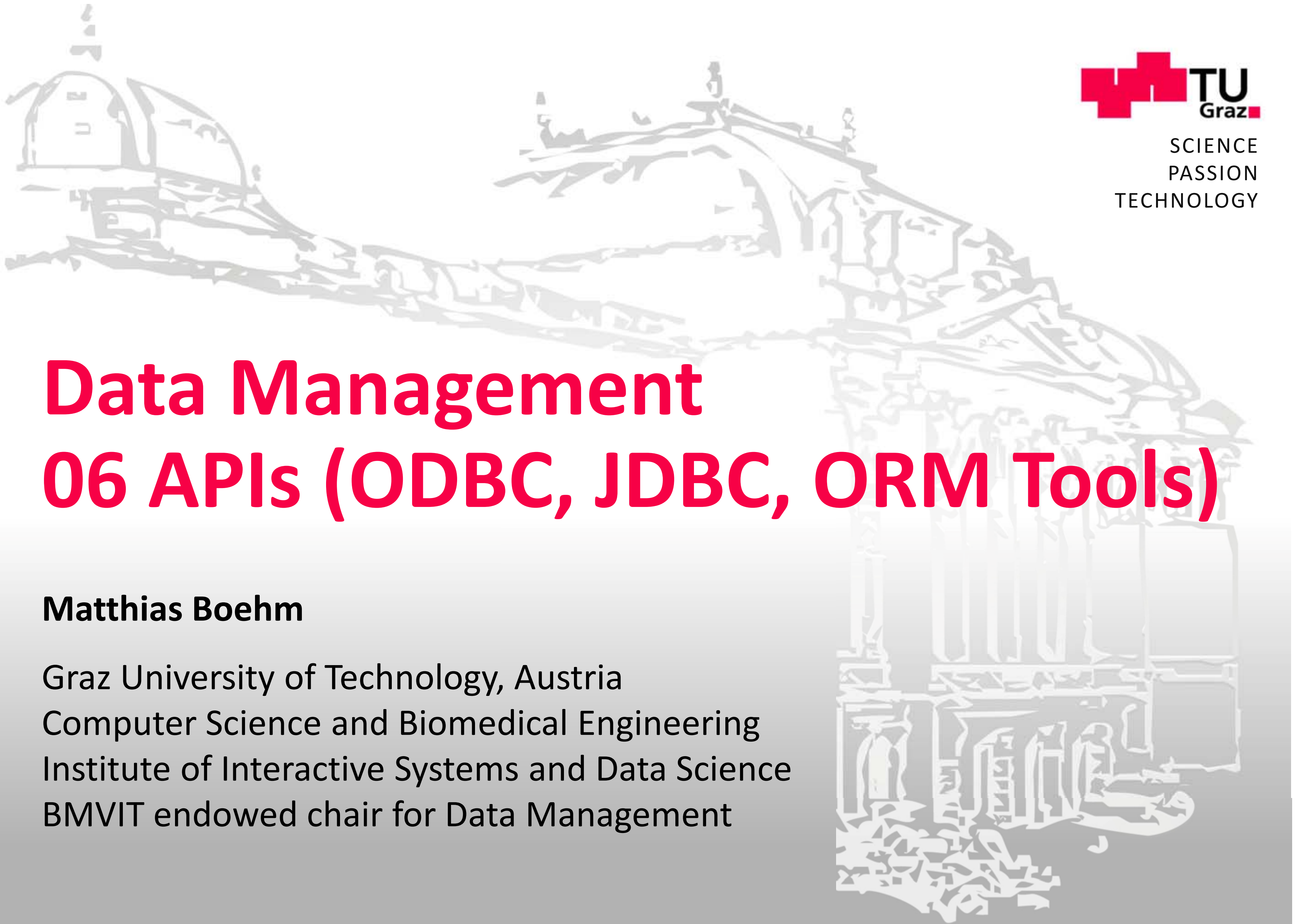


SCIENCE
PASSION
TECHNOLOGY

Data Management
06 APIs (ODBC, JDBC, ORM Tools)

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

Last update: Apr 18, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
  - **Live Streaming** Mo 4.10pm until end of semester (June 30)

- **#2 Reminder Communication**
  - **Newsgroup:** news://news.tugraz.at/tu-graz.lv.dbase; **no TeachCenter forum!** (https://news.tugraz.at/cgi-bin/usenet/nntp.csh?tu-graz.lv.dbase)
  - **Office hours:** Mo 1pm-2pm (https://tugraz.webex.com/meet/m.boehm)

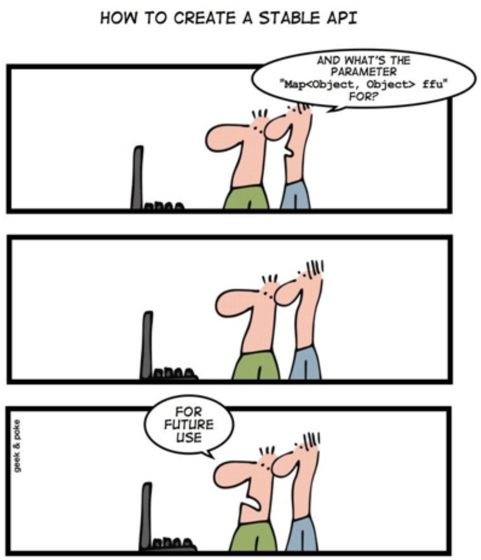
- **#3 Exercise 1 Summary**
  - **Started grading Apr 10** (after 7+3 late days)

