Databases & SQL

CSCI 3308

Liz Boese

Objectives

- Database Basics
- SQL
- NoSQL

RELATIONAL DATABASES

Databases

ORACLE DATABASE

Microsoft® SQL Server®

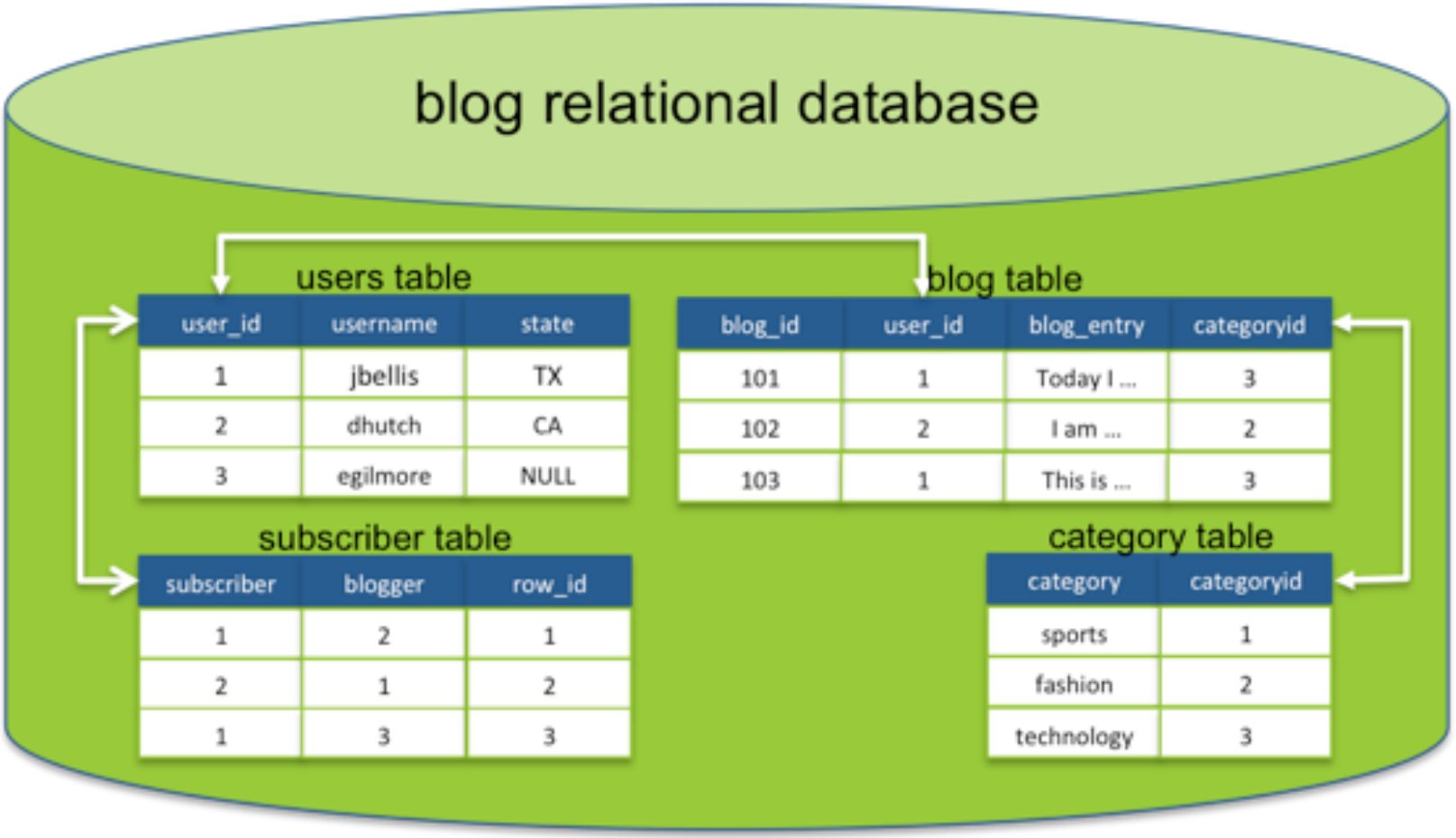
SQLite

PostgreSQL

MySQL®

Relational Databases

- **Relational Databases** store data according to a set of defined **entities** and **relations**
- **Entities** (tables) consist of related **attributes** (fields)



Entity Data Stored in Tables

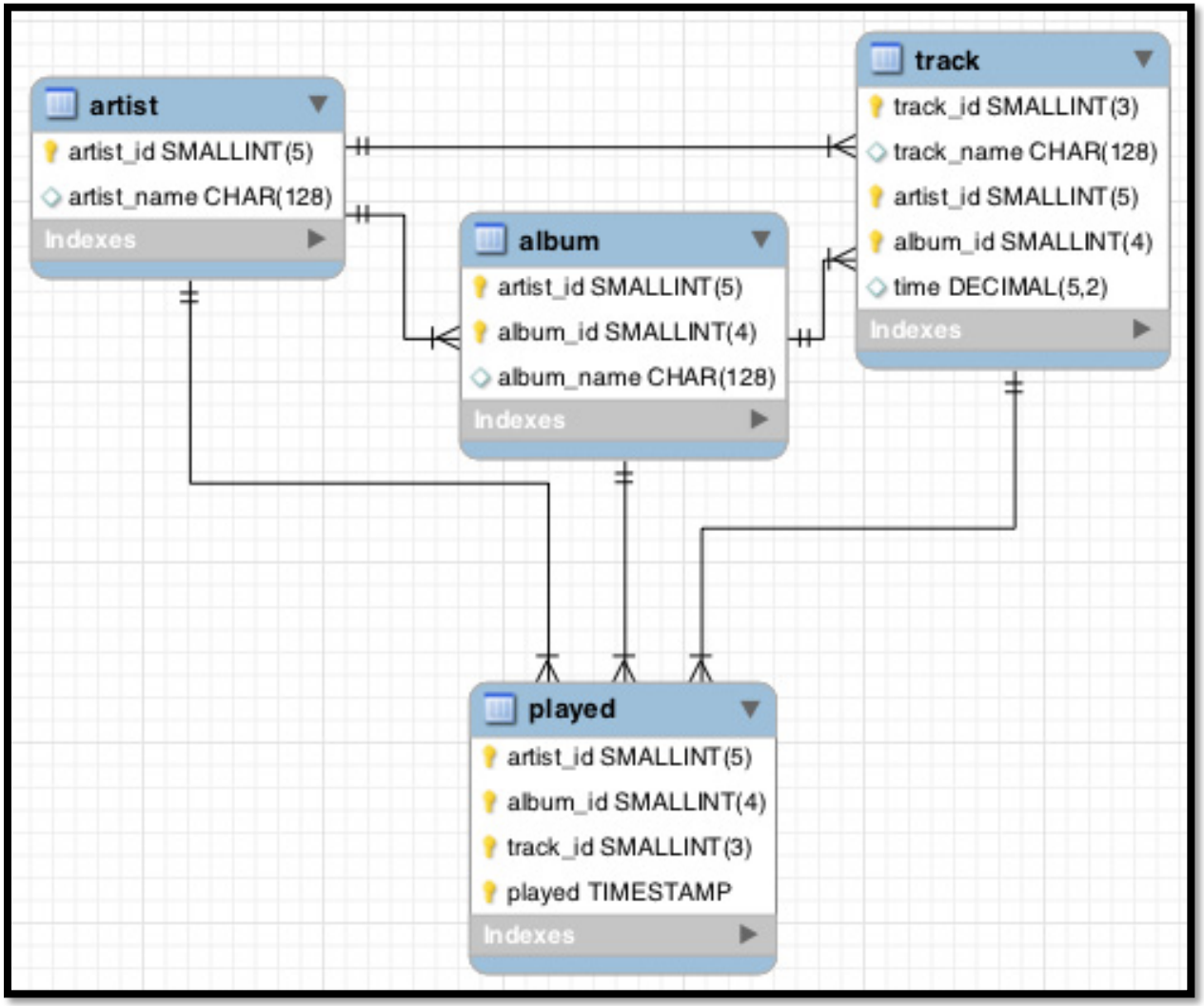
- Table columns contain attributes
- Table rows (tuples) contain items

Table: Category

ID Category	Name	Description
1	Auto Parts	Things to service a car
2	Electronics	TVs, DVD players, etc.

ER Diagrams

- **ER Diagrams** support modeling and analysis of database relational structure
  - **Relations** depict how two entities are related: "*X entity contains zero or more of Y entity*"



Definitions:

entity something about which data is collected, stored, and maintained

attribute a characteristic of an entity

relationship an association between entities

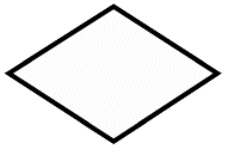
entity type a class of entities that have the same set of attributes

record an ordered set of attribute values that describe an instance of an entity type

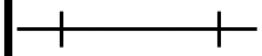
Symbols:

entity type

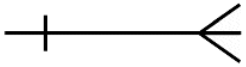
Also 1-to-*



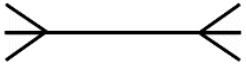
relationship between entities



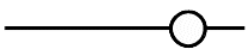
one-to-one association



one-to-many association



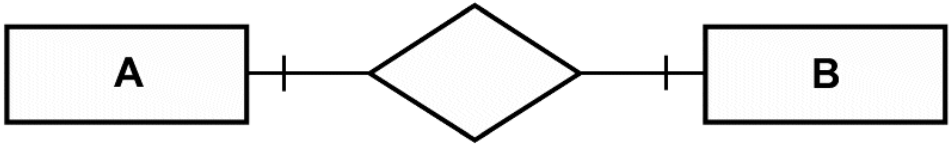
many-to-many association



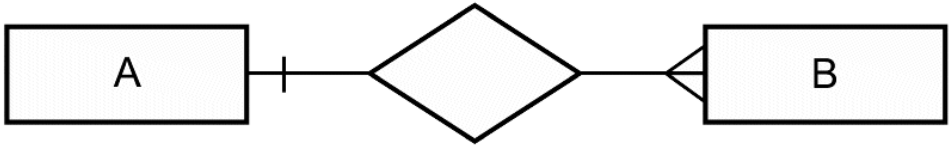
partly optional association

Examples:

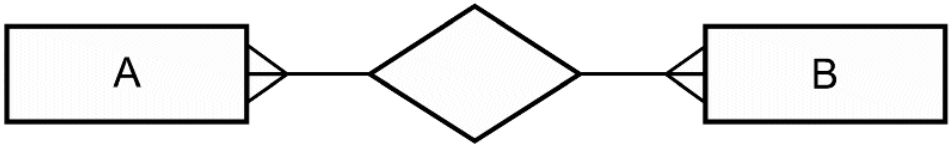
One A is associated with one B:



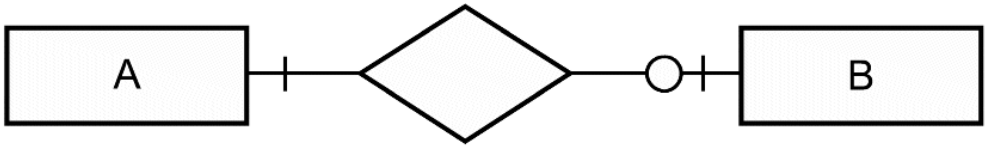
One A is associated with one or more B's:



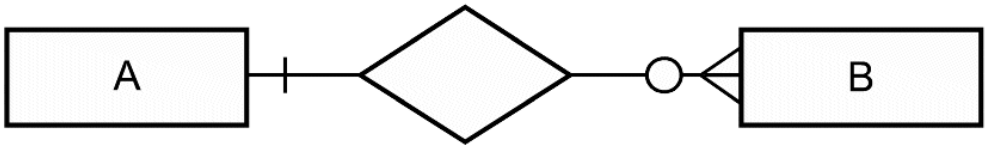
One or more A's are associated with one or more B's:



One A is associated with zero or one B:

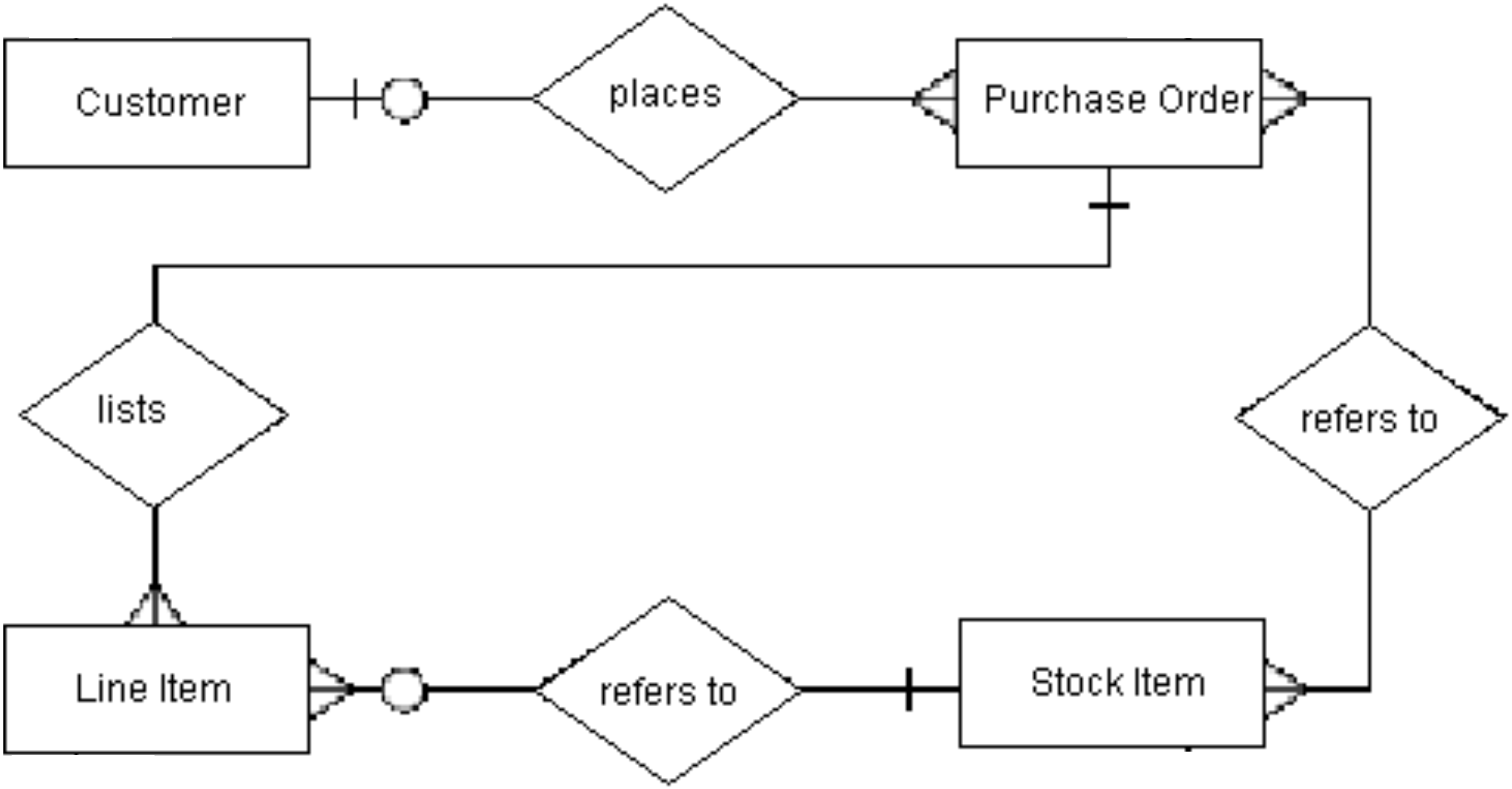


One A is associated with zero or more B's:

