Databases

Lecture 10 -

Emily Lucia Antosch

HAW Hamburg

18.03.2025

Contents

1. Introduction

1.1 Where are we right now?

- Last time, we looked at the basics of subqueries and views
- Today, we'll be discussing
  - how we can expand our knowledge of views,
  - how we can use transactions to increase the safety of our data manipulation statements
  - how transactions are executed.

1.1 Where are we right now?

1. Introduction
2. Basics
3. SQL
4. Entity-Relationship-Model
5. Relationships
6. Constraints
8. **Subqueries & Views**
9. **Transactions**
10. Database Applications
11. Integrity, Trigger & Security

1.2 What is the goal of this chapter?

- At the end of this lesson, you should be able to
  - create views in PostgresQL and use them effectively and
  - use transactions to make safe changes, that can be undone if necessary.

2. Repetition

2.1 Transactions

ACID - Durability

- Once committed, changed data is safe
- Error types
  1. Computer failure
  2. Transaction or system error (constraint violation, $\frac{x}{0}$, blackout, system crash)
  3. Local Errors
  4. Concurrency control enforcement
  5. Disk error (harddisk broken)
  6. Physical problems and catastrophes (fire, earthquake, robbery, …)

2.1 Transactions

Distributed Transactions

2.2 Integrity, Trigger and Security Integrity Constraints

Integrity Constraints

- Static Constraints
  - Conditions on states
  - Conditions must be fulfilled before and after operations
  - Used until now
    - Primary Key
    - Foreign Key
    - `UNIQUE`, `NOT NULL`, `CHECK`
- Dynamic Constraints (**Assertions**)
  - Integrity conditions that affect multiple tables
  - Conditions on state transitions

2.2 Integrity, Trigger and Security Integrity Constraints

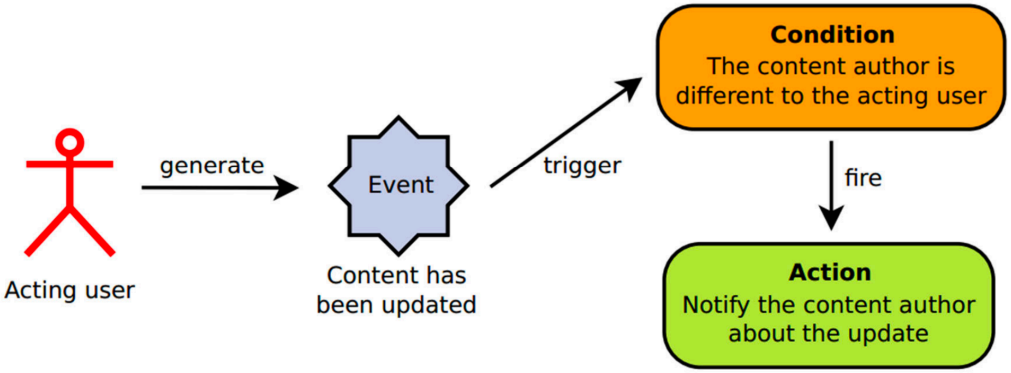
Example

status of order ⟶ new ⟶ payed ⟶ processing ⟶ shipped

2.2 Integrity, Trigger and Security Integrity Constraints

Integrity Constraints - ECA

- ECA rules
  - on event (E)
  - under certain conditions (C)
  - perform actions (A)



2.2 Integrity, Trigger and Security Integrity Constraints

Extension Delimiter: Example

SQL

```sql
delimiter |
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
    BEGIN
        IF NEW.SALARY > (SELECT SALARY
                         FROM EMPLOYEE
                         WHERE SSN = NEW.SUPER_SSN )
        THEN SET NEW.Salary = (SELECT SALARY
                               FROM EMPLOYEE
```

2.2 Integrity, Trigger and Security Integrity

Co

```
11                                          WHERE SSN =
   NEW.SUPER_SSN )-1;
12          END IF;
13  END;
14  |
15  delimiter;
```

