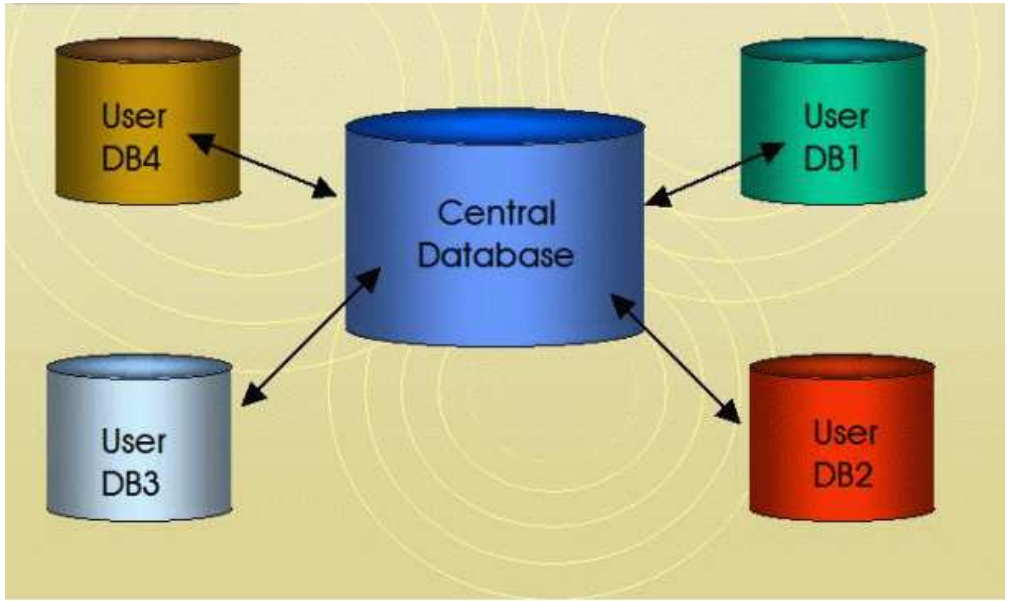
מסדי נתונים
persistency



פיתוח מערכות תוכנה מבוססות Java
אוהד ברזילי

מה בתוכנית?

- נדון בחלופות להשגת persistency בתוכנה:
  - Relational DB and SQL
  - JDBC
  - persistence layer
  - ORM

Persistency

- ...ומה קורה אם באמצע התוכנית יש הפסקת חשמל?

- על רוב התוכניות להיות מסוגלות לשמור מידע בין ריצות
  - Persistent Data (מידע מתמשך)
  - Transient Data (מידע נדיף, ארעי)

- מה כל כך קשה בלשמור מידע? אנו יכולים לשמור את המידע לקובץ בעצמנו

Persistency Challenges

- System crashes
  - What is the problem?
- Large data sets (say 50GB)
  - What is the problem?
- Simultaneous access by many users
  - Need locks: we know them from OS, but now data on disk; and is there any fun to re-implement them?
- Heterogeneous Clients
  - Local and remote
  - Different OS
  - Different programming languages
- Complex data and complex operations

Transaction (ACID)

- **Atomicity**: guarantee that either all of the tasks of a transaction are performed or none of them are (COMMIT or ROLLBACK).
- **Consistency**: ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not).
- **Isolation**: no operation outside the transaction can ever see the data in an intermediate state. This enables transactions to operate independently of and transparent to each other.
- **Durability**: once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction

What *Is* a Relational Database Management System ?

Database Management System = DBMS
Relational DBMS = RDBMS

- A collection of files that store the data
- A big C program written by someone else that accesses and updates those files for you
- Where are RDBMS used?
  - Backend for traditional “database” applications:
    - CRM
    - ERP
    - Banking, insurance,…
  - Backend for large Websites
  - Backend for Web services

Crash course on Relational Databases and SQL

- כל דבר זה טבלה
  - שדות של רשומה => עמודות בטבלה
- פעולות על מבנה הטבלאות (schema):
  - CREATE, ALTER, DROP
- פעולות על תוכן הטבלאות:
  - INSERT, SELECT, UPDATE, DELETE
- פעולות על כמה טבלאות:
  - JOIN (INNER, LEFT, RIGHT)
  - KEYS (PRIMARY, FOREIGN)

