**Data Modeling 2020-02-09**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Model**  A collection of concepts for describing data.  Like other models, we are not providing the "final" product. We want to be able to show the data model and change it before we "finalize" the physical database.  **Schema** A description of a collection of data including how it is organized and how the data may be manipulated.  **Entity** Objects. Usually denote a person, place, or thing.  Examples: Student, Hospital, Library, Book, Insurance Policy, Video Rental, Address, Employee, Department  **Relationship** A connection between objects. | **Example #1: Sample relationships**   |  |  | | --- | --- | | **Relationship** | **Component Objects** | | enrolled-in | Student, Section | | has-offering | Course, Section | | lives-at | Person, Address | | works-in | Employee, Department | | manages | Employee, Employee | | manages-dept | Employee, Department | | supplies | Supplier, Part, Customer | | marriage | Person, Person | | has-prereq | Course, Course | | payments | Student, PaymentItem | | isa-employee | Professor, Employee | | isa-employee | Admin, Employee |   Some relationships are recursive between the same entity types (e.g., manages, marriage, has-prereq). |
| **Characterizing Relationships**   1. arity - (the number of objects involved)   binary relates two entities  ternary relates three entities   1. cardinality - the number of each entity involved   one-to-one (1:1)  one-to-many (1:N)  many-to-many (N:M)   1. **characteristic** vs **association**   characteristic connects an object to an item that characterizes it; typically, the object has only one value for that characteristic    association connects two or more entities | **Example #2: arity**  binary enrolled-in relates Students and Sections  ternary Supplies relates the Parts supplied by a Supplier to Customers.  **Example #3: cardinality**  one-to-one (1:1) marriage  one-to-many (1:N) one Manager manages many Employees  many-to-many (N:M) a Student is enrolled in many Courses; there are many Students in a Course  **Example #4: characteristic vs association**  **Characteristics for Student?**  studentId, name, totalGradePoints, totalCreditHours  **Characteristics for Course?**  ??  **Characteristics for Person?**  name, birthDate, gender  **Associations for Student?**  ??  **Characteristics for Employee?**  ??  What about student and classification (FR, SO, JR, SR, GR)?  ??  What about student and major?  Does the university allow multiple majors? |
| **Characterizing Relationships continued**   1. **optionality** - whether the relationship is required for an entity to exist. 2. **isa**  - specifies that an entity **isa** subclass of another entity. The subclass can be generalized to the superclass. | **Example #5: optionality**   * Must a student be enrolled-in a Section? When a student is initially admitted to the university, he/she doesn't have any enrollments.   Must a course have an offering of a section? ??   * Does a Section require a Course? The Course must exist for the Section to exist. * An Employee must work-in a Department. A Department could be initially created without any employees (yet). * A marriage requires that both persons exist. Each person can exist without being married.   **Example #6: isa**   * A professor isa employee; therefore, a professor inherits characteristics from employee . A professor may have his/her own attributes such as researchArea (e.g., AI, PL, ARCH, DB, CRYPTO, GEO) * An admin isa employee; therefore, an admin inherits characteristics from employee. An admin may have his/her own attributes such as specialty (PAYROLL, LINUXADMIN, MSADMIN). |
| **What should a Data Model do?**   * Clearly define entities, including the characteristics. * Clearly define relationships between entities, including the five ways of characterizing relationships. Relationships are important business rules (constraints).   What should an **implementation** of a data model do?   * Include a Data Definition Language(DDL) for describing the **structure**, allowable **operations**, and the **constraints** on the database. * Provide a mechanism for describing **subschemas** (**views**) of the database. Why? * simplifies what is needed. Ignore what is of no relevance * enhances data independence by not seeing what isn't needed, * restricts access of certain attributes and tables which the user shouldn't see because of security or privacy reasons. * Include a Data Manipulation Language (DML) for retrieving, adding, deleting, and updating the database. The language is characterized as: * Procedural (tell it how to achieve the result) or non-procedural (tell it what you want) * Record-at-a-time or set-at-a-time | **Example #7: Views**  Suppose we have the following Employee table:  Employee(empId, name, department, annualSalary, empStatusCd, birthDate))   * empId uniquely defines an Employee * empStatusCd specifies an employee's status (e.g., Applied, Working, Retired, Terminated, Quit) * Suppose the company has a policy against employees sharing their salary with other employees   With SQL, we could create the table using:  CREATE TABLE Employee (empId CHAR(6)  , name VARCHAR(40)  , department CHAR(3)  , annualSalary DECIMAL(10,2)  , empStatusCd CHAR(1)  , birthDate DATE);  Some applications need to know employee names; however, we don't want to provide the annualSalary nor birthDate to those applications. A Data Definition Language can provide a view:  CREATE VIEW ViewEmpName as  SELECT empId, name, department FROM  Employee;  We can then provide access to all users (PUBLIC) for that view  GRANT SELECT ON ViewEmpName TO PUBLIC; |
| **Relational Data Model EXAM SH**  **Relational Data Model** is a data model where data is organized into ***relations***.    **Relation** A table with rows (database records) and columns (fields). A relation, R, over a set of domains D1, …, Dn is a set of n-tuples <d1, …,dn> where each di is an element of Di  **Domain** The set of possible values for a column.  **Tuple** A row in the table. | **Example #8: EmployeeWorks**   |  |  |  | | --- | --- | --- | | **Name** | **Dept** | **Manager** | | Gauss | Math | Euclid | | Newton | Math | Euclid | | Einstein | Physics | Archimedes |   Since a relation is a set, it follows that:  i. No two tuples are identical.  ii. Tuples are unordered. (relations differ from files in those two properties)  iii. The order of the columns is insignificant. (relations differ from math relations in this property) |
| **Relational Model Example**   * Show a relation name followed by a list of its attributes * Underline the attributes in the primary key. | **Example #9: Relational Model Example for Student DB**  Student(studentNr, name, classCd, totalGradePoints, totalHours)  Course(courseNr, title, hours, discipline)  Section(courseNr, semester, sectionNr, prof)  Enroll(studentNr, courseNr, semester, sectionNr, grade) |
| **Supplier Parts DB Example**  We will use this classical schema in MANY examples.  SUPPLIER(S#, SNAME, STATUS, CITY)  S# - supplier number  SNAME - supplier name(e.g., SMITH, JONES),  STATUS - a number representing the quality of this supplier with the higher number representing a better quality of service (e.g., 10, 20)  PART(P#, PNAME, COLOR, WEIGHT)  P# - part number  PNAME - part name (e.g., NUT)  COLOR - color of the part (e.g., RED)  WEIGHT - weight of the part in grams  SHIPMENT(S#, P#, QUANTITY)  S# - supplier number  P# - part number  QUANTITY - the number of this part shipped by this supplier  **Constraints**: keys are underlined  S5 ships nothing, s1 ships every part. | **Example #10: Supplier Parts DB sample Data**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **SUPPLIER**   |  |  |  |  | | --- | --- | --- | --- | | **S#** | **SNAME** | **STATUS** | **CITY** | | S1 | SMITH | 20 | LONDON | | S2 | JONES | 10 | PARIS | | S3 | BLAKE | 30 | PARIS | | S4 | FORD | 20 | LONDON | | S5 | ADAMS | 30 | ATHENS |   **PART**   |  |  |  |  | | --- | --- | --- | --- | | **P#** | **PNAME** | **COLOR** | **WEIGHT** | | P1 | NUT | RED | 12 | | P2 | BOLT | GREEN | 17 | | P3 | SCREW | BLUE | 17 | | P4 | SCREW | RED | 14 | | P5 | CAM | BLUE | 12 | | P6 | COG | RED | 19 | | **SHIPMENT**   |  |  |  | | --- | --- | --- | | **S#** | **P#** | **QUANTITY** | | 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 | | |
