61FIT3JSD
Fall 2023

Lecture 10

Application Development with Relational Database

Lecture outline

- Relational database
- SQLite essentials
- Working with SQLite database
- Application architecture
- Database-driven application

Relational database

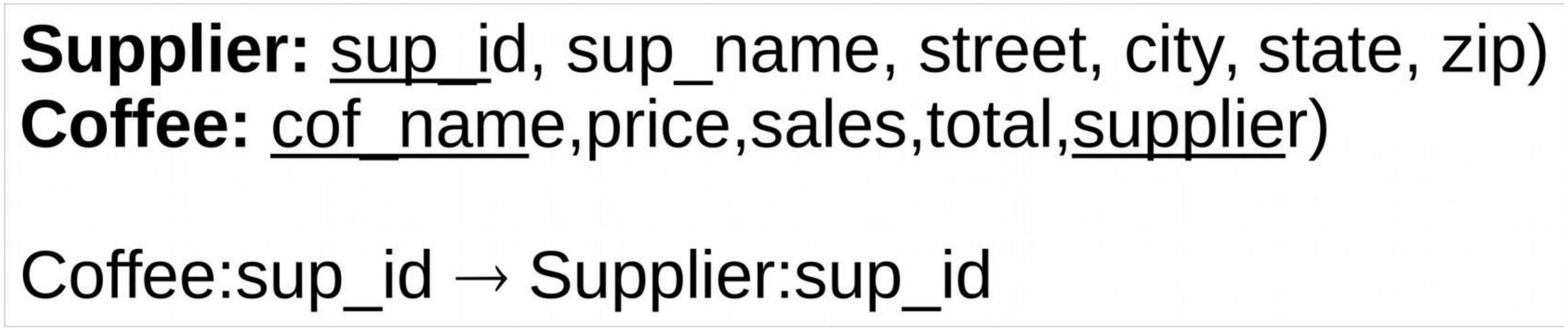
- Overview
- Integrity rules
- Structured query language (SQL)

Relational database overview

- An organised collection of data using two-dimensional tables
- A table (with rows & cols) is called a relation
- Tables are related via common keys
- Relational DBMS (RDBMS) manages how data are stored, maintained and retrieved in a relational database

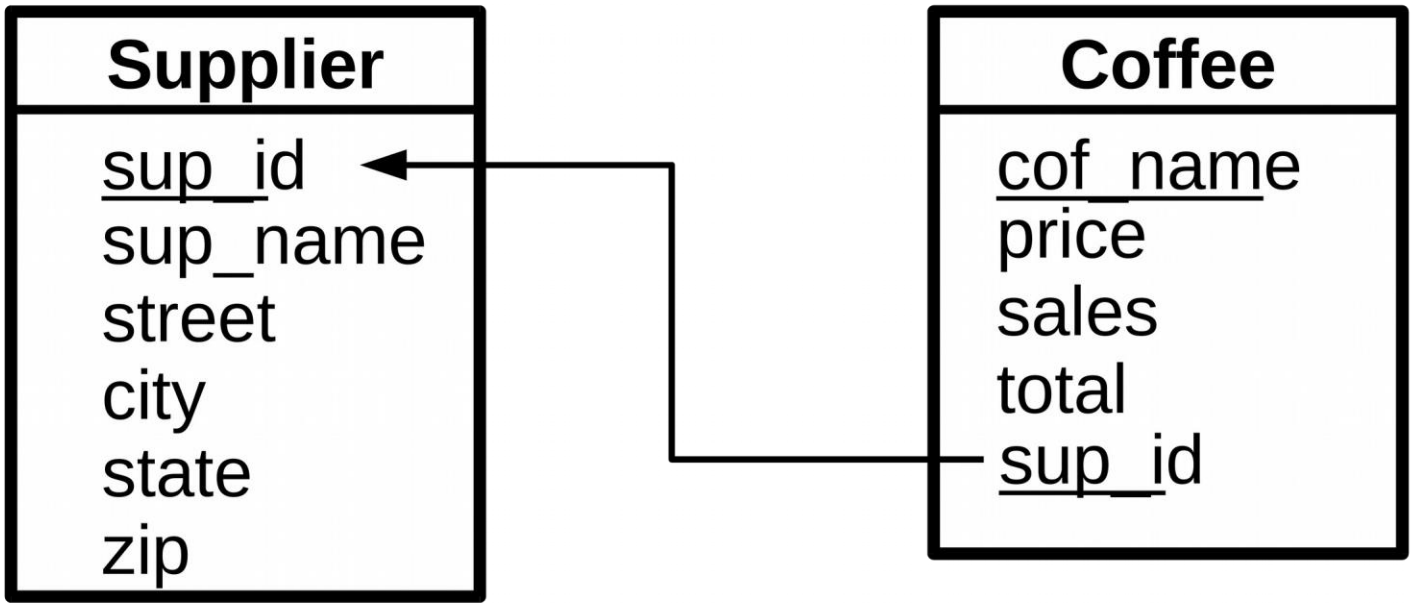
Relational Database

Database example: Coffeedb



Supplier: sup_id, sup_name, street, city, state, zip)
Coffee: cof_name,price,sales,total,supplier)

Coffee:sup_id → Supplier:sup_id



Relational Database

Example: Supplier table

sup_id	sup_name	street	city	state	zip
49	Superior coffee	Party Place	Mendocino	CA	95460
101	Acm, Inc.	99 Market Street	Groundsville	CA	95199
150	Restaurant Supplies, Inc.	200 Magnolia street	Meadows	CA	93966
927	Professional kitchen	300 Daisy Avenue	Groundsville	CA	95199

