Relational Databases

Thomas P. Sturm
University of St. Thomas

Today’s Software Business Challenge
Outline

Data Concepts
   Why learn about Relational Databases?
   Data - Information - Database
   Properties of Data Items
   Descriptors / Identifiers
   Sample Database

Relational Model
   Relational Database model
   Translation of Relational Terms
   Requirements of a Relation
   Advantages of the Relational Model
   Details of Department Relation
   Org. of Relations in Sample D.B.
   (Single Table) Relational Operations
   Relational Algebra
   Cartesian Product
   Joins

Logical Data Structures
   Attributes for a Relational model
   Full Functional Dependence
   Logical Data Structures (LDS)
   Basic LDS Components
   Example Relationships
   Handling an M-M Relationship
   Identifier Representation
   Sample Database
   LDS for Sample Database
   Modeling Concepts
   Map LDS to Well-Formed Relations
Why learn about Relational Databases?

• A way to put end users into direct touch with the information stored in computers.
• A way to increase the productivity of data processing professionals.
• Can obtain high-performance implementation of relational models
• “No surprises” theoretical underpinnings (no “special rules, no “that’s a feature, not a bug”)
• Universal acceptance from the smallest to the largest databases
• Readily available design tools
• A standardize language for doing queries (SQL)
Data - Information - Database

INFORMATION:
The meaning that a human assigns to data via the known conventions used in their representation.

DATA:
A formalized representation of facts, concepts, or instructions suitable for communication, interpretation, or processing by human or automatic means.

BASE:
The bottom of anything, considered as its support or foundation
The fundamental part of a thing
The chief ingredient of anything, viewed as its fundamental constituent
Base in its most general sense equals bottom, but, more specifically, implies a broad bottom by which something is held up or stabilized

DATABASE:
A collection of stored operational data used by the application systems of some particular enterprise.

A stable foundation to support an information process.
Properties of Data Items

- There are things about which data is collected - entities
- These entities can optionally have a name or names (both a class/type name and individual/instance names)
  
  **Entity Type:** A category, arbitrarily defined (but agreed to) so that membership within the category can be established, at least at a point in time, e.g. a department
  
  **Entity Instance:** Occurrence of a member in the category in the world, e.g. the payroll department

- There are certain things that it is desirable to describe about the entities. The various qualities (characteristics) of the entity that are to be described are referred to as attributes

- For each of these attributes for each entity there is potentially a value (taken from a legal set of values that obey certain constraints or rules)

- There is some structure in the data or stored values (relationships, associations, dependencies)

- Most important, the stored data items must have meaning
Descriptors / Identifiers

DESCRIPTOR:
A descriptor for an entity is an attribute/value pair.

IDENTIFIER:
An identifier is an attribute whose value is different for each entity.
• usually relegated to values necessarily different
• where necessary, an identifier can be made up of the concatenation of two attributes (which should be thought of as yet another attribute)

RETRIEVAL
can be based on:
• identifier (for an identifier, find some descriptors)
• descriptor (for a descriptor, find some identifiers of entities possessing the descriptor)
• location (for a particular location, retrieve the data that is stored there)
  • absolute location
  • relative location
(This third method of access is not allowable in the relational model)
Sample Database - Employees

Overview of Content:
The database contains organization, budget, and scheduling information for a software group that is developing an academic information system

Entities:
Employees - who have
- a name
- a job title
- a manager who, in turn, is an employee
- a hire date
- an hourly billing rate
- (possibly) a dollar annual bonus amount
- membership in a department which in turn has a name, location, and budget
- a set of assigned tasks on projects
  - each task by each employee on each project has a time estimate in hours
  - each project has a name, description, budget, and due date
Sample Database - Departments and Projects

Entities (continued)

Departments - which have
- a department number
- a department name
- a department location (room number)
- an annual dollar budget
- employees, who in turn have a name, job description, manager, hire date, hourly rate annual bonus, and a set of assigned tasks (as described above)

Projects - which have
- a project name
- a project description
- a project budget
- a project due date
- a set of tasks, each of which is to be performed by one or more employees (who in turn have a name, job description, manager, hire date, ...) with a time estimate for each employee for each task
Sample Database - Tasks

Entities (continued)