| **Supplier Parts Queries (procedural)**  **Retrievals:**  get first *table* where *condition*;  gets the **first** row matching the specified *condition* in the specified *table.*  get next *table* where *condition*;  gets the **next** row matching the specified *condition* in the specified *table.* | **Example #11: Find the supplier number for suppliers who supply part P2.**  get first SHIPMENT where P# = 'P2';  while ( SHIPMENT found):  print SHIPMENT.S#  get next SHIPMENT where P# = 'P2'  end while.  results = {S1, S2, S3, S4}  **Example #12: Find the part number for parts supplied by supplier S2.**  get first SHIPMENT where S# = 'S2';  while ( SHIPMENT found):  print SHIPMENT.P#  get next SHIPMENT where S# = 'S2'  end while.  results = {P1, P2}  **Example #13: Find suppliers (print S#) of red parts. Parts 1, 4, and 6**  get first PART where COLOR = 'RED'  while (PART found):  get first SHIPMENT where P# = PART.P#  while (SHIPMENT found):  print SHIPMENT.S#  get next SHIPMENT where P# = PART.P#  end while  get next PART where COLOR = 'RED'  end while |
| **Supplier Parts Insert, Update and Delete**  **Insertions:**  insert into *table* (*columnList*) values (*valueList*);  **Deletions**:  delete from *table* where *condition*;  **Updates**:  update *table* set *column*=*value …* where *condition*;  We can list multiple column value assignments. | **Example #14: insertion of a new supplier S6.**  insert into SUPPLIER (S#, SNAME, STATUS, CITY)  VALUES ('S6', 'JACKSON', 15, 'DALLAS');  **Example #15: delete the shipment for supplier S2 of part P2.**  delete from SHIPMENT where S# = 'S2' and P# = 'P2'.  **Example #16: show that supplier S1 has moved from London to Paris.**  update SUPPLIER set CITY = 'Paris' where CITY = 'London';  **Why is that in red?**  Set the city to Paris for S1 and S4.  Correct answer?  update SUPPLIER set CITY = 'Paris' where S# = 'S2'. |
| **Entity Relationship Model**  This data modeling approach is used for diagrams; however, physical implementations use the Relational Model. There are multiple diagramming techniques including ER Diamond Relationships and UML.  ER Diamond - originated with Peter Chen   * diamonds - associations between entities * entities - rectangles * ovals - characteristics | **Example #17: ER Diamond Diagram for Supplier Parts DB**    The cardinalities are shown on either side of the diamond relationships. |
| **Unified Modeling Language (UML)**  This is more popular than the ER Diamond approach and has been adopted by many UML diagramming tools.   * Rectangles surround entities and their characteristics (i.e., attributes). * Relationships use lines. * **isa** relationships include an additional triangle which point to the superclass * cardinality and optionality are shown with either the crow's feet or numeric range approach.     In the UML crow's feet diagram on the right:   * For a SHIPMENT to exist, it must have a Supplier and it must have a PART. That is indicated by exactly ONE of each of those. (recognized by horizontal bar) ind below * SHIPMENT is optional for SUPPLIER to exist. There can be zero to many of them for a particular SUPPLIER. * SHIPMENT is optional for PART to exist. There can be zero or many shipments of parts.   KEY | **Example #18: UML Crows Feet for Supplier Parts DB** |
| **UML Numeric Range**  The UML diagram on the right uses numeric ranges instead of "crow's feet".  Notice that the associated entity SHPMENT contains the key of Supplier and Part. This is different from what you have done with Class Models in Software Engineering. This is NECESSARY in data modeling so that we can clearly show the key of SHIPMENT.  We **do not show many- to-many relationships** in a data model; instead, we replace them with multiple one-to-many relationships.  Universal | **Example #19: UML Numeric Range for Supplier Parts DB**    **1 Exactly one**  **0 Zero or one**  **0..N Zero or many**  **1..N One or many** |
| **UML isa**  **isa** relationships are shown using a triangle with the top part pointing to the superclass. In the diagram, Professor and Admin are subclasses of Employee. Both of those classes inherit the attributes of Employee.  Although not necessary for **isa** relationships, we show the empId in each to indicate that these have the same key as Employee. | **Example #20: UML isa relationships** |
| **Additional Example with isa and Associations** | **Example #21: UML showing isa and associations** |