KEYS

Primary Key of Order Table

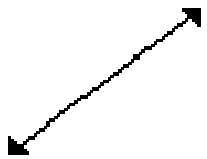
Order Table

Order Number	Name	Address	Phone	...	Order Size	Total Amount
115	Fred Yo	123 Mair	772-8831		762	877,999
116	Mike Boo	321 West	772-9813		54	62000
117	Ed Tibbs	222 East	772-8163		1100	1,267,450
118	Fred Yo	123 Mair	772-8831		552	607,000

בעיה: המידע משוכפל שלא לצורך

Order Table

Order Number	Customer Number	...	Order Size	Total Amount
115	23		762	877,999
116	65		54	62000
117	43		1100	1,267,450
118	23		552	607,000



Primary Key of Customer Table

Foreign Key in Order Table

Customer Table

Customer Number	Name	Address	Phone
23	Fred Yo	123 Main	772-8831
43	Mike Boo	321 West	772-9813
65	Ed Tibbs	222 East	772-8163
....			

How SELECT works

People				
First_Name	Last_Name	Gender	Age	Phone
John	Smith	M	27	2-4315
Sally	Jones	F	27	3-1542
John	White	M	32	2-4315
Mary	Smith	F	42	5-4321

```
SELECT First_Name, Last_Name FROM People
  WHERE Age > 30;
```

Result:

John	White
Mary	Smith

אתגרי ה SQL

- שלמות המידע
- נורמליזציה של טבלאות
  - עקביות, יתירות, נכונות, יעילות

- Behind the scenes the DBMS has:
  - Query optimizer
  - Query engine
  - Storage management
  - Transaction Management (concurrency, recovery)

- מדריך מקוון:

http://code.google.com/edu/tools101/mysql.html

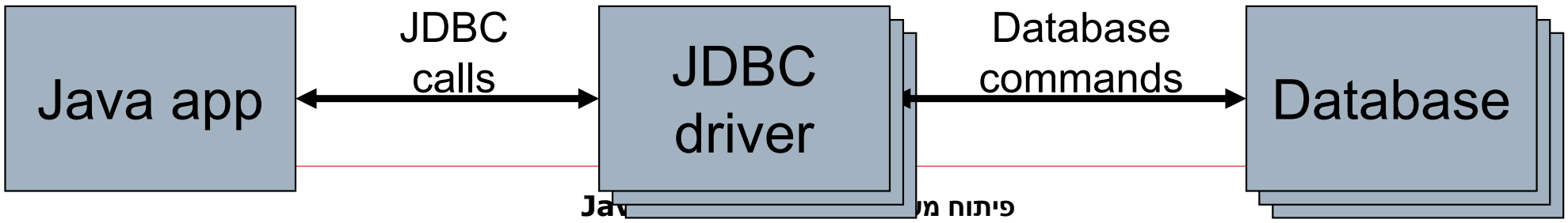
When RDBMS met Java

JDBC

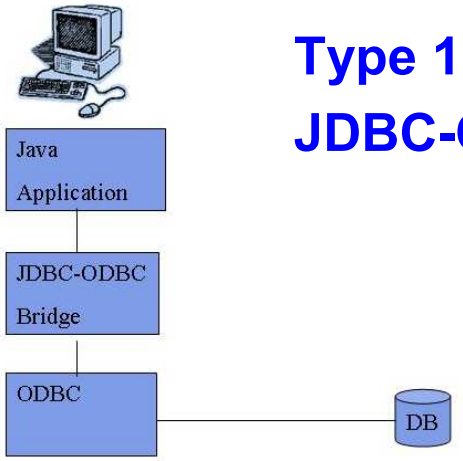
Java Database Connectivity (JDBC)

- An interface to communicate with a relational database
  - Allows database agnostic Java code
  - Treat database tables/rows/columns as Java objects

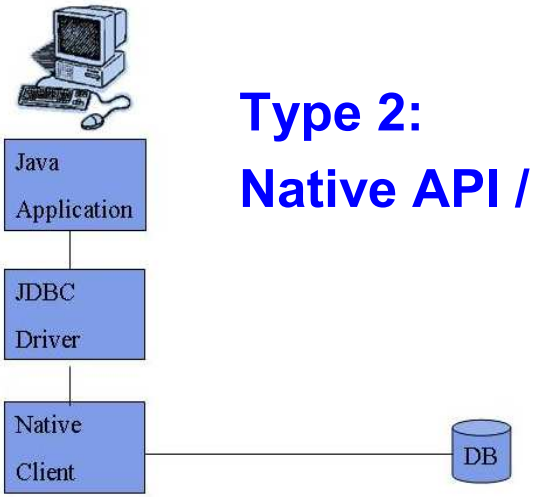
- JDBC driver
  - An implementation of the JDBC interface
  - Communicates with a particular database



