SQL and Programming Languages

Integrity constraints

- conditions that every legal instance of a relation must satisfy
- actions that violate ICs are disallowed
- classifying an IC
  - tuple-level IC
  - table-level IC

Checks

↳ in create

Check (condition)

checks only occur with insert/update,
not delete

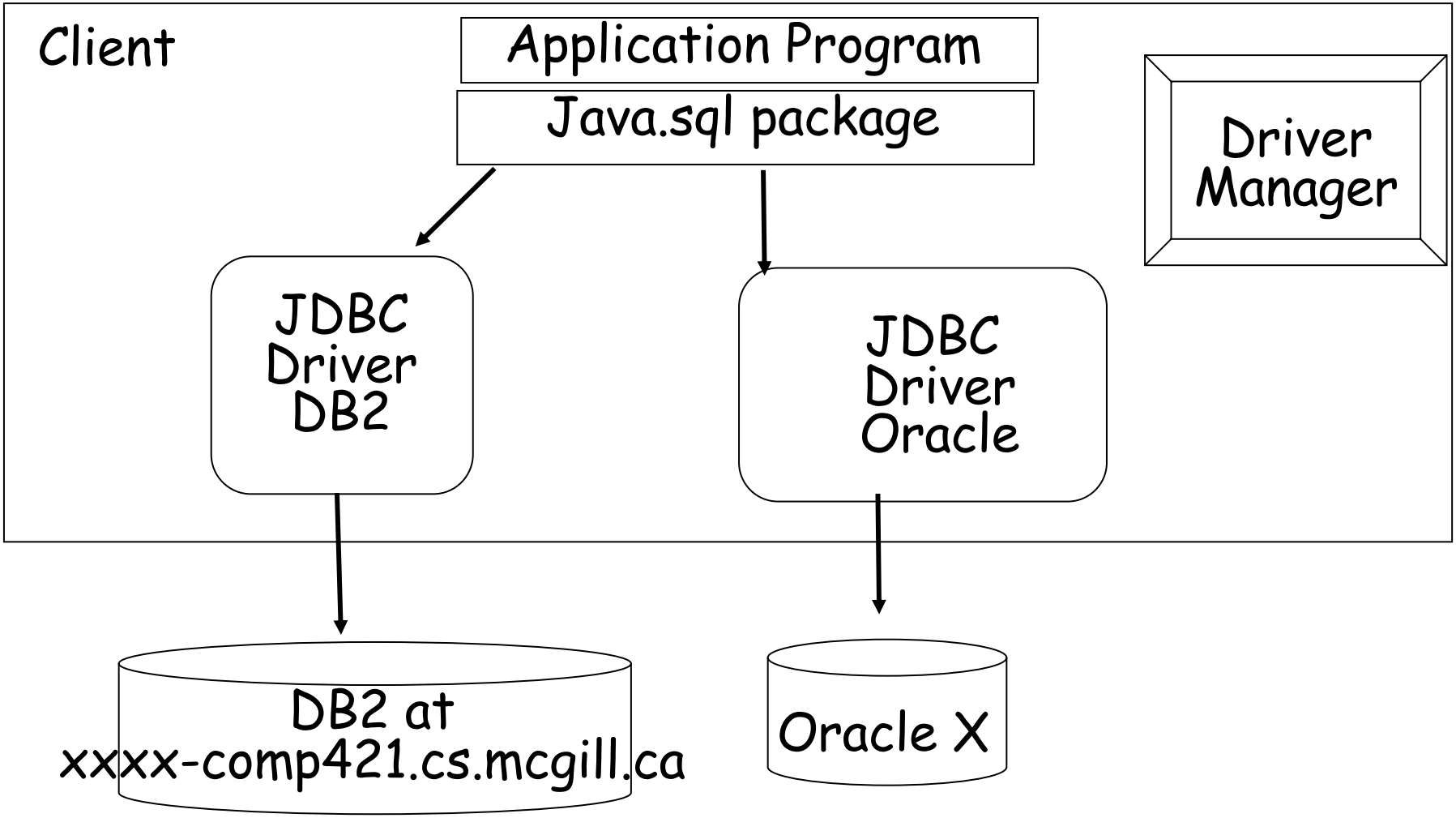
SQL Pure

- Example: Interactive User interface of DB2/Postgresql as we have used it so far
- Execution
  - ☆ We can type any SQL statement.
  - ☆ Statement is sent to DBMS
  - ☆ Statement is executed within DBMS
  - ☆ Response is sent back
  - ☆ User can be on same machine or other machine than DBMS
- Limitations
  - ☆ SQL is not TURING complete (no if/then/else, loops, etc)
  - ☆ Is intended to retrieve data
  - ☆ lacks features of traditional application programming languages.

