Good Database Design Principles

1. **no redundancy**
   - A field is stored in *only one table*, unless it happens to be a foreign key.
   - Replication of foreign keys is permissible, because they allow two tables to be joined together.

2. **no “bad” dependencies**
   - In the dependency diagram of any relation in the database, the determinant should be the whole primary key, or a candidate key. Violations of this rule include:
     - partial dependencies
     - transitive dependencies

**Normalization** is the process of eliminating “bad” dependencies by splitting up tables and linking them with foreign keys.

- There are six recognized *normal forms* (**NF**):
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
First Normal Form

- A table is said to be in the first normal form (1NF) if all its attributes are atomic. Attributes that are not atomic go by the names
  - Nested relation(s)
  - Repeating groups
  - Repeating sections

- Example of a table that is not in first normal form:

<table>
<thead>
<tr>
<th>Client ID</th>
<th>Client Name</th>
<th>VetID</th>
<th>VetName</th>
<th>PetID</th>
<th>PetName</th>
<th>PetType</th>
</tr>
</thead>
<tbody>
<tr>
<td>2173</td>
<td>Barbara Hennessey</td>
<td>27</td>
<td>PetVet</td>
<td>1</td>
<td>Sam</td>
<td>Bird</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Hoober</td>
<td>Dog</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>Tom</td>
<td>Hamster</td>
</tr>
<tr>
<td>4519</td>
<td>Vernon Noordsy</td>
<td>31</td>
<td>PetCare</td>
<td>2</td>
<td>Charlie</td>
<td>Cat</td>
</tr>
<tr>
<td>8005</td>
<td>Sandra Amidon</td>
<td>27</td>
<td>PetVet</td>
<td>1</td>
<td>Beefer</td>
<td>Dog</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Kirby</td>
<td>Cat</td>
</tr>
<tr>
<td>8112</td>
<td>Helen Wandzell</td>
<td>24</td>
<td>PetsRUs</td>
<td>3</td>
<td>Kirby</td>
<td>Dog</td>
</tr>
</tbody>
</table>

CLIENT(ClientID, ClientName, VetID, VetName, PET(PetID, PetName, PetType) )

- This kind of nested or hierarchical form is a very natural way for people to think about or view data.
- However, the relational database philosophy claims that it is may not be a very good way for computers to store data.
- Over the years, a lot of information systems have stored data in this kind of format – but they were not relational databases.
In order to eliminate the nested relation, pull out the nested relation and form a new table.

Be sure to include the old key in the new table so that you can connect the tables back together.

CLIENT(ClientID, ClientName, VetID, VetName)

PET(ClientID, PetID, PetName, PetType)

CLIENTID foreign key to CLIENT
Second Normal Form

- Recall: a *partial dependency* occurs when
  - You have a composite primary key
  - A non-key attribute depends on part of the primary key, but not all of it
- A table in 1NF is said to be in the *second normal form* (2NF) if it does not contain any partial dependencies.
- Example of a partial dependency: ACTIVITY(StudentID, Activity, Fee) on pages 6, 7, and 9

- Our new CLIENT-PET database does not have any partial dependencies
- So, it already in second normal form
- But it still has a *transitive dependency*:

![Diagram]

1. StudentID
2. Fee
3. Activity
4. Client Name
5. ClientID
6. VetID
7. Vet Name
Third Normal Form

• Recall: a *transitive dependency* happens when a non-key attribute depends on another non-key attribute, and that attribute could not have been used as an alternative primary key (or same thing for a composition of several attributes).

• A table of 2NF is said to be in the **third normal form (3NF)** if it does not contain any transitive dependencies,

• In order to eliminate transitive dependency, we split the CLIENTS table again:

  CLIENTS(ClientID, ClientName, VetID)
  VetID foreign key to VET

  PETS(ClientID, PetID, PetName, PetType)
  ClientID foreign key to CLIENT

  VETS(VetID, VetName)
Third Normal Form (Cont.)

