INFSCI 1022
Database Management Systems

Entities

- A thing that can exist independently and that can be identified uniquely.
- A class, group or category of similar objects.
- A real world object such as a car or an employee.
- Can be thought of as nouns that come up during the description of the problem to be solved.

http://www.differencebetween.com/difference-between-entity-and-vs-attribute

Entities

- Represented as tables in relational databases
- Each entity will map to exactly one table in the database.
- Individual rows in the tables correspond to the actual instances of the object/thing represented by the entity.
- For example, in an Employee database, each row corresponds to records of individual employees of the company.

http://www.differencebetween.com/difference-between-entity-and-vs-attribute/

Super Awesome Exercise # 1

What are the entities of a flight reservation system?

Attributes

- Properties of entities are called attributes.
- Attributes represent a subgroup of information of the object represented by the entity.
- Attributes define the individual instances and help to differentiate between each instance by describing their characteristic.

http://www.differencebetween.com/difference-between-entity-and-vs-attribute/

Attributes

- Each column in a database table represents the attributes of an entity.
- For example, in the Employee table, columns such as department, rank and salary are examples of attributes of the employees.

http://www.differencebetween.com/difference-between-entity-and-vs-attribute/

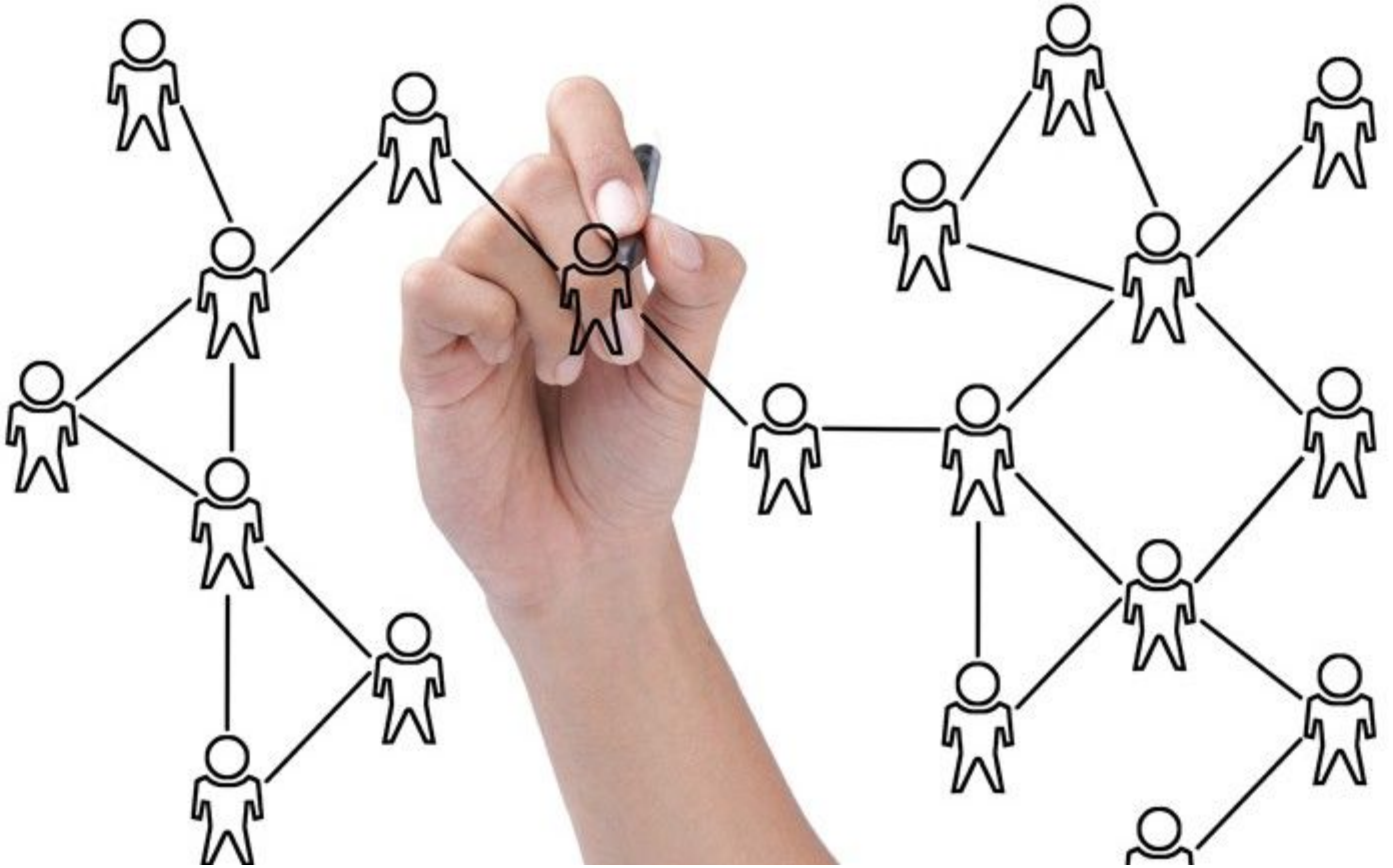
Super Awesome Exercise # 2

You are designing a flight tracking database for a small privately-owned airline. In your design, should **Airport** be an attribute of **Flight entity** or should Airport be its own entity? Why or why not?

Super Awesome Exercise # 3

In a banking application, should **Address** be an attribute of **Customer** entity or should Address be its own entity?

Relations

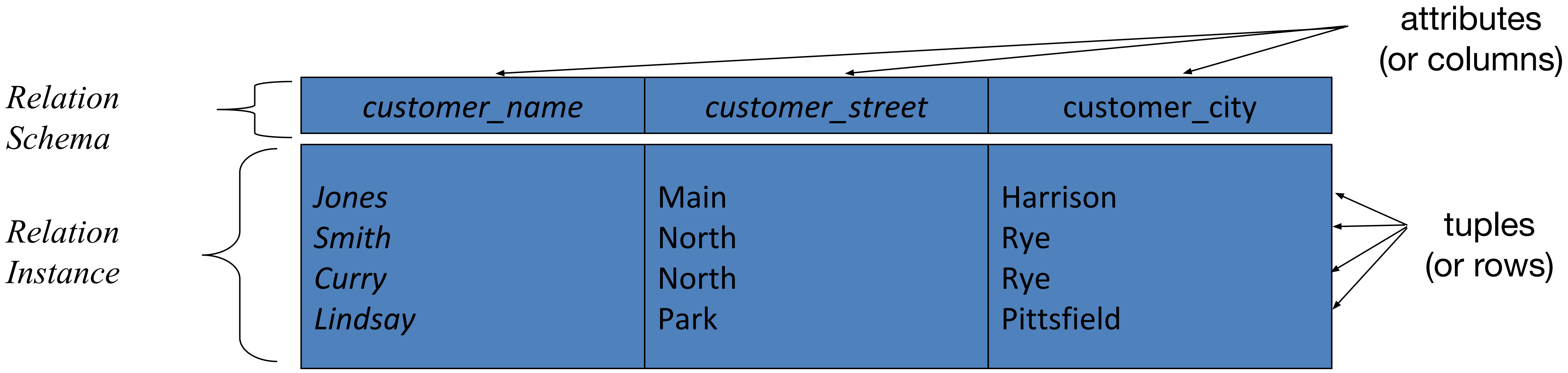


https://thornleyfallis.com/public-relations/

Relations

- Relational Database stores data in form of *relations*.
- Roughly speaking relation is a table.
- A data item is represented by a *row* in a table (a *tuple).*
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order).