79.5%

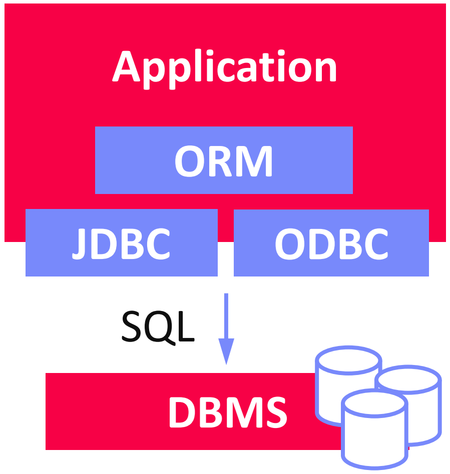
- **#4 Exercise 2 Reminder**
  - **Published Apr 7**, recommend schema **Apr 10**
  - Submission Deadline **Apr 28, 11.59 pm**

What’s an API and again, why should I care?

- **Application Programming Interface (API)**
  - Defined **set of functions or protocols** for system or component communication
  - Interface independent of concrete implementation → **decoupling of applications** from underlying libraries / systems
  - API stability of utmost importance


HOW TO CREATE A STABLE API
AND WHAT'S THE PARAMETER "Map<Object, Object> ffu" FOR?
FOR FUTURE USE
geek & poke

- **Examples**
  - **Linux:** kernel-user space API → system calls, POSIX (Portable Operating System Interface)
  - **Cloud Services:** often dedicated REST (Representational State Transfer) APIs
  - **DB Access: ODBC/JDBC and ORM frameworks**



Agenda

- Exercise 2: **Query Languages and APIs**
- **Call-level Interfaces (ODBC/JDBC) and Embedded SQL**
- **Object-Relational Mapping Frameworks**

Exercise 2:
Query Languages and APIs

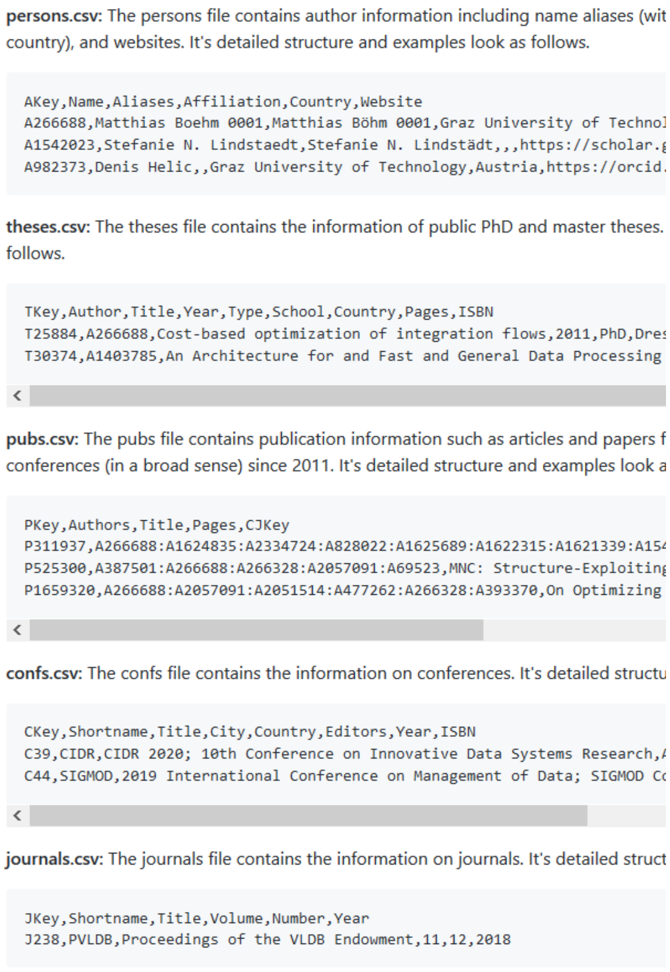
Extension of Exercise 2 Preview
from Mar 30

Published: **Apr 07, 2020**

Deadline: **Apr 28, 2020**

Exercises: DBLP Publications

- **Dataset**
  - CC0-licensed, derived (extracted, cleaned) from **DBLP** (https://dblp.org Feb 1, 2020) for publication **year ≥ 2011 + DM venues**
  - Clone or download your copy from https://github.com/tugraz-isds/datasets.git
- **Exercises**
  - **01** Data modeling (relational schema)
  - **02 Data ingestion and SQL query processing**
    - **Relational schema** + **ingestion**
    - SQL query processing + extra credit
  - **03** Physical design tuning, query processing, and transaction processing
  - **04** Large-scale data analysis (distributed data ingestions and query processing)



persons.csv: The persons file contains author information including name aliases (wit
country), and websites. It's detailed structure and examples look as follows.

```
AKey,Name,Aliases,Affiliation,Country,Website
A266688,Matthias Boehm 0001,Matthias Böhm 0001,Graz University of Technol
A1542023,Stefanie N. Lindstaedt,Stefanie N. Lindstädt,,,https://scholar.
A982373,Denis Helic,,Graz University of Technology,Austria,https://orcid.
```

theses.csv: The theses file contains the information of public PhD and master theses.
follows.

```
TKey,Author,Title,Year,Type,School,Country,Pages,ISBN
T25884,A266688,Cost-based optimization of integration flows,2011,PhD,Dres
T30374,A1403785,An Architecture for and Fast and General Data Processing
```

pubs.csv: The pubs file contains publication information such as articles and papers f
conferences (in a broad sense) since 2011. It's detailed structure and examples look a

```
PKey,Authors,Title,Pages,CJKey
P311937,A266688:A1624835:A2334724:A828022:A1625689:A1622315:A1621339:A154
P525300,A387501:A266688:A266328:A2057091:A69523,MNC: Structure-Exploitin
P1659320,A266688:A2057091:A2051514:A477262:A266328:A393370,On Optimizing
```

confs.csv: The confs file contains the information on conferences. It's detailed structu

```
CKey,Shortname,Title,City,Country,Editors,Year,ISBN
C39,CIDR,CIDR 2020; 10th Conference on Innovative Data Systems Research,
C44,SIGMOD,2019 International Conference on Management of Data; SIGMOD C
```

journals.csv: The journals file contains the information on journals. It's detailed struct

```
JKey,Shortname,Title,Volume,Number,Year
J238,PVLDB,Proceedings of the VLDB Endowment,11,12,2018
```