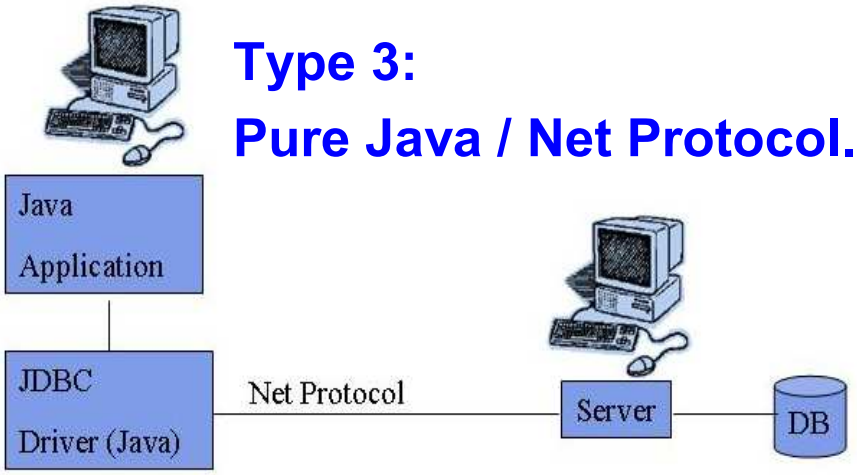
JDBC Driver Types



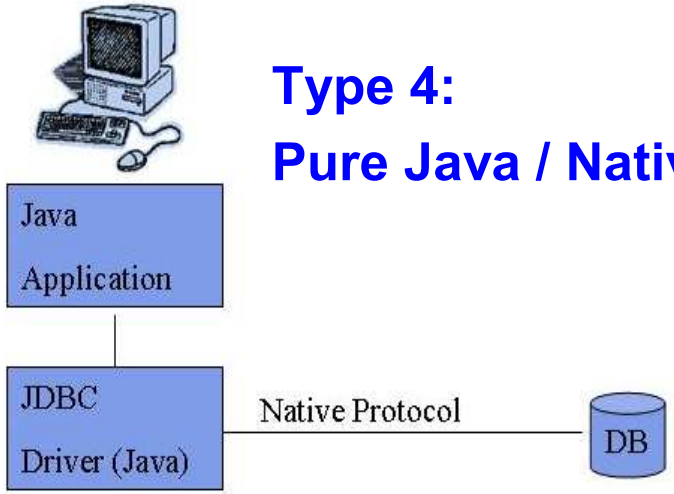
Type 1:
JDBC-ODBC Bridge



Type 2:
Native API / Partially Java



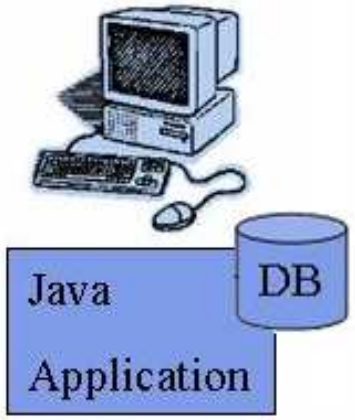
Type 3:
Pure Java / Net Protocol.



Type 4:
Pure Java / Native Protocol.

Embedding the DB - Java DB

- Java DB is Sun's supported distribution of the open source Apache Derby
  - Included as part of Java SE v.6
- Written in Java
- Transactional, secure, easy-to-use
- standards-based — SQL, JDBC API, and Java EE
- Small footprint - only 2MB
- Community
- http://java.sun.com/developer/technicalArticles/J2SE/Desktop/javadb/



Eclipse JDBC setup

- Install driver
  - Download MySQL JDBC driver from the Web
    - http://dev.mysql.com/downloads/connector/j/5.0.html
  - Unzip mysql-connector-xxx.jar
  - Add mysql-connector-xxx.jar to Eclipse project
    - Project → Properties → Java Build Path → Libraries → Add External JARs

