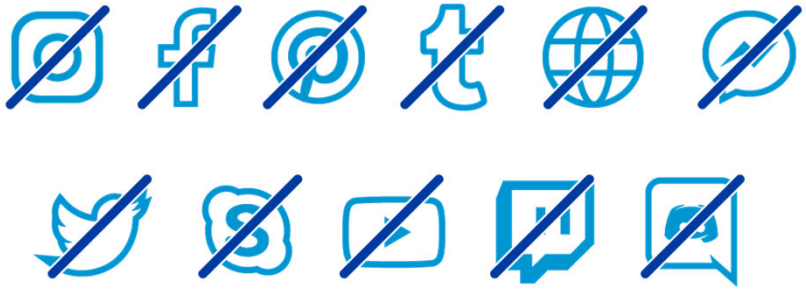
DATABASES

Source: https://en.itpedia.nl/2017/11/26/wat-is-een-database/

Prof. Dr. Ulrike Herster
Hamburg University of Applied Sciences

HAW HAMBURG

COPYRIGHT



The publication and sharing of
slides, images and sound recordings of this
course is not permitted

TRANSACTIONS
ACID - DURABILITY

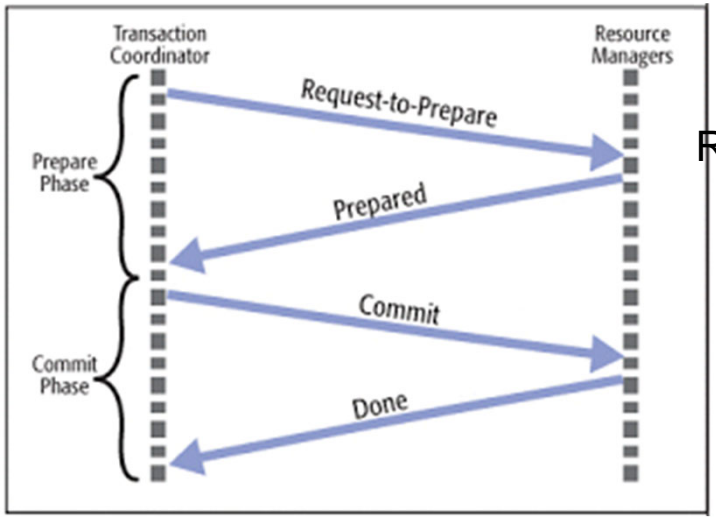
Repetition

- 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, ...)

Source: Elmasri, Fundamentals of Database Systems, Page 750ff

TRANSACTIONS
DISTRIBUTED TRANSACTIONS



Repetition

- To ensure interoperability between the participating resource managers the *2-phase commit protocol* is realized

- It defines the final synchronization of different parts of a transaction of a global transaction
- In the first phase the transaction manager asks participating resource managers to announce the results of their local transaction part
- This leads to a global result (commit or rollback) that is then in the second phase announced to the participants

Source: https://medium.com/@balrajasubbiah/consensus-two-phase-and-three-phase-commits-4e35c1a435ac

ORGANIZATION
OUR JOURNEY IN THIS SEMESTER

Repetition

- **Integrity, Trigger & Security**
- Database Applications
- Transactions
- Subqueries & Views
- More SQL
- Notations & Guidelines
- Constraints
- Relationships
- Simple Entities and Attributes
- Basics

Source: Foto von Justin Kauffman auf Unsplash

INTEGRITY, TRIGGER & SECURITY
INTEGRITY CONSTRAINTS

Repetition

- Static Constraints
  - Conditions on states
  - Conditions must be fulfilled before and after operations
  - Used until now
    - Primary Key
    - Foreign Key
    - `UNIQUE`, `NOT NULL`, `CHECK`

