Database Systems

Instructor:
Dr. Hamid Turab Mirza

Department of Computer Science
CUI, Lahore

Definitions

- Database: organized collection of logically related data
- Data: stored representations of meaningful objects and events
  - Structured: numbers, text, dates
  - Unstructured: images, video, documents
- Information: data processed to increase knowledge in the person using the data
- Metadata: data that describes the properties and context of user data

Data in context

Class Roster

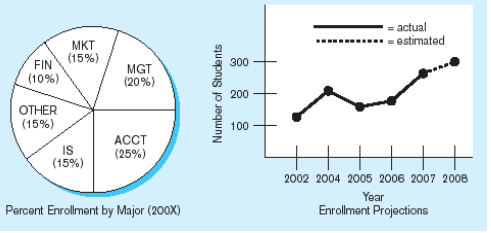
Course: MGT 500 Business Policy Semester: Spring 200X

Section: 2

Name	ID	Major	GPA
Baker, Kenneth D.	324917628	MGT	2.9
Doyle, Joan E.	476193248	MKT	3.4
Finkle, Clive R.	548429344	PRM	2.8
Lewis, John C.	551742186	MGT	3.7
McFerran, Debra R.	409723145	IS	2.9
Sisneros, Michael	392416582	ACCT	3.3

Context helps users understand data

Summarized data



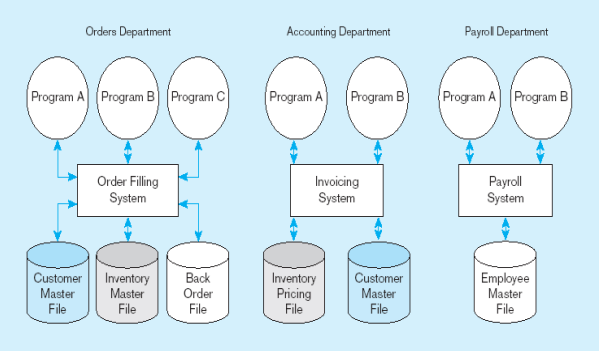
Graphical displays turn data into useful information that managers can use for decision making and interpretation

Table 1-1 Example Metadata for Class Roster

Data Item			*Value*			
Name	**Type**	**Length**	**Min**	**Max**	**Description**	**Source**
Course	Alphanumeric	30			Course ID and name	Academic Unit
Section	Integer	1	1	9	Section number	Registrar
Semester	Alphanumeric	10			Semester and year	Registrar
Name	Alphanumeric	30			Student name	Student IS
ID	Integer	9			Student ID (SSN)	Student IS
Major	Alphanumeric	4			Student major	Student IS
GPA	Decimal	3	0.0	4.0	Student grade point average	Academic Unit

Descriptions of the properties or characteristics of the data, including data types, field sizes, allowable values, and data context

File Processing Systems



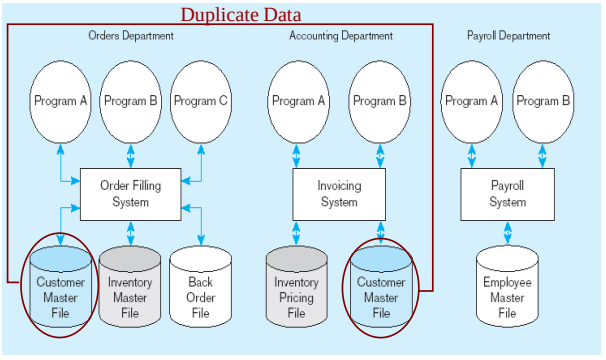
Disadvantages of File Processing

- **Program-Data Dependence**
  - All programs maintain metadata for each file they use
- **Duplication of Data**
  - Different systems/programs have separate copies of the same data
- **Limited Data Sharing**
  - No centralized control of data
- **Lengthy Development Times**
  - Programmers must design their own file formats
- **Excessive Program Maintenance**
  - 80% of information systems budget

Problems with Data Dependency

- Each application programmer must maintain his/her own data
- Each application program needs to include code for the metadata of each file
- Each application program must have its own processing routines for reading, inserting, updating, and deleting data
- Lack of coordination and central control
- Non-standard file formats

Old file processing systems at Pine Valley Furniture Company



Problems with Data Redundancy

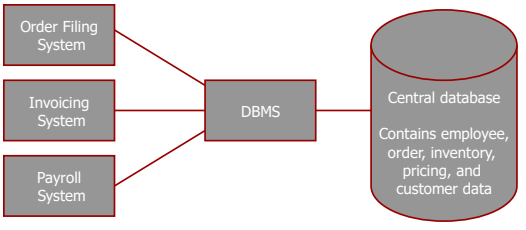
- Waste of space to have duplicate data
- Causes more maintenance headaches
- The biggest problem:
  - **Data changes in one file could cause inconsistencies**
  - Compromises in ***data integrity***

SOLUTION: The DATABASE Approach

- Central repository of shared data
- Data is managed by a controlling agent
- Stored in a standardized, convenient form

Database Management System

- A software system that is used to create, maintain, and provide controlled access to user databases



DBMS manages data resources like an operating system manages hardware resources

Advantages of the Database Approach

- Program-data independence
- Planned data redundancy
- Improved data consistency
- Improved data sharing
- Increased application development productivity
- Enforcement of standards
- Improved data quality
- Improved data accessibility and responsiveness
- Reduced program maintenance
- Improved decision support

Costs and Risks of the Database Approach

- New, specialized personnel
- Installation and management cost and complexity
- Conversion costs
- Need for explicit backup and recovery
- Organizational conflict

The Range of Database Applications

- Personal databases
- Workgroup databases
- Departmental/divisional databases
- Enterprise database

Table 1-6 Summary of Database Applications (adapted from White, 1995)

Type of Database	*Typical Number of Users*	*Typical Architecture*	*Typical Size of Database*
Personal	1	Desktop/laptop computer, PDA	Megabytes
Workgroup	5–25	Client/server (two-tier)	Megabytes–gigabytes
Department/Division	25–100	Client/server (three-tier)	Gigabytes
Enterprise	>100	Client/server (distributed or parallel server)	Gigabytes–terabytes
Web-enabled	>1000	Web server and application servers	Megabytes–gigabytes

Customer

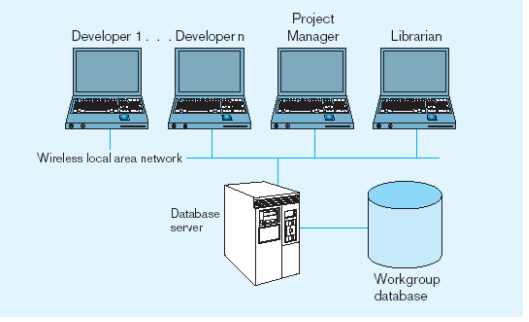
Customer Name: **Multi Media, Inc.**
Address: **1000 River Road**
City: **San Antonio**
State: **TX**
Zip: **76235**
Phone: **(219) 864-2000**
Next Contact Date: **10/17/2006** Time: **10:30 AM**