JDBC steps

1. Connect to database
2. Query database (or insert/update/delete)
3. Process results
4. Close connection to database

```
import java.sql.*;

public class Tester {
   public static void main(String[] args) {
      try {
         // Load JDBC driver
         Class.forName("com.mysql.jdbc.Driver").newInstance();

         // Make connection
         String url =
               "jdbc:mysql://128.100.53.33/GRP?user=USER&password=PASS"
         Connection conn = DriverManager.getConnection(url);

         // Create statement
         Statement stmt = conn.createStatement();

         // Query database
         ResultSet rs = stmt.executeQuery("SELECT * FROM users");

         // Process results
         while (rs.next()) {
               <next slide...>
         }

         // Cleanup
         rs.close(); stmt.close(); conn.close();

   } catch (Exception e) { System.out.println("exception " + e); }
}
```

Process Result

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

while (rs.next()) {
    String userid = rs.getString(1);
    String firstname = rs.getString("firstname");
    String lastname = rs.getString("lastname");
    String password = rs.getString(4);
    int type = rs.getInt("type");
    System.out.println(userid + " " + firstname + " " +
       lastname + " " + password + " " + type);
}
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

Initial Cursor position

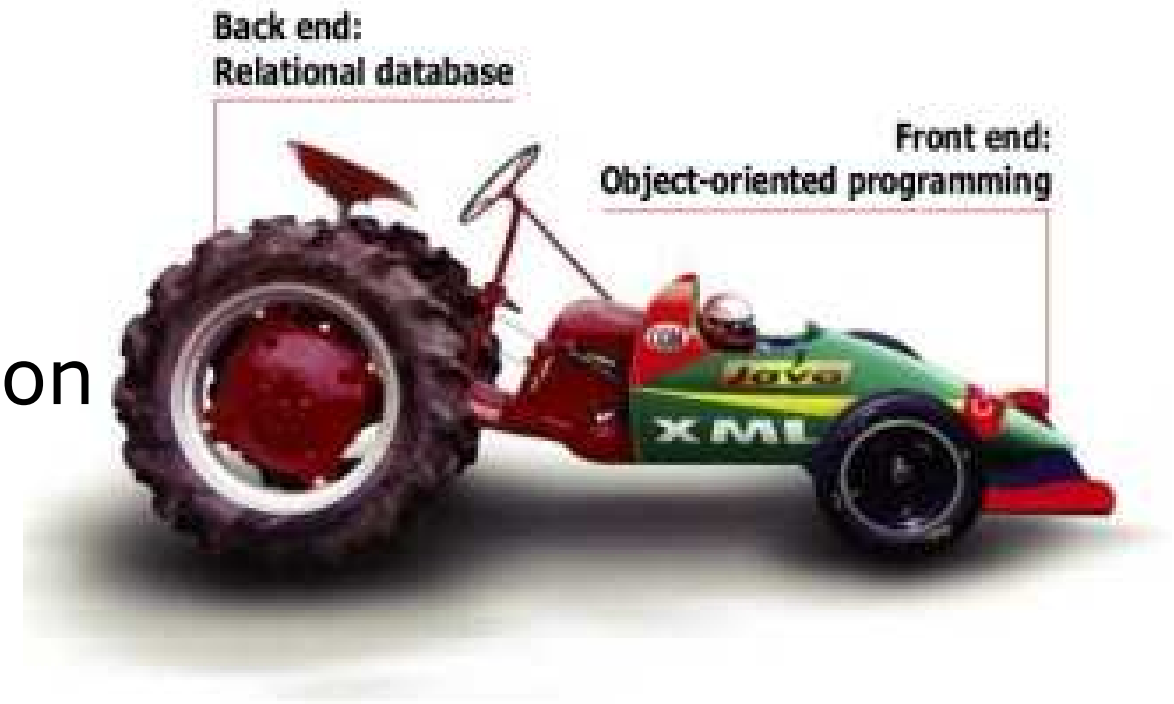
Cursor after first call to *rs.next()*

JDBC Features

- JDBC 2.1
  - Updates, Queries, joins
  - Transactions
  - Prepared Statements
  - Meta Data
  - Callable Statements
- JDBC 3.0
  - Savepoints
  - Pooling
  - New Data Types
  - RowSets
- JDBC 4.0
  - ease-of-use and programmer productivity
  - SQL 2003
  - XML as a First-Class SQL Data Type

Paradigm Mismatch

- Problems caused when objects need to be stored in relational tables.
  - Granularity
  - Subtypes
  - Identity
  - Associations
  - Graph Navigation
  - Cost