Tasks - each of which have

- the name of the employee working on the task (who in turn has name, job description, ...)
- the name of the project that the task is related to (which in turn has name, description, ...)
- the name of the task being performed
- the time estimate (in hours) of how long an employee will work on a particular type of task for a particular project
Sample Data  
*(stated in relational form)*

### Employees - *(Table name emp)*

<table>
<thead>
<tr>
<th>Ename</th>
<th>Job</th>
<th>Mgr</th>
<th>Hired</th>
<th>Rate</th>
<th>Bonus</th>
<th>DeptNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>allen</td>
<td>programmer</td>
<td>barger</td>
<td>09-jun-1991</td>
<td>30.00</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>barger</td>
<td>supervisor</td>
<td>turner</td>
<td>23-jan-1993</td>
<td>65.00</td>
<td>550.00</td>
<td>402</td>
</tr>
<tr>
<td>jones</td>
<td>programmer</td>
<td>radl</td>
<td>20-feb-1991</td>
<td>35.00</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>king</td>
<td>clerk</td>
<td>barger</td>
<td>22-feb-1991</td>
<td>18.00</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>martin</td>
<td>programmer</td>
<td>barger</td>
<td>09-nov-1991</td>
<td>25.00</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>olson</td>
<td>analyst</td>
<td>radl</td>
<td>28-apr-1991</td>
<td>55.00</td>
<td>0.00</td>
<td>401</td>
</tr>
<tr>
<td>pearson</td>
<td>programmer</td>
<td>radl</td>
<td>01-may-1991</td>
<td>30.00</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>radl</td>
<td>supervisor</td>
<td>turner</td>
<td>03-dec-1992</td>
<td>65.00</td>
<td>600.00</td>
<td>401</td>
</tr>
<tr>
<td>rogers</td>
<td>programmer</td>
<td>barger</td>
<td>08-sep-1992</td>
<td>25.00</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>smith</td>
<td>programmer</td>
<td>barger</td>
<td>17-dec-1990</td>
<td>35.00</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>sturm</td>
<td>clerk</td>
<td>radl</td>
<td>23-sep-1992</td>
<td>18.00</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>thomas</td>
<td>analyst</td>
<td>barger</td>
<td>03-dec-1992</td>
<td>50.00</td>
<td>0.00</td>
<td>402</td>
</tr>
<tr>
<td>turner</td>
<td>supervisor</td>
<td>turner</td>
<td>02-mar-1991</td>
<td>75.00</td>
<td>1000.00</td>
<td>400</td>
</tr>
<tr>
<td>vogel</td>
<td>consultant</td>
<td>turner</td>
<td>17-nov-1991</td>
<td>80.00</td>
<td></td>
<td>400</td>
</tr>
</tbody>
</table>

### Departments *(Table name dept)*

<table>
<thead>
<tr>
<th>DeptNo</th>
<th>Dname</th>
<th>Loc</th>
<th>Dbudget</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>programming</td>
<td>200</td>
<td>150000.00</td>
</tr>
<tr>
<td>401</td>
<td>financial</td>
<td>200</td>
<td>275000.00</td>
</tr>
<tr>
<td>402</td>
<td>academic</td>
<td>100</td>
<td>390000.00</td>
</tr>
<tr>
<td>403</td>
<td>support</td>
<td>300</td>
<td>7000.00</td>
</tr>
</tbody>
</table>

### Projects *(Table name proj)*

<table>
<thead>
<tr>
<th>Project_id</th>
<th>Description</th>
<th>Pbudget</th>
<th>Due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>admit</td>
<td>Admissions</td>
<td>15000.00</td>
<td>07-apr-1998</td>
</tr>
<tr>
<td>alumni</td>
<td>Alumni development</td>
<td>7500.00</td>
<td>30-jan-1999</td>
</tr>
<tr>
<td>billing</td>
<td>Student billing</td>
<td>11000.00</td>
<td>30-jan-1998</td>
</tr>
<tr>
<td>budget</td>
<td>Budgeting</td>
<td>12500.00</td>
<td>12-mar-1998</td>
</tr>
<tr>
<td>payroll</td>
<td>Payroll</td>
<td>9000.00</td>
<td>15-may-1998</td>
</tr>
<tr>
<td>records</td>
<td>Students records</td>
<td>6000.00</td>
<td>11-feb-1998</td>
</tr>
</tbody>
</table>
## Tasks (Table name *task*)

