

# COMP3311 Week 5



By Manhua Lu

# Announcements

- Assignment 1 Due next Friday (22 March @10pm)
  - Make sure to check your submission status on WebCMS
  - Make sure it loads with no errors before submitting
  - This week will also be helpful for the last few questions
- No Quiz this week!
- No tutorials or Quiz next week (Flex week)
- Help sessions in K17 G05
  - Tue 2-4pm, Wed 2-4pm, Thu 3-5pm

# Learning Objectives

**01**

→ SQL Views

**02**

→ PL/pgSQL Functions

**03**

→ SQL Functions

**04**

→ Combining SQL Statements w/ Functions

**05**

→ Putting it all to practice...!

**01**

# **SQL Views**

# 01



## SQL Views

- A “stored” query
- To make querying simpler (and more reusable)
- Return a **view of the database off some query**
  - This view can then be **used in other queries**

# 01

## SQL Views

- Order matters!!
  - i.e. You must declare the view before using it

```
create view
|   CourseMarksAndAverages(course, term, student, mark, avg)
as
|   select  s.code, termName(t.id), e.student, e.mark, avg(mark)
|   from    CourseEnrolments e
|   join    Courses c on c.id = e.course
|   join    Subjects s on s.id = c.subject
|   join    Terms t on t.id = c.term
|   ;

select  course, term, student, mark
from    CourseMarksAndAverages
where   mark < avg;
```

# 01 → SQL Views

This is what the first few questions in the assignment template sql file is doing

```
-- Question 1 --

/**
 * Write a SQL View, called Q1, that:
 * Retrieves the 10 movies with the highest number of votes.
 */
CREATE OR REPLACE VIEW Q1(Title, Year, Votes) AS
-- TODO: Write your SQL query here
;
```

In the 'Example Output' they get you to select the SQL view (Q1) and you should display the same output

Q1

Dump 1

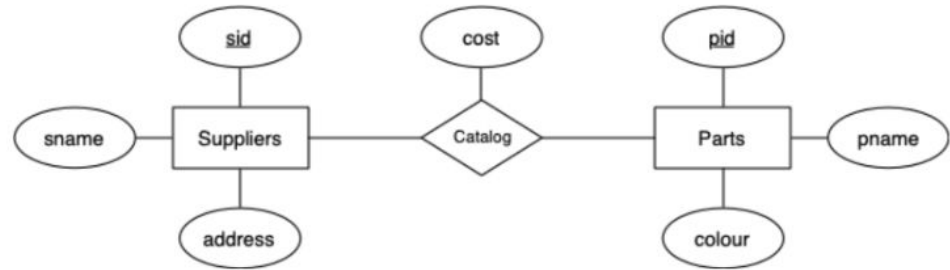
IMDB=# SELECT * FROM Q1;			
title	year	votes	
The Dark Knight	2008	2843738	
Good Will Hunting	1997	1054895	
Watchmen	2009	578999	
Super 8	2011	366264	
Ghost Rider	2007	250940	
Luca	2021	190404	
The Drop	2014	161287	
Get Hard	2015	146177	
Licorice Pizza	2021	138879	
Wild Hogs	2007	122967	
(10 rows)			

**01**

## SQL Views Question

Q. Find the pids of the most expensive part(s) supplied by suppliers named "Yosemite Sham".

Consider the following data model for a business that supplies various parts:



Based on the ER design and the above considerations, here is a relational schema to represent this scenario:

```
create table Suppliers (
    sid integer primary key,
    sname text,
    address text
);
create table Parts (
    pid integer primary key,
    pname text,
    colour text
);
create table Catalog (
    sid integer references Suppliers(sid),
    pid integer references Parts(pid),
    cost real,
    primary key (sid,pid)
);
```



**02**

# **PL/pgSQL Functions**

## 02

## → PL/pgSQL Functions

- Sometimes a view isn't enough, sometimes we want **more flexibility**
- We need a function
- PL/pgSQL is a PostgreSQL procedural programming language

```
create or replace
|   funcName(param1 type, param2 type, ...) returns type
as
$$
declare
|   variable1 type;
|   variable2 type;
begin
|   -- code for function
end;
$$ language plpgsql
;
```

