CSCI 4050/6050
Software Engineering

Introduction to
Relational Databases
and
Object-Relational Mapping

Dr. Eman Saleh eman.saleh@uga.edu

Outline

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

Slides by Prof Kris Kouchut: In part, 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, login, age, gpa)

    Student identifies the table, while

    sid, name, login, age, gpa identify attributes

    sid is the primary key

An Example Table

- Student (*<u>sid</u>*: integer, *name*: string, *login*: string, *age*: integer, *gpa*: real)

<u>sid</u>	name	login	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

Data types

- Student (*<u>sid</u>*: integer, *name*: string, *login*: string, *age*: integer, *gpa*: real)

<u>sid</u>	name	login	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
- Row represents a tuple (related values)

attributes

sid	name	login	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
- Row represents a tuple (related values)

tuple

sid	name	login	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
- Row represents a tuple (related values)

column

sid	name	login	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 – minimal subset of fields that *uniquely identifies a tuple*
  - sid is primary key for Students
  - cid is primary key for Courses

primary key

sid	name	login	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 login values are unique for all students, login could also be used as a key, which is called a candidate key

Candidate key

sid	name	login	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

Composite key

fid	date	seat	fname	lname
734	4/5/11	22A	Joe	Smith
734	5/19/11	22A	Peggy	Brooks
734	5/20/11	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

primary key

foreign key

Course

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

offeredBy

Department

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

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

Course

cid	instr	sem
Piano101	Jane	Fall19
Jazz203	Bob	Sum19
Calc101	Mary	Spr19
Hist105	Alice	Fall19

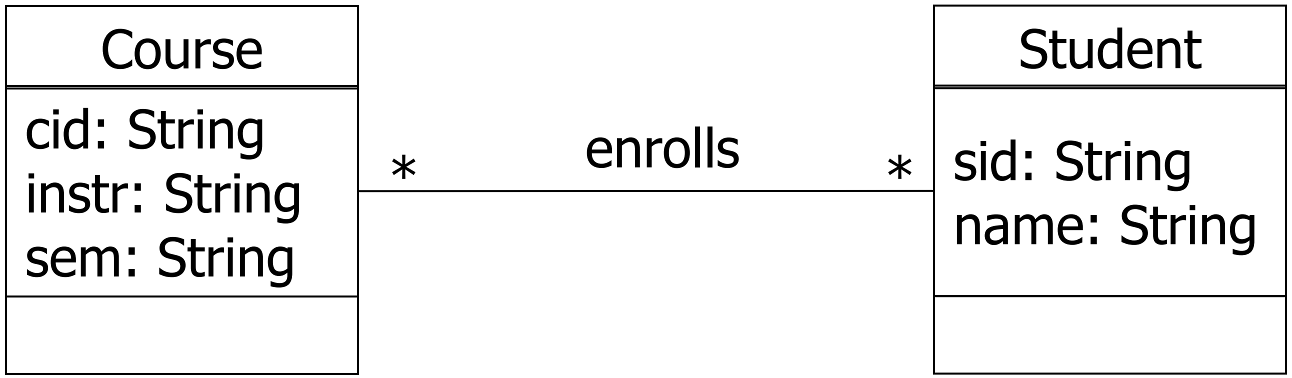
Enrolls

crsid	studid
Piano101	53666
Jazz203	53832
Calc101	53832
Calc101	53666

Student

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

Relational tables and class diagrams



Course

<u>cid</u>	instr	sem
Piano101	Jane	Fall19
Jazz203	Bob	Sum19
Calc101	Mary	Spr19
Hist105	Alice	Fall19

Enrolls

crsid	studid
Piano101	53666
Jazz203	53832
Calc101	53832
Calc101	53666

Student

<u>sid</u>	name
50000	Dave
53666	Jones
53688	Smith
53650	Smith
53831	Patel
53832	Zak

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 *rows* that meet some condition
- Projection – return *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

Select 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

Project name and gpa of all students in S1:

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

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

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

Combine Selection and Projection

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

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

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
- Example: courses owned by departments:

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

C1

<u>cid</u>	instr	sem	deptid
Piano101	Jane	Fall12	101
Jazz203	Bob	Sum12	101
Calc101	Mary	Spr12	102
Hist105	Alice	Fall12	103