Problem of Granularity

- Data types can have various kinds of granularity
  - STREET is a field in CUSTOMER

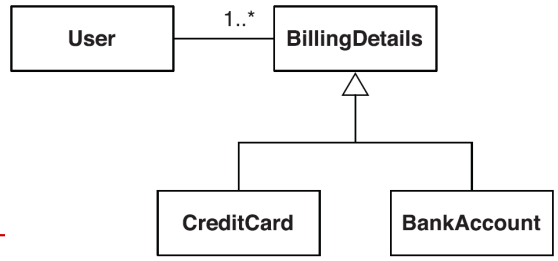
  vs.
  - STREET is a field in ADDRESS

- Performance vs. Modeling

- Poor support for used defined types (UDT) in SQL.
  - No implementation is portable.

- Forces developers to use less flexible representations on the object model.

Problem of Subtypes



- Java objects implements inheritance (super & sub classes)
- Each sub or super class will define different data & completely different functionality.
- An object can be associated with objects of different classes, but of same type (polymorphism).
- SQL provides no support for **inheritance** or **polymorphism**.

Problem of Identity

- Checking if objects are identical
- Java defines two notations:
  - Object **identity** [==]
  - **Equality** of value [.equals()]
- In SQL it is just checking if the two **primary keys** are the same
- Two or more objects can represent the same row of data
- SQL **surrogate keys** are not part of the **model**

Problem of Association

- Associations represent relation between entities
- Object **references** represents associations
- In SQL association is by **foreign keys**
- Object references are directional, foreign keys are not
- It isn't possible to determine **multiplicity** of a association just by looking at a **class**
- **SQL multiplicity** is always one-to-one or one-to-many, and can be determined by looking at the foreign key

Object Graph Navigation

- Difference in the way objects are accessed in Java and SQL

- `obj.getDetails().getFirstField()`
- In SQL it would be a set of select statements with joins.

- “n+1 select problem”

Cost Of The Mismatch

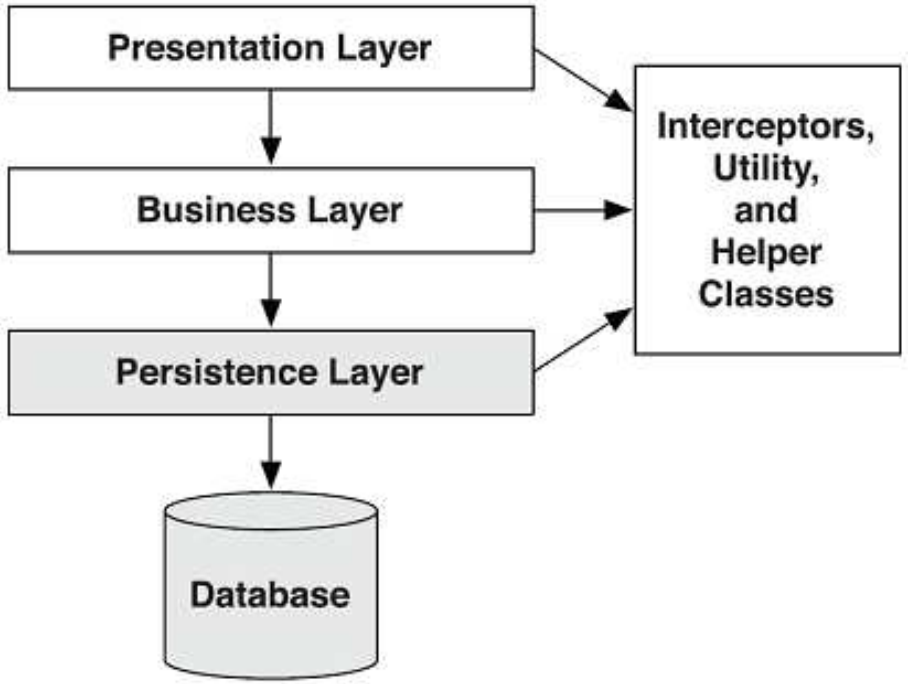
- Requires significant time & effort
- Up to 30%* of code is devoted to handle tedious SQL/JDBC
- Difficulty in modeling the business entities

- Need to bridge Object/Relational mismatch

*** Java Persistence with Hibernate, Christian Bauer and Gavin King**

Layered Architecture

- A layered architecture defines interfaces between various concerns
- Layers communicate top to bottom
- A layer is dependent only on the layer directly below it



