RELATIONAL DATABASE DESIGN

Functional Dependency

• attribute B is functionally dependent on attribute A if given a value of attribute A, there is only one possible corresponding value of attribute B

- that is, any two rows with the same value of A must have the same value for B

• attribute A is the determinant of attribute B if attribute B is functionally dependent on attribute A

- in the STATE relation, StateAbbrev is a determinant of all other attributes
- in the STATE relation, the attribute StateName is also a determinant of all other attributes
- so, StateAbbrev and StateName are both candidate keys for STATE
- in the CITY relation above, the attributes (StateAbbrev, CityName) together are a determinant of the attribute CityPopulation
- in the CITY relation, the attribute CityName is not a determinant of the attribute CityPopulation because multiple cities in the table may have the same name

STATE table:

State Abbrev	StateName	Union Order	StateBird	State Population
CT	Connecticut	5	American robin	3,287,116
MI	Michigan	26	robin	9,295,297
SD	South Dakota	40	pheasant	696,004
TN	Tennessee	16	mocking bird	4,877,185
TX	Texas	28	mocking bird	16,986,510

CITY table:

State Abbrev	CityName	City Population
CT	Hartford	139,739
CT	Madison	14,031
CT	Portland	8,418
MI	Lansing	127,321
SD	Madison	6,257
SD	Pierre	12,906
TN	Nashville	488,374
TX	Austin	465,622
TX	Portland	12,224

• primary key in STATE relation: StateAbbrev

• primary key in CITY relation: (StateAbbrev, CityName)

• foreign key in CITY relation: StateAbbrev

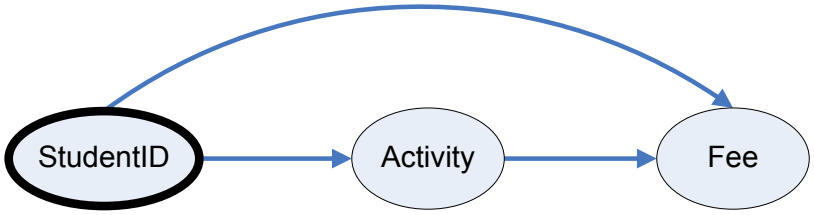
Dependency Diagrams

• a dependency diagram or bubble diagram is a pictorial representation of functional dependencies

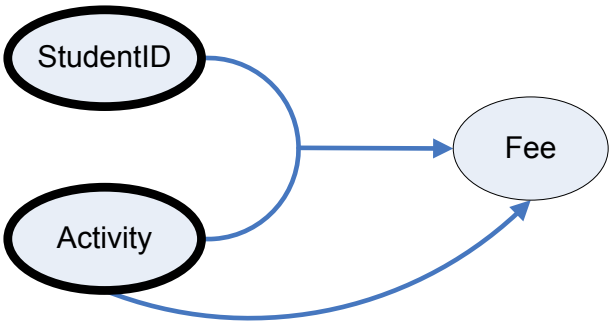
- an attribute is represented by an oval
- you draw an arrow from A to B when attribute A is a determinant of attribute B

StudentID	Activity	Fee
100	Skiing	200
150	Swimming	50
175	Squash	50
200	Swimming	50

• example: in the table above, if students were only allowed one sports activity, we have ACTIVITY(StudentID, Activity, Fee)



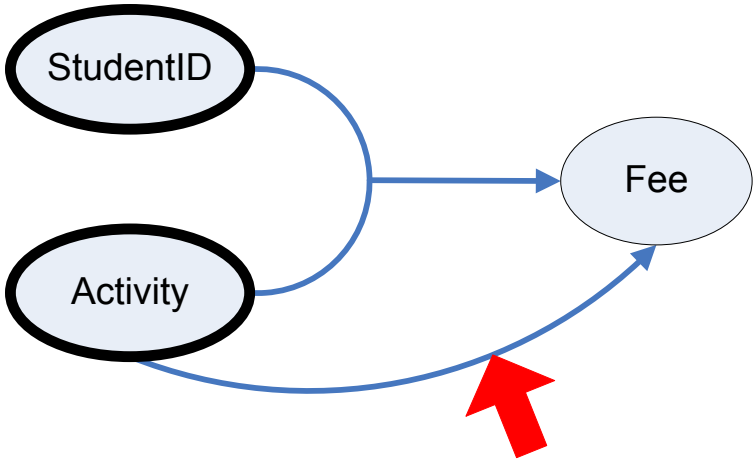
• example: when students can have multiple activities, we have ACTIVITY(StudentID, Activity, Fee)