Relational Database

Example: Coffee table

cof_name	price	sales	total	sup_id
Colombian	7.99	0	0	101
French_Roast	8.99	0	0	49
Expresso	9.99	0	0	150
Colombian_Decaf	8.99	0	0	101
French_Roast_Decaf	9.99	0	0	049

Integrity rules

- To ensure that data in a relational database are accurate and always accessible
- Entity integrity
  - Every table must have a primary key
  - Neither the PK nor any part of it can contain `null` values
- Referential integrity
  - Foreign key must have a matching primary key or it must be `null`

Structured query language (SQL)

- A query language for relational databases
- SQL statements are divided into two categories:
  - Data manipulation language (DML)
  - Data definition language (DDL)

DML	DDL
SELECT	CREATE DATABASE
INSERT	CREATE TABLE
UPDATE	ALTER TABLE
DELETE	DROP TABLE

Relational Database

DDL: Supplier

```
CREATE TABLE SUPPLIER (
   SUP_ID      INTEGER NOT NULL,
   SUP_NAME    TEXT    NOT NULL,
   STREET      TEXT    NOT NULL,
   CITY        TEXT    NOT NULL,
   STATE       TEXT    NOT NULL,
   ZIP         TEXT,
   PRIMARY KEY(SUP_ID)
);
```

DDL: Coffee

```
CREATE TABLE COFFEE (
   COF_NAME  TEXT     NOT NULL,
   SUP_ID    INTEGER  NOT NULL,
   PRICE     REAL     NOT NULL,
   SALES     INTEGER  NOT NULL,
   TOTAL     INTEGER  NOT NULL,
   FOREIGN KEY(SUP_ID) REFERENCES SUPPLIER(SUP_ID),
   PRIMARY KEY(COF_NAME)
);
```

DML: Supplier

```
INSERT INTO SUPPLIER VALUES(
    49,  'Superior Coffee',
    '1 Party Place', 'Mendocino',
    'CA', '95460'
);
```

Relational Database

DML: Coffee

```
INSERT INTO COFFEE
VALUES('Colombian', 101, 7.99, 0, 0);
```

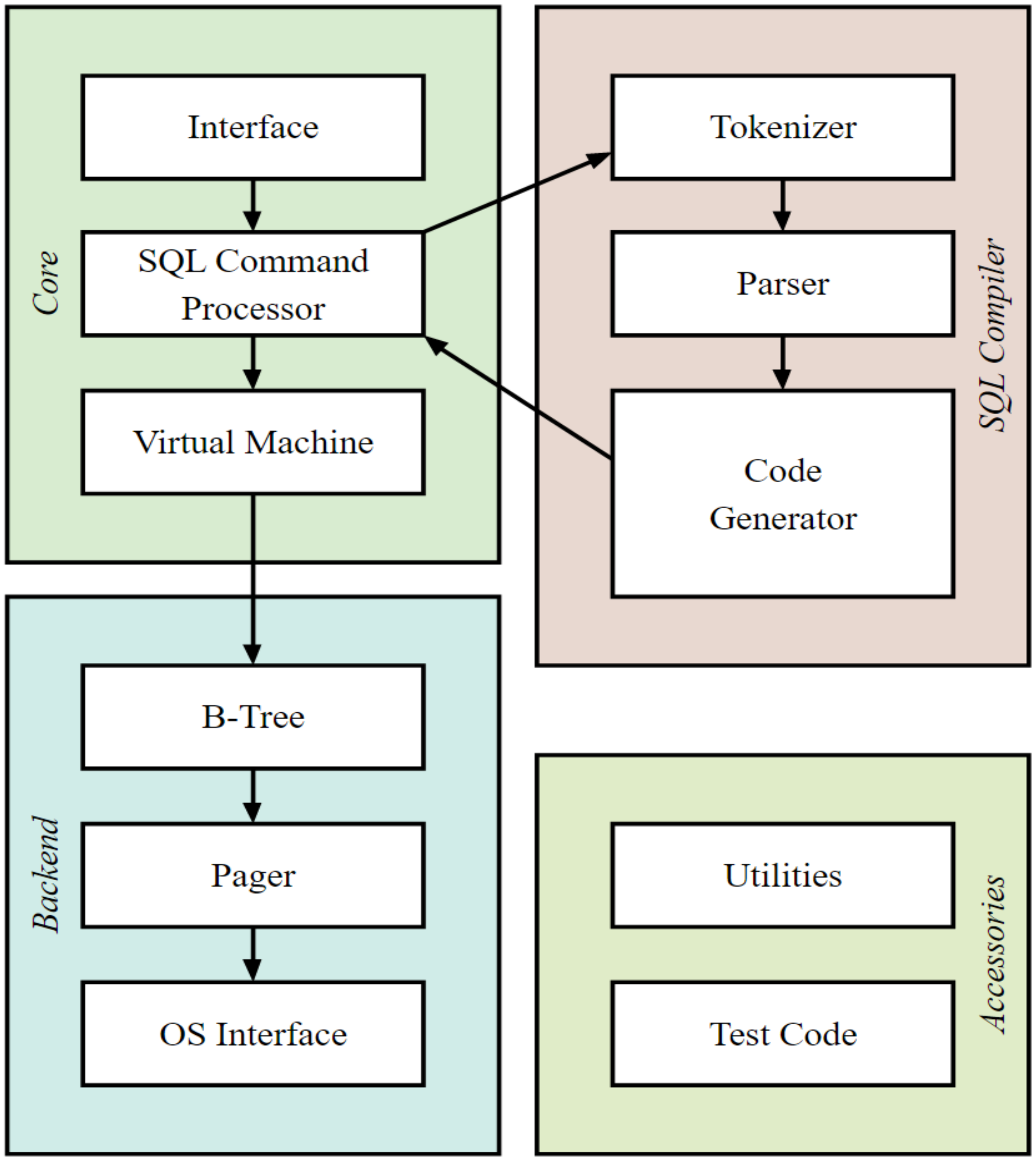
SQLite essentials

- What is SQlite?
- SQLite architecture
- SQlite database & types
- JDBC overview
- SQLite JDBC driver
  - Database connection URL

What is SQLite?