D

<u>did</u>	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	Alice	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	Alice	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

    Creata 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

Intro to SQL

- CREATE TABLE
  - Create a new table, e.g., students, courses, enrolled
- Also, ALTER TABLE, DROP TABLE, and other statements
- CREATE INDEX and DROP INDEX
- SELECT-FROM-WHERE
  - List all CS courses
- INSERT
  - Add new students, courses, or enroll students in courses
- UPDATE
  - Update attributes of students, courses, or change student enrollments
- DELETE
  - Delete students or courses

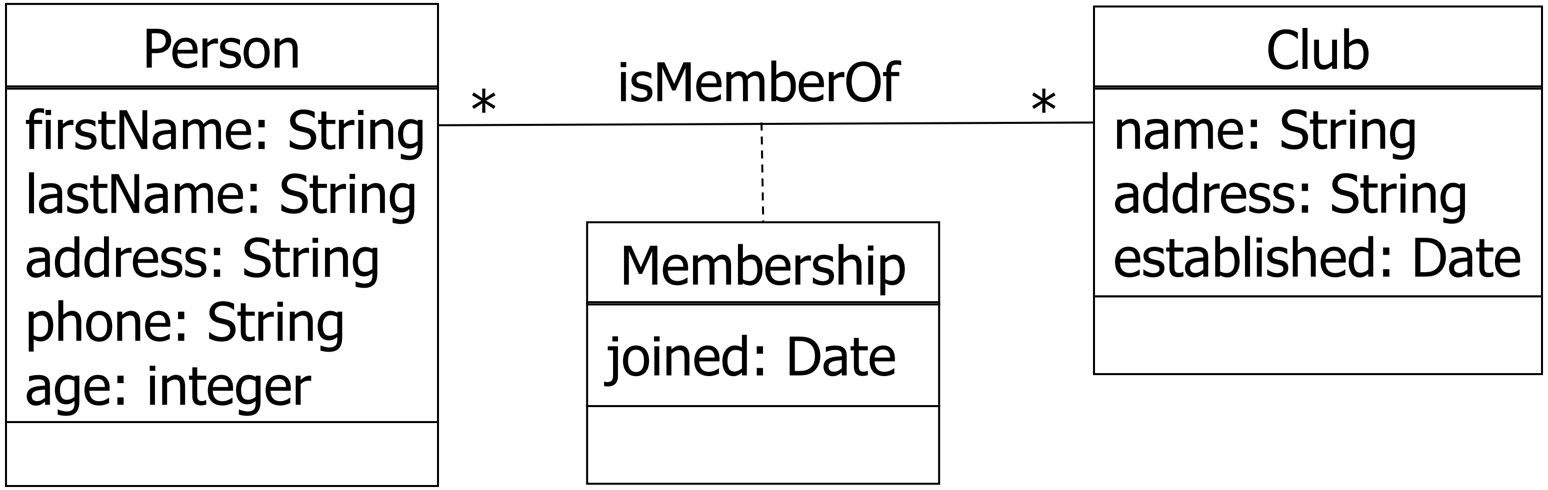
Create Table

```
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, system rejects any attempt to insert a NULL value in the column.
- Can specify a DEFAULT value for the 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  IsMemberOf  (
  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 all clubs”

```
SELECT * FROM Club
```

The * above means “get all column values”

“Find all persons who are under 18”

```
SELECT  *
FROM    Person p
WHERE   p.Age < 18
```

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"

Person

<u>Id</u>	FIrstName	Age
1	Jane	12
2	Bob	22
3	Mary	31
4	Alice	17

IsMemberOf

<u>PersonId</u>	ClubId
1	1
2	2
4	3
4	2

Club

<u>Id</u>	Name
1	Chess
2	Tennis
3	Bridge
4	Swim

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, IsMemberOf 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"

an alias

```
SELECT   p.FirstName,  p.Address
FROM     Person p, Club c, IsMemberOf m
WHERE    p.Age < 20  AND  c.Name = 'Tennis'
         AND  m.PersonId = p.Id
         AND  m.ClubId = c.Id
```

join clauses

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 table was created, unless DEFAULT was specified when creating column

INSERT

- 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 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]
```

- TableName can be name of a base table or an updatable view
- 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

DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- TableName can be name of a base table or an updatable view.
- searchCondition is optional; if omitted, all rows are deleted from table. This does not delete table. If searchCondition is specified, only those rows that satisfy condition are deleted.