Partial Dependencies

• a partial dependency is a functional dependency whose determinant is part of the primary key (but not all of it)

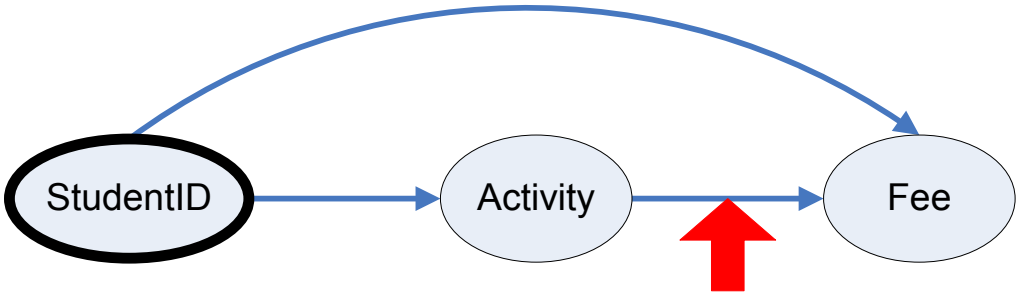
• example: ACTIVITY(StudentID, Activity, Fee)



Transitive Dependencies

• a transitive dependency is a functional dependency whose determinant is not the primary key, part of the primary key, or a candidate key

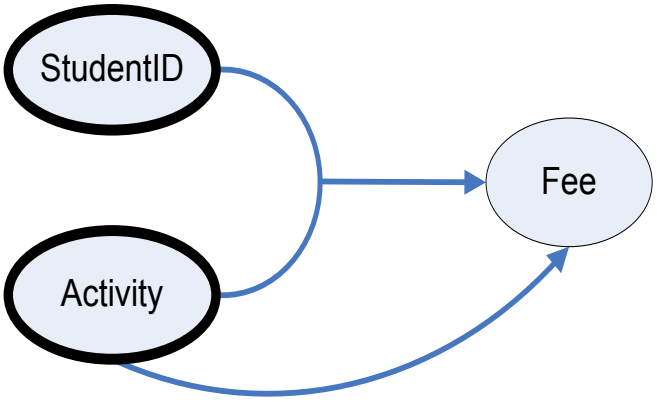
• example: ACTIVITY(StudentID, Activity, Fee)



Cause of Anomalies

- anomalies are primarily caused by:
  - *data redundancy*: replication of the same field in multiple tables, other than foreign keys
  - Functional dependencies whose determinants are not candidate keys, including
    - *partial dependency*
    - *transitive dependency*

example: ACTIVITY(StudentID, Activity, Fee)



StudentID	Activity	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

- Activity by itself is not a candidate key, so we get anomalies (in this case, from a partial dependency):
  - insertion anomaly example: we want to store that scuba diving costs $175, but have no place to put this information until a student signs up for scuba diving (unless we create a fake student)
  - a deletion anomaly example: if the student with StudentID = 100 takes the semester off and we delete his/her records, we forget that skiing costs $200
  - an update anomaly example: if the cost of swimming changes, then all entries with the swimming activity must be changed as well

Fixing Anomalies (Normalizing)

- Break up tables so all dependencies are from primary (or candidate) keys

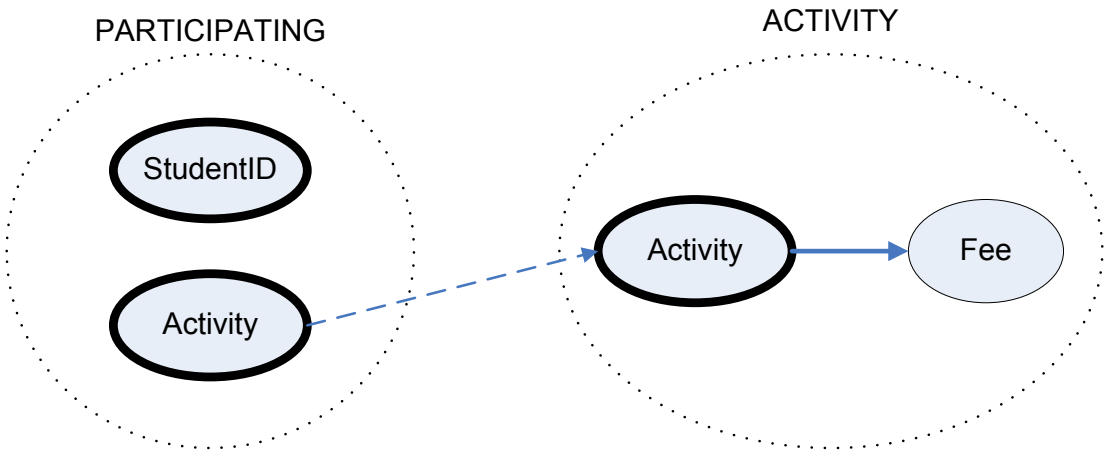
PARTICIPATING(StudentID, Activity)