Contact History for Customer

Date	Time	Contact	Comments
08/04/2006	10:00 AM	Roberts	Review proposal
08/19/2006	08:00 AM	Roberts	Revise schedule
09/10/2006	09:00 AM	Pearson	Sign contract
09/21/2006	02:00 PM	Roberts	Follow up

Typical data from a personal database

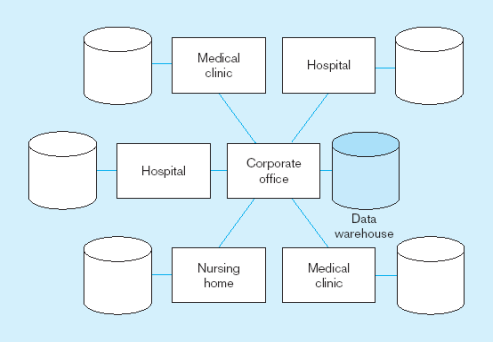
Workgroup database with wireless local area network



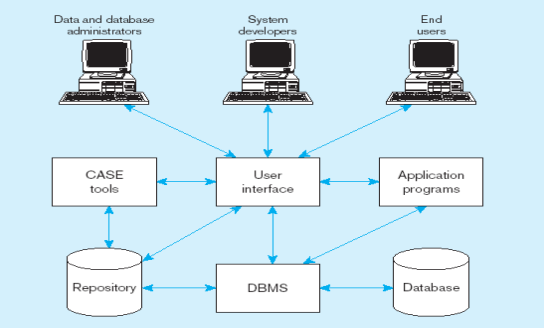
Enterprise Database Applications

- Enterprise Resource Planning (ERP)
  - Integrate all enterprise functions (manufacturing, finance, sales, marketing, inventory, accounting, human resources)
- Data Warehouse
  - Integrated decision support system derived from various operational databases

An enterprise data warehouse



Components of the Database Environment



Components of the Database Environment

- **CASE Tools**–computer-aided software engineering
- **Repository**–centralized storehouse of metadata
- **Database Management System (DBMS)** –software for managing the database
- **Database**–storehouse of the data
- **Application Programs**–software using the data
- **User Interface**–text and graphical displays to users
- **Data/Database Administrators**–personnel responsible for maintaining the database
- **System Developers**–personnel responsible for designing databases and software
- **End Users**–people who use the applications and databases

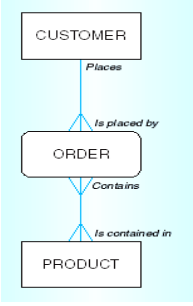
Elements of the Database Approach

- Data models
  - Graphical system capturing nature and relationship of data
  - Enterprise Data Model–high-level entities and relationships for the organization
- Relational Databases
  - Database technology involving tables (relations) representing entities and primary/foreign keys representing relationships
- Use of Internet Technology
  - Networks and telecommunications, distributed databases, client-server, and 3-tier architectures
- Database Applications
  - Application programs used to perform database activities (create, read, update, and delete) for database users

Enterprise Data Model

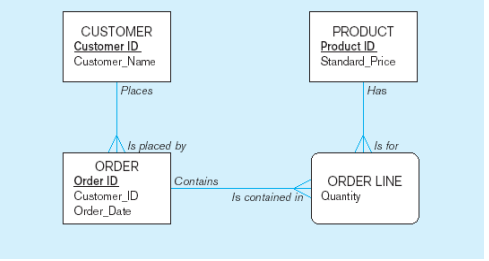
- First step in database development
- Specifies scope and general content
- Overall picture of organizational data at high level of abstraction
- Entity-relationship diagram
- Descriptions of entity types
- Relationships between entities
- Business rules

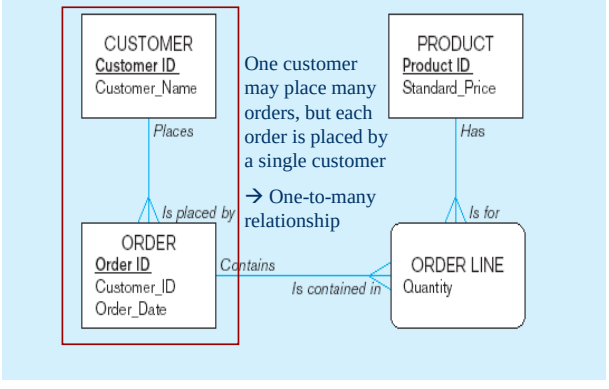
Segment from enterprise data model



Enterprise data model describes the high-level entities in an organization and the relationship between these entities

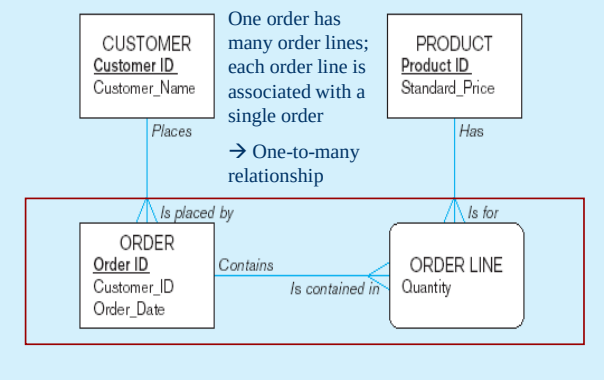
Segment of an Enterprise Data Model





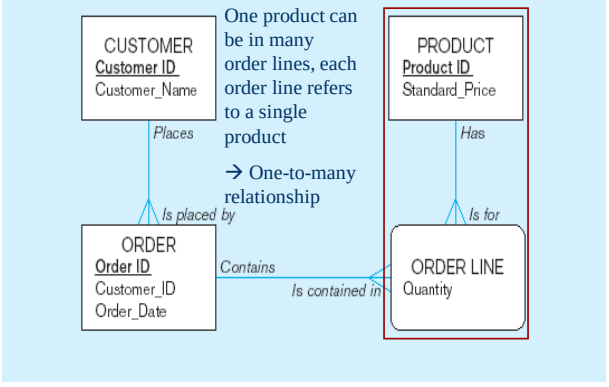
One customer may place many orders, but each order is placed by a single customer

→ One-to-many relationship



One order has many order lines; each order line is associated with a single order

→ One-to-many relationship



Managing Projects: People Involved

- Business analysts
- Systems analysts
- Database analysts and data modelers
- Users
- Programmers
- Database architects
- Data administrators
- Project managers
- Other technical experts

Business Rules

- Statements that define or constrain some aspect of the business
- Assert business structure
- Control/influence business behavior
- Expressed in terms familiar to end users
- Automated through DBMS software

A Good Business Rule is:

- Declarative–what, not how
- Precise–clear, agreed-upon meaning
- Atomic–one statement
- Consistent–internally and externally
- Expressible–structured, natural language
- Distinct–non-redundant
- Business-oriented–understood by business people

A Good Data Name is:

- Related to business, not technical, characteristics
- Meaningful and self-documenting
- Unique
- Readable
- Composed of words from an approved list
- Repeatable