Relation is a Table



Relation Schema

Relation Instance

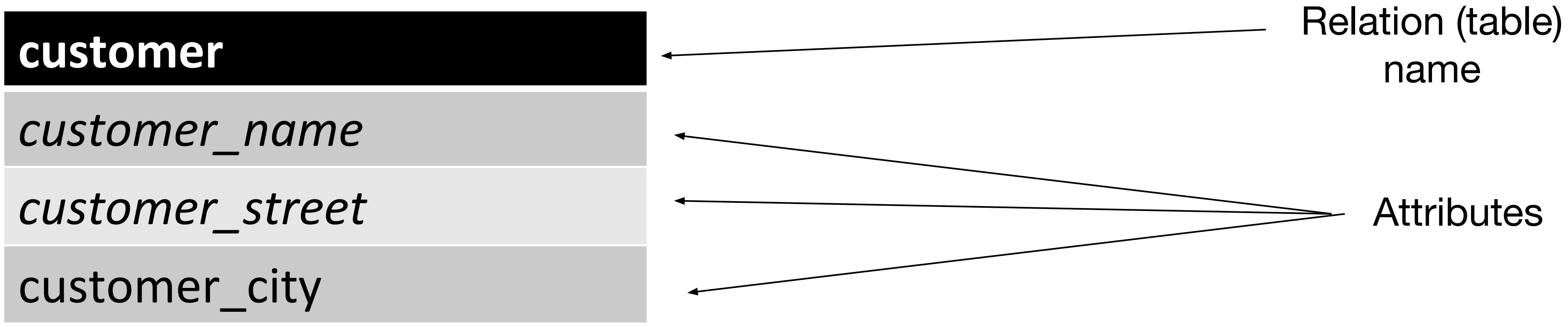
attributes (or columns)

tuples (or rows)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

Relation Name: customer

Relational Schema Shorthand



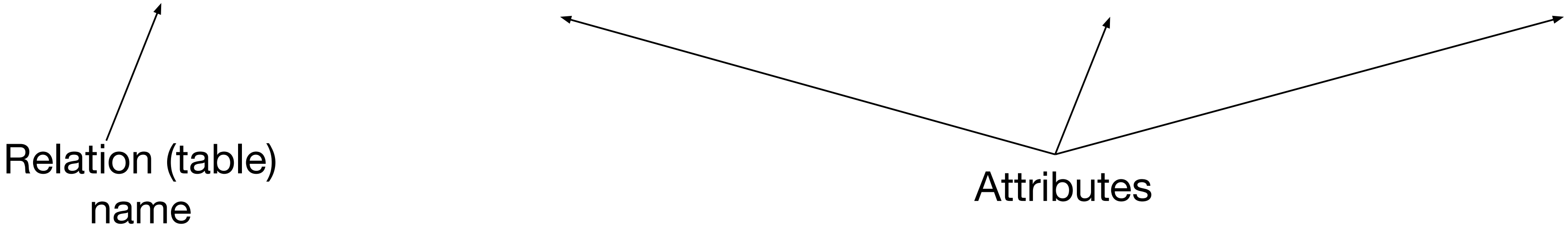
customer
customer_name
customer_street
customer_city

Relation (table) name

Attributes

sometime is written as:

customer(customer_name, customer_street, customer_city)

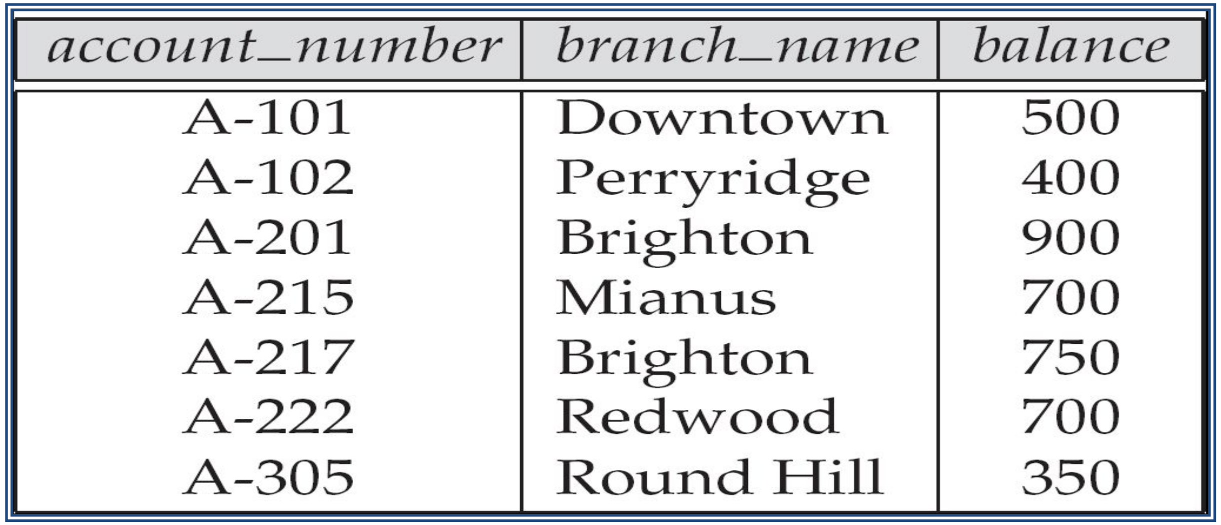


Relation (table) name

Attributes

Attribute Names and Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute



account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

- Attribute *balance* is of *integer* domain
- Attributes *account_number* and *branch_name* are *strings*