Activity foreign key to ACTIVITIES

ACTIVITY(Activity, Fee)

StudentID	Activity
100	Skiing
100	Golf
150	Swimming
175	Squash
175	Swimming
200	Swimming
200	Golf

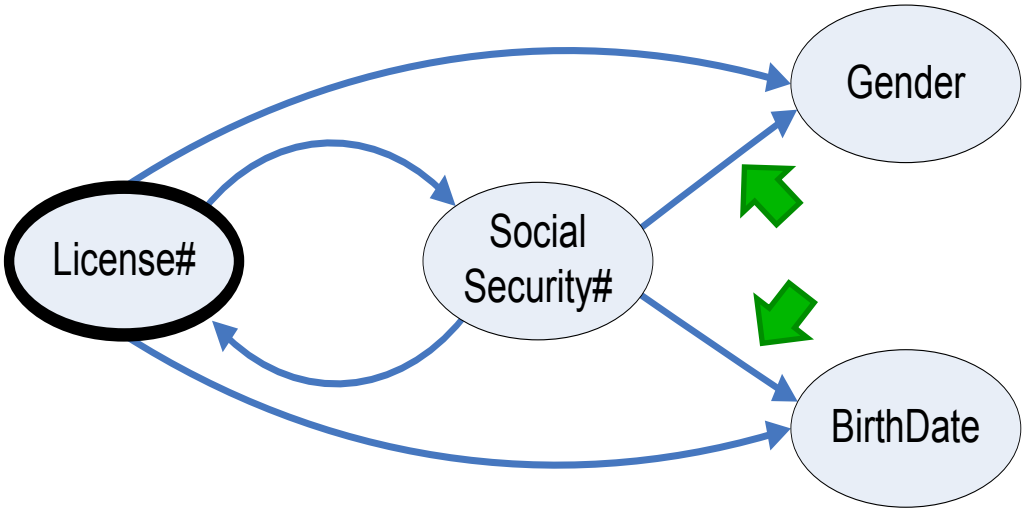
Activity	Fees
Skiing	200
Golf	65
Swimming	50
Squash	50
ScubaDiving	200



- the above relations do not have any of the anomalies
  - we can add the cost of diving in ACTIVITIES even though no one has taken it in STUDENTS
  - if StudentID 100 drops Skiing, no skiing-related data will be lost
  - if the cost of swimming changes, that cost need only be changed in one place only (the ACTIVITIES table)
- the Activity field is in both tables, but that's needed to relate ("join") the information in the two tables

Example With Multiple Candidate Keys

DRIVER(License#, SocialSecurity#, Gender, BirthDate)



- The dependencies **SocialSecurity# → Gender** and **SocialSecurity# → BirthDate** are *not* considered transitive because SocialSecurity# is a candidate key, that is, it could be chosen as the primary key for the table.