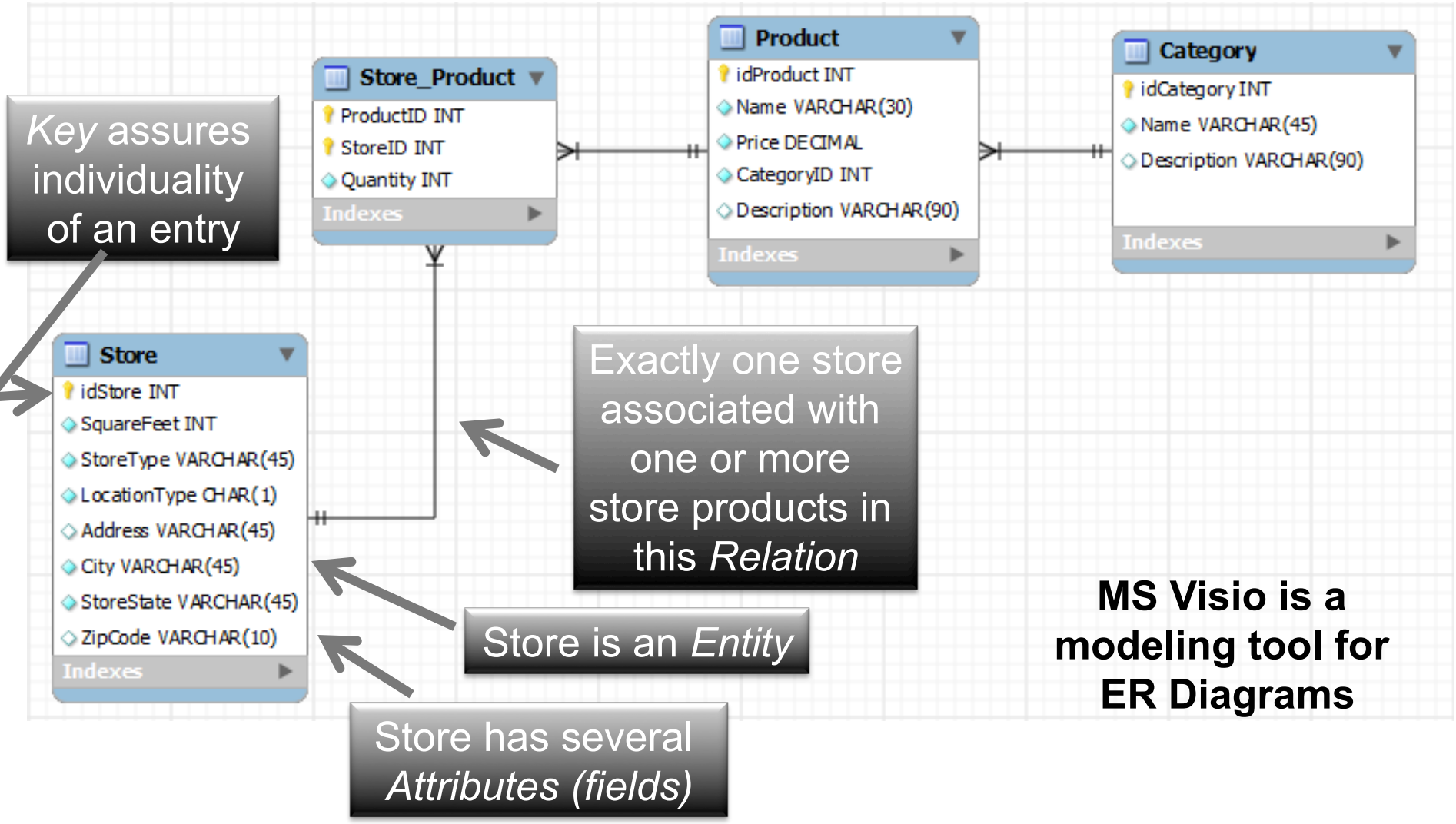


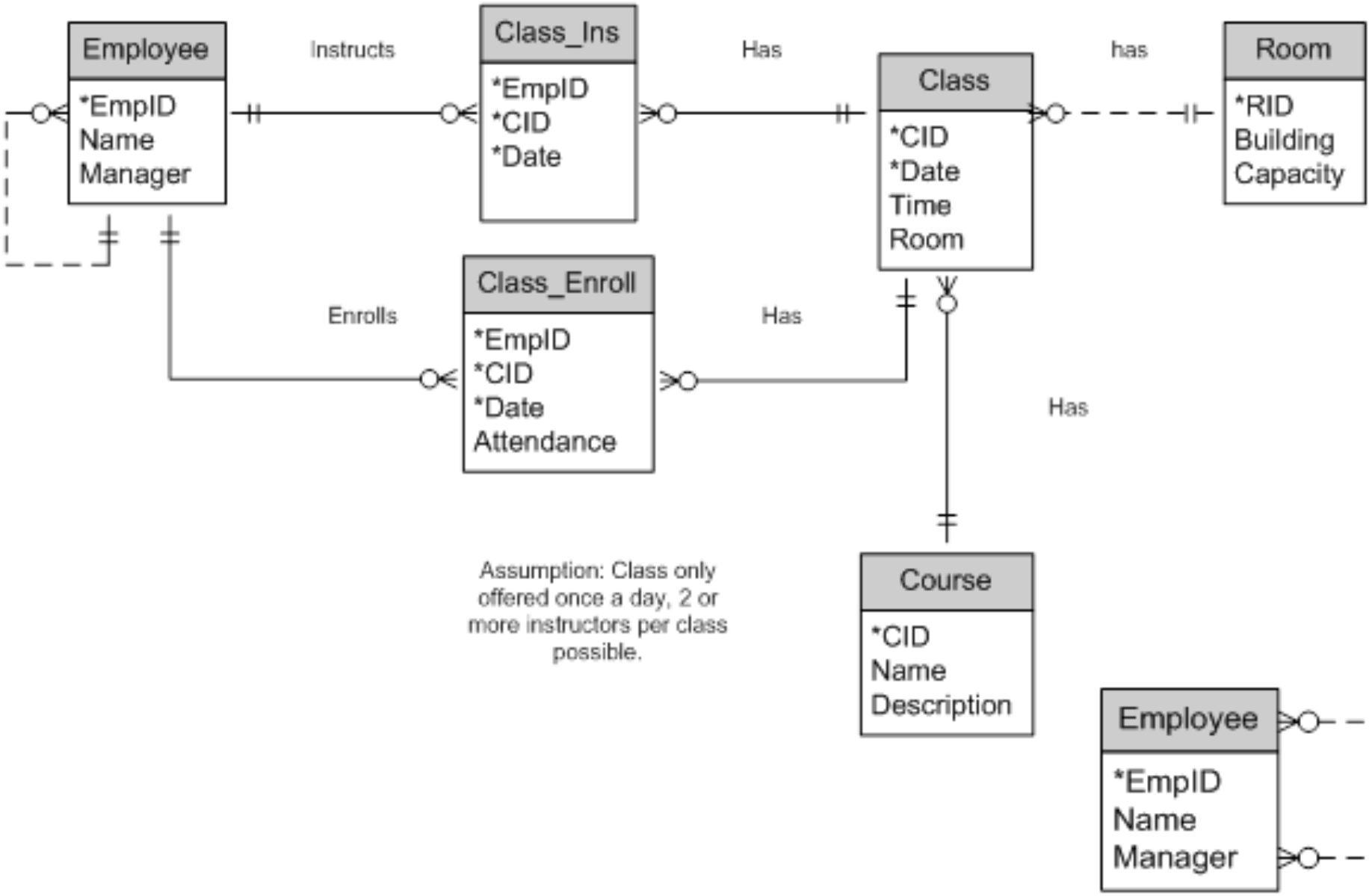
One A is associated with one B and one C:

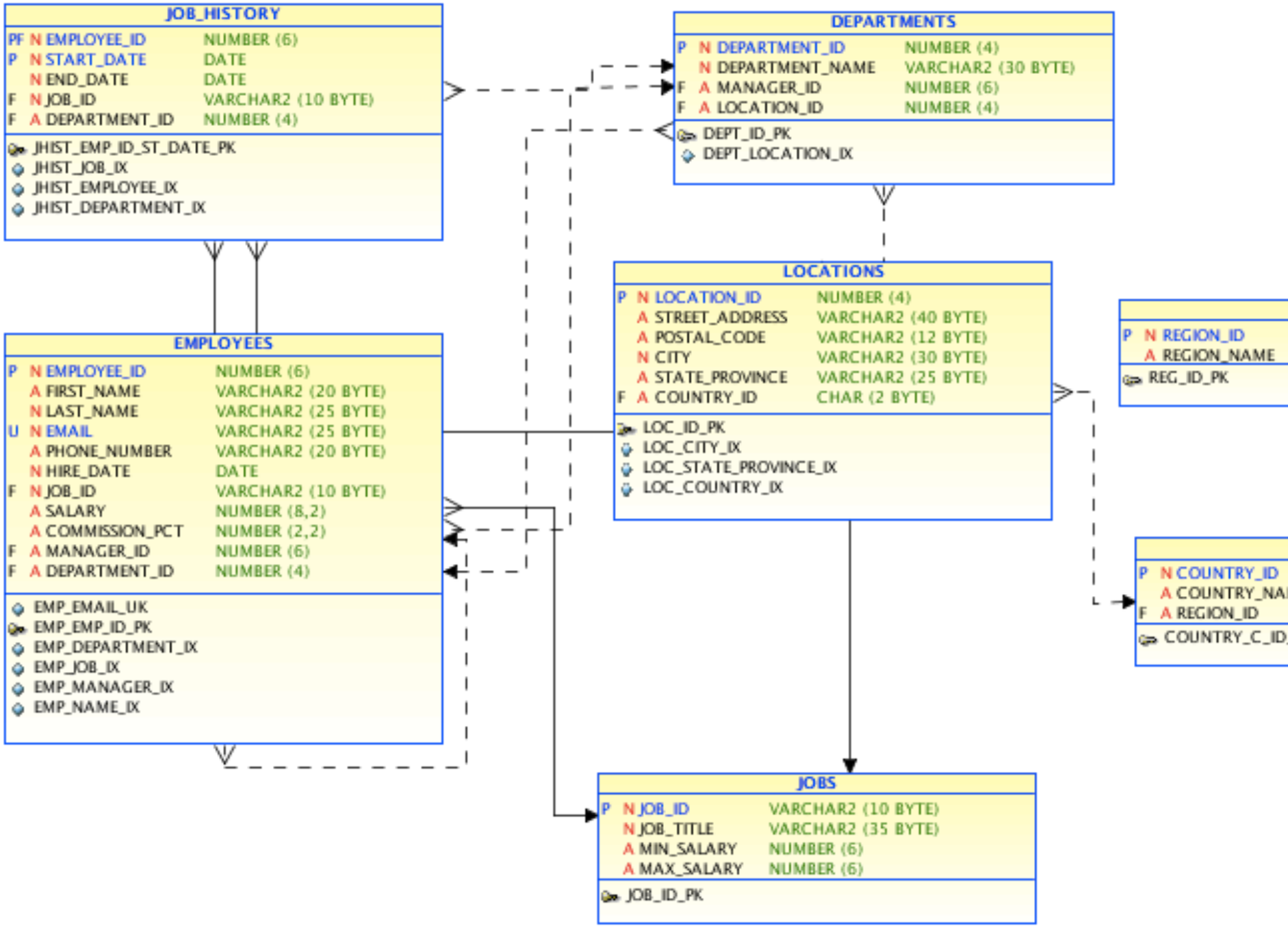
ER Diagram

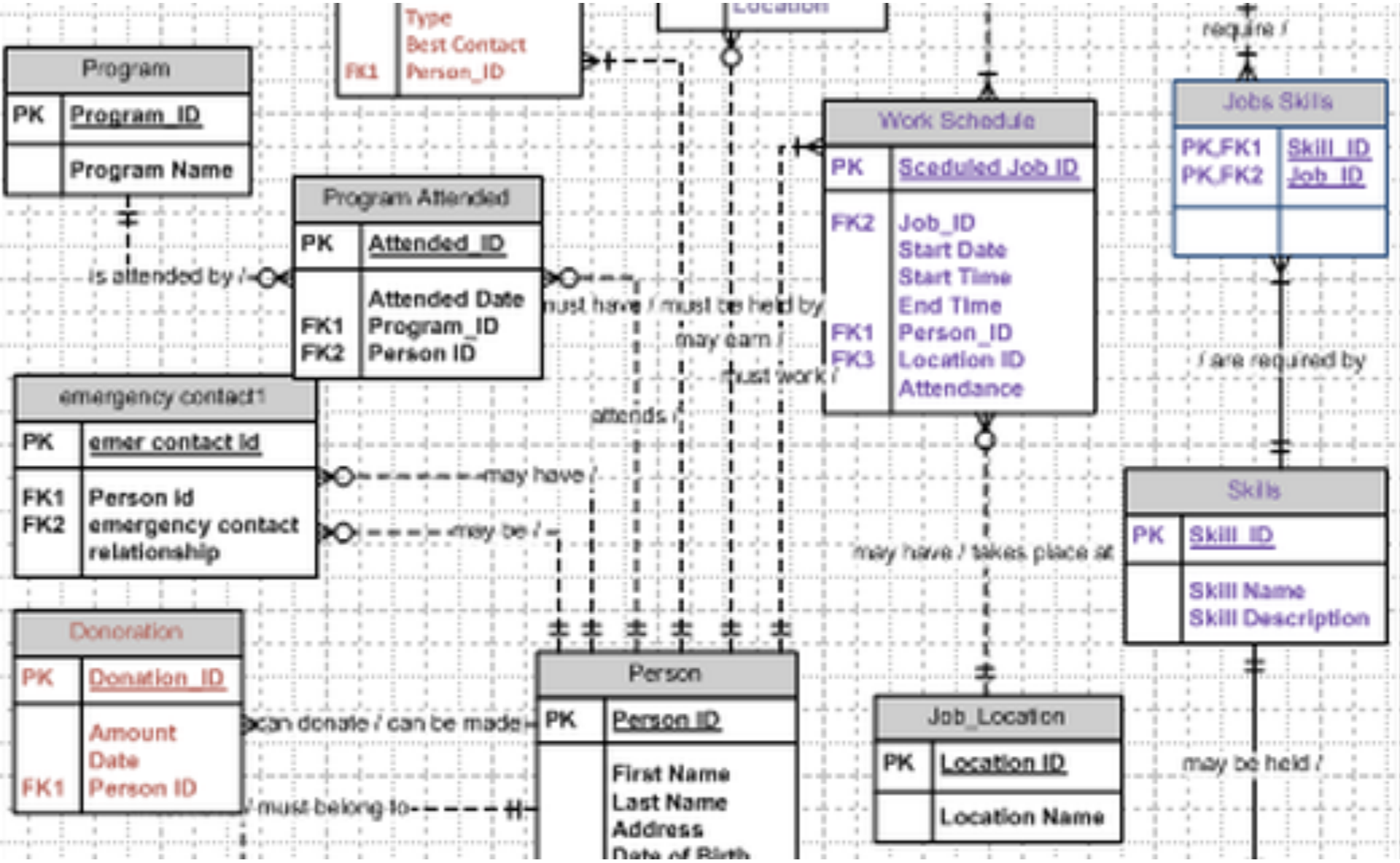

Customer
places
Purchase Order
lists
refers to
Line Item
refers to
Stock Item

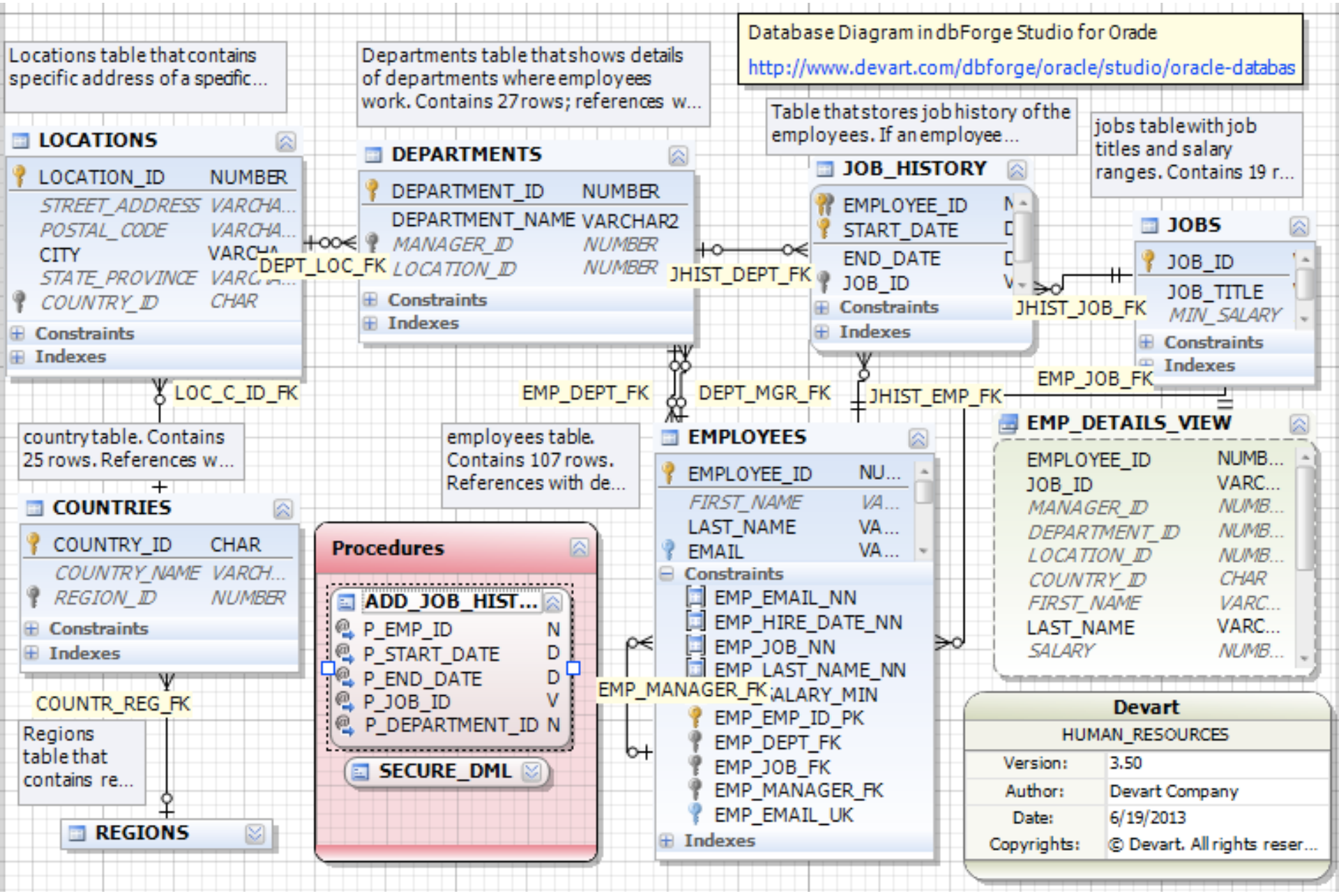
ER Diagram


Key assures individuality of an entry
Store_Product
ProductID INT
StoreID INT
Quantity INT
Indexes
Product
idProduct INT
Name VARCHAR(30)
Price DECIMAL
CategoryID INT
Description VARCHAR(90)
Indexes
Category
idCategory INT
Name VARCHAR(45)
Description VARCHAR(90)
Indexes
Store
idStore INT
SquareFeet INT
StoreType VARCHAR(45)
LocationType CHAR(1)
Address VARCHAR(45)
City VARCHAR(45)
StoreState VARCHAR(45)
ZipCode VARCHAR(10)
Indexes
Exactly one store associated with one or more store products in this Relation
Store is an Entity
Store has several Attributes (fields)
MS Visio is a modeling tool for ER Diagrams


Employee
*EmpID
Name
Manager
Instructs
Class_Ins
*EmpID
*CID
*Date
Has
Class
*CID
*Date
Time
Room
has
Room
*RID
Building
Capacity
Enrolls
Class_Enroll
*EmpID
*CID
*Date
Attendance
Has
Has
Assumption: Class only offered once a day, 2 or more instructors per class possible.
Course
*CID
Name
Description
Employee
*EmpID
Name
Manager


JOB_HISTORY
PF N EMPLOYEE_ID NUMBER (6)
P N START_DATE DATE
N END_DATE DATE
F N JOB_ID VARCHAR2 (10 BYTE)
F A DEPARTMENT_ID NUMBER (4)
JHIST_EMP_ID_ST_DATE_PK
JHIST_JOB_IX
JHIST_EMPLOYEE_IX
JHIST_DEPARTMENT_IX
DEPARTMENTS
P N DEPARTMENT_ID NUMBER (4)
N DEPARTMENT_NAME VARCHAR2 (30 BYTE)
F A MANAGER_ID NUMBER (6)
F A LOCATION_ID NUMBER (4)
DEPT_ID_PK
DEPT_LOCATION_IX
LOCATIONS
P N LOCATION_ID NUMBER (4)
A STREET_ADDRESS VARCHAR2 (40 BYTE)
A POSTAL_CODE VARCHAR2 (12 BYTE)
N CITY VARCHAR2 (30 BYTE)
A STATE_PROVINCE VARCHAR2 (25 BYTE)
F A COUNTRY_ID CHAR (2 BYTE)
LOC_ID_PK
LOC_CITY_IX
LOC_STATE_PROVINCE_IX
LOC_COUNTRY_IX
P N REGION_ID
A REGION_NAME
REG_ID_PK
P N COUNTRY_ID
A COUNTRY_NA
F A REGION_ID
COUNTRY_C_ID
EMPLOYEES
P N EMPLOYEE_ID NUMBER (6)
A FIRST_NAME VARCHAR2 (20 BYTE)
N LAST_NAME VARCHAR2 (25 BYTE)
U N EMAIL VARCHAR2 (25 BYTE)
A PHONE_NUMBER VARCHAR2 (20 BYTE)
N HIRE_DATE DATE
F N JOB_ID VARCHAR2 (10 BYTE)
A SALARY NUMBER (8,2)
A COMMISSION_PCT NUMBER (2,2)
F A MANAGER_ID NUMBER (6)
F A DEPARTMENT_ID NUMBER (4)
EMP_EMAIL_UK
EMP_EMP_ID_PK
EMP_DEPARTMENT_IX
EMP_JOB_IX
EMP_MANAGER_IX
EMP_NAME_IX
JOBS
P N JOB_ID VARCHAR2 (10 BYTE)
N JOB_TITLE VARCHAR2 (35 BYTE)
A MIN_SALARY NUMBER (6)
A MAX_SALARY NUMBER (6)
JOB_ID_PK


Type
Best Contact
FK1 Person_ID
require /
Program
PK Program_ID
Program Name
Work Schedule
PK Sceduled Job ID
FK2 Job_ID
Start Date
Start Time
End Time
FK1 Person_ID
FK3 Location ID
Attendance
Jobs Skills
PK,FK1 Skill_ID
PK,FK2 Job_ID
Program Attended
PK Attended_ID
Attended Date
FK1 Program_ID
FK2 Person ID
is attended by /
must have / must be held by
may earn /
must work /
/ are required by
attends /
emergency contact1
PK emer contact Id
FK1 Person id
FK2 emergency contact
relationship
may have /
may be / -
Skills
PK Skill_ID
Skill Name
Skill Description
may have / takes place at
Donoration
PK Donation_ID
Amount
Date
FK1 Person ID
can donate / can be made
/ must belong to
Person
PK Person ID
First Name
Last Name
Address
Date of Birth
Job_Location
PK Location ID
Location Name
may be held /


Database Diagram in dbForge Studio for Orade
http://www.devart.com/dbforge/oracle/studio/oracle-databas
Locations table that contains specific address of a specific...
Departments table that shows details of departments where employees work. Contains 27 rows; references w...
Table that stores job history of the employees. If an employee...
jobs table with job titles and salary ranges. Contains 19 r...
LOCATIONS
LOCATION_ID NUMBER
STREET_ADDRESS VARCHA...
POSTAL_CODE VARCHA...
CITY VARCH...
STATE_PROVINCE VARC...
COUNTRY_ID CHAR
Constraints
Indexes
DEPARTMENTS
DEPARTMENT_ID NUMBER
DEPARTMENT_NAME VARCHAR2
MANAGER_ID NUMBER
LOCATION_ID NUMBER
Constraints
Indexes
JOB_HISTORY
EMPLOYEE_ID N
START_DATE D
END_DATE D
JOB_ID V
Constraints
Indexes
JOBS
JOB_ID
JOB_TITLE
MIN_SALARY
Constraints
Indexes
DEPT_LOC_FK
JHIST_DEPT_FK
JHIST_JOB_FK
EMP_JOB_FK
LOC_C_ID_FK
EMP_DEPT_FK
DEPT_MGR_FK
JHIST_EMP_FK
country table. Contains 25 rows. References w...
employees table. Contains 107 rows. References with de...
EMPLOYEES
EMPLOYEE_ID NU...
FIRST_NAME VA...
LAST_NAME VA...
EMAIL VA...
Constraints
EMP_EMAIL_NN
EMP_HIRE_DATE_NN
EMP_JOB_NN
EMP_LAST_NAME_NN
EMP_SALARY_MIN
EMP_EMP_ID_PK
EMP_DEPT_FK
EMP_JOB_FK
EMP_MANAGER_FK
EMP_EMAIL_UK
Indexes
EMP_DETAILS_VIEW
EMPLOYEE_ID NUMB...
JOB_ID VARC...
MANAGER_ID NUMB...
DEPARTMENT_ID NUMB...
LOCATION_ID NUMB...
COUNTRY_ID CHAR
FIRST_NAME VARC...
LAST_NAME VARC...
SALARY NUMB...
COUNTRIES
COUNTRY_ID CHAR
COUNTRY_NAME VARCH...
REGION_ID NUMBER
Constraints
Indexes
COUNTR_REG_FK
Procedures
ADD_JOB_HIST...
P_EMP_ID N
P_START_DATE D
P_END_DATE D
P_JOB_ID V
P_DEPARTMENT_ID N
SECURE_DML
EMP_MANAGER_FK
Regions table that contains re...
REGIONS
Devart
HUMAN_RESOURCES
Version: 3.50
Author: Devart Company
Date: 6/19/2013
Copyrights: © Devart. All rights reser...

E-R Diagrams

- There are variations to ER Diagrams
- For this course we are focused on the ones we discussed. This is based on the diagrams that are usually auto-produced for databases.
  - Designate keys
  - Attributes inside box *(not ovals outside of it)*
- You need to be able to read an ER Diagram as you may have one to work with for coding

Structured Query Language

SQL

Select Basics

- **SQL** is a language to write queries over the data
- Most basic of all SQL statements

SELECT *columnlist* **FROM** *table*

- *Columnlist* named by comma separated list.
  SELECT firstName, lastName FROM t_contacts
- Use * as a shortcut for all columns.
  SELECT * FROM t_contacts
- **Aliasing** can be used to change the display
  - This is useful to improve readability of reports

  SELECT data3 AS start_date FROM t_contacts

All caps for SQL keywords is a common convention

Filtering and Ordering

```
SELECT columnlist
FROM table
WHERE columnCondition
ORDER BY columnOrder [ASC/DESC]
```

- **WHERE** clause filters based on columns using Boolean and logical operators

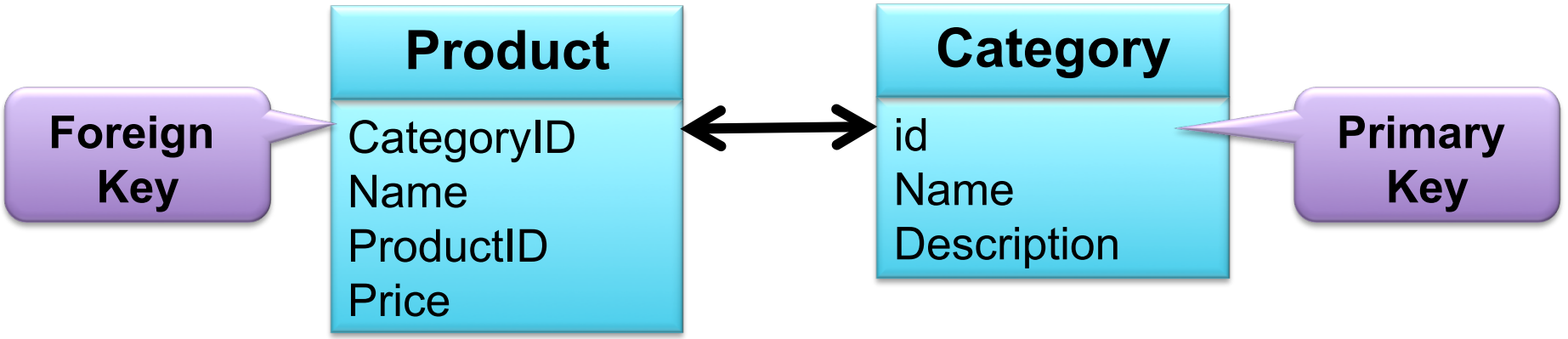
```
SELECT * FROM t_contacts
WHERE lastName = “Boese”
```

- **ORDER BY** defaults to **ASC**

```
SELECT * FROM t_contacts
ORDER BY lastName DESC
```

Joining tables

- Relationships are generally modeled through “key” columns.
- Primary and Foreign keys
- Example – “CategoryID” column in the Product table references the “id” column in the Category table.



Join Syntax

- Get results from multiple tables using where

```
SELECT columnlist
FROM table1, table2
WHERE table1.col_1 = table2.col_2
```

- Example

```
SELECT name, price, description
FROM Product, Category
WHERE Category.id = Product.categoryID
```

Join Syntax

- Get results from multiple tables

```
SELECT columnlist
FROM table1
JOIN table2 ON table1.col_1 = table2.col_2
```

- Example

```
SELECT name, price, description
FROM Product
JOIN Category ON categoryID = id
```

Join Syntax

- WHERE vs JOIN?
- SIDEBAR… *advanced concepts [optional]*

INNER JOIN is ANSI syntax which you should use.
It is generally considered more readable, especially when you join lots of tables.
It can also be easily replaced with an OUTER JOIN whenever a need arises.

The WHERE syntax is more relational model oriented.

A result of two tables JOIN'ed is a cartesian product of the tables to which a filter is applied which selects only those rows with joining columns matching.

It's easier to see this with the WHERE syntax.

Also note that MySQL also has a STRAIGHT_JOIN clause.

Using this clause, you can control the JOIN order: which table is scanned in the outer loop and which one is in the inner loop.

You cannot control this in MySQL using WHERE syntax.

Name Resolution

```
SELECT name, price, description
FROM Product
JOIN Category ON categoryID = id
```

- Query engine must be able to resolve columns;
  in the example: “name” is ambiguous if there are columns called “name” in both tables.

- Fully qualified syntax can be used

  SELECT tableName.columnName, …

  Ex:

  SELECT Product.Name, Category.Name, …

Aliases

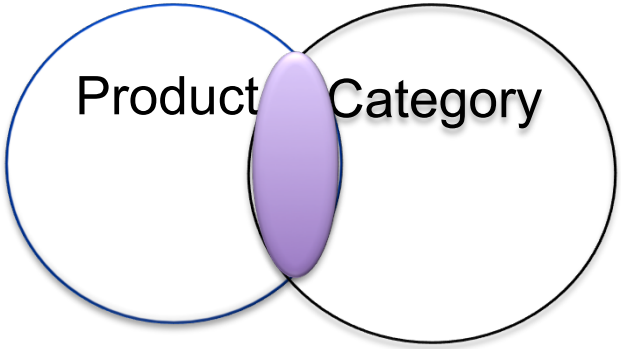
Aliases

- Shorten typing
- Rename columns.