NULL

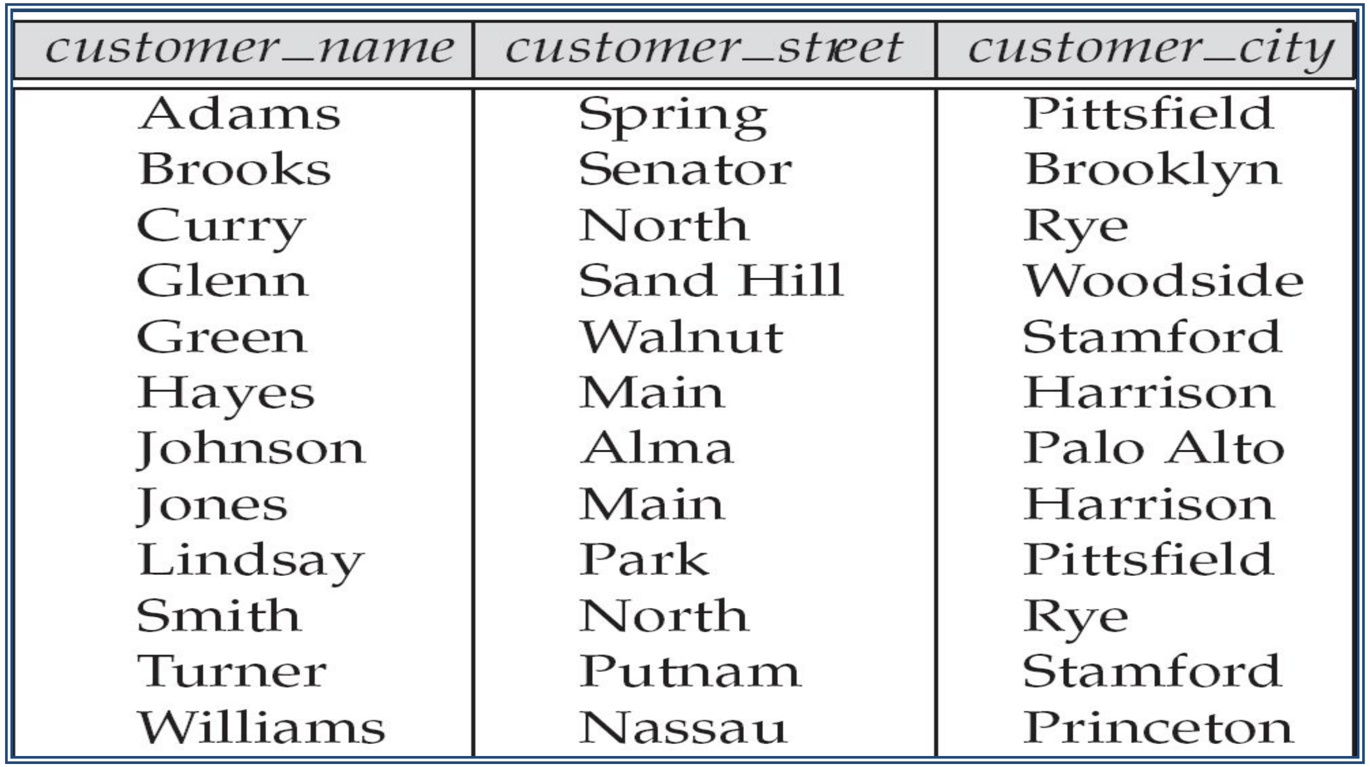
- The special value *null* is a member of every domain (more about it later)

Relation Schema

- *List of attributes is known as* a ***relation schema***

  Example: *Customer_schema* = (*customer_name, customer_street, customer_city*)

customer:



customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Relation vs. Relation Instance

- Relation is used to refer to a table
- Relation instance refers to a specific instance of a relation, i.e., containing a specific set of rows

Relation vs. Relation Instance

- Relation:

  vehicles (vehicleID, make, model, color)

- Relation instance:

vehicleID	make	model	color	year
1	Ford	Taurus	Beige	2005
2	Honda	Civic	Red	2012
3	Toyota	Corolla	White	2009

Naming Conventions - CamelCase

- Database schema name:
  - Start with a capital letter – ***Registration, Vehicles, University, Birds***
  - If name consists of multiple words, each word in the name starts with a capital letter – ***StudentProjects, ResumeSystem, PatientRecordsSystem***
  - Make sure to be consistent!

Naming Conventions - CamelCase

- Table names:
  - Start with a capital letter – ***Students, Faculty, Grades***
  - If name consists of multiple words, every word in the table name starts with a capital letter – ***StudentProjects, FacultyEvaluations, FinancialTransactions***

Naming Conventions - CamelCase

- Attribute names:
  - Start with a lower case letter – ***amount, balance, penalty***
  - If name consists of multiple words, only the first word starts with a lowercase letter – each subsequent word in the table name starts with a capital letter – ***userID, firstName, dateOfBirth***

Naming Conventions - Underscore

- Database schema name:
  - The entire database name is **always** lowercase – ***registration, vehicles, university, birds***
  - If name consists of multiple words, all words in a database names are separated with underscores – ***student_projects, resume_system, patient_records_system***

Naming Conventions - Underscore

- Table names:
  - The entire table name is **<u>always</u>** lowercase – ***students, faculty, grades***
  - If name consists of multiple words, all words in an table name are separated with an underscore – ***student_projects, faculty_evaluations, financial_transactions***

Naming Conventions - Underscore

- Attribute names:
  - The entire database name is **always** lowercase – ***amount, balance, penalty***
  - If an attribute name consists of multiple words, all words in the attribute name are separated with underscores – ***user_id, first_name, date_of_birth***

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information
  - *account* : stores information about accounts
  - *depositor* : stores information about which customer owns which account
  - *customer* : stores information about customers

Database Schema vs. Instance

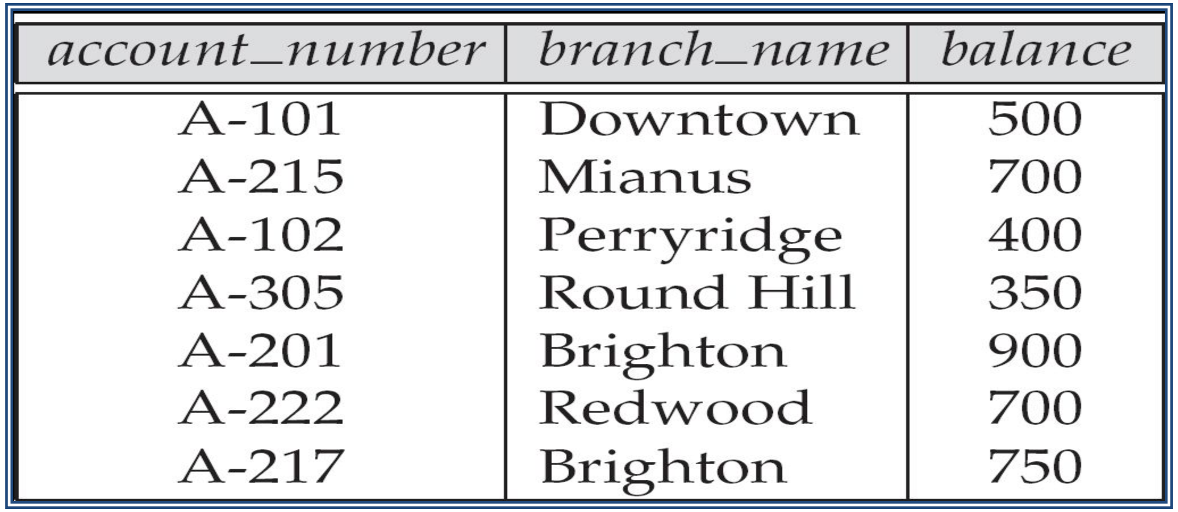
- **Database schema** – logical design of the database
- **Database instance** – a snapshot of the data in the database at a given instant in time