- **This kind of design will not give rise to anomalies.**

RELATIONAL DATABASE DESIGN

Good Database Design Principles

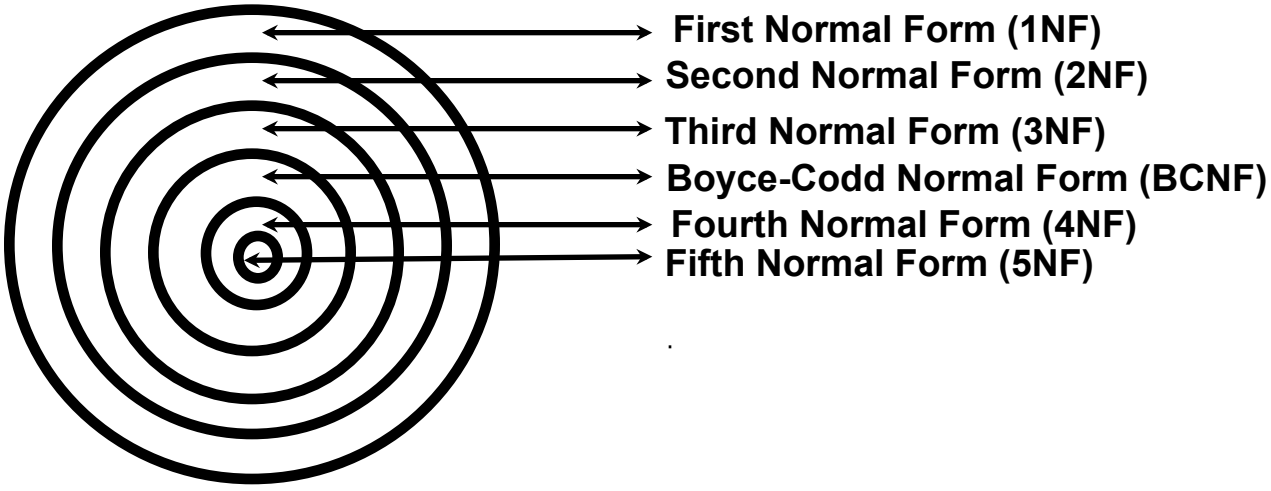
Recall that two of the DB Design Principles were:

- **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)
- **no "bad" dependencies: : in a table, fields should depend only and fully on the primary key (or other candidate keys) and not on any other fields.**
    - in the dependency diagram of any relation in the database, this means that the determinant should be the whole primary key, or a candidate key. Violations of this rule include:
        - partial dependencies
        - transitive dependencies

Normalization

normalization is the process of eliminating "bad" dependencies by splitting up tables and linking them with foreign keys

- "normal forms" are categories that classify how completely a table has been normalized
- there are six recognized *normal forms* (*NF*):



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 relations, nested tables, or sub-tables**
- **Repeating groups or repeating sections**
- **List-valued attributes**

• example of a table that is *not* in first normal form:

Client ID	Client Name	VetID	VetName	PetNo	PetName	PetType
2173	Barbara Hennessey	27	PetVet	1 2 3	Sam Hoober Tom	Bird Dog Hamster
4519	Vernon Noordsy	31	PetCare	2	Charlie	Cat
8005	Sandra Amidon	27	PetVet	1 2	Beefer Kirby	Dog Cat
8112	Helen Wandzell	24	PetsRUs	3	Kirby	Dog

CLIENT(ClientD, ClientName, VetID, VetName, PET(PetNo, PetName, PetType))

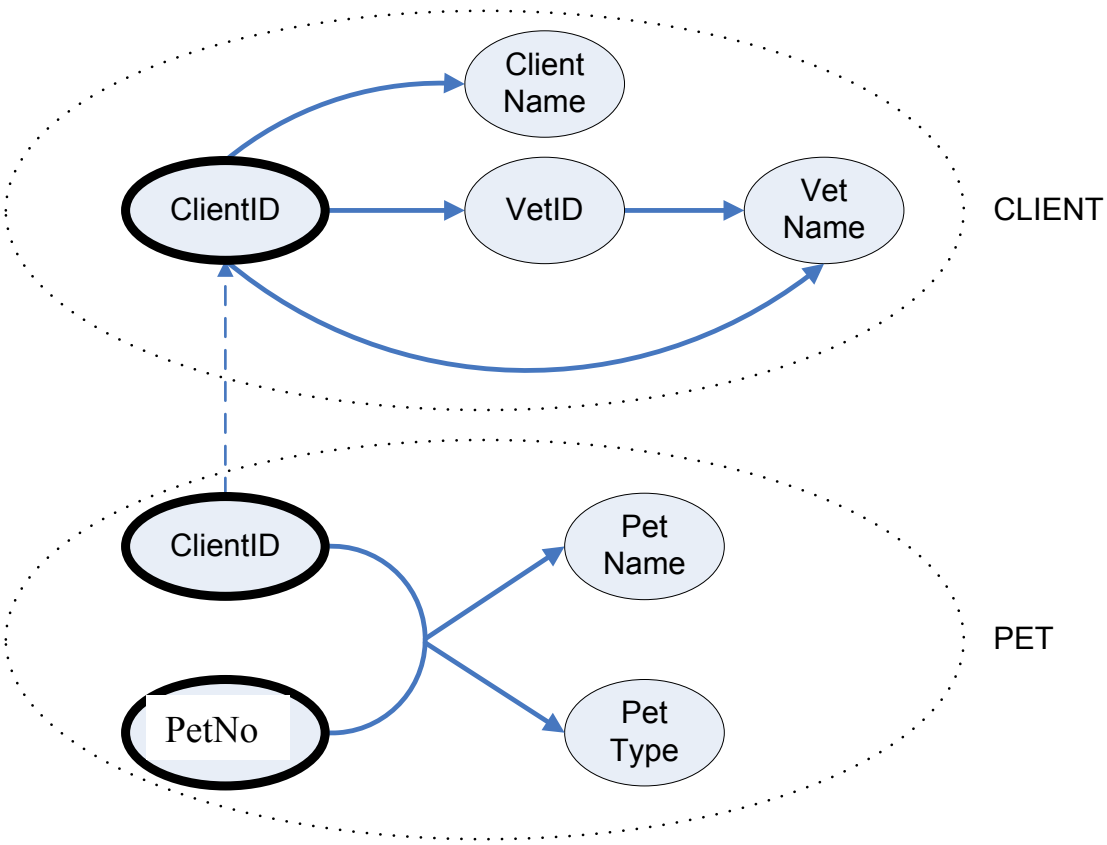
• This kind of nested or hierarchical form is a very natural way for people to think about or view data.