Data Definitions

- Explanation of a term or fact
  - Term–word or phrase with specific meaning
  - Fact–association between two or more terms
- Guidelines for good data definition
  - Gathered in conjunction with systems requirements
  - Accompanied by diagrams
  - Iteratively created and refined
  - Achieved by consensus

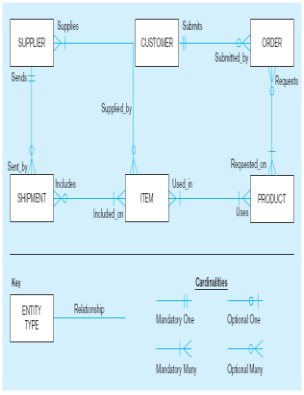
E-R Model Constructs

- Entities:
  - Entity instance–person, place, object, event, concept (often corresponds to a row in a table)
  - Entity Type–collection of entities (often corresponds to a table)
- Relationships:
  - Relationship instance–link between entities (corresponds to primary key-foreign key equivalencies in related tables)
  - Relationship type–category of relationship…link between entity types
- Attribute–property or characteristic of an entity or relationship type (often corresponds to a field in a table)

Sample E-R Diagram - 1

Pine valley furniture company can purchase items from a number of different suppliers, who then ship the items to the manufacturer. The items are assembled into products that are sold to customers who order the products. Each customer order may include one or more lines corresponding to the Products appearing on that order.

Sample E-R Diagram



•A SUPPLIER may supply many ITEMs (by "may supply" we mean the supplier may not supply any items). Each ITEM is supplied by any number of SUPPLIERs (by "is supplied" we mean must be supplied by at least one supplier).

•Each ITEM must be used in the assembly of at least one PRODUCT, and may be used in many products. Conversely, each PRODUCT must use one or more ITEMs.

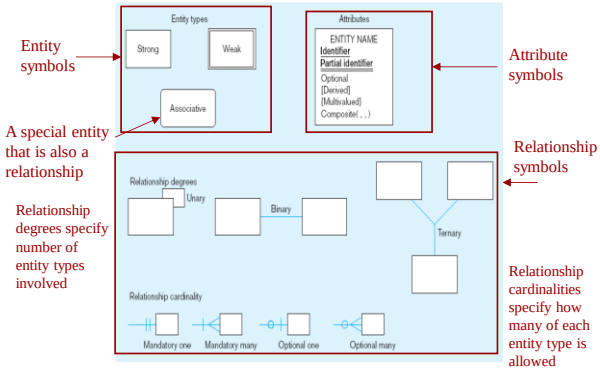
•A SUPPLIER may send many SHIPMENTs. On the other hand, each shipment must be sent by exactly one SUPPLIER.

•A SHIPMENT must include one (or more) ITEMs. An ITEM may be included on several SHIPMENTs.

•A CUSTOMER may submit any number of ORDERs. However, each ORDER must be submitted by exactly one CUSTOMER.

•An ORDER must request one (or more) PRODUCTs. A given PRODUCT may not be requested on nay ORDER, or may be requested one one or more orders.

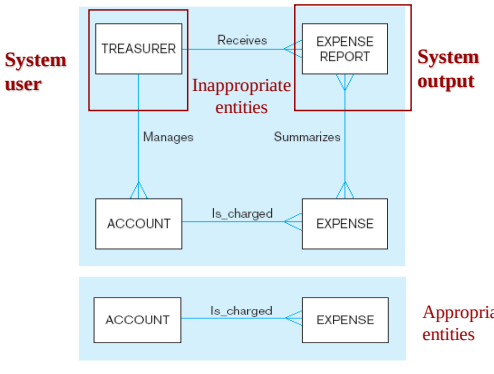
Basic E-R notation



What Should an Entity Be?

- SHOULD BE:
  - An object that will have many instances in the database
  - An object that will be composed of multiple attributes
  - An object that we are trying to model
- SHOULD NOT BE:
  - A user of the database system
  - An output of the database system (e.g., a report)

Example of inappropriate entities



Attributes

- Attribute–property or characteristic of an entity or relationahip type
- Classifications of attributes:
  - Required versus Optional Attributes
  - Simple versus Composite Attribute
  - Single-Valued versus Multivalued Attribute
  - Stored versus Derived Attributes
  - Identifier Attributes

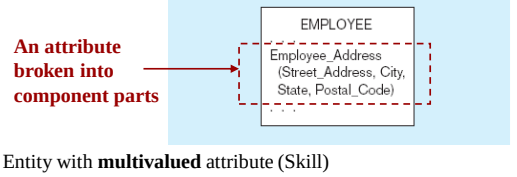
Identifiers (Keys)

- Identifier (Key)–An attribute (or combination of attributes) that uniquely identifies individual instances of an entity type
- Simple versus Composite Identifier
- Candidate Identifier–an attribute that could be a key…satisfies the requirements for being an identifier

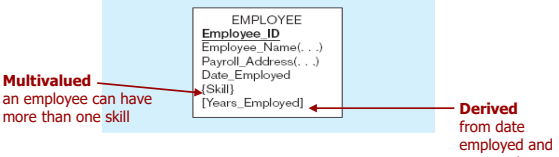
Characteristics of Identifiers

- Will not change in value
- Will not be null
- No intelligent identifiers (e.g., containing locations or people that might change)

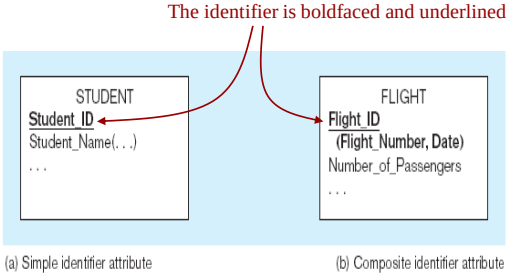
A **composite** attribute



Entity with **multivalued** attribute (Skill) and **derived** attribute (Years_Employed)



Simple and composite identifier attributes



Simple example of time-stamping

This attribute that is both multivalued *and* composite

Relationships

Examples of relationship types:

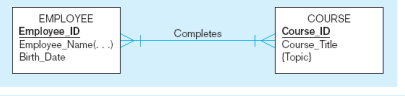
- A branch is managed by at most one manager
- A manager can manage at most one branch

- A transaction can update one account
- An account can be updated by many transactions

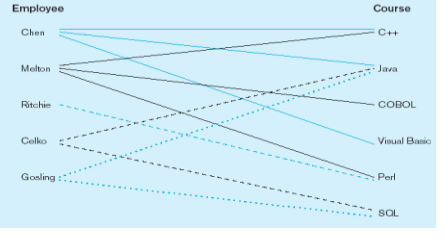
- A customer can own many accounts
- An account may be owned by more than one customer

