Chapter 16: A Table with a View: Introduction to Database Concepts

Fluency with Information Technology
Third Edition

by
Lawrence Snyder

Differences Between Tables and Databases

- When we think of databases, we often think of tables of information
- Comparing Tables
  - Database tables
    - Metadata tag identifying each of the data fields
  - Spreadsheet tables
    - Rely on position to keep the integrity of their data
  - HTML tables
    - Data as table entries with no unique identity at all
    - Concerned only with how to display the data, not with its meaning

The Database's Advantage

- Metadata is key advantage of databases over other systems recording data as tables
- Two of the most important roles in defining metadata
  - Identify the type of data with a unique tag
  - Define the affinity of the data

XML: A Language for Metadata Tags

- Extensible Markup Language
  - Tagging scheme similar to HTML
  - No standard tags to learn
    - Self-describing, think up the tags you need
  - Works well with browsers and Web-based applications
  - Use a simple text editor
  - XML tag names cannot contain spaces

An Example from Tahiti

- Area in km^2 for Tahiti & neighboring islands

```
<?xml version = "1.0" encoding="ISO-8859-1" ?>
<archipelago>
<island><iName>Tahiti</iName>    <area>1048</area></island>
<island><iName>Moorea</iName>    <area>130</area></island>
<island><iName>Maiao</iName>     <area>9.5</area></island>
<island><iName>Mehetia</iName>   <area>2.3</area></island>
<island><iName>Tetiaroa</iName>  <area>12.8</area></island>
</archipelago>
```

Figure 16.1 XML file encoding data for the Windward Islands database. The first line states that the file contains XML tags.

An Example from Tahiti (cont'd)

- First line
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- File should be ASCII text
- File extension should be `.xml`

Table 16.1 Rules for writing XML.

Required first line	`<?xml version="1.0" encoding="ISO-8859-1"?>` must appear on the first line, starting in the first position.
First tag	The first tag encountered is the *root* element, and it must enclose all of the file's content; it appears on the second or possibly third line.
Closing tags	All tags must be closed.
Element naming	Observe these rules: • Names can contain letters, numbers, and underscore characters. • Names must not start with a number or punctuation character. • Names must not start with the letters xml (or XML, or Xml, etc.). • Names cannot contain spaces.
Case sensitivity	Tags and attributes are case sensitive.
Proper nesting	All tags must be well-nested.
Attribute quoting	All attribute values must be quoted; paired single quotes (apostrophes) or paired double quotes are okay; use "dumb" quotes only; choose 'opposite' quotes to enclose quoted values.
White space	White space is preserved and converted to a single space.
Comments	XML comments have the form `<!-- This is a comment. -->`.

Expanding the Use of XML

- Combine encodings of two archipelagos – the Windward and the Galapagos Islands
- Root element is the tag that encloses all of the content of the XML file
  - <archipelago> in Fig. 16.1
  - <geo_feature> in Fig. 16.2
- Indenting for readability and structure


```
<?xml version = "1.0"
      encoding="ISO-8859-1" ?>
<geo_feature>
   <archipelago>
      <a_name>Windward Islands
      </a_name>
      <island>
         <iName>Tahiti</iName>
         <area>1048</area>
      </island>
      <island>
         <iName>Moorea</iName>
         <area>130</area>
      </island>
      <island>
         <iName>Maiao</iName>
         <area>9.5</area>
      </island>
      <island>
         <iName>Mehetia</iName>
         <area>2.3</area>
      </island>
      <island>
         <iName>Tetiaroa</iName>
         <area>12.8</area>
      </island>
   </archipelago>
```

```
   <archipelago>
      <a_name>Galapagos Islands
      </a_name>
      <island>
         <iName>Isabella</iName>
         <area>4588</area>
         <elevation>1707</elevation>
      </island>
      <island>
         <iName>Fernandina</iName>
         <area>642</area>
         <elevation>1494</elevation>
      </island>
      <island>
         <iName>Tower</iName>
         <area>14</area>
         <elevation>76</elevation>
      </island>
      <island>
         <iName>Santa Cruz</iName>
         <area>986</area>
         <elevation>846</elevation>
      </island>
   </archipelago>
</geo_feature>
```

Figure 16.2 XML file for the Geographic Features database. XML ignores white space, so the text in the file has been indented for easier reading.