• However, the relational database philosophy claims that it may not be a very good way for computers to *store* some kinds of 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(ClientD, ClientName, VetID, VetName)

PET(ClientID, PetNo, PetName, PetType)
ClientID foreign key to CLIENT

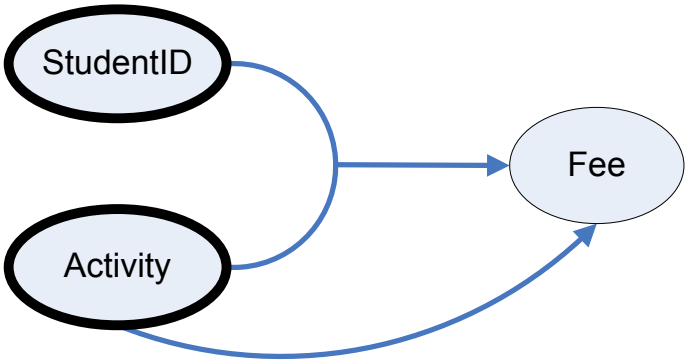
• In this particular example, note that PetID is only unique within sets of pets with the same owner.

Second Normal Form

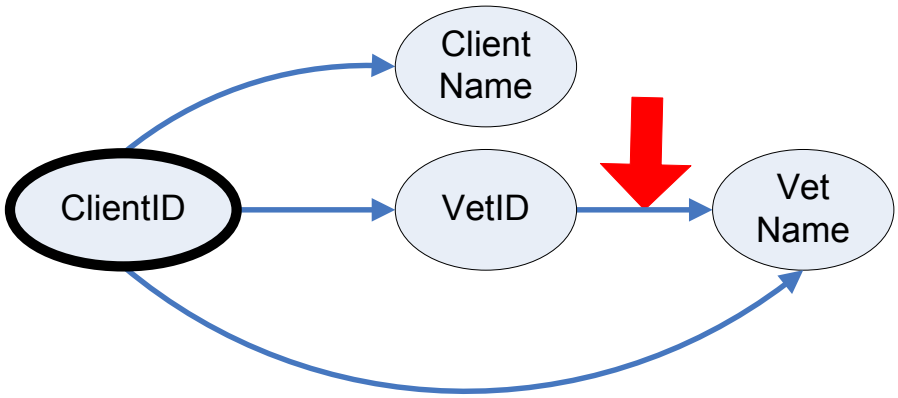
- Recall: a *partial dependency* occurs when
  - You have a composite primary key
  - A non-key attribute depends on part of the primary key, but not all of it

- **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 another functional dependency (*transitive)*:



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 the same thing for a composition of several attributes).