Relationship types and instances

a) Relationship type



b) Relationship instances



Cardinality of Relationships

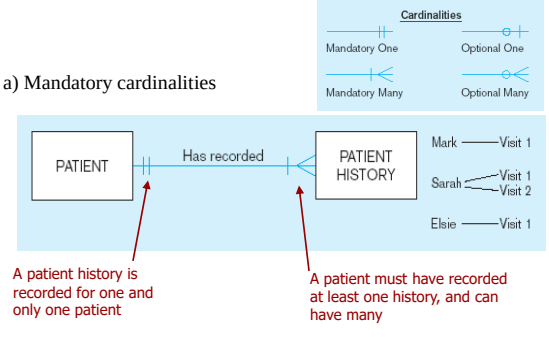
- One-to-One
  - Each entity in the relationship will have exactly one related entity
- One-to-Many
  - An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
- Many-to-Many
  - Entities on both sides of the relationship can have many related entities on the other side

Cardinality Constraints

- Cardinality Constraints - the number of instances of one entity that can or must be associated with each instance of another entity
- Minimum Cardinality
  - If zero, then optional
  - If one or more, then mandatory
- Maximum Cardinality
  - The maximum number

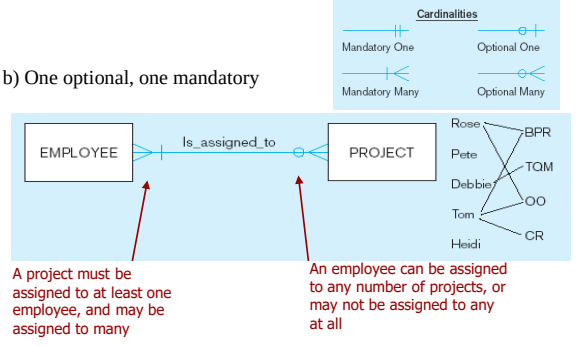
Examples of cardinality constraints

a) Mandatory cardinalities



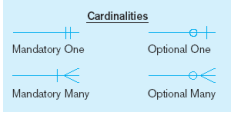
Examples of cardinality constraints (cont.)

b) One optional, one mandatory

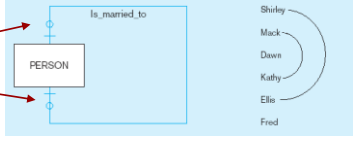


Examples of cardinality constraints (cont.)

a) Optional cardinalities



A person is is married to at most one other person, or may not be married at all



Sample E-R Diagram – 2

ABC housing society is a medium sized local authority. One of the responsibilities of the Society is the maintenance and repair of Society owned housing within its boundaries. The authority wishes to develop an information system to monitor information on housing repair work, a description of which is as follows:

"For the purpose of housing repair work, the society is divided into a number of areas. Each area is subdivided into streets or roads, each street or road in likely to have a number of houses along it. Details of each house are held, along with details of each instance of repair work carried out to a specific house. Each area is maintained by a single repair team, although a team may be responsible for more than one area. A repair team consists of a number of employees, one of whom is designated the team supervisor. The team are allocated a number of vehicles for use in their work, which are generally used by the "owning" team only, although are occasionally "borrowed" by other Teams. Each team is based at a maintenance depot, and in some cases, more than one team may be based at a single depot".

Sample E-R Diagram - 2 (Entities Identified)

ABC housing society is a medium sized local authority. One of the responsibilities of the Society is the maintenance and repair of Society owned housing within its boundaries. The authority wishes to develop an information system to monitor information on housing repair work, a description of which is as follows:

"For the purpose of housing repair work, the society is divided into a number of **areas**. Each area is subdivided into **streets or roads**, each street or road in likely to have a number of **houses** along it. Details of each house are held, along with details of each instance of **repair** work carried out to a specific house. Each area is maintained by a single repair **team**, although a team may be responsible for more than one area. A repair team consists of a number of **employees**, one of whom is designated the team supervisor. The team are allocated a number of **vehicles** for use in their work, which are generally used by the "owning" team only, although are occasionally "borrowed" by other Teams. Each team is based at a maintenance **depot**, and in some cases, more than one team may be based at a single depot".

Relation

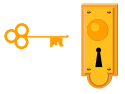
- Definition: A relation is a named, two-dimensional table of data
- Table consists of rows (records) and columns (attribute or field)
- Requirements for a table to qualify as a relation:
  - It must have a unique name
  - Every attribute value must be atomic (not multivalued, not composite)
  - Every row must be unique (can't have two rows with exactly the same values for all their fields)
  - Attributes (columns) in tables must have unique names
  - The order of the columns must be irrelevant
  - The order of the rows must be irrelevant

Correspondence with E-R Model

- Relations (tables) correspond with entity types and with many-to-many relationship types
- Rows correspond with entity instances and with many-to-many relationship instances
- Columns correspond with attributes

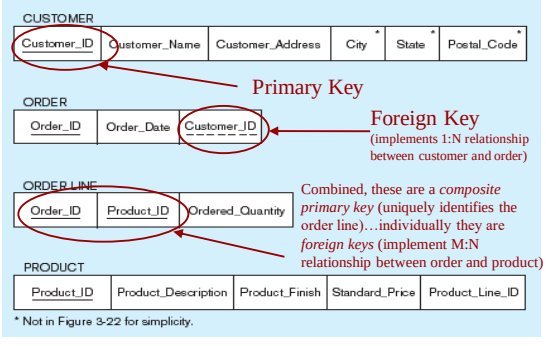
- NOTE: The word ***relation*** (in relational database) is NOT the same as the word ***relationship*** (in E-R model)

Key Fields



- Keys are special fields that serve two main purposes:
  - ***Primary keys*** are unique identifiers of the relation in question. Examples include employee numbers, social security numbers, etc. *This is how we can guarantee that all rows are unique*
  - ***Foreign keys*** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)
- Keys can be ***simple*** (a single field) or ***composite*** (more than one field)
- Keys usually are used as indexes to speed up the response to user queries (Discussed Latter)

Schema for four relations (Pine Valley Furniture Company)



Integrity Constraints

- Domain Constraints
  - Allowable values for an attribute. See Table 5-1
- Entity Integrity
  - No primary key attribute may be null. All primary key fields **MUST** have data
- Action Assertions
  - Business rules.