Attributes in XML

- Use attributes for additional metadata, not for additional content
  - Not good, name is content:

    <archipelago name="Galapagos">
  - Better to give alternate form of the data

<a_name accents="Galápagos">Galapagos</a_name>

Effective Design with XML Tags

- Identification Rule: Label Data with Tags Consistently
  - You can choose whatever tag names you with to name data, but once you've decided on a tag for a particular kind of data, you must always surround it with that tag.

Effective Design with XML Tags (cont'd)

- Affinity Rule: Group Related Data
  - Enclose in a pair of tags all tagged data referring to the same entity. Grouping it keeps it all together, but the idea is much more fundamental: Grouping makes an association of the tagged data items as being related to each other, properties of the same thing.
  - Groups together data for a single thing – an island
    - Association is among properties of an object

Effective Design with XML Tags (cont'd)

- Collection Rule: Group Related Instances
  - When you have several instances of the same kind of data, enclose them in tags; again, it keeps them together and implies that they are related by being instances of the same type.
  - Groups together data of several instance of the same thing – islands
    - Association is among the objects themselves (entities)

The XML Tree

- XML encodings of information produce hierarchical descriptions that can be thought of as trees
  - Hierarchy a consequence of how tags enclose one another and the data

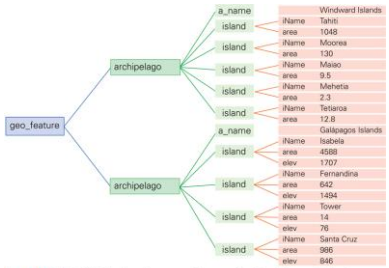

Figure 16.3 The XML displayed as a tree. The encoding from Figure 16.2 is shown with the root element (`geo_feature`) to the left and the leaves (content) shown to the right.

Tables and Entities

- A *relational database* describes the relationships among different kinds of data
  - Allows the software to answer queries about them

Entities

- Anything that can de identified by a fixed number of its characteristics (*attributes*)
  - Attributes have names and values
  - The values are the data that's stored in the table
- An entity defines a table
  - Name of the entity is the name of the table
  - Each attribute is assigned a column with column heading being the attribute name

Island		
Name	*Area*	*Elevation*
Isabela	4588	1707
Fernandina	642	1494
Tower	14	76
Santa Cruz	986	846

Figure 16.4 A table instance for the island entity.

Entities (cont'd)

- Entity instances
  - Rows of data
- Table instance
  - Any table containing specific rows
- Data type
  - Defines the form of the information that can be stored in a field
    - Number, text, image, …

    <name type="text"> <area type="number">

Properties of Entities

- A relational database table can be empty
- Instances Are Unordered
  - Order of the rows and columns does not matter in databases
  - Freedom to move the data is limited to exchanging entire rows or exchanging entire columns

Properties of Entities (cont'd)

- Uniqueness
  - No two rows can be the same
  - Two rows can have the same value for some attributes, just not all attributes

Properties Of Entities (cont'd)

- Keys
  - Any set of attributes for which all attributes are different is called a *candidate key*
  - Pick one and call it the *primary key* to decide uniqueness
  - Key must distinguish all potential and actual entities, not just those that happen to be in the table at a given time
  - If no combination of attributes qualify as a candidate key, assign a unique ID to each entity
    - Like a student ID number issued by school

Properties Of Entities (cont'd)

- Atomic Data
  - Not decomposable into any smaller parts
    - Separate fields for street, city, state, postal code
  - "Only atomic data" rule relaxed for certain types of data
    - Dates, times, currency

Database schemes

- Database schema – way to define a table
  - Collection of table definitions that gives the name of the table, lists the attributes and their data types, and identifies the primary key

```
Island
  iName            Text        Island Name
  area             Number      Area in square kilometers
  elevation        Number      Highest point on the island
Primary Key: iName
```

Figure 16.5 Database table definition for an `Island` table.

