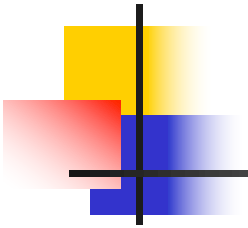
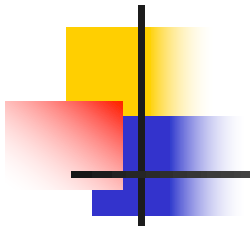
A Quick Introduction to Relational Databases and Object Persistence



Outline

- Relational Databases
- Defining tables
- Structured Query Language
- Relational Database Management Systems
- JDBC
- Object-Relational Mapping
- Object persistence

Some parts based on slides by Bill Howe at Portland State and other sources.



Introduction

- Database
  - a collection of persistent data
- Database Management System (DBMS)
  - a software system that supports creation, population, querying, and administering of a database
- Relational Database Management System
  - DBMS based on a Relational Model, created by Edgar Codd in 1969

Relational Database

- Relational Database (RDB)
  - Consists of a number of *tables* and a single *schema* (*definition* of tables and their attributes)
  - For example:

    Student (sid, name, email, age, gpa)

    Student identifies the table, while

    sid, name, login, email, gpa identify attributes

    sid is the primary key (uniquely identifies a row)

An Example Table

- Student (sid: integer, name: string, email: string, age: integer, gpa: real)

sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	4.0
53832	Guldu	guldu@music	12	3.9

An Example Table

Data type

- Student (sid: integer, name: string, email: string, age: integer, gpa: real)

sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

An Example Table

- Student is a relation on Int x String x String x Int x Float
- One row represents a tuple (related values)

tuple

sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

An Example Table

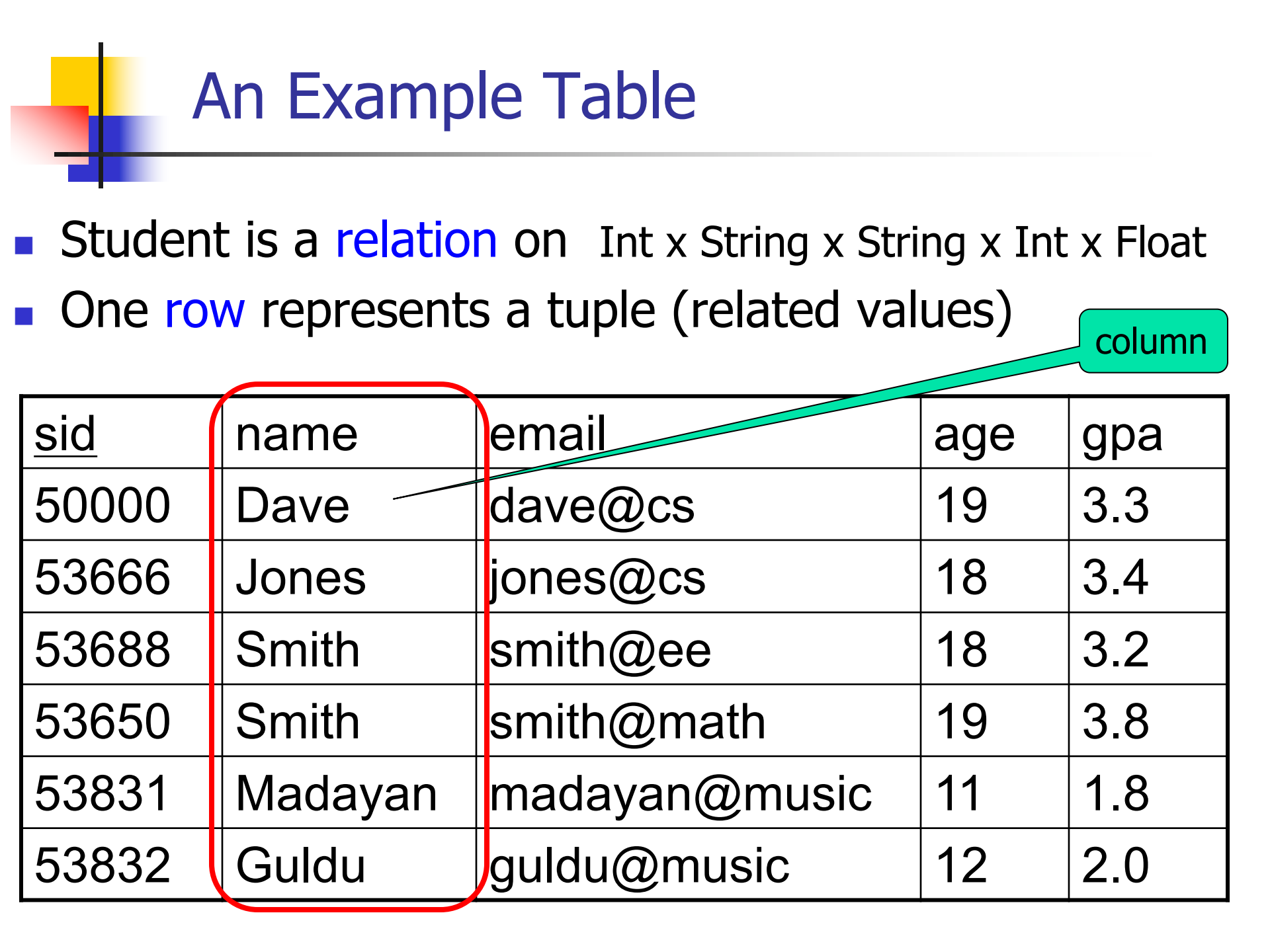
- Student is a relation on Int x String x String x Int x Float
- One row represents a tuple (related values)

attributes

sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

An Example Table

- Student is a relation on Int x String x String x Int x Float
- One row represents a tuple (related values)



sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

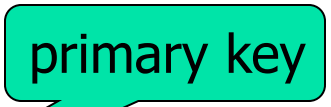
Another example: Courses

- Course (cid, instructor, semester)

cid	instructor	semester
Piano101	Jane	Fall 12
Jazz203	Bob	Sum 12
Calc101	Mary	Spr 12
Hist105	Alice	Fall 12

Keys

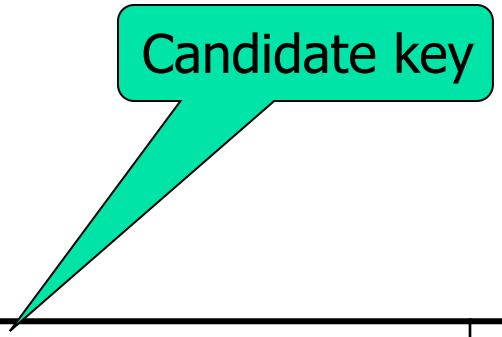
- Primary key – a minimal subset of fields that *uniquely identifies a tuple (row)*
  - sid is primary key for Students
  - cid is primary key for Courses



sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

Keys

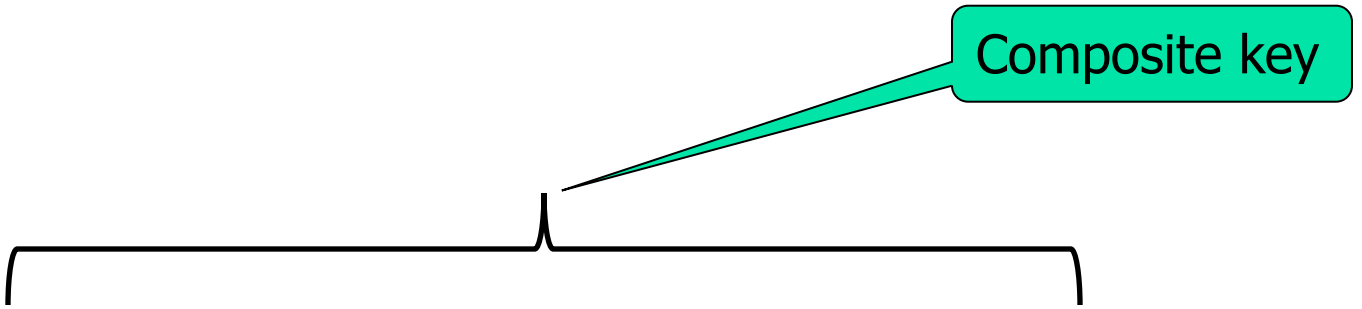
- If we know that email values are unique for all students, email could also be used as a key, which is called a candidate key



sid	name	email	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

Keys

- A (minimum) set of attributes that uniquely identifies tuples may also be used as a candidate key; it is called a composite key



fid	date	seat	fname	lname
DL734	4/5/17	22A	Joe	Smith
DL734	5/19/17	22A	Peggy	Brooks
AA221	5/19/17	22A	Mary	Holcombe

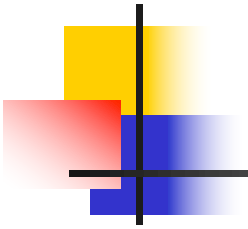


Table relationships

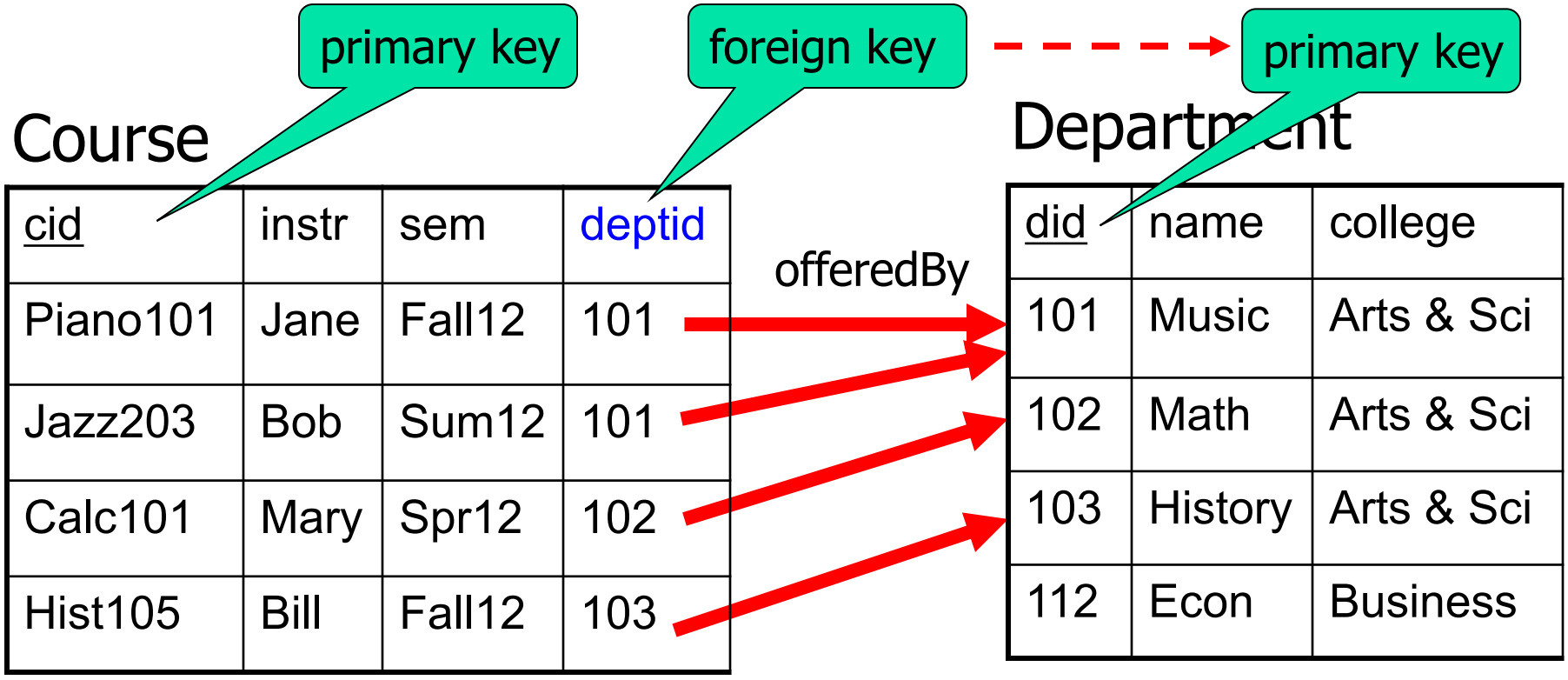
- Tables can be related, representing dependencies among tuples
- For example, a Course is offered by a Department
- A tuple (row) in one table must identify (reference) a related tuple in another table
- This is done with the use of a foreign key
- A foreign key *references* a primary key in the other table

Table relationships

Foreign key – used for relationships between tables

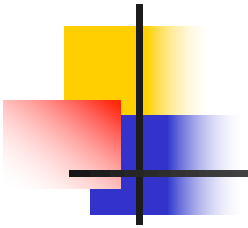
- Course (cid, instructor, quarter, deptid)
- Department (did, name, college)

extra attribute



cid	instr	sem	deptid
Piano101	Jane	Fall12	101
Jazz203	Bob	Sum12	101
Calc101	Mary	Spr12	102
Hist105	Bill	Fall12	103

did	name	college
101	Music	Arts & Sci
102	Math	Arts & Sci
103	History	Arts & Sci
112	Econ	Business



Simple relationships

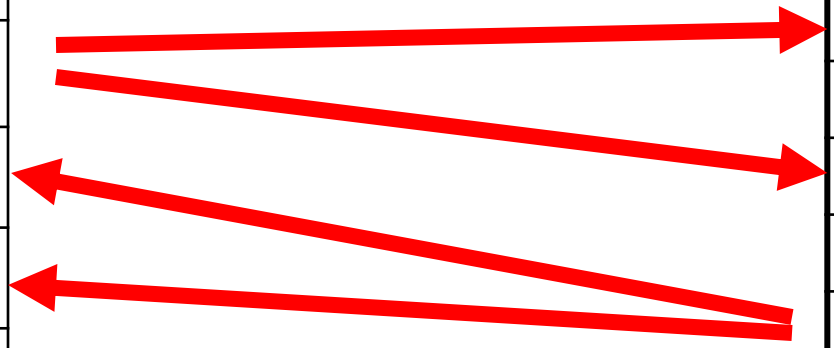
- Previous relationship is called one-to-many (1-m), as it related one department to many courses that department administers (or owns).
- There exist many relations of this type: one mother has many children, one book has many chapters, one list has many elements, etc.
- A one-to-one relationship relates exactly a pair of elements. For example, one student has one student id, one car has one VIN number, etc.

Many-to-many relationships

- Many Students are enrolled in the same Course. Also,
- The same Student may be enrolled in many Courses.

