RELATIONAL DATABASE DESIGN

Basic Concepts

- a **database** is an collection of logically related *records*
- a **relational database** stores its data in 2-dimensional *tables*
- a **table** is a two-dimensional structure made up of *rows* (*tuples*, *records*) and *columns* (*attributes*, *fields*)
- example: a table of students engaged in sports activities, where a student is allowed to participate in at most one activity

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

Table Characteristics

- each row is unique and stores data about one entity
- row order is unimportant
- each column has a unique **attribute name**
- each column (attribute) description (*metadata*) is stored in the database
    - Access metadata is stored and manipulated via the Table Design View grid
- column order is unimportant
- all entries in a column have the same data type
    - Access examples: Text(50), Number(Integer), Date/Time
- each cell contains atomic data: no lists or sub-tables

Primary Keys

- a **primary key** is an attribute or a collection of attributes whose value(s) uniquely identify each row in a relation
- a primary key should be minimal: it should not contain unnecessary attributes

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

- we assume that a student is allowed to participate in at most one activity
- the only possible primary key in the above table is StudentID
- Sometimes there is more than one possible choice; each possible choice is called a candidate key
- what if we allow the students to participate in more than one activity?

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

- now the only possible primary key is the combined value of (StudentID, Activity),
- such a multi-attribute primary key is called a composite key or concatenated key

Composite Keys

- a table can only have one primary key
- but sometimes the primary key can be made up of several fields
- concatenation means putting two things next to one another: the concatenation of "burger" and "foo" is "burgerfoo".
- consider the following table of cars

LicensePlate	State	Make	Model	Year
LVR120	NJ	Honda	Accord	2003
BCX50P	NJ	Buick	Regal	1998
LVR120	CT	Toyota	Corolla	2002
908HYY	MA	Ford	Windstar	2001
UHP33X	NJ	Nissan	Altima	2006

- LicensePlate is not a possible primary key, because two different cars can have the same license plate number if they're from different states
- but if we concatenate LicensePlate and State, the resulting value of (LicensePlate, State) must be unique:
    - example: "LVR120NJ" and "LVR120CT"
- therefore, (LicensePlate, State) is a possible primary key (a candidate key)
- Sometimes we may invent a new attribute to serve as a primary key (sometimes called a synthetic key)
    - if no suitable primary key is available
    - or, to avoid composite keys
    - in Access, "Autonumber" fields can serve this purpose

Foreign Keys

- **a foreign key is an attribute or a collection of attributes whose value are intended to match the primary key of some related record (usually in a different table)**
- **example: the STATE and CITY table below**

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**

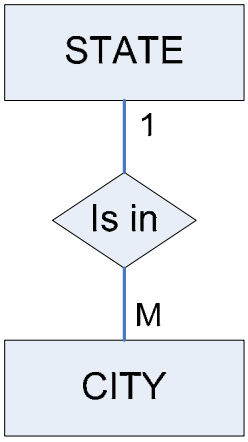
Outline Notation

STATE(StateAbbrev, StateName, UnionOrder, StateBird, StatePopulation)

CITY(StateAbbrev, CityName, CityPopulation)
StateAbbrev foreign key to STATE

- **Underline all parts of each primary key**
- **Note foreign keys with "*attribute* foreign key to *TABLE*"**

Entity-Relationship Diagrams



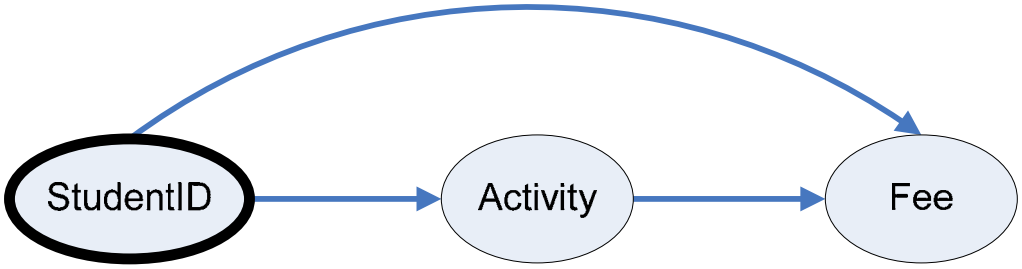
- **one-to-many relationships: to determine the direction, always start with "one"**
  - **"*one* city is in *one* state"**
  - **"*one* state contains *many* cities"**
- **the foreign key is always in "the many" – otherwise it could not be atomic (it would have to be a list)**
- **We will study other kinds of relationships (one-to-one and many-to-many) shortly**