Database

- Storing all information as a single relation such as *bank*(*account_number, balance, customer_name*, ..) results in multiple problems.
- We will discuss this when we cover **Normalization theory** that deals with designing relational schemas

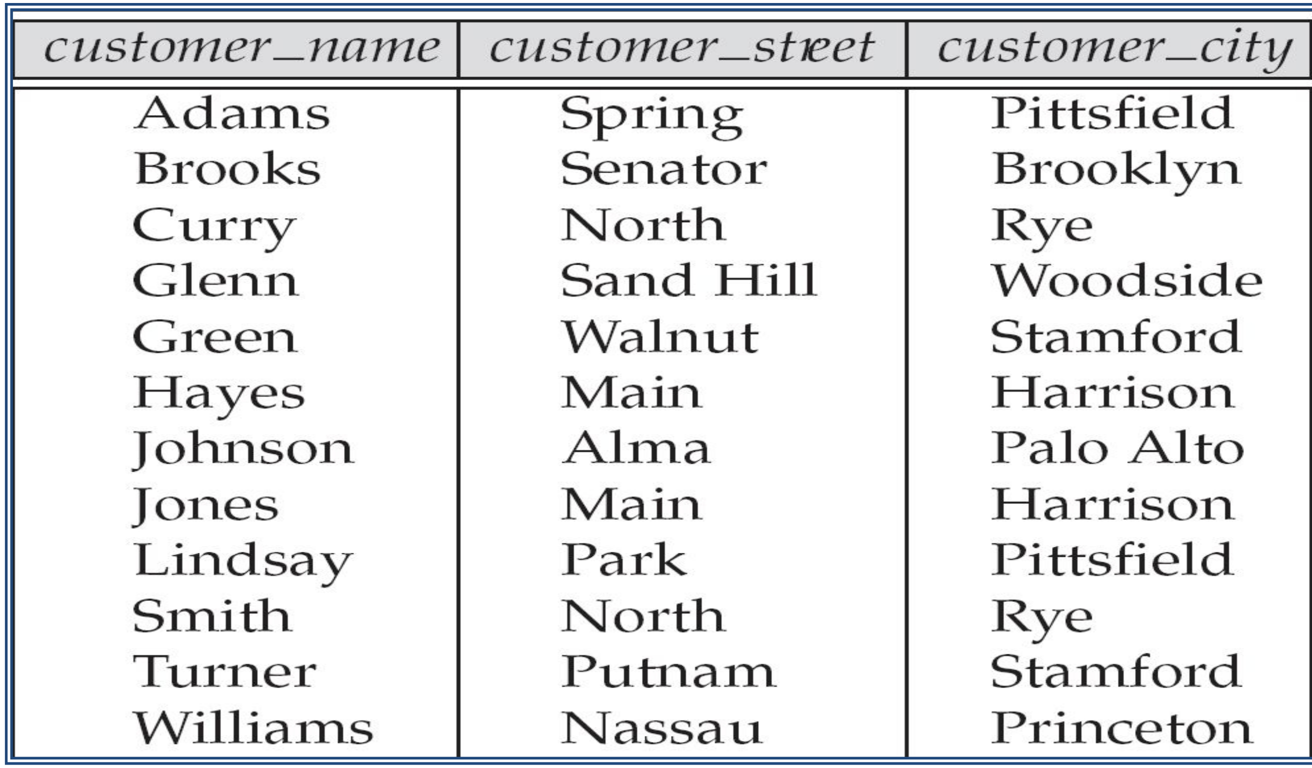
Example of a Database

account



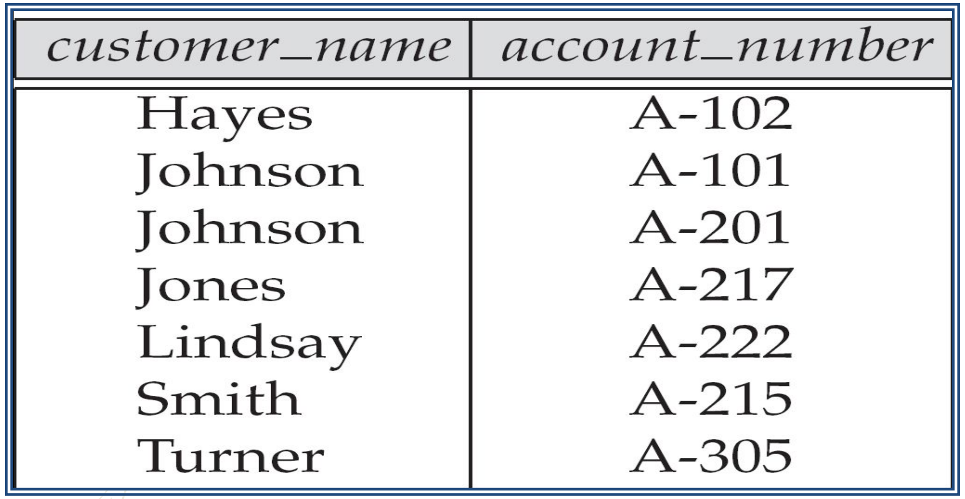
account_number	*branch_name*	*balance*
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

customer



customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

depositor



customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Keys

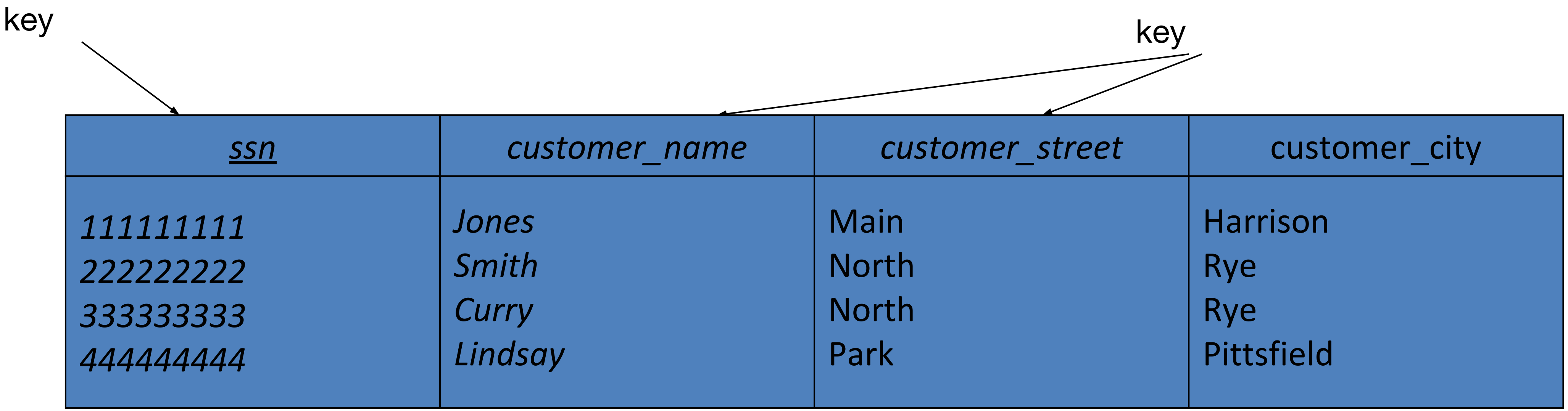
Let K is a list of attributes of a schema R. *K* is a **key** (also called a **superkey**) of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r* of that schema.