Course

cid	instr	sem
Piano101	Jane	Fall12
Jazz203	Bob	Sum12
Calc101	Mary	Spr12
Hist105	Bill	Fall12



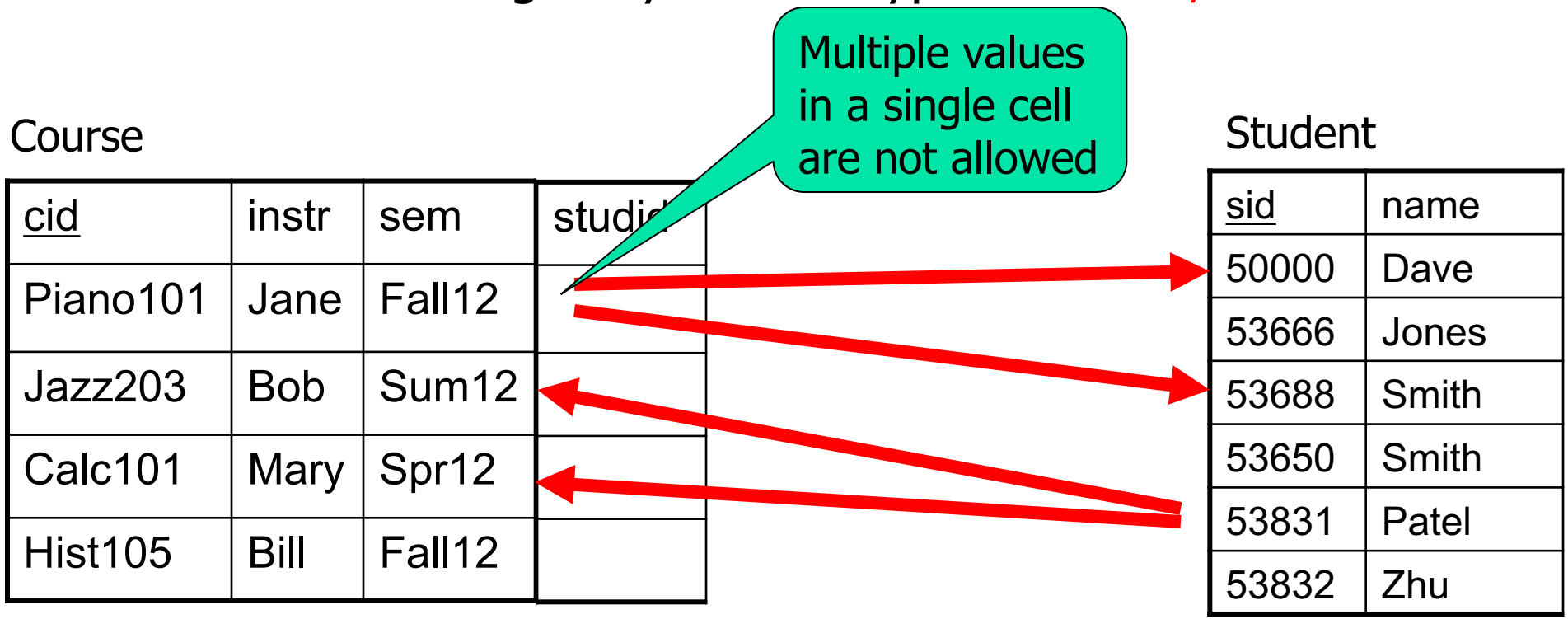
Student

sid	name
50000	Dave
53666	Jones
53688	Smith
53650	Smith
53831	Patel
53832	Zhu

Many-to-many relationships

- Many Students are enrolled in the same Course. Also,
- The same Student may be enrolled in many Courses.

- Can't use foreign keys of this type! One cell, one value!

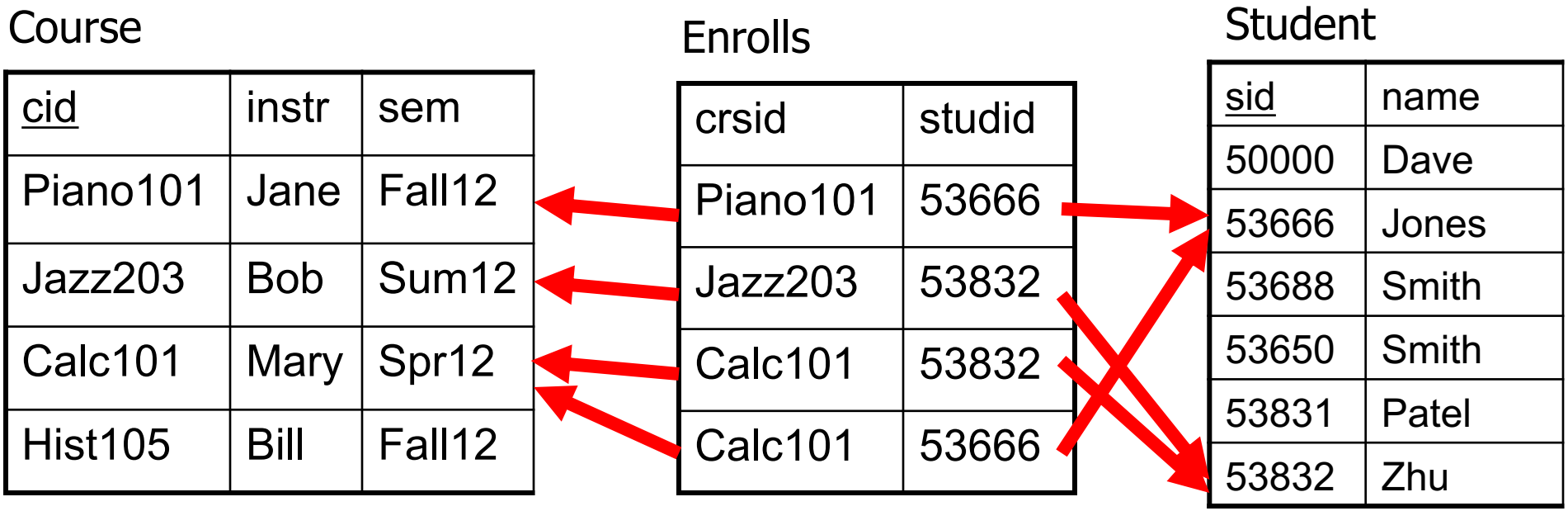


Many-to-many relationships

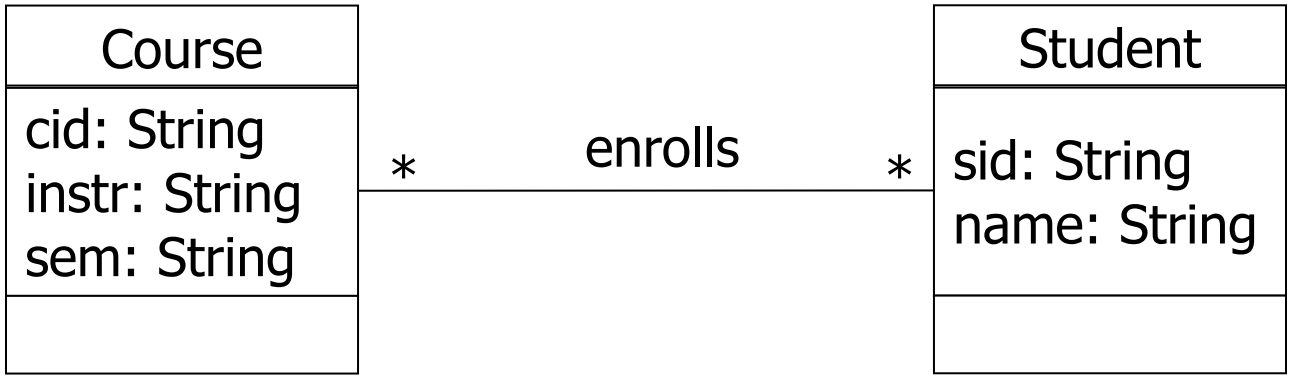
- In general, we need a new table Enrolls(crsid, studid)

  crsid is a foreign key that references cid in the Course table

  studid is a foreign key that references sid in the Student table



Relational tables and UML diagrams



Course

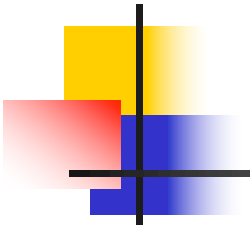
cid	instr	sem
Piano101	Jane	Fall12
Jazz203	Bob	Sum12
Calc101	Mary	Spr12
Hist105	Bill	Fall12