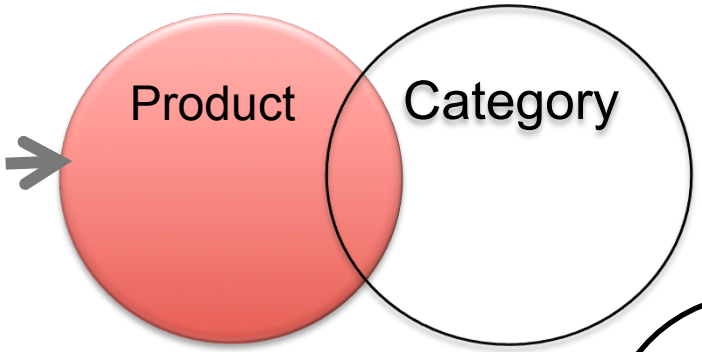
- On tables and/or column names
  - FROM Product p
- "AS" keyword optional
  - *(usually used on column names but not on table names)*
  - *p.name as "ProductName"*

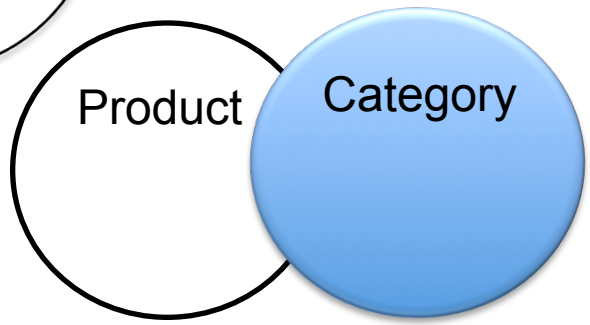
```
SELECT p.name as "ProductName", c.name
FROM Product p
JOIN category c ON p.categoryID = c.id
```

JOIN Types



- **JOIN** is shortcut for **INNER JOIN**
  - Only records that match are returned.
- **OUTER JOIN** may return records that do not match the filter condition.
  - **LEFT JOIN, RIGHT JOIN** syntax signifies which table has "optional" values.

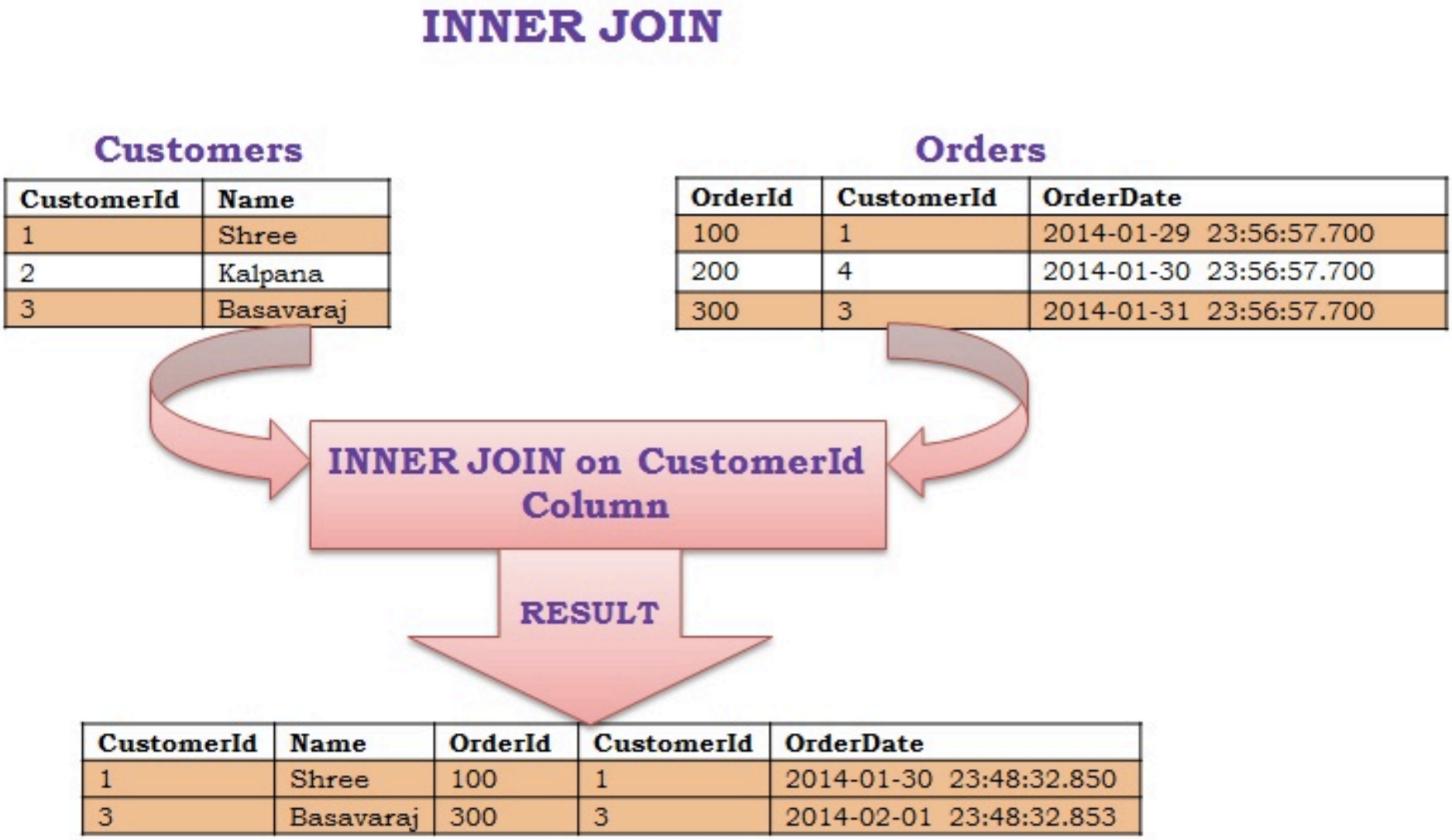




- Null values returned for non-matches.

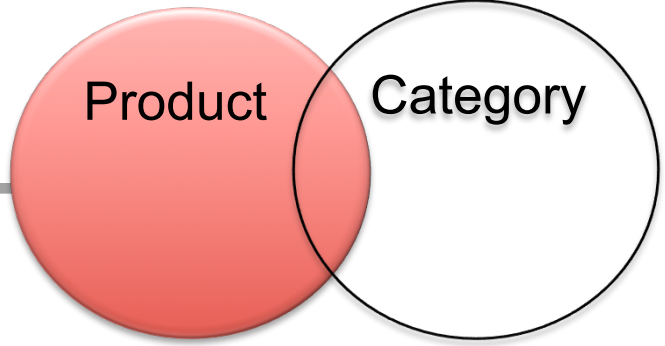
Inner Join

```
SELECT *
FROM  Customers C
        INNER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



http://sqlhints.com/category/sql-server/sql-server-tutorial/

Left Join

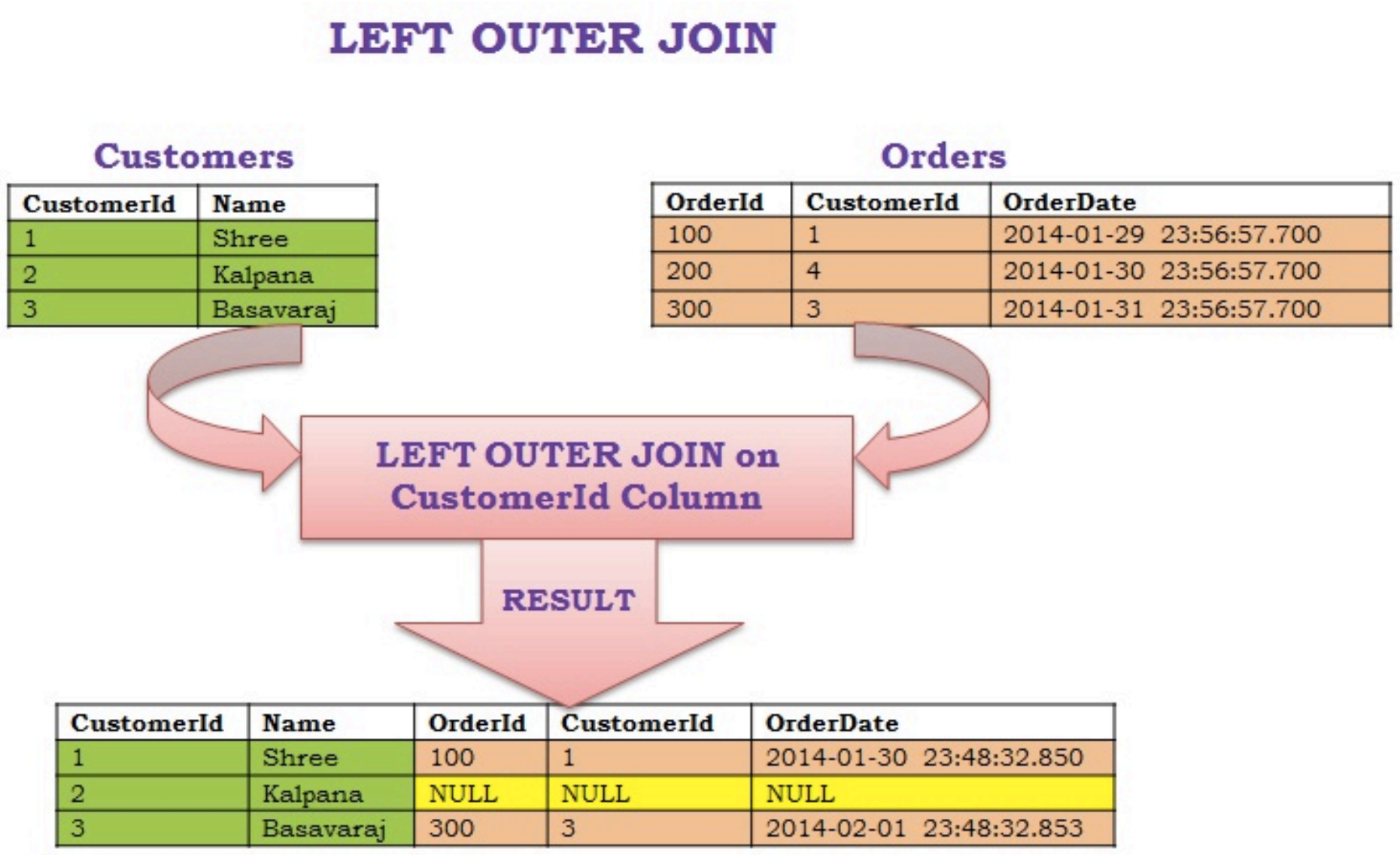


- Left join, sometimes called left outer join
- The LEFT JOIN keyword returns all the rows from the left table (Product), even if there are no matches in the right table (Category).
- *A list of all products and their categories even if a product not associated to a category.*
  If no products use a particular category, that category does not show up

```
SELECT p.name, c.name
FROM Product p
LEFT JOIN category c ON p.categoryID = c.id
```

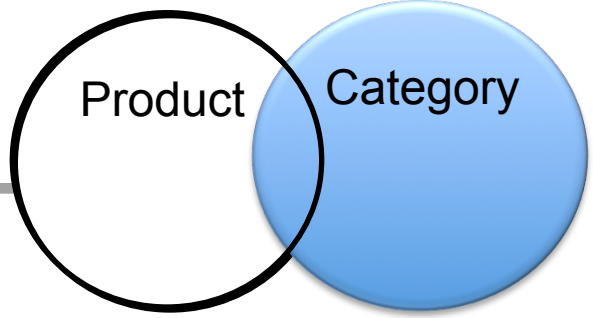
Left Join

```
SELECT *
FROM  Customers C
        LEFT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```



http://sqlhints.com/category/sql-server/sql-server-tutorial/

Right Join



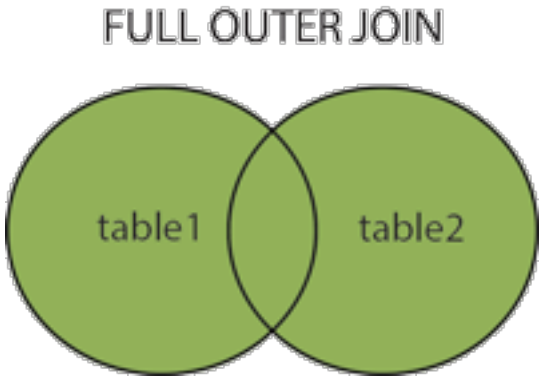
- Right join, sometimes called right outer join
- The RIGHT JOIN keyword returns all the rows from the right table (Category), even if there are no matches in the left table (Product).
- *A list of all categories and their matching products even if a category is not associated to a product.*
  If no categories reference a particular product, that product does not show up

```
SELECT p.name, c.name
FROM Product p
RIGHT JOIN category c ON p.categoryID = c.id
```

Right Outer Join

```
SELECT *
FROM  Customers C
        RIGHT OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```

Full Outer Join



- Full outer join
- The FULL OUTER JOIN keyword returns all the rows from both tables.

```
SELECT p.name, c.name
FROM Product p
FULL OUTER JOIN category c ON p.categoryID = c.id
```

Full Outer Join

```
SELECT *
    FROM Customers C
        FULL OUTER JOIN Orders O
            ON O.CustomerId = C.CustomerId
```

FULL OUTER JOIN

Customers

CustomerId	Name
1	Shree
2	Kalpana
3	Basavaraj

Orders

OrderId	CustomerId	OrderDate
100	1	2014-01-29 23:56:57.700
200	4	2014-01-30 23:56:57.700
300	3	2014-01-31 23:56:57.700

FULL OUTER JOIN on CustomerId Column

RESULT

CustomerId	Name	OrderId	CustomerId	OrderDate
1	Shree	100	1	2014-01-30 23:48:32.850
2	Kalpana	NULL	NULL	NULL
3	Basavaraj	300	3	2014-02-01 23:48:32.853
NULL	NULL	200	4	2014-01-31 23:48:32.853

Aggregate Functions

- Return a single value calculated from values in the column.
- Examples: **AVG(), COUNT(), MAX(), MIN(), SUM()**
- Can use a "**GROUP BY**" clause that includes all columns not aggregated to achieve results by groups

```
SELECT AVG(p.Price) 'Average Price'
,c.Name 'Category Name'
,c.Description 'Category Description'
FROM Product p JOIN Category c
ON p.categoryID = c.idCategory
GROUP BY c.Name, c.Desription
```

Unions

- Sometimes easier to break a query into pieces.
- Combine the separate result sets into one final set.
  - Default is to remove duplicates
  - Include duplicates with: UNION ALL
- Column number and type must match

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Modifying Data

- Update statement used to change existing data.

```
UPDATE table_name
SET column1 = value1, column2 = value2,...
WHERE some_column = some_value;
```

- Delete used to remove records

```
DELETE FROM table_name
WHERE some_column = some_value;
```

- Insert used to create new records

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Transactions

- **ACID** Transactions
  - **Atomic** – transactions are all or nothing
  - **Consistent** – all transactions must leave database in a valid state as per constraints
  - **Isolated** – ordering is preserved in face of concurrency
  - **Durable** – committed transaction persists even in face of system failure
- Multiple statements are either completely executed or fail as a whole.
  - Ensures integrity of the database.

STORED PROCEDURES

Stored Procedures

- Pre-compiled SQL statement that reside in the database.