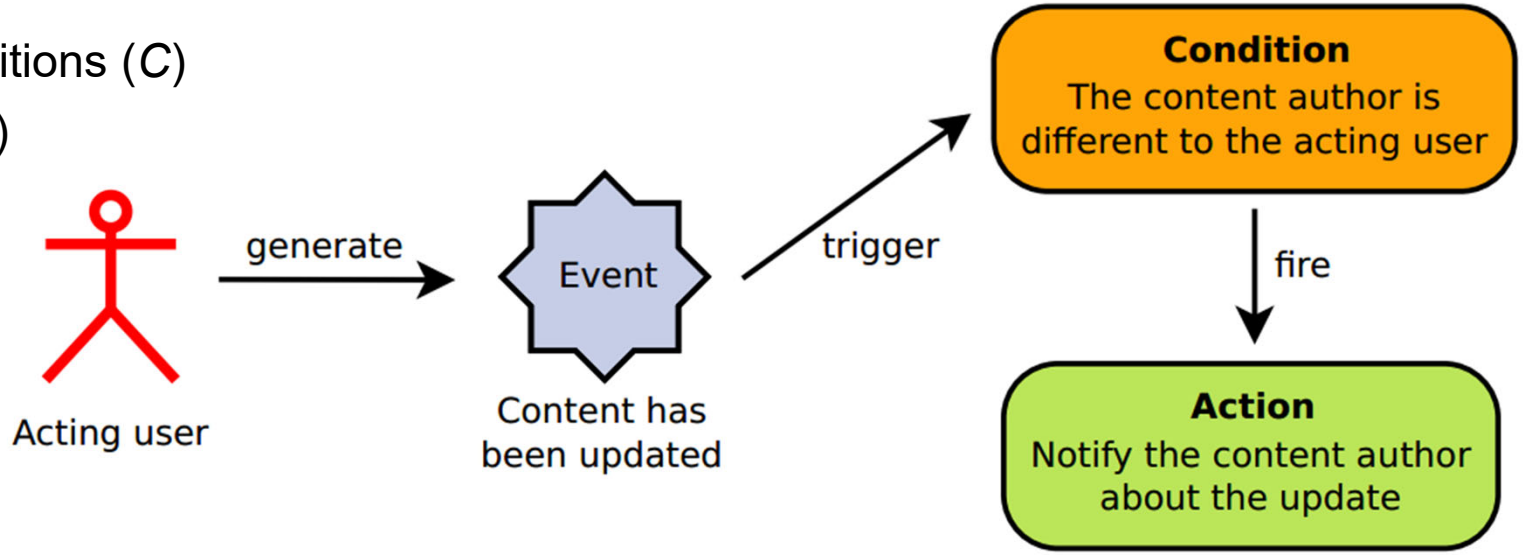
- Dynamic Constraints (*Assertions*)
  - Integrity conditions that affect multiple tables
  - Conditions on state transitions
    → Example: status of order

    new → payed → processing → shipped

INTEGRITY, TRIGGER & SECURITY
TRIGGER – ECA RULE

Repetition

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



Quelle: https://dev.acquia.com/blog/drupal-8-module-of-the-week/drupal-8-module-of-the-week-rules/15/06/2016/15681

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN MYSQL

Repetition

What is the goal of this trigger?

```
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
                             WHERE SSN = NEW.SUPER_SSN )-1;
      END IF;
END;
|
delimiter ;
```

Source: Elmasri, Fundamentals of Database Systems

INTEGRITY, TRIGGER & SECURITY
TRIGGER – TYPES

Repetition

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

`INSTEAD OF` for views not supported by mySQL!

INTEGRITY, TRIGGER & SECURITY
TRIGGER – TRANSITION VARIABLES

Repetition

- Row triggers can access old and new tuples
  - MySQL
    - :old or old → `NULL` for **`INSERT`**s
    - :new or new → `NULL` for **`DELETE`**s
  - Oracle
    - NEW and OLD

- Before row triggers:
  - Can even modify new!