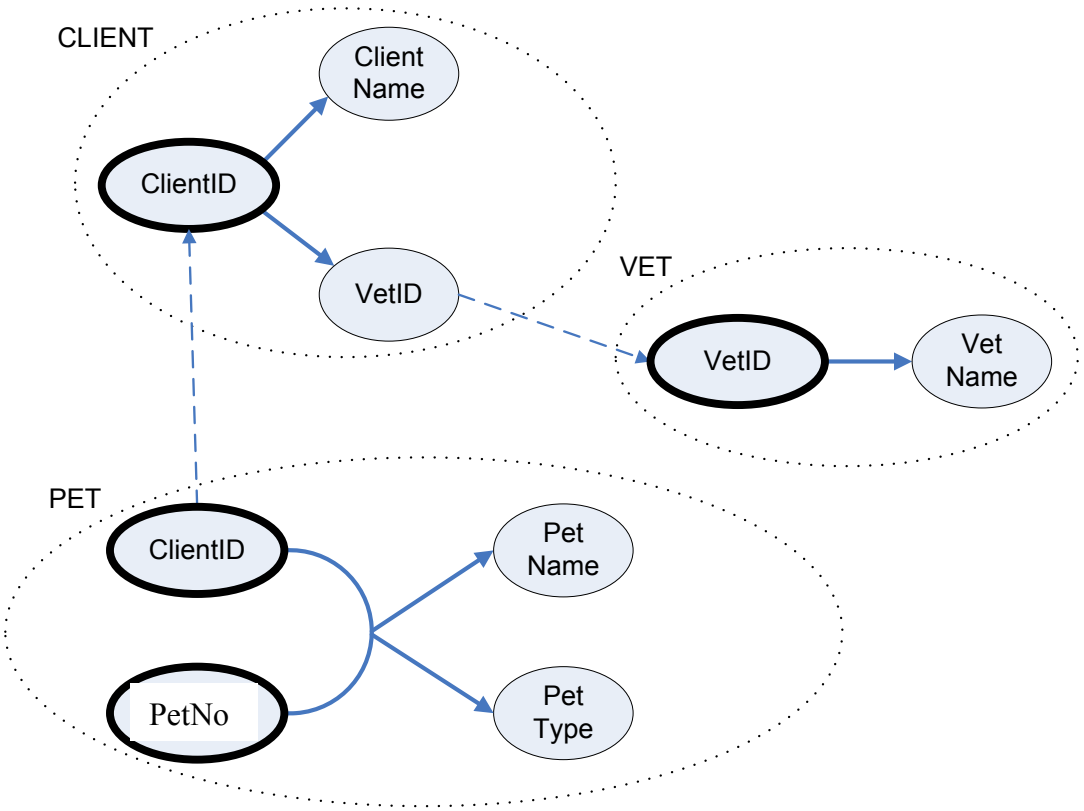
• **A table of 2NF is said to be in the third normal form (3NF) if it does not contain any transitive dependencies,**

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

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

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

VETS(VetID, VetName)



Third Normal Form (Cont.)

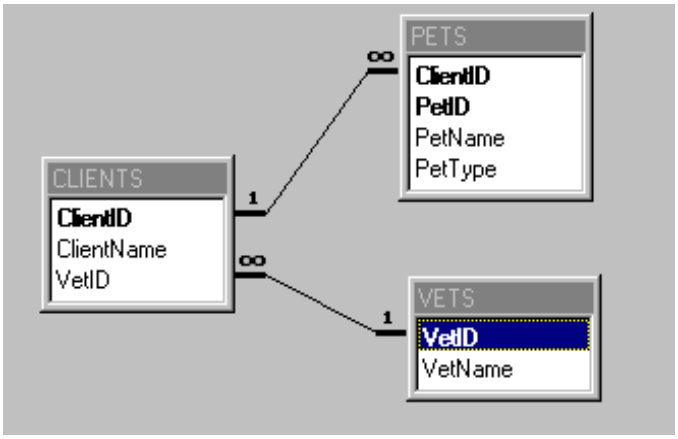
• CLIENTS-PETS-VETS database in third normal form:

Client ID	Client Name	VetID
2173	Barbara Hennessey	27
4519	Vernon Noordsy	31
8005	Sandra Amidon	27
8112	Helen Wandzell	24

Client ID	PetID	PetName	PetType
2173	1	Sam	Bird
2173	2	Hoober	Dog
2173	3	Tom	Hamster
4519	2	Charlie	Cat
8005	1	Beefer	Dog
8005	2	Kirby	Cat
8112	3	Kirby	Dog

VetID	VetName
27	PetVet
31	PetCare
24	PetsRUs

with MS Access table relationships



• 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

- Database theorists define some additional normal forms: Boyce-Codd normal form (BCNF, a sort of 3.5th normal form), fourth normal form (4NF), and fifth normal form (5NF).
- 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.
    - You should be able to recognize when a database is in 3NF
    - It is also good to be able to recognize 1NF and 2NF, but not as important
- Key factors to recognize 3NF:
    - All attributes atomic – gives you 1NF.
    - Every determinant in every attribute relationship within each table is the whole primary key, or could have been chosen as an alternative primary key – guarantees no partial or transitive dependencies.