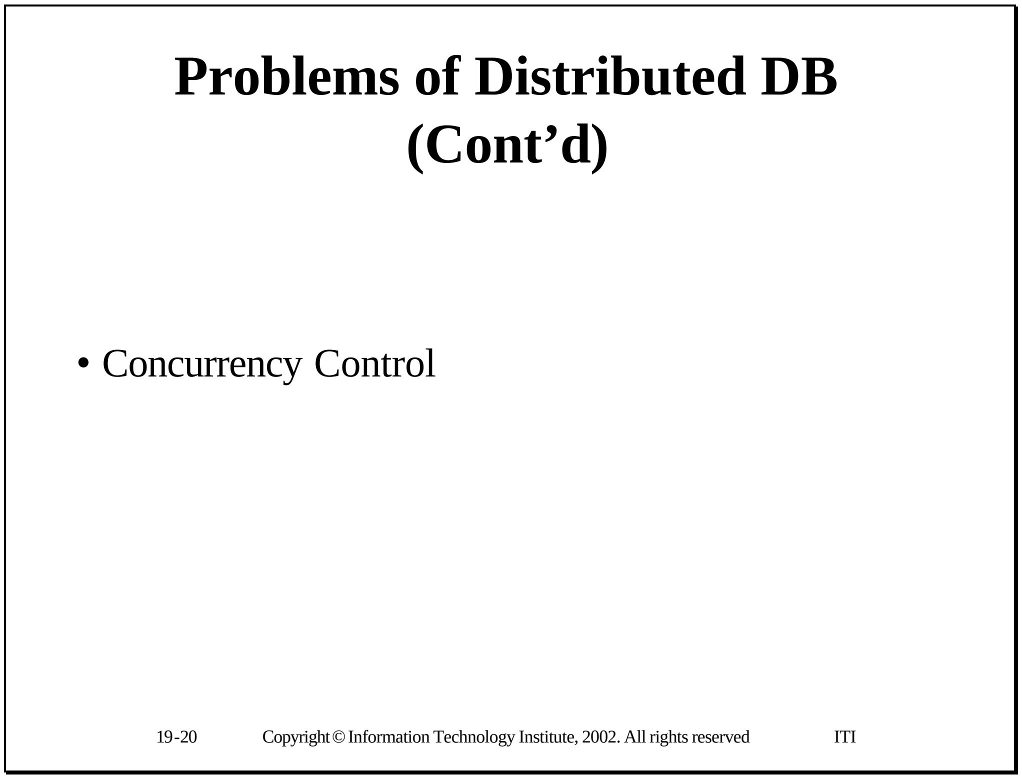
### Recovery Control

Recovery control in distributed systems is typically based on the two-phase commit protocol. Two-phase commit is required in any environment in which a single transaction can interact with multiple autonomous resource managers. As a result, the following points rise:

- The "no dependence on a central site" objective imposes that the coordinator function must be performed by different sites for different transactions rather than assigned to one distinguished site in the network.

- The two-phase commit process requires the coordinator to communicate with every participant site, which means more messages and more overheads.

- If site Y acts as a participant in a two-phase commit process coordinated by site X, then site Y must do what site X impose which results in a loss of local autonomy.



## Problems of Distributed DB (Cont’d)

• Concurrency Control

19-20 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Concurrency Control

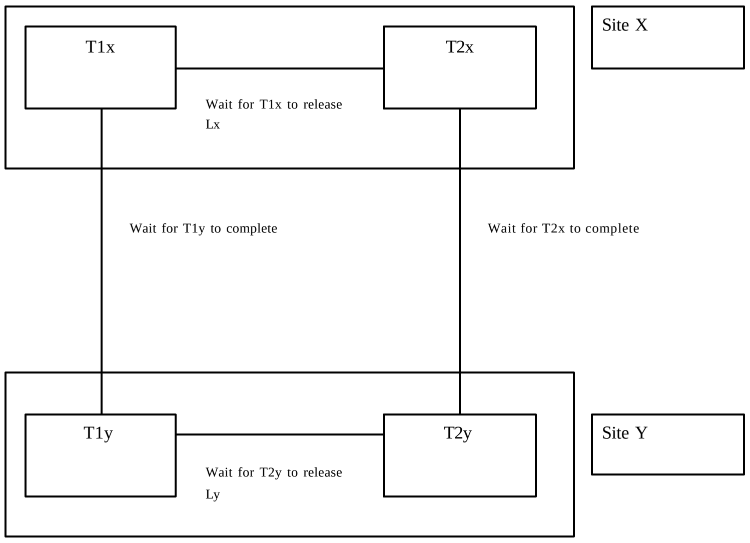
Concurrency control in most distributed systems is based on locking, just as in non-distributed systems. In a distributed system, requests to test, set, and release locks involve the exchange of messages increasing the overhead on the system.

Consider a transaction T that needs to update an object for which there exist replicas at n remote sites. If each site is responsible for locks on objects stored at that site, then a straightforward implementation will require at least 5n messages:
n lock requests.
n lock grants.
n update messages.
n acknowledgments.
n unlock requests.