<table>
<thead>
<tr>
<th>Ename</th>
<th>Project_id</th>
<th>Tname</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>allen</td>
<td>admit</td>
<td>debug</td>
<td>25</td>
</tr>
<tr>
<td>allen</td>
<td>admit</td>
<td>implement</td>
<td>20</td>
</tr>
<tr>
<td>allen</td>
<td>billing</td>
<td>debug</td>
<td>30</td>
</tr>
<tr>
<td>allen</td>
<td>billing</td>
<td>implement</td>
<td>20</td>
</tr>
<tr>
<td>barger</td>
<td>admit</td>
<td>manage</td>
<td>15</td>
</tr>
<tr>
<td>barger</td>
<td>alumni</td>
<td>manage</td>
<td>10</td>
</tr>
<tr>
<td>barger</td>
<td>billing</td>
<td>manage</td>
<td>8</td>
</tr>
<tr>
<td>barger</td>
<td>records</td>
<td>manage</td>
<td>12</td>
</tr>
<tr>
<td>jones</td>
<td>billing</td>
<td>implement</td>
<td>35</td>
</tr>
<tr>
<td>jones</td>
<td>budget</td>
<td>implement</td>
<td>70</td>
</tr>
<tr>
<td>jones</td>
<td>payroll</td>
<td>debug</td>
<td>40</td>
</tr>
<tr>
<td>king</td>
<td>admit</td>
<td>clerical</td>
<td>25</td>
</tr>
<tr>
<td>king</td>
<td>alumni</td>
<td>clerical</td>
<td>9</td>
</tr>
<tr>
<td>king</td>
<td>records</td>
<td>clerical</td>
<td>9</td>
</tr>
<tr>
<td>martin</td>
<td>admit</td>
<td>implement</td>
<td>30</td>
</tr>
<tr>
<td>olson</td>
<td>admit</td>
<td>design</td>
<td>75</td>
</tr>
<tr>
<td>olson</td>
<td>alumni</td>
<td>design</td>
<td>40</td>
</tr>
<tr>
<td>olson</td>
<td>billing</td>
<td>design</td>
<td>20</td>
</tr>
<tr>
<td>olson</td>
<td>records</td>
<td>design</td>
<td>45</td>
</tr>
<tr>
<td>pearson</td>
<td>budget</td>
<td>debug</td>
<td>40</td>
</tr>
<tr>
<td>pearson</td>
<td>budget</td>
<td>implement</td>
<td>60</td>
</tr>
<tr>
<td>pearson</td>
<td>payroll</td>
<td>implement</td>
<td>80</td>
</tr>
<tr>
<td>radl</td>
<td>billing</td>
<td>design</td>
<td>15</td>
</tr>
<tr>
<td>radl</td>
<td>billing</td>
<td>manage</td>
<td>10</td>
</tr>
<tr>
<td>radl</td>
<td>budget</td>
<td>manage</td>
<td>15</td>
</tr>
<tr>
<td>radl</td>
<td>payroll</td>
<td>manage</td>
<td>20</td>
</tr>
<tr>
<td>rogers</td>
<td>records</td>
<td>debug</td>
<td>20</td>
</tr>
<tr>
<td>rogers</td>
<td>records</td>
<td>design</td>
<td>30</td>
</tr>
<tr>
<td>rogers</td>
<td>records</td>
<td>implement</td>
<td>45</td>
</tr>
<tr>
<td>smith</td>
<td>alumni</td>
<td>debug</td>
<td>30</td>
</tr>
<tr>
<td>smith</td>
<td>alumni</td>
<td>implement</td>
<td>90</td>
</tr>
<tr>
<td>smith</td>
<td>billing</td>
<td>implement</td>
<td>40</td>
</tr>
<tr>
<td>sturm</td>
<td>billing</td>
<td>clerical</td>
<td>38</td>
</tr>
<tr>
<td>sturm</td>
<td>budget</td>
<td>clerical</td>
<td>20</td>
</tr>
<tr>
<td>sturm</td>
<td>budget</td>
<td>debug</td>
<td>20</td>
</tr>
<tr>
<td>sturm</td>
<td>payroll</td>
<td>clerical</td>
<td>15</td>
</tr>
<tr>
<td>thomas</td>
<td>alumni</td>
<td>design</td>
<td>5</td>
</tr>
<tr>
<td>thomas</td>
<td>billing</td>
<td>design</td>
<td>45</td>
</tr>
<tr>
<td>thomas</td>
<td>budget</td>
<td>design</td>
<td>40</td>
</tr>
<tr>
<td>thomas</td>
<td>payroll</td>
<td>design</td>
<td>70</td>
</tr>
<tr>
<td>turner</td>
<td>billing</td>
<td>manage</td>
<td>12</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>turner</td>
<td>budget</td>
<td>design</td>
<td>45</td>
</tr>
</tbody>
</table>
Relational Database Model