Task 2.1: Schema Creation via SQL (3/25 points)

- **Schema creation via SQL**
  - Relies on lectures **04 Relational Algebra** and **05 Query Languages (SQL)**
  - Setup DBMS PostgreSQL
  - Create database `db<studentID>` and setup relational schema
    - **Ignore** (1) **person aliases**, and (2) **conference editors**
    - Primary keys, foreign keys, `NOT NULL`, `UNIQUE`, specific `CHECK`s
    - `CreateSchema.sql`

- **Recommended Schema (published Apr 10)**

Task 2.2 Data Ingestion via CLI (8/25 points)

- **Data Ingestion Program via ODBC/JDBC**
  - Relies on lectures **05 Query Languages (SQL)** and **06 APIs (ODBC, JDBC)**
  - Write a program that performs **deduplication and data ingestion**
  - Programming language of your choosing (Python, Java, C#, C++ recommended)

- **Data Ingestion Process**
  - Data: https://github.com/tugraz-isds/datasets/tree/master/dblp_publications
  - Invoke your ingestion program as follows → script to compile and run

```
IngestData ./confs.csv ./journals.csv \
  ./persons.csv ./pubs.csv ./theses.csv \
  <host> <port> <database> <user> <password>
```

Task 2.3: SQL Query Processing (9/25 points)

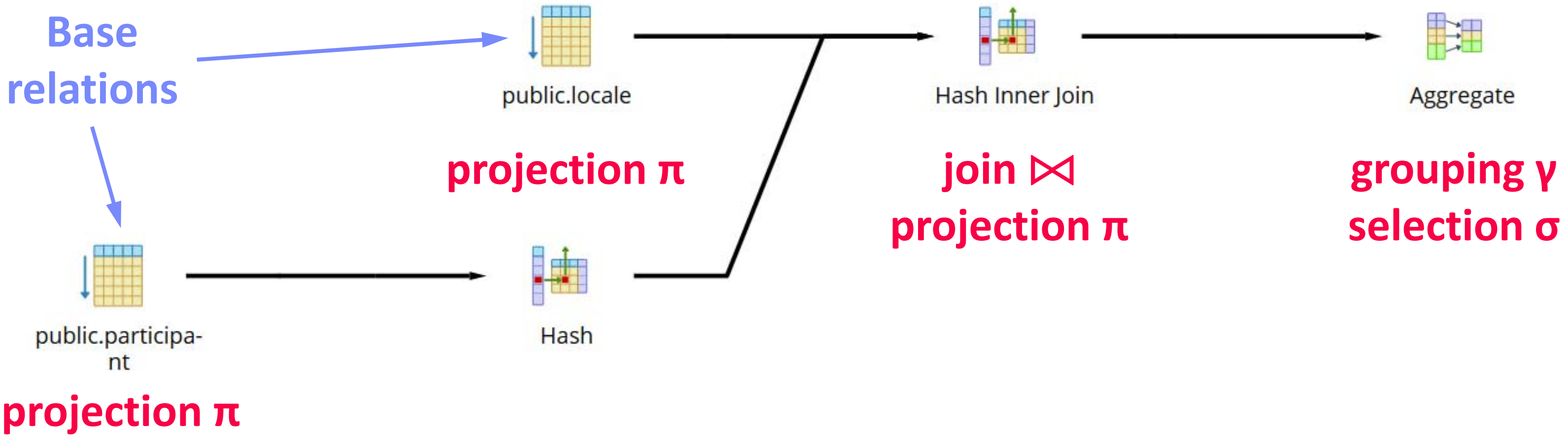
- **SQL Query Processing**
  - Relies on lecture **05 Query Languages (SQL)**
  - Write SQL queries (w/ results in comments) → `Queries.sql`

- **Example Queries**
  - **Q01:** Where did the conference SIGMOD 2019 (short name, year) take place? (return city and country)
  - **Q02:** Which persons are affiliated with Graz University of Technology? (return name, website; sorted ascending by name)
  - **Q05:** Which cities hosted more than 2 conferences? (return city, country, count; sorted decreasing by count)
  - **Q06:** Create a histogram that counts the number of papers for all groups of papers with equal #authors. (return #authors, #papers; sorted asc # authors)
  - **Q07:** How many distinct theses and papers did persons currently affiliated with Austrian institutions publish? (return single count)

Task 2.4: Query Plans (5/25 points)

- **Explain Query Plans**
  - Relies on lecture **04 Relational Algebra** and **05 Query Languages (SQL)**
  - Obtain and **analyze execution plans** of at least two queries
  - `ExplainQueries.sql`
- **Example: Recap: Participants/Locations from Lecture 04**
  - Text Explain

```
EXPLAIN VERBOSE SELECT L.location, count(*)
  FROM Participant P, Locale L WHERE P.lid = L.lid
  GROUP BY L.location HAVING count(*)>1
```