2.2 Integrity, Trigger and Security Integrity Constraints

- Time of execution, relative to event
  - `BEFORE`
  - `AFTER`
- `INSTEAD OF`
- Statement trigger
  - Once per statement
  - Even if no row is affected!
  - Default trigger type
- Row trigger

2.2 Integrity, Trigger and Security Integrity Constraints

- For every affected row
  - Syntax: `FOR EACH ROW`

2.2 Integrity, Trigger and Security Integrity Constraints Transition Variables

- Row triggers can access old and new tuples
  - PostgresQL
    - `:old` or `old` ⟶ `NULL` for `INSERT`
    - `:new` or `new` ⟶ `NULL` for `DELETE`
- Oracle
  - `NEW` and `OLD`
  - Before row triggers:
    - Can even modify new!

2.2 Integrity, Trigger and Security Integrity Constraints Permissions

- DBMS are multi-user systems
- You need permissions to do anything with the DB:
  - login
  - `CREATE` table, `DROP` table, etc.
  - `SELECT`
  - `INSERT`, `UPDATE`, `DELETE`
- Permissions can be `GRANT`ed and `REVOKE`d

2.2 Integrity, Trigger and Security Integrity Constraints

GRANT and REVOKE

- Permissions can be GRANTed and REVOKEd

```sql
GRANT <privilege_name> ON <object_name>
TO { <user_name> | PUBLIC | <role_name>} [ WITH GRANT OPTION ] ;
```

- GRANT

```sql
GRANT SELECT ON tab_a TO user_a;
GRANT UPDATE ON tab_b TO user_a;
```

- REVOKE

2.2 Integrity, Trigger and Security Integrity

Cons

```sql
REVOKE SELECT ON tab_a FROM user_a;
```

SQL

3. Database Applications

Problems

- Cannot solve every problem with SQL
  - No loops
- Recursion not widely implemented
- Need to query DB out of an application

> 💡 **Idea**
>
> Combination with procedural or object-oriented programming languages (**host languages**)

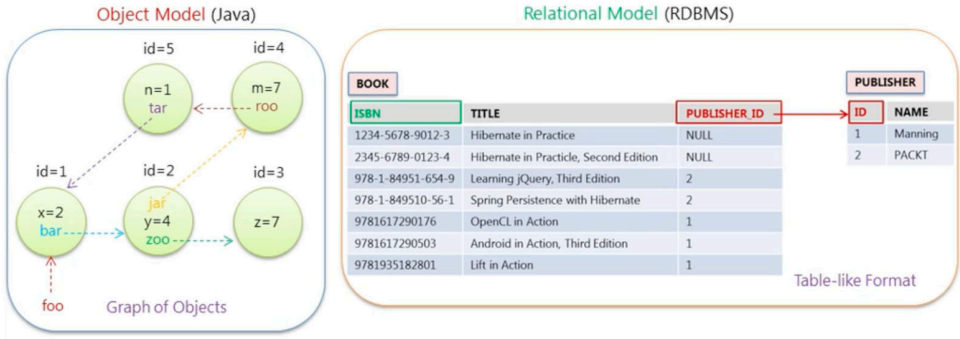
Combining SQL with 3GL

1. **Embed SQL commands into host language**
   - Embedded SQL, SQL/OLB
2. **SQL commands through API calls**
   - SQL: Call Level Interface (CLI)
   - ODBC, JDBC
3. **Extend SQL**
   - SQL: Persistent Stored Modules (SQL/PSM)
   - Oracle: PL/SQL

3.1 Basics

Impedance Mismatch

- E.g., Object–relational impedance mismatch
  - Object-oriented concepts, like inheritance in OO, polymorphism in OO,…
    - Data type differences, like pointers in OO,…
    - Structural and integrity differences, like constraints in RM, objects