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

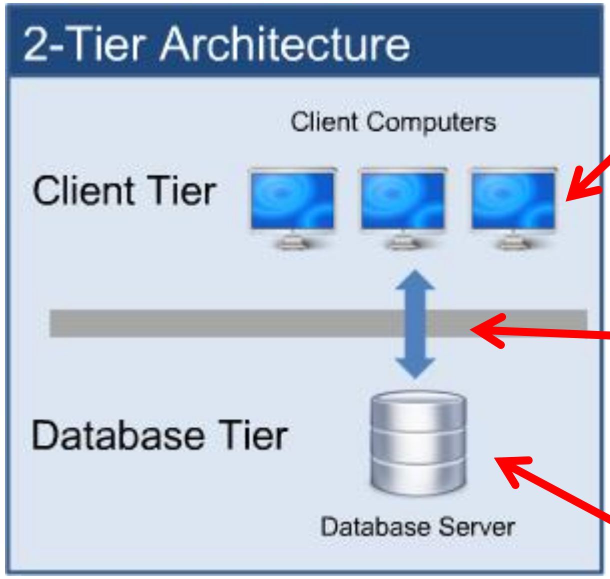
Examples of commercial RDBMS systems:

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

Examples of open source RDBMS systems:

- MySQL
- PostgreSQL
- MSQL

Relational DBMS

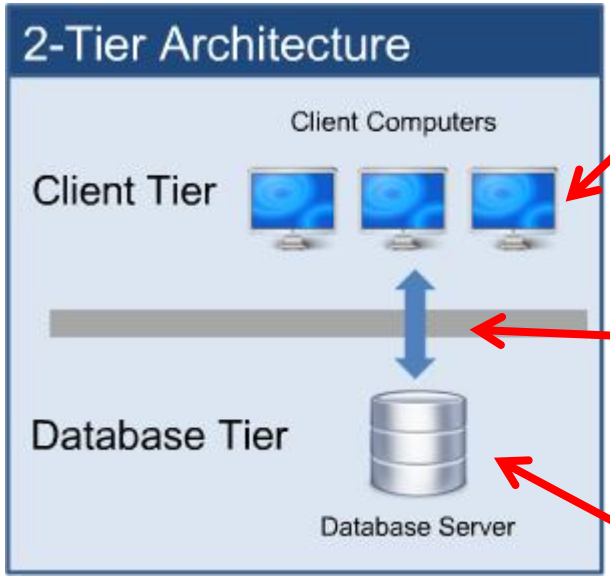


Client program can be in any language, but must "speak" the RDBMS communication language, usually over the network

Communication language/protocol or an API

RDBMS server, which accepts request over the network

Relational DBMS



Java client

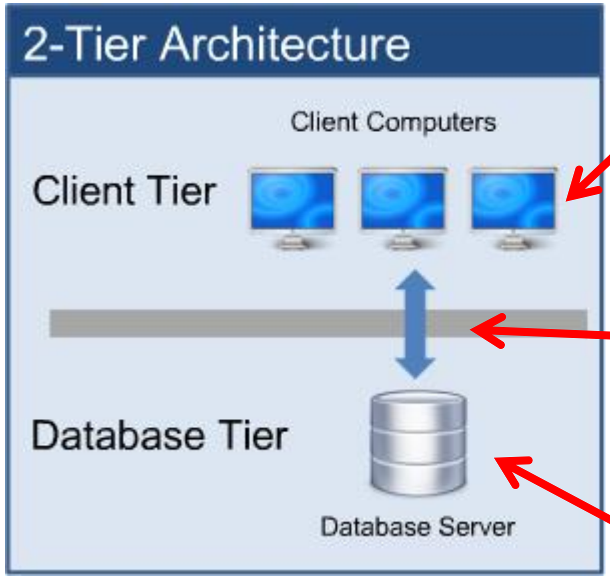
JDBC

C++ client

ODBC

RDBMS with ODBC/JDBC drivers

Relational DBMS



Java client → JDBC → RDBMS with ODBC/JDBC drivers

C++ client → ODBC → RDBMS with ODBC/JDBC drivers

JDBC

- JDBC is a Sun (now Oracle) trademark
  - It is often taken to stand for Java Database Connectivity
- Java is standardized, but there are many versions of RDBMS servers
- JDBC is a standard way of accessing SQL databases from Java

