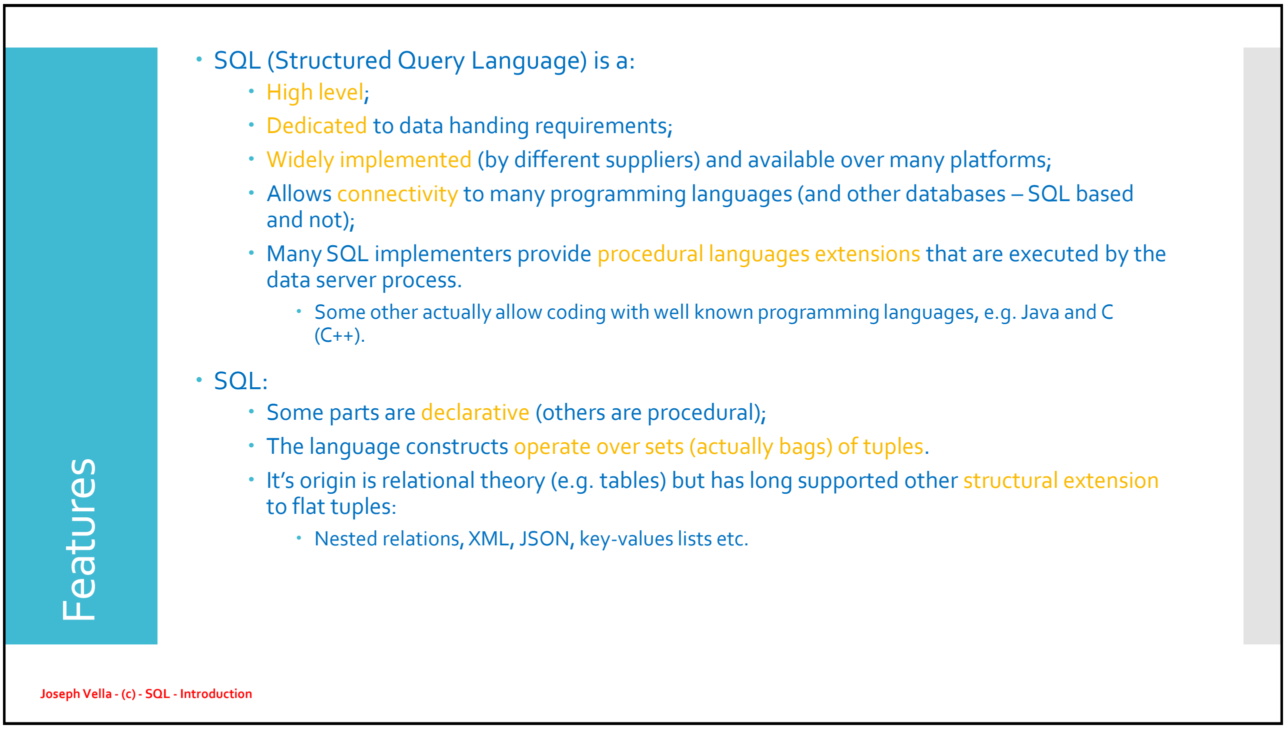
Prof. Joseph G. Vella ©
joseph.g.vella@um.edu.mt

1

Quick overview of SQL

Joseph Vella - (c) - SQL - Introduction

2

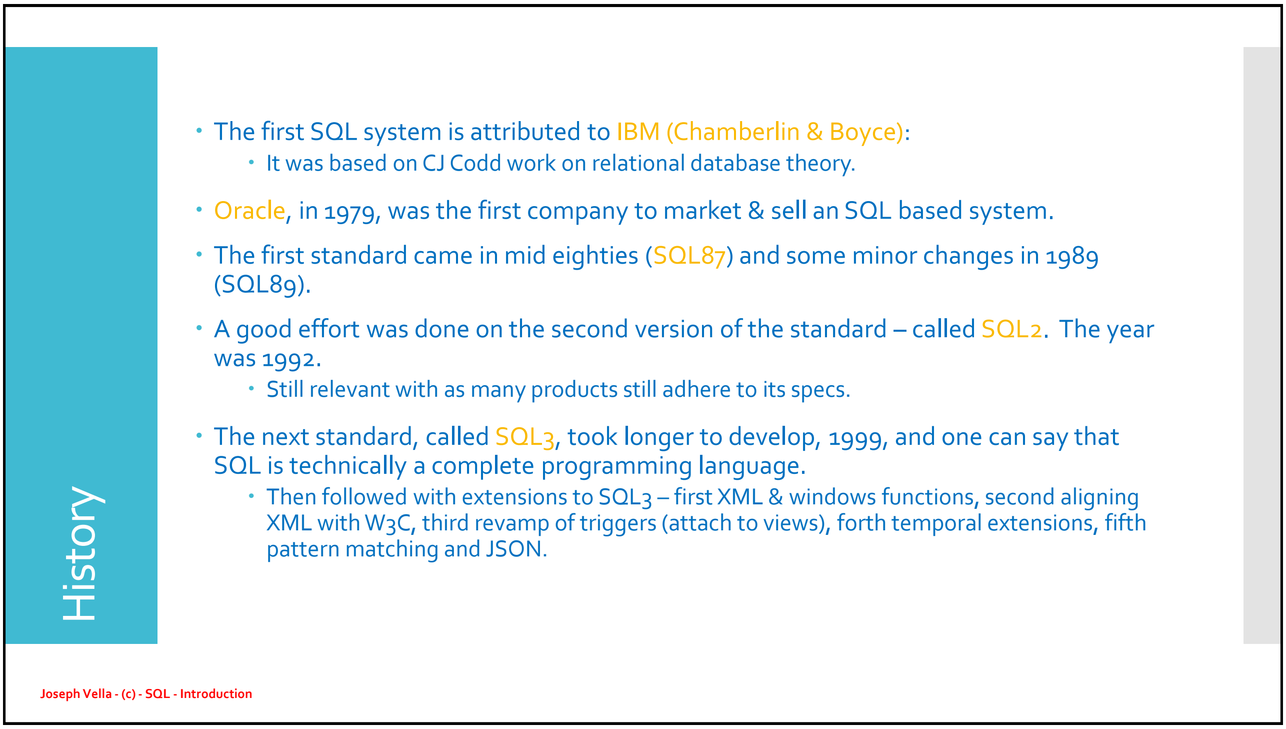


Features

- SQL (Structured Query Language) is a:
  - High level;
  - Dedicated to data handing requirements;
  - Widely implemented (by different suppliers) and available over many platforms;
  - Allows connectivity to many programming languages (and other databases – SQL based and not);
  - Many SQL implementers provide procedural languages extensions that are executed by the data server process.
    - Some other actually allow coding with well known programming languages, e.g. Java and C (C++).
- SQL:
  - Some parts are declarative (others are procedural);
  - The language constructs operate over sets (actually bags) of tuples.
  - It's origin is relational theory (e.g. tables) but has long supported other structural extension to flat tuples:
    - Nested relations, XML, JSON, key-values lists etc.

Joseph Vella - (c) - SQL - Introduction

3



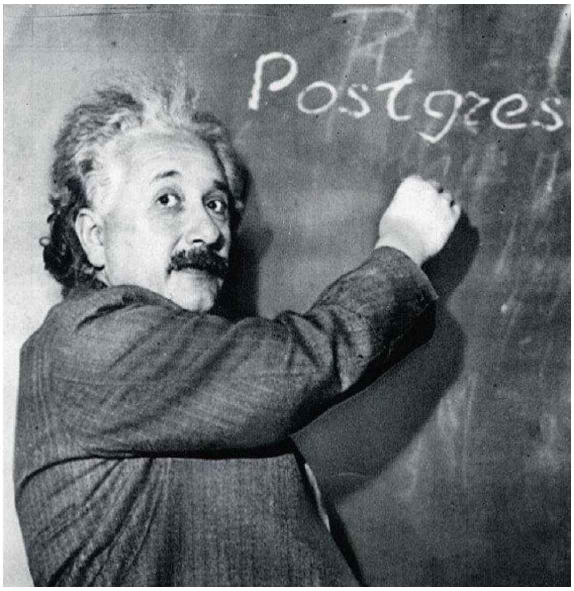
History

- The first SQL system is attributed to IBM (Chamberlin & Boyce):
  - It was based on CJ Codd work on relational database theory.
- Oracle, in 1979, was the first company to market & sell an SQL based system.
- The first standard came in mid eighties (SQL87) and some minor changes in 1989 (SQL89).
- A good effort was done on the second version of the standard – called SQL2. The year was 1992.
  - Still relevant with as many products still adhere to its specs.
- The next standard, called SQL3, took longer to develop, 1999, and one can say that SQL is technically a complete programming language.
  - Then followed with extensions to SQL3 – first XML & windows functions, second aligning XML with W3C, third revamp of triggers (attach to views), forth temporal extensions, fifth pattern matching and JSON.

Joseph Vella - (c) - SQL - Introduction

4

Option taken … enough said!



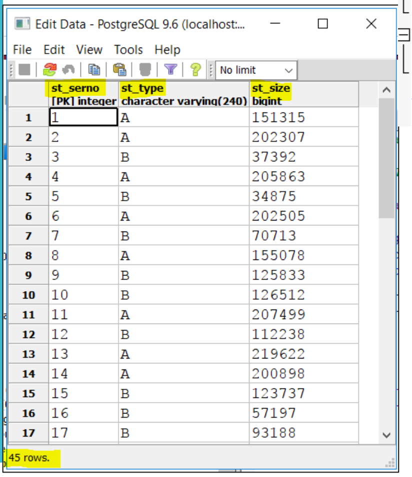
General Structure of simple SELECT statement is:

```
   SELECT [DISTINCT] list of attributes
     FROM list of tables
    WHERE row level selection condition expression
ORDER BY list of attributes;
```

Rows Selection

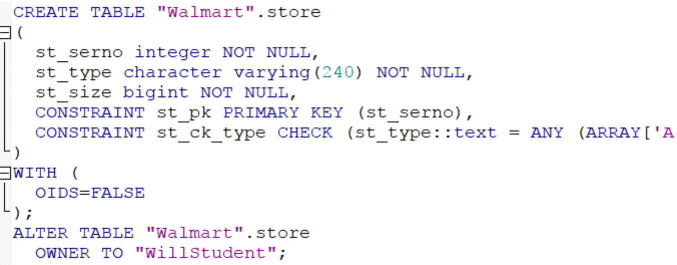
Row Restrictions (aka Selection predicates)

Overview of table Walmart.stores


Edit Data - PostgreSQL 9.6 (localhost:...
File Edit View Tools Help
No limit

	st_serno [PK] integer	st_type character varying(240)	st_size bigint
1	1	A	151315
2	2	A	202307
3	3	B	37392
4	4	A	205863
5	5	B	34875
6	6	A	202505
7	7	B	70713
8	8	A	155078
9	9	B	125833
10	10	B	126512
11	11	A	207499
12	12	B	112238
13	13	A	219622
14	14	A	200898
15	15	B	123737
16	16	B	57197
17	17	B	93188

45 rows.



```
CREATE TABLE "Walmart".store
(
  st_serno integer NOT NULL,
  st_type character varying(240) NOT NULL,
  st_size bigint NOT NULL,
  CONSTRAINT st_pk PRIMARY KEY (st_serno),
  CONSTRAINT st_ck_type CHECK (st_type::text = ANY (ARRAY['A
)
WITH (
  OIDS=FALSE
);
ALTER TABLE "Walmart".store
  OWNER TO "WillStudent";
```

- How to retrieve all data in a table? (no selection!?).

```
SELECT st_serno, st_type, st_size
  FROM "Walmart".store;
```

or

```
SELECT *
  FROM "Walmart".store;
```

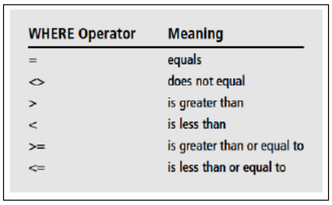
or

```
SELECT s.*
  FROM "Walmart".store s;
```

Simple Selection Conjunction

- Simple selection condition based on conjunction;
  - all conjuncts must be true.

```
SELECT s.st_serno, s.st_size
  FROM "Walmart".store s
WHERE st_type = 'A';
```



WHERE Operator	Meaning
=	equals
<>	does not equal
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

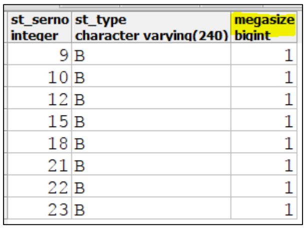
- Simple selection conjunction (AND operator):

```
SELECT st_serno, st_size
  FROM "Walmart".store
WHERE st_type = 'A'
    AND st_size < 100000;
```

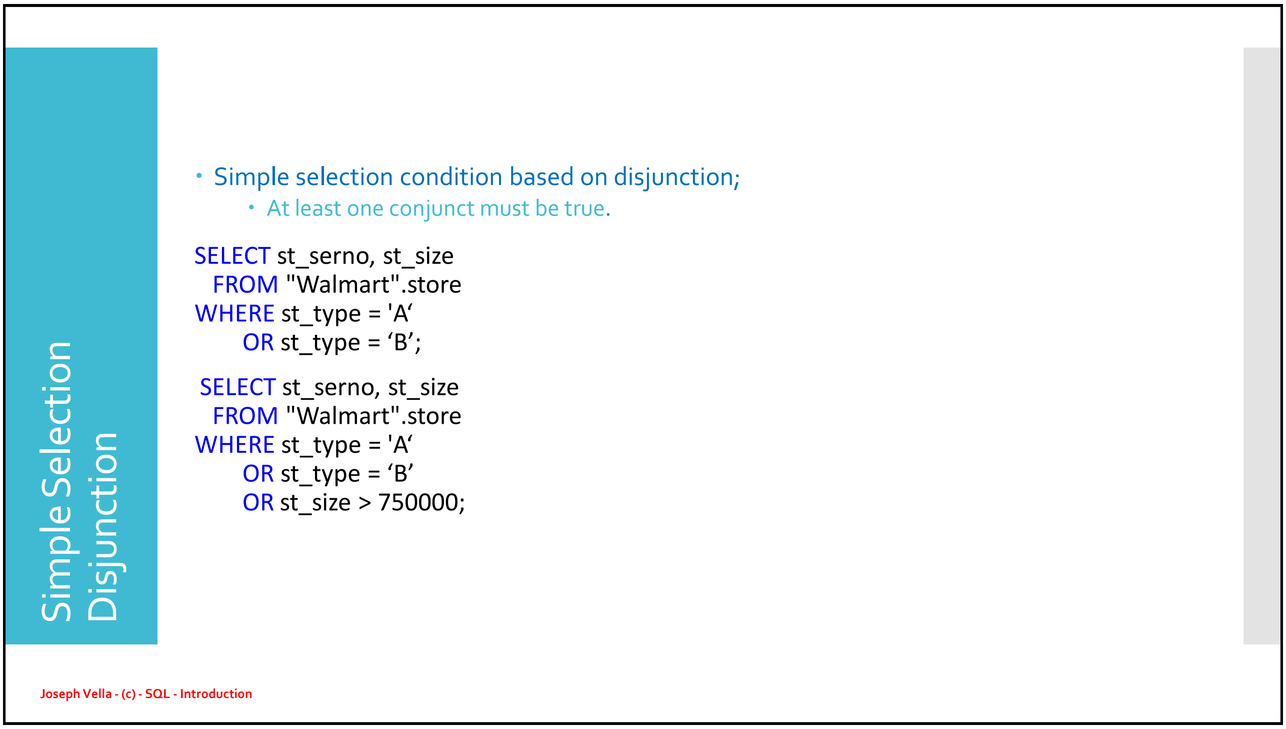
- Again a simple selection with three conjuncts:

```
SELECT st_serno, st_type, st_size/100000 as megasize
  FROM "Walmart".store
WHERE st_type <> 'A'
    AND st_size >= 100000
    AND st_serno < 25;
```

Expression with an alias (for column).



st_serno integer	st_type character varying(240)	megasize bigint
9	B	1
10	B	1
12	B	1
15	B	1
18	B	1
21	B	1
22	B	1
23	B	1



Simple Selection Disjunction

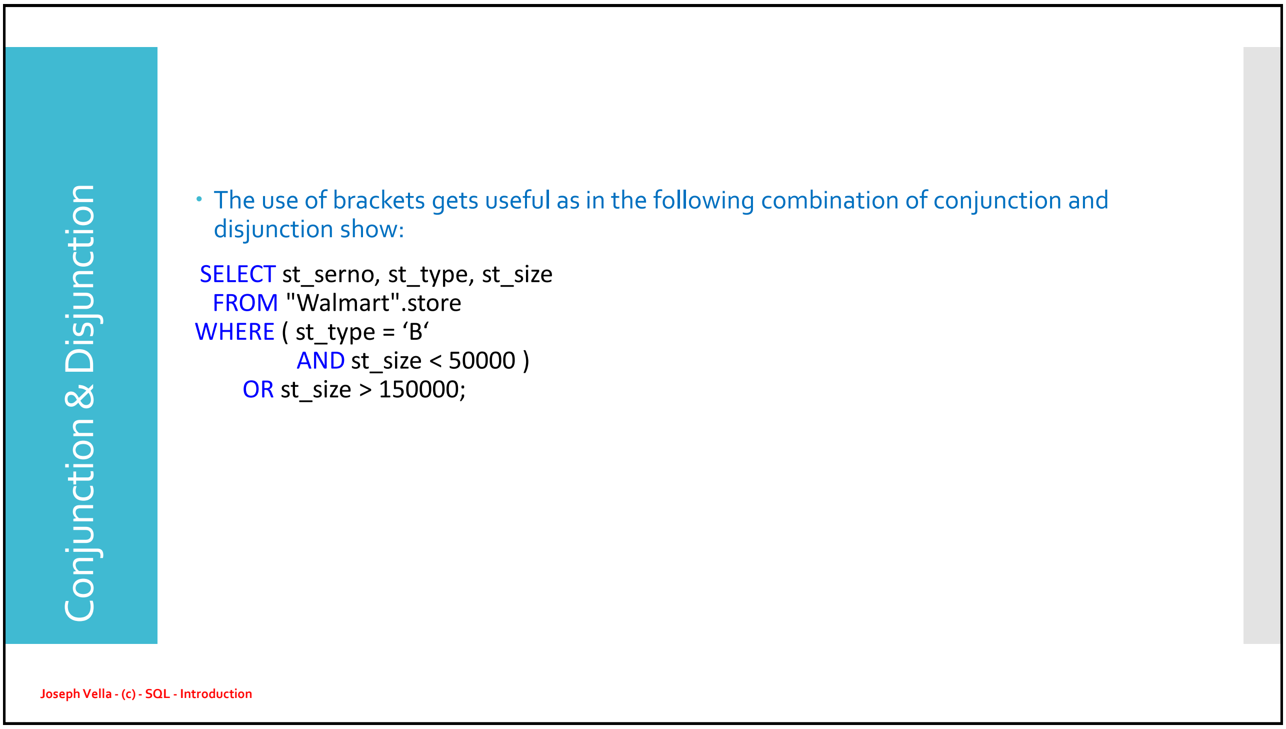
- Simple selection condition based on disjunction;
  - At least one conjunct must be true.

```
SELECT st_serno, st_size
  FROM "Walmart".store
WHERE st_type = 'A‘
      OR st_type = ‘B’;
```

```
 SELECT st_serno, st_size
  FROM "Walmart".store
WHERE st_type = 'A‘
      OR st_type = ‘B’
      OR st_size > 750000;
```

Joseph Vella - (c) - SQL - Introduction

9



Conjunction & Disjunction

- The use of brackets gets useful as in the following combination of conjunction and disjunction show:

```
 SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE ( st_type = ‘B‘
            AND st_size < 50000 )
      OR st_size > 150000;
```

Joseph Vella - (c) - SQL - Introduction

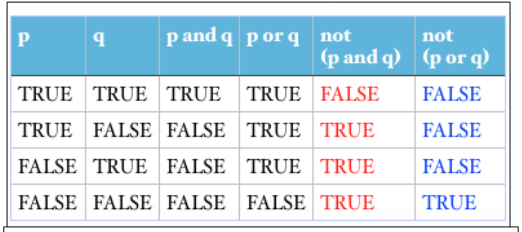
10

Negation (as in NOT)

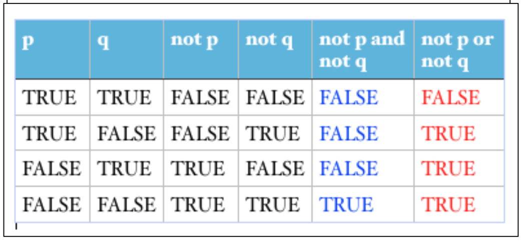
- Simple example of negation as in the "pattern is not found" in over table data or "pattern is not derivable" by an expression over table data.

```
SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE NOT st_type = 'A‘;

-- one could use st_type <> ‘A’ too


SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE NOT ( st_type = 'A‘
                AND st_type = ‘B‘ );


SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE NOT  st_type = 'A‘
    AND NOT  st_type = ‘B‘;
```

De Morgan's Laws



p	q	p and q	p or q	not (p and q)	not (p or q)
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE



p	q	not p	not q	not p and not q	not p or not q
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

The BETWEEN shortcut

- The BETWEEN operator abbreviates an AND expression with greater than or equal to (>=) and less than or equal to (<=) operators:

```
SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE st_size BETWEEN 75000 AND 100000 ;
```

The IN operator

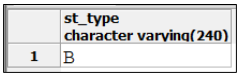
- This is a very useful operator. Let’s start with the syntax (and a simple example):

```
SELECT st_serno, st_type, st_size
  FROM "Walmart".store
WHERE st_type IN ( 'A', ‘B’); -- think of ( 'A', ‘B’) as a set of tuples {[attr:’A’],[attr:’B’]}
```

- But look at this data driven query:

```
SELECT st_out.st_serno, st_out.st_type, st_out.st_size
   FROM "Walmart".store AS st_out
  WHERE st_out.st_type
     IN (SELECT st_in.st_type
          FROM "Walmart".store AS st_in
         WHERE st_in.st_size BETWEEN 25000 AND 35000
        );
```

The inner query computes “B”.



```
     st_type
     character varying(240)
1    B
```