```
CREATE PROCEDURE uspGetAddress
AS
SELECT * FROM AdventureWorks.Person.Address
GO
```

- Usually consist of input parameters that may perform several related actions.

- Called from other statements using EXEC.

```
EXEC uspGetAddress
```

Programming Frameworks

- Allow for interacting with databases via language constructs.
- Connections, statements, and result sets are represented by classes.
- Can send dynamically generated SQL to the database and process the results.
- Example: Yii framework *(see slides from previous week)*

Object Relational Mapping

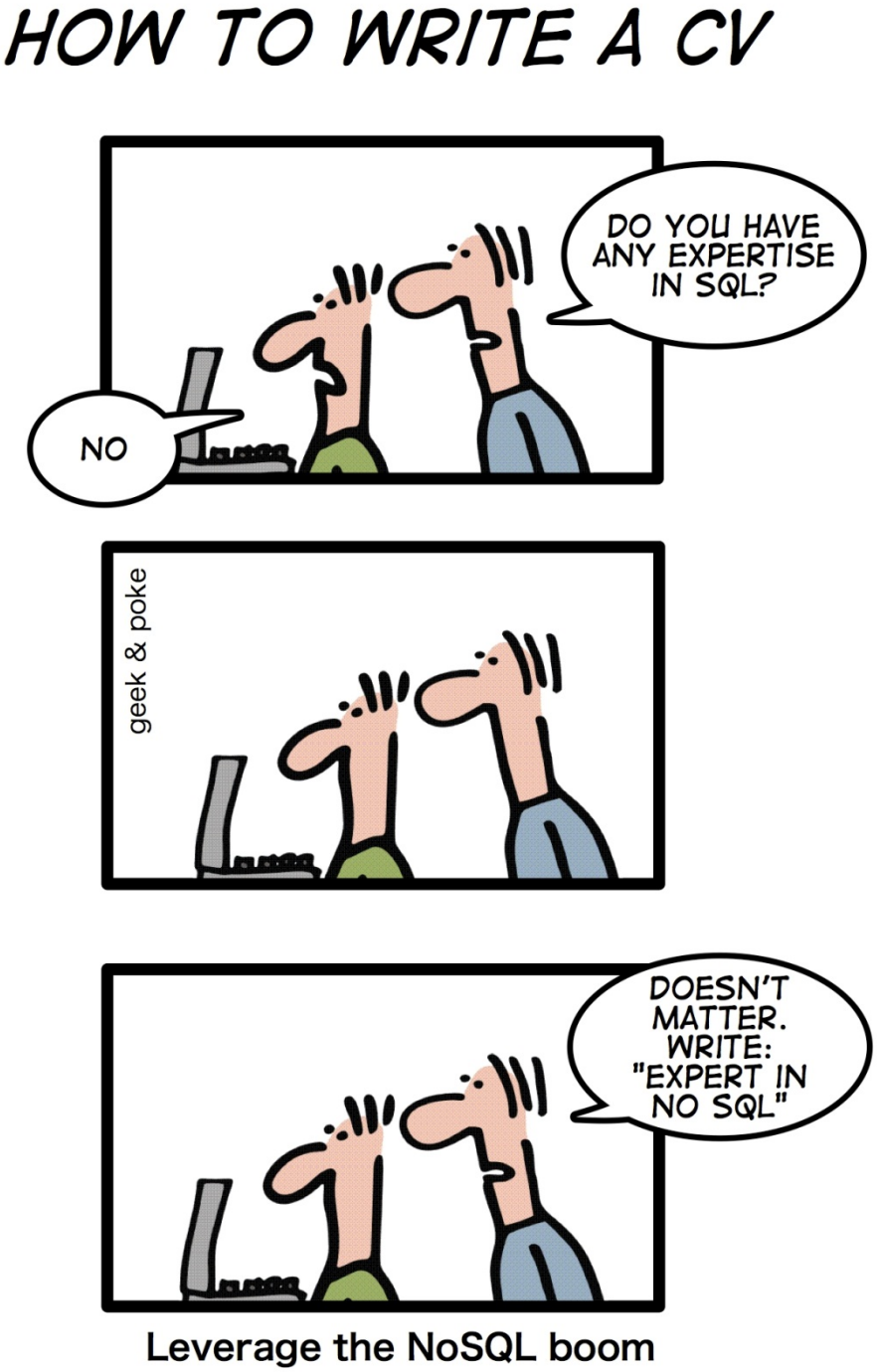
- "**Domain object modeling**" implies a class represents a single table.
  - Store table represented by a Store class.
- Manually mapping columns to object properties.
- Modeling objects via annotations.
  - Queries done in programming language.

Example: **ActiveRecord** in Ruby on Rails & many Frameworks

NOSQL: NO SQL

NOSQL:

NOT ONLY SQL