“Codd's” Model

Developed in mid-1970’s

Based on the mathematical theory of relations

Codd's definition:
Given sets $S_1, S_2, \ldots, S_n$ (not necessarily distinct), $R$ is a relation on these $n$ sets if it is a set of $n$-tuples each of which has its first element from $S_1$, its second element from $S_2$, and so on.

We shall refer to $S_j$ as the $j^{th}$ domain of $R$.
$R$ is said to have degree $n$.
If $R$ has $m$ $n$-tuples (or just tuples), $R$ is said to have cardinality $m$. 
Translation of Relational Terms

<table>
<thead>
<tr>
<th>Relational Term</th>
<th>Loose Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Table</td>
</tr>
<tr>
<td>Tuple</td>
<td>Row</td>
</tr>
<tr>
<td>Degree</td>
<td># of attributes</td>
</tr>
<tr>
<td>Cardinality</td>
<td># of table entries</td>
</tr>
<tr>
<td>Domain</td>
<td>field-level edit criteria and integrity constraints</td>
</tr>
</tbody>
</table>

Conceptual (but not physical) ideas:
- A relation is a table or a flat file
  with n columns or fields
  and m rows or records

- Column (or field) j represents a set of values (from a possible set of values, S_j, the “domain”) for a particular attribute of all the entities

- Each row (or record represents a set of values for an entity, one for each attribute (column, field)

- Degree - number of columns (fields, domains)

- Cardinality - number of rows (records, entities, tuples)
Requirements of a Relation

All rows of the relation must have the same attributes in the same order

No repeating groups

Each row must be unique
  (No duplicate rows - if there are, they are “cast out”)

A set of columns that forms an identifier is the table $key$
Advantages of the Relational Model

Logical not physical model
- easy to communicate, *what* not *how*

Data Independence
- implementation independent

Record interconnections are dynamically generated based on data value
- (no user-visible navigation links)

Set-at-a-time database operations (relational operators) locate, permute, join, select, project, derive, order, format, present

Join - the operator that “connects” tables - is unrestricted
- it is not necessary to pre-define access paths
Details of Department Relation

attributes (columns)

<table>
<thead>
<tr>
<th>DeptNo</th>
<th>Dname</th>
<th>Loc</th>
<th>Dbudget</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>programming</td>
<td>200</td>
<td>150000</td>
</tr>
<tr>
<td>401</td>
<td>financial</td>
<td>200</td>
<td>275000</td>
</tr>
<tr>
<td>402</td>
<td>academic</td>
<td>100</td>
<td>390000</td>
</tr>
<tr>
<td>403</td>
<td>support</td>
<td>300</td>
<td>7000</td>
</tr>
</tbody>
</table>

tuple (row)

domains 1

domains 4
## Organization of Relations in Sample Database

<table>
<thead>
<tr>
<th>Relation (Entity type)</th>
<th>Attributes (Key underlined)</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp</td>
<td>(Ename, Job, Mgr, Hired, Rate, Bonus, DeptNo)</td>
</tr>
<tr>
<td>dept</td>
<td>(DeptNo, Dname, Loc, Dbudget)</td>
</tr>
<tr>
<td>task</td>
<td>(Ename, Project_id, Tname, Hours)</td>
</tr>
<tr>
<td>proj</td>
<td>(Project_id, Description, Pbudget, Due_date)</td>
</tr>
</tbody>
</table>
(Single Table) Relational Operations

- located relation
- selection
- projection
- entry-level derivations
- order
- file-level derivations
- formatting & presentation
Relational Algebra

Relational operators take one or two relations as their “operands” or arguments

Result of applying a relational operator to a relation (or pair of relations) is another relation