Enrolls

crsid	studid
Piano101	53666
Jazz203	53832
Calc101	53832
Calc101	53666

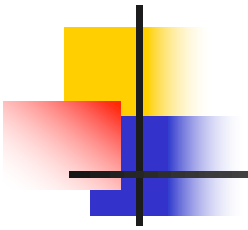
Student

sid	name
50000	Dave
53666	Jones
53688	Smith
53650	Smith
53831	Patel
53832	Zhu



Relational Algebra

- Created by Codd
- Collection of operators for specifying queries
- Query describes step-by-step procedure for computing answer (i.e., operational)
- Each operator accepts one or two relations as input and returns a relation as output
- Relational algebra expression composed of multiple operators

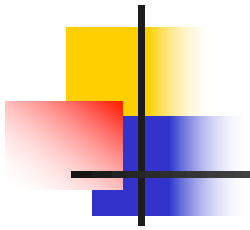


Basic operators

- Selection – return all or some *rows* that meet a given condition
- Projection – return some or all *column* values
- Union
- Cross product
- Difference
- Other operators can be defined in terms of basic operators

We will only *outline* a few of them

Example Schema (simplified)



- Course (cid, instructor, quarter, dept)
- Student (sid, name, gpa)
- Enrolls (cid, grade, studid)

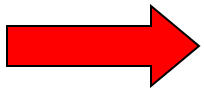
Selection

Find students with gpa higher than 3.3 from S1:

$$\sigma_{gpa>3.3}(S1)$$

S1

sid	name	gpa
50000	Dave	3.3
53666	Jones	3.4
53688	Smith	3.2
53650	Smith	3.8
53831	Madayan	1.8
53832	Guldu	2.0



sid	name	gpa
53666	Jones	3.4
53650	Smith	3.8

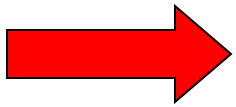
Projection

Find name and gpa of all students in S1:

$\Pi_{name,\ gpa}(S1)$

S1

Sid	name	gpa
50000	Dave	3.3
53666	Jones	3.4
53688	Smith	3.2
53650	Smith	3.8
53831	Madayar	1.8
53832	Guldu	2.0



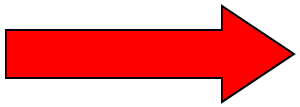
name	gpa
Dave	3.3
Jones	3.4
Smith	3.2
Smith	3.8
Madayan	1.8
Guldu	2.0

Combine Selection and Projection

Find name and gpa of students with gpa > 3.3 in S1:

$$\Pi_{name,gpa}(\sigma_{gpa>3.3}(S1))$$

Sid	name	gpa
50000	Dave	3.3
53666	Jones	3.4
53688	Smith	3.2
53650	Smith	3.8
53831	Madayan	1.8
53832	Guldu	2.0



name	gpa
Jones	3.4
Smith	3.8

Joins

- Combine information from two or more tables using a natural join
- Example: find which departments own courses:

$C1 \bowtie_{C1.deptid = D.did} D$

C1

cid	instr	sem	deptid
Piano101	Jane	Fall12	101
Jazz203	Bob	Sum12	101
Calc101	Mary	Spr12	102
Hist105	Bill	Fall12	103

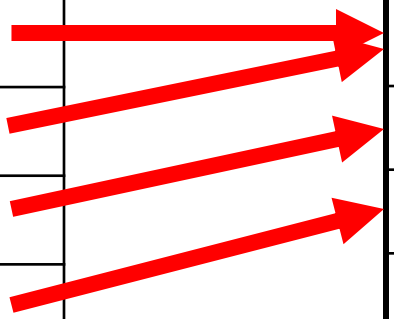
D

did	name	college
101	Music	Arts & Sci
102	Math	Arts & Sci
103	History	Arts & Sci
112	Econ	Business

Joins

C1

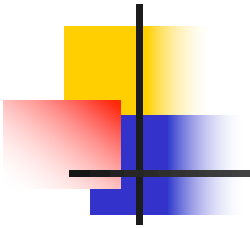
cid	instr	sem	deptid
Piano101	Jane	Fall12	101
Jazz203	Bob	Sum12	101
Calc101	Mary	Spr12	102
Hist105	Bill	Fall12	103



D

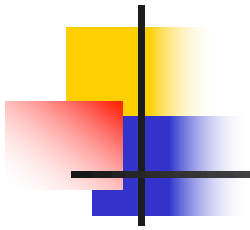
did	name	college
101	Music	Arts & Sci
102	Math	Arts & Sci
103	History	Arts & Sci
112	Econ	Business

cid	instr	sem	deptid	did	name	college
Piano101	Jane	Fall12	101	101	Music	Arts & Sci
Jazz203	Bob	Sum12	101	101	Music	Arts & Sci
Calc101	Mary	Spr12	102	102	Math	Arts & Sci
Hist105	Bill	Fall12	103	103	History	Arts & Sci



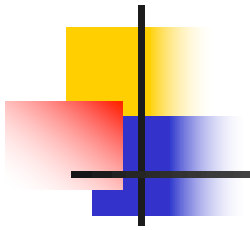
History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
- IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.



History of SQL

- In late 70s, ORACLE was introduced as (likely) the first commercial RDBMS based on SQL
- In 1987, ANSI and ISO published an initial standard for SQL
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92
- In 1999, SQL3 was released with support for object-oriented data management



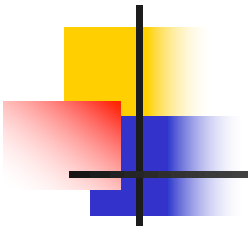
Objectives of SQL

- SQL includes two major components:
  - A Data Definition Language (DDL) for defining a database structure

    Create table, create indexes, alter table, etc.
  - A Data Manipulation Language (DML) for retrieving and updating data

    Select rows from one or more tables (using joins), insert, update, or delete rows

Good tutorial: https://www.w3schools.com/sql/default.asp



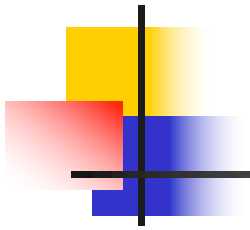
Intro to SQL

Data Definition Language

- CREATE TABLE
  - Create a new table, e.g., students, courses, enrolled
- Also, ALTER TABLE, DROP TABLE, and other statements

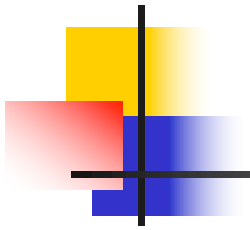
Data Manipulation Language

- SELECT-FROM-WHERE
  - For example, retrieve all CS courses
- INSERT
  - E.g., store new students, enroll students in courses
- UPDATE
  - E.g., update data on students, change student enrollments
- DELETE
  - E.g., delete students or student enrollments