Base relations

projection π

join ⋈
projection π

grouping γ
selection σ

projection π

Task 2.5 Extra Credit (5 extra points)

- **Data Ingestion Program via ODBC/JDBC**
  - Relies on lectures **05 Query Languages (SQL)** and **06 APIs (ODBC, JDBC)**
  - Write a program that reconstructs and **prints a person's publication list**
  - Programming language of your choosing (Python, Java, C#, C++ recommended)

- **Invocation**
  - Print pub list for person to **stdout**

```
PrintPubs <person_name> \
<host> <port> <database> <user> <password>
```

- **Example Output**

```
* Johanna Sommer, Matthias Boehm 0001, Alexandre V. Evfimievski, Berthold Reinwald,
Peter J. Haas; MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions;
SIGMOD; 2019; 1607-1623.

* Matthias Boehm 0001, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V.
Evfimievski, Niketan Pansare; On Optimizing Operator Fusion Plans for Large-Scale Machine
Learning in SystemML; PVLDB; 2018; 1755-1768.

* Ahmed Elgohary, Matthias Boehm 0001, Peter J. Haas, Frederick R. Reiss, Berthold
Reinwald; Compressed Linear Algebra for Large-Scale Machine Learning; PVLDB; 2016; 960-971.

* Matthias Boehm 0001, Benjamin Schlegel, Peter Benjamin Volk, Ulrike Fischer, Dirk Habich,
Wolfgang Lehner; Efficient In-Memory Indexing with Generalized Prefix Trees; BTW; 2011;
227-246.
```

Call-level Interfaces (ODBC/JDBC) and Embedded SQL

Call-level Interfaces vs Embedded SQL

- **#1 Call-level Interfaces**
  - Standardized in ISO/IEC SQL – Part 3: CLI
  - **API of defined functions for dynamic SQL**
  - **Examples:** ODBC (C/C++), JDBC (Java), DB-API (Python)

- **#2 Embedded SQL**
  - Standardized in ISO/IEC SQL – Part 2: Foundation / Part 10 OLB
  - **Embedded SQL in host language** (typically static)
  - **Preprocessor** to compile CLI protocol handling
    ➔ **SQL syntax and type checking**, **but static** (SQL queries, DBMS)
  - **Examples:** ESQL (C/C++), SQLJ (Java)

Embedded SQL

- **Overview**
  - Mix host language constructs and SQL in data access program ➔ simplicity?
  - Precompiler translates program into valid host language program
  - Primitives for creating cursors, queries and updates, etc

➔ In practice, limited relevance

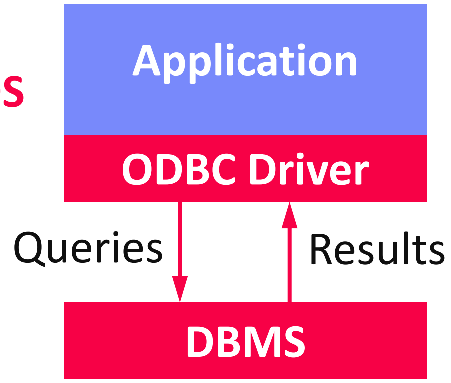
- **Example SQLJ**
  - Cursors with and without explicit variable binding

```
#sql iterator StudIter
   (int sid, String name);
StudIter iter;
#sql iter = {SELECT * FROM Students};

while( iter.next() )
   print(iter.sid, iter.name);

iter.close();
```

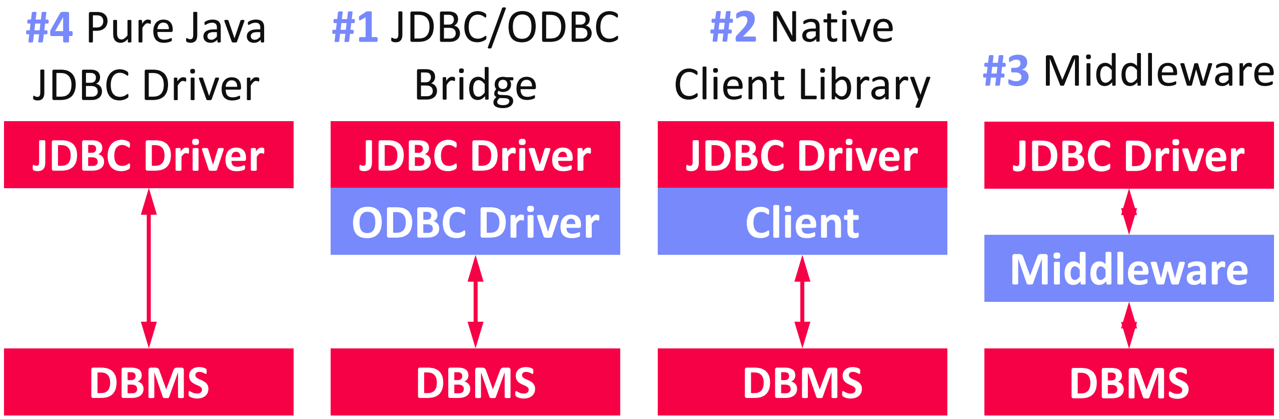
```
int id = 7;
String name;

#sql {SELECT LName INTO :name
   FROM Students WHERE SID=:id};

print(id, name);
```

CLI: ODBC and JDBC Overview

- **Open Database Connectivity (ODBC)**
  - **API for accessing databases independent of DBMS and OS**
  - Developed in the **early 1990s → 1992** by Microsoft (superset of ISO/IEC SQL/CLI and Open Group CLI)
  - **All relational DBMS have ODBC implementations**, good programming language support

