CHAPTER 16

Introduction to Database Concepts
A Table with a View
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 (tags enclose all data that is logically related)
Tables and Entities

• A relational database describes the relationships among different kinds of data
  – Captures ideas like those defined in the Affinity and Collection rules
  – Allows software to answer queries about them

• Any relational DB can be described in XML
  – But it is not the case that every XML description defines a relational DB
Entities

• Anything that can be 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
<table>
<thead>
<tr>
<th>Island</th>
<th>Name</th>
<th>Area</th>
<th>Elevation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Isabela</td>
<td>4588</td>
<td>1707</td>
</tr>
<tr>
<td></td>
<td>Fernandina</td>
<td>642</td>
<td>1494</td>
</tr>
<tr>
<td></td>
<td>Tower</td>
<td>14</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>Santa Cruz</td>
<td>986</td>
<td>846</td>
</tr>
</tbody>
</table>

**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

• Attributes have a 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 (NULL instance)

• 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 attribute values 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
  - Values stored for attributes
  - 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

<table>
<thead>
<tr>
<th>Island</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iName</td>
<td>text</td>
<td>Island Name</td>
</tr>
<tr>
<td>area</td>
<td>number</td>
<td>Area in square kilometers</td>
</tr>
<tr>
<td>elevation</td>
<td>number</td>
<td>Highest point on the island</td>
</tr>
</tbody>
</table>

Primary Key: iName

Figure 16.5 Database table definition for an Island table.
Spreadsheets and Entities

- Relational database tables and spreadsheets are not the same
- Relational databases are more restrictive than Excel tables
- The limits make them more powerful
- Restriction is on the Collection rule
  - When entity instances are grouped, they must all have the same structure (to make a table)
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 new tables (that are not explicitly part of the schema)
• 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Capital</th>
<th>Lat</th>
<th>NS</th>
<th>Lon</th>
<th>EW</th>
<th>Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ireland</td>
<td>IE</td>
<td>Dublin</td>
<td>52</td>
<td>N</td>
<td>7</td>
<td>W</td>
<td>History</td>
</tr>
<tr>
<td>Israel</td>
<td>IR</td>
<td>Jerusalem</td>
<td>32</td>
<td>N</td>
<td>35</td>
<td>E</td>
<td>History</td>
</tr>
<tr>
<td>Italy</td>
<td>IT</td>
<td>Rome</td>
<td>42</td>
<td>N</td>
<td>12</td>
<td>E</td>
<td>Art</td>
</tr>
<tr>
<td>Jamaica</td>
<td>JM</td>
<td>Kingston</td>
<td>18</td>
<td>N</td>
<td>77</td>
<td>W</td>
<td>Beach</td>
</tr>
<tr>
<td>Japan</td>
<td>JP</td>
<td>Tokyo</td>
<td>35</td>
<td>N</td>
<td>143</td>
<td>E</td>
<td>Kabuki</td>
</tr>
</tbody>
</table>

**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: \textbf{Select Test From Table}
    - Test is a formula combining attribute names, constants, and relational operators
  - Test is applied to each row of the table to determine if it should be included in result table
    - If the test is true for a given row, the row is included in the result table; otherwise it is ignored

\textbf{Select Interest='Beach' From Nations}
Selection tests can test multiple attributes also, using logical operators

\[ \text{Select} \ \text{Interest}='\text{Beach}' \ \text{From} \ \text{Nations} \]

![Table showing countries with interest in Beach](image)

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

- Selection tests can test multiple attributes also, using logical operators

  \[ \text{Select} \ \text{Latitude}>60 \ \text{AND} \ \text{N}_S='N' \ \text{From} \ \text{Nations} \]

  \[ \text{Select} \ \text{Interest}=\text{'Beach'} \ \text{OR} \ \text{Interest}=\text{'Art'} \ \text{From} \ \text{Nations} \]
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 given in the operation, and the same number of rows as the original table, *unless*
  
  – The new table **eliminates** a key field; if the new table has duplicate rows, the duplicates will be eliminated
Example:

Project Name, Domain, Interest From Nations

<table>
<thead>
<tr>
<th>Name</th>
<th>Dom</th>
<th>Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nauru</td>
<td>NR</td>
<td>Beach</td>
</tr>
<tr>
<td>Nepal</td>
<td>NP</td>
<td>Mountains</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
<td>Canals</td>
</tr>
<tr>
<td>New Caledonia</td>
<td>NC</td>
<td>Beach</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
<td>Adventure</td>
</tr>
</tbody>
</table>

Figure 16.8 Sample entries for a Project operation on Nations.
Project Operation (cont'd)

- **Select** keeps some rows, all columns
- **Project** keeps all rows, some columns
- Can use **Select** and **Project** operations together to "trim" base tables, keeping some of the rows and some columns

\[ \text{Project Name, Domain, Latitude From} \]
\[ (\text{Select Latitude } \geq 60 \text{ AND N_S='N' From Nations}) \]
Example:

Project Name, Domain, Latitude From

(Select Latitude >= 60 AND N_S='N' From Nations)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dom</th>
<th>Lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finland</td>
<td>FI</td>
<td>61</td>
</tr>
<tr>
<td>Greenland</td>
<td>GL</td>
<td>72</td>
</tr>
<tr>
<td>Iceland</td>
<td>IS</td>
<td>65</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>60</td>
</tr>
</tbody>
</table>

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

• Combines two tables (that have the same set of attributes)
• Syntax: \textit{Table1} \textbf{+} \textit{Table2}

Using names to save tables...