Create Table

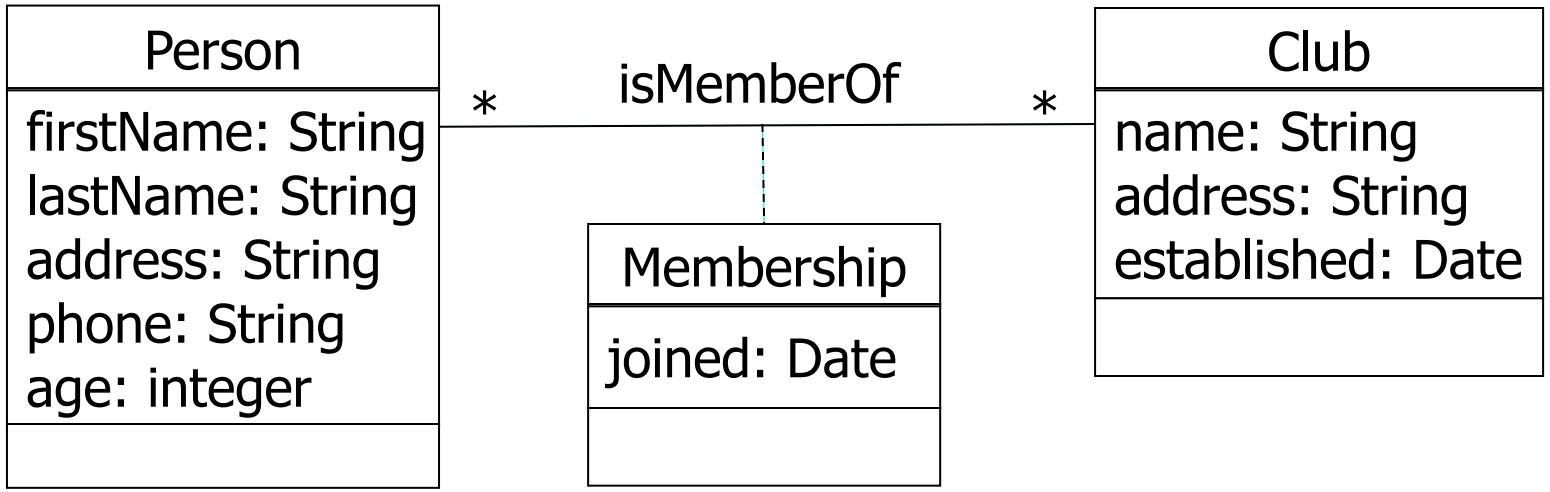
CREATE TABLE *tableName*
{(*colName dataType* [NOT NULL] [UNIQUE]
[DEFAULT *defaultOption*]
[CHECK *searchCondition*] [,...]}
[PRIMARY KEY (*listOfColumns*),]
{[UNIQUE (*listOfColumns*),] [...,]}
{[FOREIGN KEY (*listOfFKColumns*)
REFERENCES *parentTableName* [(*listOfCKColumns*)],
[ON UPDATE *referentialAction*]
[ON DELETE *referentialAction*]] [,...]}
{[CHECK (*searchCondition*)] [,...] })



Create Table

- Creates a table with one or more columns of the specified *dataType*.
- With NOT NULL, the system rejects any attempt to insert a NULL value in the column.
- Can specify a DEFAULT value for a column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action

Create Table




```
create table  person  (
    id              int             unsigned primary key,
    firstname       varchar(255)    not null,
    lastname        varchar(255)    not null,
    address         varchar(255),
    phone           varchar(255),
    age             int unsigned
);
```

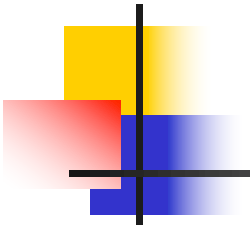
Create Table

```
create table  club  (
  id                    int                   unsigned primary key,
  name                  varchar(255)          not null,
  address               varchar(255),
  established           datetime
);


create table  membership  (
  id                    int                   unsigned primary key,
  personid              int unsigned          not null,
  clubid                int unsigned          not null,
  joined                datetime,

  foreign key    (personid)  references person(id),
  foreign key       (clubid)     references club(id)
);
```

SELECT Statement

SELECT [DISTINCT | ALL]
{* | [*columnExpression* [AS *newName*]] [,...] }
FROM *tableName* [*alias*] [, ...]
[WHERE *condition*]
[GROUP BY *columnList*] [HAVING *condition*]
[ORDER BY *columnList*]

- Order of the clauses cannot be changed
- Only SELECT and FROM are mandatory



SELECT Statement

FROM	Specifies table(s) to be used.
WHERE	Filters rows.
GROUP BY	Forms groups of rows with same column value.
HAVING	Filters groups subject to some condition.
SELECT	Specifies which columns are to appear in output.
ORDER BY	Specifies the order of the output.

Select-From-Where query

"Find everything you know about all clubs"

```
select * from club
```

The * above means "get all column values"

"Find everything you know about all persons who are under 18"

```
select * from person p where p.age < 18
```

Select-From-Where query

"Find everything you know about all clubs"

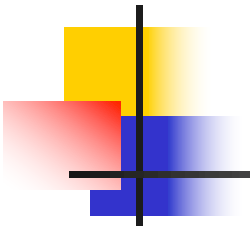
```
select  *  from  club
```

The * above means "get all column values"

"Find everything you know about all persons who are under 18"

```
select  *
from     person p
where   p.age < 18
```

An SQL query can be formatted using white space, tabs and newlines.



Select-From-Where query

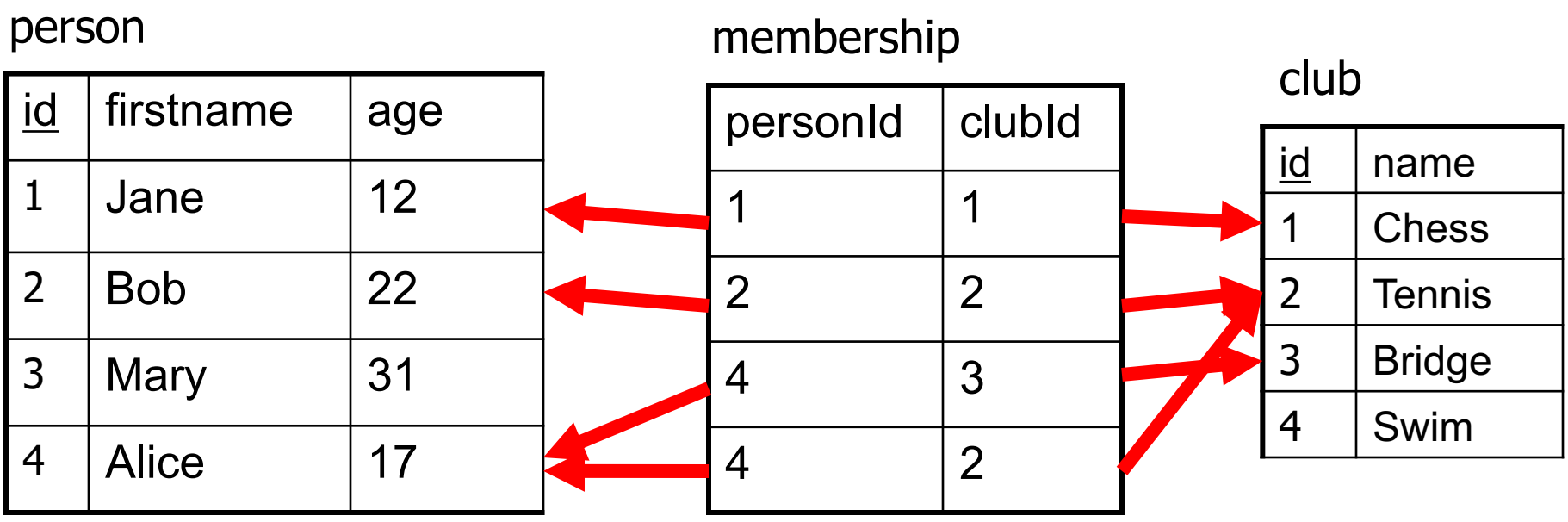
"Find names of all persons who are under 18"

