# **Data Modeling Using the Entity-Relationship (ER) Model**

# ER Model Concepts

## 

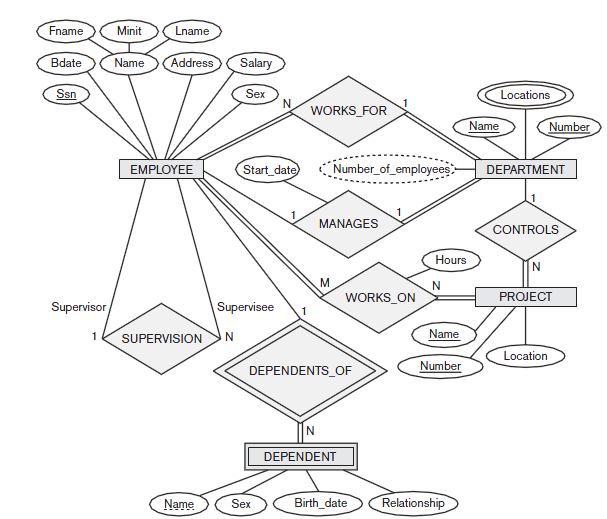
## Entities and Attributes

### Entities are specific objects or things in the mini-world that are represented in the database. For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT

### Attributes are properties used to describe an entity. For example an EMPLOYEE entity may have a Name, SSN, Address, Sex, BirthDate

### A specific entity will have a value for each of its attributes. For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘

### Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, subrange, enumerated type, …



# Types of Attributes

## Simple

### Each entity has a single atomic value for the attribute. For example, SSN or Sex.

## Composite

### The attribute may be composed of several components. For example, Address (Apt#, House#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName). Composition may form a hierarchy where some components are themselves composite.

## Multi-valued

### An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}.

# Types of Attributes

## In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels although this is rare. For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}.

# Entity Types and Key Attributes

# Entities with the same basic attributes are grouped or typed into an entity type. For example, the EMPLOYEE entity type or the PROJECT entity type.

## An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type. For example, SSN of EMPLOYEE.

## A key attribute may be composite. For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).

## An entity type may have more than one key. For example, the CAR entity type may have two keys:

### VehicleIdentificationNumber (popularly called VIN) and

### VehicleTagNumber (Number, State), also known as license\_plate number.

# Relationships and Relationship Types

## A relationship relates two or more distinct entities with a specific meaning. For example, EMPLOYEE John Smith works on the ProductX PROJECT or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.

## Relationships of the same type are grouped or typed into a relationship type. For example, the WORKS\_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.

## The degree of a relationship type is the number of participating entity types. Both MANAGES and WORKS\_ON are binary relationships.

## More than one relationship type can exist with the same participating entity types. For example, MANAGES and WORKS\_FOR are distinct relationships between EMPLOYEE and DEPARTMENT, but with different meanings and different relationship instances.

# Weak Entity Types

## An entity that does not have a key attribute

## A weak entity must participate in an identifying relationship type with an owner or identifying entity type

## Entities are identified by the combination of:

### A partial key of the weak entity type

### The particular entity they are related to in the identifying entity type

## **Example:**

## Suppose that a DEPENDENT entity is identified by the dependent’s first name and birhtdate, *and* the specific EMPLOYEE that the dependent is related to. DEPENDENT is a weak entity type with EMPLOYEE as its identifying entity type via the identifying relationship type DEPENDENT\_OF

# Constraints on Relationships

## Constraints on Relationship Types

### ( Also known as ratio constraints )

### Maximum Cardinality

#### One-to-one (1:1)

#### One-to-many (1:N) or Many-to-one (N:1)

#### Many-to-many

### Minimum Cardinality (also called participation constraint or existence dependency constraints)

#### zero (optional participation, not existence-dependent)

#### one or more (mandatory, existence-dependent)

# Relationships and Relationship Types

## We can also have a **recursive** relationship type.

## Both participations are same entity type in different roles.

## For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).

## In following figure, first role participation labeled with 1 and second role participation labeled with 2.

## In ER diagram, need to display role names to distinguish participations.

# Attributes of Relationship types

## A relationship type can have attributes; for example, HoursPerWeek of WORKS\_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.