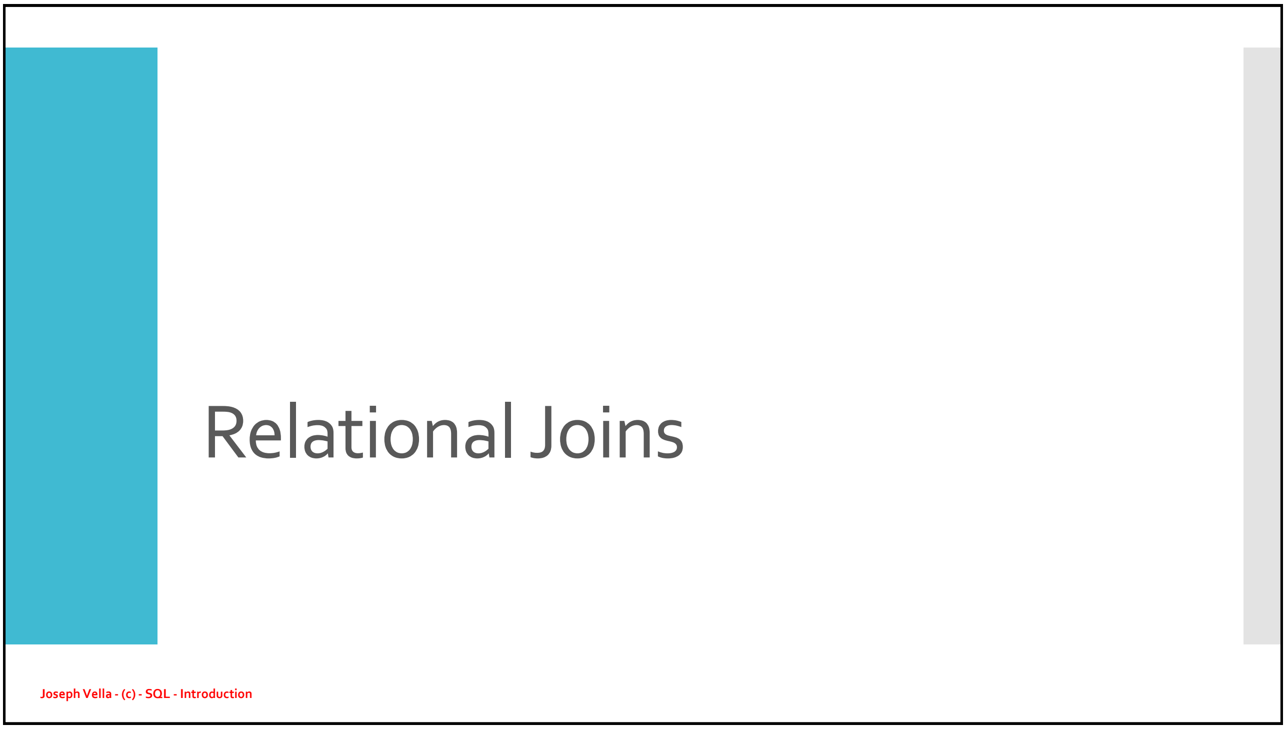
The Nulls

- **Nulls reduce our logic to three valued!?**
  - E.g.
    - SELECT 1+null "result"; -> null
    - SELECT 1+coalesce(null,0) "result"; -> 1
  - COALESCE() function
    if first argument’s value is null then replace it with second argument’s value

```
SELECT fe_store, fe_date, fe_temp_in_f, fe_fuel, fe_markdown1
   FROM "Walmart".feature
WHERE fe_store = 1
     AND fe_markdown1 IS NULL;
```

- And how to handle it on the output:

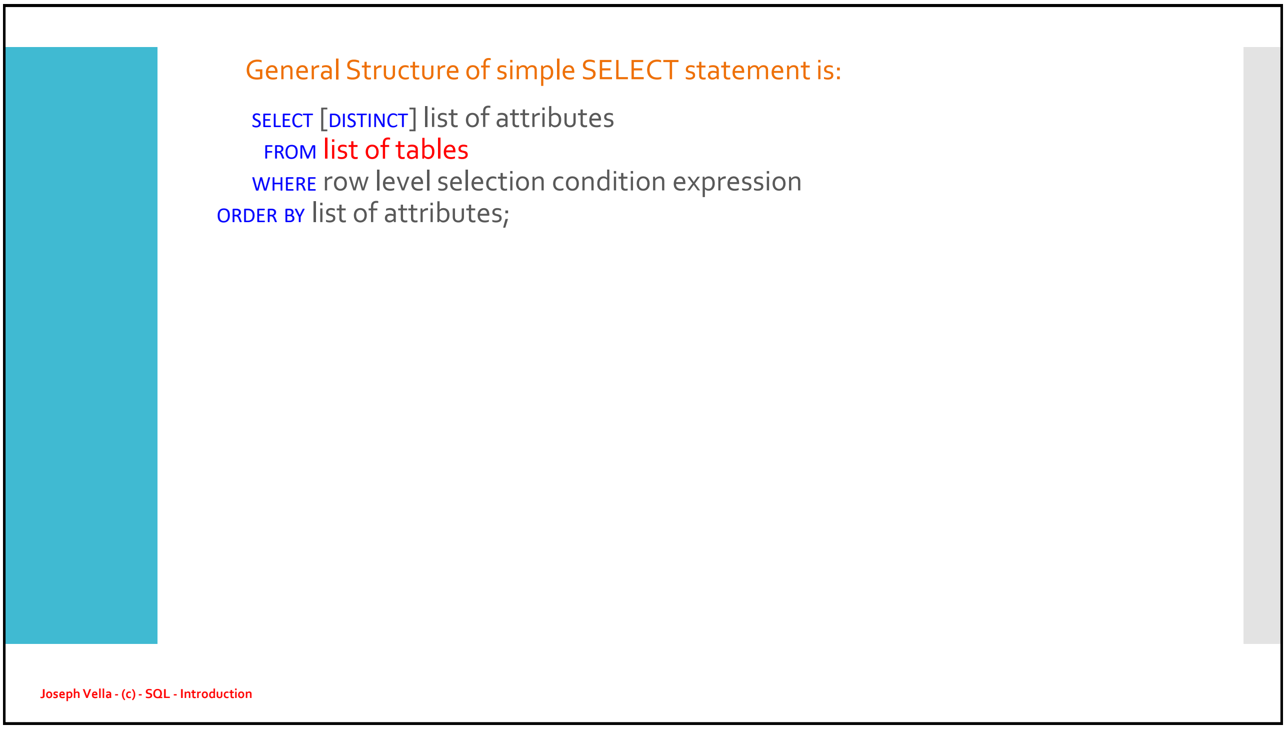
```
SELECT fe_store, fe_date, fe_temp_in_f, fe_fuel, COALESCE (fe_markdown1,0)
  FROM "Walmart".feature
WHERE fe_store = 1
     AND fe_markdown1 IS NULL;  -- negation of IS NULL ->>> IS NOT NULL
```



Relational Joins

Joseph Vella - (c) - SQL - Introduction

15



General Structure of simple SELECT statement is:

```
 SELECT [DISTINCT] list of attributes
   FROM list of tables
 WHERE row level selection condition expression
ORDER BY list of attributes;
```

Joseph Vella - (c) - SQL - Introduction

16

CJ Date example schema

- CJ Date database is well known with simple structure and easy to recall it's data content (i.e. easy to verify yourself that all is well with your query attempt).
  - PLEASE note we add some tuples to the database to make some queries and their responses easier to follow.
- The database is about works scheduling and has four tables called:
  - Product (p), Supplier (s), Job (j) and Works Schedule (spj).
  - Each table has a PK and FKs are found in Works table that relates (in many to one relationship) with each of the other tables.

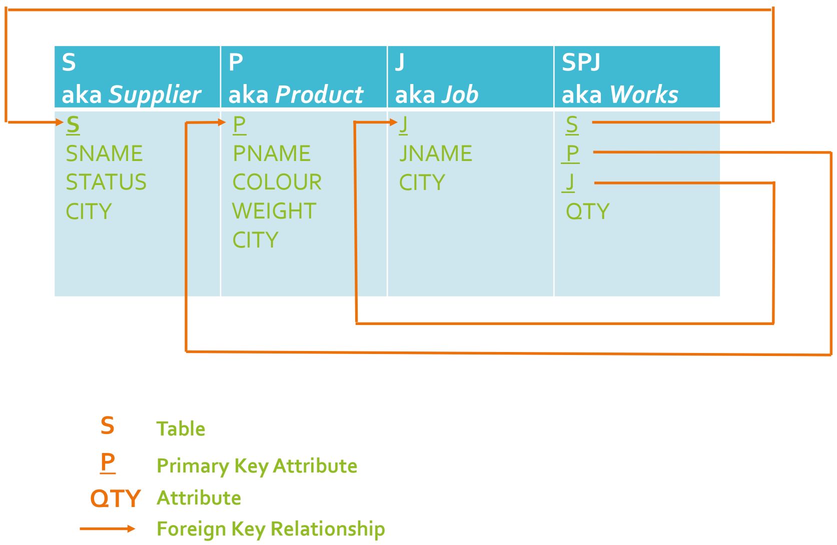
NOTE:

- Please run the following amendments:

```
ALTER TABLE date.p ALTER COLUMN weight
TYPE integer USING (weight::integer);
INSERT INTO date.p( p, pname, colour, weight, city)
VALUES ('P7','LOCK NUT','GREY',13,'PALO ALTO');
```

Joseph Vella - (c) - SQL - Introduction

Date's SPJ schema

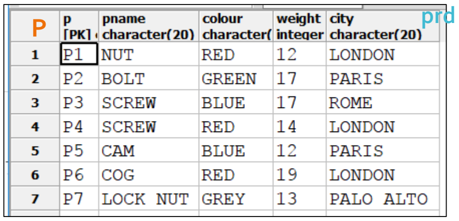


S aka *Supplier*	P aka *Product*	J aka *Job*	SPJ aka *Works*
S	P	J	S
SNAME	PNAME	JNAME	P
STATUS	COLOUR	CITY	J
CITY	WEIGHT		QTY
	CITY		

S Table
P Primary Key Attribute
QTY Attribute
→ Foreign Key Relationship

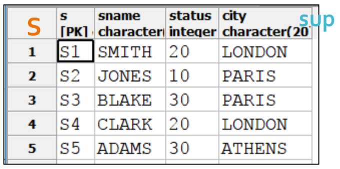
Joseph Vella - (c) - SQL - Introduction

Date's SPJ Data
(with some additions!?)



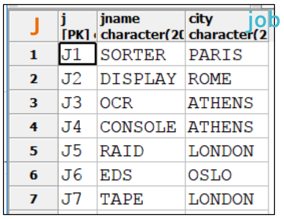
P	p	pname	colour	weight	city
	[PK] ‹	character(20)	character(	integer	character(20)
1	P1	NUT	RED	12	LONDON
2	P2	BOLT	GREEN	17	PARIS
3	P3	SCREW	BLUE	17	ROME
4	P4	SCREW	RED	14	LONDON
5	P5	CAM	BLUE	12	PARIS
6	P6	COG	RED	19	LONDON
7	P7	LOCK NUT	GREY	13	PALO ALTO

prd



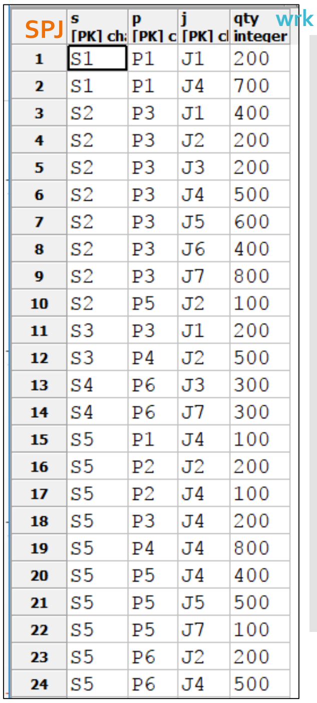
S	s	sname	status	city
	[PK] ‹	character(	integer	character(20
1	S1	SMITH	20	LONDON
2	S2	JONES	10	PARIS
3	S3	BLAKE	30	PARIS
4	S4	CLARK	20	LONDON
5	S5	ADAMS	30	ATHENS

sup



J	j	jname	city
	[PK] ‹	character(2(	character(2
1	J1	SORTER	PARIS
2	J2	DISPLAY	ROME
3	J3	OCR	ATHENS
4	J4	CONSOLE	ATHENS
5	J5	RAID	LONDON
6	J6	EDS	OSLO
7	J7	TAPE	LONDON

job



SPJ	s	p	j	qty
	[PK] ch:	[PK] c	[PK] cl	integer
1	S1	P1	J1	200
2	S1	P1	J4	700
3	S2	P3	J1	400
4	S2	P3	J2	200
5	S2	P3	J3	200
6	S2	P3	J4	500
7	S2	P3	J5	600
8	S2	P3	J6	400
9	S2	P3	J7	800
10	S2	P5	J2	100
11	S3	P3	J1	200
12	S3	P4	J2	500
13	S4	P6	J3	300
14	S4	P6	J7	300
15	S5	P1	J4	100
16	S5	P2	J2	200
17	S5	P2	J4	100
18	S5	P3	J4	200
19	S5	P4	J4	800
20	S5	P5	J4	400
21	S5	P5	J5	500
22	S5	P5	J7	100
23	S5	P6	J2	200
24	S5	P6	J4	500

wrk

Inner Joins
based on the Equality operator

Relational Joins

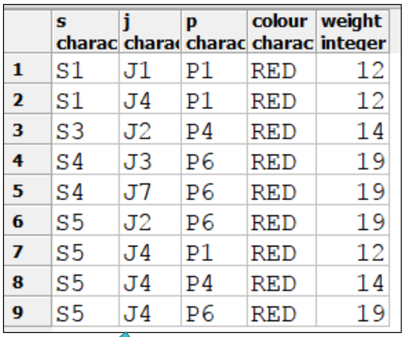
Equi-Join between 2 tables – using FK-PK links

Give work schedule details with product's colour and weight for RED coloured parts.

Note wrk.p is a FK in table spj and prd.p is a PK in table p.

```
SELECT wrk.s, wrk.j, prd.p, prd.colour, prd.weight
  FROM date.spj wrk, date.p prd
 WHERE wrk.p = prd.p
    AND prd.colour = 'RED';
```

```
SELECT wrk.s, wrk.j, prd.p, prd.colour, prd.weight
  FROM date.spj wrk  INNER JOIN
          date.p prd ON (wrk.p = prd.p)
 WHERE prd.colour = 'RED';
```



	s charac	j charac	p charac	colour charac	weight integer
1	S1	J1	P1	RED	12
2	S1	J4	P1	RED	12
3	S3	J2	P4	RED	14
4	S4	J3	P6	RED	19
5	S4	J7	P6	RED	19
6	S5	J2	P6	RED	19
7	S5	J4	P1	RED	12
8	S5	J4	P4	RED	14
9	S5	J4	P6	RED	19

Equi-Join between 3 tables – using FK-PK links

Give work schedule details with product's colour and weight for non RED coloured parts, and supplier's city.

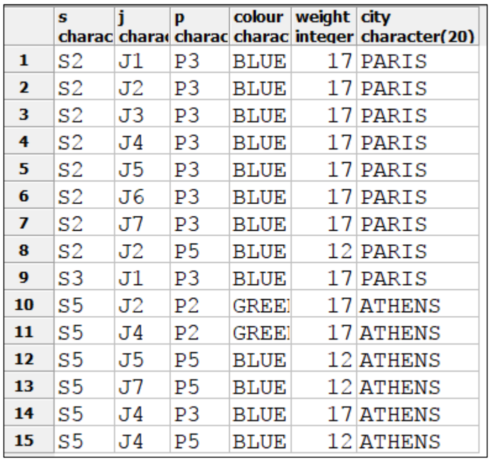
In the first join we have wrk.p is a FK in table spj and prd.p is a PK in table p, and in the second join spl.s is a PK in table s and wrk.s is a FK in table spj

```
SELECT wrk.s, wrk.j, prd.p,
          prd.colour, prd.weight, spl.city
 FROM date.spj wrk, date.p prd, date.s spl
WHERE wrk.p = prd.p
      AND wrk.s = spl.s
      AND prd.colour <> 'RED';
```

```
SELECT wrk.s, wrk.j, prd.p, prd.colour,
          prd.weight, spl.city
  FROM date.spj wrk INNER JOIN
          date.p prd ON (wrk.p = prd.p)
             INNER JOIN date.s spl ON (wrk.s = spl.s)
WHERE prd.colour <> 'RED';
```

Equi-Join between 3 tables – using FK-PK links

```
SELECT wrk.s, wrk.j, prd.p, prd.colour,
           prd.weight, spl.city
  FROM date.spj wrk INNER JOIN
           date.p prd ON (wrk.p = prd.p)
             INNER JOIN date.s spl ON (wrk.s = spl.s)
WHERE prd.colour <> 'RED';
```



	s charac	j chara	p charac	colour charac	weight integer	city character(20)
1	S2	J1	P3	BLUE	17	PARIS
2	S2	J2	P3	BLUE	17	PARIS
3	S2	J3	P3	BLUE	17	PARIS
4	S2	J4	P3	BLUE	17	PARIS
5	S2	J5	P3	BLUE	17	PARIS
6	S2	J6	P3	BLUE	17	PARIS
7	S2	J7	P3	BLUE	17	PARIS
8	S2	J2	P5	BLUE	12	PARIS
9	S3	J1	P3	BLUE	17	PARIS
10	S5	J2	P2	GREE	17	ATHENS
11	S5	J4	P2	GREE	17	ATHENS
12	S5	J5	P5	BLUE	12	ATHENS
13	S5	J7	P5	BLUE	12	ATHENS
14	S5	J4	P3	BLUE	17	ATHENS
15	S5	J4	P5	BLUE	12	ATHENS

Joseph Vella - (c) - SQL - Introduction

23

Equi-Join between 4 tables – using FK-PK links

```
SELECT wrk.s, wrk.j, prd.p, prd.colour, prd.weight, spl.city, job.city
  FROM date.spj wrk, date.p prd, date.s spl, date.j job
WHERE wrk.p = prd.p
     AND wrk.s = spl.s
    AND wrk.j = job.j
    AND prd.colour <> 'RED';
```

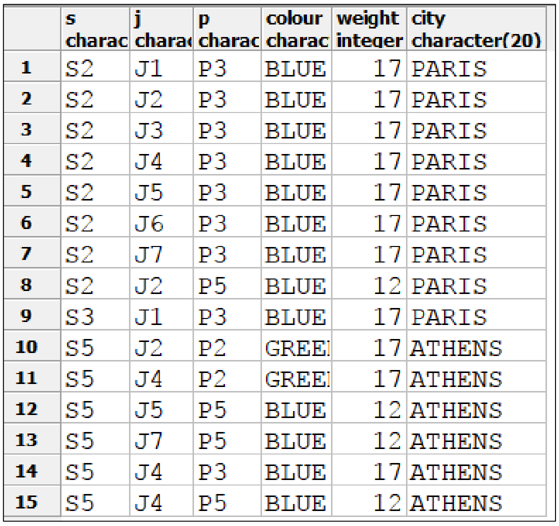
```
SELECT wrk.s, wrk.j, prd.p, prd.colour, prd.weight, spl.city, job.city
  FROM date.spj wrk INNER JOIN
            date.p prd ON (wrk.p = prd.p)  INNER JOIN
            date.s spl ON (wrk.s = spl.s)  INNER JOIN
             date.j job ON (wrk.j = job.j)
WHERE prd.colour <> 'RED';
```

Joseph Vella - (c) - SQL - Introduction

24

Equi-Join between 4 tables – using FK-PK links

```
SELECT wrk.s, wrk.j, prd.p,
        prd.colour, prd.weight,
        spl.city, job.city

  FROM date.spj wrk,
        date.p prd,
        date.s spl,
        date.j job

WHERE wrk.p = prd.p

    AND wrk.s = spl.s

   AND wrk.j = job.j

   AND prd.colour <> 'RED';
```



	s charac	j chara	p charac	colour charac	weight integer	city character(20)
1	S2	J1	P3	BLUE	17	PARIS
2	S2	J2	P3	BLUE	17	PARIS
3	S2	J3	P3	BLUE	17	PARIS
4	S2	J4	P3	BLUE	17	PARIS
5	S2	J5	P3	BLUE	17	PARIS
6	S2	J6	P3	BLUE	17	PARIS
7	S2	J7	P3	BLUE	17	PARIS
8	S2	J2	P5	BLUE	12	PARIS
9	S3	J1	P3	BLUE	17	PARIS
10	S5	J2	P2	GREE	17	ATHENS
11	S5	J4	P2	GREE	17	ATHENS
12	S5	J5	P5	BLUE	12	ATHENS
13	S5	J7	P5	BLUE	12	ATHENS
14	S5	J4	P3	BLUE	17	ATHENS
15	S5	J4	P5	BLUE	12	ATHENS

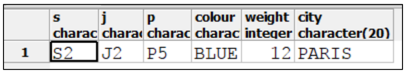
Joseph Vella - (c) - SQL - Introduction

25

Equi-Join between tables using FK-PK links & others

Give details of work schedule for RED coloured parts and where jobs and parts are collocated. In the first join we have wrk.p is a FK in table spj and prd.p is a PK in table p, in the second join spl.s is a PK in table s and wrk.s is a FK in table spj and in the third join condition we require colocation (have the same city value) between project and supplier.

```
SELECT wrk.s, wrk.j, prd.p,
        prd.colour, prd.weight, spl.city

  FROM date.spj wrk,
        date.p prd,
        date.s spl

WHERE wrk.p = prd.p

    AND wrk.s = spl.s

    AND prd.city = spl.city

    AND prd.colour <> 'RED';
```



	s charac	j chara	p charac	colour charac	weight integer	city character(20)
1	S2	J2	P5	BLUE	12	PARIS

```
SELECT wrk.s, wrk.j, prd.p,
        prd.colour, prd.weight, spl.city

  FROM date.spj wrk INNER JOIN

        date.p prd ON (wrk.p = prd.p) INNER JOIN

        date.s spl ON (wrk.s = spl.s AND prd.city = spl.city)

WHERE prd.colour <> 'RED';
```

Joseph Vella - (c) - SQL - Introduction

26

Outer Joins
based on the Equality operator

Relational Joins

Let's look at Cities values in our tables!

Print cities values in the products and jobs tables; eliminate duplicate values and sort output by city names.

```
SELECT DISTINCT city
  FROM date.p prd
 ORDER BY city;
```

```
LONDON
PALO ALTO
PARIS
ROME
```

```
SELECT DISTINCT city
  FROM date.j job
 ORDER BY city;
```

```
ATHENS
LONDON
OSLO
PARIS
ROME
```

Print all cities values in the products and jobs tables in **one** list; eliminate duplicate values and sort output by city names.

```
SELECT DISTINCT city
  FROM date.p prd
 UNION
SELECT DISTINCT city
  FROM date.j job
 ORDER BY city;
```

```
ATHENS
LONDON
OSLO
PALO ALTO
PARIS
ROME
```

Lossy Joins

Find all collocated parts (p) and jobs (j).

```
 SELECT prd.p, job.j, prd.city
    FROM date.p prd INNER JOIN
            date.j job ON prd.city = job.city
ORDER BY prd.city;
```

- Cities on Output are:

LONDON

PARIS

ROME

- But we **lost**!

ATHENS

OSLO

PALO ALTO

We need
Outer Joins!

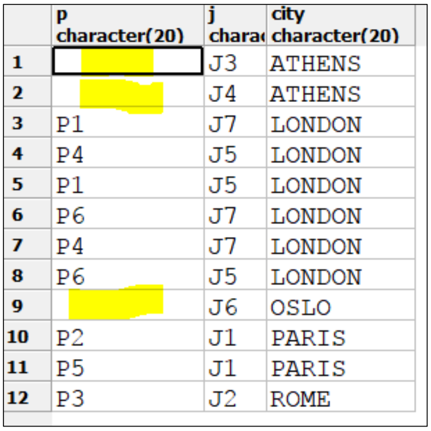
Basic Right Outer Join Example

- What if we want to see all co-located parts and jobs and ensure all job cities are mentioned; even if not co-located with any part.

```
SELECT prd.p, job.j, job.city
   FROM date.p prd RIGHT OUTER JOIN
            date.j job ON prd.city=job.city
ORDER BY job.city;
```

- RIGHT OUTER JOIN means keep all of the RIGHT table's (i.e. job tuples) attribute values.

- Note: the "null" values for columns that come from the non right table(s) where no match is satisfied.