- A light-weight, full-featured open source RDBMS
- Originally developed by Richard Hipp in 2000
  - https://www.sqlite.org
- The most widely deployed database engine
- Noticable features: Transactions, Zero-Configuration, Serverless, Single Database File, Cross-Platform
- Latest version requires Java 1.8 or higher
- Hibernate support:
  - Unofficial support for Hibernate 3, 4, 5
  - Official community support for Hibernate 6

SQLite Architecture



SQLite database

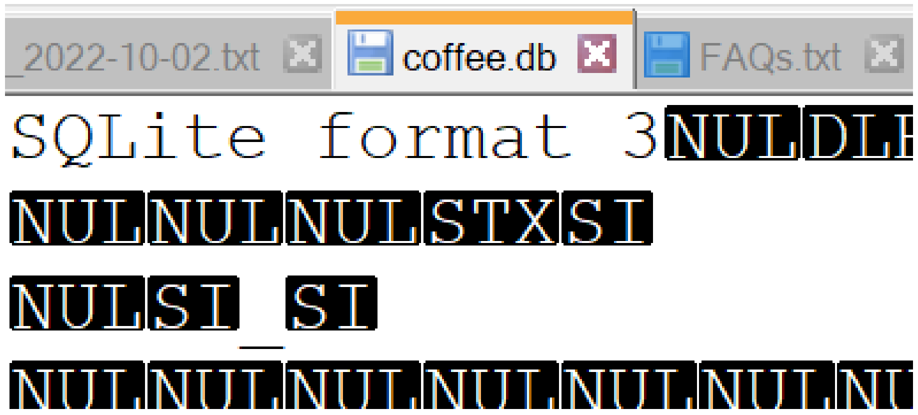
- Resides in a single file
- Contains tables, indexes, views and triggers
- During transactions, SQLite stores additional information in a second file called “rollback journal”.

SQLite Types

- SQLite supports only the following value classes:
  - **NULL** -- the value is a NULL value.
  - **INTEGER** -- the value is a signed integer, stored in 0, 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - **REAL** -- the value is a floating point value, stored as an 8-byte IEEE floating point number.
  - **TEXT** -- the value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
  - **BLOB** -- the value is a blob of data (binary), stored exactly as it was input.

Create a database

- Connect to the database will create it (if not created already)
- Alternatively, use a GUI tool to create an SQLite database.
- Find out a database's version:
    - Open the DB file with some text editor.



```
SQLite format 3NULDLE
NULNULNULSTXSI
NULSI_ SI
NULNULNULNULNULNULNU
```

Delete a database

- Delete the file which contains the database

JDBC overview

- Java Database Connectivity
- a Java API that provides applications with access to relational databases
- Consists of two packages:
  - `java.sql`
  - `javax.sql`
- Each supported RDBMS must provide a driver class that implements access to its databases:
  - For SQLite database: `org.sqlite.JDBC`

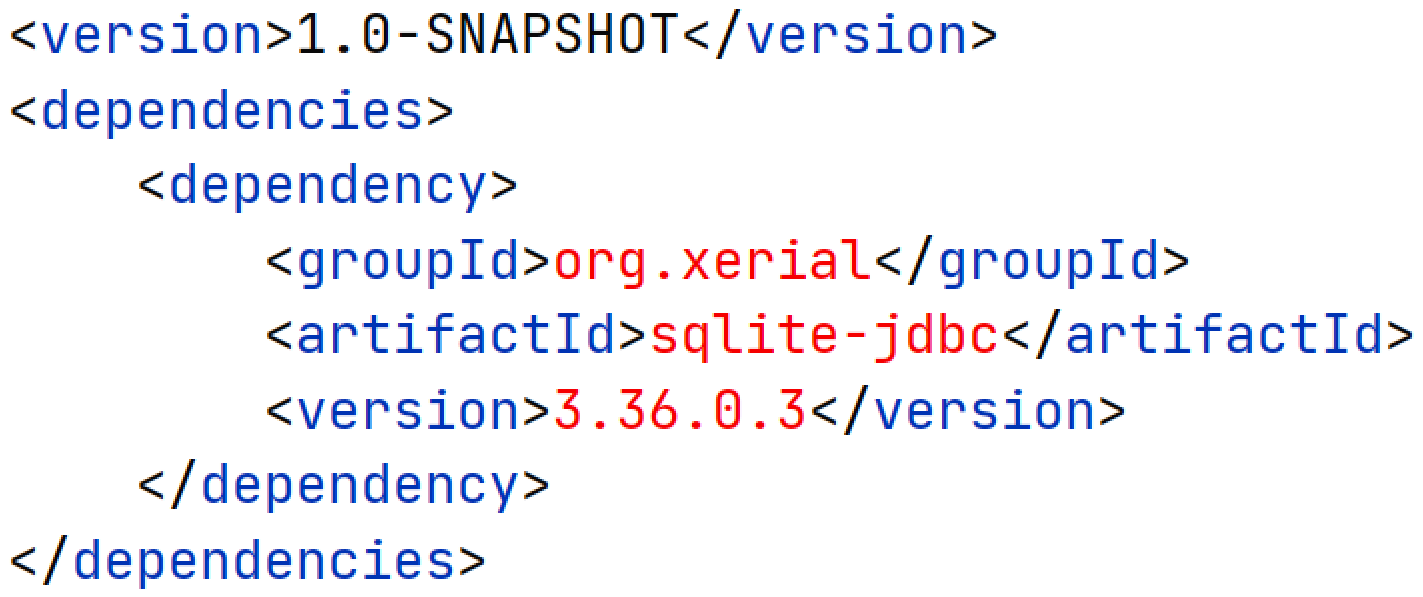
JDBC Basic Example

```
Connection con = DriverManager.getConnection(
        "jdbc:myDriver:myDatabase",
        "username",
        "password");

Statement stmt = con.createStatement();
String sql = "SELECT a, b, c FROM Table1";
ResultSet rs = stmt.executeQuery(sql);

while (rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
}
```