Keys

Example: {*customer_name, customer_street*} and {ssn}

are both keys of *Customer*, if no two customers can possibly have

the same name/address combination.



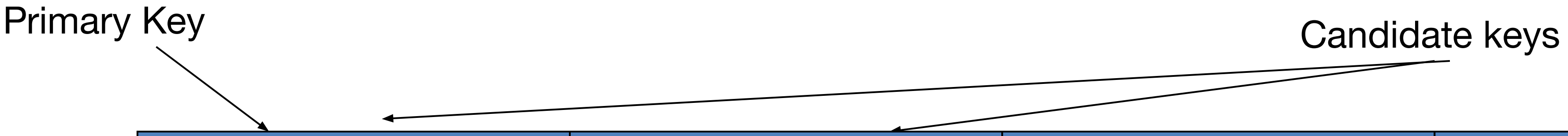
ssn	*customer_name*	*customer_street*	customer_city
111111111	*Jones*	Main	Harrison
222222222	*Smith*	North	Rye
333333333	*Curry*	North	Rye
444444444	*Lindsay*	Park	Pittsfield

Candidate Keys and Primary Key

- *K* is a **candidate key** if *K* is *minimal (i.e.,* no subset of K is a key).
- Relation schema may have more then one candidate key. Example: {*ssn*} and {*customer_name*} are candidate keys for *Customer*, since they are superkeys (assuming no two customers can possibly have the same name, or the same ssn)

Candidate Keys and Primary Key

- Among all candidate keys we must select *one* **Primary Key (e.g. ssn)**



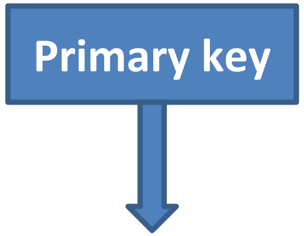
Primary Key

Candidate keys

ssn	*customer_name*	*customer_street*	customer_city
111111111	*Jones*	Main	Harrison
222222222	*Smith*	North	Rye
333333333	*Curry*	North	Rye
444444444	*Lindsay*	Park	Pittsfield

Keys

- **A key** is a logical way to access a record in a table.
- A key that **uniquely** identifies a record is called a **primary key**.



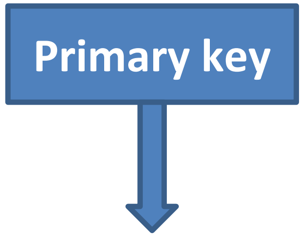
Primary key

employeeID	firstName	lastName	ssn	dob
232453	John	Doe	123-45-6789	04/07/1977
453437	Jane	Doe	987-65-4321	01/20/1991

Primary Key

- Uniquely defines the characteristics of each row
- Has to consist of characteristics that cannot be duplicated by any other row
- May consist of a single attribute or a multiple attributes in combination

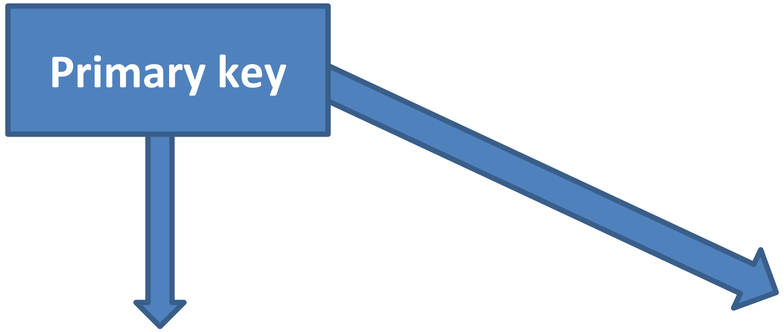
Single Attribute Primary Key



Primary key

employeeID	firstName	lastName	ssn	dob
232453	John	Doe	123-45-6789	04/07/1977
453437	Jane	Doe	987-65-4321	01/20/1991
459509	Michael	Smith	434-59-3952	02/29/1970

Multi-Attribute Primary Key



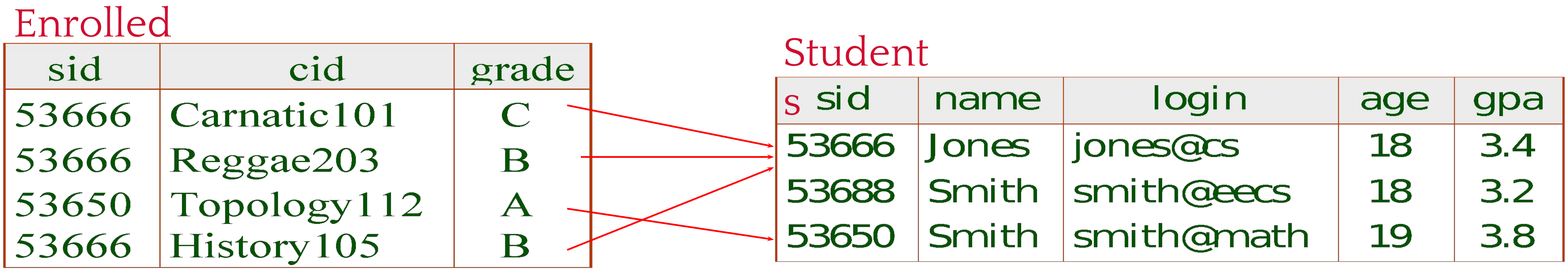
Primary key

orderNumber	itemId	itemSequence	itemName	quantity
232453	1	1	Book	3
453437	4	1	Notepad	7
459509	7	1	Pen	20
232453	1	2	Pencil	30

Foreign Keys

- **Foreign key**: Set of fields in one relation that is used to “refer” to a tuple (a row) in another relation.
- Each foreign key must correspond to a primary key of the second relation.
- Foreign keys are like “logical pointers”.