SQL in Application Code

- SQL commands can be called from within a host language (e.g., C++ or Java) program
- Two main integration approaches
  - ☆ Embed SQL in host language (C with Embedded SQL, SQLJ)
  - ☆ Create special API to call SQL commands (JDBC)
- Program - DB interaction
  - ☆ Application program executes at client side.
  - ☆ Each SQL statement is sent to the server, executed there, and the result returned to the client

JDBC API: Architecture



Database APIs: JDBC

- Special, standardized interface of objects and method calls
- Attempts to be DBMS-neutral
  - A "driver" traps the calls and translates them into DBMS specific calls
  - Database can be across a network
- Four components
  - At client:
    - Application (submits SQL statements)
    - Driver manager (handles drivers for different DBMS)
    - Driver (connects to a data source, transmits requests, and returns/translates results and error codes
  - At server:
    - Data source (processes SQL statements)

Execution Overview

- Load Driver
  - Since only known at runtime which DBMS the application is going to access, drivers are loaded dynamically
- Connect to Data Source
  - Connection Object represents the connection between a Java client and DBMS
  - Each connection identifies a logical session.
- Execute SQL statements

① Connecting to Database

```
import java.sql.*;
class connectExample {
  public static void main( String args [] ) throws SQLException {
    // Load the DB2 JDBC driver
    // Alternative 1:
    // Class.forName("com.ibm.db2.jcc.DB2Driver");
    // Alternative 2:
    DriverManager.registerDriver (new com.ibm.db2.jcc.DB2Driver());

    // Connect to the database
    Connection conn = DriverManager.getConnection(
        "jdbc:db2://comp421.cs.mcgill.ca:50000/cs421",
        "user_name", "user_password"
    ...
    // Close the connection
    conn.close();
}
```

driver name of DB

Replace with actual server name

JDBC drivers are not part of standard Java runtime environment. You may have to download them from the vendors separately.

SQL statements

- Create a statement object so we can submit SQL statements to the driver

```
Statement stmt = con.createStatement();
```

- Close statement when no more needed

```
stmt.close();
```

- Update/insert

```
String sqlString = "INSERT INTO Skaters " +
                   "VALUES (58, 'Lilly', 10, 12)";
stmt.executeUpdate(sqlString);
```

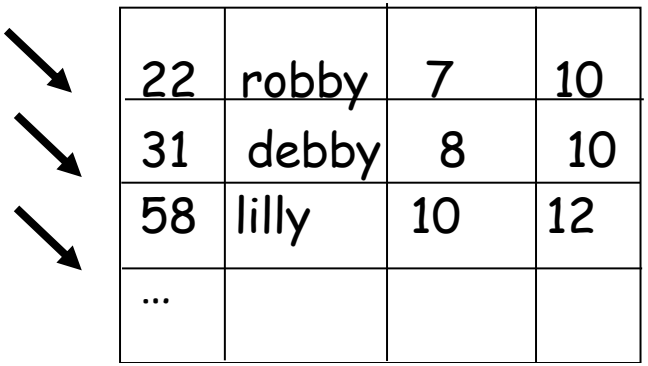
String containing SQL Statement

  - ☆ insert/update/delete return the number of affected tuples

Queries with several result tuples

- Mismatch between standard SQL query and programming language:
  - ☆ Result of a query is usual a SET of tuples with no a priori bound on the number of records
- Solution: get result tuples step by step
- *Cursor* Concept:
  - ☆ Think about a cursor as a pointer to successive tuples in the result relation. At a given moment the cursor can be
    - Before the first tuple
    - On a tuple
    - After the last tuple

like a list of the tuples



SQL Select Statements

using * is bad practice !