XML Trees and Entities

- Relational database tables and XML trees are not the same
- Relational databases are more restrictive than XML trees
  - The limits make them more powerful

Database Tables Recap

- Tables in databases have a structure that is specified by metadata
- The structure is separate from its content
- A table structures a set of entities
  - Things that we can tell apart by their attributes
- The entities of the table are represented as rows
  - Rows and columns are unordered
- Tables and fields should have names that describe their contents
  - Fields must be atomic (indivisible)
  - One of more attributes define the primary key

Operations on Tables

- A database is a collection of database tables
- Main use of database is to look up information
  - Users specify what they want to know and the database software finds it
- We can perform operations on tables to produce tables
- The questions we ask of a database are answered with a whole table
- Five fundamental operations that can be performed on tables: Select, Project, Union, Difference, Product

Nations

Name	text	*Common rather than official name*
Domain	text	*Internet top-level domain name*
Capital	text	*Nation's capital*
Latitude	number	*Approx. latitude of capital*
N_S	Boolean	*Latitude is N(orth) or S(outh)*
Longitude	number	*Approx. longitude of capital*
E_W	Boolean	*Longitude is E(ast) or W(est)*
Interest	text	*A short description of the country*

Primary Key: Name

Name	Dom	Capital	Lat	NS	Lon	EW	Interest
Ireland	IE	Dublin	52	N	7	W	History
Israel	IR	Jerusalem	32	N	35	E	History
Italy	IT	Rome	42	N	12	E	Art
Jamaica	JM	Kingston	18	N	77	W	Beach
Japan	JP	Tokyo	35	N	143	E	Kabuki

Figure 16.6 The `Nations` table definition and sample entries.

Select Operation

- Takes rows from one table to create a new table
  - Specify the table from which rows are to be taken, and the *test* for selection
  - Syntax: **`Select`** *Test* **`From`** *Table*
  - Test is applied to each rows of the table to determine if it should be included in result table
  - Test uses attribute names, constants, and relational operators
  - If the test is true for a given row, the row is included in the result table; otherwise it is ignored

```
Select Interest='Beach' From Nations
```


Name	Dom	Capital	Lat	NS	Lon	EW	Interest
Australia	AU	Canberra	37	S	148	E	Beach
Bahamas	BS	Nassau	25	N	78	W	Beach
Barbados	BB	Bridgetown	13	N	59	W	Beach
Belize	BZ	Belmopan	17	N	89	W	Beach
Bermuda	BM	Hamilton	32	N	64	W	Beach

Figure 16.7 Part of the table created by selecting countries with a Test for Interest equal to Beach.

Project Operation

- Builds a new table from the columns of an existing table
- Specify name of exiting table and the columns (field names) to be included in the new table
- Syntax: **`Project`** *Field_List* **`From`** *Table*
- The new table will have the number of columns specified and the same number of rows as the original table, unless
  - The new table eliminates a key field. If rows duplicate in the new table, duplicates will be eliminated

```
Project Name, Domain, Interest From Nations
```


Name	Dom	Word
Nauru	NR	Beach
Nepal	NP	Mountains
Netherlands	NL	Canals
New Caledonia	NC	Beach
New Zealand	NZ	Adventure

Figure 16.8 Sample entries for a `Project` operation on `Nations`.

Project Operation (cont'd)

- Can use Select and Project operations together to "trim" base tables to keep only some of the rows and some of the columns

```
Project Name, Domain, Lattitute From
  (Select Lattitude >= 60 AND NS='N' From Nations)
```


Name	Dom	Lat
Finland	FI	61
Greenland	GL	72
Iceland	IS	65
Norway	NO	60

Figure 16.9 `Northern`, the table of countries with northern capitals.

Union Operation

- Combines two tables (that have the same set of attributes)
- Syntax: *Table1* **+** *Table2*

```
ExtremeGovt = At60OrAbove + At45OrBelow
```

Name	Dom	Capital	Lat	NS	Lon	EW	Interest
Falkland Is	FK	Stanley	51	S	58	W	Nature
Finland	FI	Helsinki	61	N	26	E	Nature
Greenland	GL	Nuuk	72	N	40	W	Nature
Iceland	IS	Reykjavik	65	N	18	W	Geysers
Norway	NO	Oslo	60	N	10	E	Vikings

Figure 16.10 The `ExtremeGovt` table created with `Union`.