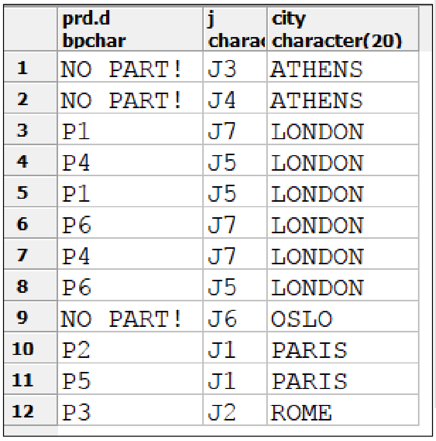


	p character(20)	j charac	city character(20)
1		J3	ATHENS
2		J4	ATHENS
3	P1	J7	LONDON
4	P4	J5	LONDON
5	P1	J5	LONDON
6	P6	J7	LONDON
7	P4	J7	LONDON
8	P6	J5	LONDON
9		J6	OSLO
10	P2	J1	PARIS
11	P5	J1	PARIS
12	P3	J2	ROME

Handling Nulls from Outer Joins

- Note: the "null" values for columns that come from the non right table(s) where no match is satisfied.
- IF you do not want to generate any nulls in our output THEN use the COALESCE() function.

```
SELECT COALESCE(prd.p, 'NO PART!') AS "prd.d",
        job.j, job.city
  FROM date.p prd RIGHT OUTER JOIN
       date.j job ON prd.city=job.city
ORDER BY job.city;
```



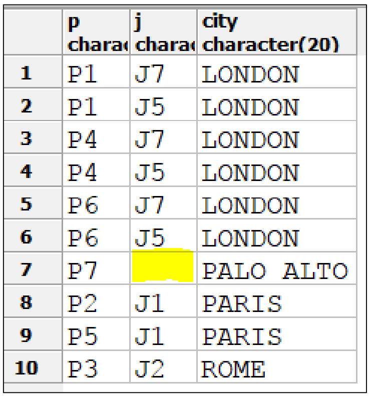
	prd.d bpchar	j charac	city character(20)
1	NO PART!	J3	ATHENS
2	NO PART!	J4	ATHENS
3	P1	J7	LONDON
4	P4	J5	LONDON
5	P1	J5	LONDON
6	P6	J7	LONDON
7	P4	J7	LONDON
8	P6	J5	LONDON
9	NO PART!	J6	OSLO
10	P2	J1	PARIS
11	P5	J1	PARIS
12	P3	J2	ROME

Basic Left Outer Join Example

- What if we want to see all co-located parts and jobs but ensure all part's cities are mentioned (even if not co-located with any job)

```
SELECT prd.p, job.j, prd.city
 FROM date.p prd LEFT OUTER JOIN
        date.j job ON prd.city=job.city
ORDER BY prd.city;
```

- LEFT OUTER JOIN means keep all of the LEFT table (i.e. product) attribute values.



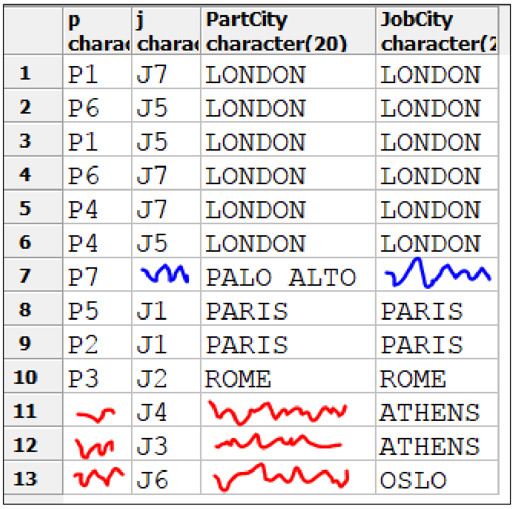
	p charac	j charac	city character(20)
1	P1	J7	LONDON
2	P1	J5	LONDON
3	P4	J7	LONDON
4	P4	J5	LONDON
5	P6	J7	LONDON
6	P6	J5	LONDON
7	P7		PALO ALTO
8	P2	J1	PARIS
9	P5	J1	PARIS
10	P3	J2	ROME

Basic Full Outer Join Example

- What if we want to see all co-located parts and jobs but ensure all parts and job cities are mentioned (even if not co-located).

```
SELECT prd.p, job.j,
        prd.city AS "PartCity",
        job.city AS "JobCity"
 FROM   date.p prd FULL OUTER JOIN
        date.j job ON prd.city = job.city
ORDER BY prd.city;
```

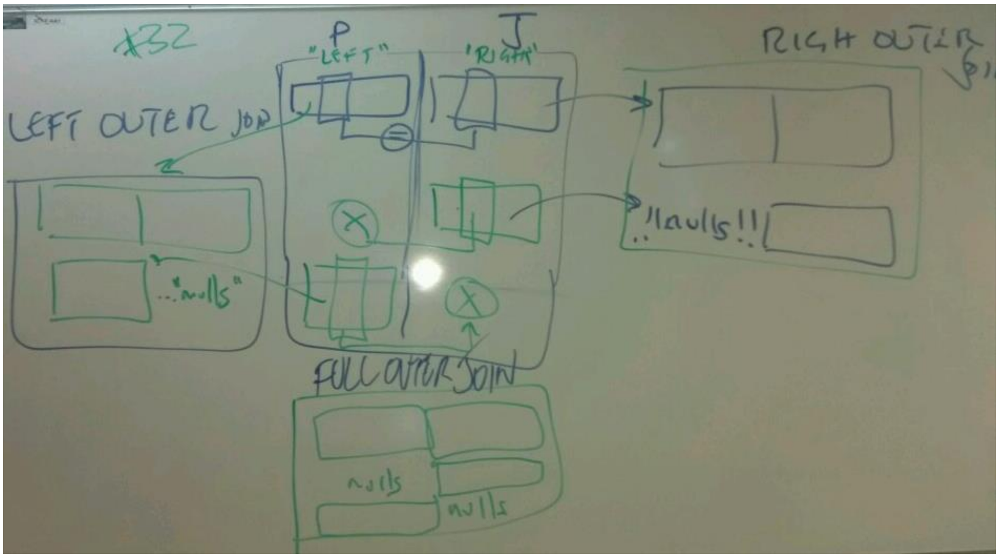
- FULL OUTER JOIN means keep all of the LEFT & RIGHT tables' (i.e. product and job) attribute values.



	p charac	j charac	PartCity character(20)	JobCity character(2
1	P1	J7	LONDON	LONDON
2	P6	J5	LONDON	LONDON
3	P1	J5	LONDON	LONDON
4	P6	J7	LONDON	LONDON
5	P4	J7	LONDON	LONDON
6	P4	J5	LONDON	LONDON
7	P7		PALO ALTO	
8	P5	J1	PARIS	PARIS
9	P2	J1	PARIS	PARIS
10	P3	J2	ROME	ROME
11		J4		ATHENS
12		J3		ATHENS
13		J6		OSLO

Joseph Vella - (c) - SQL - Introduction

Class of 2018!



Joseph Vella - (c) - SQL - Introduction

Other Joins
(e.g. Self Joins, Non Equi-Joins)

Relational Joins

Self Joins

Self joins are self joins characterised by using a table more than once in the from list of tables list.
An example query will be pairs of projects that are co-located.

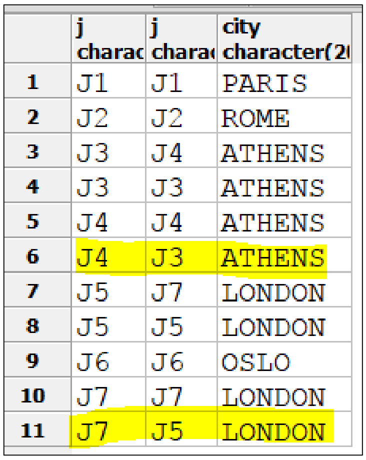
- First attempt:

```
SELECT j1.j, j2.j, j1.city
  FROM  date.j j1 INNER JOIN
            date.j j2 ON( j1.city = j2.city );
```

- Second attempt i.e. solve spurious data output; we only want the two highlighted tuples as output!

```
SELECT j1.j, j2.j, j1.city
  FROM  date.j j1 INNER JOIN
            date.j j2 ON ( j1.city = j2.city AND j1.j > j2.j );
```

- We use these type of joins to reproduce paths and sub-structures from hierarchic structures (e.g. trees and graphs).
  - More later!?



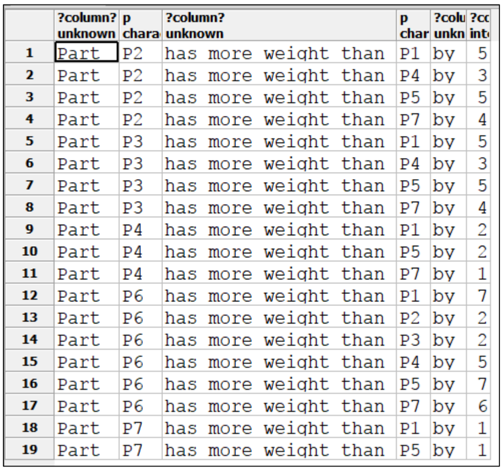
	j charac	j charac	city character(2
1	J1	J1	PARIS
2	J2	J2	ROME
3	J3	J4	ATHENS
4	J3	J3	ATHENS
5	J4	J4	ATHENS
6	J4	J3	ATHENS
7	J5	J7	LONDON
8	J5	J5	LONDON
9	J6	J6	OSLO
10	J7	J7	LONDON
11	J7	J5	LONDON

Non-equi Joins

The join comparison expression between attributes uses a non equality operator; for example>.

Consider the query to name pairs of parts with one part "having more weight" than the second.

```
SELECT 'Part',
          p1.p,
          'has more weight than',
          p2.p,
          'by',
          p1.weight - p2.weight
  FROM    date.p p1 INNER JOIN
             date.p p2 ON (p1.weight > p2.weight)
 ORDER BY p1.p, p2.p;
```



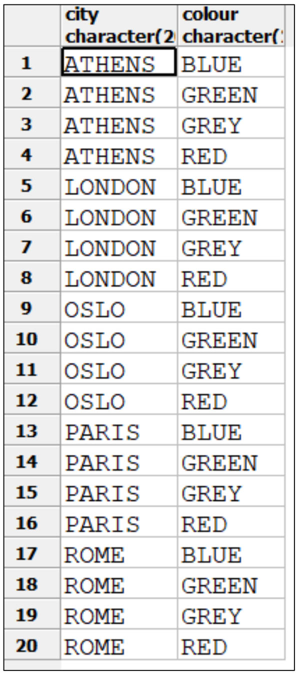
	?column? unknown	p chara	?column? unknown	p char	?colu unkn	?cc int
1	Part	P2	has more weight than	P1	by	5
2	Part	P2	has more weight than	P4	by	3
3	Part	P2	has more weight than	P5	by	5
4	Part	P2	has more weight than	P7	by	4
5	Part	P3	has more weight than	P1	by	5
6	Part	P3	has more weight than	P4	by	3
7	Part	P3	has more weight than	P5	by	5
8	Part	P3	has more weight than	P7	by	4
9	Part	P4	has more weight than	P1	by	2
10	Part	P4	has more weight than	P5	by	2
11	Part	P4	has more weight than	P7	by	1
12	Part	P6	has more weight than	P1	by	7
13	Part	P6	has more weight than	P2	by	2
14	Part	P6	has more weight than	P3	by	2
15	Part	P6	has more weight than	P4	by	5
16	Part	P6	has more weight than	P5	by	7
17	Part	P6	has more weight than	P7	by	6
18	Part	P7	has more weight than	P1	by	1
19	Part	P7	has more weight than	P5	by	1

Cross Product

- How can one build all possible pairs?
  - This is an example of Cross Product (or Cartesian Product).
- Output a list of all job's cities and product's colours pairs.

```
SELECT DISTINCT  job.city , prd.colour
  FROM date.j job, date.p prd
ORDER BY job.city, prd.colour;


SELECT DISTINCT  job.city , prd.colour
  FROM date.j job CROSS JOIN
            date.p prd
ORDER BY job.city, prd.colour;
```

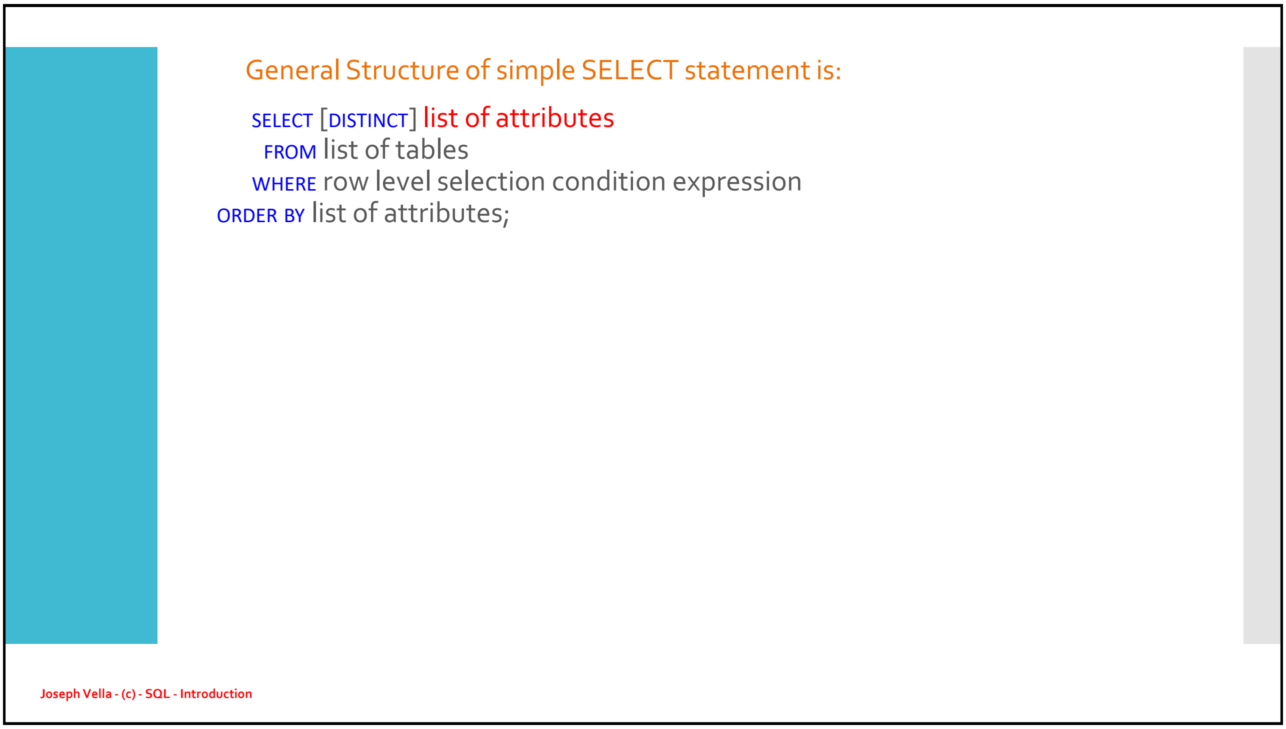
- Note: Many times we do not really mean to run a product!?



	city character(2	colour character(
1	ATHENS	BLUE
2	ATHENS	GREEN
3	ATHENS	GREY
4	ATHENS	RED
5	LONDON	BLUE
6	LONDON	GREEN
7	LONDON	GREY
8	LONDON	RED
9	OSLO	BLUE
10	OSLO	GREEN
11	OSLO	GREY
12	OSLO	RED
13	PARIS	BLUE
14	PARIS	GREEN
15	PARIS	GREY
16	PARIS	RED
17	ROME	BLUE
18	ROME	GREEN
19	ROME	GREY
20	ROME	RED

Output Expressions

Specify out-put of data and expressions



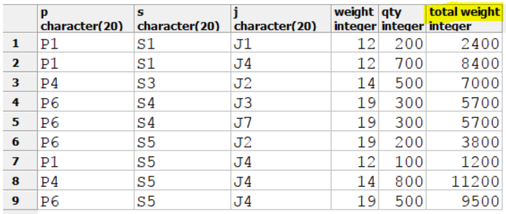
General Structure of simple SELECT statement is:

```
 SELECT [DISTINCT] list of attributes
   FROM list of tables
  WHERE row level selection condition expression
ORDER BY list of attributes;
```

Simple data expressions

- One can easily choose the attributes required in the query response by listing them in ***expressions*** after the SELECT keyword (sometimes called the ***SELECT list***).
  - Each expression can be an data attribute (from the tables listed in the FORM clause) or an expression involving the data attributes and functions available.
    - Consequently every expression in the list has a data type.
  - Also, each expression in the SELECT list can have an *alias*.
- We have a simple join and outputting a number of expressions (for data attributes note their data type corresponds to the data type defined in the table).
- The last expression is actually a product of two data attributes (work out the total weight of a product used in a works schedule). Note the alias specification.

```
SELECT wrk.p, wrk.s, wrk.j,
          prd.weight,wrk.qty,
          prd.weight*wrk.qty "total weight"
  FROM  date.p   prd INNER JOIN
          date.spj wrk ON ( prd.p = wrk.p )
WHERE prd.colour = 'RED';
```

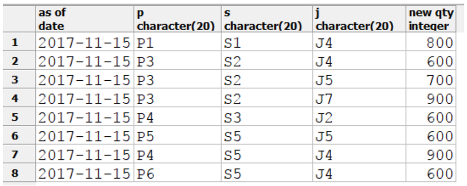


	p character(20)	s character(20)	j character(20)	weight integer	qty integer	total weight integer
1	P1	S1	J1	12	200	2400
2	P1	S1	J4	12	700	8400
3	P4	S3	J2	14	500	7000
4	P6	S4	J3	19	300	5700
5	P6	S4	J7	19	300	5700
6	P6	S5	J2	19	200	3800
7	P1	S5	J4	12	100	1200
8	P4	S5	J4	14	800	11200
9	P6	S5	J4	19	500	9500

Other Expressions

- An expression in the SELECT list need not refer to a data attribute of a table listed in the FROM clause – i.e. it could be data attribute less. In this example look at **current_date**.

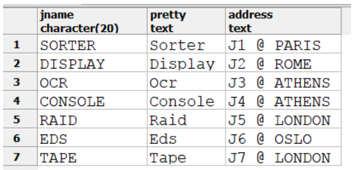
```
SELECT current_date "as of",
          p, s, j, qty + 100 "new qty“
  FROM  date.spj wrk
WHERE  wrk.qty  >= 500;
```



	as of date	p character(20)	s character(20)	j character(20)	new qty integer
1	2017-11-15	P1	S1	J4	800
2	2017-11-15	P3	S2	J4	600
3	2017-11-15	P3	S2	J5	700
4	2017-11-15	P3	S2	J7	900
5	2017-11-15	P4	S3	J2	600
6	2017-11-15	P5	S5	J5	600
7	2017-11-15	P4	S5	J4	900
8	2017-11-15	P6	S5	J4	600

- One can apply a function that takes a data attribute to compute a result per data substitution.
  - In this example **initcap**() pretty prints a text string and **||** is a concatenate operator.

```
SELECT jname, initcap(jname) "pretty",
          j || ' @ ‘ || city "address“
FROM date.j job;
```

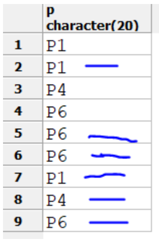


	jname character(20)	pretty text	address text
1	SORTER	Sorter	J1 @ PARIS
2	DISPLAY	Display	J2 @ ROME
3	OCR	Ocr	J3 @ ATHENS
4	CONSOLE	Console	J4 @ ATHENS
5	RAID	Raid	J5 @ LONDON
6	EDS	Eds	J6 @ OSLO
7	TAPE	Tape	J7 @ LONDON

Output Duplicate Elimination

- Consider the query: Provide part numbers for RED parts that are on a work schedule.
  - Note the duplicates in the output!?

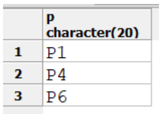
```
SELECT wrk.p
  FROM date.p   prd INNER JOIN
            date.spj wrk ON ( prd.p = wrk.p )
WHERE prd.colour = 'RED';
```



	p character(20)
1	P1
2	P1
3	P4
4	P6
5	P6
6	P6
7	P1
8	P4
9	P6

- To remove any duplicates add the DISTINCT keyword just after the SELECT keyword.

```
SELECT DISTINCT wrk.p
  FROM date.p   prd INNER JOIN
            date.spj wrk ON ( prd.p = wrk.p )
WHERE prd.colour = 'RED';
```



	p character(20)
1	P1
2	P4
3	P6

Handling Output (as in total)

Like sorting, ranking, top most, and slice.

General Structure of simple SELECT statement is:

```
 SELECT [DISTINCT] list of attributes
   FROM list of tables
  WHERE row level selection condition expression
ORDER BY list of attributes;
```

Sorting the output

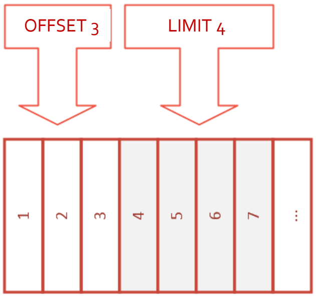
- Sorting is value based (e.g. on data attributes and expressions) – use the ORDER BY clause.
  - One can sort on a sequence of attributes/expressions.
  - Each expression can be sorted in ascending or descending order.
    - The default is ascending order.
- Examples:

```
SELECT j, s, p, qty + 100 "new qty"
  FROM date.spj wrk
 WHERE wrk.qty  >= 500
 ORDER BY  j ASC, s ASC, p DESC;


SELECT j, s, p, qty + 100 "new qty"
  FROM date.spj wrk
 WHERE wrk.qty  >= 500
 ORDER BY j, "new qty" DESC;


SELECT s, p
  FROM date.spj wrk
 WHERE wrk.qty  >= 500
 ORDER BY qty + 100 DESC;
```

Non-value based restrictions

```
SELECT select_list
    FROM table_expression
    [ ORDER BY ... ]
    [ LIMIT { number | ALL } ] [ OFFSET number ]
```

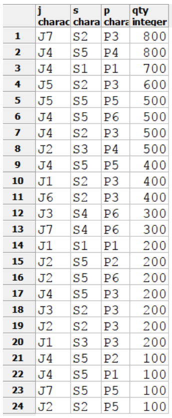
- What if we want a sub-set of the output and the subset comprehension is not value based (e.g. cannot use the WHERE clause)? One technique with modern SQL is to use ordinal directives LIMIT and OFFSET.
  - With LIMIT count one specifies the maximum number of rows to output.
  - With OFFSET count one specifies the rows to ignore from the start of the "original" output.
- In most cases LIMIT and OFFSET **are** specified with an ORDER BY expression.
  - *Otherwise* output, on the same database state, cannot be guaranteed to be the same!?