Table 5-1 Domain Definitions for INVOICE Attributes

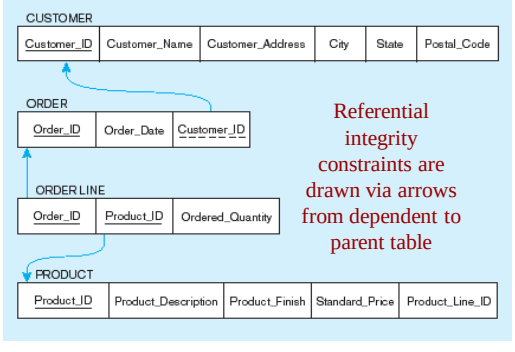
Attribute	Domain Name	Description	Domain
Customer_ID	Customer_IDs	Set of all possible customer IDs	character: size 5
Customer_Name	Customer_Names	Set of all possible customer names	character: size 25
Customer_Address	Customer_Addresses	Set of all possible customer addresses	character: size 30
City	Cities	Set of all possible cities	character: size 20
State	States	Set of all possible states	character: size 2
Postal_Code	Postal_Codes	Set of all possible postal zip codes	character: size 10
Order_ID	Order_IDs	Set of all possible order IDs	character: size 5
Order_Date	Order_Dates	Set of all possible order dates	date format mm/dd/yy
Product_ID	Product_IDs	Set of all possible product IDs	character: size 5
Product_Description	Product_Descriptions	Set of all possible product descriptions	character size 25
Product_Finish	Product_Finishes	Set of all possible product finishes	character: size 15
Standard_Price	Unit_Prices	Set of all possible unit prices	monetary: 6 digits
Product_Line_ID	Product_Line_IDs	Set of all possible product line IDs	integer: 3 digits
Ordered_Quantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

Domain definitions enforce domain integrity constraints

Integrity Constraints

- Referential Integrity–rule states that any foreign key value (on the relation of the many side) MUST match a primary key value in the relation of the one side. (Or the foreign key can be null)
  - For example: Delete Rules
    - Restrict–don't allow delete of "parent" side if related rows exist in "dependent" side
    - Cascade–automatically delete "dependent" side rows that correspond with the "parent" side row to be deleted
    - Set-to-Null–set the foreign key in the dependent side to null if deleting from the parent side → not allowed for weak entities

Referential integrity constraints (Pine Valley Furniture)



SQL table definitions

```
CREATE TABLE CUSTOMER
      (CUSTOMER_ID          VARCHAR(5)      NOT NULL,
       CUSTOMER_NAME        VARCHAR(25)     NOT NULL,
       CUSTOMER ADDRESS     VARCHAR(30)     NOT NULL,
       CITY                 VARCHAR(20)     NOT NULL,
       STATE                CHAR(2)         NOT NULL,
       POSTAL_CODE          CHAR(10)        NOT NULL,
PRIMARY KEY (CUSTOMER_ID);

CREATE TABLE ORDER
      (ORDER_ID             CHAR(5)         NOT NULL,
       ORDER DATE           DATE            NOT NULL,
       CUSTOMER_ID          VARCHAR(5)      NOT NULL,
PRIMARY KEY (ORDER_ID),
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID);

CREATE TABLE ORDER_LINE
      (ORDER_ID             CHAR(5)         NOT NULL,
       PRODUCT_ID           CHAR(5)         NOT NULL,
       ORDERED_QUANTITY     INT             NOT NULL,
PRIMARY KEY (ORDER_ID, PRODUCT_ID),
FOREIGN KEY (ORDER_ID) REFERENCES ORDER (ORDER_ID),
FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID);

CREATE TABLE PRODUCT
      (PRODUCT_ID           CHAR(5)         NOT NULL,
       PRODUCT_DESCRIPTION  VARCHAR(25),
       PRODUCT_FINISH       VARCHAR(12),
       STANDARD_PRICE       DECIMAL(8,2)    NOT NULL,
       PRODUCT_LINE_ID      INT             NOT NULL,
PRIMARY KEY (PRODUCT_ID);
```

Referential integrity constraints are implemented with foreign key to primary key references

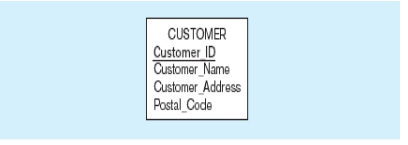
Transforming EER Diagrams into Relations

Mapping Regular Entities to Relations

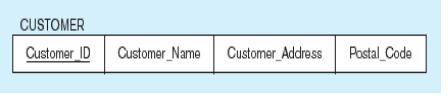
1. Simple attributes: E-R attributes map directly onto the relation
2. Composite attributes: Use only their simple, component attributes
3. Multivalued Attribute–Becomes a separate relation with a foreign key taken from the superior entity

Mapping a regular entity

(a) CUSTOMER entity type with simple attributes



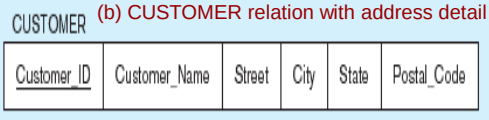
(b) CUSTOMER relation



Mapping a composite attribute

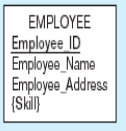
(a) CUSTOMER entity type with composite attribute

(b) CUSTOMER relation with address detail



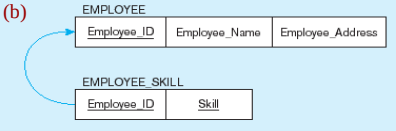
Mapping an entity with a multivalued attribute

(a)



Multivalued attribute becomes a separate relation with foreign key

(b)



One–to–many relationship between original entity and new relation

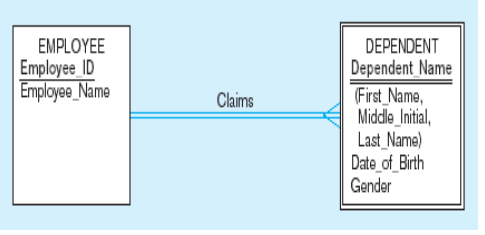
Transforming EER Diagrams into Relations (cont.)

Mapping Weak Entities

- Becomes a separate relation with a foreign key taken from the superior entity
- Primary key composed of:
  - Partial identifier of weak entity
  - Primary key of identifying relation (strong entity)

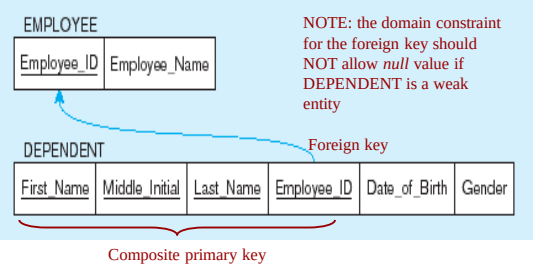
Example of mapping a weak entity

a) Weak entity DEPENDENT



Example of mapping a weak entity (cont.)

b) Relations resulting from weak entity



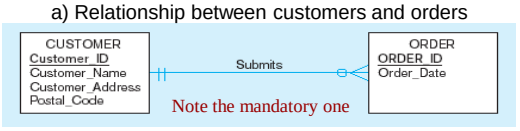
Transforming EER Diagrams into Relations (cont.)