Functional Dependency

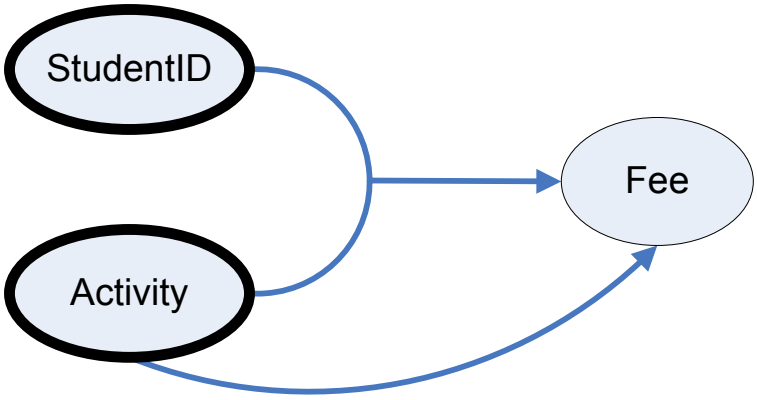
- attribute B is **functionally dependent** on attribute A if given a value of attribute A, there is only one possible corresponding value of attribute B
    - that is, any two rows with the same value of A must have the same value for B
- attribute A is the **determinant** of attribute B if attribute B is functionally dependent on attribute A
    - in the STATE relation above, **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

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
- example: when students were only allowed one sports activity, we have ACTIVITY(StudentID, Activity, Fee)

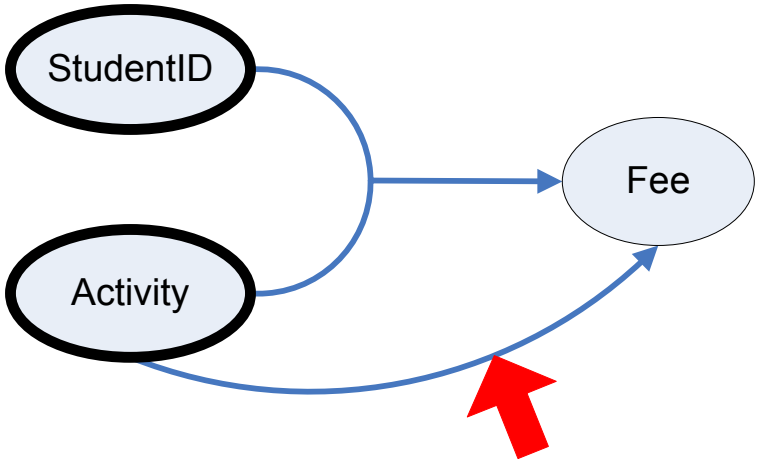


- example: when students can have multiple activities, we have ACTIVITY(StudentID, Activity, Fee)



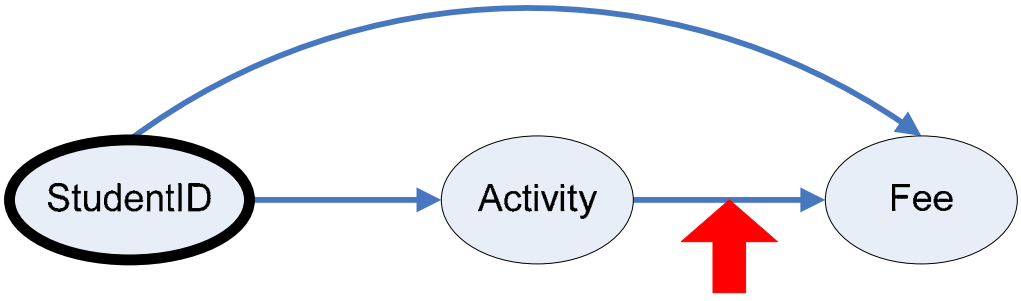
Partial Dependencies

- 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)