47

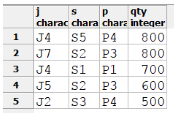
LIMIT the number of rows

SELECT j, s, p, qty
FROM date.spj wrk
WHERE wrk.qty >= 100
ORDER BY qty DESC;



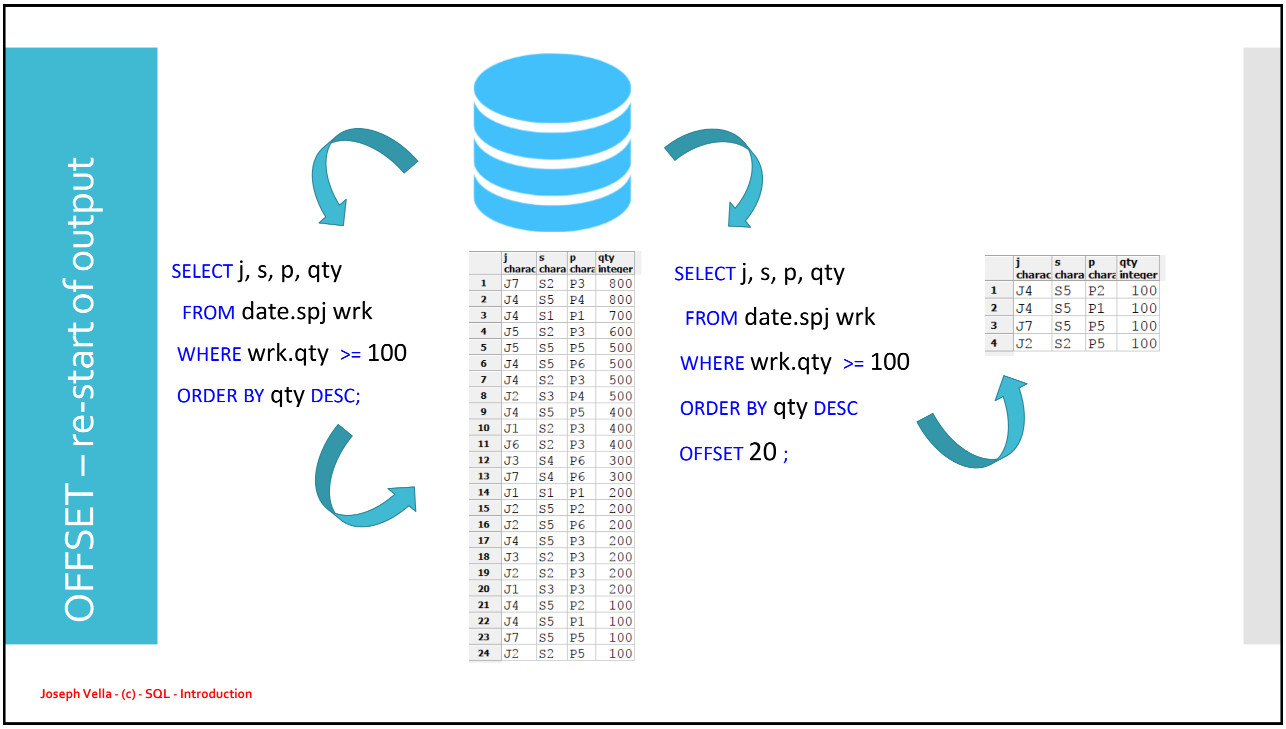
	j charac	s chara	p chara	qty integer
1	J7	S2	P3	800
2	J4	S5	P4	800
3	J4	S1	P1	700
4	J5	S2	P3	600
5	J5	S5	P5	500
6	J4	S5	P6	500
7	J4	S2	P3	500
8	J2	S3	P4	500
9	J4	S5	P5	400
10	J1	S2	P3	400
11	J6	S2	P3	400
12	J3	S4	P6	300
13	J7	S4	P6	300
14	J1	S1	P1	200
15	J2	S5	P2	200
16	J2	S5	P6	200
17	J4	S5	P3	200
18	J3	S2	P3	200
19	J2	S2	P3	200
20	J1	S3	P3	200
21	J4	S5	P2	100
22	J4	S5	P1	100
23	J7	S5	P5	100
24	J2	S2	P5	100

SELECT j, s, p, qty
FROM date.spj wrk
WHERE wrk.qty >= 100
ORDER BY qty DESC
LIMIT 5 ;



	j charac	s chara	p chara	qty integer
1	J4	S5	P4	800
2	J7	S2	P3	800
3	J4	S1	P1	700
4	J5	S2	P3	600
5	J2	S3	P4	500

48



OFFSET – re-start of output

```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 100
 ORDER BY qty DESC;
```

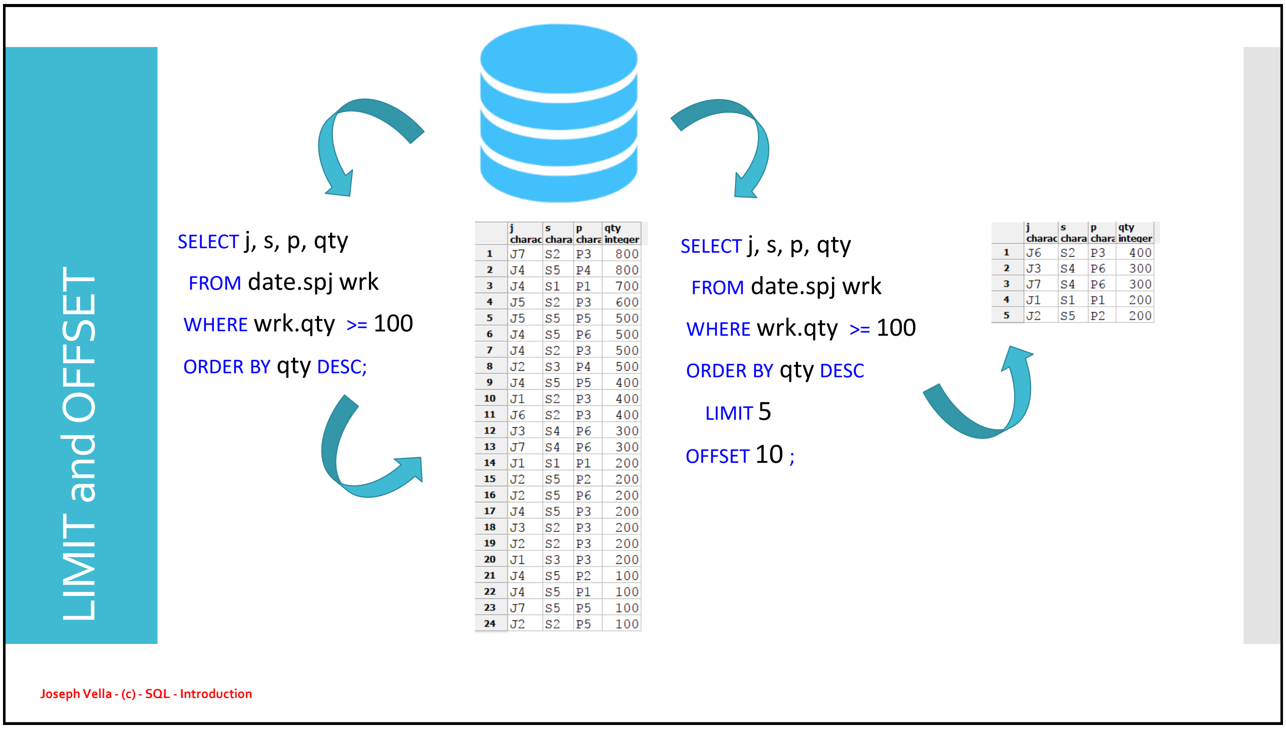
	j charac	s chara	p chara	qty integer
1	J7	S2	P3	800
2	J4	S5	P4	800
3	J4	S1	P1	700
4	J5	S2	P3	600
5	J5	S5	P5	500
6	J4	S5	P6	500
7	J4	S2	P3	500
8	J2	S3	P4	500
9	J4	S5	P5	400
10	J1	S2	P3	400
11	J6	S2	P3	400
12	J3	S4	P6	300
13	J7	S4	P6	300
14	J1	S1	P1	200
15	J2	S5	P2	200
16	J2	S5	P6	200
17	J4	S5	P3	200
18	J3	S2	P3	200
19	J2	S2	P3	200
20	J1	S3	P3	200
21	J4	S5	P2	100
22	J4	S5	P1	100
23	J7	S5	P5	100
24	J2	S2	P5	100

```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 100
 ORDER BY qty DESC
 OFFSET 20 ;
```

	j charac	s chara	p chara	qty integer
1	J4	S5	P2	100
2	J4	S5	P1	100
3	J7	S5	P5	100
4	J2	S2	P5	100

Joseph Vella - (c) - SQL - Introduction

49



LIMIT and OFFSET

```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 100
 ORDER BY qty DESC;
```

	j charac	s chara	p chara	qty integer
1	J7	S2	P3	800
2	J4	S5	P4	800
3	J4	S1	P1	700
4	J5	S2	P3	600
5	J5	S5	P5	500
6	J4	S5	P6	500
7	J4	S2	P3	500
8	J2	S3	P4	500
9	J4	S5	P5	400
10	J1	S2	P3	400
11	J6	S2	P3	400
12	J3	S4	P6	300
13	J7	S4	P6	300
14	J1	S1	P1	200
15	J2	S5	P2	200
16	J2	S5	P6	200
17	J4	S5	P3	200
18	J3	S2	P3	200
19	J2	S2	P3	200
20	J1	S3	P3	200
21	J4	S5	P2	100
22	J4	S5	P1	100
23	J7	S5	P5	100
24	J2	S2	P5	100

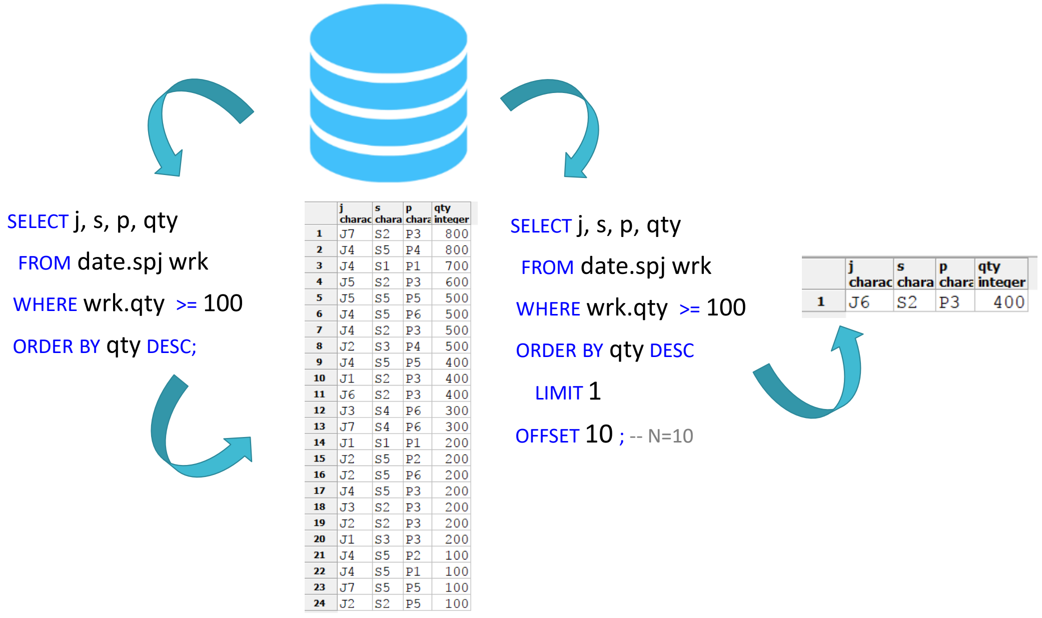
```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 100
 ORDER BY qty DESC
   LIMIT 5
 OFFSET 10 ;
```

	j charac	s chara	p chara	qty integer
1	J6	S2	P3	400
2	J3	S4	P6	300
3	J7	S4	P6	300
4	J1	S1	P1	200
5	J2	S5	P2	200

Joseph Vella - (c) - SQL - Introduction

50

N th Row (e.g. get the 10^{th})



```
SELECT j, s, p, qty
 FROM date.spj wrk
WHERE wrk.qty  >= 100
ORDER BY qty DESC;
```

	j charac	s chara	p chara	qty integer
1	J7	S2	P3	800
2	J4	S5	P4	800
3	J4	S1	P1	700
4	J5	S2	P3	600
5	J5	S5	P5	500
6	J4	S5	P6	500
7	J4	S2	P3	500
8	J2	S3	P4	500
9	J4	S5	P5	400
10	J1	S2	P3	400
11	J6	S2	P3	400
12	J3	S4	P6	300
13	J7	S4	P6	300
14	J1	S1	P1	200
15	J2	S5	P2	200
16	J2	S5	P6	200
17	J4	S5	P3	200
18	J3	S2	P3	200
19	J2	S2	P3	200
20	J1	S3	P3	200
21	J4	S5	P2	100
22	J4	S5	P1	100
23	J7	S5	P5	100
24	J2	S2	P5	100

```
SELECT j, s, p, qty
 FROM date.spj wrk
WHERE wrk.qty  >= 100
ORDER BY qty DESC
  LIMIT 1
OFFSET 10 ; -- N=10
```

	j charac	s chara	p chara	qty integer
1	J6	S2	P3	400

Functions for Output Expressions

Actually these functions, with restrictions, can be used in for example WHERE and ORDER BY expressions.

Row

vs

Aggregate

vs

Hybrid (Group By queries)

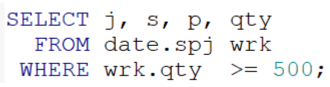
vs

Data frames (e.g. windows)*

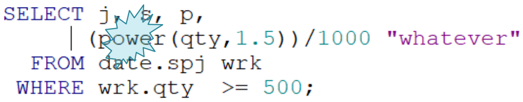
And some other stuff like adding conditional expression to SELECT list and generating data on the fly.

*) topic in forthcoming units.

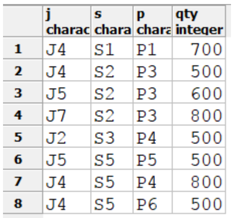
Row function example



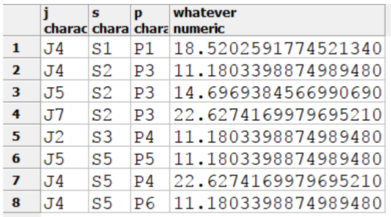
```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```



```
SELECT j, s, p,
     | (power(qty,1.5))/1000 "whatever"
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```



	j charac	s chara	p chara	qty integer
1	J4	S1	P1	700
2	J4	S2	P3	500
3	J5	S2	P3	600
4	J7	S2	P3	800
5	J2	S3	P4	500
6	J5	S5	P5	500
7	J4	S5	P4	800
8	J4	S5	P6	500

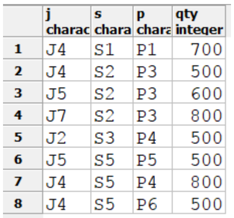


	j charac	s chara	p chara	whatever numeric
1	J4	S1	P1	18.5202591774521340
2	J4	S2	P3	11.1803398874989480
3	J5	S2	P3	14.6969384566990690
4	J7	S2	P3	22.6274169979695210
5	J2	S3	P4	11.1803398874989480
6	J5	S5	P5	11.1803398874989480
7	J4	S5	P4	22.6274169979695210
8	J4	S5	P6	11.1803398874989480

- Function power() is a row function because it output’s one value for each application per row.
  - If the output has 8 rows then applying the same filter and a row function on the output one remains with 8.

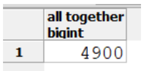
Aggregate function example

```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```



	j charac	s chara	p chara	qty integer
1	J4	S1	P1	700
2	J4	S2	P3	500
3	J5	S2	P3	600
4	J7	S2	P3	800
5	J2	S3	P4	500
6	J5	S5	P5	500
7	J4	S5	P4	800
8	J4	S5	P6	500

```
SELECT sum(qty) "all together"
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```



	all together bigint
1	4900

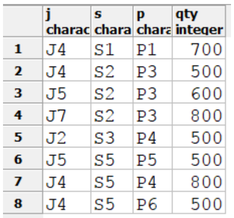
- Function sum() is an aggregate function because it output's one value for all rows on the output.
  - If the output has 8 rows then applying the same filter and an aggregate function on the output reduces to one row.

Joseph Vella - (c) - SQL - Introduction

55

Aggregate functions example

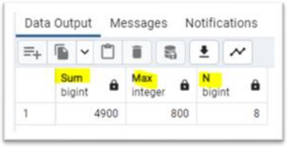
```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```



	j charac	s chara	p chara	qty integer
1	J4	S1	P1	700
2	J4	S2	P3	500
3	J5	S2	P3	600
4	J7	S2	P3	800
5	J2	S3	P4	500
6	J5	S5	P5	500
7	J4	S5	P4	800
8	J4	S5	P6	500

One can have **multiple aggregate functions** in one output list

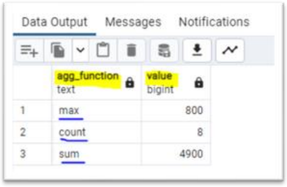
```
SELECT sum(qty)  "Sum",
       max(qty)  "Max",
       count(qty)  "N"

 FROM date.spj wrk

WHERE wrk.qty >= 500;
```



	Sum bigint	Max integer	N bigint
1	4900	800	8

But, **what is happening here?**

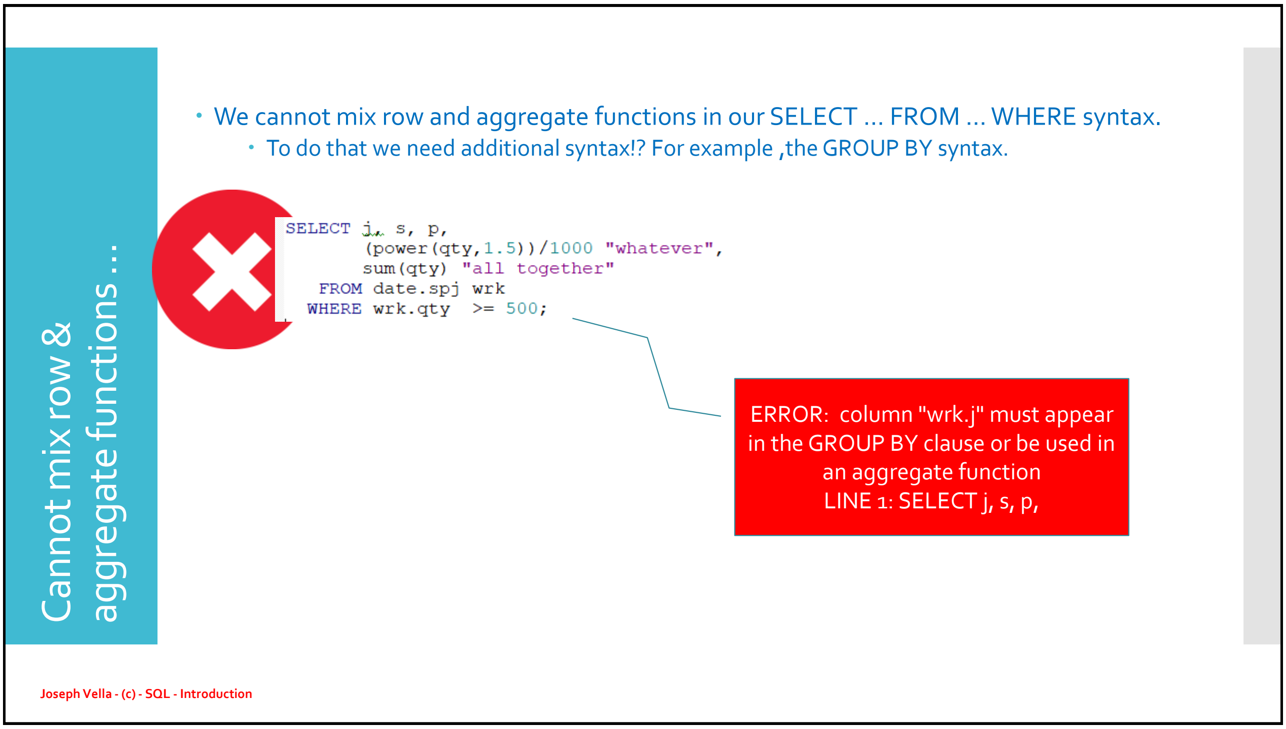
```
SELECT 'sum' "agg_function", sum(qty) "value"
  FROM date.spj wrk
 WHERE wrk.qty >= 500
 UNION ALL
SELECT 'count', count(qty)
  FROM date.spj wrk
 WHERE wrk.qty >= 500
 UNION ALL
SELECT 'max', max(qty)
  FROM date.spj wrk
 WHERE wrk.qty >= 500;
```



	agg_function text	value bigint
1	max	800
2	count	8
3	sum	4900

Joseph Vella - (c) - SQL - Introduction

56



Cannot mix row & aggregate functions …

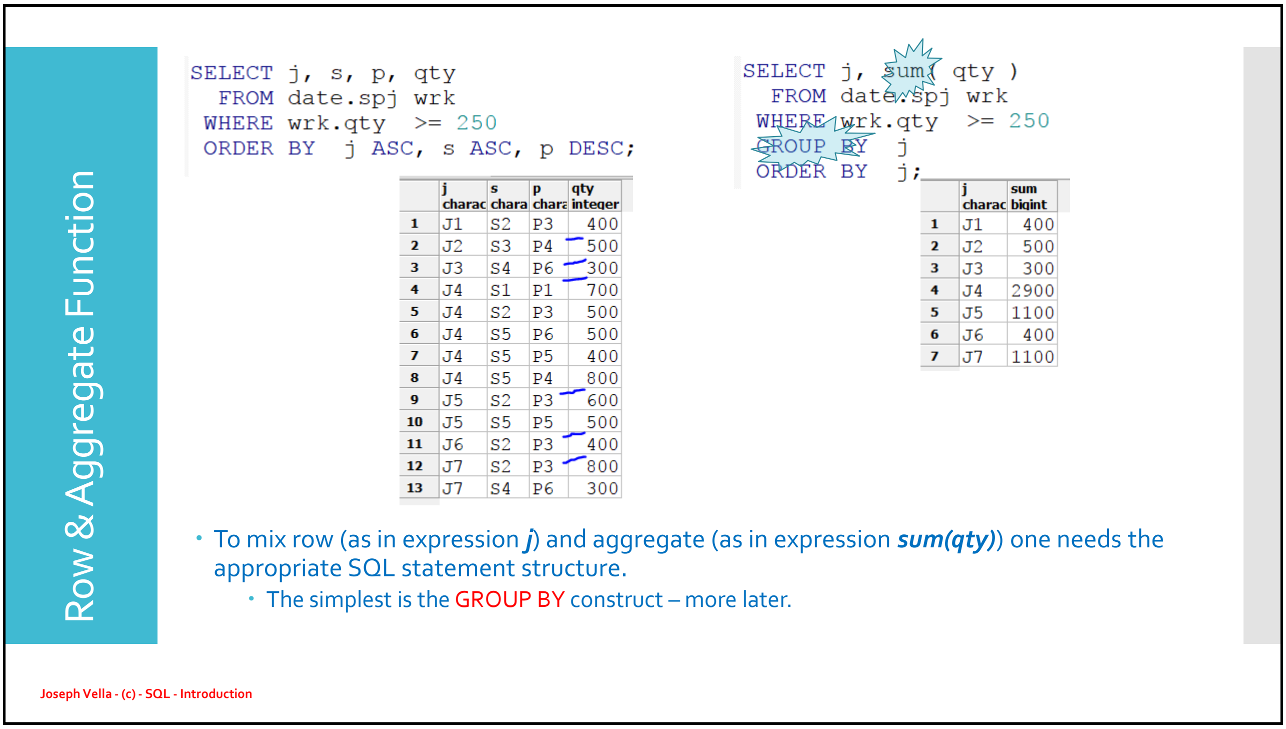
- We cannot mix row and aggregate functions in our SELECT … FROM … WHERE syntax.
  - To do that we need additional syntax!? For example ,the GROUP BY syntax.