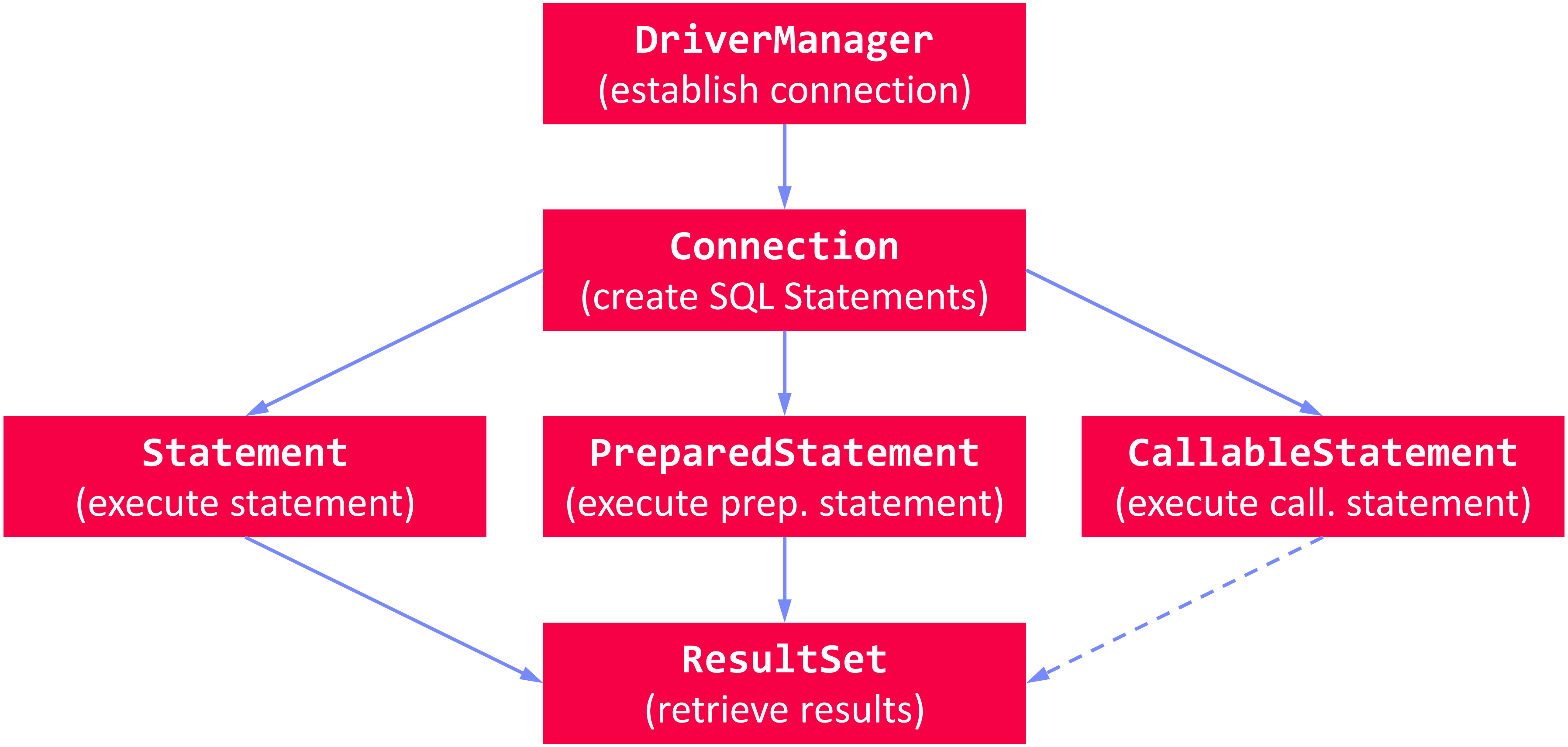


- **Java Database Connectivity (JDBC)**
  - **API for accessing databases independent of DBMS from Java**
  - Developed and released by Sun in **1997**, JDBC 4.0 (2006), JDBC 4.3 in Java 9
  - Most relational DBMS have JDBC implementations
  - Types of Drivers



Note: Reuse of drivers from open source DBMS

JDBC Components and Flow



JDBC Connection Handling

- **Establishing a Connection**
  - **DBMS-specific URL strings** including host, port, and database name

```
Connection conn = DriverManager
    .getConnection(“jdbc:postgresql:”+
    ”//localhost:5432/db1234567”,
    username, password);
```

  - Stateful handles representing user-specific DB sessions
  - JDBC driver is usually a jar on the class path
  - **Connection and statement pooling** for performance

```
META-INF/services/
java.sql.Driver
```

- **JDBC 4.0**
  - **Explicit driver class loading and registration no longer required**

```
Class.forName(
    "org.postgresql.Driver");
```

  - Improved connection management (e.g., status of DB connections)
  - Other: XML, Java classes, row ID, better exception handling

JDBC Statements

- **Execute Statement**
  - Use for simple SQL statements w/o parameters
  - **Beware of SQL injection**
  - API allows fine-grained control over fetch size, fetch direction, batching, and multiple result sets

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql1);
...
int rows = stmt.executeUpdate(sql2);
stmt.close();
```

Note: PostgreSQL does not support fetch size but sends entire result

- **Process ResultSet**
  - Iterator-like cursor (app-level) w/ on-demand fetching
  - Scrollable / updatable result sets possible
  - Attribute access via column names or positions

```
ResultSet rs = stmt.executeQuery(
   “SELECT SID, LName FROM Students”);