NoSQL:

A database that provides a mechanism for storage and retrieval of data that is modeled in means **other than the tabular relations** used in relational databases. Motivations for this approach include **simplicity of design**, **horizontal scaling** and finer **control over availability**.

-Wikipedia

Use Cases

Motivations

- Big data
- Scalability
- Data format
- Manageability

Use Cases

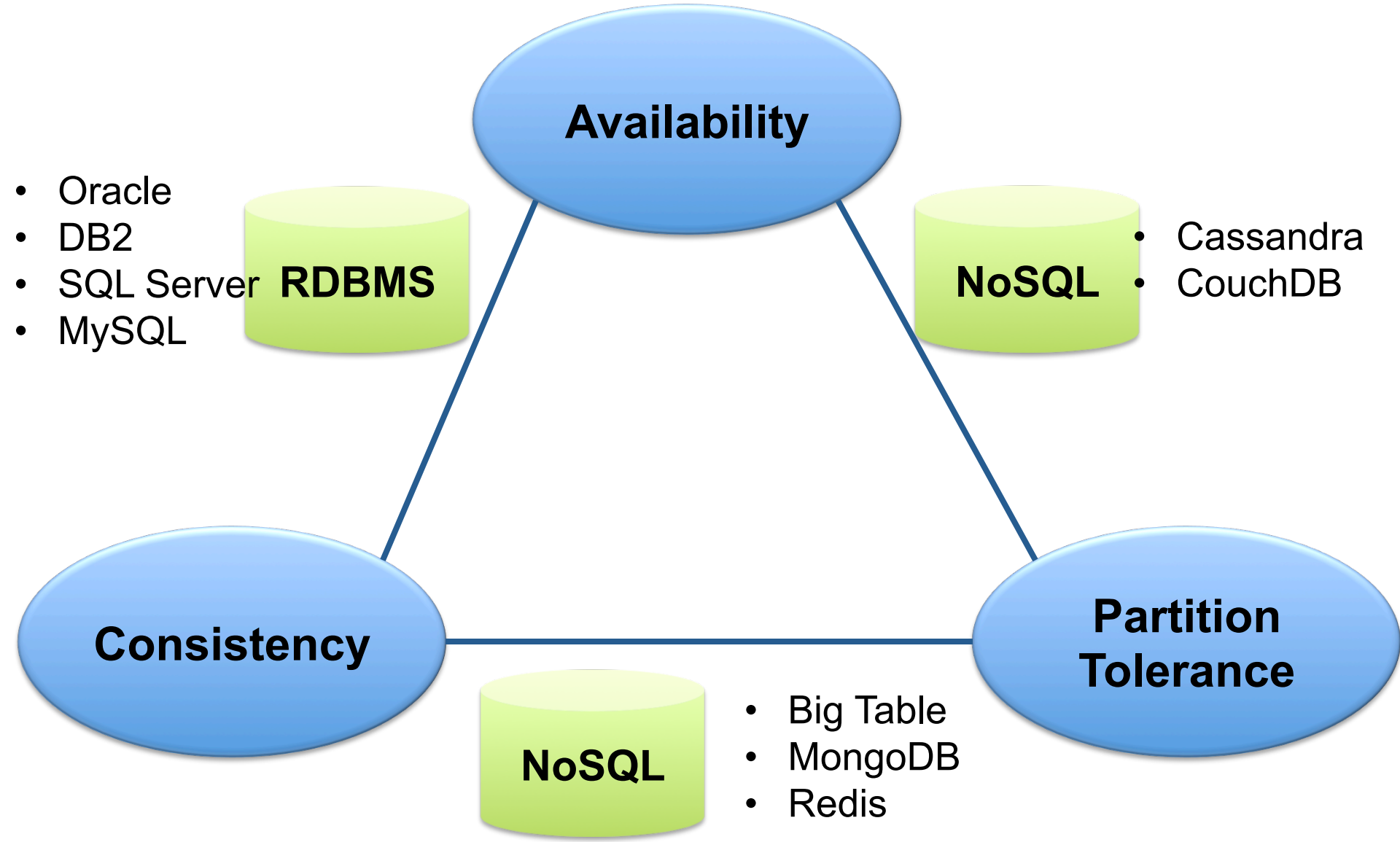
- Massive write performance.
- Fast key value look ups.
- Flexible schema and data types.
- No single point of failure.
- Fast prototyping and development.
- Out of the box scalability.
- Easy maintenance.

Big Data

- Collect
- Store
- Organize
- Analyze
- Share

Data growth outruns the ability to manage it so we need **scalable** solutions.

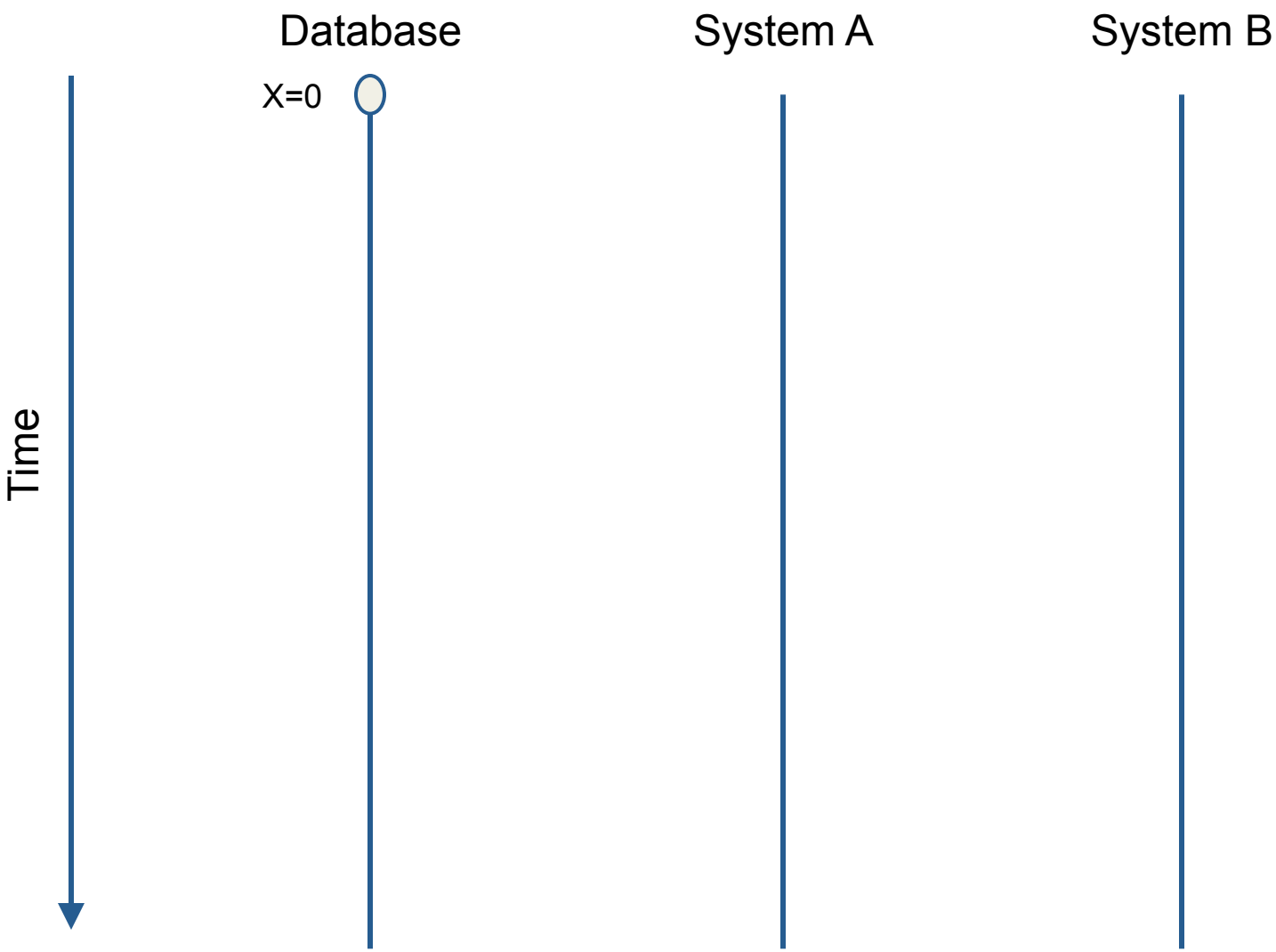
RDBMS vs NoSQL



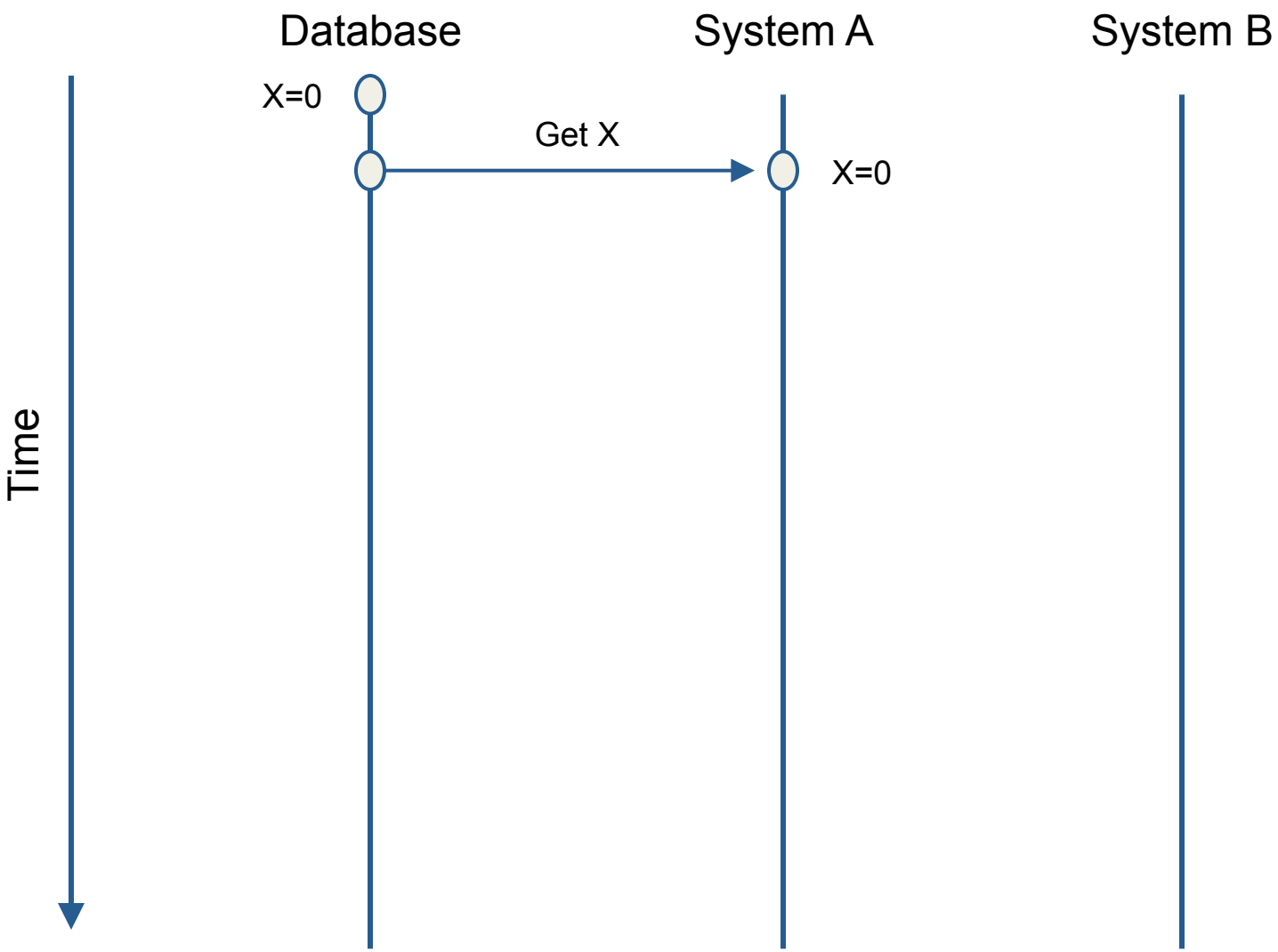
- NoSQL DBs (often) Lack:
  - SQL-based Interfaces
  - ACID Transactions