The total time for the update could be several orders of magnitude greater than in a centralized system.

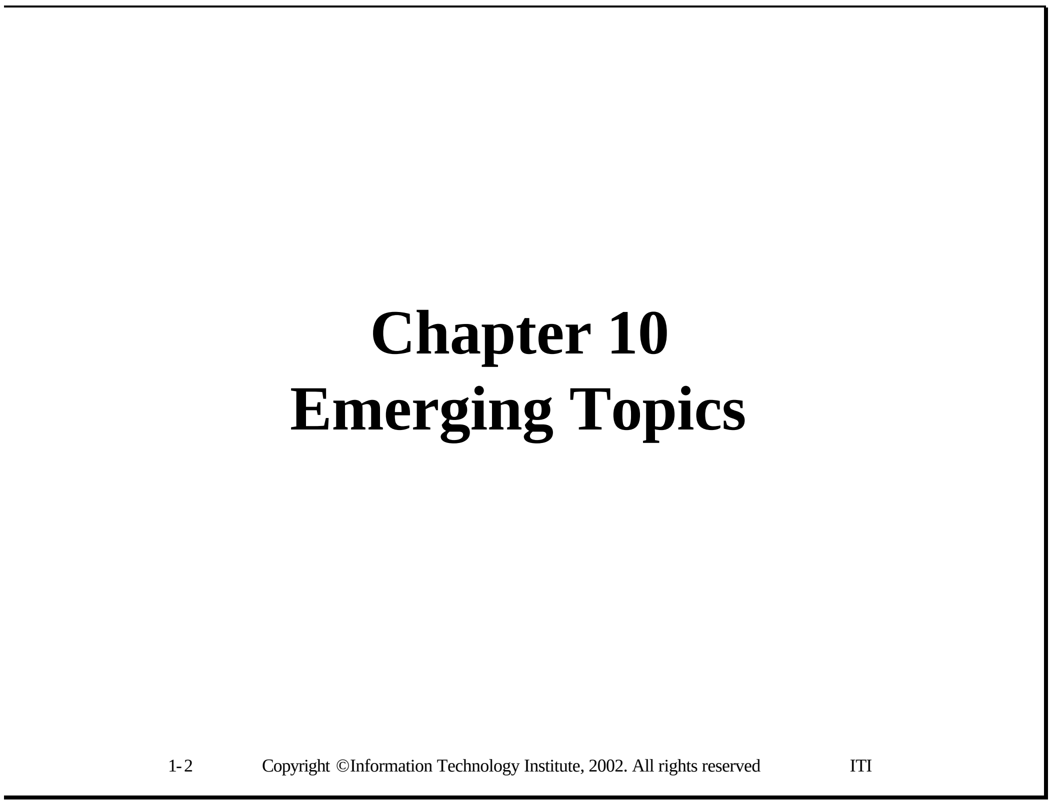
The solution to this problem is to adopt the primary copy under the strategy of update propagation. Under this strategy the se t of all copies of an object can be considered as a single object for locking purposes, and the total number of messages will be reduced from 5n to 2n + 3 (one lock request, one lock grant, n updates, n acknowledgments, and one unlock request). But this solution causes a loss of autonomy, where a transaction can fail if a primary copy is unavailable, even if the transaction is read-only and a lock copy is available.