List<Student> ret = new ArrayList<>();
while( rs.next() ) {
   int id = rs.getInt(“SID”);
   String name = rs.getString(“LName”);
   ret.add(new Student(id, name));
}
```

JDBC Prepared Statements

- **Execute `PreparedStatement`**
  - Use for precompiling SQL statements w/ input params
  - Inherited from Statement
  - **Precompile SQL once**, and execute many times w/ different parameters
  - → **Performance**
  - → **No danger of SQL injection**

```
PreparedStatement pstmt =
   conn.prepareStatement(
     “INSERT INTO Students VALUES(?,?)”);

for( Student s : students ) {
   pstmt.setInt(1, s.getID());
   pstmt.setString(2, s.getName());
   pstmt.executeUpdate();
}

pstmt.close();
```

- **Queries and Updates**
  - Queries → `executeQuery()`
  - Insert, delete, update → `executeUpdate()`

JDBC Callable Statements

- **Recap: (Stored Procedures, see 05 Query Languages (SQL))**
  - Can be **called standalone via CALL** <proc_name>(<args>);
  - Procedures return no outputs, but might have **output parameters**

- **Execute `CallableStatement`**
  - Create prepared statement for call of a procedure
  - Explicit registration of output parameters
  - Example

```
CallableStatement cstmt = conn.prepareCall(
   “{CALL prepStudents(?, ?)}”);

cstmt.setInt(1, 2019);
cstmt.registerOutParameter(2, Types.INTEGER);
cstmt.executeQuery();

int rows = cstmt.getInt(2);
```

Psycopg (Python PostgreSQL Adapter)

- **Overview Psycopg**
  - Implements **Python Database API Specification v2.0**
  - Call-level interface for dynamic SQL, very similar to JDBC

- **Establish Connection**

```
conn = psycopg2.connect(
    host=“localhost”, port=“5432”,
    database=“db1234567”, user=username,
    password=password)
```

- **Execute Statements**
  - Use local cursors

```
cur = conn.cursor()
cur.execute(“INSERT INTO Students VALUES(...)”)
```

- **Process Result Sets**

```
cur.execute(“SELECT SID, LName FROM Students”)
students = cur.fetchall()
for row in students:
    print(“SID = ”, row[0], end = ‘’)
    print(“Lname = ”, row[1])
```

Psycopg (Python PostgreSQL Adapter), cont.

- **Execute Prepared Statements**

```
cur = conn.cursor()
sql = “INSERT INTO Students VALUES(%s, %s)”
for s in students:
    cur.execute(sql, (s.getID(),s.getName()))
conn.commit()
```

- **Execute Callable Statement**
  - Result set
  - No output parameters

```
cur = conn.cursor()
cur.callproc(“prepStudents”, (2019, 2))
cur.fetchone()
```

- **Close Connection**

```
cur.close()
conn.close()
```

Preview Transactions

- **Database Transaction**
  - A transaction (TX) is a **series of steps** that brings a database from a **consistent state** into another (not necessarily different) **consistent state**
  - **ACID properties** (atomicity, consistency, isolation, durability)
  - See lecture **08 Transaction Processing and Concurrency**

- **Example**
  - Transfer 100 Euros from Account 107 to 999

```
START TRANSACTION ISOLATION LEVEL SERIALIZABLE;
   UPDATE Account SET Balance=Balance-100
      WHERE AID = 107;
   UPDATE Account SET Balance=Balance+100
      WHERE AID = 999;
COMMIT TRANSACTION;
```

- **Transaction Isolation Level**
  - **Tradeoff:** isolation (and related guarantees) vs performance
  - READ UNCOMMITTED (~~lost update~~, dirty read, unrepeatable read, phantom R)
  - READ COMMITTED (~~lost update~~, ~~dirty read~~, unrepeatable read, phantom R)
  - REPEATABLE READ (~~lost update~~, ~~dirty read~~, ~~unrepeatable read~~, phantom R)
  - SERIALIZABLE (~~lost update~~, ~~dirty read~~, ~~unrepeatable read~~, ~~phantom R~~)

JDBC Transaction Handling

- **JDBC Transaction Handling**
  - Isolation levels (incl NONE) and (auto) commit option
  - Savepoint and rollback (undo till begin or savepoint)
  - **Note:** TX handling on connection not statements

- **Beware of Defaults**
  - DBMS-specific default isolation levels

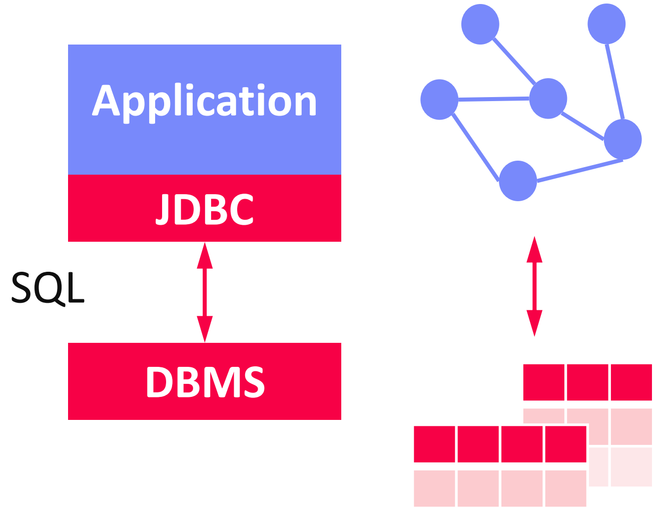
(SQL Standard: **SERIALIZABLE**, PostgreSQL: **READ COMMITTED**)