  - Strong Consistency

By giving up ACID properties, one can achieve higher performance and scalability.

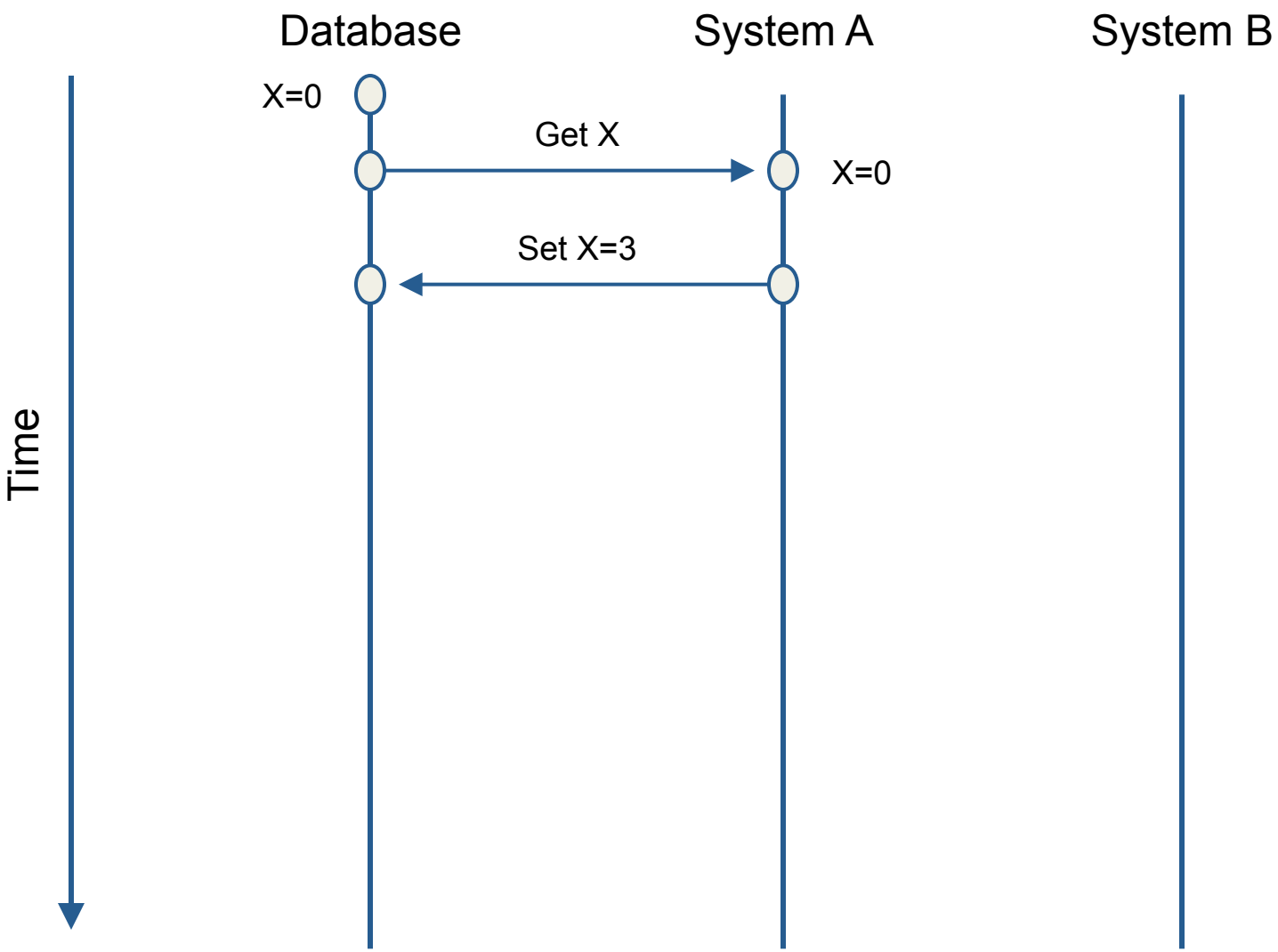
Strong Consistency



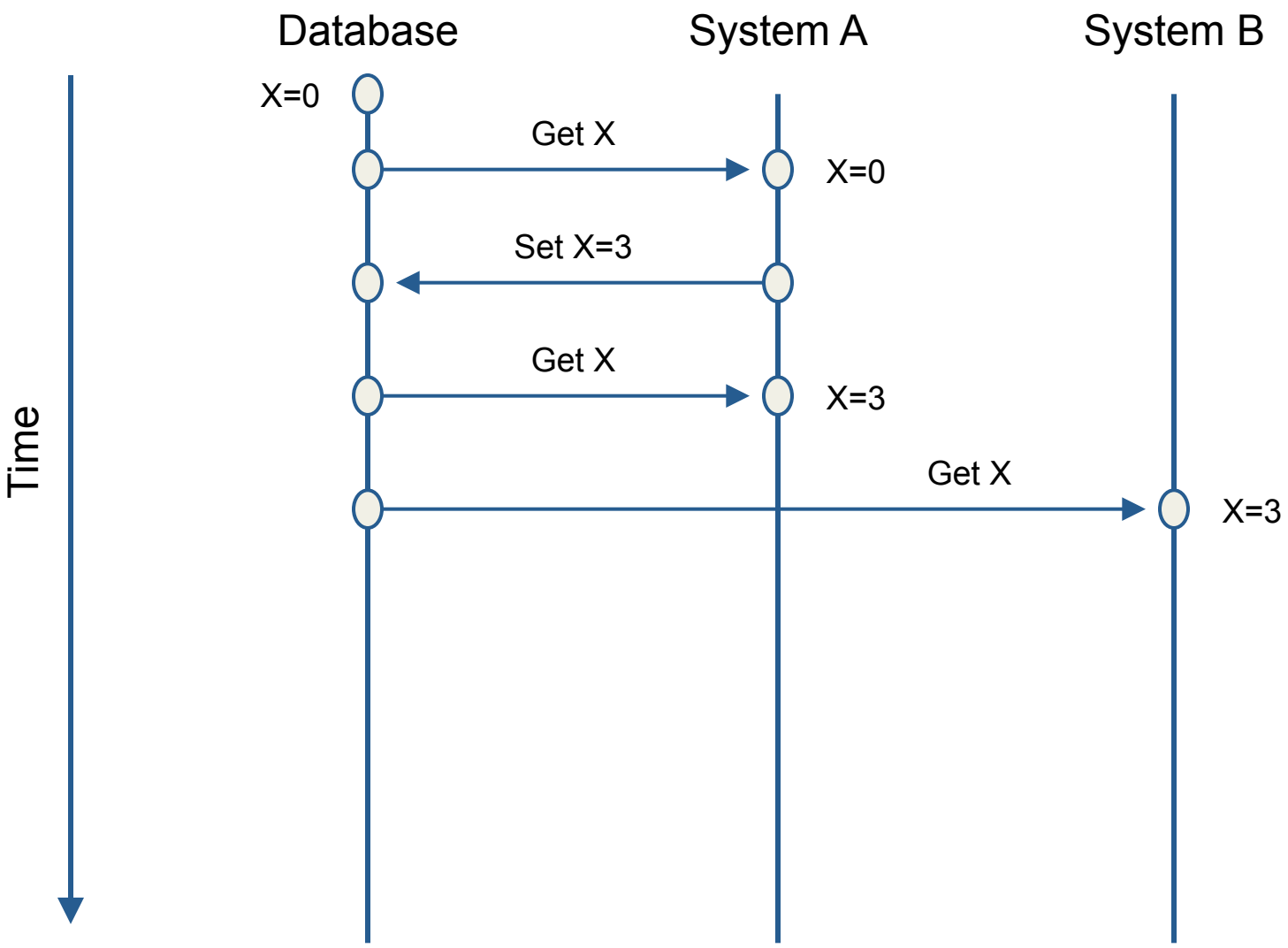
Strong Consistency



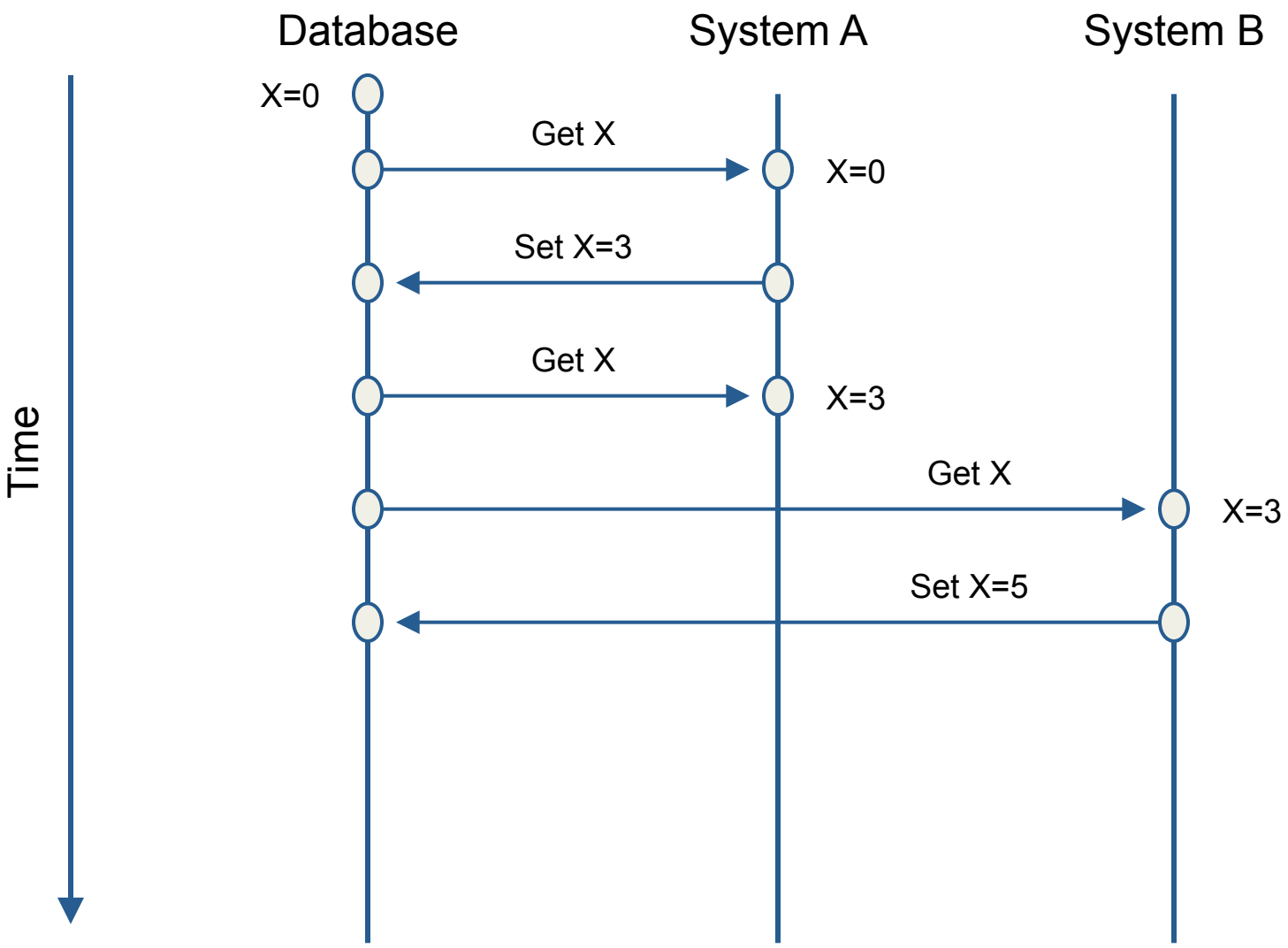
Strong Consistency



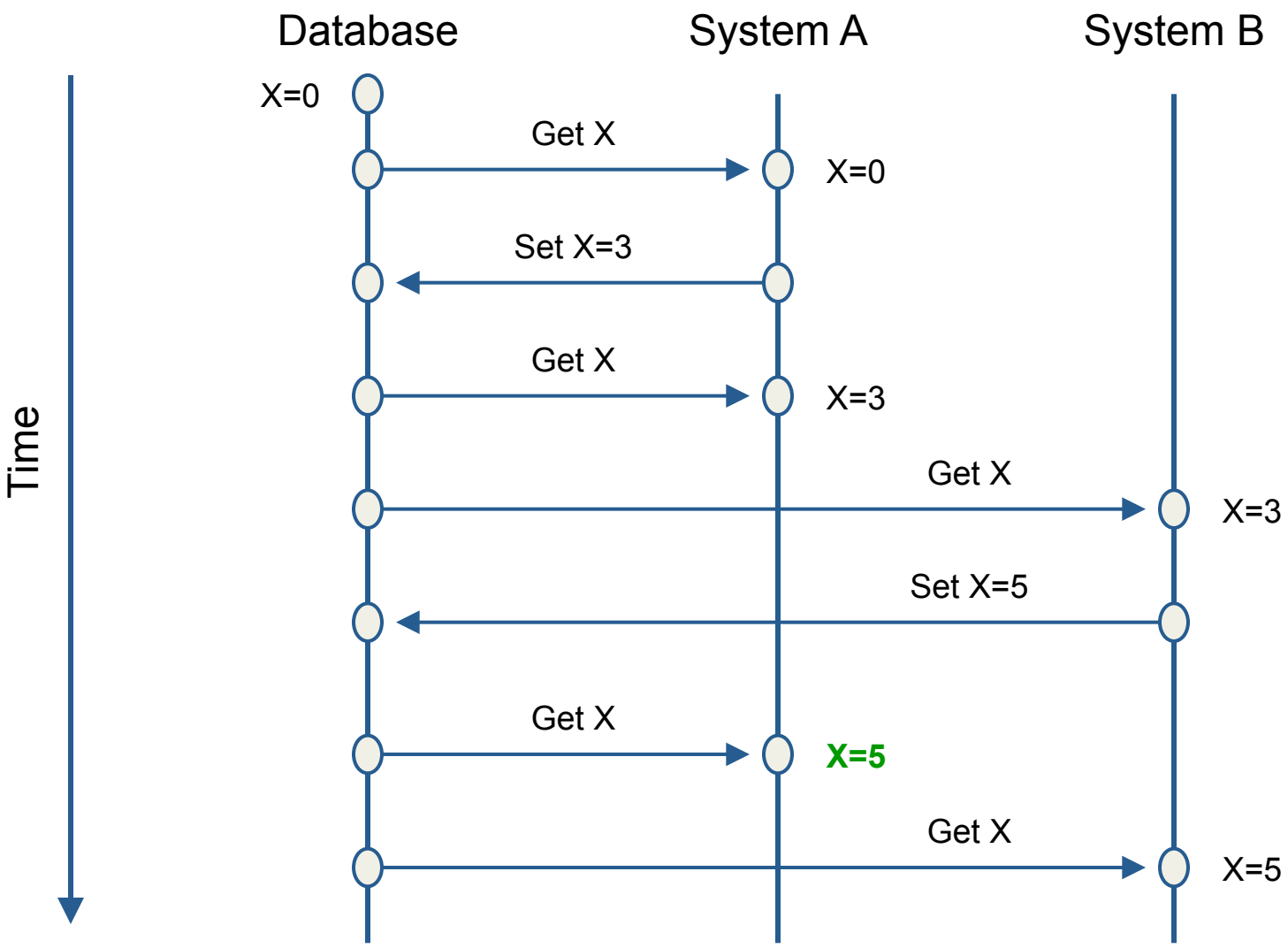
Strong Consistency



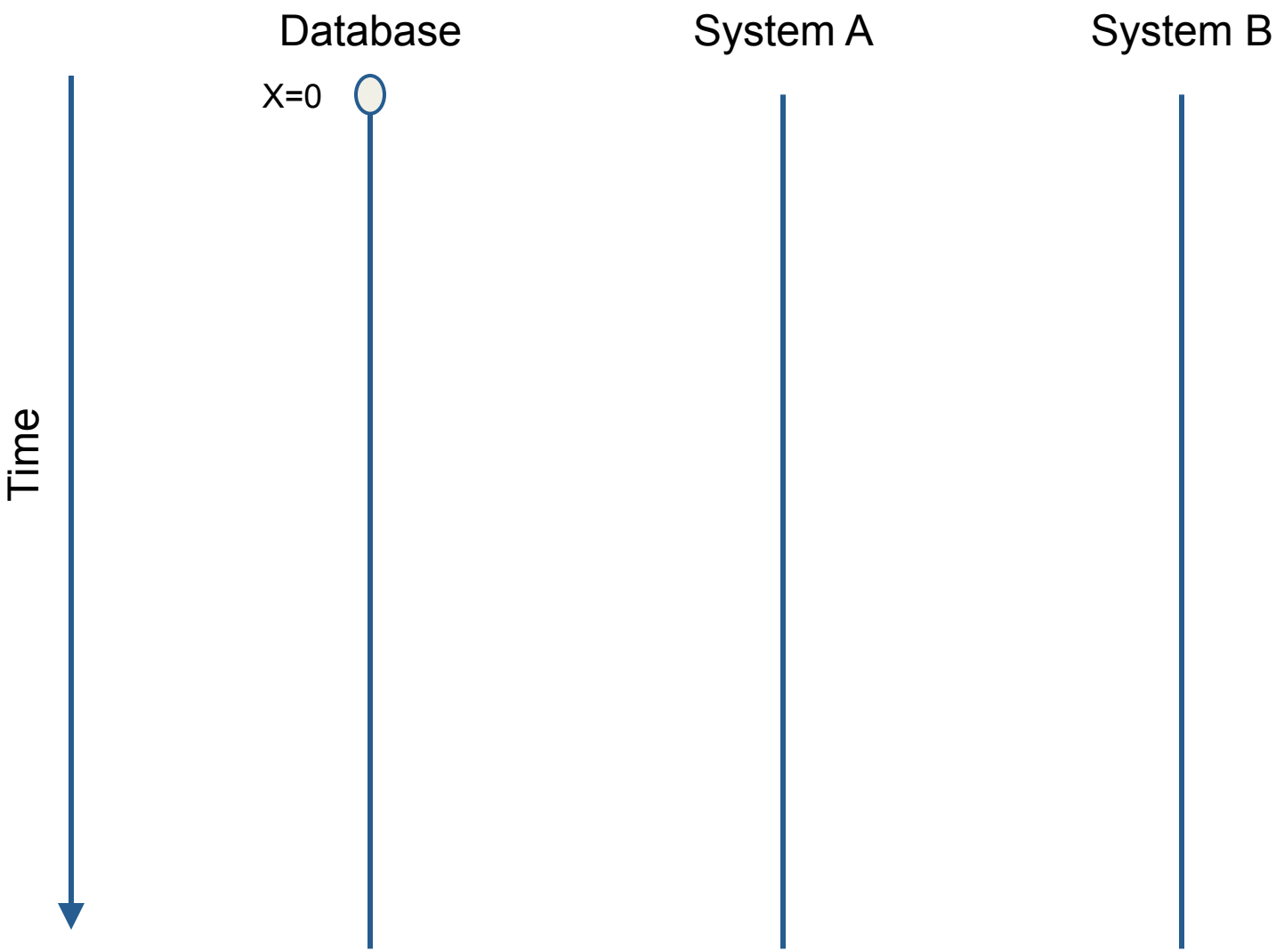
Strong Consistency



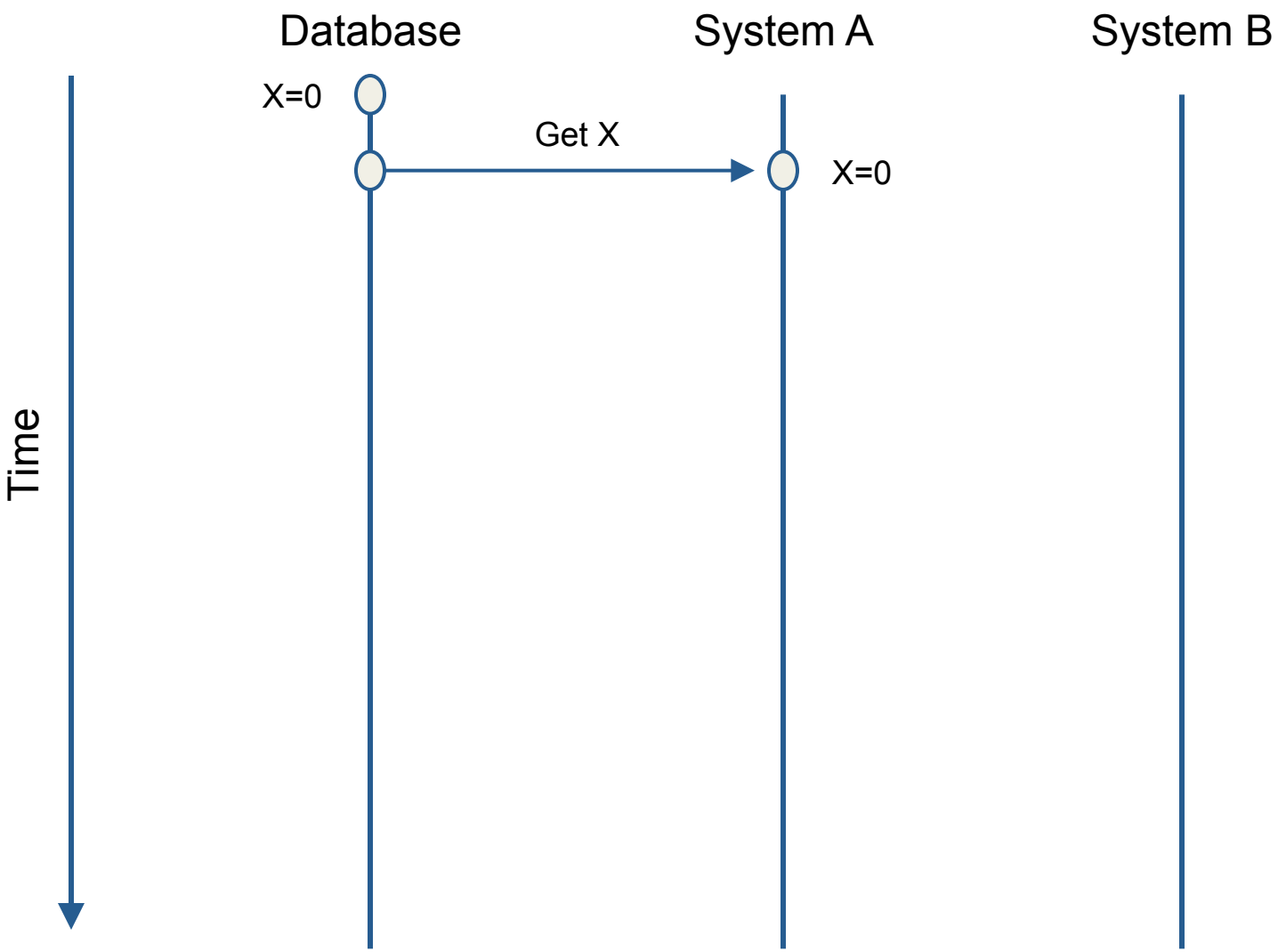
Strong Consistency



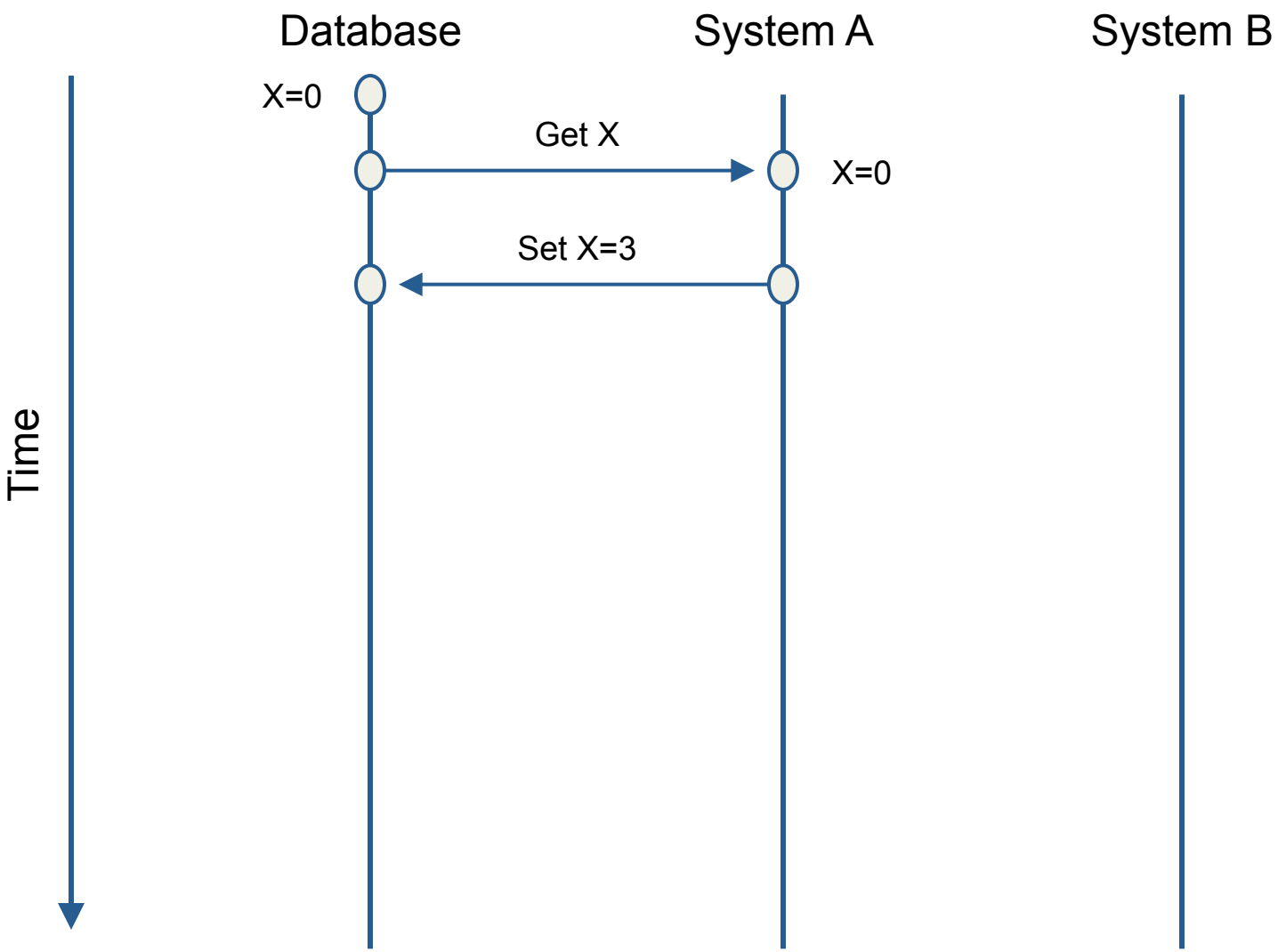
Weak Consistency



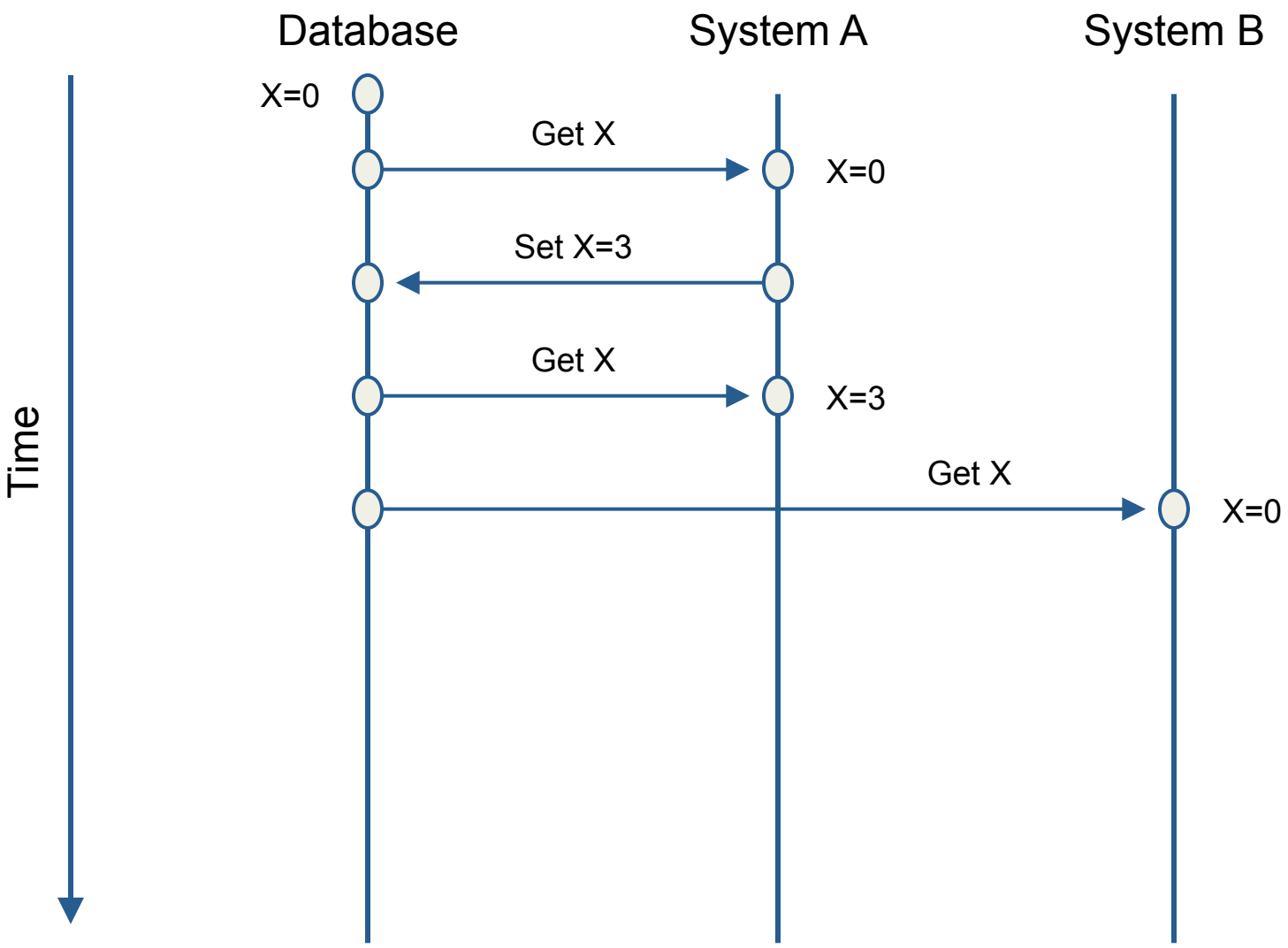
Weak Consistency



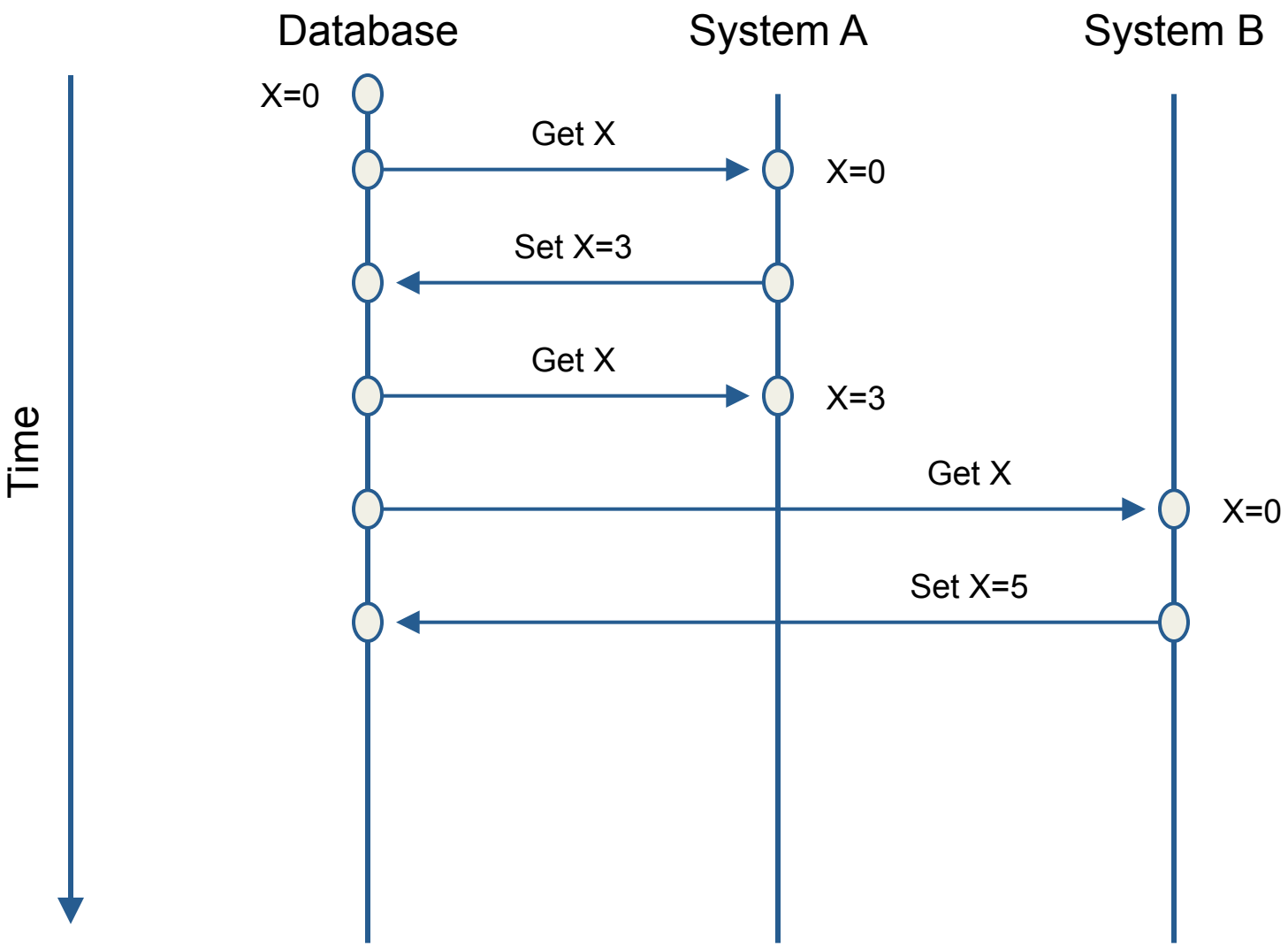
Weak Consistency



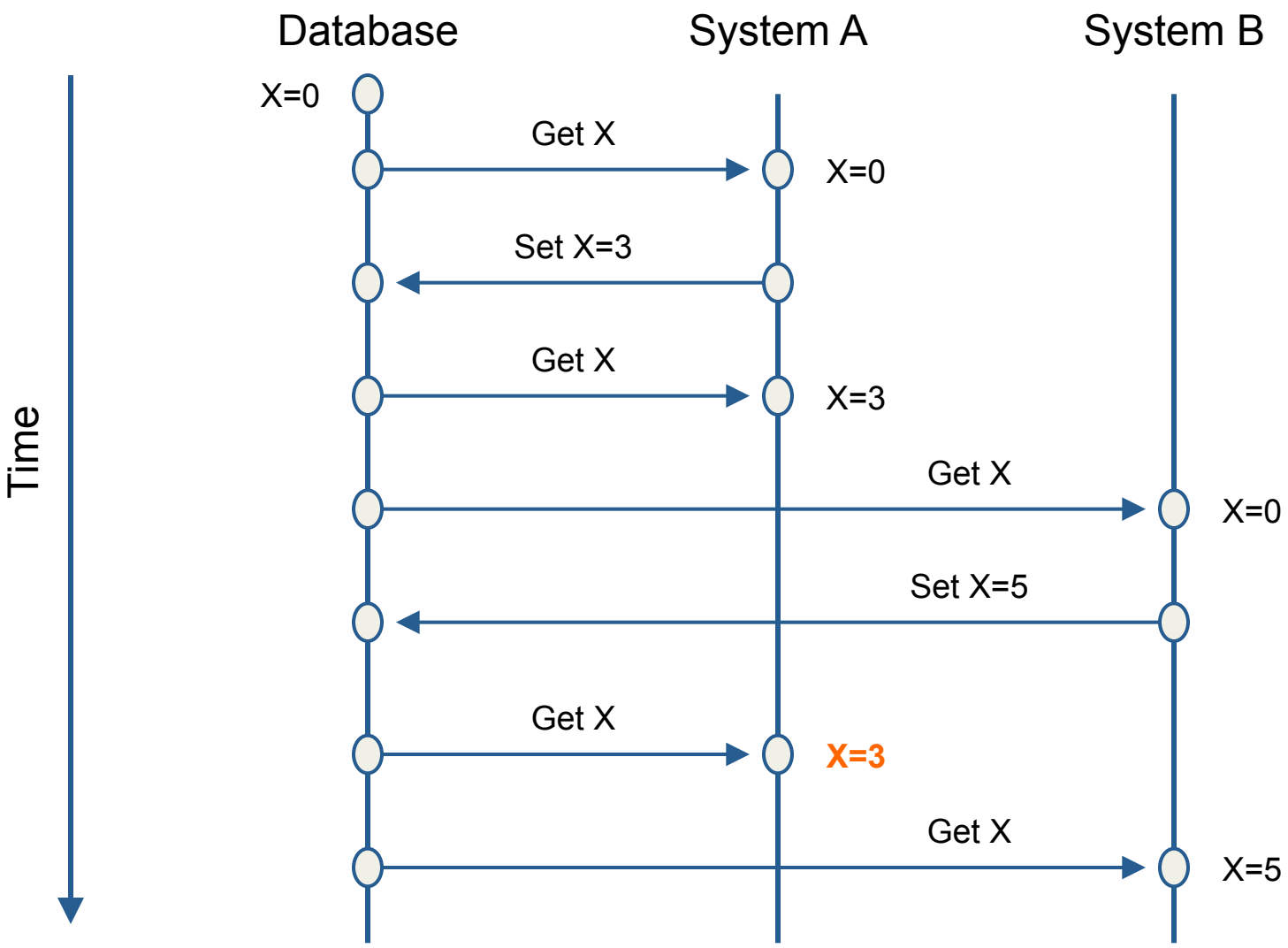
Weak Consistency



Weak Consistency



Weak Consistency



Consistency

- Does it matter if not consistent?
  - Results of search on Amazon….

NoSQL Types

Types

- Column Store
- Key-Value Store
- Document Store
- Graph Store

Column Store