Difference Operation

- Remove from one table the rows also listed in a second table (remove from Table1 any rows also in Table2)
- Syntax: *Table1* **–** *Table2*

```
Nations - At60OrAbove
```

Product Operation

- Creates a super table with all fields from both tables
- Puts the rows together
  - Each row of Table 2 is appended to each row of Table 1
- Syntax: *Table1* **x** *Table2*

```
Super = Nations x Travelers
```



```
Travelers
  Friend     Text  A Traveling Companion
  Homeland   Text  Friend's Home Country
Primary Key: Friend
(a)
```

Friend	Homeland
Isabela	Argentina
Brian	South Africa
Wen	China
Clare	Canada

(b)

Figure 16.11 (a) The definition of the `Travelers` table, and (b) its values.

Name	Dom	Capital	Lat	NS	Log	EW	Interest	Friend	Homeland
Cyprus	CY	Nicosia	35	N	32	E	History	Clare	Canada
Czech Rep.	CZ	Prague	51	N	15	E	Pilsner	Isabella	Argentina
Czech Rep.	CZ	Prague	51	N	15	E	Pilsner	Brian	South Africa
Czech Rep.	CZ	Prague	51	N	15	E	Pilsner	Wen	China
Czech Rep.	CZ	Prague	51	N	15	E	Pilsner	Clare	Canada
Denmark	DK	Copenhagen	55	N	12	E	History	Isabella	Argentina

Figure 16.12 Some rows from the supertable that is the product of `Nations` and `Travelers`. For each row in `Nations` and each row in `Travelers`, there is a row in the product table that combines them.

Name	Friend
Chad	Wen
Chile	Isabella
China	Wen
Christmas Is.	Clare
Cocos Is.	Brian

Figure 16.13 A portion of the `Master` table of your friends' assignments.

Join Operation

- Combines two tables, like the Product Operation, but doesn't necessarily produce all pairings
  - If the two tables each have fields with a common data type, the new table combines only the rows from the given tables that match on the fields
  - Syntax: *Table1* ⋈ *Table2* **On** *Match*

Join Operation (cont'd)

- *Match* is a comparison test involving a fields from each table (*Table.Field*)
- When match is true for a row from each table produces a result row that is their concatenation

Join Applied

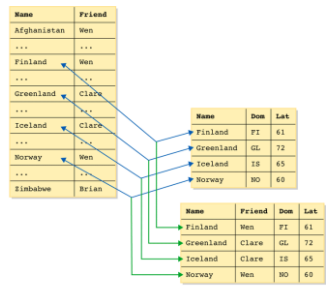

Figure 16.14 The `Join` operation: `Master` ⋈ `Northern`.

Join Applied (cont'd)

- Lookup operation on tables
  - For each row in one table, locate a row (or rows) in the other table with the same value in the common field; if found, combine the two; if not, look up the next row.
    - This match on equality is called a *natural join*
  - Possible to join using any relational operator, not just = (equality) to compare fields

Structure of a Database

- We want to arrange the information in a database in a way that users see a relevant-to-their-needs view of the data that they will use continually
- Physical database and logical view of the database

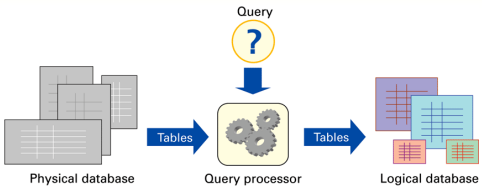

Figure 16.15 Structure of a database system. The physical database is the permanent repository of the data; the logical database, or view of the database, is the form of the database the users see. The transformation is implemented by the query processor, and is based on queries that define the logical database tables from the physical database tables.

Physical and Logical Databases

- The point of the two-level system is to separate the management of the data (physical database) from the presentation of the data (logical view of the database)

Physical Database

- Designed by database administrators
  - Fast to access
  - No redundancy/duplicating information
    - Multiple data can lead to inconsistent data
  - Backup copies in case of accidental data deletion or disk crash

Logical Database

- Creating specialized versions/views of the data for different users' needs
  - Creating a new copy from the single data each time

Queries

- A query is a specification using the five operations and Join that define a table from other tables
- SQL (Structured Query Language)
  - Standard database language to write queries

Defining Physical Tables

- Database schemes (schema)
  - Metadata specification that describes the database design

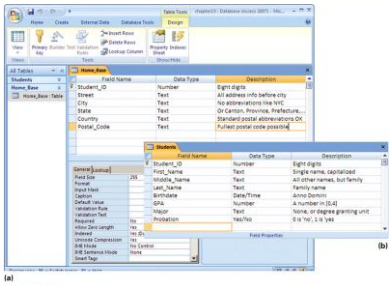

Figure 16.16 Table declarations from Microsoft Access 2007: (a) `Home_Base` table declaration shown in the design view; and (b) students table declaration. Notice that the key is specified by the tiny key next to `Student_ID` in the first column.