## Locking Problem in Distributed System

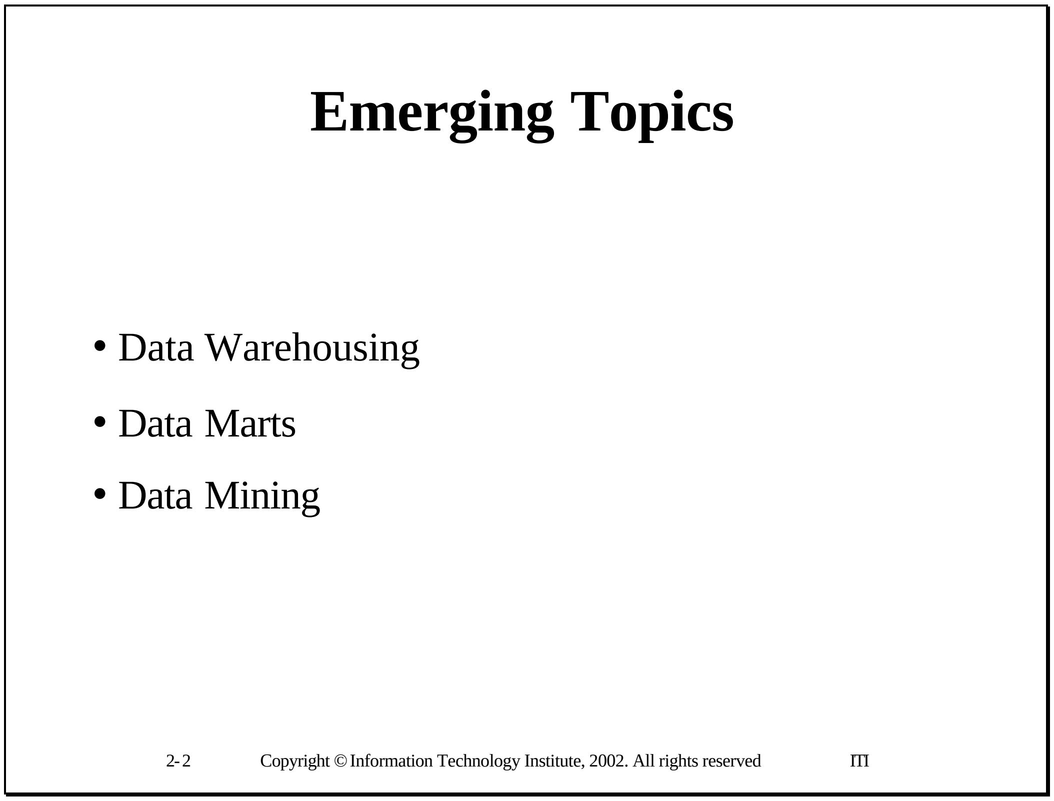


Another problem with locking in a distributed system is that it can lead to a global deadlock that involves two or more sites as shown in the above figure.

- The agent of transaction T2 at site X is waiting for the agent of transaction T1 at site X to release a lock;
- The agent of transaction T1 at site X is waiting for the agent of transaction T1 at site Y to complete;
- The agent of transaction T1 at site Y is waiting for the agent of transaction T2 at site Y to release a lock;
- The agent of transaction T2 at site Y is waiting for the agent of transaction T2 at site X to complete.



# Chapter 10
Emerging Topics



## Emerging Topics

- Data Warehousing
- Data Marts
- Data Mining

2-2 Copyright © Information Technology Institute, 2002. All rights reserved ITI

### Data Warehousing

A data warehouse has been defined in many different ways making it difficult to find such a rigorous definition for the term. The warehouse can be seen as a repository of data that often comes from many sources and is designed to serve the needs of a variety of end users, including those who ask ad hoc questions. It can also refer to the effective technology of integrating multiple operational databases that enables the strategic use of the massive amounts of data that companies are accumulating to make business decisions.

Data warehousing is not only about gathering data; it also involves issues of architecture and decisions about the tools used to collect, transfer, query and analyze the data.

### Data Mart

In general, a data mart is an inexpensive alternative to a data warehouse. A data mart is directed towards a partition of data that is created for the use of a dedicated group of users. Sometimes, the misleading statements about the simplicity and low cost of data marts result in a false belief that a “data mart is the subset version of a matured data warehouse”. In fact the two are designed and implemented for very different purposes.

Data marts often appear to be more attractive options than data warehouses as they are often smaller, faster to be implemented, and seem more manageable and responsive.

Of course, building a clinical application is much easier than building a complete hospital information system. Both involve hospital business automation, but each meets very different needs.

A data mart is built to answer an immediate problem (for ex. Predefined report), but they are not built to allow the users to ask any questions about the data.

So, the users (usually decision makers) benefit from quick short-term solutions, but they may suffer from the inability to pursue new questions in long term.

Data mart usually gives only one functional business unit the ability to get reports relevant to its own unit. There is no delivery of cross-functional environment, the concept of data integration is so weak, hence, data mart is used under high restrictions in case of business intelligent solution, decision support process demands require lots of investigation throughout many operational databases or even many different business models in one operational database.

Data marts and Data Warehouses, provide the same impact on the investment process, as they contain a huge amount of detailed and summary data that can answer almost any question requested by the user.

## Data Mining

Data mining is the process of discovering hidden information in a lot of amount of data. This discovery can be carried out via certain algorithms, some for clustering, or classification, or association of the data….etc.

Typically, data mining techniques classified by the type of the underlying algorithm used to identify the hidden patterns.

Several kinds of knowledge can be discovered including:

- **Classifications**

Classification in the context of data mining is learning a function that maps or classifies a data item into one of several predefined classes. There are many different methodologies that are used to carry out the classification task, including the decision trees, neural networks, genetic algorithms and others.

- **Association Rules**

It is the process of discovering the correlation between a large set of data items, association also referred to as “affinity analysis”. Correlation discovered can be between two items or more, although two-item affinities can be performed manually,
a five-item affinity would be next to impossible to define accurately, using SQL statement.

With the advent of customer relationship management, association analysis is at the forefront of data mining because it identifies the products customers purchase along with which products and services drive additional sales.

- **Clustering**

Clustering is the process of partitioning or grouping a given set of patterns into disjoint clusters and trying to maximize the intra-class similarity and minimize the interclass similarity. Unlike classification, clustering is classified as an unsupervised machine learning technique, as the class label is not previously known.
While query tools are certainly useful for examining cluster results, they are not capable of identifying clusters, clustering requires a completely different data manipulation technique from what a query language can provide.

- **Sequencing (Regression)**

It helps identifying a set of order-specific items or events.

**Example**

- Clustering algorithms can be used to group the data in clusters.
- Association algorithms can be used to decide which factors to associate with each other. In a market database, for example, association algorithms can be used to find out what items in the store should be bought together.