\texttt{At60orAbove} =
\quad (\texttt{Select} \enspace \texttt{Latitude} \geq 60 \text{ AND } \texttt{N\_S}='N' \text{ From} \enspace \texttt{Nations})

\texttt{At45orBelow} =
\quad (\texttt{Select} \enspace \texttt{Latitude} \geq 45 \text{ AND } \texttt{N\_S}='S' \text{ From} \enspace \texttt{Nations})

\texttt{ExtremeGovt} = \texttt{At60orAbove} \textbf{ + } \texttt{At45orBelow}
Example:

\[ \text{ExtremeGovt} = \text{At60OrAbove} + \text{At45OrBelow} \]

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

**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
• Example:
  - Nations - At60orAbove
  - Nations - (Select Latitude >= 60 AND N_S='N' From Nations)

Creates a table with all Nations rows except the countries above 60 N latitude
Join Operation

• The five basic operations (select, project, +, -, x) are all we need to make any relational table
  – other operations we can imagine can be done with combinations of the basic 5

• One combination is so commonly wanted and useful that we name it and provide it as a direct operation: Join
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 \( \bowtie \) Table2 On Match
Join Operation (cont'd)

• “Bowtie” symbol suggests a special form of product where the tables “match up”

• Match is a comparison test involving fields from each table (Table.Field)

• When match is true for a row from each table it produces a result row that is their concatenation
Join Operation Applied

Consider the table \( \text{Northern} = (\text{Project Name, Domain, Latitude From At60orAbove}) \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dom</th>
<th>Lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finland</td>
<td>FI</td>
<td>61</td>
</tr>
<tr>
<td>Greenland</td>
<td>GL</td>
<td>72</td>
</tr>
<tr>
<td>Iceland</td>
<td>IS</td>
<td>65</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>60</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chad</td>
<td>Wen</td>
</tr>
<tr>
<td>Chile</td>
<td>Isabella</td>
</tr>
<tr>
<td>China</td>
<td>Wen</td>
</tr>
<tr>
<td>Christmas Is.</td>
<td>Clare</td>
</tr>
<tr>
<td>Cocos Is.</td>
<td>Brian</td>
</tr>
</tbody>
</table>

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

Consider this Join

Master \( \text{Northern On Master.Name=Northern.Name} \)
Join Applied

**Figure 16.14** The Join operation: Master $\bowtie$ 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 (stored on disk)
- Logical view of the database (made on the fly and customized for a user)
**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)
  – All users work with the same physical database
  – Different users will work with different views, one for each
Physical Database

• Designed by database administrators
  – Optimized structure for fast to access
  – No redundancy/duplicated information
    • Multiple data copies, (or multiple representations, multiple versions of data), can lead to inconsistent data values
  – 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 copies of view tables from the single physical database each time
  – Each user gets to see the data he/she cares about, and is not cluttered with data they don’t need
Queries

- A query is a *specification* using the five basic operations, and Join, that defines a new table from other tables
- New table is the *result* of the query
- SQL (Structured Query Language)
  - widely-used standard database language for query writing
Defining Physical Tables

• Database schemes (schema)
  – Metadata specifications that describes the overall design of a database
  – Gives the table names, fields, types
  – When we want to analyze a DB design, we examine its schema
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.
### Student

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_ID</td>
<td>number</td>
<td>Eight digits</td>
</tr>
<tr>
<td>First_Name</td>
<td>text</td>
<td>Single name, capitalized</td>
</tr>
<tr>
<td>Middle_Name</td>
<td>text</td>
<td>All other names, but family</td>
</tr>
<tr>
<td>Last_Name</td>
<td>text</td>
<td>Family name</td>
</tr>
<tr>
<td>Birthdate</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>Grade_Point</td>
<td>number</td>
<td>$0 \leq \text{GPA} \leq 4$</td>
</tr>
<tr>
<td>Major</td>
<td>text</td>
<td>None, or degree granting unit</td>
</tr>
<tr>
<td>On_Probation</td>
<td>Boolean</td>
<td>$0$ is ‘no’; $1$ is ‘yes’</td>
</tr>
</tbody>
</table>