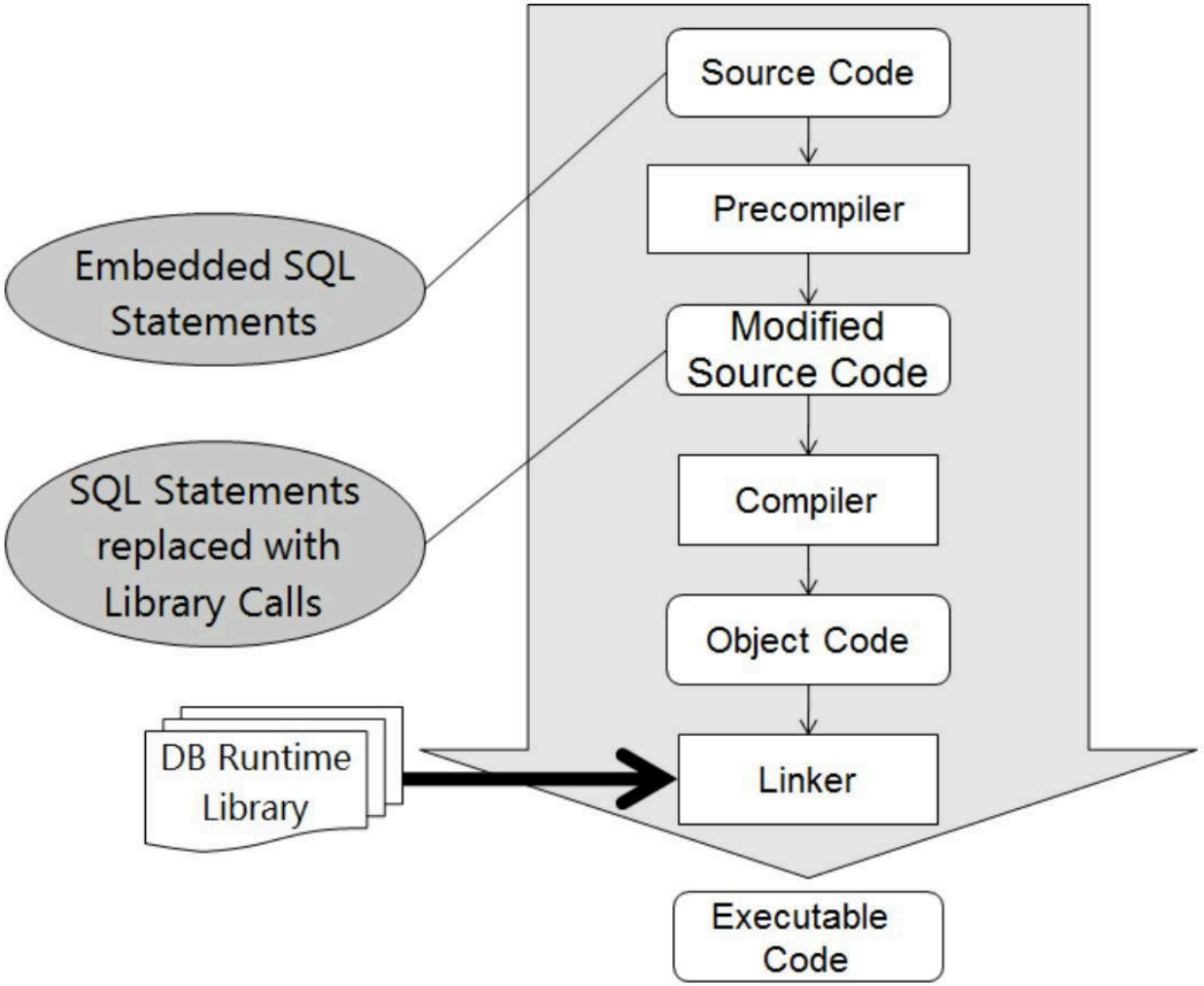
can be composed of other objects in OO, …

- Transactional differences, like transactions in RM
- Manipulative differences, like declarative querys in RM

3.1 Basics

Embedded SQL

3.1 Basics


Source Code
Precompiler
Embedded SQL Statements
Modified Source Code
SQL Statements replaced with Library Calls
Compiler
Object Code
DB Runtime Library
Linker
Executable Code