Implementing A Layered Architecture

- Hand-coding a persistence layer
- Using serialization
- Other technologies
  - Object-oriented database systems
  - XML
- Object/Relational Mapping framework
  - Using EJB entity beans
  - Hibernate

Hand-coding a persistence layer

- Typically using design patterns such as DAO
- Involves manual coding of the persistence layer
  - Hibernate is about 80,000 LOC + 25,000 lines of Unit test code

Using Serialization

- Used by RMI
- Data can only be accessed as a whole
- Serial access

Other leads

- Object-oriented database systems
  - An object-oriented database management system (OODBMS) is more like an extension to the application environment than an external data store
  - The Object Data Management Group (ODMG) is no longer active
  - The Java Data Objects (JDO) specification (published in April 2002)
- XML persistence
  - A variation on the serialization theme
  - object/hierarchical mismatch

Why relational DB survives?

- "Relational technology is a **known quantity**, and this alone is sufficient reason for many organizations to choose it". (Java Persistence with Hibernate, Christian Bauer and Gavin King)

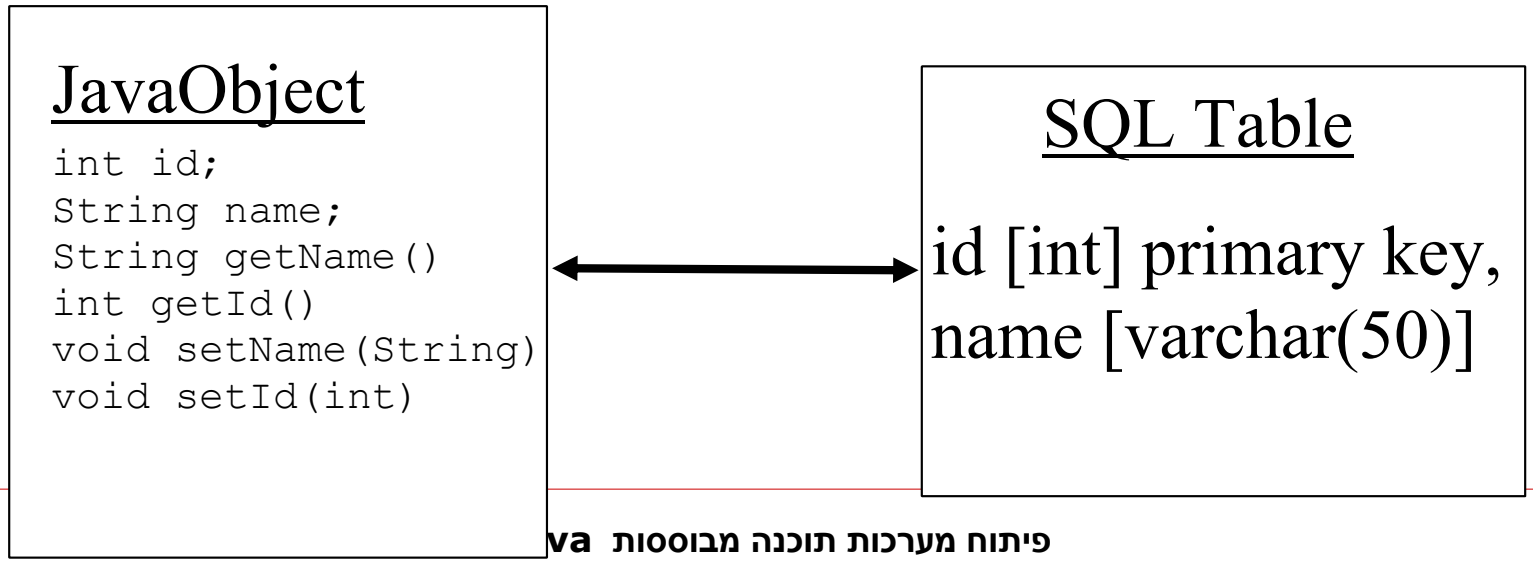
- Relational databases are entrenched because:
  - they're an **incredibly flexible** and **robust** approach to data management.
  - Due to the **complete and consistent theoretical foundation** of the relational data model, relational databases can effectively guarantee and protect the **integrity** of the data, among other desirable characteristics

- *Data independence: data lives longer than any application does.*

Object/**R**elational **M**apping (ORM)

- Object/Relational Mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database