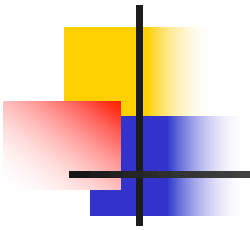
```
select   p.firstname
from     person p
where    p.age < 18
```

The above query performs a selection and projection

Queries across multiple tables (joins)

"Print names and address of persons younger than 20 who are members of the Tennis club"





Queries across multiple tables (joins)

"Print names and address of persons younger than 20 who are members of the Tennis club"

```
select   p.firstname,  p.address
from     person p, club c, membership m
where    p.age < 20  and  c.name = 'tennis'
           and  m.personid = p.id
           and  m.clubid = c.id
```

Queries across multiple tables (joins)

"Print names and address of persons younger than 20 who are members of the Tennis club"

```
select   p.firstname,  p.address
from     person p, club c, membership m
where    p.age < 20  and  c.name = 'tennis'
           and  m.personid = p.id
           and  m.clubid = c.id
```

an alias

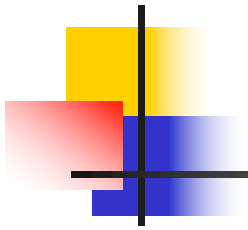
join clauses

Queries across multiple tables (joins)

"Print names and address of persons younger than 20 who are members of the Tennis club"

```
select * from membership
inner join person on membership.personid = person.id
inner join club on membership.clubid = club.id;
```

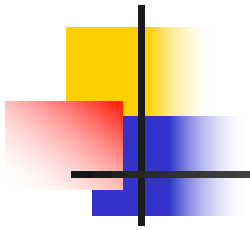
The above select used the **join clause**; here, it is an `inner join` (you can use just `join`)



INSERT

INSERT INTO *tableName* [(*columnList*)]
VALUES (*dataValueList*)

- A new row is inserted, where the columns are assigned listed data values, pairwise
- dataValueList must match columnList as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in dataValueList must be compatible with data type of corresponding column

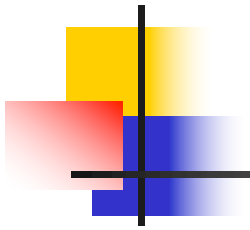


INSERT

INSERT INTO *tableName* [(*columnList*)]
VALUES (*dataValueList*)

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order
  - any columns omitted must have been declared as NULL when the table was created, unless DEFAULT was specified when creating a column

insert into person (firstname, lastname, address, phone, age)
values ('Jeff', 'Roberts', '11 Oak St', '123-444-5566', 24)

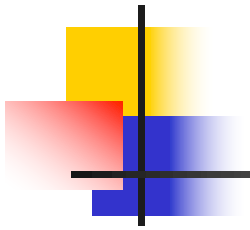


UPDATE

UPDATE *tableName*
SET *columnName1* = *dataValue1*
[, *columnName2* = *dataValue2* ...]
[WHERE *searchCondition*]

Rows are updated with the given column values

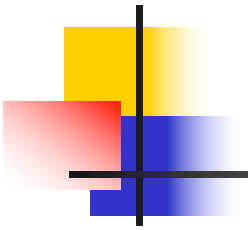
- *tableName* can be name of a base table or an updatable view
- The SET clause specifies names of one or more columns that are to be updated



UPDATE

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy searchCondition are updated.
- New dataValue(s) must be compatible with data type for corresponding column

```
update person set firstname = 'Mark'
where lastname = 'Roberts'
```



DELETE

DELETE FROM *tableName*
[WHERE *searchCondition*]

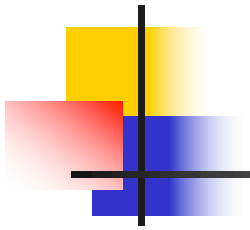
Rows are deleted

- *searchCondition* is optional; if omitted, all rows are deleted from the table. This does not delete the table itself.
- If *searchCondition* is specified, only those rows that satisfy condition are deleted.

```
delete from person
where lastname = 'Roberts'
```

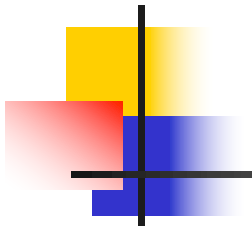
Examples of other SQL statements

```
insert into club (name, address, established)
values ( 'Chess', '33 Leaf St., Blossom, OR. 88888',
                  '2007-07-12 12:00:00' )

update club
set address = ’11 Trunk St., Blossom, OR. 77777’
where name = 'Chess’

delete from club
where name = 'Chess’
```



Other SQL features

- MIN, MAX, AVG
  - Find highest grade in fall database course
- COUNT, DISTINCT
  - How many students enrolled in CS courses in the fall?
- ORDER BY, GROUP BY
  - Rank students by their grade in fall database course
- Transactions



Relational DBMS

Examples of commercial RDBMS systems:

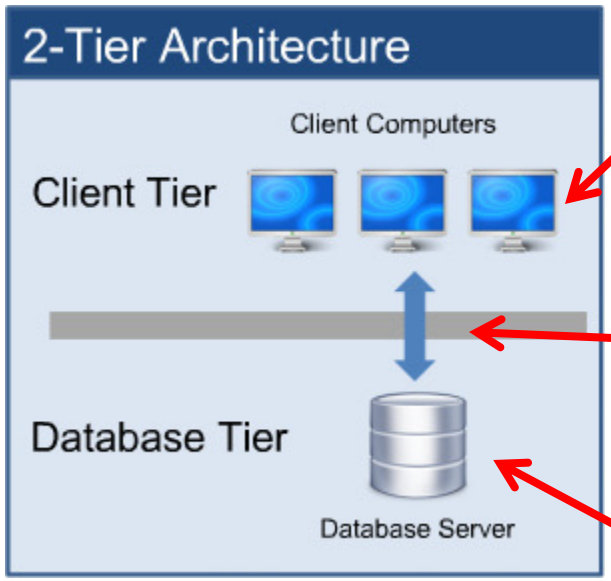
- ORACLE
- DB2 (IBM)
- SQL Server (Microsoft)

Examples of open-source RDBMS systems:

- MySQL
- PostgreSQL
- SQLite *

* SQLite is not a complete implementation of SQL

Relational DBMS

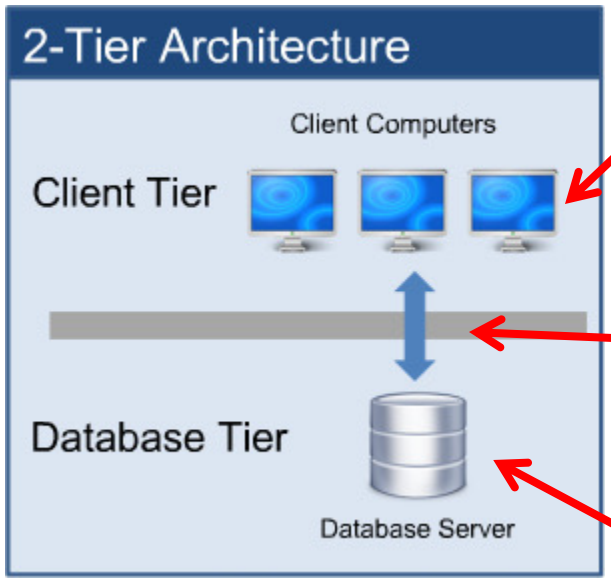


Client program can be in any language, but must "speak" the RDBMS communication language, usually over the network

Communication language/protocol or an API

RDBMS server, which accepts request over the network

Relational DBMS



Java client

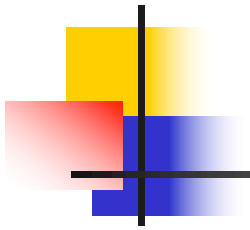
C++ client

JDBC

ODBC

RDBMS with drivers

ODBC/JDBC



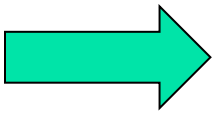
Object persistence

- Java (and C++, etc.) objects may be stored in and retrieved from a relational database, such as MySQL or SQLite
- **Object-Relational Mapping (ORM)**
  - Classes are mapped onto tables
  - Associations onto foreign keys and/or relation tables
  - A newly created object is inserted into its class's table by storing its state as a single row with its table's attribute values
  - An existing object can be restored by retrieving its state from the database and creating a new instance initialized to the values retrieved from the database

Object persistence

- Classes are mapped onto tables

Club
name: String address: String established: Date
op(arg:int): long



Club varchar(255) datetime

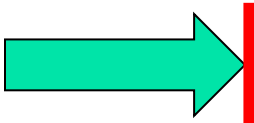
id	name	address	established
...	...	...	...

- Preserve the attribute names and select suitable types from SQL
- Define a column to serve as a key (automatically generated)
- Set constraints, if needed (unique, enum-like values, ...)
- Operations are not represented

Object persistence

- Objects are represented as rows

: Club
name = "Chess" address = "11 Oak St." established = 4/22/2004



Club	varchar(255)		datetime
id	name	address	established
1	Chess	11 Oak St.	4/22/2004
...	...	...	...

- Identifier (primary key) is automatically generated
- Java/C++ data values are automatically converted
- Beware of dates:

  java.util.date is not quite the same as java.sql.date !

Object persistence

```
class Person {
  String name;
  String address;
  int age;
  Person() {...}
  ...
}
```

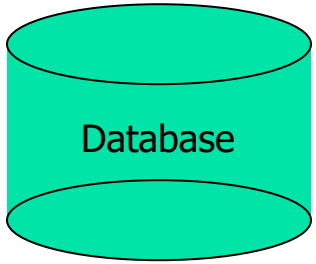
Java program A

object

: Person
name="Smith" address="Athens" age=21

pid = store(object)

pid == 111



person table

pid	name	address	age
111	Smith	Athens	21

Java program B

restored object

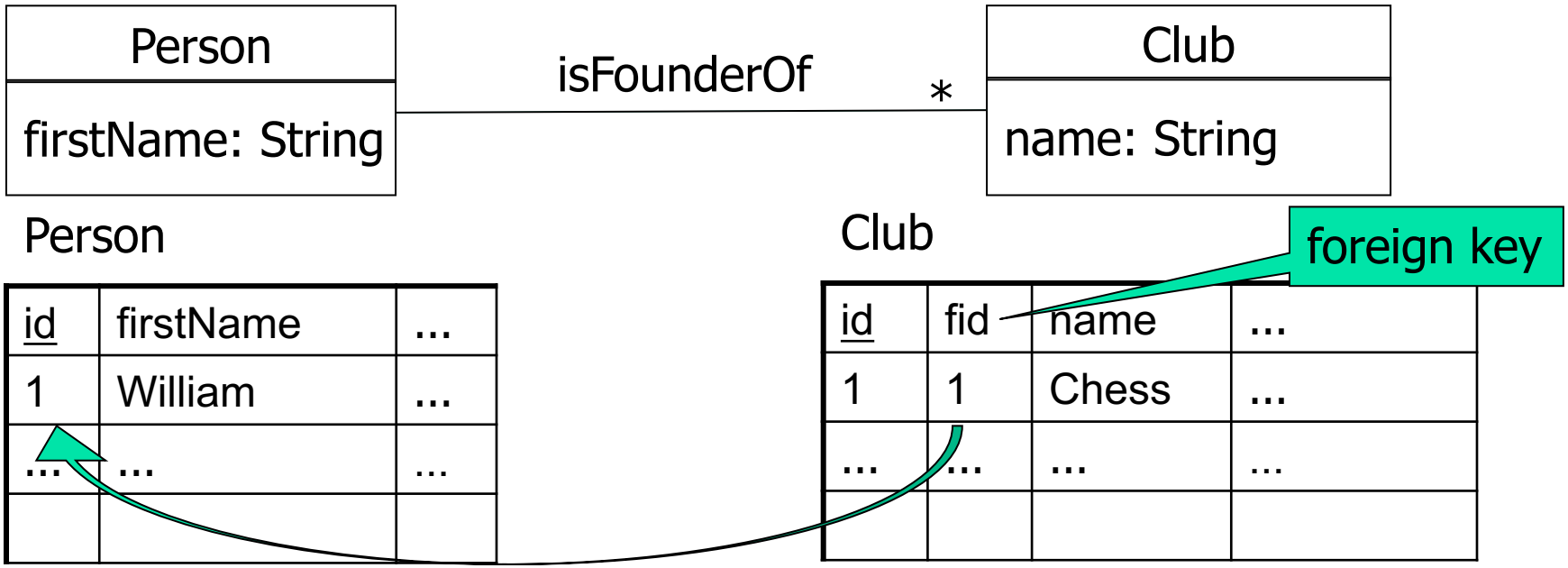
: Person
name="Smith" address="Athens" age=21

object = restore(pid)

pid == 111

Object persistence

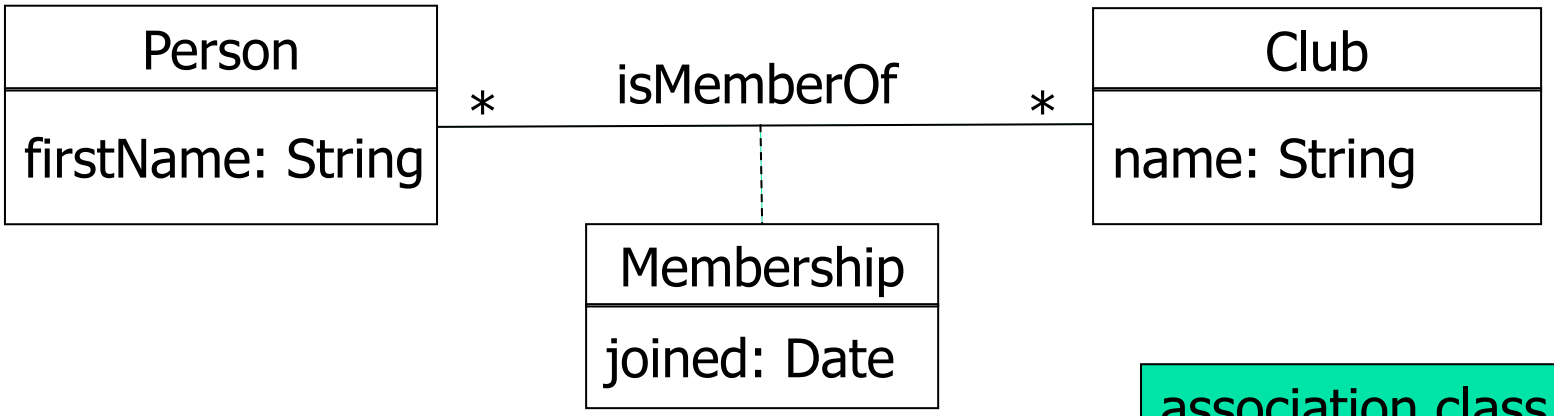
- 1-1 and 1-m associations are mapped onto foreign keys