3.1 Basics

SQL in C

```c
int main() {
    exec sql begin declare section;
    int sv_new_price;
    int sv_isbn;
    exec sql end declare section;
    printf("Please enter ISBN: \n ");
    scanf("%d", &sv_isbn);
    printf("Please enter new_price: \n");
    scanf("%d", &sv_new_price);
    exec sql update book
```

```
11              set price = :sv_new_price
12              where isbn = :sv_isbn;
13  } Shared variables
```

SQL in Java

Java

```java
int maxSalary, avgSalary;

#sql{
SELECT MAX(SALARY) , AVG(SALARY)
        INTO :maxSalary , :avgSalary
        FROM EMPLOYEE
};
```

Embedded SQL

- Mainly static SQL
  - SQL statement is fixed
  - SQL syntax is checked at (pre-)compile time
- Exchange data with application by **host variables** (`:varname`)
- Precompilers exist for many languages
  - C/C++, Java (SQLJ), Ada, Cobol, Fortran, PL1, …

API Calls

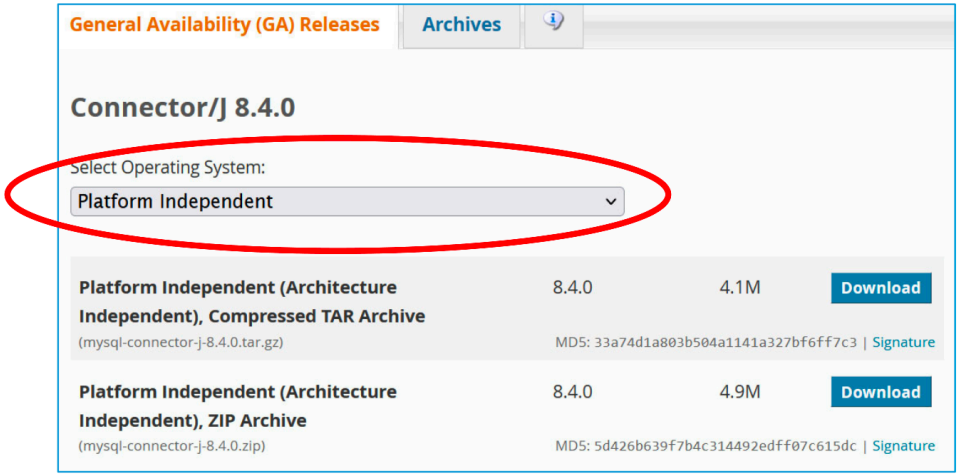
- SQL commands through library/API calls
- Dynamic SQL
  - Application can dynamically set up the SQL command string
  - SQL syntax is checked at runtime
- Standard SQL: Call Level Interface (CLI), e.g.,
  - ODBC (for any language like C,C++,Java, but restricted on MS Windows)
  - JDBC (for Java, can be used for any platform)
  - OCI (Oracle Call Interface)

API Calls - JDBC

- JDBC: Java Database Connectivity
- Part of Java API
- Typical steps:
  1. Load JDBC driver
  2. Define DB connection URL
  3. Connect to DB
  4. Create command object
  5. Execute command
  6. Process result
  7. Cleanup: Close resources and DB connection

