>>

Programming with Databases

- Programming with Databases
- PL/DB Interface
- PL/DB Mismatch

∧ >>

❖ Programming with Databases

So far, we have seen ...

- accessing data via SQL queries
- packaging SQL queries as views/functions
- building functions to return tables
- implementing assertions via triggers

All of the above programming

- is very close to the data
- takes place inside the DBMS

<< ∧ >>

❖ Programming with Databases (cont)

While SQL (+ PLpgSQL) gives a powerful data access mechanism

- it is *not* an application programming language

Complete applications require code to

- handle the user interface (GUI or Web)
- interact with other systems (e.g. other DBs)
- perform compute-intensive work (vs. data-intensive)

"Conventional" programming languages (PLs) provide these.

We need PL + DBMS connectivity.

<< ∧ >>

❖ Programming with Databases (cont)

Requirements of an interface between PL and RDBMS:

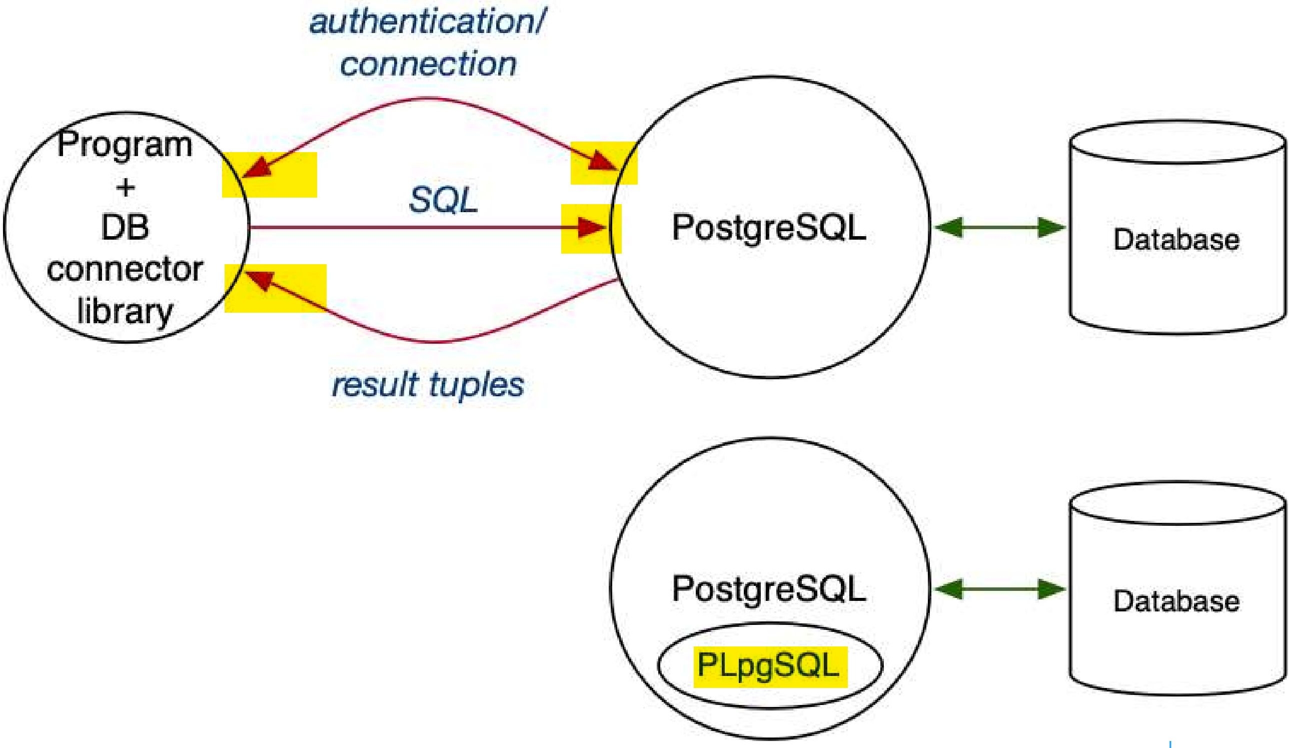
- mechanism for connecting to the DBMS (authentication)
- mechanism for mapping PL "requests" to DB queries
- mechanism for iterating over query results
- mapping betwen tuples and PL objects

Distance between PL and DBMS is variable, e.g.