Mapping Binary Relationships

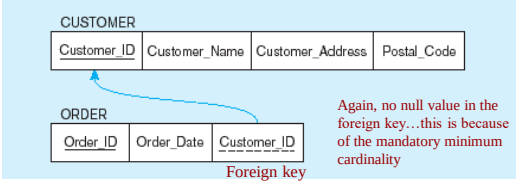
- One-to-Many–Primary key on the one side becomes a foreign key on the many side
- Many-to-Many–Create a ***new relation*** with the primary keys of the two entities as its primary key
- One-to-One–Primary key on the mandatory side becomes a foreign key on the optional side

Example of mapping a 1:M relationship

a) Relationship between customers and orders

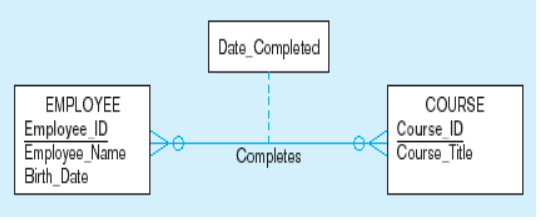


b) Mapping the relationship



Example of mapping an M:N relationship

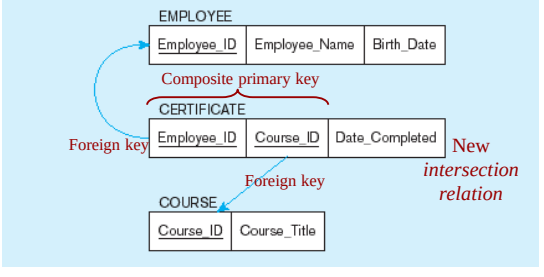
a) Completes relationship (M:N)



The *Completes* relationship will need to become a separate relation

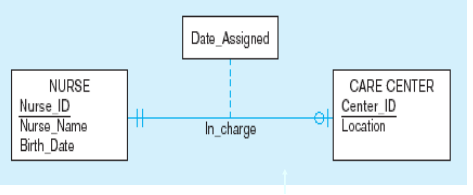
Example of mapping an M:N relationship (cont.)

b) Three resulting relations



Example of mapping a binary 1:1 relationship

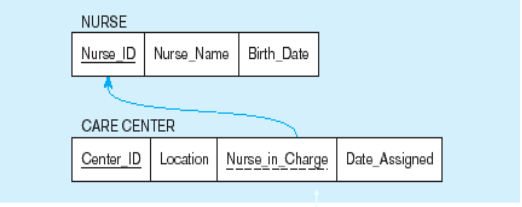
a) In_charge relationship (1:1)



Often in 1:1 relationships, one direction is optional.

Example of mapping a binary 1:1 relationship (cont.)

b) Resulting relations



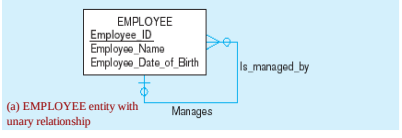
Foreign key goes in the relation on the optional side, Matching the primary key on the mandatory side

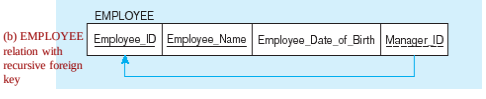
Transforming EER Diagrams into Relations (cont.)

Mapping Unary Relationships

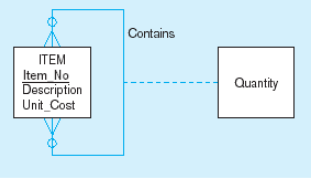
- One-to-Many–Recursive foreign key in the same relation
- Many-to-Many–Two relations:
  - One for the entity type
  - One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

Mapping a unary 1:N relationship



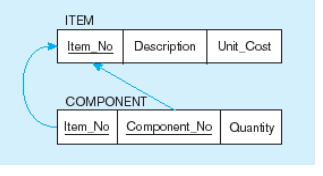


Mapping a unary M:N relationship



(a) Bill-of-materials relationships (M:N)

(b) ITEM and COMPONENT relations



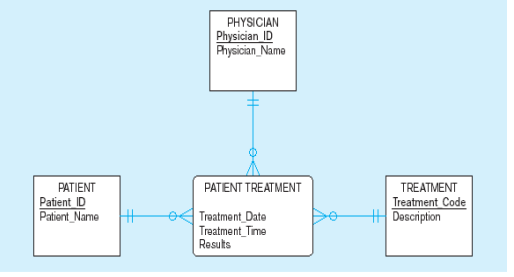
Transforming EER Diagrams into Relations (cont.)

Mapping Ternary (and n-ary) Relationships

- One relation for each entity and one for the associative entity
- Associative entity has foreign keys to each entity in the relationship

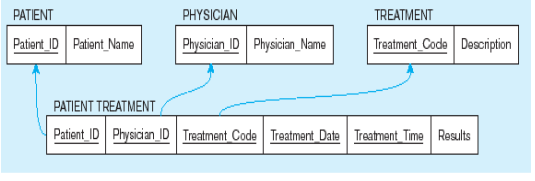
Mapping a ternary relationship

a) PATIENT TREATMENT Ternary relationship with associative entity



Mapping a ternary relationship (cont.)

b) Mapping the ternary relationship PATIENT TREATMENT



Remember that the primary key MUST be unique

This is why treatment date and time are included in the composite primary key

But this makes a very cumbersome key…

It would be better to create a surrogate key like Treatment#

Data Normalization

- Primarily a tool to validate and improve a logical design so that it satisfies certain constraints that ***avoid unnecessary duplication of data***
- The process of decomposing relations with anomalies to produce smaller, ***well-structured*** relations

Well-Structured Relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid anomalies
  - **Insertion Anomaly**–adding new rows forces user to create duplicate data
  - **Deletion Anomaly**–deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly**–changing data in a row forces changes to other rows because of duplication

General rule of thumb: A table should not pertain to more than one entity type

Example

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Question–Is this a relation?

Answer–Yes: Unique rows and no multivalued attributes

Question–What should be the primary key?

Answer–Composite: Emp_ID, Course_Title

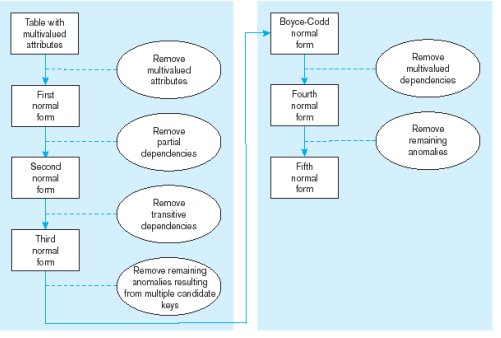
Anomalies in this Table

- **Insertion**–can't enter a new employee without having the employee take a class
- **Deletion**–if we remove employee 140, we lose information about the existence of a Tax Acc class
- **Modification**–giving a salary increase to employee 100 forces us to update multiple records

Why do these anomalies exist?