Driver types

- There are four types of drivers:
  - **JDBC Type 1 Driver** -- JDBC/ODBC Bridge drivers
    - ODBC (Open DataBase Connectivity) is a standard software API designed to be independent of specific programming languages
    - Sun provides a JDBC/ODBC implementation
  - **JDBC Type 2 Driver** -- use platform-specific APIs for data access
  - **JDBC Type 3 Driver** -- 100% Java, use a net protocol to access a remote listener and map calls into vendor-specific calls
  - **JDBC Type 4 Driver** -- 100% Java

Connector/J

- We will be using MySQL open-source RDBMS
- MySQL documentation:

  https://dev.mysql.com/doc/refman/5.1/en/index.html
- Connector/J is a JDBC Type 4 Driver for connecting Java to MySQL
- It is available as a jar file in /opt/classes

  mysql-connector-java-5.1.26-bin.jar

- As usual, it must be available on the Java CLASSPATH

Typical use of JDBC

1. Establish a **connection**
2. Create a JDBC **Statement**
3. **Execute** an SQL Statement
4. GET the **ResultSet**
5. Process the results from the ResultSet
6. **Close** the connection

Mapping types JDBC - Java

Impedance mismatch: differences between an OO language world and an RDB world;

for example, data type differences:

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Object persistence

- Objects may be written to and retrieved from disk using serialization; a class needs to implement the serializable interface (even with no methods)

```
public class Data implements serializable
{ ... }
```

- ObjectOutputStream and ObjectInputStream should be used with the writeObject and readObject methods
  - What are the problems with this approach?

Object persistence

- Objects may be written to and retrieved from disk using serialization; a class needs to implement the serializable interface (even with no methods)

```
public class Data implements serializable
{ ... }
```

- ObjectOutputStream and ObjectInputStream should be used with the writeObject and readObject methods
  - What are the problems with this approach?
  - Persisting thousands of objects may result in creating and managing thousands of files. What about millions of objects?

Object persistence

- Objects may be written to and retrieved from a relational database, such as MySQL
- 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

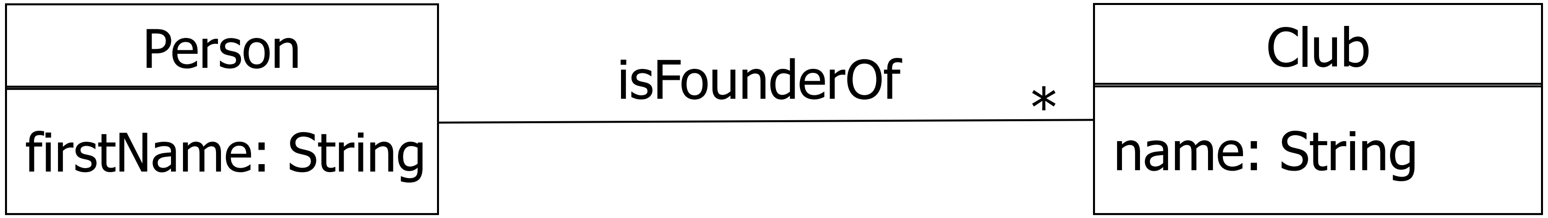
id	name	address	established
1	Chess	11 Oak St.	4/22/2004
…	…	…	…

