Limitations of SQL

- Most programming language are Turing-complete
- SQL is not

PSM and PL

- **Persistent Stored Modules (PSM)**
  - Commonly known as Stored Procedures
  - Stored in the database just like other schema objects
- **Procedural Languages (PL)**
  - Programming language for writing stored procedures
  - Based on SQL, Java, C#, Perl, Python, ...

Oracle PL/SQL

- SQL + things you would expect from a conventional programming language:
  - Variables and types
  - Control flow statements
  - Procedures and functions
  - Packages
- No just for creating stored procedures e.g. can be used like other SQL statements
- [http://sun.calstatela.edu/~cysun/documentatio
on/oracle/appdev.102/b14261/toc.htm](http://sun.calstatela.edu/~cysun/documentatio
on/oracle/appdev.102/b14261/toc.htm)

Example: Hello World

```plsql
/** A simple PL/SQL example */
begin
    -- print out "Hello World!"
    dbms_output.put_line( 'Hello World!' );
    end;
/
```

- NOTE: the slash (/) at the end execute the PL/SQL code

Comments

- C-style comments: /* comments */
- SQL-style comments: -- comments
Output

◆ **DBMS_OUTPUT** is one of the built-in packages in Oracle
  ◆ **PUT_LINE()**
  ◆ Display the content of the output buffer in SQL*Plus
    ◆ **SET SERVEROUTPUT ON**

Block Structure

[DECLARE
  declaration_statements
BEGIN
  executable_statements
[EXCEPTION
  exception_handling_statements
END;]

Example: Sum

```sql
declare
  a integer := 10;
  b integer default 2;
  s integer;
begin
  b := 5;
  s := a + b;
  dbms_output.put_line( 'sum is ' || s );
end;
```

◆ **NOTE:** Be careful with SQL keywords

Variable Types

◆ All SQL types
◆ Some PL/SQL types
  ◆ boolean: true, false, or null
  ◆ string: same as varchar2
  ◆ record: composite type
  ◆ cursor
  ◆ Collection and object types

TYPE and ROWTYPE

Type of a table column:
```sql
price items.price\texttt{type};
```

Type of a table row:
```sql
item items\texttt{rowtype};
```

Operators

◆ Assignment
  ◆ :=
◆ Arithmetic
  ◆ +, -, *, /, mod
◆ Comparison
  ◆ =
  ◆ >, >=, <, <=
  ◆ !=, <>, ^, ^=
◆ Logical
  ◆ AND, OR, NOT
◆ Concatenation
  ◆ ||
◆ SQL
  ◆ LIKE, IS NULL
  ◆ IN, BETWEEN...AND
  ◆ All functions
**Example: Price Cap**

```sql
declare
    l_price items.price%type;
begin
    select max(price) into l_price from items;
    if l_price <= 9.99 then
        dbms_output.put_line( ' highest price is ' || l_price);
    else
        update items set price = 9.99 where price > 9.99;
        dbms_output.put_line( 'price capped at 9.99.' );
    end if;
end;
```

**Naming Conventions**

- We want to avoid using the same names for variables and table columns.
- A simple naming convention:
  - Prefix local variable with `l_`
  - Prefix package global variable with `g_`
  - Prefix parameters with `p_`

**SELECT...INTO**

```sql
SELECT select_list INTO variable_list
    FROM table_list
    [WHERE condition]
    [ORDER BY order_list];
```

- SELECT result must be a *single row*.

**Branch Statement**

```sql
IF condition1 THEN
    statements1
ELSIF condition2 THEN
    statements2
ELSE
    statements3
END IF;
```

- NOTE: don’t forget the semicolon (;) after END IF.

**CASE Statement**

```sql
CASE expression
    WHEN value1 THEN
        statements
    [WHEN value2 THEN
        statements]
    [ELSE
        statements]
END CASE;
```

- Note the difference between CASE Statement and CASE Expression.

**Example – Factorial**

```sql
declare
    n integer;
    factorial integer := 1;
    i integer := 1;
begin
    n := 5;
    while i <= n loop
        factorial := factorial * i;
        i := i+1;
    end loop;
    dbms_output.put_line( n || '!' = ' ' || factorial );
end;
```
Loop Statements

LOOP
  statements
  EXIT WHEN condition;
  statements
END LOOP;

FOR loop_variable IN [REVERSE]
  lower_bound..upper_bound
  LOOP
  statements
END LOOP;

Cursors

An iterator of a collection of tuples
We can use a cursor to process the rows returned by a SELECT statement
### Cursors

- An iterator of a collection of tuples
- We can use a cursor to process the *rows* returned by a SELECT statement

### Example: Random Output

```sql
declare
    l_name string(32);
    l_price number;
    cursor c is
        select name, price from items;
begin
    open c;
    fetch c into l_name, l_price;
    while c%found loop
        if dbms_random.random > 0 then
            dbms_output.put_line( l_name || ' ' || l_price );
        end if;
        fetch c into l_name, l_price;
    end loop;
    close c;
end;
```

### Attributes

- PL/SQL objects like tables, rows, columns, and cursors have *attributes* associated with them.
- Attributes can be accessed with the `%` operator.
- Some useful attributes:
  - Column attributes: `%TYPE`
  - Table attributes: `%ROWTYPE`
  - Cursor attributes: `%FOUND`, `%NOTFOUND`
Cursor FOR Loop

FOR record_name IN cursor_name LOOP
  statements
END LOOP;

Cursors with Parameters

declare
  cursor c (p_min_price number, p_max_price number) is
  select name, price from items
  where price >= p_min_price
  and price <= p_max_price;
begin
  open c (1.99, 19.99);
  close c;
  open c (99.99, 199.99);
  close c;
end;

Generate Random Number and Strings

- Random numbers
  - dbms_random.random()
  - dbms_random.value()
  - dbms_random.value(low, high)
- Random strings
  - dbms_random.string('U', length)
  - dbms_random.string('L', length)
  - dbms_random.string('A', length)

Example: Exception

declare
  l_price items.price%type;
begin
  select price into l_price from items;
  dbms_output.put_line( l_price );
exception
  when too_many_rows
  then
    dbms_output.put_line( 'there are too many prices.' );
end;

NOTE: the program does not resume after an exception is handled.

System Exceptions

- Some predefined system exceptions:
  - TOO_MANY_ROWS
  - ZERO_DIVIDE
  - INVALID_NUMBER
  - SELF_IS_NULL
  - SUBSCRIPT_OUTSIDE_LIMIT
  - LOGIN_DENIED
  - ...
  - OTHERS
    - Error code is stored in SQLCODE

User Defined Exception

DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
    statements
END;
About PL/SQL Programming

◆ It's just programming like you always do
◆ Bring out your CS201 textbook and do some exercises with PL/SQL
◆ Ask "How to do X" questions in the class forum
◆ Avoid re-implementing SQL
  - For example, to compute max(price), use SELECT MAX(price) instead of a cursor to iterate through all tuples