Connecting Database Tables by Relationships

- Student and Home_Base tables
  - The tables can have different security access restrictions based on their data
    - Other units can access Home_Base data without having access to more sensitive data in Student
  - Separate tables but not independent
    - `Student_ID` connects (establishes a relationship) between the two tables
      - Primary key

The Idea of Relationship

- A **relationship** is a correspondence between rows of one table and the rows of another table
  - Because the key `Student_ID` is used in each table, can not only find the address for each student (*Lives_At*), but can also find the student for each address (*Home_Of*)
- Relationship examples

Relationships in Practice

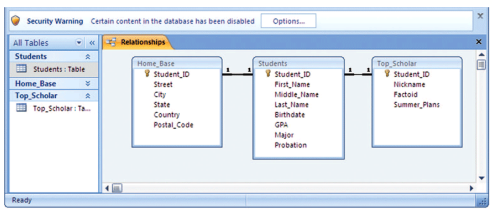

Figure 16.17 The *Relationships* window from the Microsoft Access database system; the 1-to-1 *Lives_At* and *Home_Of* relationships are shown between `Home_Base` and `Students`.

Defining Logical Tables

- Construction Using `Join`
  - Match on the common field of `Student_ID`

```
Master_List = Student JOIN Home_Base
  On Student.Student_ID = Home_Base.Student_ID
```

```
Student_ID
First_Name
Middle_Name
Last_Name
Birthdate
On_Probation
Street_Address
City
State
Country
Postal_Code
```

Figure 16.18 Attributes of the `Master_List` table. Being created from `Student` and `Home_Base` allows `Master_List` to inherit its data types and key (`Student_ID`) from the component tables.

Practical Construction Using QBE

- Query By Example
  - Given a template of a table we fill in what we want in the fields

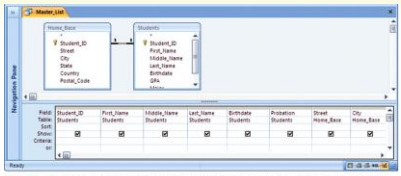

Figure 16.19 The Query By Example definition of the `Master_List` table from MS Access.

Figure 16.20 SQL query created from the Query By Example data in Figure 16.19.

The Dean's View

- Storing the Dean's Data
  - Top_Scholar is information of interest only to the dean

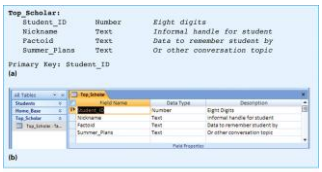

Figure 16.21 The `Top_Scholar` definition: (a) informal form, (b) in MS Access.

Creating a Dean's View

Deans_View

Name	Source Table	
Nickname	Top_Scholar	Used by the dean to seem "chummy"
First_Name	Student	Name information required because
Last_Name	Student	the dean forgets the person's actual name, being so chummy
Birthdate	Student	Is student of "drinking age"?
City	Home_Base	Hometown (given by city, state) is
State	Home_Base	important for small talk, but full address not needed by dean
Major	Student	Indicates what the student's doing in college besides hanging out
GPA	Student	How's student doing grade-wise
Factoid	Top_Scholar	Data to remember student by
Summer_Plans	Top_Scholar	Or other conversation topic

Figure 16.22 The Dean's View fields showing their source in physical database tables.

Join Three Tables into One

- `Join` using **Top_Scholar**, **Student**, and **Home_Base** tables matching on the `Student_ID` attribute across all three tables
- Trim the Table
  - `Project` – retrieve certain columns
- Join-then-trim strategy

Software Creates Dean's View

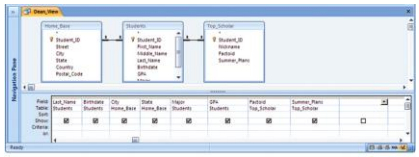

Figure 16.23 The Query By Example definition of the Dean's View table as expressed in Microsoft Access 2007.

Figure 16.24 SQL query created for the Dean's View by the Query By Example data in Figure 16.22.