- **`libpq`** allows C programs to use PG structs
- JDBC transmits SQL strings, retrieves tuples-as-objects

❖ Programming with Databases (cont)

Programming Language / DBMS archtecture:



<< ∧ >>

❖ PL/DB Interface

Common DB access API used in programming languages

```
db = connect_to_dbms(DBname,User/Password);

query = build_SQL("SqlStatementTemplate",values);

results = execute_query(db,query);

while (more_tuples_in(results))
{
    tuple = fetch_row_from(results);
    // do something with values in tuple ...
}
```

This pattern is used in many different libraries:

- Java/JDBC, PHP/PDO, Perl/DBI, Python/Psycopg2, Tcl, ...

&lt;&lt; ∧ &gt;&gt;

❖ PL/DB Interface (cont)

DB access libraries have similar overall structure.

But differ in the details:

- whether object-oriented or procedural flavour
- function/method names and parameters
- how to get data from program into SQL statements
- how to get data from tuples into program variables

Object-relational mappers (ORMs) ...

- aim to hide the details of the database schema and queries
- allow programmers to manipulate objects, not tuples
- potentially use the PLDB connection inefficiently

<< ∧ >>

❖ PL/DB Mismatch

There is a tension between PLs and DBMSs

- DBMSs deal very efficiently with large sets of tuples
- PLs encourage dealing with single tuples/objects

If not handled carefully, can lead to inefficient use of DB.

Note: relative costs of DB access operations from PL:

- establishing a DBMS connection ... very high
- initiating an SQL query ... high
- accessing individual tuple ... small

therefore, only have one connection (which is all you should need anyway), and then do as little queries as possible (i.e. do joins etc for 1 big query, rather than doing 3 smaller queries). This is because the overhead of sending/receiving between the db and program is significant.

<< ^ >>

❖ PL/DB Mismatch (cont)

Consider this (imaginary) PL/DBMS access method:

```
--  establish connection to DBMS
db = dbAccess("DB");
query = "select a,b from R,S where ... ";
--  invoke query and get handle to result set
results = dbQuery(db, query);
--  for each tuple in result set
while (tuple = dbNext(results)) {
    --  process next tuple
    process(tuple['a'], tuple['b']);
}
```

Estimated costs: **dbAccess** = 500ms, **dbQuery** = 200ms, **dbNext** = 10ms

In later cost estimates, ignore **dbAccess** ... same base cost for all examples

<< ^ >>

❖ PL/DB Mismatch (cont)

Example: find mature-age students (e.g. 10000 students, 500 over 40)

```
query = "select * from Student";
results = dbQuery(db, query);
while (tuple = dbNext(results)) {
    if (tuple['age'] >= 40) {
        --  process mature-age student
    }
}
```

We transfer 10000 tuples from DB, 9500 are irrelevant

Cost = 1*200 + 10000*10 = 100200ms = 100s

<< ∧ >>

❖ PL/DB Mismatch (cont)

E.g. should be implemented as:

```
query = "select * from Student where age >= 40";
results = dbQuery(db, query);
while (tuple = dbNext(results)) {
    --  process mature-age student
}
```

Transfers only the 500 tuples that are needed.

Cost = 1*200 + 500*10 = 5200ms = 5s

<< ∧ >>

❖ PL/DB Mismatch (cont)

Example: find info about all marks for all students

```
query1 = "select id,name from Student";
res1 = dbQuery(db, query1);
while (tuple1 = dbNext(res1)) {
    query2 = "select course,mark from Marks"
             + " where student = " + tuple1['id'];
    res2 = dbQuery(db,query2);
    while (tuple2 = dbNext(res2)) {
        --  process student/course/mark info
    }
}
```

E.g. 10000 students, each with 8 marks, ⇒ run 10001 queries

Cost = 10001*200 + 80000*10 = 2800s = 46min

<< ∧

❖ PL/DB Mismatch (cont)

E.g. should be implemented as:

```
query = "select id,name,course,mark"
        + " from Student s join Marks m "
        + " on (s.id=m.student)"
results = dbQuery(db, query);
while (tuple = dbNext(results)) {
    --  process student/course/mark info
}
```

We invoke 1 query, and transfer same number of tuples.

Cost = 1*200ms + 80000*10ms = 800s = 13min

Produced: 25 Oct 2020