- Remember, this works for 1-1, 1-optional, and 1-m associations!
- Only one table should have the foreign key defined (for 1-m, it should be the table on the "many" side)

Object persistence

- m-m associations and association classes are mapped onto relation tables



association class table

Person

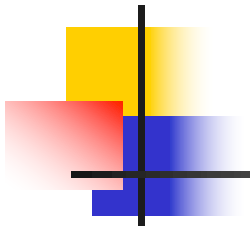
id	fIrstName	...
1	Jane	...

Membership

pId	cid	joined
1	1	3/4/2011

Club

id	name	...
1	Chess	...

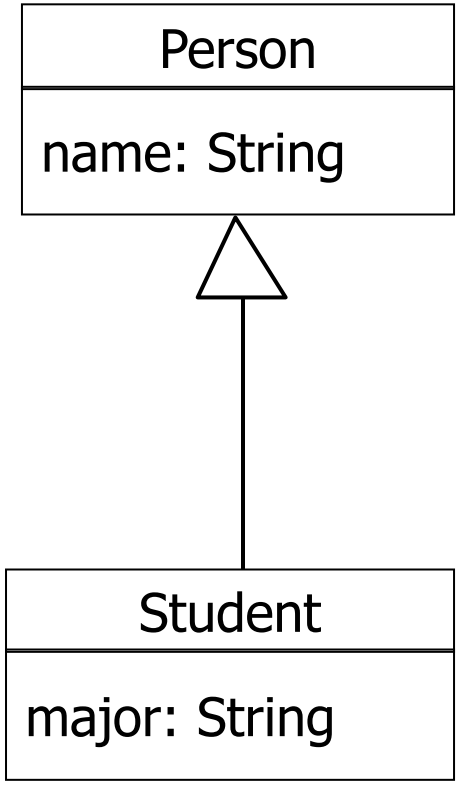


Object persistence

- Mapping of generalization (inheritance) relationships is more involved
- Method 1:
  - Parent and Child classes are mapped onto their own tables
  - The Child class table has a foreign key to the Parent table
  - A Child class object is represented partially in the Child table (Child class attributes) and partially in the Parent class (Parent class attributes)
  - Retrieval of a complete Child class object requires a join SQL statement to retrieve all attributes; the foreign key value connects the two parts
  - Retrieval of objects in a hierarchy requires a left outer join of parent and child
  - With large/deep hierarchies this method may be inefficient

Object persistence

- Generalization mapping, method 1



Person

id	type	name	...
3	student	William	...
4	person	Mary	...
...		...	...

Student

id	parentId	major	...
1	3	Math	...
...	...	...	...

foreign key to parent

Object persistence

- Retrieving objects in method 1

```
select * from
Person INNER JOIN Student on
    Person.id = Student.parentId;
```

to retrieve just Student objects

OR

```
select * from
Person LEFT OUTER JOIN Student on
    Person.id = Student.parentId;
```

to retrieve both Person **and** Student objects

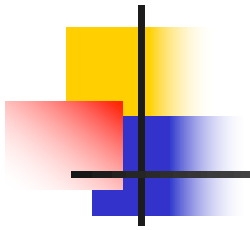
a 'type tag' used to distinguish objects of specific classes

Person

id	type	name	...
3	student	William	...
4	person	Mary	...
...		...	...

Student

id	parentId	major	...
1	3	Math	...
...	...	...	...

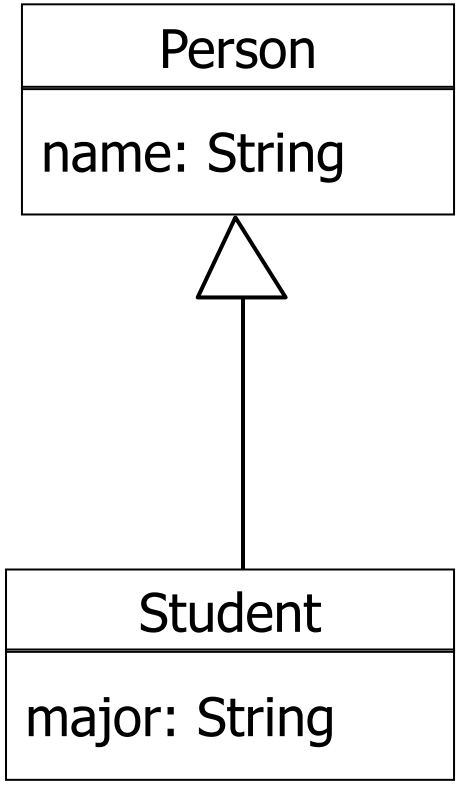


Object persistence

- Method 2:
  - Parent and Child classes are mapped onto a single table, which includes all attributes (from Parent and Child classes)
  - A Parent class object uses only the Parent's columns; the Child columns are wasted
  - A Child class object uses all columns (its columns and the Parent's columns)
  - Mapping a class hierarchy requires creating a table with the union of all attributes and some wasted space is unavoidable

Object persistence

- Generalization mapping, method 2



PersonHierarchy

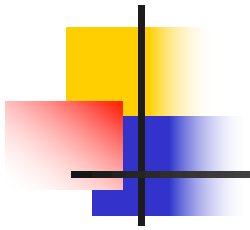
				Person attrs		Student attrs
id	type	name	...	major	...	
1	student	William	...	Math	...	
2	person	Bill	...	null	null	
...	...	...	...	...	...	

select * from PersonHierarchy where type = 'student'
to retrieve just Student objects

OR

select * from PersonHierarchy
to retrieve both Person and Student objects

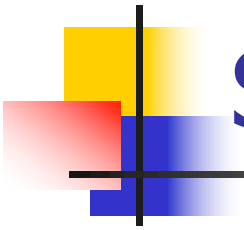
Object persistence



- A persistence middleware system may be used to store and retrieve objects from an RDBMS
- Example: Hibernate (from RedHat)

  http://www.hibernate.org/

  It is a framework for mapping an object-oriented domain model to a relational database
  - Mapping is placed in an XML file
  - Classes and relationships (1-m, m-n) are mapped onto relational tables
  - Objects are stored and retrieved "seamlessly"
- Another such framework is ROOM, popular in Android apps.



Summary: Why are RDBMS useful?

- Data independence – provides abstract view of the data, without details of storage
- Efficient data access – uses techniques to store and retrieve data efficiently
- Reduced application development time – many important functions already supported
- Centralized data administration
- Data Integrity and Security
- Concurrency control and recovery