```
SELECT j, s, p,
       (power(qty,1.5))/1000 "whatever",
       sum(qty) "all together"
  FROM date.spj wrk
 WHERE wrk.qty  >= 500;
```

ERROR: column "wrk.j" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT j, s, p,

Joseph Vella - (c) - SQL - Introduction

57



Row & Aggregate Function

```
SELECT j, s, p, qty
  FROM date.spj wrk
 WHERE wrk.qty  >= 250
 ORDER BY  j ASC, s ASC, p DESC;
```

	j charac	s chara	p chara	qty integer
1	J1	S2	P3	400
2	J2	S3	P4	500
3	J3	S4	P6	300
4	J4	S1	P1	700
5	J4	S2	P3	500
6	J4	S5	P6	500
7	J4	S5	P5	400
8	J4	S5	P4	800
9	J5	S2	P3	600
10	J5	S5	P5	500
11	J6	S2	P3	400
12	J7	S2	P3	800
13	J7	S4	P6	300

```
SELECT j, sum( qty )
  FROM date.spj wrk
 WHERE wrk.qty  >= 250
 GROUP BY  j
 ORDER BY  j;
```

	j charac	sum bigint
1	J1	400
2	J2	500
3	J3	300
4	J4	2900
5	J5	1100
6	J6	400
7	J7	1100

- To mix row (as in expression ***j***) and aggregate (as in expression ***sum(qty)***) one needs the appropriate SQL statement structure.
  - The simplest is the GROUP BY construct – more later.

Joseph Vella - (c) - SQL - Introduction

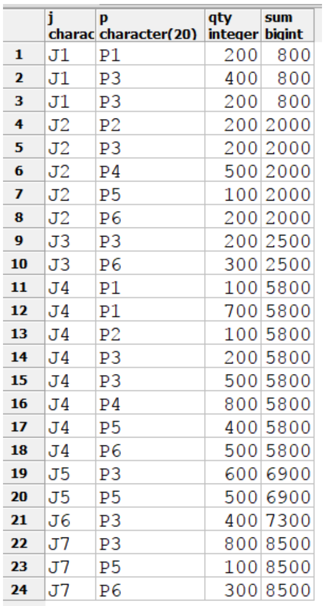
58

Partitioning & Running Totals

- In normal SELECT … FROM … WHERE statements we have made it a point to say that each row, from the data sources, is evaluated independently of any other row.
  - Also the output for a row that passed the filter can only access the data in that row!
- To circumvent this we need a new syntax! For example:

```
SELECT j, p, qty, sum(qty)
  OVER (ORDER BY j)
  FROM date.spj wrk
 ORDER BY j, p;
```

- The idea here is to compute a running total. The partition, or window, in this case is each distinct Job reference. For the first case we have 'J1'. All the parts allocated to 'J1', i.e. 'P1' twice and 'P3', total 800 (200+ 400 + 200).
  - The output include The Project, parts used and their respective quantity.
- The simplest is the **Windows Functions** construct – more later.



	j charac	p character(20)	qty integer	sum bigint
1	J1	P1	200	800
2	J1	P3	400	800
3	J1	P3	200	800
4	J2	P2	200	2000
5	J2	P3	200	2000
6	J2	P4	500	2000
7	J2	P5	100	2000
8	J2	P6	200	2000
9	J3	P3	200	2500
10	J3	P6	300	2500
11	J4	P1	100	5800
12	J4	P1	700	5800
13	J4	P2	100	5800
14	J4	P3	200	5800
15	J4	P3	500	5800
16	J4	P4	800	5800
17	J4	P5	400	5800
18	J4	P6	500	5800
19	J5	P3	600	6900
20	J5	P5	500	6900
21	J6	P3	400	7300
22	J7	P3	800	8500
23	J7	P5	100	8500
24	J7	P6	300	8500

Joseph Vella - (c) - SQL - Introduction

Examples: numeric functions

- Note we are using the most basic syntax for exposition.

```
-- absolute value

SELECT abs(-5);
-- 5::int
```

```
-- nearest integer greater than
-- or equal to argument

SELECT ceil(-1.713);
-- -1::numeric

-- nearest integer less than
-- or equal to argument

SELECT floor(-1.713);
-- -2::numeric

-- truncates toward zero or decimal places

SELECT trunc(1.7149), trunc(1.7149,3);
-- 1::numeric, 1.714::numeric

-- round to nearest integer

SELECT round(1.7149), round(1.7149,3);
-- 2::numeric, 1.715::numeric
```

```
-- exponential & natural log

SELECT exp(2);
-- 7.389::dp

SELECT ln(7.38905609893065);
-- 2.0::numeric
```

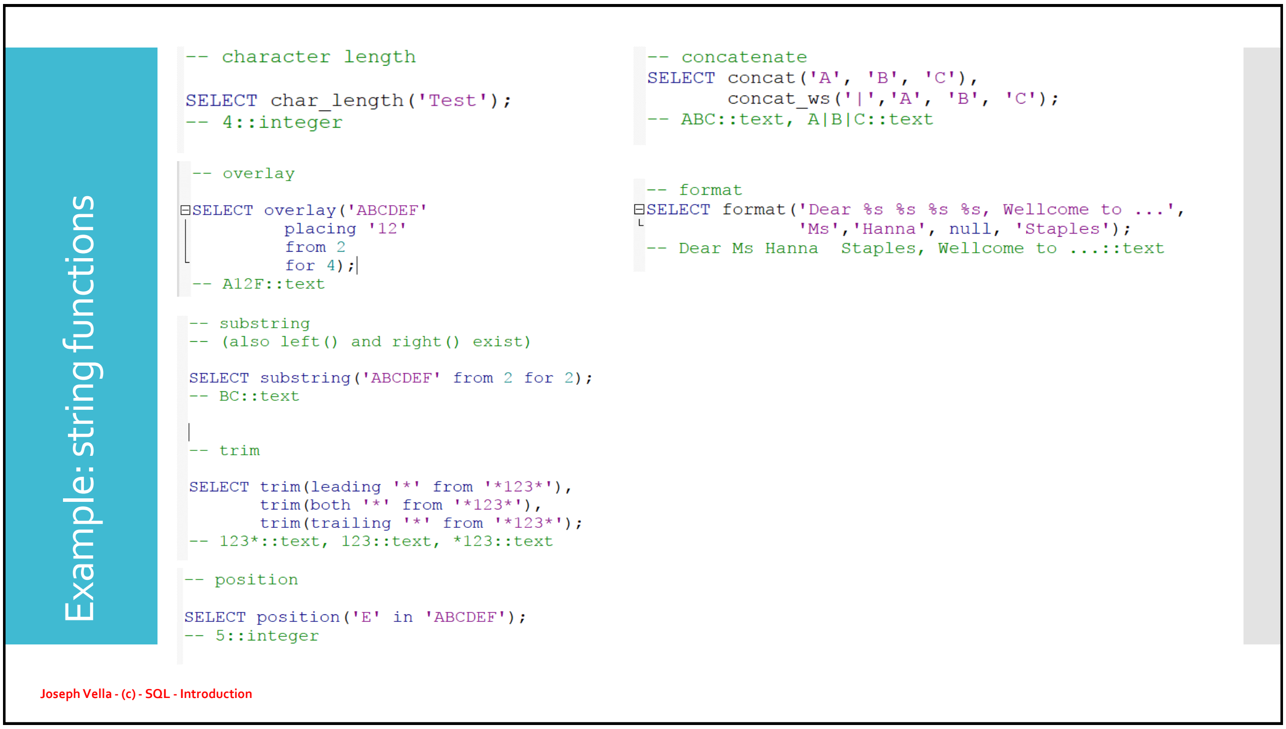
```
-- raise to the power

SELECT power(2,3);
-- 8::dp
```

```
-- bucketting (exp, min, max, numb buckets)
SELECT width_bucket(6.5,0,10,10);
-- 7::integer
```

```
-- randon number (and re-seed if required)

SELECT random();
-- 0 <= x <= 1
```

```
-- trigonometric and reverse functions

SELECT sind(30);

SELECT asind(.5);
```

Joseph Vella - (c) - SQL - Introduction



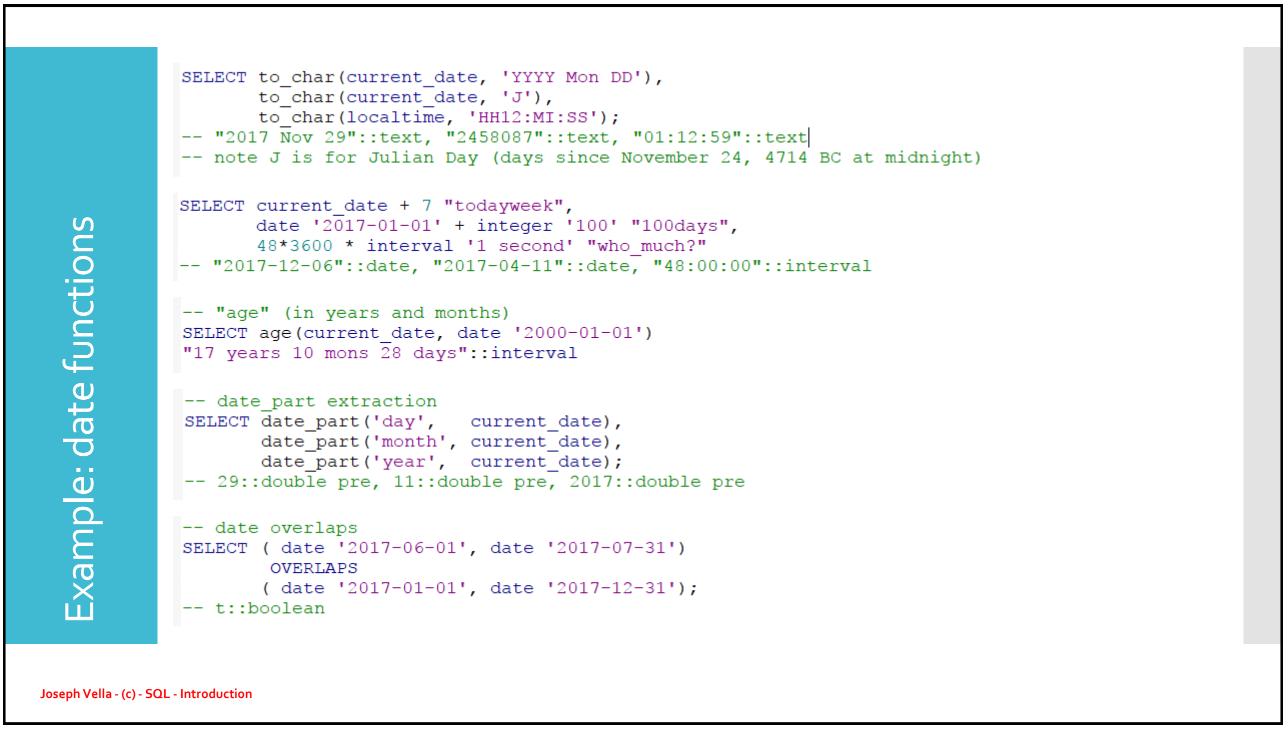
Example: string functions

```
-- character length

SELECT char_length('Test');
-- 4::integer

-- overlay

SELECT overlay('ABCDEF'
          placing '12'
          from 2
          for 4);
-- A12F::text

-- substring
-- (also left() and right() exist)

SELECT substring('ABCDEF' from 2 for 2);
-- BC::text


-- trim

SELECT trim(leading '*' from '*123*'),
       trim(both '*' from '*123*'),
       trim(trailing '*' from '*123*');
-- 123*::text, 123::text, *123::text

-- position

SELECT position('E' in 'ABCDEF');
-- 5::integer
```

```
-- concatenate
SELECT concat('A', 'B', 'C'),
       concat_ws('|','A', 'B', 'C');
-- ABC::text, A|B|C::text


-- format
SELECT format('Dear %s %s %s %s, Wellcome to ...',
              'Ms','Hanna', null, 'Staples');
-- Dear Ms Hanna  Staples, Wellcome to ...::text
```

Joseph Vella - (c) - SQL - Introduction

61



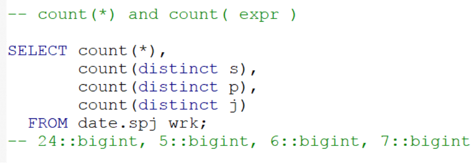
Example: date functions

```
SELECT to_char(current_date, 'YYYY Mon DD'),
       to_char(current_date, 'J'),
       to_char(localtime, 'HH12:MI:SS');
-- "2017 Nov 29"::text, "2458087"::text, "01:12:59"::text
-- note J is for Julian Day (days since November 24, 4714 BC at midnight)

SELECT current_date + 7 "todayweek",
       date '2017-01-01' + integer '100' "100days",
       48*3600 * interval '1 second' "who_much?"
-- "2017-12-06"::date, "2017-04-11"::date, "48:00:00"::interval

-- "age" (in years and months)
SELECT age(current_date, date '2000-01-01')
"17 years 10 mons 28 days"::interval

-- date_part extraction
SELECT date_part('day',   current_date),
       date_part('month', current_date),
       date_part('year',  current_date);
-- 29::double pre, 11::double pre, 2017::double pre

-- date overlaps
SELECT ( date '2017-06-01', date '2017-07-31')
        OVERLAPS
       ( date '2017-01-01', date '2017-12-31');
-- t::boolean
```

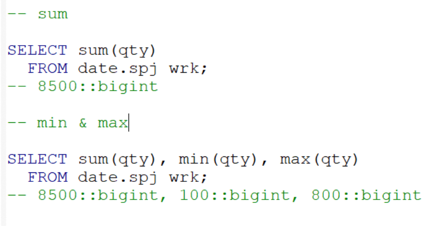
Joseph Vella - (c) - SQL - Introduction

62

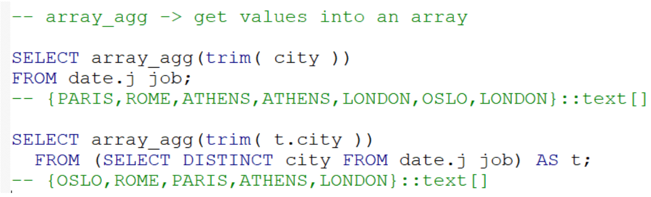
Example Aggregate Funct.s



```
-- count(*) and count( expr )

SELECT count(*),
       count(distinct s),
       count(distinct p),
       count(distinct j)
  FROM date.spj wrk;
-- 24::bigint, 5::bigint, 6::bigint, 7::bigint
```



```
-- sum

SELECT sum(qty)
  FROM date.spj wrk;
-- 8500::bigint

-- min & max

SELECT sum(qty), min(qty), max(qty)
  FROM date.spj wrk;
-- 8500::bigint, 100::bigint, 800::bigint
```

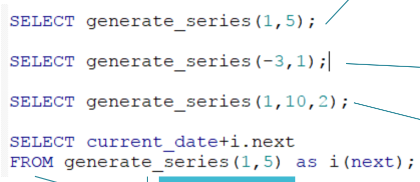


```
-- array_agg -> get values into an array

SELECT array_agg(trim( city ))
FROM date.j job;
-- {PARIS,ROME,ATHENS,ATHENS,LONDON,OSLO,LONDON}::text[]

SELECT array_agg(trim( t.city ))
  FROM (SELECT DISTINCT city FROM date.j job) AS t;
-- {OSLO,ROME,PARIS,ATHENS,LONDON}::text[]
```

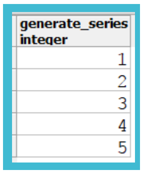
Joseph Vella - (c) - SQL - Introduction

Quick Aside: A set generating function!

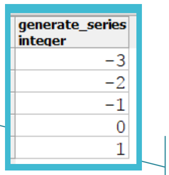
- The function generating_series() returns a number of tuples on the fly. The function takes two to three arguments: first term, last term, and an optional jump.
    - Each argument is a numeric and usually is an integer.
- The generate_series() can be used in SELECT list expressions or as a data source in the FROM list of expressions.



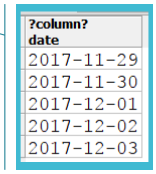
```
SELECT generate_series(1,5);

SELECT generate_series(-3,1);

SELECT generate_series(1,10,2);

SELECT current_date+i.next
FROM generate_series(1,5) as i(next);
```



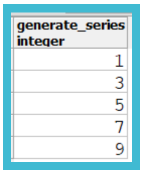
generate_series integer
1
2
3
4
5



generate_series integer
-3
-2
-1
0
1



?column? date
2017-11-29
2017-11-30
2017-12-01
2017-12-02
2017-12-03

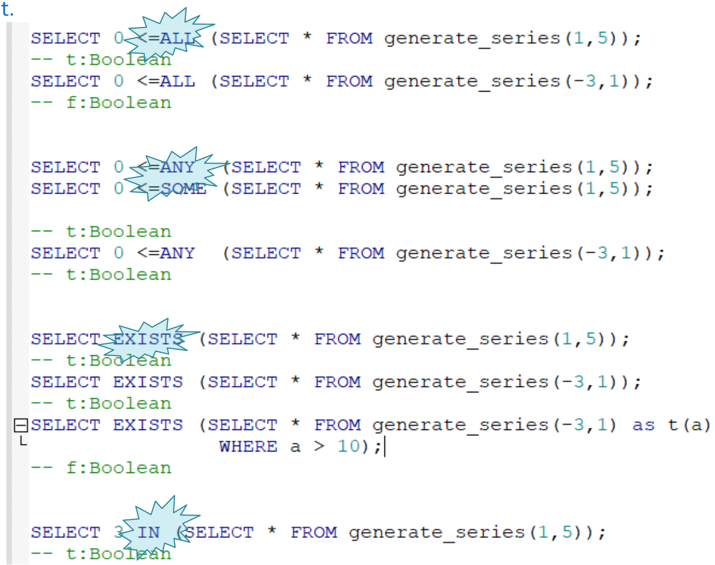


generate_series integer
1
3
5
7
9

Joseph Vella - (c) - SQL - Introduction

More example Aggregate Functions

- The following Boolean functions enable comparisons of a value to a set.
  - ALL – all tuples have the stated property. (I.e. tuples [1], [2], [3], [4] & [5] are all greater than constant 0 (zero)).
  - ANY (or SOME) – at least one tuple has the stated property.
  - EXISTS is a test for non-empty set.
  - IN confirms presence (already covered earlier).


```
SELECT 0 <=ALL (SELECT * FROM generate_series(1,5));
-- t:Boolean
SELECT 0 <=ALL (SELECT * FROM generate_series(-3,1));
-- f:Boolean


SELECT 0 <=ANY (SELECT * FROM generate_series(1,5));
SELECT 0 <=SOME (SELECT * FROM generate_series(1,5));

-- t:Boolean
SELECT 0 <=ANY  (SELECT * FROM generate_series(-3,1));
-- t:Boolean


SELECT EXISTS (SELECT * FROM generate_series(1,5));
-- t:Boolean
SELECT EXISTS (SELECT * FROM generate_series(-3,1));
-- t:Boolean
SELECT EXISTS (SELECT * FROM generate_series(-3,1) as t(a)
                WHERE a > 10);
-- f:Boolean


SELECT 3 IN (SELECT * FROM generate_series(1,5));
-- t:Boolean
```

Joseph Vella - (c) - SQL - Introduction

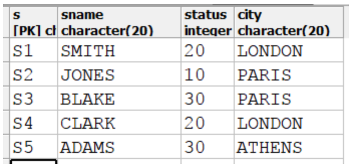
65

Example: output conditionals

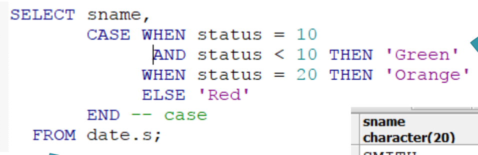
- The CASE expression can alter the output of a SELECT expression.
  - The condition, expressed in the WHEN must return a Boolean (i.e. True or False). If the expression returns a 'True' then the expression listed after the relative THEN keyword is executed.
  - Only one (the first) WHEN condition is executed.
  - If no WHEN condition is true and if an ELSE expression is present then it is executed!
  - **The data type of all result expression must match!**

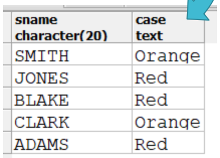
```
CASE WHEN condition THEN result
     [WHEN ...]
     [ELSE result]
END
```

```
CASE expression
    WHEN value THEN result
    [WHEN ...]
    [ELSE result]
END
```



s [PK] cl	sname character(20)	status integer	city character(20)
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS


```
SELECT sname,
       CASE WHEN status = 10
             AND status < 10 THEN 'Green'
            WHEN status = 20 THEN 'Orange'
            ELSE 'Red'
       END -- case
  FROM date.s;
```



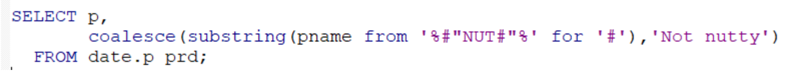
sname character(20)	case text
SMITH	Orange
JONES	Red
BLAKE	Red
CLARK	Orange
ADAMS	Red

Joseph Vella - (c) - SQL - Introduction

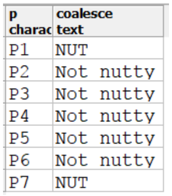
66

Example: output conditionals

- Another two output expressions are: COALESCE(), seen before, and related NULLIF().



```
SELECT p,
       coalesce(substring(pname from '%#"NUT#"%' for '#'),'Not nutty')
  FROM date.p prd;
```

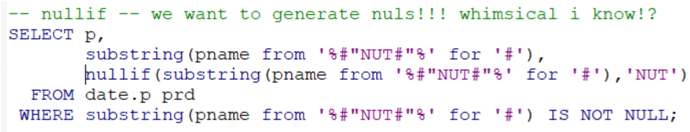


p charac	coalesce text
P1	NUT
P2	Not nutty
P3	Not nutty
P4	Not nutty
P5	Not nutty
P6	Not nutty
P7	NUT

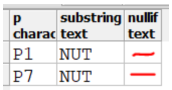
-- searching with regular expressions:

Meta character sequence #” is the return string indicator.

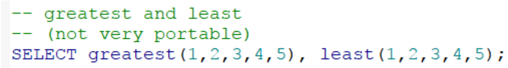
Therefore the above returns ‘NUT’ if ‘NUT’ is present in the string.



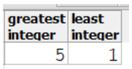
```
-- nullif -- we want to generate nuls!!! whimsical i know!?
SELECT p,
       substring(pname from '%#"NUT#"%' for '#'),
       nullif(substring(pname from '%#"NUT#"%' for '#'),'NUT')
  FROM date.p prd
 WHERE substring(pname from '%#"NUT#"%' for '#') IS NOT NULL;
```



p charac	substring text	nullif text
P1	NUT	—
P7	NUT	—

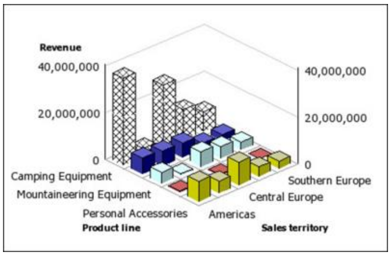


```
-- greatest and least
-- (not very portable)
SELECT greatest(1,2,3,4,5), least(1,2,3,4,5);
```



greatest integer	least integer
5	1

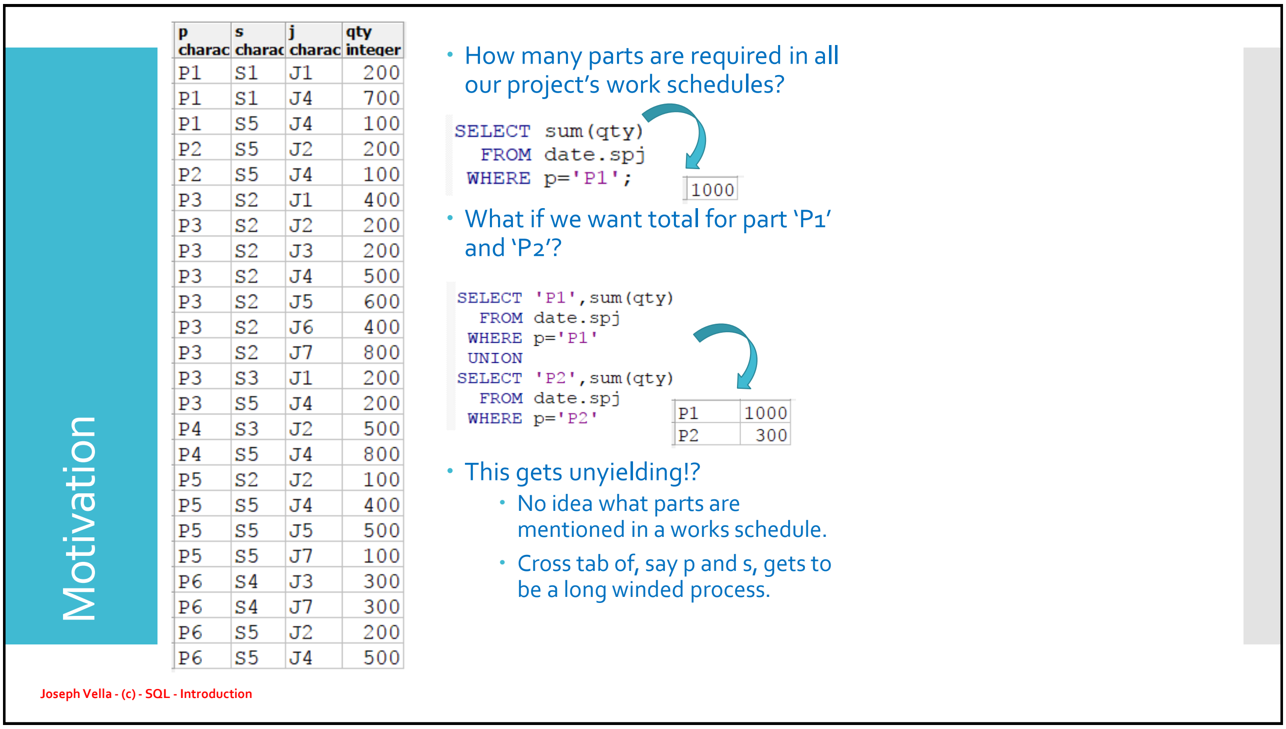
67



Group By Queries

Great for cross tabulations - basic ones for now

68



Motivation

p	s	j	qty
charac	charac	charac	integer
P1	S1	J1	200
P1	S1	J4	700
P1	S5	J4	100
P2	S5	J2	200
P2	S5	J4	100
P3	S2	J1	400
P3	S2	J2	200
P3	S2	J3	200
P3	S2	J4	500
P3	S2	J5	600
P3	S2	J6	400
P3	S2	J7	800
P3	S3	J1	200
P3	S5	J4	200
P4	S3	J2	500
P4	S5	J4	800
P5	S2	J2	100
P5	S5	J4	400
P5	S5	J5	500
P5	S5	J7	100
P6	S4	J3	300
P6	S4	J7	300
P6	S5	J2	200
P6	S5	J4	500

- How many parts are required in all our project’s work schedules?