Foreign Keys Example



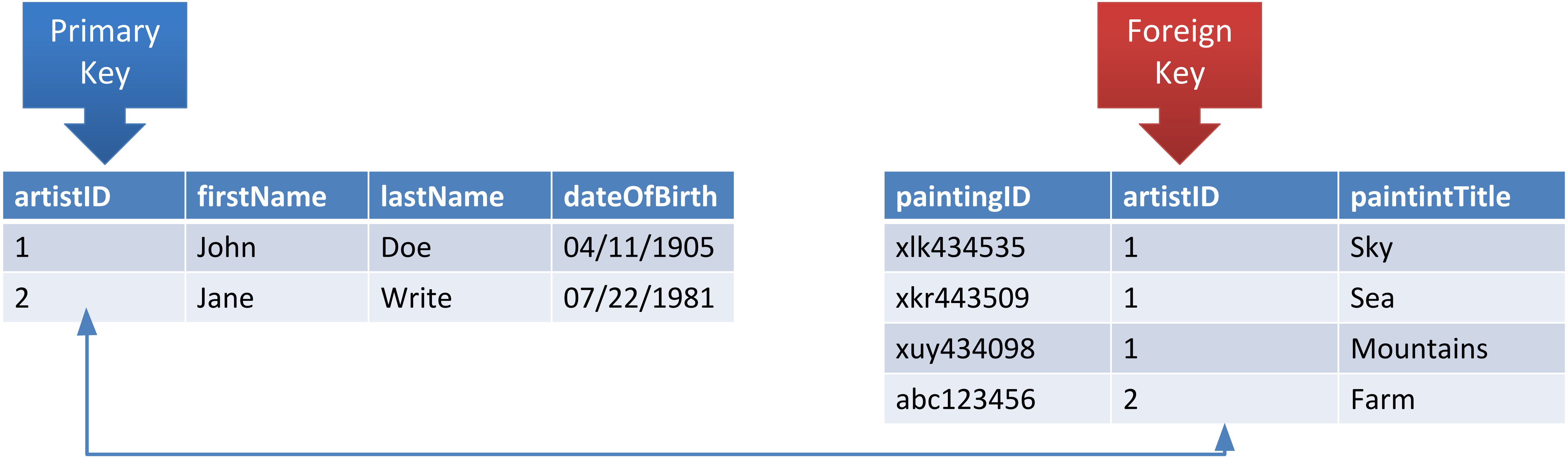
Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Student

s sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Foreign Keys Example



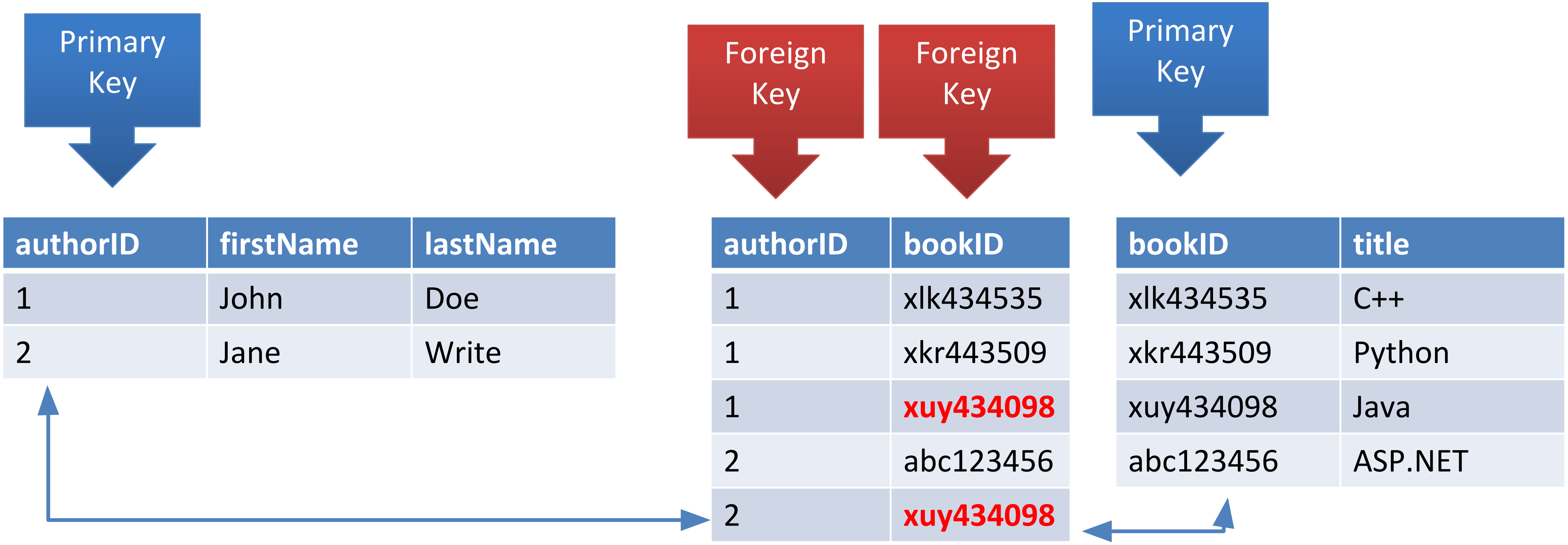
Primary Key

artistID	firstName	lastName	dateOfBirth
1	John	Doe	04/11/1905
2	Jane	Write	07/22/1981

Foreign Key

paintingID	artistID	paintintTitle
xlk434535	1	Sky
xkr443509	1	Sea
xuy434098	1	Mountains
abc123456	2	Farm

Foreign Keys Example



Primary Key

authorID	firstName	lastName
1	John	Doe
2	Jane	Write

Foreign Key

Foreign Key

authorID	bookID
1	xlk434535
1	xkr443509
1	**xuy434098**
2	abc123456
2	**xuy434098**

Primary Key

bookID	title
xlk434535	C++
xkr443509	Python
xuy434098	Java
abc123456	ASP.NET

Referential Integrity

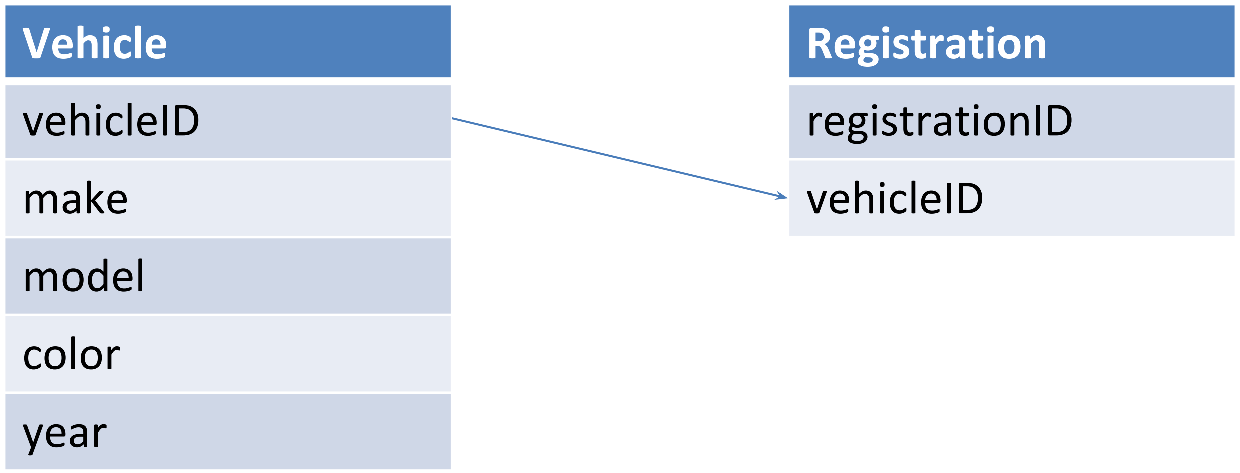
A property of data which, when satisfied, requires every value of one attribute (column) of a relation (table) to exist as a value of another attribute in a different (or the same) relation (table)