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – BASICS

Repetition

- 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

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – GRANT AND REVOKE

Repetition

- Permissions can be **GRANT**ed and **REVOKE**d

- Syntax:

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

- Example: **GRANT**

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

- Example: **REVOKE**

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

ORGANIZATION
OUR JOURNEY IN THIS SEMESTER

- Integrity, Trigger & Security
- **Database Applications**
- Transactions
- Subqueries & Views
- More SQL
- Notations & Guidelines
- Constraints
- Relationships
- Simple Entities and Attributes
- Basics

Source: Foto von Justin Kauffman auf Unsplash

DATABASE APPLICATIONS BASICS

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

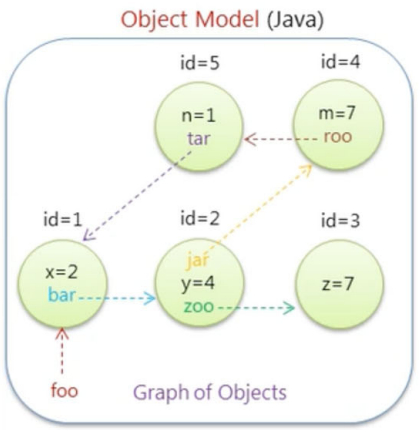
→ Solution: Combination with procedural or object-oriented programming languages (*host languages*)

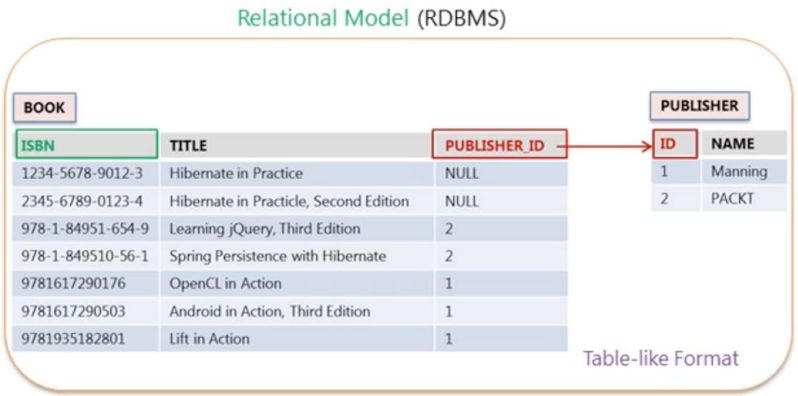
DATABASE APPLICATIONS
BASICS - COMBINING SQL AND 3GL

Options:

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

DATABASE APPLICATIONS BASICS – COMMON PROBLEMS

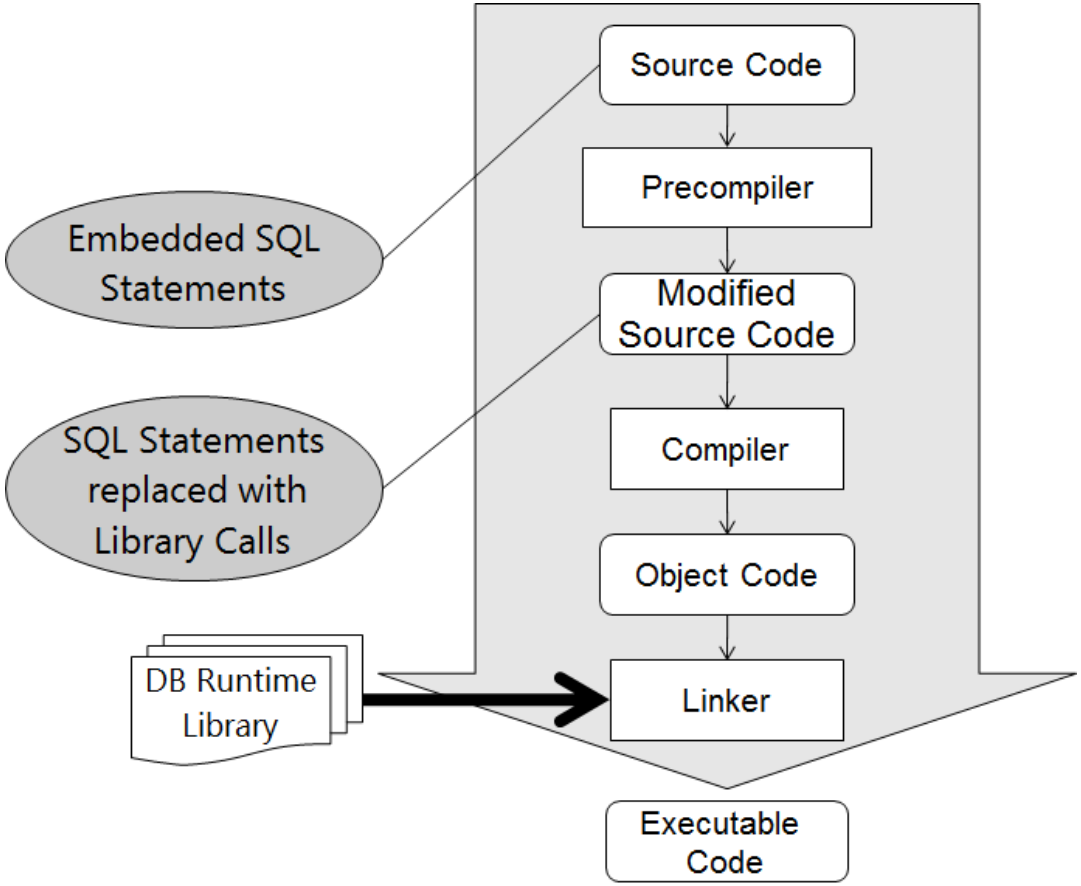




- "*Impedance Mismatch*"
  - E.g., Object–relational impedance mismatch
    - Object-oriented concepts
      E.g., inheritance in OO, polymorphism in OO,…
    - Data type differences
      E.g., Pointers in OO,…
    - Structural and integrity differences
      E.g., constraints in RM, objects can be composed of other objects in OO, …
    - Transactional differences,
      E.g., transactions in RM
    - Manipulative differences
      E.g., declarative querys in RM

Source: https://www.youtube.com/watch?v=wg-NCF5KXNk
https://walkingtechie.blogspot.com/2017/12/object-relational-impedance-mismatch.html
https://en.wikipedia.org/wiki/Object%E2%80%93relational_impedance_mismatch

DATABASE APPLICATIONS
1. EMBEDDED SQL



DATABASE APPLICATIONS
1. EMBEDDED SQL - EXAMPLE: SQL EMBEDDED INTO C (FRAGMENT)

```
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
         set price = :sv_new_price
         where isbn = :sv_isbn;
}
```