Consequently, relational operators can be used in sequence to achieve the desired results
Cartesian Product

If $R_1$ and $R_2$ are relations, the Cartesian product is written `SELECT * FROM R1, R2;` (in SQL)

A new relation is generated that consists of every tuple in $R_1$ followed by every tuple in $R_2$

relation empl

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
</tr>
</tbody>
</table>

relation group

<table>
<thead>
<tr>
<th>dept</th>
<th>loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

Cartesian product $empl \times group$

<table>
<thead>
<tr>
<th>empl.name</th>
<th>empl.age</th>
<th>empl.dept</th>
<th>group.dept</th>
<th>group.loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>
Relational Join Operation

Join
A form of parallel table lookup
Both tables must share a domain

Conceptual Step 1: Form the Cartesian product between the two relations

Conceptual Step 2: Apply join conditions to select a subset of the Cartesian product that accomplishes the lookup (selection)
Natural Join Operation
(Simple join, inner equijoin)

- Start with two different tables, form the Cartesian product (e.g. empl x group)

<table>
<thead>
<tr>
<th>empl.name</th>
<th>empl.age</th>
<th>empl.dept</th>
<th>group.dept</th>
<th>group.loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

- Select rows where values of a pair of fields are equal (e.g. empl.dept and group.dept)

<table>
<thead>
<tr>
<th>empl.name</th>
<th>empl.age</th>
<th>empl.dept</th>
<th>group.dept</th>
<th>group.loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

- Project all except the duplicated column

<table>
<thead>
<tr>
<th>empl.name</th>
<th>empl.age</th>
<th>dept</th>
<th>group.loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>able</td>
<td>20</td>
<td>35</td>
<td>100</td>
</tr>
<tr>
<td>baker</td>
<td>40</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>codd</td>
<td>60</td>
<td>45</td>
<td>200</td>
</tr>
<tr>
<td>date</td>
<td>30</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>
Expressing the Natural Join

The natural join is written:

```sql
SELECT * FROM EMPL, GROUP
WHERE EMPL.DEPT = GROUP.DEPT;
```

in SQL

The natural join performs a “table lookup” function by “looking up” data from the second table for a field in the first table.

Unfortunately, if no match is found for an item “looked up” in the first table, that row in the first table is “lost”
Attributes for a Relational model

Each entity instance has exactly one value for each attribute (within the scope of the data model)

- atomic
- repeating groups are not allowed
- vectors are not allowed
- pointers and other abstract references are not allowed
- values for a particular attribute come from a specified pool of values (called its domain)

An attribute (or a specific set of attributes) forms an identifier for each entity instance

- if the entity instances are different, so is the value (or set of values) for the attribute (or set of attributes)
- an identifier (key) must be found and cannot have a null value
- there may be more than one, especially since a set of attributes can be an identifier
- must be minimal (cannot discard any attribute without losing uniqueness)
Full Functional Dependence

If each value of an attribute has associated with it precisely one value for a second attribute, then that second attribute is *functionally dependent* on the first.

Example:

In the emp relation:
- *Ename* is an identifier, and we choose it as the primary key
- other attributes of the employee will then generally be functionally dependent on *Ename*
- so *Job* is functionally dependent on *Ename* (or *Ename* functionally determines *Job*)

All attributes in a relation must necessarily be functionally dependent on the primary key.

Have functional dependency if agreement on the first value necessarily implies agreement on the dependent value. But remember that the primary key can be a *set* of fields. Full functional dependence implies that there is no subset of the set of fields that has functional dependence.
Logical Data Structures (LDS)

Graphical means of
- naming and
- depicting
the types of data in a database

Simple, yet precise

Useful to
- technically-oriented analysts
- application-oriented users

Easy to read

Supports the design task
- logical structure design is hard
- tool aids the design task
- notation does not get in the way

Constructive approach

Considers semantics

Documents
- data dependencies
- identifiers
- entities
- needed relations
- “rules”
Basic LDS Components

Entity
- any type of thing about which information is maintained

\[ \text{entity\_name} \]

EXAMPLE

\[ \text{student} \]

Attribute
- a characteristic of exactly one entity (fully functionally dependent on the entity)

\[ \text{attribute\_name} \]

EXAMPLE: Student attributes

\[ \text{student} \]

\[ \text{student\_name} \]

\[ \text{student\_id\#} \]

\[ \text{soc\_sec\#} \]

Relationships
- an association between a pair of entities (or “roles”), one-to-one, one-to-many only

\[ \text{or} \]

\[ \text{but} \]

\[ \text{never} \]
Example Relationships

1 - 1 Example: Monogamous marriage

Can label relationship

1-M Example: Students of a college

- Need not label a relationship if it can be stated as:
  college of student / students of college
  or
  student has college / college has students
Handling an M-M Relationship

M-M Example: Brother - Sister

Problem: how do you represent the presence of sibling rivalry?