API Calls - JDBC: Preparation

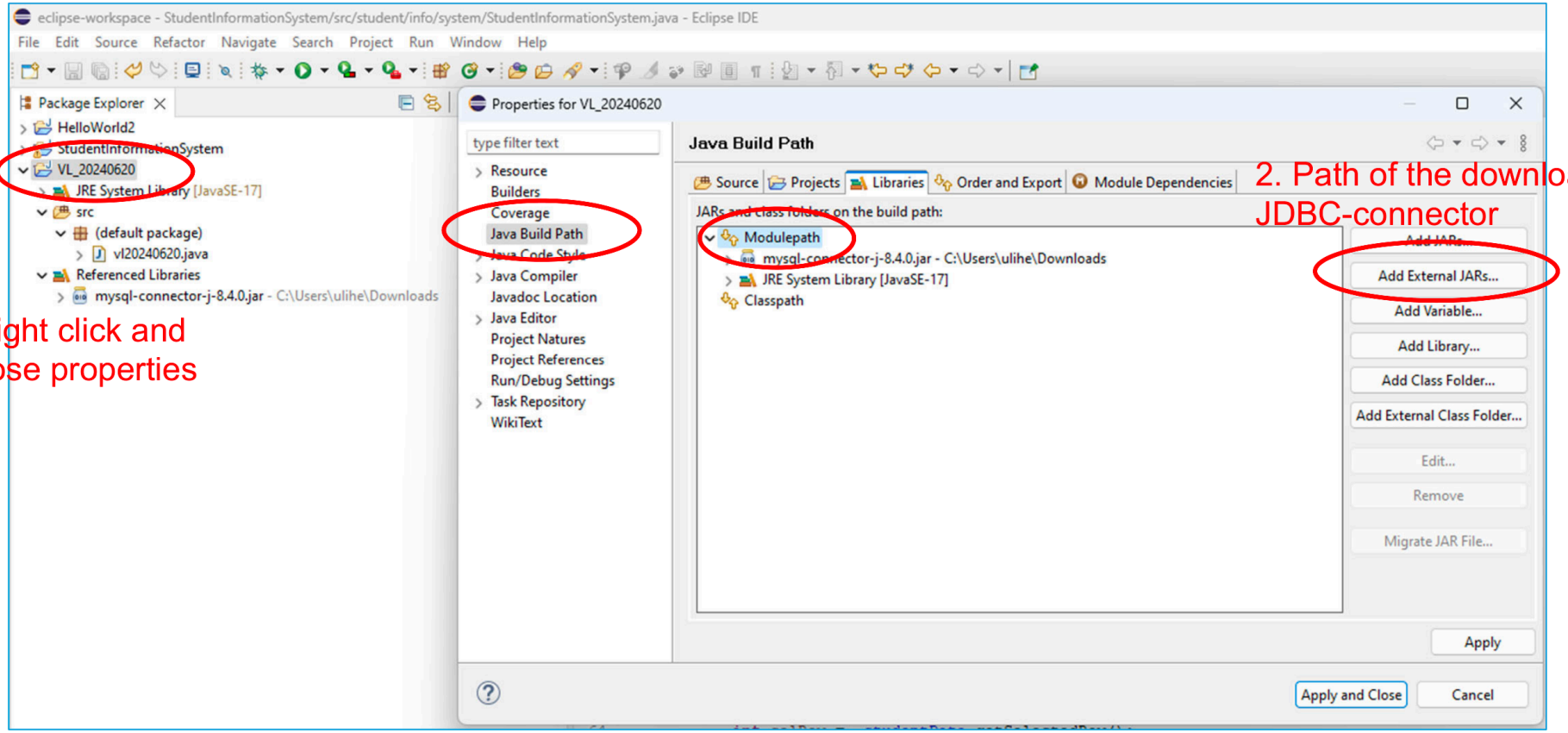
- Download JDBC Connector:
  - Oracle: http://java.sun.com/products/jdbc/download.html
  - MySQL: https://dev.mysql.com/downloads/connector/j/
- Prepare a Java Project, e.g. in Eclipse
- Prepare a MySQL database