```
SELECT  sum(qty)
  FROM  date.spj
 WHERE  p='P1';
```

```
1000
```

- What if we want total for part ‘P1’ and ‘P2’?

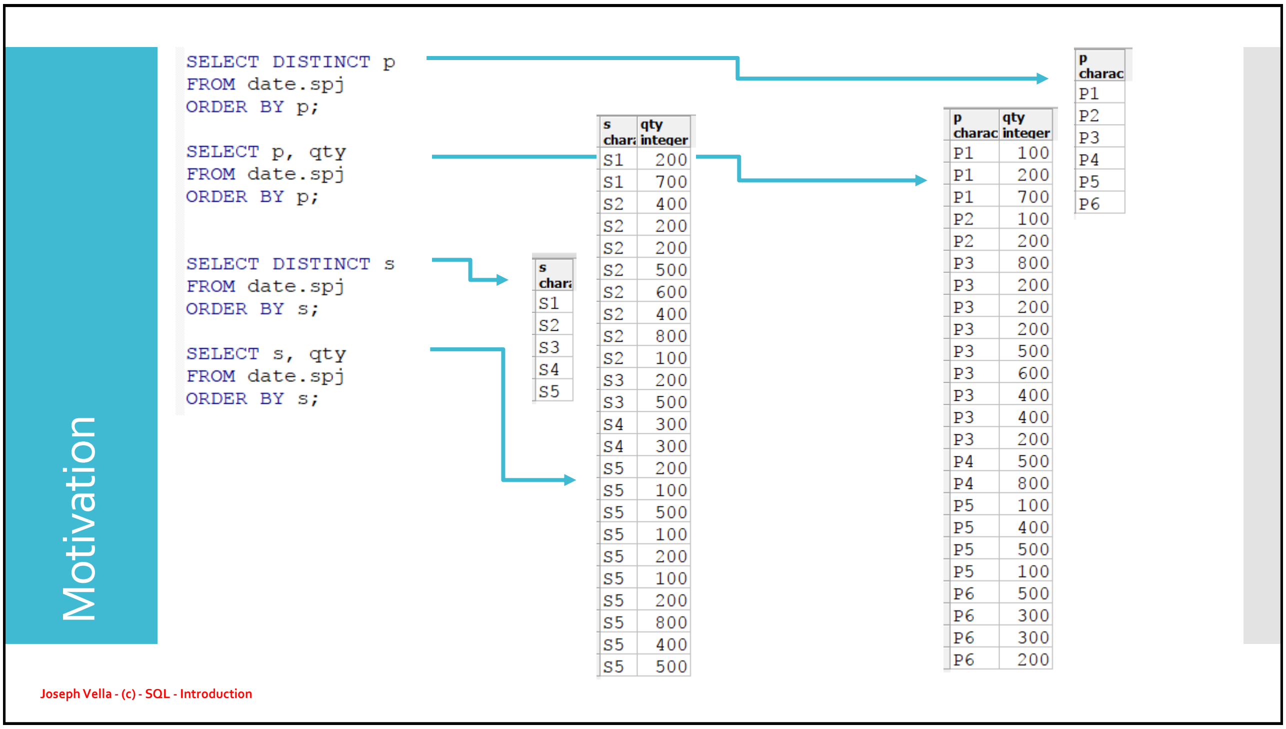
```
SELECT 'P1',sum(qty)
  FROM date.spj
 WHERE p='P1'
 UNION
SELECT 'P2',sum(qty)
  FROM date.spj
 WHERE p='P2'
```

P1	1000
P2	300

- This gets unyielding!?
  - No idea what parts are mentioned in a works schedule.
  - Cross tab of, say p and s, gets to be a long winded process.

Joseph Vella - (c) - SQL - Introduction

69



Motivation

```
SELECT DISTINCT p
FROM date.spj
ORDER BY p;
```

p
charac
P1
P2
P3
P4
P5
P6

```
SELECT p, qty
FROM date.spj
ORDER BY p;
```

p	qty
charac	integer
P1	100
P1	200
P1	700
P2	100
P2	200
P3	800
P3	200
P3	200
P3	200
P3	500
P3	600
P3	400
P3	400
P3	200
P4	500
P4	800
P5	100
P5	400
P5	500
P5	100
P6	500
P6	300
P6	300
P6	200

```
SELECT DISTINCT s
FROM date.spj
ORDER BY s;
```

s
chara
S1
S2
S3
S4
S5

```
SELECT s, qty
FROM date.spj
ORDER BY s;
```

s	qty
chara	integer
S1	200
S1	700
S2	400
S2	200
S2	200
S2	500
S2	600
S2	400
S2	800
S2	100
S3	200
S3	500
S4	300
S4	300
S5	200
S5	100
S5	500
S5	100
S5	200
S5	100
S5	200
S5	800
S5	400
S5	500

Joseph Vella - (c) - SQL - Introduction

70

Issue!? (already mentioned)

- Remember we cannot mix row and aggregate function in a SELECT list of expressions in our in our simple syntax of SELECT FROM WHERE:

```
SELECT wrk.p,
       sum(wrk.qty) "Parts"
  FROM date.spj wrk;
```

ERROR: column "wrk.p" must appear in the GROUP BY clause or be used in an aggregate function

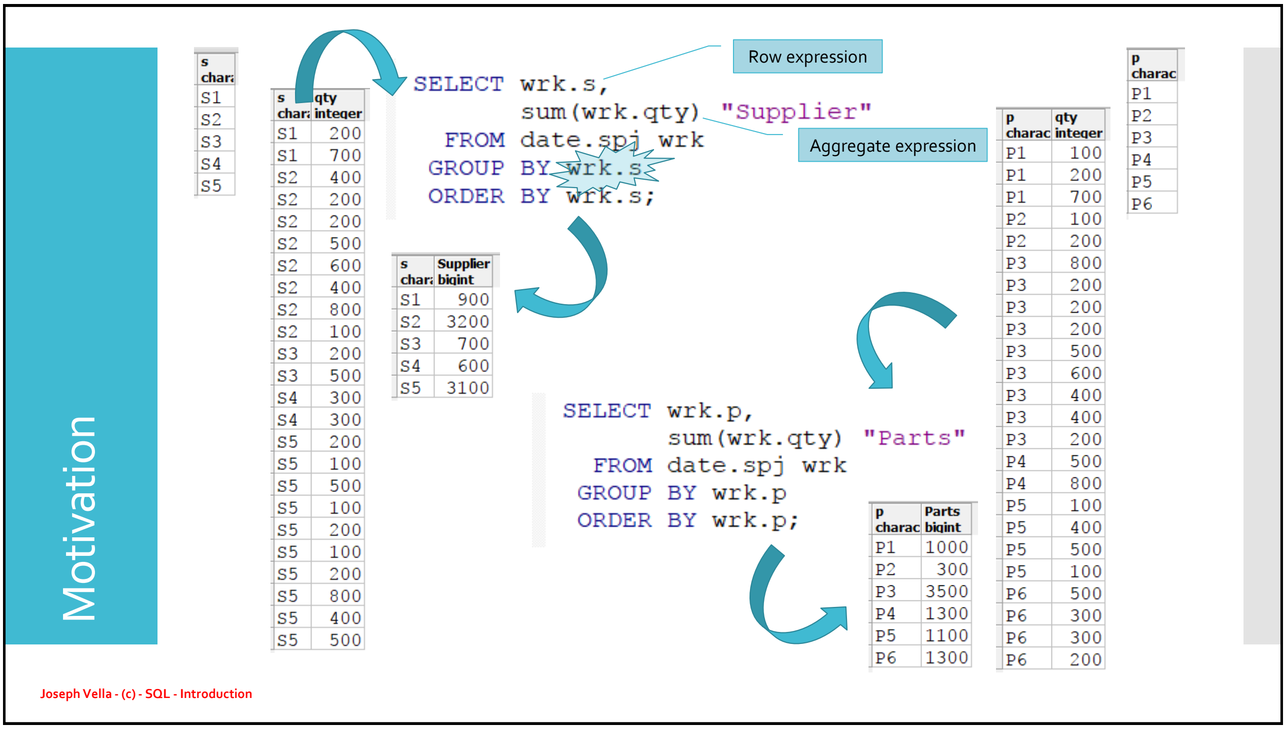
The Generic GROUP BY syntax

General Structure of simple SELECT statement is:

```
 SELECT [DISTINCT] list of attributes
   FROM list of tables
  WHERE row level selection condition expression
ORDER BY list of attributes;
```

General Structure of SELECT statement with Group By

```
   SELECT [DISTINCT] list of expressions
     FROM list of tables
    WHERE row level selection condition expression

   [GROUP BY aggregate on expression list
     [HAVING      aggregate condition expression ] ]
ORDER BY list of attributes;
```



Motivation

s char
S1
S2
S3
S4
S5

s char	qty integer
S1	200
S1	700
S2	400
S2	200
S2	200
S2	500
S2	600
S2	400
S2	800
S2	100
S3	200
S3	500
S4	300
S4	300
S5	200
S5	100
S5	500
S5	100
S5	200
S5	100
S5	200
S5	800
S5	400
S5	500

```
SELECT wrk.s,
       sum(wrk.qty) "Supplier"
  FROM date.spj wrk
 GROUP BY wrk.s
 ORDER BY wrk.s;
```

Row expression

Aggregate expression

s char	Supplier bigint
S1	900
S2	3200
S3	700
S4	600
S5	3100

```
SELECT wrk.p,
       sum(wrk.qty) "Parts"
  FROM date.spj wrk
 GROUP BY wrk.p
 ORDER BY wrk.p;
```

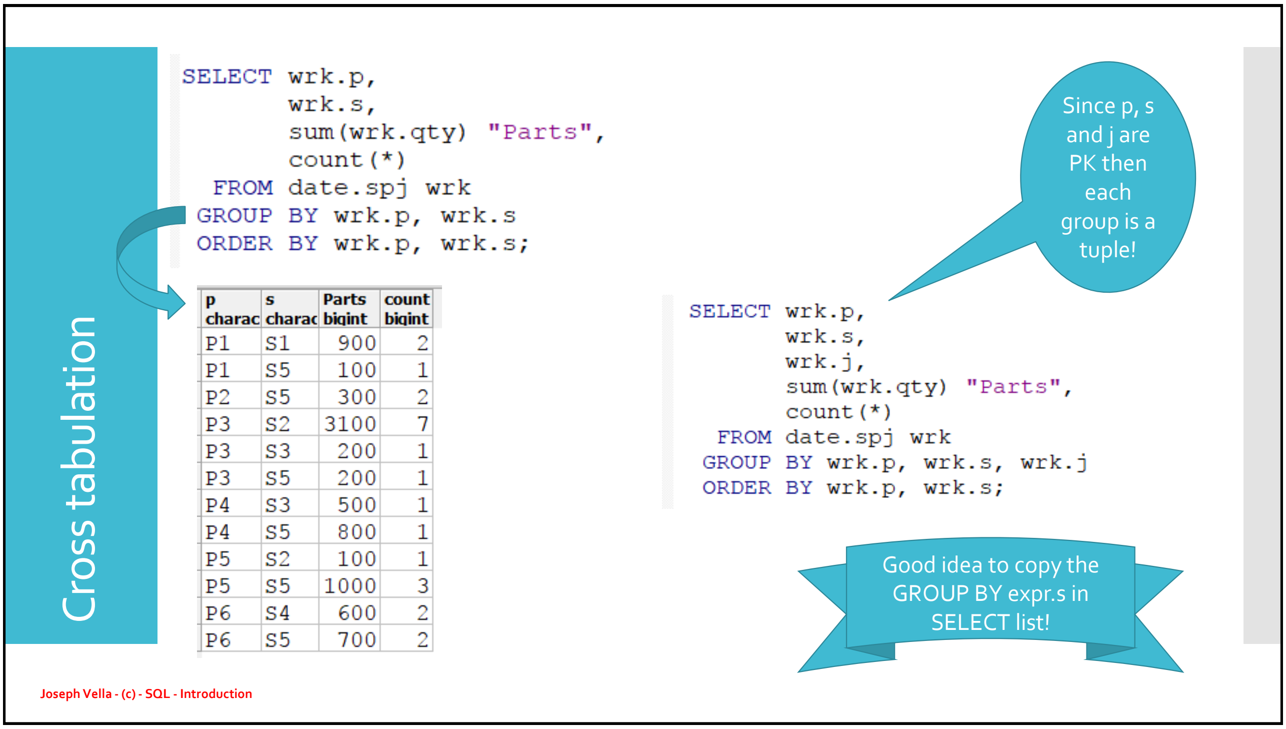
p charac	Parts bigint
P1	1000
P2	300
P3	3500
P4	1300
P5	1100
P6	1300

p charac	qty integer
P1	100
P1	200
P1	700
P2	100
P2	200
P3	800
P3	200
P3	200
P3	200
P3	500
P3	600
P3	400
P3	400
P3	200
P4	500
P4	800
P5	100
P5	400
P5	500
P5	100
P6	500
P6	300
P6	300
P6	200

p charac
P1
P2
P3
P4
P5
P6

Joseph Vella - (c) - SQL - Introduction

73



Cross tabulation

```
SELECT wrk.p,
       wrk.s,
       sum(wrk.qty) "Parts",
       count(*)
  FROM date.spj wrk
 GROUP BY wrk.p, wrk.s
 ORDER BY wrk.p, wrk.s;
```

p charac	s charac	Parts bigint	count bigint
P1	S1	900	2
P1	S5	100	1
P2	S5	300	2
P3	S2	3100	7
P3	S3	200	1
P3	S5	200	1
P4	S3	500	1
P4	S5	800	1
P5	S2	100	1
P5	S5	1000	3
P6	S4	600	2
P6	S5	700	2

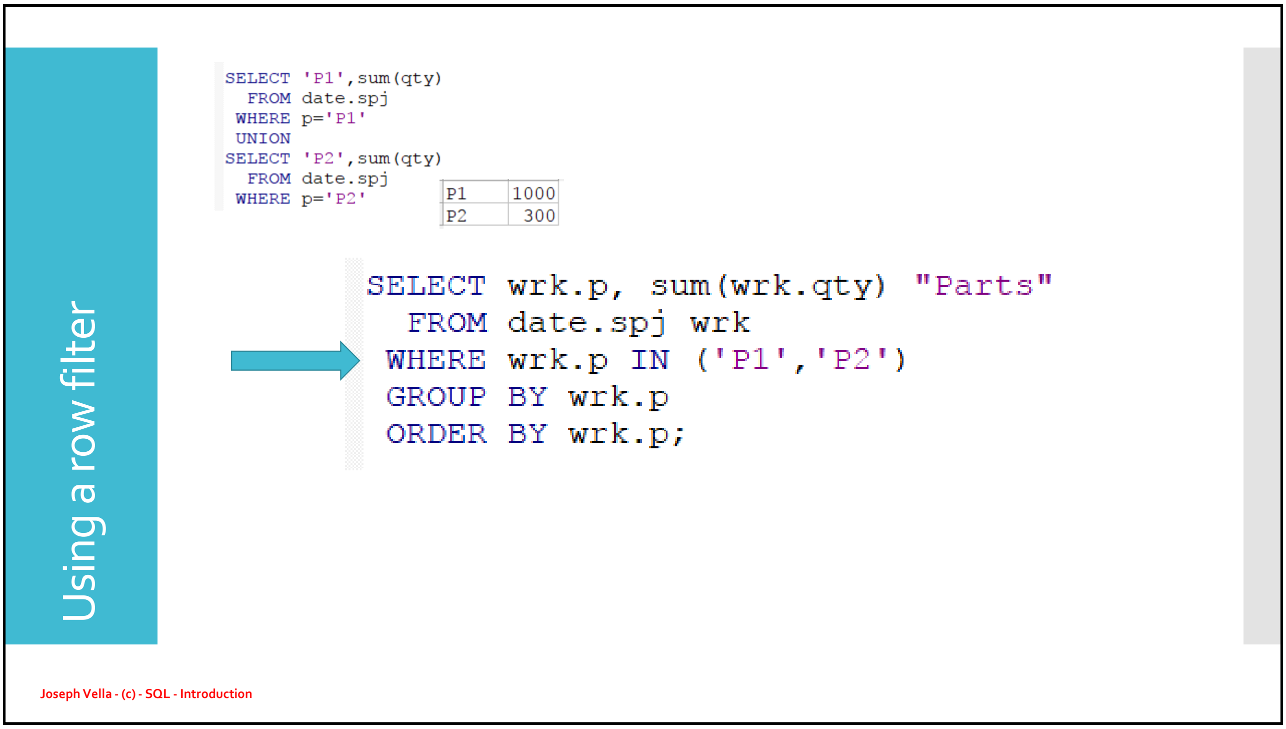
Since p, s and j are PK then each group is a tuple!

```
SELECT wrk.p,
       wrk.s,
       wrk.j,
       sum(wrk.qty) "Parts",
       count(*)
  FROM date.spj wrk
 GROUP BY wrk.p, wrk.s, wrk.j
 ORDER BY wrk.p, wrk.s;
```

Good idea to copy the GROUP BY expr.s in SELECT list!

Joseph Vella - (c) - SQL - Introduction

74



Using a row filter

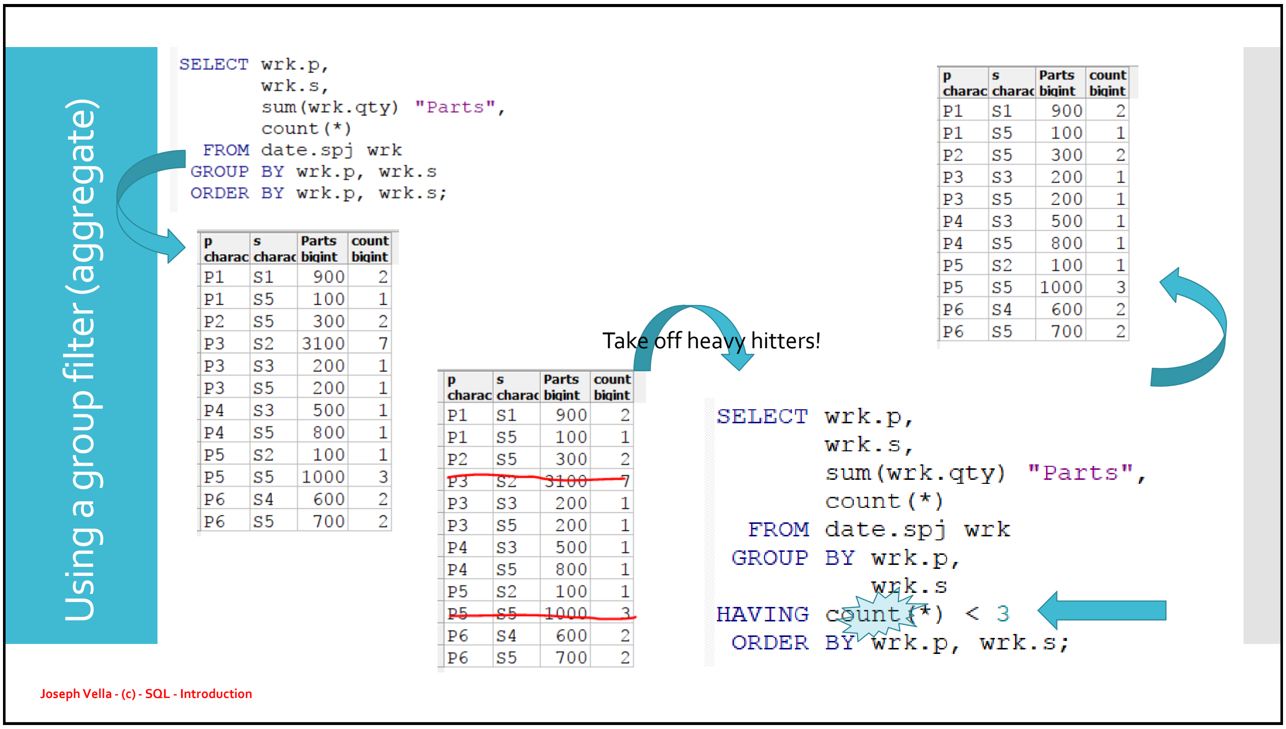
```
SELECT 'P1',sum(qty)
  FROM date.spj
 WHERE p='P1'
 UNION
SELECT 'P2',sum(qty)
  FROM date.spj
 WHERE p='P2'
```

P1	1000
P2	300

```
SELECT wrk.p, sum(wrk.qty) "Parts"
  FROM date.spj wrk
 WHERE wrk.p IN ('P1','P2')
 GROUP BY wrk.p
 ORDER BY wrk.p;
```

Joseph Vella - (c) - SQL - Introduction

75



Using a group filter (aggregate)

```
SELECT wrk.p,
       wrk.s,
       sum(wrk.qty) "Parts",
       count(*)
  FROM date.spj wrk
 GROUP BY wrk.p, wrk.s
 ORDER BY wrk.p, wrk.s;
```

p charac	s charac	Parts bigint	count bigint
P1	S1	900	2
P1	S5	100	1
P2	S5	300	2
P3	S2	3100	7
P3	S3	200	1
P3	S5	200	1
P4	S3	500	1
P4	S5	800	1
P5	S2	100	1
P5	S5	1000	3
P6	S4	600	2
P6	S5	700	2

Take off heavy hitters!

p charac	s charac	Parts bigint	count bigint
P1	S1	900	2
P1	S5	100	1
P2	S5	300	2
~~P3~~	~~S2~~	~~3100~~	~~7~~
P3	S3	200	1
P3	S5	200	1
P4	S3	500	1
P4	S5	800	1
P5	S2	100	1
~~P5~~	~~S5~~	~~1000~~	~~3~~
P6	S4	600	2
P6	S5	700	2

```
SELECT wrk.p,
       wrk.s,
       sum(wrk.qty) "Parts",
       count(*)
  FROM date.spj wrk
 GROUP BY wrk.p,
          wrk.s
HAVING count(*) < 3
 ORDER BY wrk.p, wrk.s;
```

p charac	s charac	Parts bigint	count bigint
P1	S1	900	2
P1	S5	100	1
P2	S5	300	2
P3	S3	200	1
P3	S5	200	1
P4	S3	500	1
P4	S5	800	1
P5	S2	100	1
P5	S5	1000	3
P6	S4	600	2
P6	S5	700	2

Joseph Vella - (c) - SQL - Introduction

76

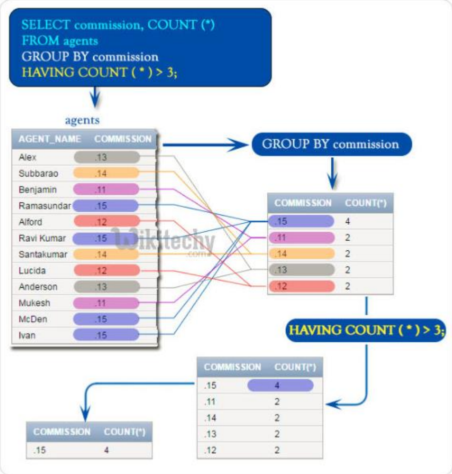
Operational view

Follow the steps for a possible operational view!

SELECT [DISTINCT] list of expressions 5
1 FROM list of tables
2 WHERE row level selection condition expression

3 [GROUP BY aggregate on expression list
4 [HAVING aggregate condition expression]]

ORDER BY list of attributes; 6



```
SELECT commission, COUNT (*)
FROM agents
GROUP BY commission
HAVING COUNT ( * ) > 3;
```

agents

AGENT_NAME	COMMISSION
Alex	.13
Subbarao	.14
Benjamin	.11
Ramasundar	.15
Alford	.12
Ravi Kumar	.15
Santakumar	.14
Lucida	.12
Anderson	.13
Mukesh	.11
McDen	.15
Ivan	.15

GROUP BY commission

COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

HAVING COUNT (*) > 3;

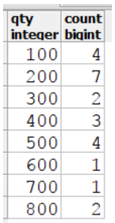
COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

COMMISSION	COUNT(*)
.15	4

Example

- See all "bins"!