- SELECT

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Skaters");
while ( rs.next() ) {
    int sid = rs.getInt("sid");
        sname = rs.getString("name");
        age = rs.getInt(3);
        rating = rs.getInt(4);
        System.out.println("skater " + sname + " has age " +
                           age + " and rating " + rating);
    }
rs.close();
```

attribute name in tuple

using index number is bad practice ! Use column names

  - Cursor initially set just before first row
  - rs.next makes cursor point to the next row
  - rs.getInt/rs.getString etc. retrieve individual attributes of row
    - We have to do type conversion – SQL data types vs. Java data types
    - Input is either name or position of attribute
  - methods to find out where you are in the result set:
    - getRow, isFirst, isBeforeFirst, isLast, isAfterLast.
  - Methods to get tuples
    - next(), previous(), absolute(int num); relative(int num); first(); last()

JDBC Statements

- Statement
  - ☆ Includes methods for executing text queries
- ResultSet
  - ☆ Contains the results of the execution of a query
  - ☆ We can receive all result rows iteratively
  - ☆ For each row we can receive each column explicitly

SQL Injection Attacks

What could go wrong?

```
inssql = "INSERT INTO customers ('" + username + "',");
stmt.executeUpdate(inssql);
```

What if somebody entered a username that is `D'Silva` ?

What if somebody entered a username that is `D';DROP TABLE customers;--'` ?

https://en.wikipedia.org/wiki/SQL_injection

Dynamic vs. Prepared Statements

- Dynamic Execution at DBMS
  - Parses statement (1), builds execution plan (2), optimizes execution plan (3), executes (4), and returns (5)
  - If statement is called many times (e.g., loop), steps (1)-(3) are executed over and over again
- PreparedStatement
  - Represents precompiled and stored queries
  - Can contain parameters; values determined at run-time

Prepared Statements

- **JDBC**

```
// Statement can have input parameters; indicated with ?
PreparedStatement prepareCid =
    con.prepareStatement("SELECT cid
                          FROM Participates where sid = ?")
while () {
  // get cid input from user into variable x;
   ….
   // provide values for input parameters
   prepareCid.setInt(1, x);
    ResultSet rs = prepareCid.executeQuery() ;
   while (rs.next())
       System.out.println("skater " + x +
                          "participates in competition "
                          + rs.getInt(1));
  }
```

different each time

put x into ?

Error Handling

- two levels of error conditions:
  - SQLException and SQLWarning
- In Java, statements which are expected to ``throw'' an exception or a warning are enclosed in a try block.
  - If a statement in the try block throws an exception or a warning, it can be ``caught'' in one of the corresponding catch statements.
  - Each catch statement specifies which exceptions it is ready to ``catch''.

Error Handling

```
stmt.executeUpdate("CREAT TABLE Skaters " +
   "(sid INT, name CHAR(40), rating INT, age INT)" );
}catch(SQLException e) {
   System.err.println("msg: " + e.getMessage() +
                      "code: " + e.getErrorCode() +
                       state: " + e.getSQLState());
    }
```

Results

- msg: An unexpected token "CREAT" was found following "BEGIN-OF-STATEMENT". Expected tokens may include: "<space>".
- code: -104
- state: 42601

To get next exception: e.getNextException

Data Type Mapping

- There exist default mappings between JDBC datatypes (defined in java.sql.Types), and native Java datatypes
  - ☆ Examples

JDBC	*Java*	ResultSet
CHAR	Java.lang.String	getString
VARCHAR	Java.lang.String	getString
DATE	Java.sql.Date	getDate
BIT	boolean	getBoolean
INTEGER	int	getInt
REAL	float	getFloat

Database Specific Programming Languages

- Java with JDBC or C with Embedded SQL run as an application in an outside process
  - ☆ Application program makes calls to the DBS
  - ☆ Transfer of requests and results through a communication channel
- Stored Procedures
  - ☆ Programs that run within the DBS process
  - ☆ They are called from the outside but execute within the DBS

Database Specific Programming Languages

- Procedural extension to SQL
  - ☆ Certain Database Systems allow the use of standard programming languages (with extensions)
    - E.g. DB2 supports Java, C
  - ☆ Specialized programming languages specifically designed for DB access
    - DB2: SQL stored procedure (standard): specially designed programming language
    - PostgreSQL: PL/pgSQL - SQL Procedural Language
    - Has constructs of a standard procedural programming language
      - ▲variables, assignments, flow control constructs, …
    - Can have input and output parameters

SQL Procedure