```
<Name> -  <Value>  - <Timestamp>

Andy   -  Boulder  - 140612
John   -  Boston   - 140612
Alice  -  Seattle  - 140613
Andy   -  Denver   - 140614
Stacy  -  Berlin   - 140615
John   -  New York - 140615
Alice  -  Seattle  - 140616
Andy   -  Boulder  - 140615
```

Column Store

Key-Value Store

```
<Key>: <Value>

1675 : “Andy”
1676 : “John”
1677 : “Alice”
1678 : “Stacey”
1679 : “Dave”
1680 : “Rick”
1681 : “Jill”
```

Key-Value Store

redis

FOUNDATIONDB

riak

amazon DynamoDB

Document Store

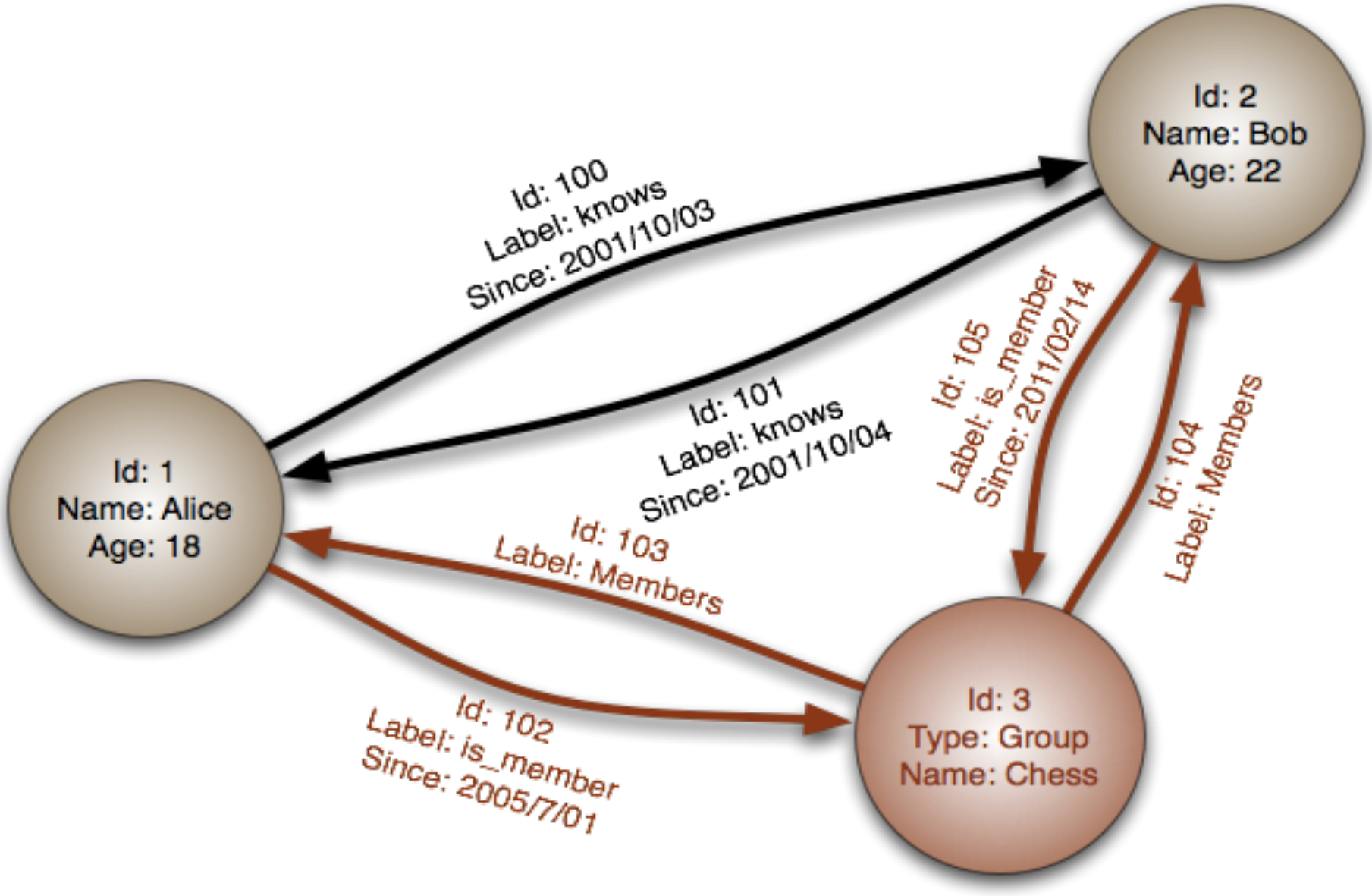
```
<Key>: <Document>

1675 : {Name: “Andy”, Location: “Boulder”, Activity: “School”}
1676 : {Name: “John”, Location: “Boston”}
1677 : {Name: “Alice”, Age: 27, Location: “Seattle”}
1678 : {Name: “Stacey”, Location: “Berlin”, Activity: “Work”}
1679 : {Name: “Dave”, Age: 30, Activity: “Work”}
1680 : {Name: “Rick”, Location: “Denver”}
1681 : {Name: “Jill”, Activity: “Work”}
```

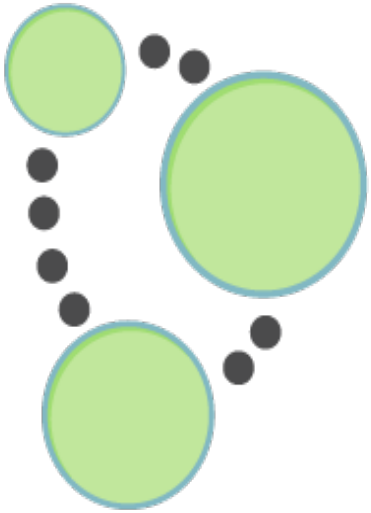
Document Store

Couchbase

Graph Store


Id: 2
Name: Bob
Age: 22
Id: 100
Label: knows
Since: 2001/10/03
Id: 101
Label: knows
Since: 2001/10/04
Id: 105
Label: is_member
Since: 2011/02/14
Id: 104
Label: Members
Id: 1
Name: Alice
Age: 18
Id: 103
Label: Members
Id: 102
Label: is_member
Since: 2005/7/01
Id: 3
Type: Group
Name: Chess

Graph Store



Neo4j

OrientDB®

ArangoDB

When to Use NoSQL

- Simpler Interface
- Loosely Structured Data
- Weaker Consistency
- Higher Performance
- Higher Availability
- Horizontal Scaling

- MongoDB (from "hu**mongo**us")
  - Open-source document database
  - Written in C++
  - Document-Oriented Storage

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the

BSON was designed to have th three characteristics:

1. Lightweight

mongoDB

- Examples SQL vs MongoDB

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM users`	`db.users.find()`

`SELECT id,`
`user_id,`
`status`
`FROM users`
	`db.users.find(`
	`{ },`
	`{ user_id: 1, status: 1 }`
	`)`

SQL SELECT Statements	MongoDB find() Statements
`SELECT *` `FROM users`	`db.users.find()`

`SELECT id,`
`  user_id,`
`  status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1 }`
	`)`

`SELECT user_id, status`
`FROM users`
	`db.users.find(`
	`  { },`
	`  { user_id: 1, status: 1, _id: 0 }`
	`)`

`SELECT *`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`  { status: "A" }`
	`)`

`SELECT user_id, status`
`FROM users`
`WHERE status = "A"`
	`db.users.find(`
	`  { status: "A" },`
	`  { user_id: 1, status: 1, _id: 0 }`
	`)`