3.1 Basics

- Import the JDBC library

API Calls - JDBC: Preparation

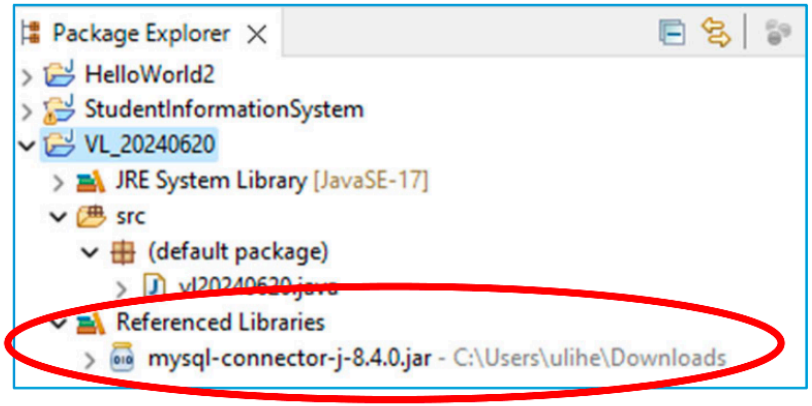


1. Right click and choose properties

2. Path of the downloaded JDBC-connector

API Calls - JDBC: Preparation

- Download JDBC Connector:
  - Oracle: http://java.sun.com/products/jdbc/download.html
  - MySQL: https://dev.mysql.com/downloads/connector/j/
- Prepare a Java Project, e.g. in Eclipse
- Prepare a MySQL database



You can see if include was successful

3.1 Basics

- Import the JDBC library

API Calls - JDBC

```java
Class.forName(driverName);

import driverName;
```

Example

Java

```java
import java.sql.Connection;
import java.sql.driverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

API Calls - JDBC

- Connection is defined by an URL
  - Oracle
    - `jdbc:oracle:thin:@<server>:1521:<dbname>`
    - For example, `Oracle@HAW` (available before the cyber attack): `jdbc:oracle:thin:@ora14:informatik.haw-hamburg.de:1521:inf14`
  - MySQL
    - `jdbc:mysql://<server>/<dbname>`
    - For example: `jdbc:mysql://localhost:3306/company`

3.1 Basics

```java
private static final String CONN = "jdbc:mysql://localhost:3306/company_2024";
```

API Calls - JDBC

- Exkursion: `localhost`
  - In computer networking, `localhost` is a hostname that refers to the current computer used to access it. It is used to access the network services that are running on the host via the loopback network interface. Using the loopback interface bypasses any local network interface hardware.
  - The local loopback mechanism may be used to run a network service on a host without requiring a physical network interface, or without making the service accessible from the networks the computer may be connected to. For example, a

locally installed website may be accessed from a Web browser by the URL http://localhost to display its home page.

- The name `localhost` normally resolves to the IPv4 loopback address `127.0.0.1`, and to the IPv6 loopback address ::1. (Ipv stands for Internet Protocol version)

API Calls - JDBC

```java
Connection conn =
DriverManager.getConnection(url, user, psw);
```

Java

Example

```java
myConn = DriverManager.getConnection(CONN,
USER, PASSWORD);
```

Java

- Info about the connection has become available:

3.1 Basics

```java
conn.getMetaData();
```

API Calls - JDBC

- Obtain Statement object

```java
Statement st = conn.createStatement() ;
```

Also: `prepareStatement()`, `prepareCall()`

API Calls - JDBC

- Execute query

```java
String query = "SELECT dnumber, dname FROM DEPARTMENT";
// No ";" in query string
ResultSet myRes2 = myStmt2.executeQuery(query);
ResultSet myRes = myStmt.executeQuery("SELECT lname, fname FROM EMPLOYEE");
```

Java

- Also: `executeUpdate()` For `INSERT`, `UPDATE`, `DELETE`, `CREATE`

API Calls - JDBC

Java

```java
while(cursor.next()) {
            // position in cursor starts at 1 !
            string s1 = cursor.getString(1) ;
            int i2 = cursor.getInt (2) ;
            System.out.println (s1) ;
            System.out.println (i2) ;
}
while(myRes.next()) {
            System.out.println(i+". Person:
"+myRes.getString("fname"));i++;
```

```
10  }
```

API Calls - JDBC

- Important!
- Connections, Statements, ResultSets, etc. hold resources
- Both locally and on the server!
- So: `close()` them as soon as possible
  - After an error, too!