```
CREATE PROCEDURE MIN_SALARY (IN deptnumber SMALLINT, IN minsal DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_salary DOUBLE;
    DECLARE v_id SMALLINT;
    DECLARE at_end INT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
   DECLARE C1 CURSOR FOR
     SELECT id, salary FROM staff WHERE did = deptnumber;
    DECLARE CONTINUE HANDLER FOR not_found  SET at_end = 1;
    OPEN C1;
    FETCH C1 INTO v_id, v_salary;
    WHILE at_end = 0 DO
       IF (v_salary < minsal)
          THEN UPDATE staff  SET salary = minsal WHERE id = v_id;
       END IF;
       FETCH C1 INTO v_id, v_salary;
    END WHILE;
    CLOSE C1;
  END
```

Execution

- Stored procedure stored as executable at DBMS
- Client makes one call to call stored procedure
  - ☆ Embedded SQL

```
EXEC SQL CALL min_salary(5,2000);
```

  - ☆ JDBC

```
CallableStatement cs =
        con.prepareCall("{call min_salary(?,?)}");
cs.setInt(1, 5); cs.setInt(2,2000);
ResultSet rs = cs.executeQuery();
```

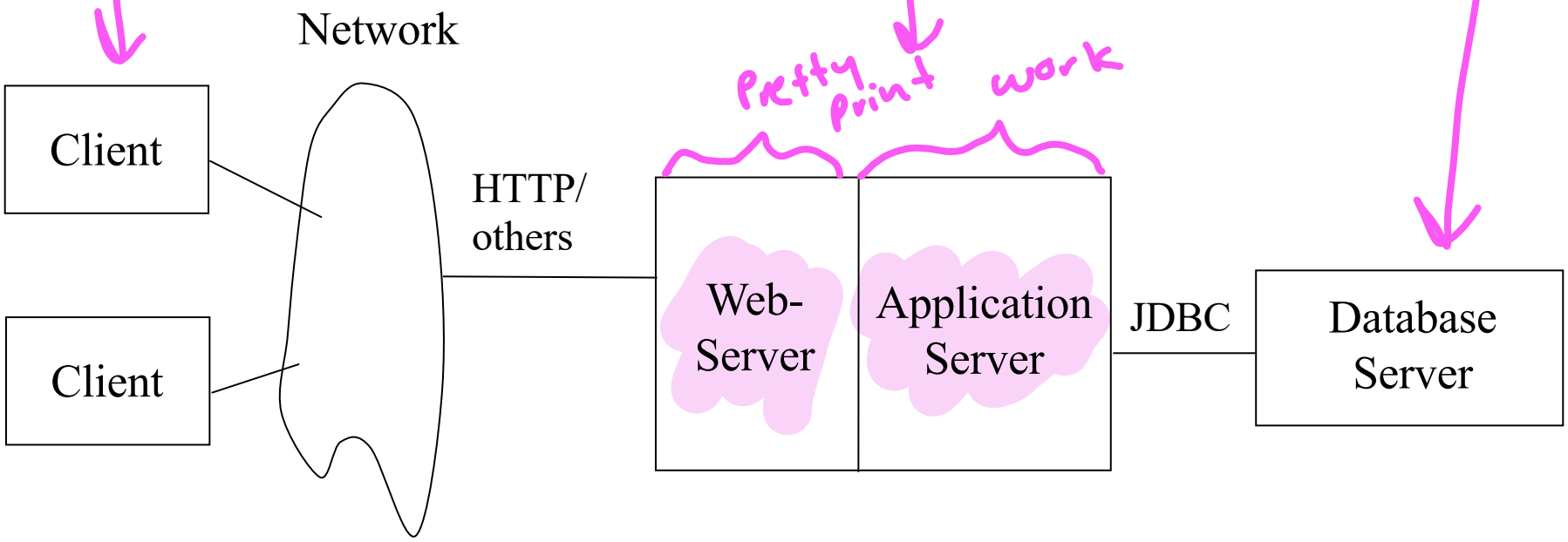
- Stored procedure executed within DBMS
  - ☆ Less context switches
  - ☆ Less calls from client to server => less network traffic

Application Architecture

- Applications can be built using one of two architectures
  - ☆ Two tier model
    - Application program running at user site directly uses JDBC/Embedded SQL to communicate with the database
    - Flexible, no restrictions
    - Security problems (passwords)
    - Code shipping problematic in case of upgrades
    - Not appropriate across organizations
  - ☆ Three tier model
    - User program only provides presentation logic (browser): client tier
    - Business semantic (JDBC programs) in application server: middle tier
    - Application server communicates with database: backend tier

web based interface

Three Tier Model



- Web-Server:
  - ☆ communication (Http and other protocols)
    - receives and unmarshals requests
    - sends responses in message format back to browser
  - ☆ presentation
    - generates the pages from result set
- Application Server