02

## PL/pgSQL Functions Factorial Example

```
create or replace function
|   factorial (n integer) returns integer
as $$
declare
|   i integer;
|   fac integer := 1;
begin
|   for i in 1..n loop
|       fac := fac * i;
|   end loop;
|   return fac;
end;
$$ language plpgsql;
```

02

## PL/pgSQL Functions Withdraw Example

SQL inside  
PL/pgSQL  
function →

```
create or replace function
|   withdraw(acctNum text, amount integer) returns text
as $$
declare
|   bal integer;
begin
|   select  balance into bal
|   from    Accounts
|   where   acctNo = acctNum;

|   if bal < amount then
|       return 'Insufficient Funds';
|   else
|       update Accounts
|       set    balance = balance - amount
|       where  acctNo = acctNum;

|       select  balance into bal
|       from    Accounts
|       where   acctNo = acctNum;

|       return 'New Balance: ' || bal;
|   end if;
end;
$$ language plpgsql;
```

**03**

# **SQL Functions**

# 03

## SQL Functions

- Differences between SQL functions and PL/pgSQL functions:
  - SQL function bodies are **a single SQL statement**
  - SQL functions **can use positional parameter notation**
    - e.g. \$1, \$2, \$3...
  - SQL functions have **NO return**
    - Result is the result of the SQL statement
  - Return types can be atomic (booleans, integers, string, floats), tuple, or **setof** tuples

**03**

## → SQL Functions vs PL/pgSQL Functions

```
create or replace function  
|   add(integer, integer) returns integer  
as $$  
|   select $1 + $2;  
$$ language sql;
```

```
create or replace function  
|   add(n1 integer, n2 integer) returns integer  
as $$  
begin  
|   return n1 + n2;  
end;  
$$ language plpgsql;
```

# 03

## SQL Functions vs PL/pgSQL Functions

```
create or replace function
|   fac(int) returns int
as $$
|   select case
|       |       |       |       when $1 = 0 then 1
|       |       |       |       else $1 * fac($1 - 1)
|       |       |       |       end;
$$ language sql;
```

“case when” is like a switch statement for SQL statements

```
create or replace function
|   fac(n int) returns int
as $$
begin
|   if (n = 0) then
|       return 1;
|   else
|       return n * fac(n-1);
|   end if;
end;
$$ language plpgsql;
```



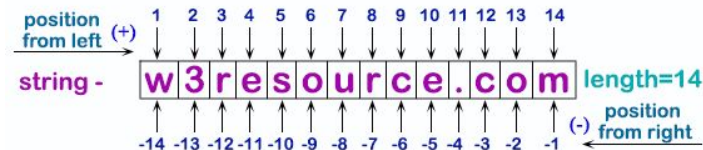
**04**

**Putting it all to practice...!**

# PostgreSQL substring() function

Extracts specific number of characters from specific position

Syntax : **substring ( string, str\_pos, ext\_char )**



Example :

count from left (+)   
**substring ( "w3resource.com", 5, 6 )**  
Result : **source**

Example :

count from left (+)   
**substring ( "w3resource.com", 5 )**  
Result : **source.com**

Example :

negative str\_pos ignore value and start counting from 1st position and return all   
**substring ( "w3resource.com", -5 )**  
Result : **w3resource.com**

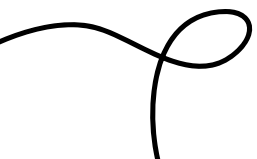
Example :

negative str\_pos ignore value and start counting from 1st position and extracts ext\_char   
**substring ( "w3resource.com", -5, 10 )**  
Result : **w3re**

Example :

count from left (+)   
**substring ( "w3resource.com" FROM 5 FOR 6 )**  
Result : **source**

1. `~` : Case-sensitive, compares two statements, returns true if the first string is contained in the second
2. `~*` : Case-insensitive, compares two statements, returns true if the first string is contained in the second
3. `!~` : Case-sensitive, compares two statements, returns false if the first string is contained in the second
4. `!~*` : Case-insensitive, compares two statements, return false if the first string is contained in the second



## Simple STRING\_AGG Example

The following example will rollup all currency names and separate each using a forward slash.

```
SELECT STRING_AGG(name, '/') as output  
FROM [Sales].[Currency]
```

Results		Messages
output		
1	Afghani/Algerian Dinar/Argentine Peso/Armenian Dram/Aruban Guilder/Austr...	