```
conn.setTransactionIsolation(
   TRANSACTION_SERIALIZABLE);
conn.setAutoCommit(false);

PreparedStatement pstmt = conn
   .prepareStatement(“UPDATE Account
   SET Balance=Balance+? WHERE AID = ?”);

Savepoint save1 = conn.setSavepoint();

pstmt.setInt(1,-100); pstmt.setInt(107);
pstmt.executeUpdate();

if( rand()<0.1 )
   conn.rollback(save1);

pstmt.setInt(1,100); pstmt.setInt(999);
pstmt.executeUpdate();

conn.commit();
```

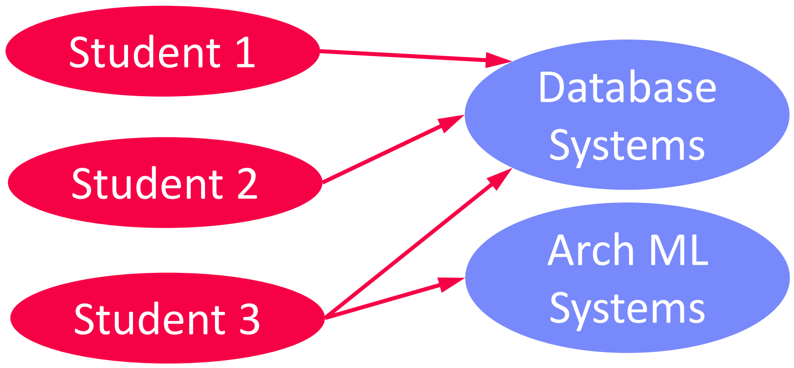
Object-Relational Mapping Frameworks

The “Impedance Mismatch” Argument

- **Problem Description**
  - Applications rely on **object-oriented programming languages** with hierarchies or graphs of objects
  - Data resides in **normalized “flat” tables** (note: ~~OODBMS~~, object-relational)
  - Application is responsible for **bridging this structural/behavioral gap**



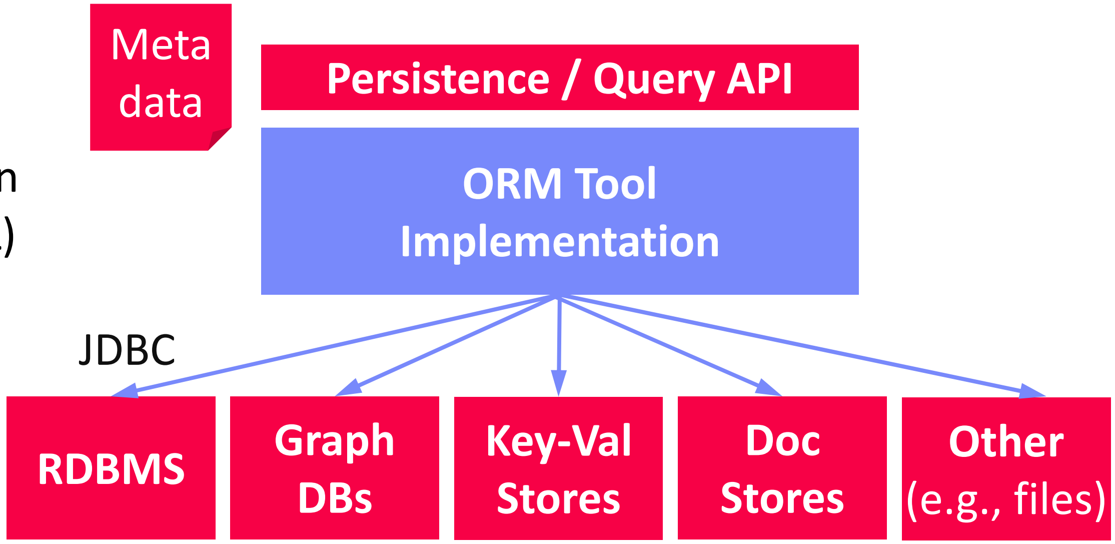
- **Example**
  - **SELECT** * **FROM** Students
  -
    ```
    SELECT C.Name, C.ECTS FROM
      Courses C, Attendance A
      WHERE C.CID = A.CID
         AND A.SID = 7;
    ```
  - … A.SID = 8;



Overview Object-Relational Mapping

- **Goals of ORM Tools**
  - Automatic **handling of object persistence lifecycle** and querying of the underlying data stores (e.g., RDBMS)
  - Reduced development effort ➔ **developer productivity**
  - Improved testing and independence of DBMS

- **Common High-Level Architecture**
  - **#1** Persistence definition (meta data → e.g., XML)
  - **#2** Persistence API
  - **#3** Query language / query API



Meta data

Persistence / Query API

ORM Tool Implementation

JDBC

RDBMS

Graph DBs

Key-Val Stores

Doc Stores

Other (e.g., files)

History and Landscape

- **History of ORM Tools** (aka persistence frameworks)
  - Since 2000 J2EE EJB **Entity Beans** (automatic persistence and TX handling)
  - Since 2001 **Hibernate** framework (close to ODMG specification)
  - Since 2002 **JDO** (Java Data Objects) via class enhancement
  - 2006 **JPA** (**Java Persistence API**), reference implementation **TopLink**
  - 2013 JPA 2, reference implementation **EclipseLink**
  - Late 2000s/early 2010s: **explosion of ORM alternatives, but criticism**
  - **2012 - today:** ORM tools just part of a much more diverse eco system

- **Example Frameworks**
  - http://java-source.net/open-source/persistence
  - Similar lists for .NET, Python, etc