Shared variables

DATABASE APPLICATIONS
1. EMBEDDED SQL - EXAMPLE: SQLJ SNIPPET

```
int maxSalary, avgSalary;

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

DATABASE APPLICATIONS
1. 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, ...

DATABASE APPLICATIONS
2. 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)

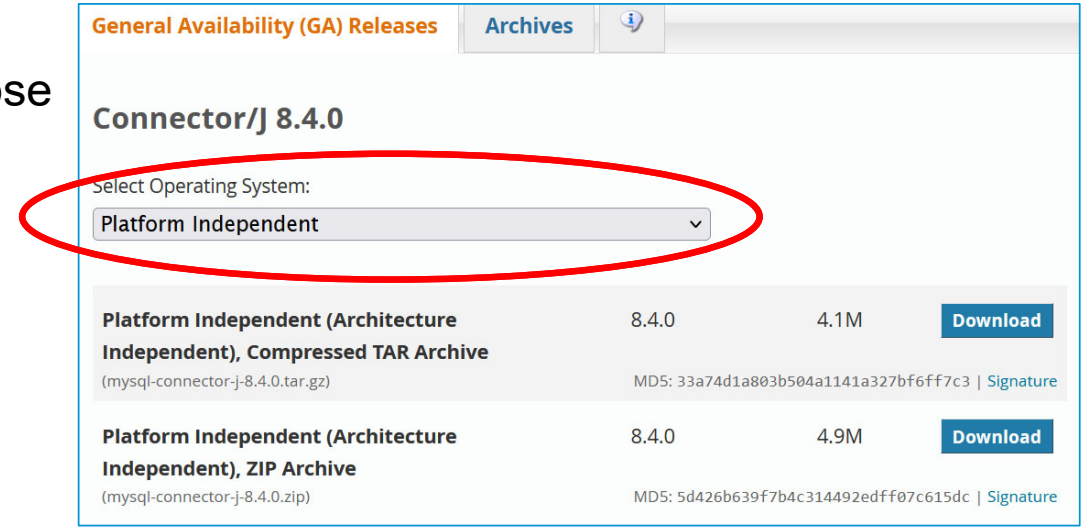
DATABASE APPLICATIONS
2. API CALLS - JDBC

- JDBC: Java Database Connectivity
- Part of Java API

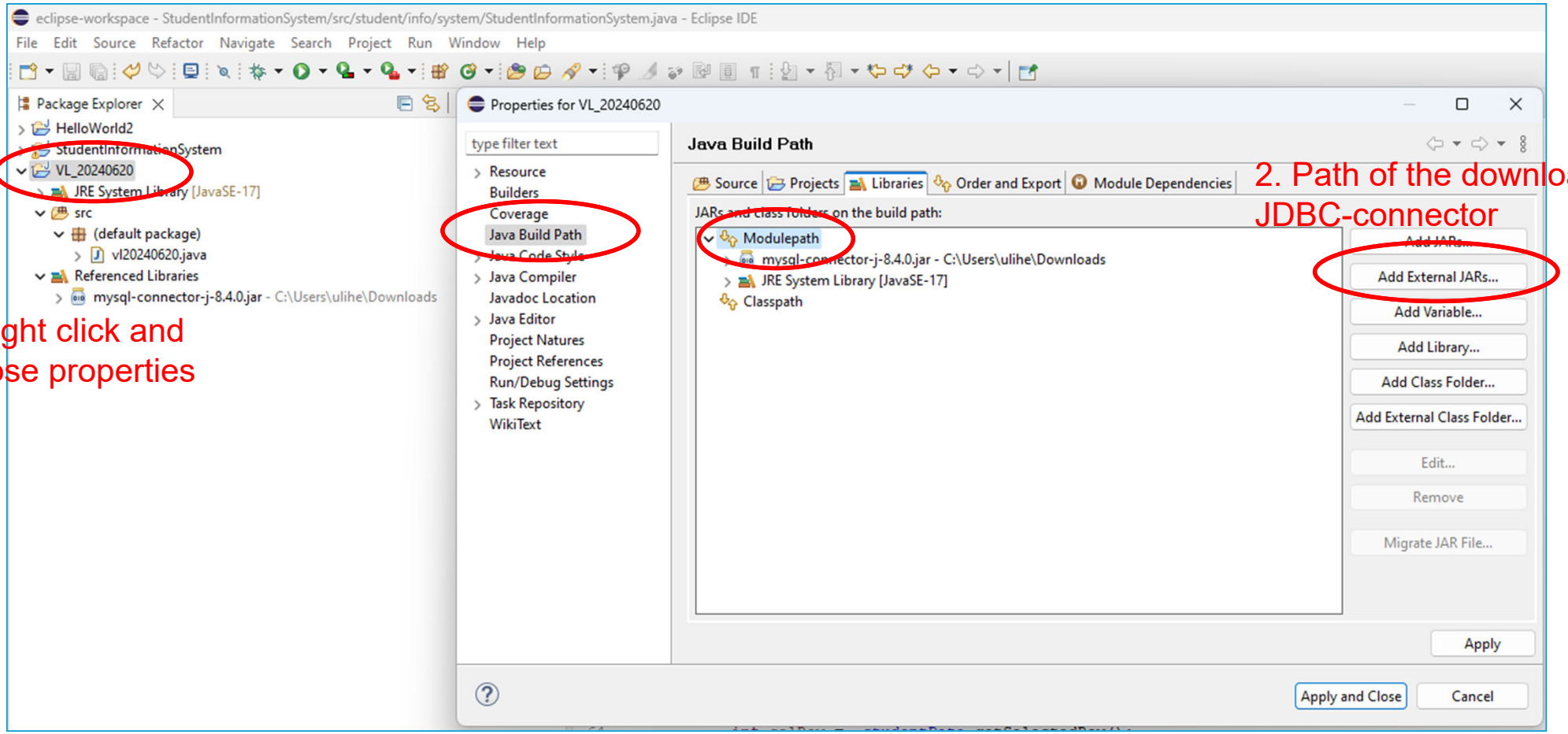
- Typical steps:
  1. Load JDBC driver
  2. Define DB connection URL
  3. Connect to DB
  4. Create command object
  5. Execute command
  6. Process result