```
SELECT wrk.qty,
       count(*)
  FROM date.spj wrk
GROUP BY wrk.qty
ORDER BY wrk.qty;
```

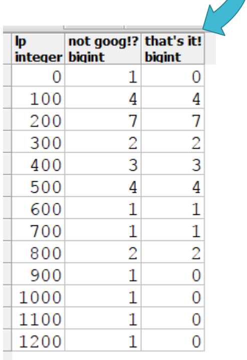


qty integer	count bigint
100	4
200	7
300	2
400	3
500	4
600	1
700	1
800	2

OK! But the boss wants bins from 0 to 1200!?

Outer join tuples have null!?

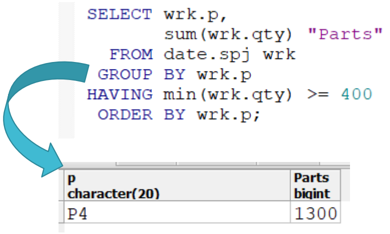
```
SELECT bin.lp,
       count(*) "not goog!?",
       sum(CASE WHEN wrk.qty IS NULL THEN 0 ELSE 1 END) "that's it!"
  FROM date.spj wrk RIGHT OUTER JOIN
       generate_series(0, 1200, 100) as bin(lp)
       ON wrk.qty = bin.lp
 GROUP BY bin.lp
 ORDER BY bin.lp;
```



lp integer	not goog!? bigint	that's it! bigint
0	1	0
100	4	4
200	7	7
300	2	2
400	3	3
500	4	4
600	1	1
700	1	1
800	2	2
900	1	0
1000	1	0
1100	1	0
1200	1	0

Another Example

- Which part is heavily used in works schedule (i.e. never used in quantity less than 400)?



```
SELECT wrk.p,
       sum(wrk.qty) "Parts"
  FROM date.spj wrk
 GROUP BY wrk.p
HAVING min(wrk.qty) >= 400
 ORDER BY wrk.p;
```

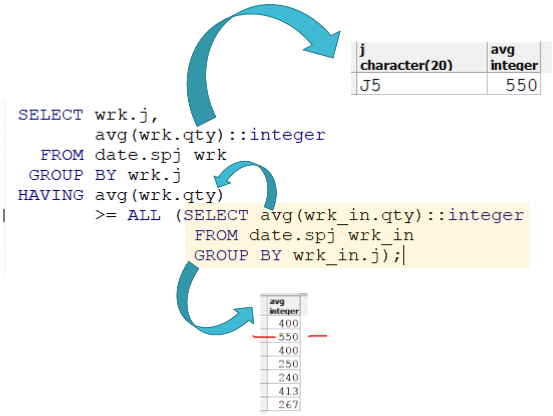
p character(20)	Parts bigint
P4	1300

Joseph Vella - (c) - SQL - Introduction

79

Example

- Which job has the highest average product consumption?



j character(20)	avg integer
J5	550

```
SELECT wrk.j,
       avg(wrk.qty)::integer
  FROM date.spj wrk
 GROUP BY wrk.j
HAVING avg(wrk.qty)
       >= ALL (SELECT avg(wrk_in.qty)::integer
                 FROM date.spj wrk_in
                GROUP BY wrk_in.j);
```

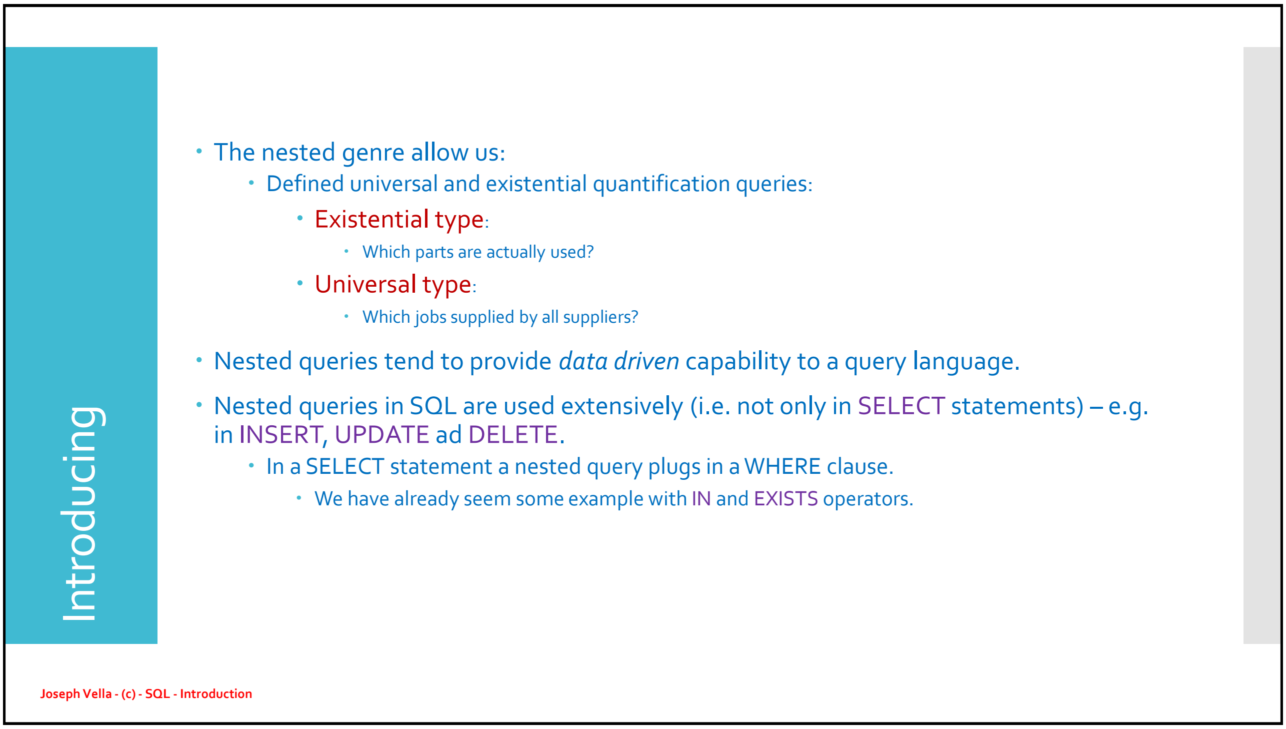
avg integer
400
550
400
250
240
413
267

Joseph Vella - (c) - SQL - Introduction

80

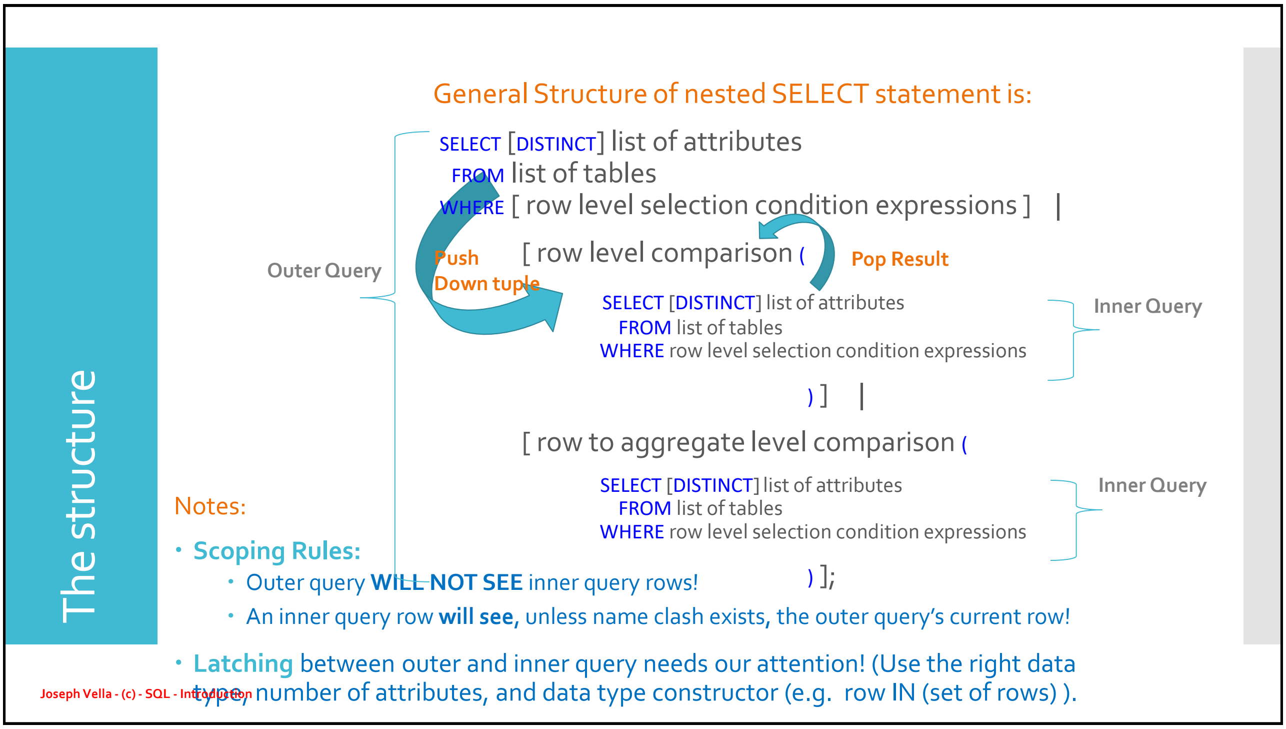
Nested Queries

Includes sub-queries, correlated queries



Introducing

- The nested genre allow us:
  - Defined universal and existential quantification queries:
    - Existential type:
      - Which parts are actually used?
    - Universal type:
      - Which jobs supplied by all suppliers?
- Nested queries tend to provide *data driven* capability to a query language.
- Nested queries in SQL are used extensively (i.e. not only in SELECT statements) – e.g. in INSERT, UPDATE ad DELETE.
  - In a SELECT statement a nested query plugs in a WHERE clause.
    - We have already seem some example with IN and EXISTS operators.



The structure

General Structure of nested SELECT statement is:

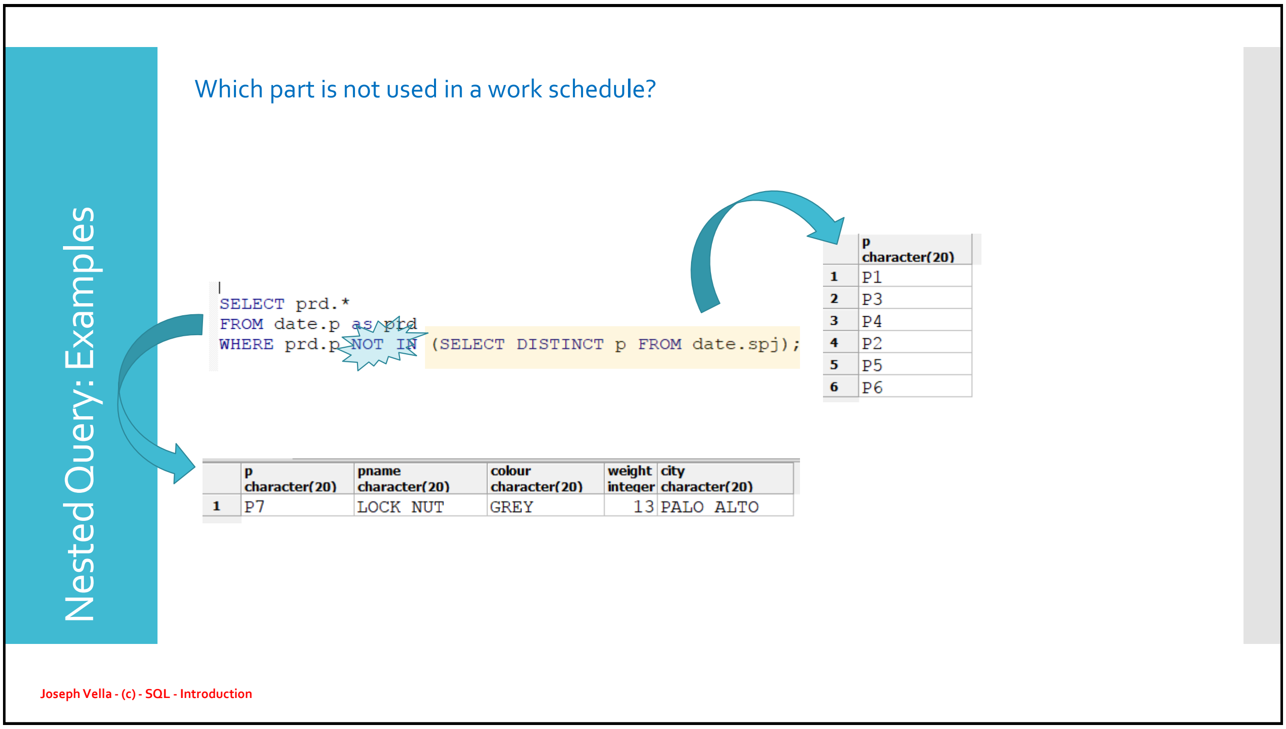
```
SELECT [DISTINCT] list of attributes
 FROM list of tables
WHERE [ row level selection condition expressions ]  |
      [ row level comparison (
            SELECT [DISTINCT] list of attributes
              FROM list of tables
            WHERE row level selection condition expressions
                              ) ]   |
      [ row to aggregate level comparison (
            SELECT [DISTINCT] list of attributes
              FROM list of tables
            WHERE row level selection condition expressions
                              ) ];
```

Notes:

- **Scoping Rules:**
  - Outer query **WILL NOT SEE** inner query rows!
  - An inner query row **will see,** unless name clash exists, the outer query's current row!
- **Latching** between outer and inner query needs our attention! (Use the right data type, number of attributes, and data type constructor (e.g. row IN (set of rows)).

Joseph Vella - (c) - SQL - Introduction

83



Nested Query: Examples

Which part is not used in a work schedule?

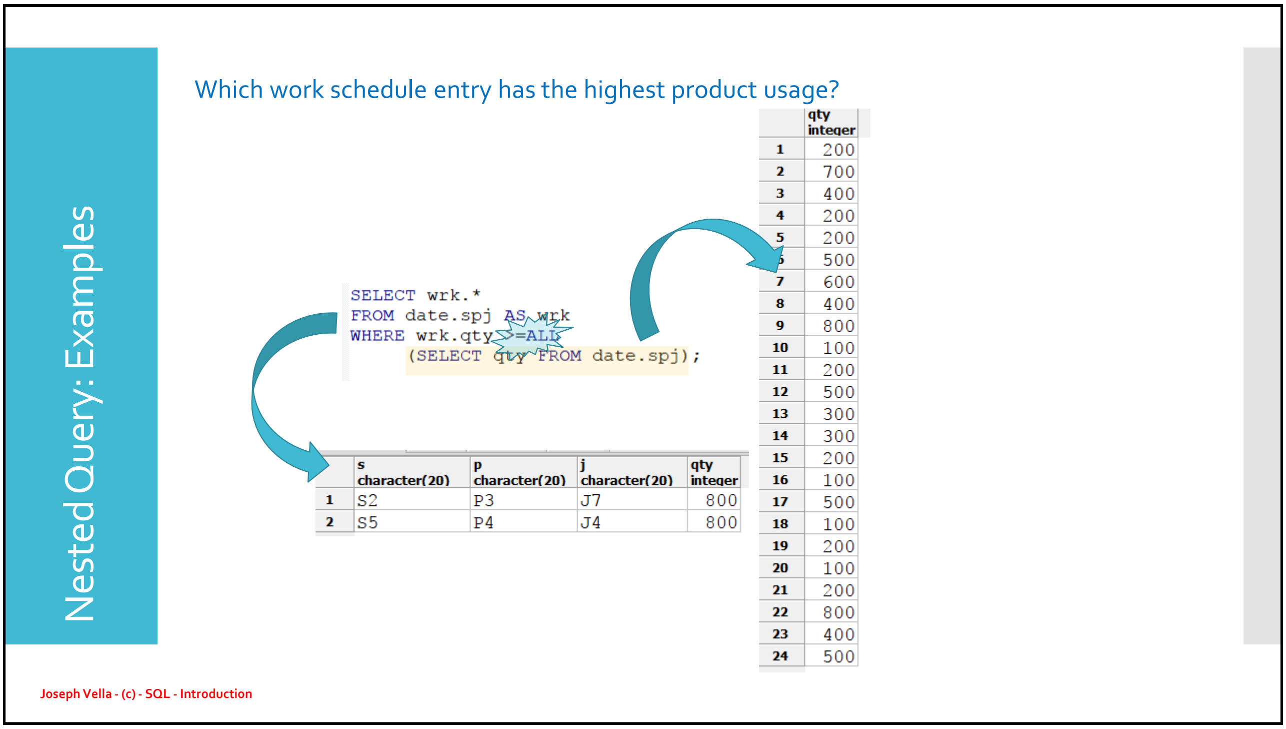
```
SELECT prd.*
FROM date.p as prd
WHERE prd.p NOT IN (SELECT DISTINCT p FROM date.spj);
```

	p character(20)
1	P1
2	P3
3	P4
4	P2
5	P5
6	P6

	p character(20)	pname character(20)	colour character(20)	weight integer	city character(20)
1	P7	LOCK NUT	GREY	13	PALO ALTO

Joseph Vella - (c) - SQL - Introduction

84



Nested Query: Examples

Which work schedule entry has the highest product usage?

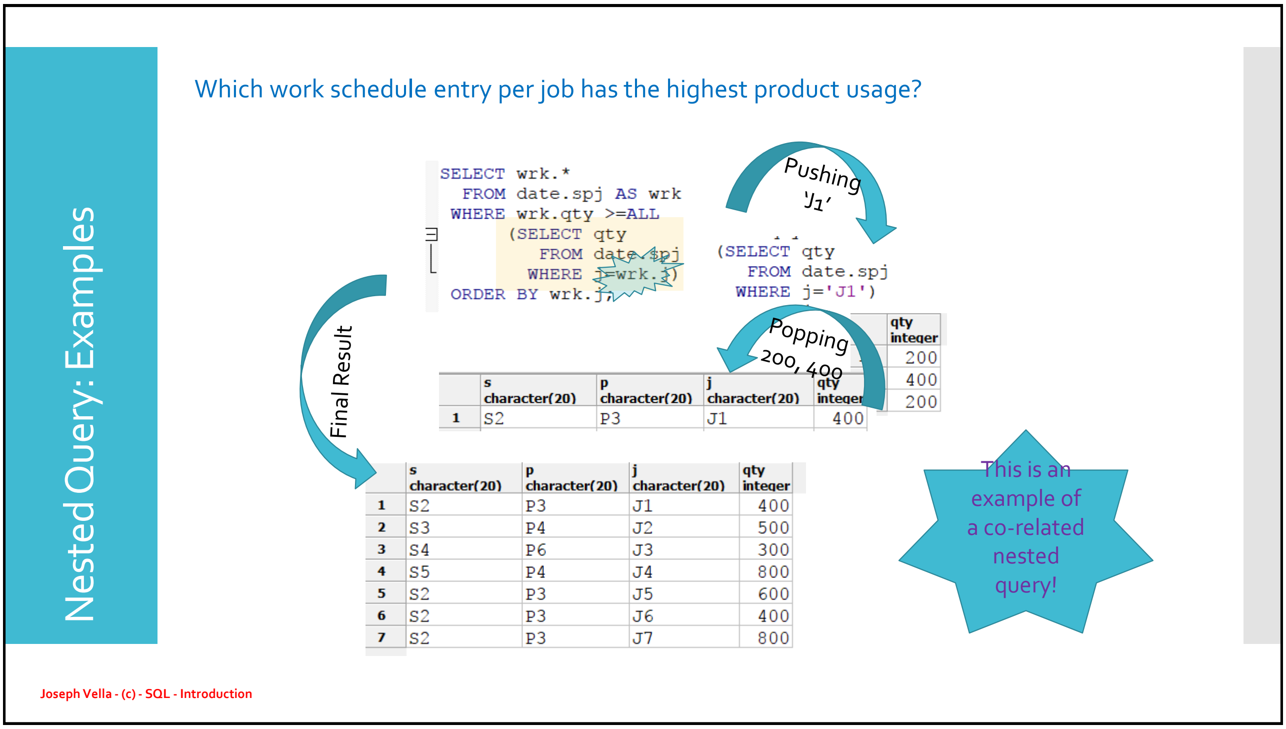
```
SELECT wrk.*
FROM date.spj AS wrk
WHERE wrk.qty >=ALL
      (SELECT qty FROM date.spj);
```

	qty integer
1	200
2	700
3	400
4	200
5	200
6	500
7	600
8	400
9	800
10	100
11	200
12	500
13	300
14	300
15	200
16	100
17	500
18	100
19	200
20	100
21	200
22	800
23	400
24	500

	s character(20)	p character(20)	j character(20)	qty integer
1	S2	P3	J7	800
2	S5	P4	J4	800

Joseph Vella - (c) - SQL - Introduction

85



Nested Query: Examples

Which work schedule entry per job has the highest product usage?