| **Data Modeling Methodology**  The information represented in a data model is gathered by interviewing primary business users (or groups of users), reviewing forms, and reading business procedures. Good facilitation skills are very useful. For complex business problems, this may take weeks or months.  1. Identify the first cut **entities**. These are usually **nouns** having significant business value.  2. Identify the first cut **relationships** between entities. These are usually **verbs**.  3. Without attributes, show the entities and relationships in a diagram.  4. Discard any entities/relationships which are irrelevant to the business problem.  5. Identify constraints. Determine the cardinality and optionality of the relationships. It is often necessary to ask the same question in many ways. This involves questions like:   * Are there more than one X for a Y? * How many Xs are there for a Y? * It seems like Y is dependent on X. Must we have recorded an X before we can record a Y? * Does X have to have a Y? * If all the Ys are removed, should we remove the X? * If an X is deleted, should its corresponding Ys be deleted? * Is X allowed to be deleted if it has corresponding Ys?   6. Identify characteristics for each entity. These describe the entity in more detail.  7. Eliminate entities/characteristics which are synonyms.  8. Identify potential primary keys.  9. Based on the primary keys, validate relationships.  10. Refine the data model.  11. Precisely define all attributes including value ranges. | **Example #22: Fizzle Data Model**  Superior Products, Inc. manufactures Fizzles, the latest craze in children's toys. Fizzles can only be ordered from the Fizzle Web page. To place an order, the customer must be a member of the Fizzle Fan Club (FFC).  To become a member of the FFC, a customer must provide his name, address, credit card number, credit card company (e.g., Discover, Visa), and expiration date. With membership, the customer receives the monthly Fizzle World magazine for the low annual dues of $20 (a bargain at twice the cost), and members receive an 8-digit FFC membership number.  FFC members can order Fizzles. To order a Fizzle, the member must specify the quantity of Fizzles wanted in each of the vibrant colors (Fizzle Frazzle Red, Fizzle Yizzle Yellow, or Fizzle Babble Blue). For each order, Superior Products creates a purchase order for the FFC member's showing the quantity of each Fizzle type requested. Each purchase order is given a unique purchase order number. Superior Products only ships an order once the order has been completely filled. Customers can check the Fizzle Web Page for the expected shipment date.  FFC members frequently place multiple orders.  Superior Products keeps track of the current inventory of each type of Fizzle. The inventory includes the current number in stock, the reorder point (ROP), the current purchase price, and the total number sold to date.  Since Fizzles are selling well, Superior Products plans to sell many new types of Fizzles and other products. |
| **Fizzle Data Model**  **Proper Nouns:**  Superior Products, Inc. , Fizzles Fan Club, Fizzles,  **Common Nouns:**  Customer, Member, Toys, order, address, credit card number, credit card company, CC expiration date, membership, monthly magazine, annual dues, memberNr, Order quantity, colors, purchase order, purchase order number, expected shipment date, current inventory, number in stick, reorder point (ROP), current purchase price, total number sold to date, types of Fizzles, products  **Verbs:**  Order, manufactures, ships, selling  **Is color a characteristic of the product or is it a characteristic of the purchase order?**  **Relationships**  Customer many orders  **Relational Model 🡺** | **What are the primary entities?**  Customer/Member  Purchase Order  Product/inventory  **What are characteristics of customer?**  Name, Address, Credit Card Number, CC expiration date, memberNR  **What are characteristics of product/inventory?**  Color, number in stock, ROP, current price, name, product type, product id,  **What are characteristics of purchase order?**  PurchaseOrderNr, memberNr,  **Relations:**  ??  **Show the UML diagram using numeric ranges.**  **Customer(**mbrNr, name, address, ccNr, ccCompany, ccExpirationD, annualRenewalDt, email**)**  **Product(**prodId, productType, color, stockQty, rop, currentPrice, productNm, roQty, numbersoldToDate**)**  **PurchaseOrder(**poNr, mbrNr, statusCd, shipmentDt, shipTrackingNr,)  **PurchaseOrderItem(**poNr, prodId, requestQty, fulfilledQty, cost, returnStatus, returnQty**)** |

**©2020 Larry W. Clark,** UTSA CS students may make copies for their personal use