- CLIENTS-PETS-VETS database in the third normal form:

<table>
<thead>
<tr>
<th>Client ID</th>
<th>Client Name</th>
<th>VetID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2173</td>
<td>Barbara Hennessey</td>
<td>27</td>
</tr>
<tr>
<td>4519</td>
<td>Vernon Noordsy</td>
<td>31</td>
</tr>
<tr>
<td>8005</td>
<td>Sandra Amidon</td>
<td>27</td>
</tr>
<tr>
<td>8112</td>
<td>Helen Wandzell</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Client ID</th>
<th>PetID</th>
<th>PetName</th>
<th>PetType</th>
</tr>
</thead>
<tbody>
<tr>
<td>2173</td>
<td>1</td>
<td>Sam</td>
<td>Bird</td>
</tr>
<tr>
<td>2173</td>
<td>2</td>
<td>Hoober</td>
<td>Dog</td>
</tr>
<tr>
<td>2173</td>
<td>3</td>
<td>Tom</td>
<td>Hamster</td>
</tr>
<tr>
<td>4519</td>
<td>2</td>
<td>Charlie</td>
<td>Cat</td>
</tr>
<tr>
<td>8005</td>
<td>1</td>
<td>Beefer</td>
<td>Dog</td>
</tr>
<tr>
<td>8005</td>
<td>2</td>
<td>Kirby</td>
<td>Cat</td>
</tr>
<tr>
<td>8112</td>
<td>3</td>
<td>Kirby</td>
<td>Dog</td>
</tr>
</tbody>
</table>

VetID | VetName  
---|---------  
27 | PetVet  
31 | PetCare  
24 | PetsRUs  

- the database consists of three types of entities, stored as distinct relations in separate tables:
  - clients (CLIENTS)
  - pets (PETS)
  - vets (VETS)

- there is no redundancy (only foreign keys are replicated)
- there are no partial and transitive dependencies
Normal Forms and Normalization

• The distinctions between third normal form (3NF), Boyce-Codd normal form (BCNF), fourth normal form (4NF), and fifth normal form (5NF) are subtle.

• They have to do with overlapping sets of attributes that could be used as primary keys (composite candidate keys).

• For our purposes, it’s enough to know about 3NF.
  • You need to be able to put a database in 3NF.
  • That is more important than recognizing 1NF and 2NF.

• Key factors to recognize 3NF:
  • All attributes atomic — gives you 1NF.
  • Every determinant in every relationship is the whole primary key (or could have been chosen as an alternative primary key) – guarantees no partial or transitive dependencies.

• Redesigning a database so it’s in 3NF is called normalization.
Normalization Example: Hardware Store Database

• the ORDERS table:

<table>
<thead>
<tr>
<th>Order Numb</th>
<th>Cust Code</th>
<th>Order Date</th>
<th>Cust Name</th>
<th>ProdDescr</th>
<th>Prod Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>5217</td>
<td>11/22/94</td>
<td>Williams</td>
<td>Hammer</td>
<td>$8.99</td>
<td>2</td>
</tr>
<tr>
<td>10001</td>
<td>5217</td>
<td>11/22/94</td>
<td>Williams</td>
<td>Screwdriver</td>
<td>$4.45</td>
<td>1</td>
</tr>
<tr>
<td>10002</td>
<td>5021</td>
<td>11/22/94</td>
<td>Johnson</td>
<td>Clipper</td>
<td>$18.22</td>
<td>1</td>
</tr>
<tr>
<td>10002</td>
<td>5021</td>
<td>11/22/94</td>
<td>Johnson</td>
<td>Screwdriver</td>
<td>$4.45</td>
<td>3</td>
</tr>
<tr>
<td>10002</td>
<td>5021</td>
<td>11/22/94</td>
<td>Johnson</td>
<td>Crowbar</td>
<td>$11.07</td>
<td>1</td>
</tr>
<tr>
<td>10002</td>
<td>5021</td>
<td>11/22/94</td>
<td>Johnson</td>
<td>Saw</td>
<td>$14.99</td>
<td>1</td>
</tr>
<tr>
<td>10003</td>
<td>4118</td>
<td>11/22/94</td>
<td>Lorenzo</td>
<td>Hammer</td>
<td>$8.99</td>
<td>1</td>
</tr>
<tr>
<td>10004</td>
<td>6002</td>
<td>11/22/94</td>
<td>Kopiusko</td>
<td>Saw</td>
<td>$14.99</td>
<td>1</td>
</tr>
<tr>
<td>10004</td>
<td>6002</td>
<td>11/22/94</td>
<td>Kopiusko</td>
<td>Screwdriver</td>
<td>$4.45</td>
<td>2</td>
</tr>
<tr>
<td>10005</td>
<td>5021</td>
<td>11/23/94</td>
<td>Johnson</td>
<td>Cordlessdrill</td>
<td>$34.95</td>
<td>1</td>
</tr>
</tbody>
</table>

• Note: in practice, we would also want to have product codes as well as descriptions, and use the product codes as keys to identify products. Here, we’ll identify products by their ProdDescr to keep the number of fields down.
Example: Hardware Store Database (Cont.)