THIS WON'T WORK

SOLUTION
Identifier Representation

Identifier: a set of attributes or relationships that uniquely identify an instance of an entity

Example:

(college_name) college
(college#) college

(student_name) student
(student_id#) student
(soc_sec#) student
Sample Database

Employee: (emp)
attributes: Ename, Job, Mgr, Hired, Rate, Bonus

Department: (dept)
attributes: DeptNo, Dname, Loc, Dbudget

Task: (task)
attributes: Tname, Hours

Project: (proj)
attributes: Project_id, Description, Pbudget, Due_date

Relationships
• employees are members of a department
• employees have a manager who is an employee
• employees are assigned to tasks on projects
LDS for Sample Database
Modeling Concepts

Entities:
“it” must have
- identifier
- attributes
- relationships

“it” must be the focus of the system

need to develop for “it”:
- name
- description
- membership criteria

must examine roles within subsets of “it”

Attributes:
must be non-transitively fully functionally dependent on
the entity it describes

must develop for each attribute:
- name
- description
- domain definition
Modeling Concepts (Continued)

Identifiers:
- determine which attributes are part of it
- verify uniqueness
- establish “not null” requirements

Relationships:
- establish degree 1-1 or 1-M
- entity on 1 side must be functionally dependent on entity on M side
- develop:
  - name
  - definition
- incorporate constraints, rules
- note referential integrity
  - (values of foreign key must exist in key field of another relation)
  - (e.g. in the emp relation, if an employee is listed as being in department 402, then in the dept relation there must contain a row with a key value of 402)
## Map LDS to Well-Formed Relations

<table>
<thead>
<tr>
<th>LDS</th>
<th>Relational Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>relation name</td>
</tr>
<tr>
<td>attribute descriptor</td>
<td>attribute</td>
</tr>
<tr>
<td>single-valued relationship</td>
<td>attribute (foreign key)</td>
</tr>
<tr>
<td>descriptor</td>
<td></td>
</tr>
<tr>
<td>multi-valued relationship</td>
<td>nothing</td>
</tr>
<tr>
<td>descriptor</td>
<td></td>
</tr>
<tr>
<td>1-1 relationship</td>
<td>either or both relationship descriptors are attributes</td>
</tr>
<tr>
<td>1-M relationship</td>
<td>relationship descriptor with degree 1 (on the M side) is an attribute</td>
</tr>
</tbody>
</table>
LDS → Relations Examples

Example: College students

college
(college#, college_name)

student
F.K.
(student#, college#, student_name, soc_sec#)

Copyright © 1971-2002 Thomas P. Sturm  Relational Databases  37
Sample Database Relations

(See page 33 for the Sample Database LDS)

depth
(DeptNo, Dname, Loc, Dbudget)

ejm
F.K. in emp
(Ename, Job, Mgr, Hired, Rate, Bonus, DeptNo)

proj
(Project_id, Description, Pbudget, Due_date)

task
F.K. F.K.
(Tname, Ename, Project_id, Hours)
References


Codd, E. F. “Is Your DBMS Really Relational.” 
*Computerworld.* October 14, 1985.

Codd, E. F. “Relational Database: A Practical Foundation for Productivity.” *Communications of the ACM.* Vol. 25, Number 2 (February, 1982).

Conte, Paul. “Understanding Relational Data Bases.” 
*Computer Language.* Vol. 4, Number 5 (May, 1987).


Date, C. J. *An Introduction to Database Systems.* Volume II. Addison-Wesley. 1984.

Date, C. J. *Relational Database: Selected Writings.* Addison-Wesley. 1986.


Sturm, Thomas P. *Data Structures, Direct Access, and Database Management.*