Obtaining SQLite’s JDBC Driver

- Download `.jar` file from Maven repository:
  - https://search.maven.org/artifact/org.xerial/sqlite-jdbc
- Use Maven dependency management:

`pom.xml`



```
<version>1.0-SNAPSHOT</version>
<dependencies>
    <dependency>
        <groupId>org.xerial</groupId>
        <artifactId>sqlite-jdbc</artifactId>
        <version>3.36.0.3</version>
    </dependency>
</dependencies>
```

SQLite database connection

Connection string:

jdbc:sqlite:*databaseName*

Code example:

```
Connection conn = null;
try {
    conn = DriverManager.getConnection(
            "jdbc:sqlite:coffee.db");
    System.out.println("Connected to database!");
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

SQLite GUI Tools

- DB Browser for SQLite
    - https://sqlitebrowser.org
- Chrome Extension: SQLite Manager
- SQLiteStudio
    - https://sqlitestudio.pl
- DataGrip
    - https://www.jetbrains.com/datagrip

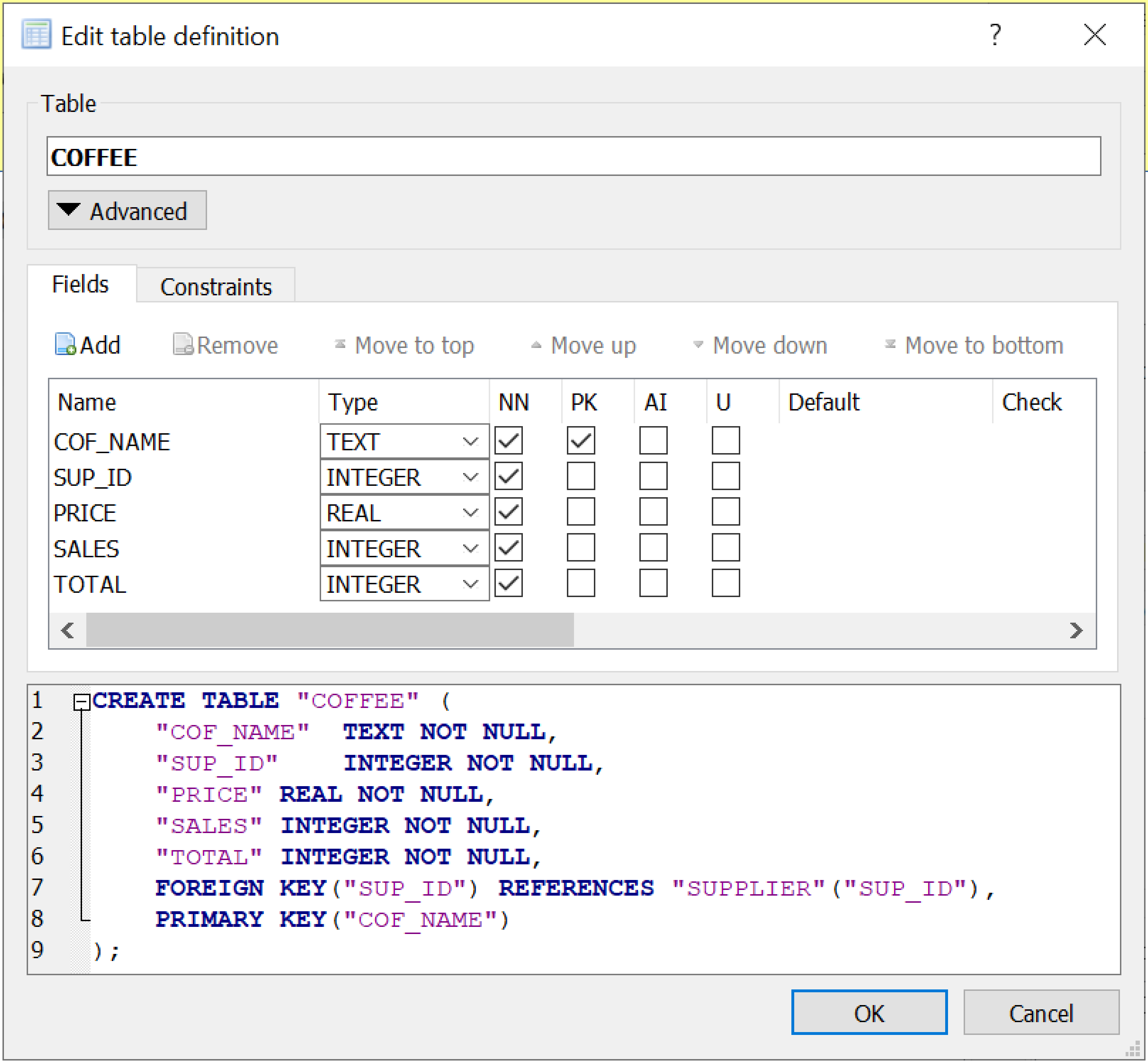


Table: COFFEE

Name	Type	NN	PK	AI	U	Default	Check
COF_NAME	TEXT	✓	✓				
SUP_ID	INTEGER	✓					
PRICE	REAL	✓					
SALES	INTEGER	✓					
TOTAL	INTEGER	✓					

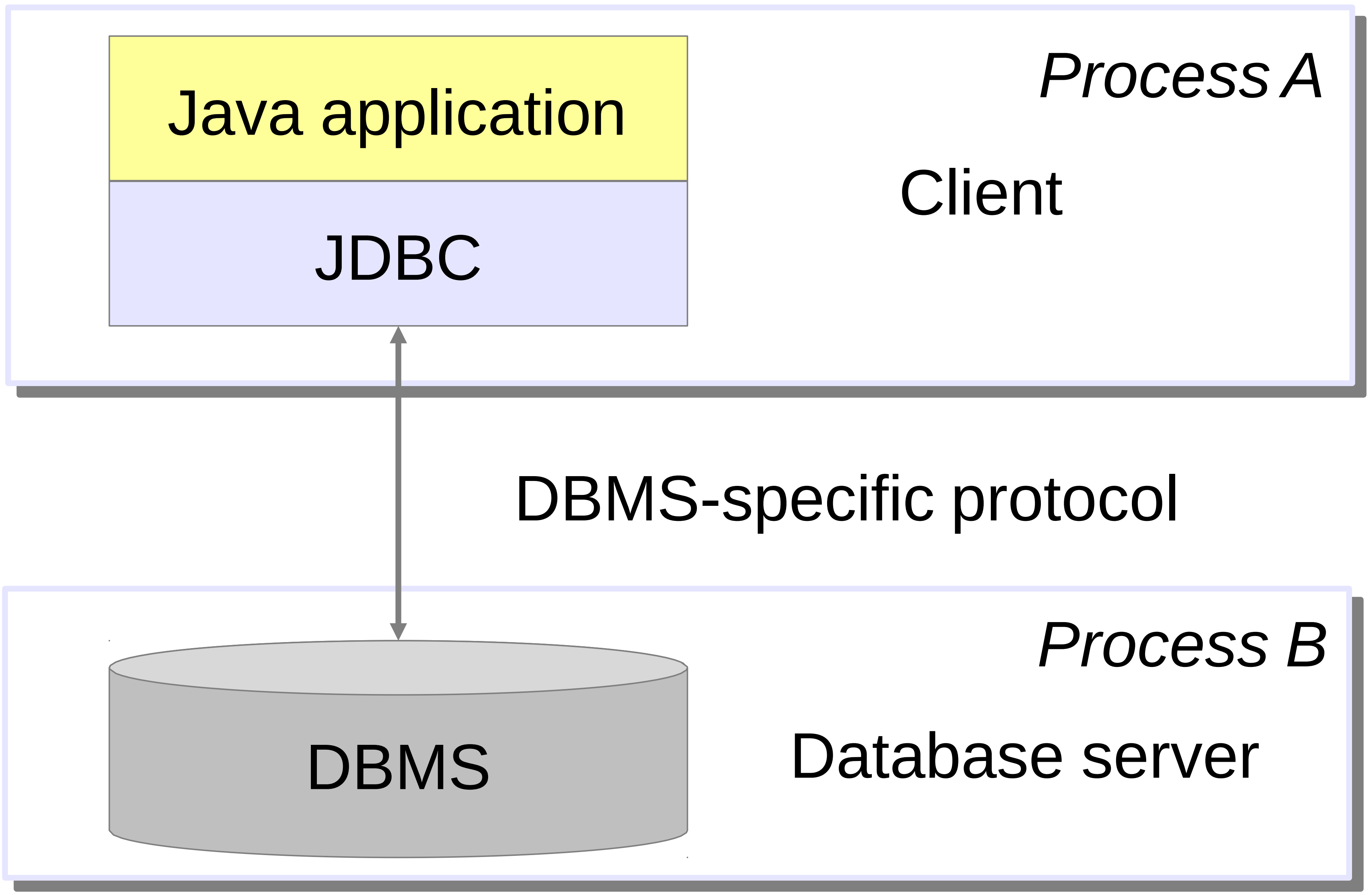
```sql
CREATE TABLE "COFFEE" (
    "COF_NAME"  TEXT NOT NULL,
    "SUP_ID"    INTEGER NOT NULL,
    "PRICE" REAL NOT NULL,
    "SALES" INTEGER NOT NULL,
    "TOTAL" INTEGER NOT NULL,
    FOREIGN KEY("SUP_ID") REFERENCES "SUPPLIER"("SUP_ID"),
    PRIMARY KEY("COF_NAME")
);
```

Application architecture

- Describes the high-level components and how they inter-connect