ORDERS(OrderNum, ProdDescr, CustCode, OrderDate, CustName, ProdPrice, Quantity)

• Conversion of the hardware store database to 2NF
  QUANTITY(OrderNum, ProdDescr, Quantity)
  OrderNum foreign key to ORDERS
  ProdDescr foreign key to PRODUCTS
  PRODUCTS(ProdDescr, ProdPrice)
  ORDERS(OrderNum, CustCode, OrderDate, CustName)
**Example: Hardware Store Database (Cont.)**

- conversion of the ORDERS relation to 3NF

  QUANTITY(OrderNum, ProdDescr, Quantity)
  OrderNum foreign key to ORDERS
  ProdDescr foreign key to PRODUCTS

  PRODUCTS(ProdDescr, ProdPrice)

  ORDERS(OrderNum, CustCode, OrderDate)
  CustCode foreign key to CUSTOMERS

  CUSTOMERS(CustCode, CustName)
RELATIONAL DATABASE DESIGN

Example: Video Store Database

• the CUSTOMER relation:

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Phone</th>
<th>Last Name</th>
<th>First Name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>502-666-7777</td>
<td>Johnson</td>
<td>Martha</td>
<td>125 Main St.</td>
<td>Alvaton</td>
<td>KY</td>
<td>42122</td>
</tr>
<tr>
<td>2</td>
<td>502-888-6464</td>
<td>Smith</td>
<td>Jack</td>
<td>873 Elm St.</td>
<td>Bowling Green</td>
<td>KY</td>
<td>42101</td>
</tr>
<tr>
<td>3</td>
<td>502-777-7575</td>
<td>Washington</td>
<td>Elroy</td>
<td>95 Easy St.</td>
<td>Smith's Grove</td>
<td>KY</td>
<td>42171</td>
</tr>
<tr>
<td>4</td>
<td>502-333-9494</td>
<td>Adams</td>
<td>Samuel</td>
<td>746 Brown Dr.</td>
<td>Alvation</td>
<td>KY</td>
<td>42122</td>
</tr>
<tr>
<td>5</td>
<td>502-474-4746</td>
<td>Steinmetz</td>
<td>Susan</td>
<td>15 Speedway Dr.</td>
<td>Portland</td>
<td>TN</td>
<td>37148</td>
</tr>
</tbody>
</table>

• the RENTAL-FORM relation:

<table>
<thead>
<tr>
<th>Trans ID</th>
<th>Rent Date</th>
<th>Customer ID</th>
<th>Video ID</th>
<th>Copy#</th>
<th>Title</th>
<th>Rent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4/18/95</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2001:SpaceOdyssey</td>
<td>$1.50</td>
</tr>
<tr>
<td>1</td>
<td>4/18/95</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>Clockway Orange</td>
<td>$1.50</td>
</tr>
<tr>
<td>2</td>
<td>4/18/95</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>Hpscotch</td>
<td>$1.50</td>
</tr>
<tr>
<td>2</td>
<td>4/18/95</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>Apocalypse Now</td>
<td>$2.00</td>
</tr>
<tr>
<td>2</td>
<td>4/18/95</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>Clockwork Orange</td>
<td>$1.50</td>
</tr>
<tr>
<td>3</td>
<td>4/18/95</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>Luggage of the Gods</td>
<td>$2.50</td>
</tr>
</tbody>
</table>

• a customer can rent multiple videos as part of the same transaction
• multiple copies of the same video exist
  • the copy# field stores the number of the copy
• one customer cannot rent two copies of the same video at the same time
• the database still contains some anomalies
Example: Video Store Database (Cont.)

- relations for the video store database
  - CUSTOMER(CustomerID, Phone, Name, Address, City, State, ZipCode)
  - RENTAL-FORM(TransID, RentDate, CustomerID, VideoID, Copy#, Title, Rent)

- dependency diagram for the video store database
Example: Video Store Database (Cont.)

- Video store database after eliminating partial and transitive dependencies

CUSTOMER(CustomerID, Phone, Name, Address, City, State, ZipCode)

RENTAL(TransID, RentDate, CustomerID)
  CustomerID foreign key to CUSTOMER

VIDEO(VideoID, Title, Rent)

VIDEOSRENTED(TransID, VideoID, Copy#)
  TransID foreign key to RENTAL
  VideoID foreign key to VIDEO
Example: Video Store Database (Cont.)

- table relationships for the video store database
Summary of Guidelines for Database Design

• identify the entities involved in the database
• identify the fields relevant for each entity and define the corresponding relations
• determine the primary key of each relation
• avoid data redundancy, but have some common fields so that tables can be joined together
• ensure that all the required database processing can be done using the defined relations
• normalize the relations by splitting them into smaller ones