  - ☆ implements the business logic: application programs

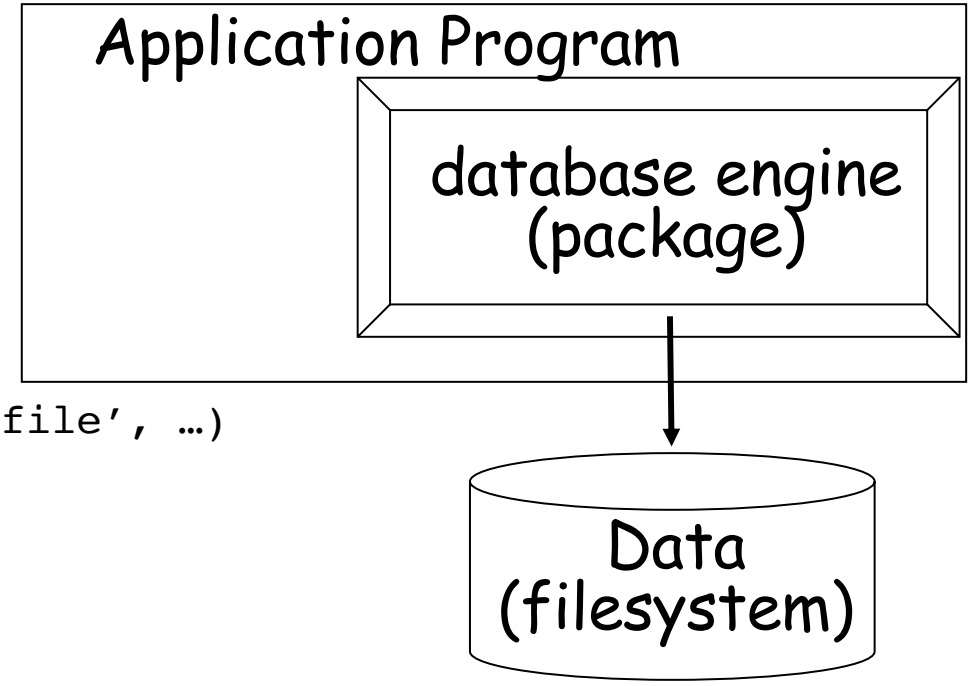
Example: Minerva

- Client Program (Browser)
  - Html pages
    - Log into System (faculty, student)
    - See courses, register, transcript, grants,
  - might use JavaScript and Cookies
- Web-Server
  - gets requests from browser
  - analyzes them and calls application server methods
  - gets responses from application server and generates dynamic web-pages
  - uses Servlets / JSP
- Application servers
  - Implements methods
  - Retrieve data from database
  - Modify database
  - uses Servlets / Beans / general programs
- Database System
  - Students, courses, grant information, …

Embedded Databases

Python example:

SQLite embedded database.




```
dbcon   = sqlite3.connect('datafile', …)

cursor = dbcon.cursor()
cursor.execute('SELECT ….')

for row in cursor.fetchall():
  print(row)
cursor.close()
```

https://docs.python.org/3/library/sqlite3.html

database system embedded into an application program

Application Design Choices

- Should we check if the date_of_birth is valid at the Webpage or at DB ?
- If we have to increment the rating of all skaters, where can it efficiently be executed ? (Cursor at the application code vs Vs Stored Procedure Vs a single update query in the DB)
- Rule of thumb
  - If the objective is to trap user errors, it is more efficient to do it at client side.
  - Complex iterative processing over large data that requires procedural logic will benefit from stored procedures, as data does not "leave" the database server and therefore does not incur network overhead.
  - For updates on large amounts of data where update to each tuple is independent of other tuples it is best to do regular SQL update – Modern DBMS systems can process tuples in multiple parallel batches, thus providing several leaps of bounds of performance ! bulk updates
  - In your home work however, we encourage you to do most of the programming in the database and not on the client side, so as to explore and learn the features and functionality !