Relationships

- One-to-one (1:1)
- One-to-many (1:m)
- Many-to-many (m:n)

One-to-one (1:1)

- For each instance of table A, only one instance of table B exists, and vice-versa.
- For example, each vehicle registration is associated with only one engine number, and vice-versa



Vehicle
vehicleID
make
model
color
year

Registration
registrationID
vehicleID

One-to-one (1:1)

vehicleID	make	model	color	year
1	Ford	Taurus	Beige	2005
2	Honda	Civic	Red	2012
3	Toyota	Corolla	White	2009

+

registrationID	vehicleID
XYZ1233	1
ZLK4566	2
LKE4376	3

=

vehicleID	make	model	color	year	registrationID	vehicleID
1	Ford	Taurus	Beige	2005	XYZ1233	1
2	Honda	Civic	Red	2012	ZLK4566	2
3	Toyota	Corolla	White	2009	LKE4376	3

Examples of one-to-one relationships

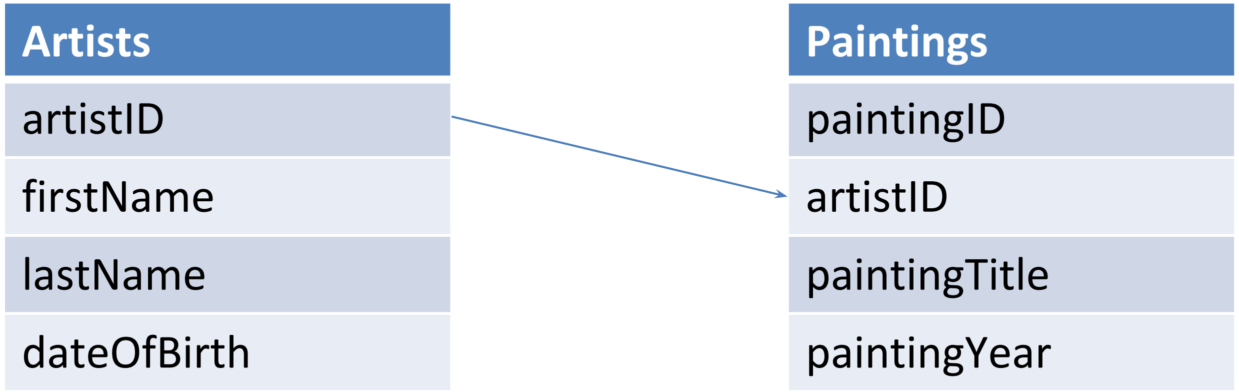
- One employee belongs to one organization.
- One dog belongs to one person (or one family).
- One person has one passport.
- A car model is made by one company.
- Water has one chemical makeup.

When to use one-to-one relationships

- Divide a table with many columns.
- Isolate part of a table for security reasons.
- Store data that is short-lived and could be easily deleted by simply deleting the table.
- Store information that applies only to a subset of the main table.
- Performance (we’ll discuss this later)

One-to-many (1:m)

- For each instance of table A, many instances of the table B exist, but for each instance of table B, only once instance of table A exists.
- For example, for each artist, there are many paintings. In this case each painting can only have been painted by one artist.



Artists
artistID
firstName
lastName
dateOfBirth

Paintings
paintingID
artistID
paintingTitle
paintingYear

One-to-many (1:m)

artistID	firstName	lastName	dateOfBirth
1	John	Doe	04/11/1905
2	Jane	Write	07/22/1981

+

paintingID	artistID	paintintTitle
xlk434535	1	Sky
xkr443509	1	Sea
xuy434098	1	Mountains
abc123456	2	Farm

=

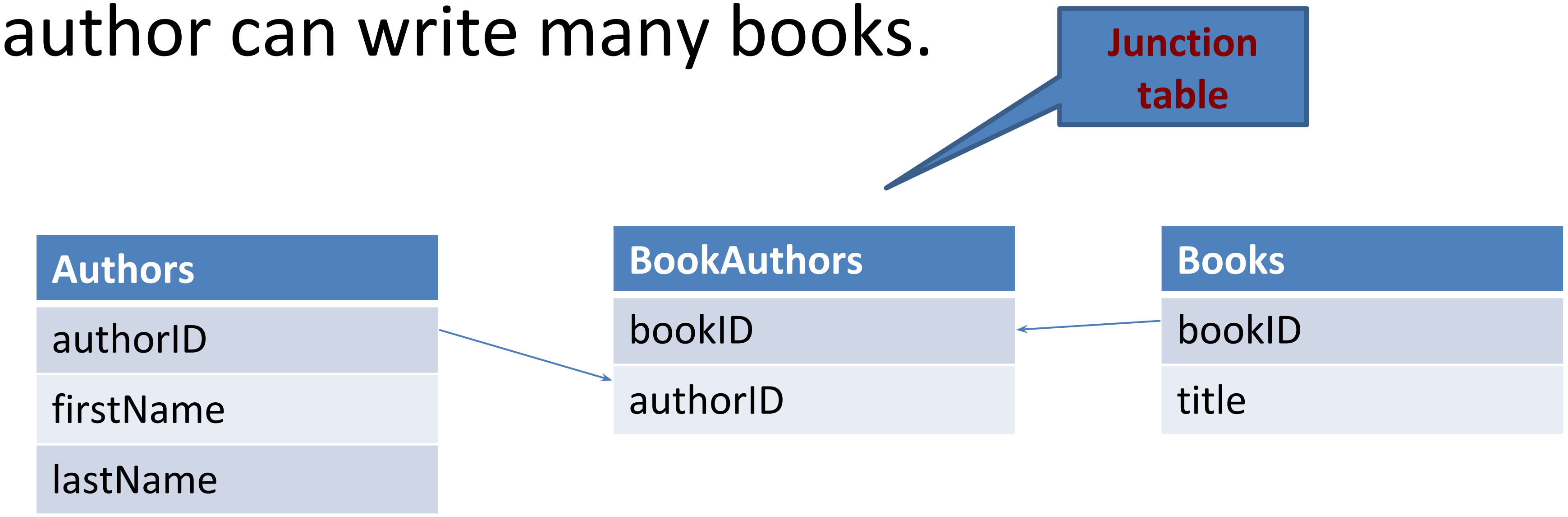
artistID	firstName	lastName	dateOfBirth	paintingID	artistID	paintintTitle
1	John	Doe	04/11/1905	xlk434535	1	Sky
1	John	Doe	04/11/1905	xkr443509	1	Sea
1	John	Doe	04/11/1905	xuy434098	1	Mountains
2	Jane	Write	07/22/1981	abc123456	2	Farm

Examples of one-to-many relationships

- **Camera/photo**: one camera can take many photos. One photo can be taken by only one camera.
- **Account/transaction**: one bank account can have many transactions. One transaction can belong to only one account

Many-to-many (m:n)

- For each instance of table A, there are many instances of table B, and for each instance of table B, there are many instances of the table A.
- For example, a book can have many authors, and each author can write many books.