- Follow the client/server architecture:
  - client: the application components (application logic)
  - server: the RDBMS
- Variations:
  - two-tier architecture
  - two-tier, embedded
  - three-tier architecture

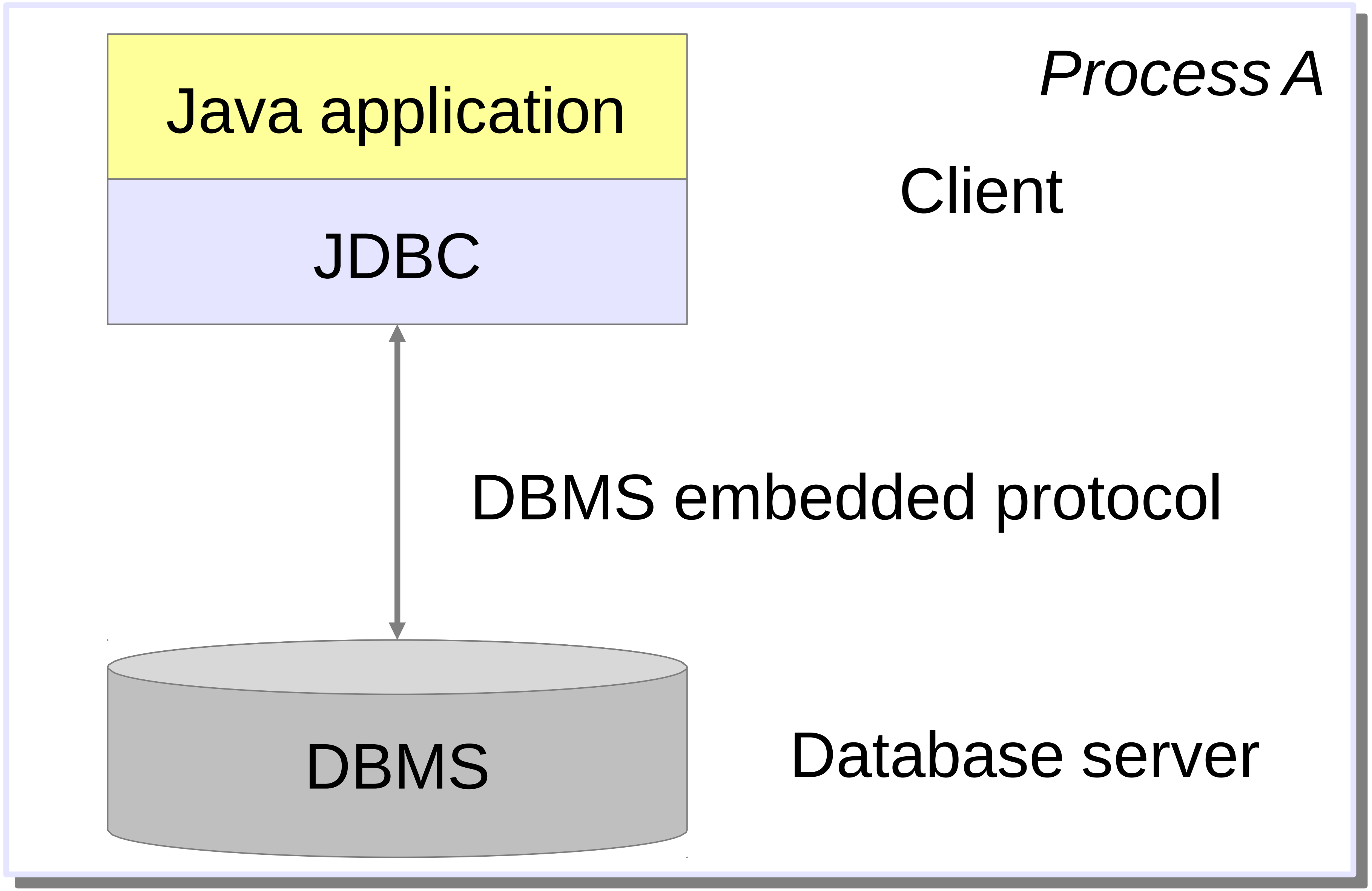
Application architecture

Two-tier architecture



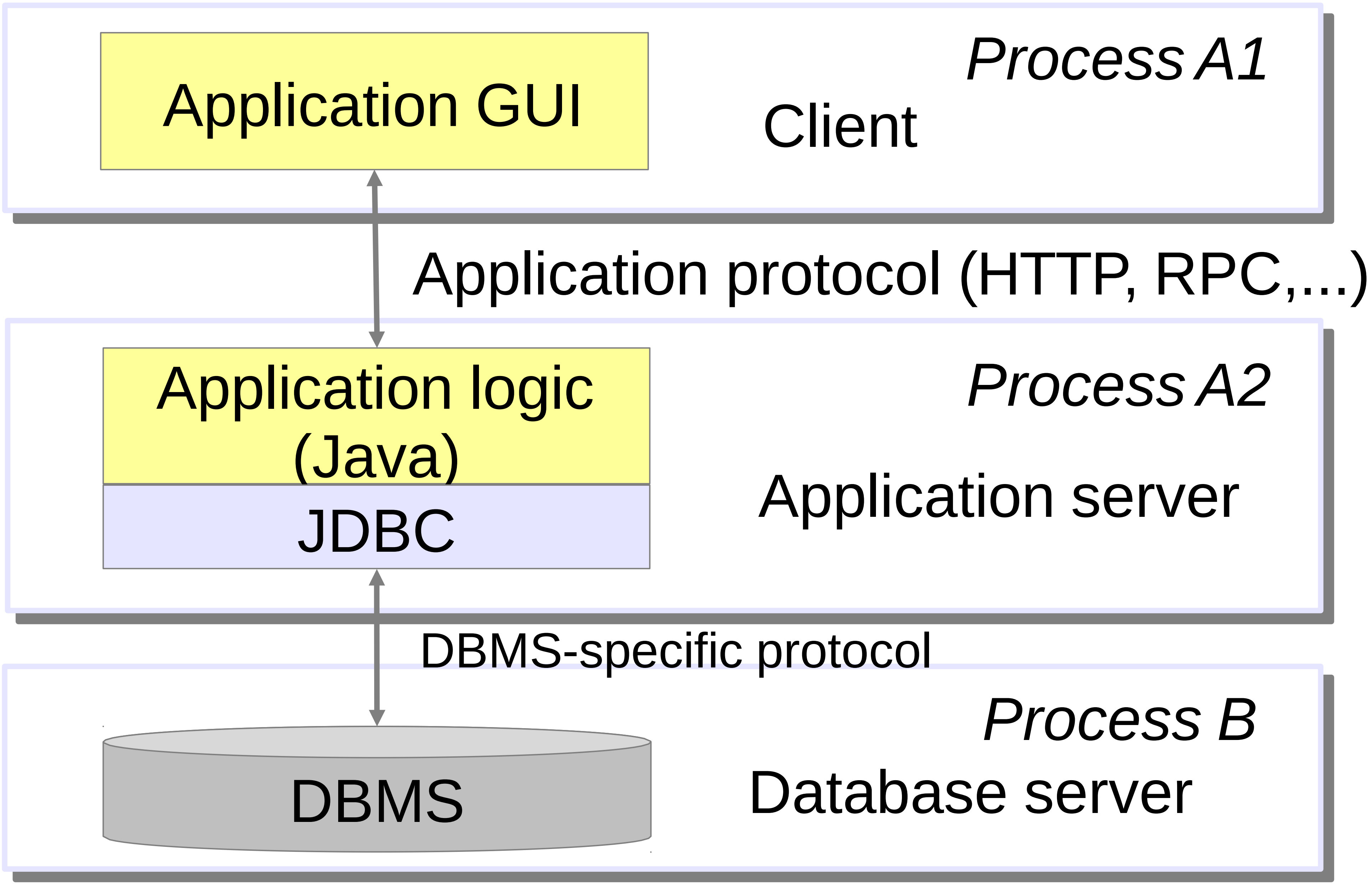
Application architecture

Two-tier (embedded) architecture



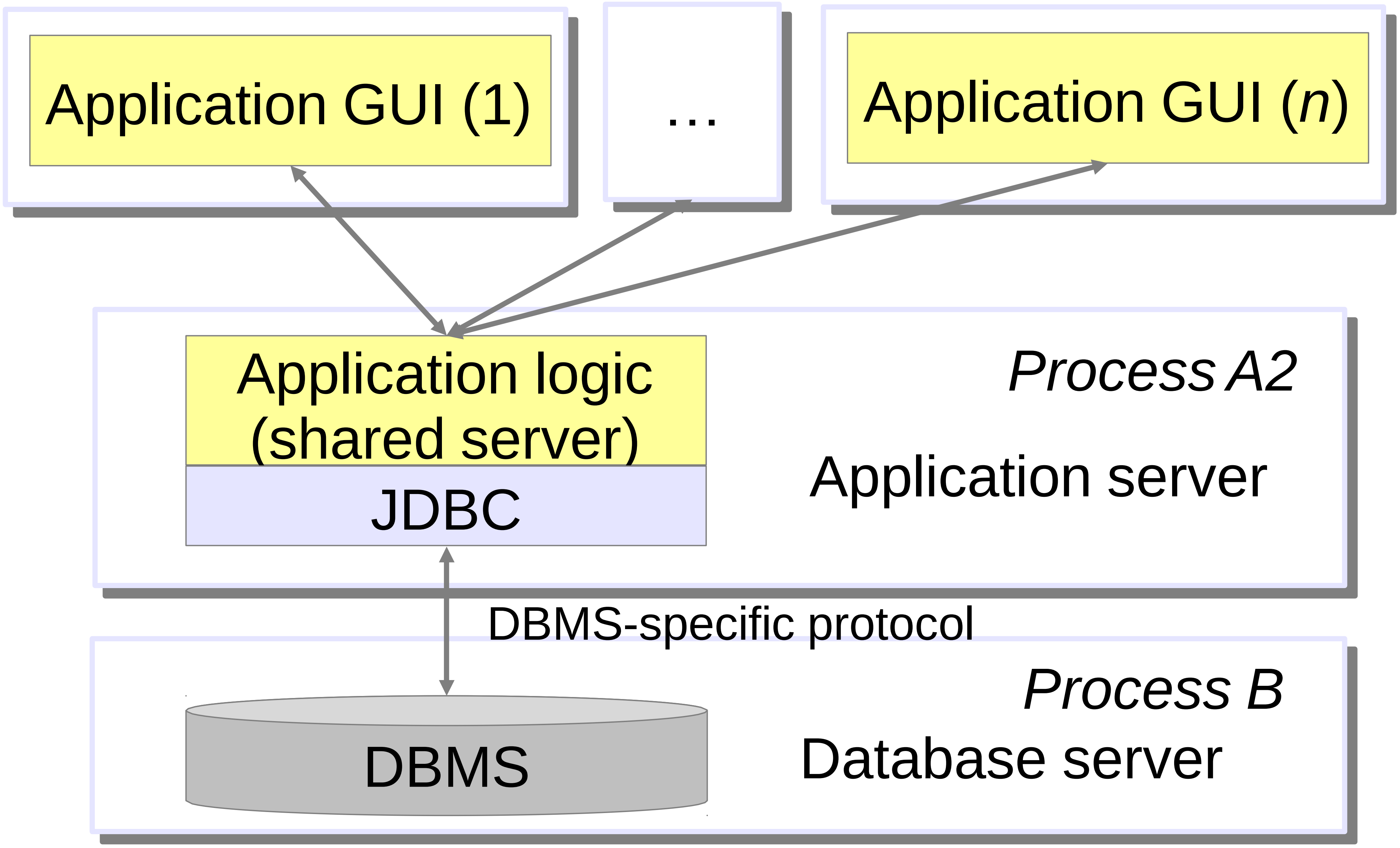
Application architecture

Three-tier architecture



Application architecture

Three-tier architecture: scalable for many clients



A database-driven application

- Architecture:
  - Embedded
- Object orientation:
  - Data model is object-oriented
- Physical:
  - Physical data model: language specific (e.g. Java)
- RDBMS:
  - SQLite
- API: Java SQL (`java.sql`)

Application development tasks

- Connect to a database
  - Create if needed
- Set up database tables
  - Can be done using GUI Tools prior to application development
- Querying/updating data, process query results
- Disconnect from the database

Connect to a database

- Create a `Connection` object to the database
- Using either of these:
  - **`DriverManager`:** simpler and easy to use
  - **`DataSource`:** more flexible and supports advanced connection properties (e.g. pooling and distributed)

DEMO

DBApp

- `connect()`

Set up database tables

- Create tables
- Populate tables with data

Create tables

- SQL statement: `CREATE TABLE`
- Create a statement object from the connection:

```
Statement s = conn.createStatement()
```

- Execute the SQL statement:

```
s.executeUpdate(sql)
```

SQL statement composition

- Online mode:
  - SQL statements are created in the application code
  - Usage: user inputs (e.g. table name and columns) are needed for statement parameters
- Offline mode:
  - SQL statements are created outside the code
  - Stored in an .sql script file
  - Requires code to read the script file

Online mode example

```
Statement s = conn.createStatement();
String sql = "CREATE TABLE SUPPLIER (" +
        "SUP_ID INTEGER NOT NULL," +
        "SUP_NAME TEXT NOT NULL," +
        "STREET TEXT NOT NULL," +
        "CITY TEXT NOT NULL," +
        "STATE TEXT NOT NULL," +
        "ZIP TEXT," +
        "PRIMARY KEY(SUP_ID));";
s.executeUpdate(sql);
```

DEMO

Offline mode example

```
DBApp.executeStatementsFromFile()
```

DEMO

DBApp

- Online:
  - **DBAppSupplier**.createSupplierTable()
- Offline:
  - **DBApp**.createTables()

Populate tables with data

- SQL statement: INSERT
  - also composed in online- or off-line mode
- Create a statement object from the connection
- Execute the SQL statement:
  - use `Statement.executeUpdate()`
- Online INSERT example:

```
String sql = "INSERT INTO SUPPLIER VALUES" +
        "(927, 'Professional Kitchen', '300 Daisy Avenue'," +
        "'Groundsville', 'CA', '95199');";
s.executeUpdate(sql);
```

DEMO

DBApp

- Online:
  - **DBAppSupplier**.populateSupplierTable()
- Offline:
  - **DBApp**.populateTables()

Query data

- SQL statement: `SELECT`
  - two modes: on-line or off-line
- Create a statement object from the connection
  - use `Statement.executeQuery()`
- Execute the SQL statement:

```
ResultSet rs = s.executeQuery(String sql)
```

- Process the `ResultSet` object

Process a `ResultSet` object

- A table of data representing a database result set
- Provides an access to the data through *cursor*:
  - a pointer to one row of data in the `ResultSet`
  - initially positioned before the first row
  - `ResultSet.next()` moves cursor to the next row. Returns `false` if no more rows

Example

```
String sql = "SELECT SUP_ID, SUP_NAME, STREET, CITY," +
        "STATE, ZIP FROM SUPPLIER";
Statement s = conn.createStatement();

ResultSet rs = s.executeQuery(sql);
while (rs.next()) {
    int supplierID = rs.getInt("SUP_ID");
    String sup = rs.getString("SUP_NAME");
    String str = rs.getString("STREET");
    String city = rs.getString("CITY");
    String state = rs.getString("STATE");
    String zip = rs.getString("ZIP");
}
```

DEMO

DBApp

- Online:
  - **`DBAppSupplier.`**`querySupplier()`
  - **`DBAppSupplier.`**`querySupplierMetadata()`
- Offline:
  - **`DBApp.`**`queryTables()`

Close the connection

- Close the statement
- Disconnect from the database

Close the statement

```
try {
   ...
} catch {
   ...
} finally {
   if (s != null)
      try { s.close(); }
      catch (Exception e) {}
}
```

Disconnect from the database

- Use method `Connection.close()`

```
try {
    conn.close();
} catch (SQLException e) {
    // ignore
}
```

DEMO

DBApp

- `close()`

Summary

- A relational database stores data in relations
- SQLite is a light-weight, popular RDBMS that works with JDK 1.8 or above
- SQLite stores a database in a single file and requires zero configuration
- SQLite can be embedded into Java project
- An embedded database is accessed directly by specifying its directory path in the conn's URL

Summary

- Database-driven applications are typically designed as a client/server architecture
- Applications using embedded database is a special case of the two-tier client/server architecture
- Application development tasks are implemented in Java using:
  - `java.sql` package
  - suitable JDBC classes