Database Anomalies

• **anomalies are problems caused by bad database design**

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

• **an insertion anomaly occurs when a row cannot be added to a relation, because not all data are available (or one has to invent "dummy" data)**

  • **example: we want to store that scuba diving costs $175, but have no place to put this information until a student takes up scuba-diving (unless we create a fake student)**

• **a deletion anomaly occurs when data is deleted from a relation, and other critical data are unintentionally lost**

  • **example: if we delete the record with StudentID = 100, we forget that skiing costs $200**

• **an update anomaly occurs when one must make many changes to reflect the modification of a single datum**

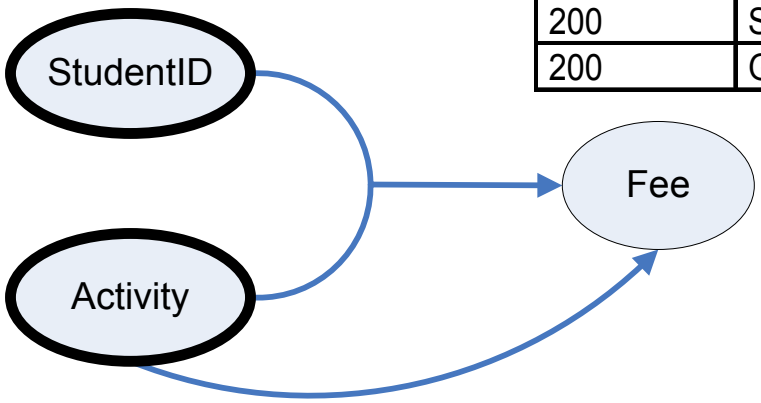
  • **example: if the cost of swimming changes, then all entries with swimming Activity must be changed too**

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)

Fixing Anomalies (Normalizing)

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

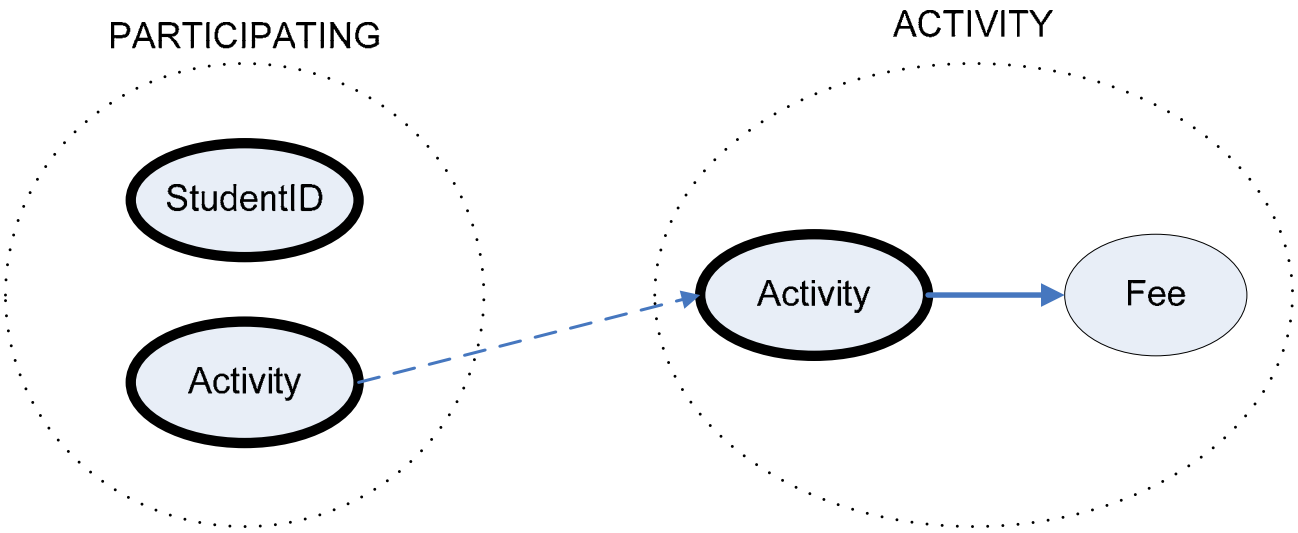
PARTICIPATING(StudentID, Activity)

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



RELATIONAL DATABASE DESIGN

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**

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

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