- 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

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



Person

id	firstName	…
1	William	…
…	…	…

Club

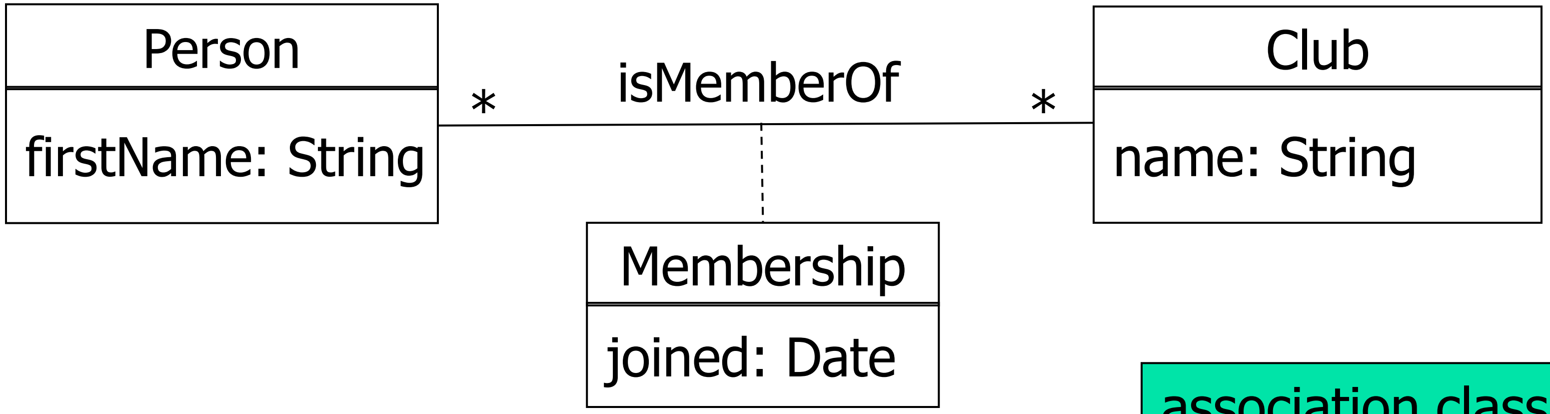
foreign key

id	fid	name	…
1	1	Chess	…
…	…	…	…

- 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	...

IsMemberOf

pId	cid	joined
1	1	3/4/2011

Club

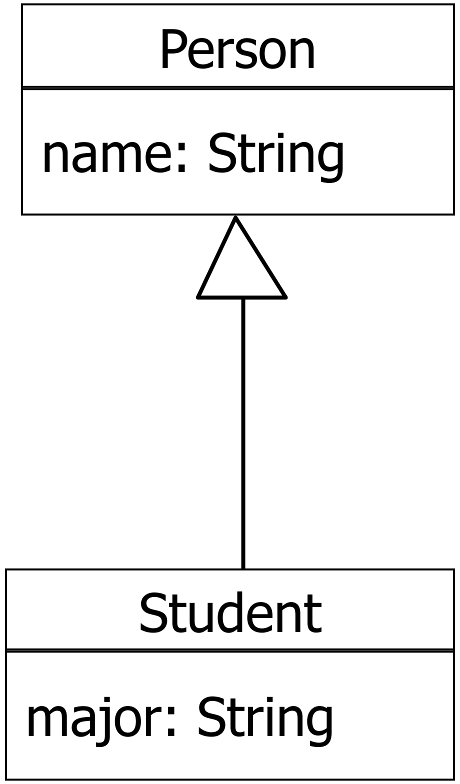
id	name	...
1	Chess	...

Object persistence

- Mapping of generalization 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

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

```
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

Person

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

Student

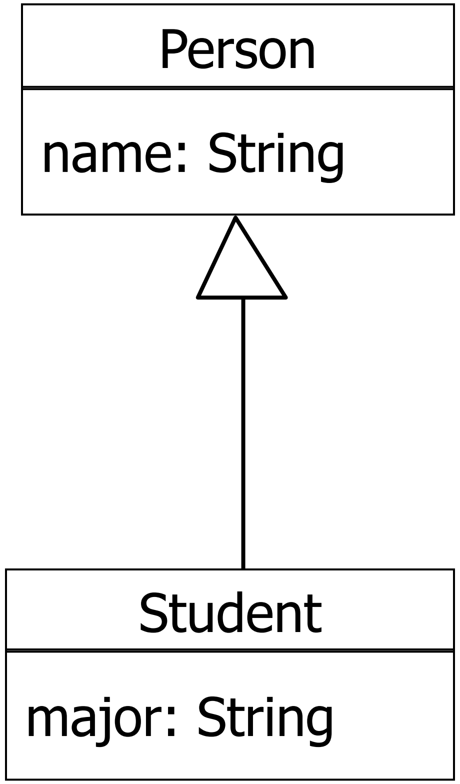
id	parentId	major	...
1	3	Math	...
...	...	...	...

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

- See also: http://agiledata.org/essays/mappingObjects.html

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”

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