## **Structural constraints on relationships:**

## **Cardinality ratio** (of a binary relationship): 1:1, 1:N, N:1, or M:N

### **SHOWN BY PLACING APPROPRIATE NUMBER ON THE LINK.**

## **Participation constraint** (on each participating entity type): total (called *existence dependency*) or partial.

## **SHOWN BY DOUBLE LINING THE LINK**

## NOTE: These are easy to specify for Binary Relationship Types.

# **Alternative (min, max) notation for relationship structural constraints:**

## Specified on *each participation* of an entity type E in a relationship type R

## Specifies that each entity e in E participates in *at least* min and *at most* max relationship instances in R

## Default(no constraint): min=0, max=n

## Must have minmax, min0, max 1

## Derived from the knowledge of mini-world constraints

## Examples:

## A department has *exactly one* manager and an employee can manage *at most one* department.

### Specify (0,1) for participation of EMPLOYEE in MANAGES

### Specify (1,1) for participation of DEPARTMENT in MANAGES

## An employee can work for *exactly one* department but a department can have *any number of employees*.

### Specify (1,1) for participation of EMPLOYEE in WORKS\_FOR

### Specify (0,n) for participation of DEPARTMENT in WORKS\_FOR

# **Relationships of Higher Degree**

## Relationship types of degree 2 are called **binary**

## Relationship types of degree 3 are called **ternary** and of degree n are called **n-ary**

In general, an n-ary relationship *is not* equivalent to n binary relationships

# **The Relational Data Model and Relational Database Constraints**

# **Relational Model Concepts**

## The relational Model of Data is based on the concept of a Relation.

## A Relation is a mathematical concept based on the ideas of sets.

## The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

# **INFORMAL DEFINITIONS**

## RELATION: A table of values

### A relation may be thought of as a **set of rows**.

### A relation may alternately be though of as a **set of columns**.

### Each row represents a fact that corresponds to a real-world **entity** or **relationship**.

### Each row has a value of an item or set of items that uniquely identifies that row in the table.

### Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.

### Each column typically is called by its column name or column header or attribute name.

# **FORMAL DEFINITIONS**

## A **Relation** may be defined in multiple ways.

## The **Schema** of a Relation: *R* (A1, A2, .....An)

## Relation schema *R* is defined over **attributes** A1, A2, .....An

## For Example -

## CUSTOMER (Cust-id, Cust-name, Address, Phone#)

## Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

## A **tuple** is an ordered set of values

## Each value is derived from an appropriate domain.

## Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.

## **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">** is a tuple belonging to the CUSTOMER relation.

## A relation may be regarded as a ***set of tuples*** (rows).

## Columns in a table are also called attributes of the relation.

## A **domain** has a logical definition: e.g., “USA\_phone\_numbers” are the set of 10 digit phone numbers valid in the U.S.

## A domain may have a data-type or a format defined for it. The USA\_phone\_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.

## An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”.

# **CHARACTERISTICS OF RELATIONS**

## **Ordering of tuples in a relation r(R)**: The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.

## **Ordering of attributes in a relation schema R** (and of values within each tuple): We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be *ordered* .

## (However, a more general *alternative definition* of relation does not require this ordering).

## **Values in a tuple**: All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

# **Relational Integrity Constraints**

## Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:

### **Key** constraints

### **Entity integrity** constraints

### **Referential integrity** constraints

# **Key Constraints**

## **Superkey** of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] **≠** t2[SK].

## **Key** of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

### **Example**: The CAR relation schema:

### CAR(State, Reg#, SerialNo, Make, Model, Year)

### has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.

## If a relation has *several* **candidate keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are *underlined*.

# **Key Constraints**

# **Entity Integrity**

## **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.

## S = {R1, R2, ..., Rn}

## **Entity Integrity**: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples.

## t[PK] **≠** null for any tuple t in r(R)

## Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

# **Referential Integrity**

## A constraint involving *two* relations (the previous constraints involve a *single* relation).

## Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.

## Tuples in the *referencing relation* R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R2. A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].

## A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

## Statement of the constraint

## The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be either:

## (1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R2,, or..

## (2) a null.

# **Other Types of Constraints**

## Semantic Integrity Constraints:

## E.g., “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”

## A *constraint specification language* may have to be used to express these

## SQL allows triggers and assertion to allow for some of these

# **Update Operations on Relations**

## INSERT a tuple.

## DELETE a tuple.

## MODIFY a tuple.

## 

## Integrity constraints should not be violated by the update operations.

## Several update operations may have to be grouped together.

## Updates may *propagate* to cause other updates automatically. This may be necessary to maintain integrity constraints.

# **Update Operations on Relations**

## In case of integrity violation, several actions can be taken:

### Cancel the operation that causes the violation (REJECT option)

### Perform the operation but inform the user of the violation

### Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)

### Execute a user-specified error-correction routine