  7. Cleanup: Close resources and DB connection

DATABASE APPLICATIONS
2. 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
- Import the JDBC library



DATABASE APPLICATIONS
2. API CALLS – JDBC: PREPARATION, E.G., MYSQL



1. Right click and choose properties

2. Path of the downloaded JDBC-connector

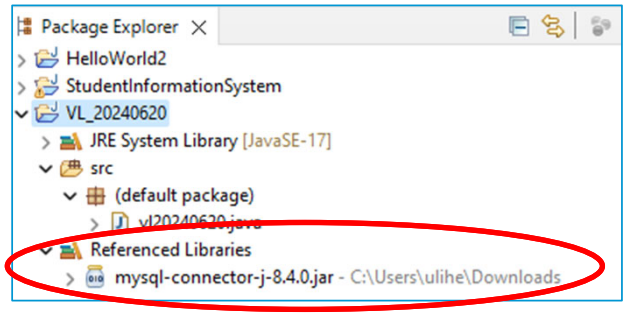
DATABASE APPLICATIONS
2. API CALLS – JDBC: PREPARATION, E.G., MYSQL

- 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

- Import the JDBC library



You can see if include was successful

DATABASE APPLICATIONS
2. API CALLS – JDBC: 1. LOAD JDBC DRIVER

- Syntax:

  `Class.forName(driverName);`
  or
  `import driverName;`
  plus create an instance

- Example:

```
// 1. Load JDBC driver
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

DATABASE APPLICATIONS
2. API CALLS – JDBC: 2. DEFINE DB CONNECTION URL

- 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"`

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: 2. DEFINE DB CONNECTION URL

- 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)

Source: https://en.wikipedia.org/wiki/Localhost

DATABASE APPLICATIONS
2. API CALLS – JDBC: 3. CONNECT TO DB

- Syntax:
  `Connection conn = DriverManager.getConnection(url,user,psw) ;`
- Example:

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

- Info about the connection is now available
  - Example:
    `conn.getMetaData () ;`

DATABASE APPLICATIONS
2. API CALLS – JDBC: 4. CREATE COMMAND OBJECT

- Obtain Statement object
  - Example
    ```
    Statement st = conn.createStatement() ;
    ```

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: 5. EXECUTE COMMAND

- Execute query
  - Example

```
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");
```

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: 6. PROCESS QUERY RESULTS

Example:

```
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++;
}
```

DATABASE APPLICATIONS
2. API CALLS – JDBC: 7. CLEANUP

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

- Syntax:

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

How to build a SQL statement programmatically?

DATABASE APPLICATIONS
2. API CALLS – JDBC: PARAMETER BINDING

- Problem: use parameters in SQL query
- Syntax

```
Statement st = conn.createStatement ( ) ;
String query
    "SELECT id FROM                    ='"+name+"'" ;
ResultSetcurso                .executeQuery (q          ) ;
```

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: SQL INJECTION

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

- Syntax

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

- Problem 2:
  - SQL injection attacks

DATABASE APPLICATIONS
2. API CALLS – JDBC: SQL INJECTION

Source: https://www.youtube.com/watch?v=WONbg6ZjiXk

Source: https://www.youtube.com/watch?v=J6v_W-LFK1c&t=122s

DATABASE APPLICATIONS
2. API CALLS – JDBC: PARAMETER BINDING

- One possible Solution: use PreparedStatement

- Syntax

```
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 () ;
```

DATABASE APPLICATIONS
2. API CALLS – JDBC CLASSES

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: TRANSACTION HANDLING

- Transaction syntax:

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

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

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

DATABASE APPLICATIONS
2. API CALLS – JDBC: 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
  - ...

DATABASE APPLICATIONS
2. API CALLS – JDBC: ANTIPATTERNS (THINGS TO AVOID)

- Do not build SQL string using user input!

- Syntax:

```
string name = ge
stringquery = "                    name = '"+name+"'" ;
```

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

→ <u>Solution</u>: Always use PreparedStatement / parameter binding!

DATABASE APPLICATIONS
2. API CALLS – JDBC: ANTIPATTERNS (THINGS TO AVOID)

- 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

DATABASE APPLICATIONS
2. 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)

DATABASE APPLICATIONS
3. EXTEND SQL – PROCEDURAL LANGUAGE EXTENSION

- 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)

DATABASE APPLICATIONS
3. 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

DATABASE APPLICATIONS
3. EXTEND SQL – PL/SQL

- Syntax for creating procedures:

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

- Syntax for creating functions:

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

DATABASE APPLICATIONS
3. EXTEND SQL – PL/SQL

- Using variables and defining data types
- Example:

```
declare
    today  date ;
    type   PersonRecordType is record
           ( PersonName varchar2 ( 50 ) ;
           BirthDate date ) ;
    employee PersonRecordType ;
```

- Cursor for processing results:

```
cursor CurBook is
    SELECT isbn , title FROM Books ;
```

DATABASE APPLICATIONS
3. EXTEND SQL – PL/SQL

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

- Example:

```
if <condition > then
    < PL/SQ-operation >
else
    <PL/SQL-operation>
end if ;
```

- and loops (for, while, loop)
- Example:

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

- Example: Executing a relation with infinite loop

```
loop
        fetch Book into BookRecord ;
exit when Book%not found ;
  …
end loop ;
```

DATABASE APPLICATIONS
3. EXTEND SQL – EXAMPLE

- Example: MySQL

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


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

DATABASE APPLICATIONS
3. EXTEND SQL – EXAMPLE

- Example: MySQL

```
delimiter |
CREATE TRIGGER IF NOT EXISTS Print_salary_changes
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
     BEGIN
        DECLARE sal_diff DECIMAL(10,2);
        IF (NEW.salary != OLD.salary)
        THEN
             BEGIN
                SET sal_diff = NEW.salary - OLD.salary ;
                CALL output(NEW.ssn, OLD.salary, NEW.salary, sal_diff);
             END;
        END IF;
END;
|
delimiter ;
```

DATABASE APPLICATIONS
3. EXTEND SQL – EXAMPLE

- Example: Oracle

```
DROP PROCEDURE add_color ;

CREATE PROCEDURE add_color ( param1 IN VARCHAR2)
    IS
    BEGIN
    INSERT INTO color VALUES
          ( (SELECT COUNT(id) FROM color) , param1 ) ;
    END;
/
EXEC add_color('Lightblue') ;
```

DATABASE APPLICATIONS
3. 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

DATABASE APPLICATIONS
3. 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

DATABASE APPLICATIONS
3. 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

DATABASE APPLICATIONS
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

Source: Elmasri, Fundamentals of Database Systems, Page 476ff

DATABASE APPLICATIONS
SUMMARY

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

Source: Elmasri, Fundamentals of Database Systems, Page 476ff

DATABASE APPLICATIONS
SUMMARY

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

Source: Elmasri, Fundamentals of Database Systems, Page 476ff