```java
finally {
        cursor.close () ;
         st.close () ;
        conn.close () ;
```

```
5  }
```

3.1 Basics

API Calls - JDBC

How to build a SQL statement programmatically?

API Calls - JDBC

- Problem: use parameters in SQL query

```java
Statement st = conn.createStatement();
```

```java
String query = "SELECT id FROM books WHERE title = '" + name + "'";
```

- Problem 1:
  - `name = "O'Reilly";`

API Calls - SQL Injection

- If there is nothing to prevent a user from entering “wrong” input, the user can enter some “smart” input like this:

```SQL
SELECT UserId , Name , Password FROM Users WHERE UserId = 105 OR 1=1;
```

- Problem 2:
  - SQL injection attacks

API Calls

- One possible Solution: use `PreparedStatement`

```java
string name = "O'Reilly" ;
string query = "SELECT id FROM tab WHERE name=?" ;
        // no quotes ( ' ' ) here !
PreparedStatement pst =
conn.prepareStatement(query) ;
pst.setString (1 , name) ;
ResultSet cursor = pst.executeQuery () ;
```

API Calls

- Classes/Interfaces in package java.sql.*
  - DriverManager
  - Connection
    - DatabaseMetaData
    - Statement, PreparedStatement, CallableStatement
    - ResultSet
    - ResultSetMetaData
    - SQLException (for error handling)

API Calls - Transaction Handling

- Transaction syntax:

```java
connection.setAutoCommit(false);
connection.commit();
connection.rollback();
```

- If you need to change the isolation level, here is the syntax:

```java
connection.setTransactionIsolation(level);
```

API Calls - Error Handling

- `java.sql.SQLException`
- `getMessage()`: retrieve error text
- `getStatus()`: XOPEN or SQL status
- `getErrorCode()`: vendor-specific error code
- Problem: application needs to know vendor's error codes!
  - Problem with connection to DB
  - SQL syntax wrong
  - Constraint violation
  - …

API Calls - Antipatterns

- Do not build SQL string using user input!

```java
Statement st = conn.createStatement();
String query = "SELECT id FROM books WHERE title = '" + name + "'";
```

- Problems
  - Correct quoting
  - Need to handle special characters like ‘&’
  - Opens the door for SQL injection attacks

! Memorize

Always use `PreparedStatement` / parameter binding!

API Calls - Antipatterns

- Do not read whole `ResultSet` into RAM
  - Problem: `ResultSet` can get huge
  - Solution: Iterate through the `ResultSet`
- Do not forget to `close()` resources
  - Problem: Resources are held on client and server!
- Do not implement selection in client code
  - Problem: `ResultSet` can get huge
  - Solution: Use `WHERE` clause in SQL

API Calls - Beyond JDBC

- Frameworks on top of JDBC
- spring-jdbc
- Object-Relational Mapping (ORM)
  - Hibernate
    - www.hibernate.org
    - Mapping is defined in XML configuration files
    - `< one - to - many >`, `< many - to - many >`, …
    - Can generate DDL out of classes + mapping
  - Different approach: Conventions
    - used by Ruby on Rails (non-Java)

Extend SQL

- The previous approaches of connecting programming languages with DBMS are very fine granular (only one operation at a time)
- Problem: The DBMS cannot optimize because it doesn't know which operation is next
- Base idea: **Extend SQL by control structures**
  - Putting the application code at the DBMS not at the programming language
- SQL-extensions were former DBMS-specific and called "Stored Procedure"

- Now they are standardized in SQL-99 and called SQL/PSM (persistently stored modules) and therefore over different DBMS useable (e.g., PL/SQL for Oracle)

Extend SQL - PL/SQL

- To structure the PL/SQL programs, it's possible to define **procedures** and **functions** and reuse them
  - A procedure uses parameters like OUT or IN OUT parameters to get the results A procedure may return one or more values through parameters or may not return at all
  - A function must return a value (of any type) by default definition
- Function can be used in SQL statements, procedures cannot be used in SQL statements

Extend SQL - PL/SQl

- Syntax for creating procedures:

```SQL
CREATE PROCEDURE procedure_name
      ( parameter1 IN parameter_type1 ,
      ( parameter2 OUT parameter_type2 ,
      …
      ( parameterN IN OUT parameter_typeN) IS
  <PL/SQL-Block>
```

)

Extend SQL - PL/SQL

- Syntax for creating functions:

```SQL
CREATE FUNCTION function_name
(parameter1 parameter_type1 ,
(parameter2 parameter_type2 ,
…
(parameterN parameter_typeN )
RETURN result_type IS
<PL/SQL-Block>
```

Extend SQL - PL/SQL

- Using variables and defining data types

