Example schema:

```sql
CREATE TABLE Student (SID INTEGER PRIMARY KEY,
  name CHAR(30),
  age INTEGER,
  GPA FLOAT);
CREATE TABLE Take (SID INTEGER,
  CID CHAR(10),
  PRIMARY KEY(SID, CID));
CREATE TABLE Course (CID CHAR(10) PRIMARY KEY,
  title VARCHAR(100) UNIQUE);
```

**Motivation**

Pros and cons of SQL:

- Very high-level, possible to optimize
- Not tuned to support general-purpose computation
  - Oracle as a calculator? `SELECT 142857*3 FROM DUAL;`
  - Strictly less expressive than general-purpose languages
    - SQL2 has no recursion and cannot even compute factorial!

Solutions:

- Augment SQL: Oracle’s PL/SQL
- Use SQL together with a general-purpose programming language: embedded SQL, dynamic SQL

**Oracle PL/SQL**

**Basics**

Rough form of a PL/SQL program:

```sql
DECLARE
  declarations
BEGIN
  executableStatements
END;
.
RUN;
```

~ DECLARE section is optional
~ . and RUN end the program and execute it
Example: go through students 142–857 and set all GPA’s under 4.0 to 4.0

```sql
DECLARE
    thisSID Student.SID%TYPE;
    thisGPA Student.GPA%TYPE;
BEGIN
    thisSID := 142;
    LOOP
        EXIT WHEN (thisSID > 857);
        SELECT GPA INTO thisGPA
        FROM Student
        WHERE SID = thisSID;
        IF (thisGPA < 4.0) THEN
            UPDATE Student SET GPA = 4.0
            WHERE SID = thisSID;
        END IF;
        thisSID := thisSID + 1;
    END LOOP;
END;
```

RUN;

Basic features:

- Local variable:
  - Use %TYPE to match its type to a column in the schema
  - Use := for assignment; = for comparison

- Branch: IF (...) THEN ... ELSE ... END IF;

- Loop: LOOP ... EXIT WHEN (...); ... END LOOP;

- The usual data modification statements: INSERT, DELETE, UPDATE

- Single-row SELECT: SELECT ... INTO ... FROM ...;
  - Oracle raises an exception if SELECT returns no rows or more than one row

**Cursors**

Inside a PL/SQL program, the result of a SELECT must go somewhere:

- If SELECT returns one row, it can go INTO a variable
- What if SELECT returns multiple rows?

→ Cursor: a variable that runs through the result of a SELECT, row by row

- Declare by: CURSOR cursorName IS query;
- Use inside a cursor loop:
  - Fetch one result row at a time: FETCH cursorName INTO vars;
  - Break the loop when there are no more rows to return:
    ```sql
    EXIT WHEN cursorName%NOTFOUND;
    ```
- OPEN/CLOSE before/after use
If cursor is over a single table and has no aggregates or DISTINCT, we can also modify data through the cursor:

- Follow the declaration by FOR UPDATE
- Use WHERE CURRENT OF cursorName in DELETE or UPDATE

Example: go through all CS145 students and set all GPA’s under 4.0 to 4.0!

∽ Note it is possible to declare a “row” type in Oracle

```
DECLARE
    thisStudent Student%ROWTYPE;
CURSOR CS145Student IS
    SELECT * FROM Student WHERE SID IN
        (SELECT SID FROM Take WHERE CID = ’CS145’)
    FOR UPDATE;
BEGIN
    OPEN CS145Student;
    LOOP
        FETCH CS145Student INTO thisStudent;
        EXIT WHEN (CS145Student%NOTFOUND);
        IF (thisStudent.GPA < 4.0) THEN
            UPDATE Student SET GPA = 4.0
            WHERE CURRENT OF CS145Student;
        END IF;
    END LOOP;
    CLOSE CS145Student;
END;
```

```
RUN;
```

**Stored Procedures**

Creating a PL/SQL stored procedure:

```
CREATE PROCEDURE procedureName (argDeclarations) AS
    localDeclarations
BEGIN
    executableStatements
END;
```

∽ The RUN above creates the procedure, but does not execute it

Running the procedure inside a PL/SQL program:

```
BEGIN
    ...
    procedureName (args);
    ...
END;
```

```
RUN;
```
Dropping the procedure:

```sql
DROP PROCEDURE procedureName;
```

Example: a procedure to enroll students in CS145

```sql
CREATE PROCEDURE CS145Enroll (thisSID IN Take.SID%TYPE) AS
BEGIN
    INSERT INTO Take VALUES(thisSID, 'CS145');
END;
.
RUN;
```

Example: students 142 and 857 enroll in CS145

```sql
BEGIN
    CS145Enroll(142);
    CS145Enroll(857);
END;
.
RUN;
```

**Embedded SQL**

Instead of making SQL do more, embed it into a general-purpose programming language (C in our examples):

1. Host program with special SQL directives and commands
   ➔ Goes through a special DBMS preprocessor to produce:
2. Host program with special DBMS function calls
   ➔ Gets compiled and linked with special DBMS library to produce:
3. Executable code that communicates with the DBMS

Main issues in embedding SQL within a program:

- Which statements are SQL?
  - Those introduced with `EXEC SQL`
- How are values passed from the program into SQL commands?
  - *Shared Variables*: variables that are accessible to both SQL and the host language
    - In C, variables are used normally
    - In SQL, they must be preceded by a colon
- How are results of SQL queries returned into program variables?
  - If query returns a single row, use `SELECT INTO`
  - If query returns many rows, use a cursor
    - Similar to PL/SQL cursors, with minor syntactic differences
    - In Oracle, `WHENEVER` can be used to break cursor loops
Example: go through all CS145 students and change all GPA’s

```sql
EXEC SQL BEGIN DECLARE SECTION;
int thisSID;
float thisGPA;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE CS145Student CURSOR FOR
  SELECT SID, GPA FROM Student WHERE SID IN
    (SELECT SID FROM Take WHERE CID = 'CS145')
  FOR UPDATE;
EXEC SQL OPEN CS145Student;
EXEC SQL WHENEVER NOT FOUND DO break;
while (1) {
  EXEC SQL FETCH CS145Student INTO :thisSID, :thisGPA;
  printf("SID %d current GPA %f\n", thisSID, thisGPA);
  printf("Enter new GPA: ");
  scanf("%f", &thisGPA);
  EXEC SQL UPDATE Student SET GPA = :thisGPA
    WHERE CURRENT OF CS145Student;
}
EXEC SQL CLOSE CS145Student;
```

Dynamic SQL

- Embedded SQL is fine for fixed applications (e.g., a program used by the registrar to enroll students in classes)
- But we cannot use it to write a program like sqlplus, because we do not know in advance what SQL statements the user will enter

~ Two special statements in embedded SQL which make it `dynamic`:

- `PREPARE`
- `EXECUTE`

Example: let’s write our own sqlplus!

```sql
EXEC SQL BEGIN DECLARE SECTION;
char query[MAX_QUERY_LENGTH];
EXEC SQL END DECLARE SECTION;

while (1) {
  /* issue SQL> prompt */
  /* read user input into query */
  EXEC SQL PREPARE q FROM :query;
  EXEC SQL EXECUTE q;
}
```