Because there are two themes (entity types) in this one relation. This results in data duplication and an unnecessary dependency between the entities

Steps in normalization



First Normal Form

- No multivalued attributes
- Every attribute value is atomic
- Fig. 5-25 *is not* in 1st Normal Form (multivalued attributes) ➔ it is not a relation
- Fig. 5-26 *is* in 1st Normal form
- ***All relations*** **are in 1st Normal Form**

Table with multivalued attributes, not in 1st normal form

Figure 5-25
INVOICE data (Pine Valley Furniture Company)

Order_ID	Order_ Date	Customer_ ID	Customer_ Name	Customer_ Address	Product_ID	Product_ Description	Product_ Finish	Unit_ Price	Ordered_ Quantity
1006	10/24/2006	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2006	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

Note: this is NOT a relation

Table with no multivalued attributes and unique rows, in 1st normal form

Order_ID	Order_ Date	Customer_ ID	Customer_ Name	Customer_ Address	Product_ID	Product_ Description	Product_ Finish	Unit_ Price	Ordered_ Quantity
1006	10/24/2006	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2006	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2006	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2006	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2006	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

Figure 5-26
INVOICE relation (1NF) (Pine Valley Furniture Company)

Product_ID → Product_Description, Product_Finish, Unit_Price
Order_ID, Product_ID → Ordered_Quantity

Note: this is relation, but not a well-structured one

Anomalies in this Table

- **Insertion**–if new product is ordered for order 1007 of existing customer, customer data must be re-entered, causing duplication
- **Deletion**–if we delete the Dining Table from Order 1006, we lose information concerning this item's finish and price
- **Update**–changing the price of product ID 4 requires update in several records

Why do these anomalies exist?

Because there are multiple themes (entity types) in one relation. This results in duplication and an unnecessary dependency between the entities

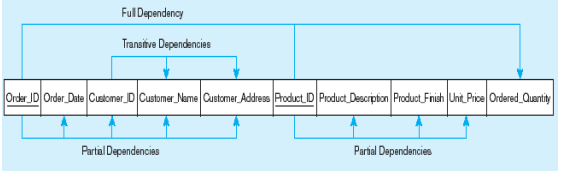
Second Normal Form

- 1NF PLUS ***every non-key attribute is fully functionally dependent on the ENTIRE primary key***
  - Every non-key attribute must be defined by the entire key, not by only part of the key
  - No partial functional dependencies

Functional Dependencies and Keys

- Functional Dependency: The value of one attribute (the ***determinant***) determines the value of another attribute
- Each non-key field is functionally dependent on every candidate key
- Candidate Key:
  - A unique identifier. One of the candidate keys will become the primary key
    - E.g. perhaps there is both credit card number and SS# in a table…in this case both are candidate keys

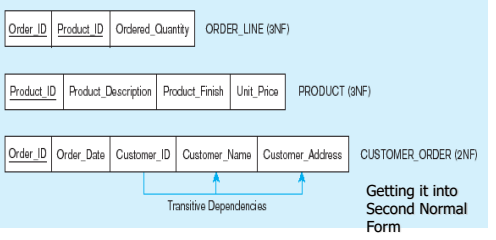
Functional dependency diagram for INVOICE



Order_ID ➔ Order_Date, Customer_ID, Customer_Name, Customer_Address
Customer_ID ➔ Customer_Name, Customer_Address
Product_ID ➔ Product_Description, Product_Finish, Unit_Price
Order_ID, Product_ID ➔ Order_Quantity

Therefore, NOT in 2nd Normal Form

Removing partial dependencies

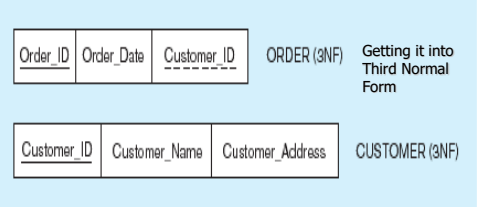


Partial dependencies are removed, but there are still transitive dependencies

Third Normal Form

- 2NF PLUS ***no transitive dependencies*** (functional dependencies on non-primary-key attributes)
- Note: This is called transitive, because the primary key is a determinant for another attribute, which in turn is a determinant for a third
- Solution: Non-key determinant with transitive dependencies go into a new table; non-key determinant becomes primary key in the new table and stays as foreign key in the old table

Removing partial dependencies



Transitive dependencies are removed

Physical Database Design

- Purpose–translate the logical description of data into the *technical specifications* for storing and retrieving data
- Goal–create a design for storing data that will provide *adequate performance* and insure *database integrity*, *security*, and *recoverability*

Physical Design Process

Inputs

- Normalized relations
- Volume estimates
- Attribute definitions
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- DBMS technology used

Leads to

Decisions

- Attribute data types
- File organizations
- Indexes and database architectures
- Query optimization

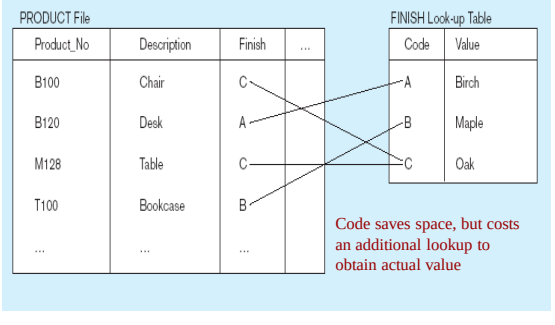
Designing Fields

- Field: smallest unit of data in database
- Field design
  - Choosing data type
  - Controlling data integrity

Choosing Data Types

- CHAR–fixed-length character
- VARCHAR2–variable-length character (memo)
- LONG–large number
- NUMBER–positive/negative number
- INEGER–positive/negative whole number
- DATE–actual date
- BLOB–binary large object (good for graphics, sound clips, etc.)

Example code look-up table
(Pine Valley Furniture Company)



Field Data Integrity

- Default value–assumed value if no explicit value
- Range control–allowable value limitations (constraints or validation rules)
- Null value control–allowing or prohibiting empty fields
- Referential integrity–foreign-key to primary-key match-ups

Denormalization

- Transforming ***normalized*** relations into ***unnormalized*** physical record specifications
- Benefits:
  - Can improve performance (speed) by reducing number of table lookups (i.e. *reduce number of necessary join queries*)
- Costs (due to data duplication)
  - Wasted storage space
  - Data integrity/consistency threats
- Common denormalization opportunities
  - One-to-one relationship (Fig. 6-3)
  - Many-to-many relationship with attributes (Fig. 6-4)
  - Reference data (1:N relationship where 1-side has data not used in any other relationship) (Fig. 6-5)

Figure 6-3 A possible denormalization situation: two entities with one-to-one relationship

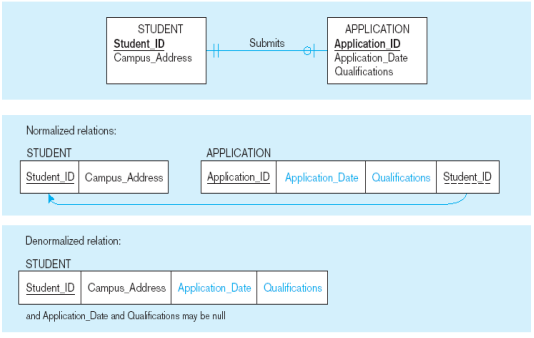


Figure 6-4 A possible denormalization situation: a many-to-many relationship with nonkey attributes