```SQL
1  declare
2  today date;
3  type PersonRecordType is record ( PersonName varchar2
   ( 50 ) ; BirthDate date ) ;
4  employee PersonRecordType;
```

- Cursor for processing results:

```SQL
1  cursor CurBook is
```

```
2  SELECT isbn , title FROM Books ;
```

Extend SQL - PL/SQL

- As control flow structures PL/SQL provides
  - sequence (by “; “)
  - condition (where the else branch is optional)

```SQL
1  if <condition > then
2  < PL/SQ-operation >
3  else
4  <PL/SQL-operation>
5  end if ;
```

Extend SQL - PL/SQL

- Loops (for, while, loop)

```sql
while < condition >
loop
< PL/SQL-operation >
end loop ;
```

- Executing a relation with infinite loop

```sql
loop
fetch Book into BookRecord;
```

```
3  exit when Book%not found ;
4  ...
5  end loop ;
```

Extend SQL - Example

SQL

```
1   delimiter |
2   CREATE PROCEDURE IF NOT EXISTS
3       output(in ssn char(9), in old_sal DECIMAL(10,2),
4       in new_sal DECIMAL(10,2), in diff_sal DECIMAL(10,2))
5   BEGIN
6       INSERT INTO EMPLOYEE_SALDIFF VALUES
7       ( ssn , old_sal , new_sal, diff_sal);
8   END|
9   delimiter ;
10
```

```
11  CALL output(123456789, 12.34, 56.78, 44.44);
```

Extend SQL - Example

```
1   Example: MySQL                                                    SQL
2       delimiter |
3     CREATE TRIGGER IF NOT EXISTS Print_salary_changes
4     BEFORE UPDATE ON EMPLOYEE
5     FOR EACH ROW
6           BEGIN
7             DECLARE sal_diff DECIMAL(10,2);
8             IF (NEW.salary != OLD.salary)
9             THEN
10                  BEGIN
```

```
11                          SET sal_diff = NEW.salary -
OLD.salary ;
12                          CALL output(NEW.ssn, OLD.salary,
NEW.salary, sal_diff);
13                      END;
14                END IF;
15      END;
16      |
17      delimiter ;
```

Extend SQL - PL/SQL

- Additional to the structuring, functions/procedures have more advantages:
  - DBMS can optimize the code because it knows the structure
  - The execution takes place on the DBMS-server, so network overhead is minimized, which is especially useful in distributed environments (client/server or internet)
  - Assignment of permissions are available for procedures
  - Procedures can be used to full integrity constraints

Extend SQL - PL/SQL

- Disadvantages
  - Software development environments (IDE) are often not optimal
  - Raised dependency on DBMS
  - Problems on scalability, because application code is executed on DBS-server instead of being executed by many clients or application servers

Extend SQL - Java Stored Procedures

- Formulating Stored procedures in Java is possible in many DBMS
- Oracle supports the execution of Java programs directly on the server
- Java programs with GUI are excluded
- Access by wrapping Java methods in PL/SQL
- The mapping of PL/SQL call on Java method must be created by the programmer

- These mapped Java methods can be accessed by all DML operations (Select, Update, Insert, Delete) and within PL/SQL blocks

3.2 Summary

- Embed SQL commands into host language
  - Advantages +
    - Query is part of source code
      - syntax checking
      - validation against the database schema
      - readable
    - Disadvantages –
      - Static queries
      - Changes of queries go through recomplication process

3.2 Summary

- SQL commands through API calls
  - Advantages +
    - More flexibibilty
      - Queries can be generated at runtime
    - Disadvantages –
      - More complex programing
      - No checking during compile time

3.2 Summary

- Extend SQL
  - Advantages +
    - No suffering from impedance mismatch problem
    - Disadvantages −
      - New language for the programmer

4. License Notice

4.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License.
- https://creativecommons.org/licenses/by-nc-sa/4.0/
- This work is based off of the work by Prof. Dr. Ulrike Herster.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.