Primary Key: Student_ID

### Home_Base

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_ID</td>
<td>number</td>
<td>Eight digits</td>
</tr>
<tr>
<td>Street</td>
<td>text</td>
<td>All address info before city</td>
</tr>
<tr>
<td>City</td>
<td>text</td>
<td>No abbreviations like NYC</td>
</tr>
<tr>
<td>State</td>
<td>text</td>
<td>Or province, canton, prefecture . . .</td>
</tr>
<tr>
<td>Country</td>
<td>text</td>
<td>Standard postal abbreviations OK</td>
</tr>
<tr>
<td>Postal_Code</td>
<td>text</td>
<td>Full postal code</td>
</tr>
</tbody>
</table>

Primary Key: Student_ID
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 the two tables (establishes a relationship between them)
      - Primary key
The Idea of Relationship

• Relationship examples
  
  *Father_of*, relationship between man and child
  *Daughter_of*, relationship between girl and parents
  *Employed_by*, between people and companies
  *Stars_in*, relationship between actors and movies

• A DB **relationship** is a correspondence between rows of one table and the rows of another table
Relationships in Practice

- **Key** `Student_ID` is used in each table
- We can find the address for each student (`Lives_At`), and also the student for each address (`Home_Of`)

**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

Build with **Join**, Match on the common field **Student_ID**

Master_List = Student   Home_Base

**On** Student.Student_ID = Home_Base.Student_ID

![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.](image)

Fig 16.18
Practical Construction Using QBE

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

Figure 16.20 SQL query created from the Query By Example data in Figure 16.19.
Query was generated by this dialog:

Figure 16.19 The Query By Example definition of the Master_List table from MS Access.
The Dean's View

• Storing the Dean's Data
  – *Top_Scholar* is information of interest only to the dean

<table>
<thead>
<tr>
<th>Top_Scholar:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_ID</td>
<td>number</td>
<td><em>Eight digits</em></td>
</tr>
<tr>
<td>Nickname</td>
<td>text</td>
<td><em>Informal handle for student</em></td>
</tr>
<tr>
<td>Factoid</td>
<td>text</td>
<td><em>Data to remember student by</em></td>
</tr>
<tr>
<td>Summer_Plans</td>
<td>text</td>
<td><em>Or other conversation topic</em></td>
</tr>
</tbody>
</table>

Primary Key: Student_ID

**(a)**

**Figure 16.21** The *Top_Scholar* definition: (a) informal form, (b) in MS Access.
Creating a Dean's View

<table>
<thead>
<tr>
<th>Name</th>
<th>Source Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nickname</td>
<td>Used by the dean to seem &quot;chummy&quot;</td>
</tr>
<tr>
<td>First_Name</td>
<td>Name information required because the dean forgets the person's actual name, being so chummy</td>
</tr>
<tr>
<td>Last_Name</td>
<td>Name information required because the dean forgets the person's actual name, being so chummy</td>
</tr>
<tr>
<td>Birthdate</td>
<td>Is student of &quot;drinking age&quot;?</td>
</tr>
<tr>
<td>City</td>
<td>Hometown (given by city, state) is important for small talk, but full address not needed by dean</td>
</tr>
<tr>
<td>State</td>
<td>Home_Base</td>
</tr>
<tr>
<td>Major</td>
<td>Student</td>
</tr>
<tr>
<td>Grade_Point</td>
<td>How's student doing grade-wise?</td>
</tr>
<tr>
<td>Factoid</td>
<td>Data to remember student by</td>
</tr>
<tr>
<td>Summer_Plans</td>
<td>Or other conversation topic</td>
</tr>
</tbody>
</table>

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 in all three tables

Dean_Data_Collect = ( Top_Scholar \Join
( Student \Join Home_base
  On Student.Student_ID=Home_Base.Student_ID )
  On Student.Student_ID=Top_Scholar.Student_ID )
Now that we have the Join table, we trim it to contain only data we really want.

Deans_View =

Project Nickname, First_Name, Last_Name, Birthdate, City, State, Major, Grade_Point, Factoid, Summer_Plan

From Dean_Data_Collect

Illustration of the *join-and-trim* technique for creating logical views.
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.
Summary

- XML tags are used to record metadata in a file.
- Metadata is used to identify values, can capture the affinity among values of the same entity, and can collect together a group of entity instances.
- Database tables have names and fields that describe the attributes of the entity contained in the table.
- The data that quantitatively records each property has a specific data type and is atomic.
Summary

• There are 5 fundamental operations on tables: select, project, union, difference, product; these operations are the only ones you need to create new tables from other database tables.

• Join is an especially useful operation that associates information from separate tables in new ways, based on matching fields.

• Relationships are the key to associating fields of the physical database.
Summary

• The physical database resides on the disk drive; it avoids storing data redundantly and is optimized for speed.

• The main approach for creating logical views from physical data is the *join-and-trim* technique.

• There is a direct connection between the theoretical ideas of database tables and the software of database systems.