- ORM, in essence, works by (reversibly) transforming data from one representation to another



Using EJB 2.1 as ORM

- EJB 2.1 specification describes:
  - *bean-managed persistence* (BMP)
  - *container-managed persistence* (CMP)

- Deficiencies:
  - CMP beans are defined in one-to-one correspondence to the tables of the relational model. Thus, they're too *coarse grained*
  - On the other hand, CMP beans are also too *fine grained* to realize the stated goal of EJB: the definition of reusable software components. A reusable component should be a *very* coarse-grained object, with an external interface that is stable in the face of small changes to the database schema.
  - Support only implementation inheritance, but don't support polymorphic associations and queries
  - Entity beans aren't portable in practice. Capabilities of CMP engines vary widely between vendors, and the mapping metadata is highly vendor-specific
  - Entity beans aren't serializable. We must define additional *data transfer objects* (DTOs, also called *value objects*) when we need to transport data to a remote client tier
  - EJB is an *intrusive* model; it mandates an unnatural Java style and makes reuse of code outside a specific container extremely difficult

Hibernate

- Popular Open Source (LGPL) Object/Relational Mapping (ORM) tool
- Lightweight
- Non-intrusive
- Transparent persistence for **POJO**s (Plain Old Java Objects)
- Core of JBoss CMP 2.0 impl.
- Used as an “inspiration” for EJB 3.0 and JPA

A persistent class

- Event.java is a simple example for a class whose objects can be made persistent

- It needs
  - An ID field
  - A parameterless constructor
  - ...that's all!

Event.java (1)

```java
package events;

import java.util.Date;

/**
 * This sample class uses Hibernate to make its state persistent.
 */
public  class Event {

    // This id is unique system-wide.
    // Such unique ids are required by Hibernate for all persistent
    // classes.
    private  Long id;

    private  String title;
    private  Date date;

    // A constructor without arguments is required by Hibernate
    // so that it can recreate this class using Reflection.
    // It should have package visibility "or greater".
    public  Event() {}
```

Event.java (2)

```
    public  Long getId() {
        return  id;
    }

    private  void setId(Long id) {
        this .id = id;
    }

    public  Date getDate() {
        return  date;
    }

    public  void setDate(Date date) {
        this .date = date;
    }

    public  String getTitle() {
        return  title;
    }

    public  void setTitle(String title) {
        this .title = title;
    }
}
```

Storing, Querying, Recalling

- EventManager.java shows how to do
  - Transactions
  - Queries

- We'll take a look at two methods, the simple **`createAndStoreEvents()`** and the more complex **`addPersonToEvent()`**

Storing

```
private  Long createAndStoreEvent(String title, Date theDate) {
    Session session =
      HibernateUtil.getSessionFactory().getCurrentSession();

    session.beginTransaction();

    Event theEvent = new Event();
    theEvent.setTitle(title);
    theEvent.setDate(theDate);

    session.save(theEvent);
    session.getTransaction().commit();

    return theEvent.getId();
}
```

Querying...

```
private  void addPersonToEvent(Long personId, Long eventId) {
    Session session =
      HibernateUtil.getSessionFactory().getCurrentSession();

    session.beginTransaction();

    Person aPerson = (Person) session
        .createQuery("select p from Person p left join fetch
          p.events where p.id = :pid")
        .setParameter("pid", personId)
        .uniqueResult();

    Event anEvent = (Event) session.load(Event.class, eventId);

    session.getTransaction().commit(); // End of first unit of work
```

...and Updating

```
        aPerson.getEvents().add(anEvent); // aPerson is detached

        // Begin second unit of work
        Session session2 =
          HibernateUtil.getSessionFactory().getCurrentSession();

        session2.beginTransaction();
        session2.update(aPerson); // Reattachment of aPerson
        session2.getTransaction().commit();
    }
```

מקורות

- http://www.cs.tau.ac.il/~jackassa/db/DBHomePage.htm
- http://www.cis.upenn.edu/~matuszek/cit597-2003/
- http://code.google.com/edu/tools101/mysql.html
- http://www.eecg.utoronto.ca/~vinod/mie456/jdbctut.ppt
  - http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html
  - http://otn.oracle.co.kr/admin/seminar/data/otn-jdbc.ppt
  - http://notes.corewebprogramming.com/student/JDBC.pdf
- http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html

Persistency

- ...ומה קורה אם פורצת שריפה?