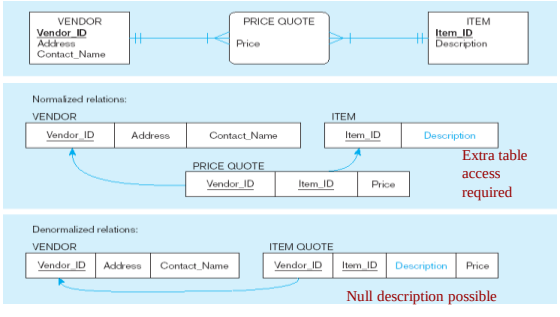
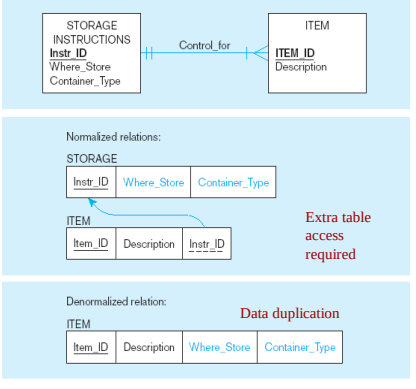


Figure 6-5 A possible denormalization situation: reference data



Data Replication

- Purposely storing the same data in multiple locations of the database
- Improves performance by allowing multiple users to access the same data at the same time with minimum contention
- Sacrifices data integrity due to data duplication
- Best for data that is not updated often

Ensuring Transaction Integrity

- Transaction = A discrete unit of work that must be completely processed or not processed at all

May involve multiple updates

If any update fails, then all other updates must be cancelled

SQL commands for transactions

- BEGIN TRANSACTION/END TRANSACTION

Marks boundaries of a transaction

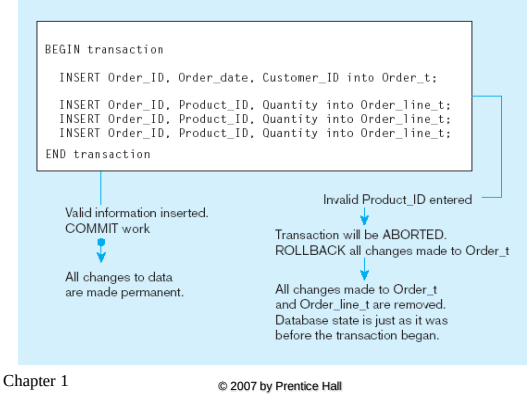
- COMMIT

Makes all updates permanent

- ROLLBACK

Cancels updates since the last COMMIT

An SQL Transaction sequence (in pseudocode)



Data Dictionary Facilities

- System tables that store metadata
- Users usually can view some of these tables
- Users are restricted from updating them
- Some examples in Oracle 10g
  - DBA_TABLES–descriptions of tables
  - DBA_CONSTRAINTS–description of constraints
  - DBA_USERS–information about the users of the system
- Examples in Microsoft SQL Server 2000
  - SYSCOLUMNS–table and column definitions
  - SYSDEPENDS–object dependencies based on foreign keys
  - SYSPERMISSIONS–access permissions granted to users

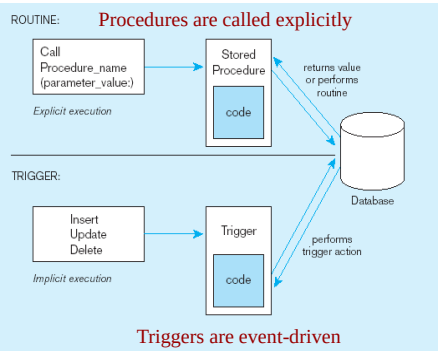
Routines and Triggers

- **Routines**
  - Program modules that execute on demand

  1. Functions–routines that return values and take input parameters

  2. Procedures–routines that do not return values and can take input or output parameters
- **Triggers**
  - Routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

Triggers contrasted with stored procedures



Routines

```
CREATE OR REPLACE PROCEDURE PRODUCT_LINE_SALE AS
BEGIN
   UPDATE PRODUCT_T
   SET SALE_PRICE = .90 * STANDARD_PRICE
   WHERE STANDARD_PRICE > = 400;

   UPDATE PRODUCT_T
   SET SALE_PRICE = .85 * STANDARD_PRICE
   WHERE STANDARD_PRICE < 400;
END;
```

To run the procedure in Oracle, use this command:

SQL> EXEC PRODUCT_LINE_SALE

Triggers

```
CREATE TRIGGER ORDER_ID_BIR
   BEFORE INSERT ON ORDER_T
   FOR EACH ROW
BEGIN
   SELECT ID_SEQUENCE.NEXTVAL
   INTO :NEW.ORDER_ID
   FROM DUAL;
END ORDER_ID_BIR;
```

Stored Procedures are pre-compile objects which are compiled for first time and its compiled format is saved which executes (compiled code) whenever it is called. But Function is compiled and executed every time when it is called.

Basic Difference
1. Function must return a value but in Stored Procedure it is optional(Procedure can return zero or n values).
2. Functions can have only input parameters for it whereas Procedures can have input/output parameters .
3. Functions can be called from Procedure whereas Procedures cannot be called from Function.

Advance Difference
1. Procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.

2. Procedures can not be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.

3. Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.

4. The most important feature of stored procedures over function is to retention and reuse the execution plan while in case of function it will be compiled every time.

5. Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.

6. We can go for Transaction Management in Procedure whereas we can't go in Function.

Indexes

Indexes are synonymous with performance on the Oracle database.

Especially on large tables, indexes are the difference between very slow and efficient applications.

Rules for Using Indexes

1. Use on larger tables
2. Index the primary key of each table
3. Index search fields (fields frequently in WHERE clause)
4. Fields in SQL ORDER BY and GROUP BY commands
5. When there are >100 values but not when there are <30 values

Rules for Using Indexes (cont.)

6. Avoid use of indexes for fields with long values; perhaps compress values first
7. DBMS may have limit on number of indexes per table and number of bytes per indexed field(s)
8. Null values will not be referenced from an index
9. Use indexes heavily for non-volatile databases; limit the use of indexes for volatile databases

   Why? Because modifications (e.g. inserts, deletes) require updates to occur in index files

Sample CREATE INDEX

```
CREATE INDEX emp_last_name_indx
ON employee (lastname);
```

Using and Defining Views

- Views provide users controlled access to tables
- Base Table–table containing the raw data
- Dynamic View
  - A "virtual table" created dynamically upon request by a user
  - No data actually stored; instead data from base table made available to user
  - Based on SQL SELECT statement on base tables or other views
- Materialized View
  - Copy or replication of data
  - Data actually stored
  - Must be refreshed periodically to match the corresponding base tables

Sample CREATE VIEW

```
CREATE VIEW EXPENSIVE_STUFF_V AS
SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE
FROM PRODUCT_T
WHERE UNIT_PRICE >300
WITH CHECK_OPTION;
```

- View has a name
- View is based on a SELECT statement
- CHECK_OPTION works only for updateable views and prevents updates that would create rows not included in the view

Advantages of Views

- Simplify query commands
- Assist with data security (but don't rely on views for security, there are more important security measures)
- Enhance programming productivity
- Contain most current base table data
- Use little storage space
- Provide customized view for user
- Establish physical data independence

Disadvantages of Views

- Use processing time each time view is referenced
- May or may not be directly updateable