Junction table

Authors
authorID
firstName
lastName

BookAuthors
bookID
authorID

Books
bookID
title

Many-to-many (m:n)

authorID	firstName	lastName
1	John	Doe
2	Jane	Write

+

bookID	authorID
xlk434535	1
xkr443509	1
xuy434098	1
abc123456	2
xuy434098	2

+

bookID	title
xlk434535	C++
xkr443509	Python
xuy434098	Java
abc123456	ASP.NET

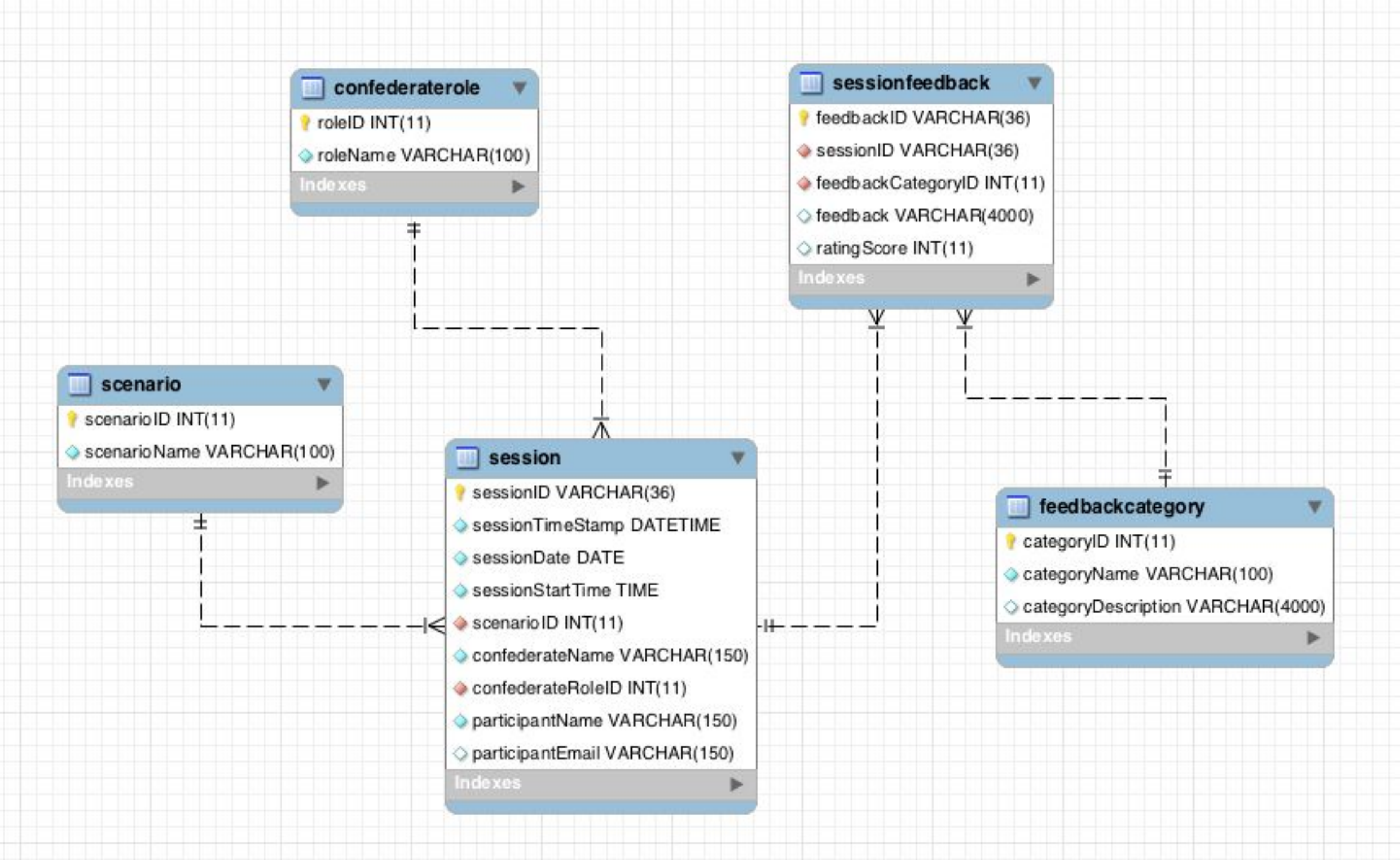
=

authorID	firstName	lastName	bookID	authorID	bookID	Title
1	John	Doe	Xlk434535	1	Xlk434535	C++
1	John	Doe	Xkr443509	1	Xkr443509	Python
1	John	Doe	**Xuy434098**	1	**xuy434098**	Java
2	Jane	Write	Abc123456	2	Abc123456	ASP.NET
2	Jane	Write	**Xuy434098**	2	**Xuy434098**	Java

Examples of many-to-many relationships

- **Doctor/patient** – one doctor can treat many patients; one patient can visit/be treated by many doctors.
- **Criminal/crime** – one criminal can commit many crimes; one crime can be committed by multiple criminals.

Schema Diagrams



confederaterole
- roleID INT(11)
- roleName VARCHAR(100)
- Indexes

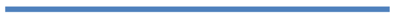
sessionfeedback
- feedbackID VARCHAR(36)
- sessionID VARCHAR(36)
- feedbackCategoryID INT(11)
- feedback VARCHAR(4000)
- ratingScore INT(11)
- Indexes

scenario
- scenarioID INT(11)
- scenarioName VARCHAR(100)
- Indexes

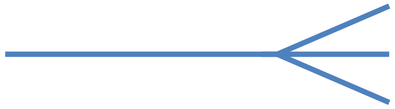
session
- sessionID VARCHAR(36)
- sessionTimeStamp DATETIME
- sessionDate DATE
- sessionStartTime TIME
- scenarioID INT(11)
- confederateName VARCHAR(150)
- confederateRoleID INT(11)
- participantName VARCHAR(150)
- participantEmail VARCHAR(150)
- Indexes

feedbackcategory
- categoryID INT(11)
- categoryName VARCHAR(100)
- categoryDescription VARCHAR(4000)
- Indexes

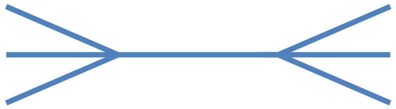
Relationship Notations



one-to-one (1:1)



one-to-many (1:m)



many-to-many (m:n)

Homework Assignment

- Chapter 3, pages 119-131
- Relational Data Model resources:
  - http://www.tutorialspoint.com/dbms/relational_data_model.htm
  - http://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm (skip Normalization section)
  - http://www.ntu.edu.sg/home/ehchua/programming/sql/relational_database_design.html (skip Normalization section)
- Assignment 1 – posted on CourseWeb