JPA – Class Definition and Meta Data

- **Entity Classes**
  - **Define persistent classes** via annotations
  - Add details for IDs, relationship types, and specific behavior on updates
  - Some JPA implementations require enhancement process as post compilation step

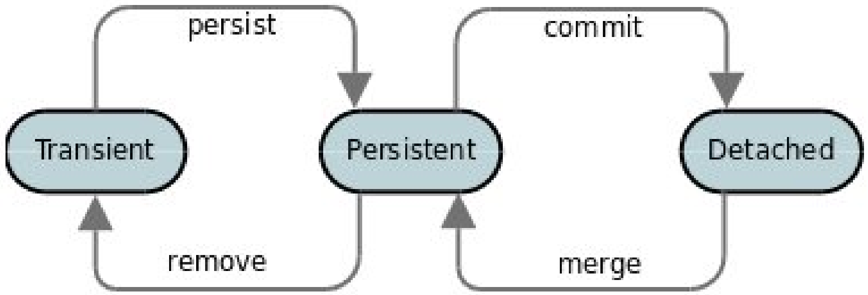
```
@Entity
public class Student {
    @Id
    private int SID = -1;
    private String Fname;
    private String Lname;
    @ManyToMany
    private List<Course> ...
}
```

- **Persistence Definition**
  - **Separate XML meta data** `META-INF/persistence.xml`
  - Includes connection details

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence
      xmlns="http://xmlns.jcp.org/xml/ns/persistence"
      xmlns:xsi=... xsi:schemaLocation=...>
   <persistence-unit name="UniversityDB">
      <class>org.tugraz.Student</class>
      <class>org.tugraz.Course</class>
      <exclude-unlisted-classes/>
      <properties> ... </properties>
   </persistence-unit>
</persistence>
```

JPA – Object Modification

- **CRUD Operations**
  - Insert by making objects persistent
  - Update and delete objects according to object lifecycle states
- **Lifecycle States**
  - Lifecycle state transitions via specific persistence contexts
  - Explicit and implicit transitions



[**Credit:** Data Nucleus, JPA Persistence Guide (v5.2), http://www.datanucleus.org/products/accessplatform/jpa/persistence.html#lifecycle]

```
EntityManager em = factory
   .createEntityManager();

tx.begin();

Student s = new
   Student(7,”Jane”,”Smith”);
s.addCourse(new Course(...));
s.addCourse(new Course(...));

em.persist(s);

tx.commit();
em.close
```

JPA – Query Languages

- **JPQL: Java Persistence Query Language**
  - SQL-like object-oriented query language
  - Parameter binding similar to embedded SQL

```
EntityManager em = factory
   .createEntityManager();
Query q = pm.createQuery(
   "SELECT s FROM Student s
      WHERE s.age > :age");
q.setParameter(“age”, 35);

Iterator iter = q
   .getResultList().iterator();
while( iter.hasNext() )
   print((Student)iter.next());
```

- **JPQL Criteria API**
  - JPQL syntax and semantics with a programmatic API

  -
    ```
    CriteriaQuery<Student> q = bld.createQuery(Student.class);
    Root<Student> c = q.from(Student.class);
    q.select(c).where(bld.gt(c.get(“age”), bld.parameter(...)));
    ```

- **Native SQL Queries**
  - Run native SQL queries if necessary

```
em.createNativeQuery(“SELECT *
   FROM Students WHERE Age > ?1”);
```

Jdbi (Java Database Interface)

[http://jdbi.org/]

- **Jdbi Overview**
  - Fluent API built on top of JDBC w/ same functionality exposed
  - Additional simplifications for row to object mapping

- **Example**

```
Jdbi jdbi = Jdbi.create(“jdbc:postgresql://.../db1234567”);
Handle handle = jdbi.open();

jdbi.registerRowMapper(Student.class, (rs, ctx)
  -> new Student(rs.getInt(“sid"), rs.getString(“lname"));

List<Student> ret = handle
  .createQuery("SELECT * FROM Students WHERE LName = :name")
  .bind(0, "Smith")
  .map(Student.class)
  .list();
```

A Critical View on ORM

- **Advantages**
  - **Simple CRUD operations** (insert/delete/update) and simple queries
  - **Application-centric development** (see boundary crossing)
- **Disadvantages**
  - **Unnecessary indirections** and complexity (meta data, mapping)
  - **Performance problems** (hard problem and missing context knowledge)
  - **Application-centric development** (schema ownership, existing data)
  - **Dependence** on evolving framework APIs
- **Sentiments** (additional perspectives)
  - Omar Rayward: Breaking Free From the ORM: Why Move On?, 2018 medium.com/building-the-system/**dont-be-a-sucker-and-stop-using-orms**-190add65add4
  - Vedra Bilopavlović: Can we talk about ORM Crisis?, 2018 linkedin.com/pulse/**can-we-talk-orm-crisis**-vedran-bilopavlovi%C4%87
  - Martin Fowler: ORM Hate, 2012 martinfowler.com/bliki/**OrmHate**.html

➔ **Awareness of strength and weaknesses / hybrid designs**

Conclusions and Q&A

- **Summary**
  - **Call-level Interfaces (ODBC/JDBC)** as fundamental access technology
  - **Object-Relational Mapping (ORM)** frameworks existing (**pros and cons**)
- **Exercise Reminder**
  - Exercise 2: Submission opened Apr 07, deadline: **Apr 28 11.59pm**
- **Next Lectures**
  - **07 Physical Design and Tuning** [Apr 27]
  - **08 Query Processing** [May 04]
  - **09 Transaction Processing and Concurrency** [May 11]