```
SELECT wrk.*
  FROM date.spj AS wrk
 WHERE wrk.qty >=ALL
        (SELECT qty
           FROM date.spj
          WHERE j=wrk.j)
 ORDER BY wrk.j;
```

Pushing 'J1'

```
(SELECT qty
   FROM date.spj
  WHERE j='J1')
```

	qty integer
	200
	400
	200

Popping 200, 400

	s character(20)	p character(20)	j character(20)	qty integer
1	S2	P3	J1	400

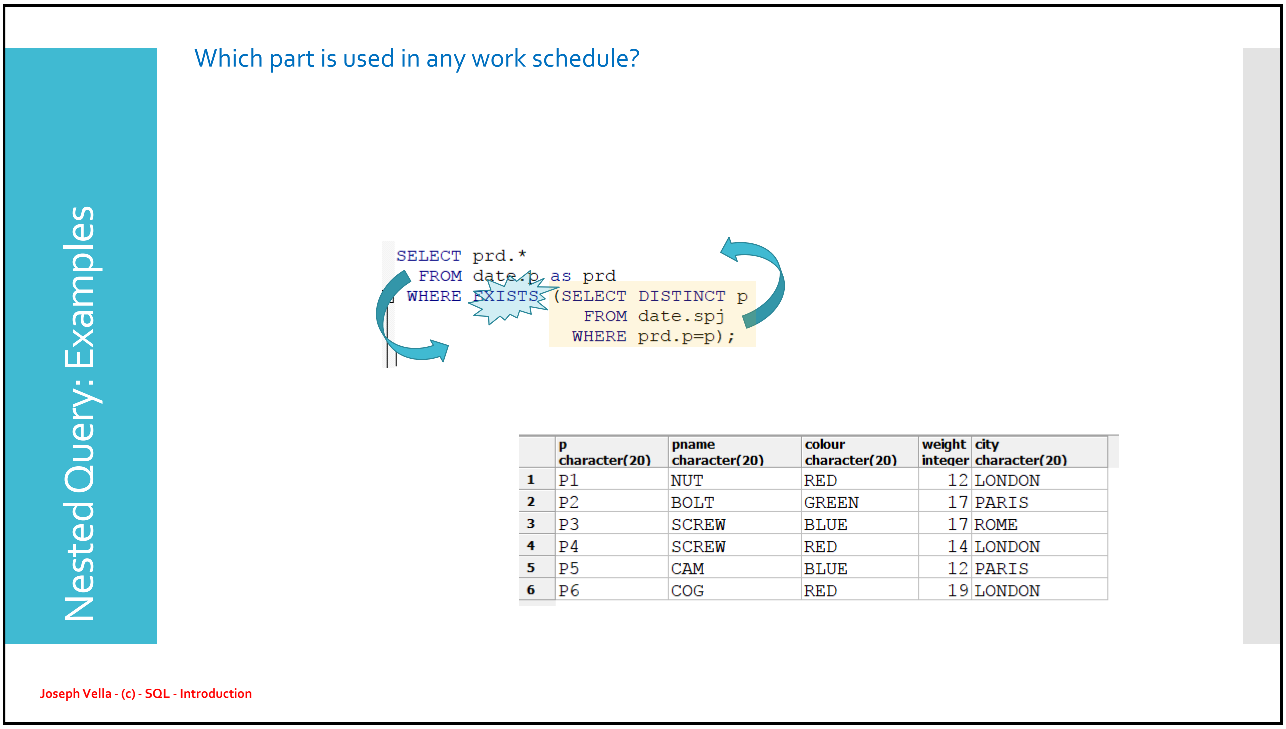
Final Result

	s character(20)	p character(20)	j character(20)	qty integer
1	S2	P3	J1	400
2	S3	P4	J2	500
3	S4	P6	J3	300
4	S5	P4	J4	800
5	S2	P3	J5	600
6	S2	P3	J6	400
7	S2	P3	J7	800

This is an example of a co-related nested query!

Joseph Vella - (c) - SQL - Introduction

86



Nested Query: Examples

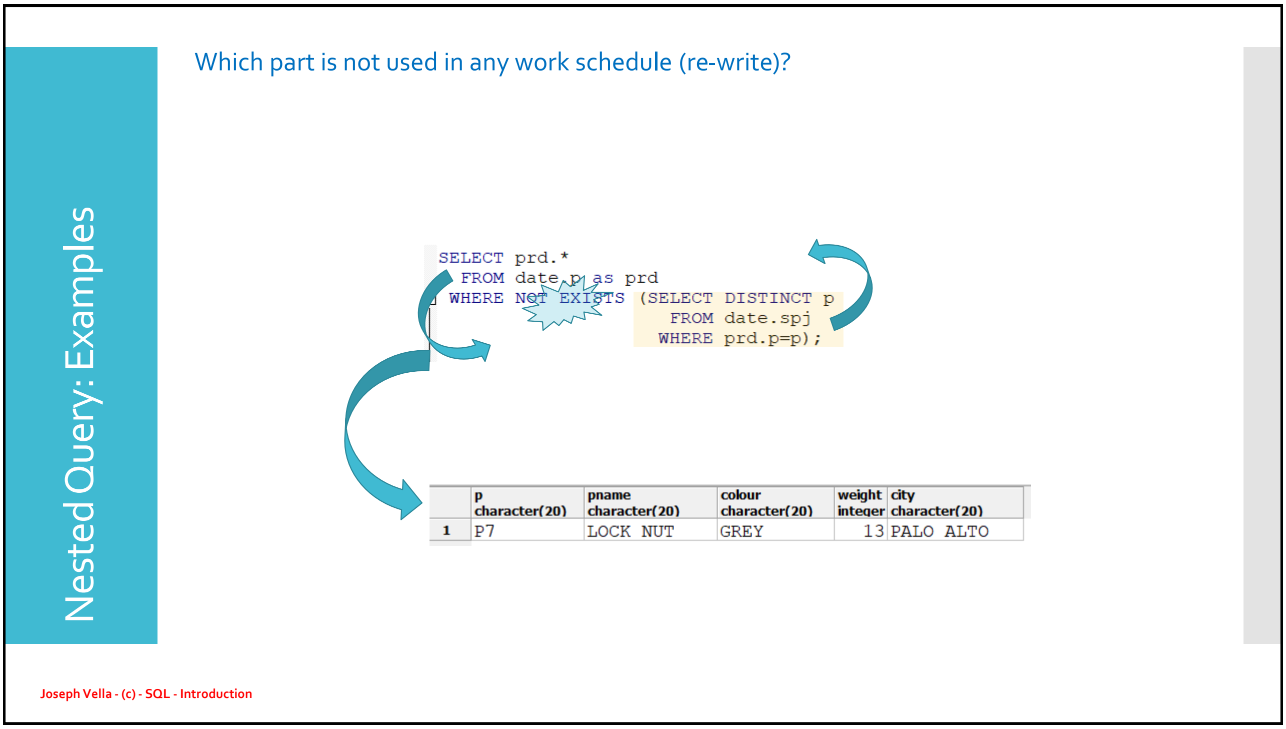
Which part is used in any work schedule?

```
SELECT prd.*
  FROM date.p as prd
 WHERE EXISTS (SELECT DISTINCT p
                FROM date.spj
              WHERE prd.p=p);
```

	p character(20)	pname character(20)	colour character(20)	weight integer	city character(20)
1	P1	NUT	RED	12	LONDON
2	P2	BOLT	GREEN	17	PARIS
3	P3	SCREW	BLUE	17	ROME
4	P4	SCREW	RED	14	LONDON
5	P5	CAM	BLUE	12	PARIS
6	P6	COG	RED	19	LONDON

Joseph Vella - (c) - SQL - Introduction

87



Nested Query: Examples

Which part is not used in any work schedule (re-write)?

```
SELECT prd.*
  FROM date.p as prd
 WHERE NOT EXISTS  (SELECT DISTINCT p
                     FROM date.spj
                   WHERE prd.p=p);
```

	p character(20)	pname character(20)	colour character(20)	weight integer	city character(20)
1	P7	LOCK NUT	GREY	13	PALO ALTO

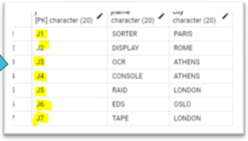
Joseph Vella - (c) - SQL - Introduction

88

Nested Query: Examples (UQ) I

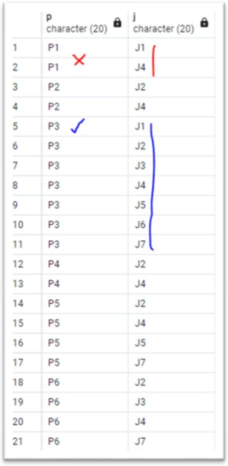
- QUERY:
  "**Which part is used in all jobs?**"
- First, let us work out what "all jobs" is!
- Look for a data source that has the data (i.e. works!)
  - P, J, and WRK
- Examine the data
  - Clearly 'P3' works!
  - All other, e.g., 'P1', fail!?

```
SELECT *
FROM date.j as job;
```



	j [PK] character (20)	jname character (20)	city character (20)
1	J1	SORTER	PARIS
2	J2	DISPLAY	ROME
3	J3	OCR	ATHENS
4	J4	CONSOLE	ATHENS
5	J5	RAID	LONDON
6	J6	EDS	OSLO
7	J7	TAPE	LONDON

```
SELECT DISTINCT wrk.p, wrk.j
FROM date.spj AS wrk
ORDER BY wrk.p, wrk.j;
```



	p character (20)	j character (20)
1	P1	J1
2	P1 ✗	J4
3	P2	J2
4	P2	J4
5	P3 ✓	J1
6	P3	J2
7	P3	J3
8	P3	J4
9	P3	J5
10	P3	J6
11	P3	J7
12	P4	J2
13	P4	J4
14	P5	J2
15	P5	J4
16	P5	J5
17	P5	J7
18	P6	J2
19	P6	J3
20	P6	J4
21	P6	J7

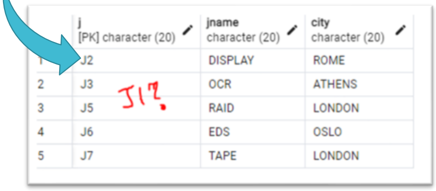
Joseph Vella - (c) - SQL - Introduction

89

Nested Query: Examples (UQ) II

- QUERY (continued):
  "**Which part is used in all jobs?**"
- How to work out success and fail query for 'P1' & 'P3'?

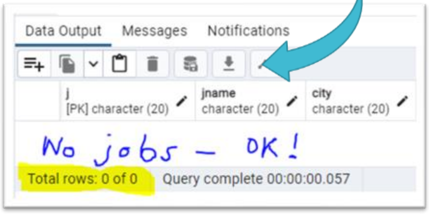
```
SELECT *
FROM DATE.J AS JOB
WHERE NOT EXISTS
          (SELECT *
              FROM DATE.SPJ AS WRK
              WHERE WRK.P = 'P1'
                AND WRK.J = JOB.J);
```



	j [PK] character (20)	jname character (20)	city character (20)
1	J2	DISPLAY	ROME
2	J3	OCR	ATHENS
3	J5	RAID	LONDON
4	J6	EDS	OSLO
5	J7	TAPE	LONDON

J1?

```
SELECT *
FROM DATE.J AS JOB
WHERE NOT EXISTS
          (SELECT *
              FROM DATE.SPJ AS WRK
              WHERE WRK.P = 'P3'
                AND WRK.J = JOB.J);
```



j [PK] character (20)	jname character (20)	city character (20)

No jobs — OK !

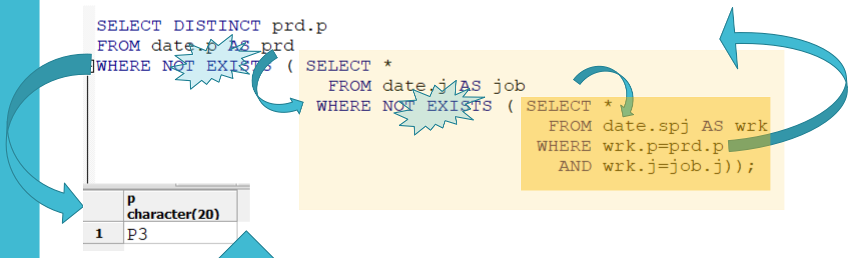
Total rows: 0 of 0 Query complete 00:00:00.057

Joseph Vella - (c) - SQL - Introduction

90

Nested Query: Examples (UQ) III

Which part is used in all jobs?



```
SELECT DISTINCT prd.p
FROM date.p AS prd
WHERE NOT EXISTS ( SELECT *
                    FROM date.j AS job
                   WHERE NOT EXISTS ( SELECT *
                                       FROM date.spj AS wrk
                                      WHERE wrk.p=prd.p
                                        AND wrk.j=job.j));
```

	p character(20)
1	P3

This is an example of a **Universal Quantification** Query (UQ)!

Joseph Vella - (c) - SQL - Introduction

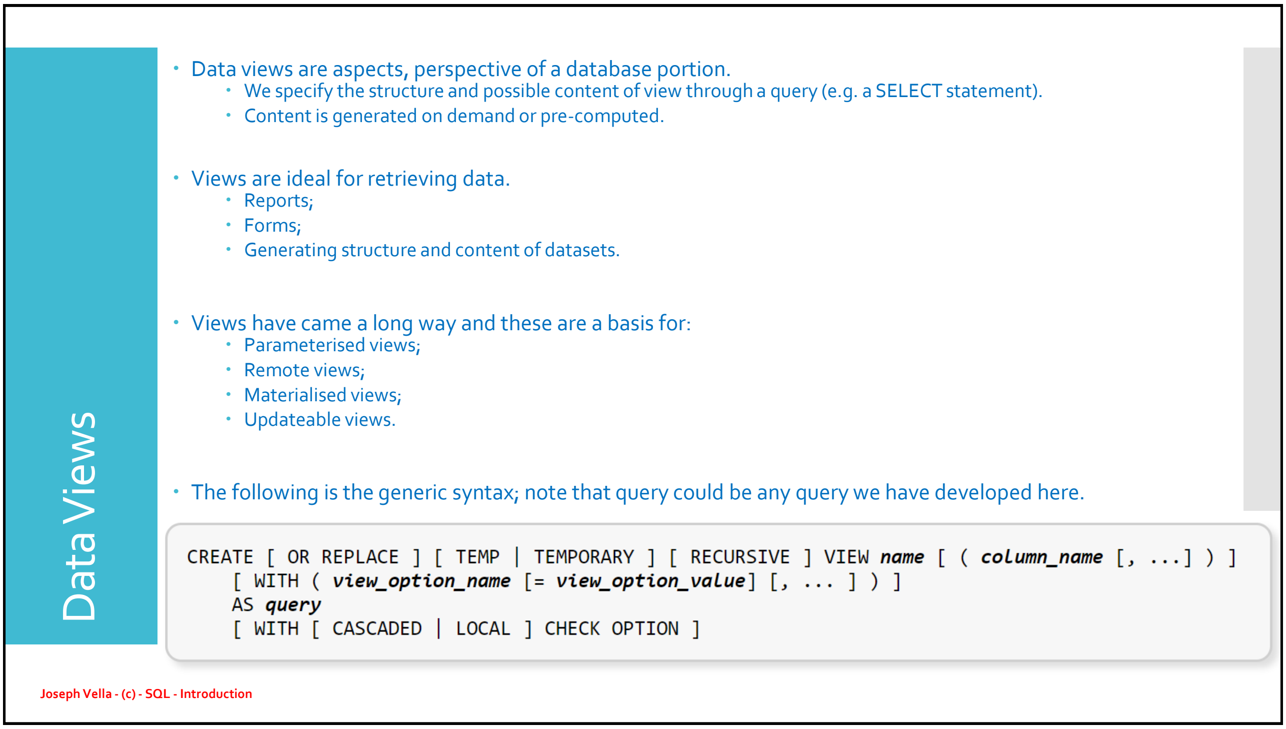
91

Views

Are a convenient way to organise and rehash database data.

Joseph Vella - (c) - SQL - Introduction

92



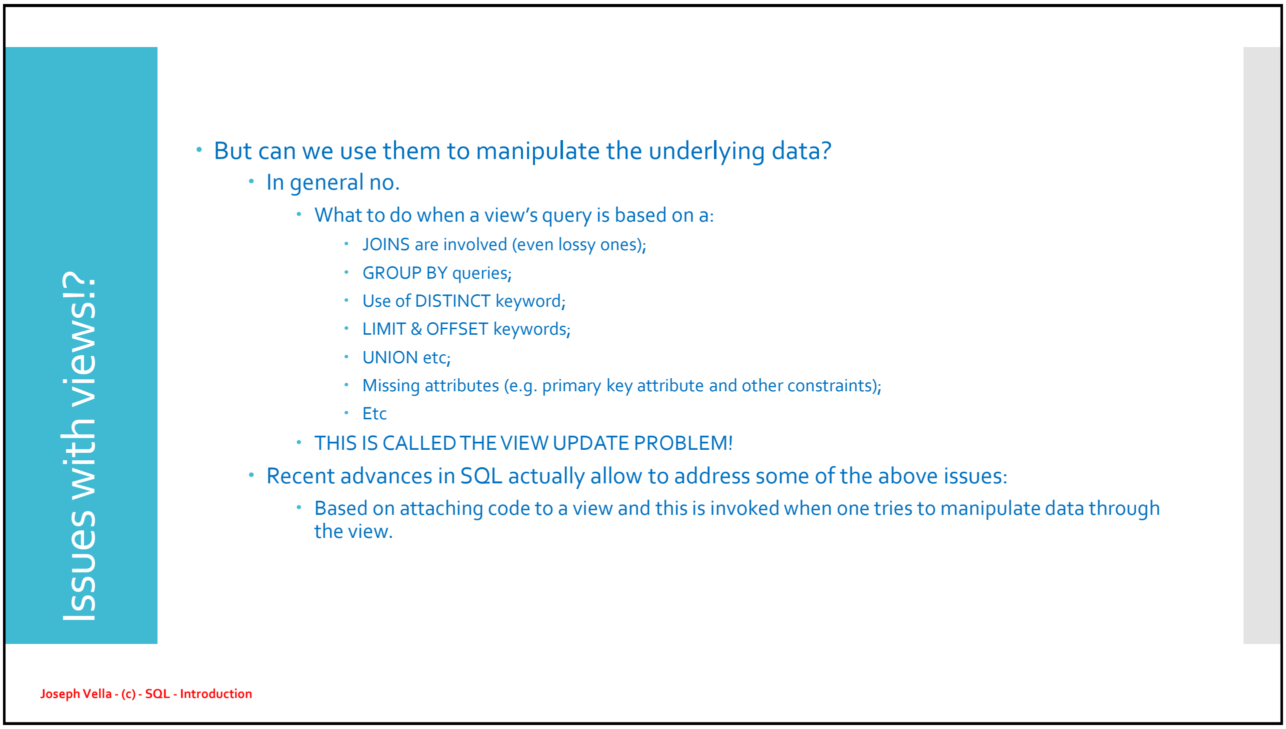
Data Views

- Data views are aspects, perspective of a database portion.
  - We specify the structure and possible content of view through a query (e.g. a SELECT statement).
  - Content is generated on demand or pre-computed.
- Views are ideal for retrieving data.
  - Reports;
  - Forms;
  - Generating structure and content of datasets.
- Views have came a long way and these are a basis for:
  - Parameterised views;
  - Remote views;
  - Materialised views;
  - Updateable views.
- The following is the generic syntax; note that query could be any query we have developed here.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Joseph Vella - (c) - SQL - Introduction

93



Issues with views!?

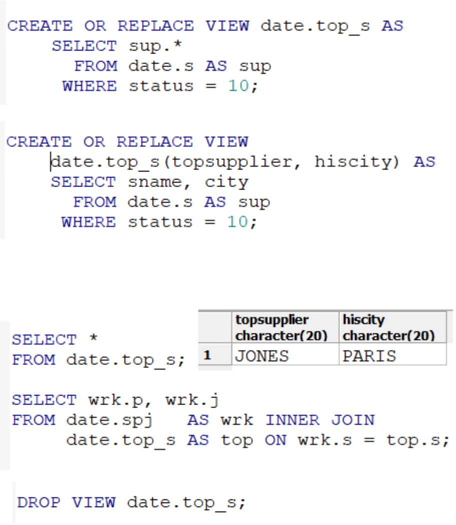
- But can we use them to manipulate the underlying data?
  - In general no.
    - What to do when a view's query is based on a:
      - JOINS are involved (even lossy ones);
      - GROUP BY queries;
      - Use of DISTINCT keyword;
      - LIMIT & OFFSET keywords;
      - UNION etc;
      - Missing attributes (e.g. primary key attribute and other constraints);
      - Etc
    - THIS IS CALLED THE VIEW UPDATE PROBLEM!
  - Recent advances in SQL actually allow to address some of the above issues:
    - Based on attaching code to a view and this is invoked when one tries to manipulate data through the view.

Joseph Vella - (c) - SQL - Introduction

94

Management of a view

- Define it!
  - CREATE VIEW



```
CREATE OR REPLACE VIEW date.top_s AS
    SELECT sup.*
      FROM date.s AS sup
     WHERE status = 10;

CREATE OR REPLACE VIEW
    date.top_s(topsupplier, hiscity) AS
    SELECT sname, city
      FROM date.s AS sup
     WHERE status = 10;
```

- Use it. (And re-use it!)

```
SELECT *
FROM date.top_s;
```

	topsupplier character(20)	hiscity character(20)
1	JONES	PARIS

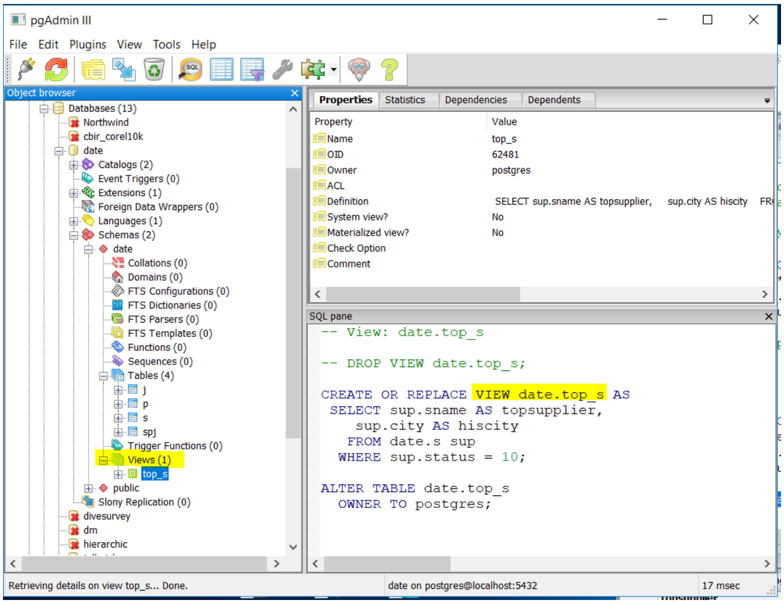
```
SELECT wrk.p, wrk.j
FROM date.spj   AS wrk INNER JOIN
     date.top_s AS top ON wrk.s = top.s;
```

- Drop it!?

```
DROP VIEW date.top_s;
```

95

Views are visible in the DD

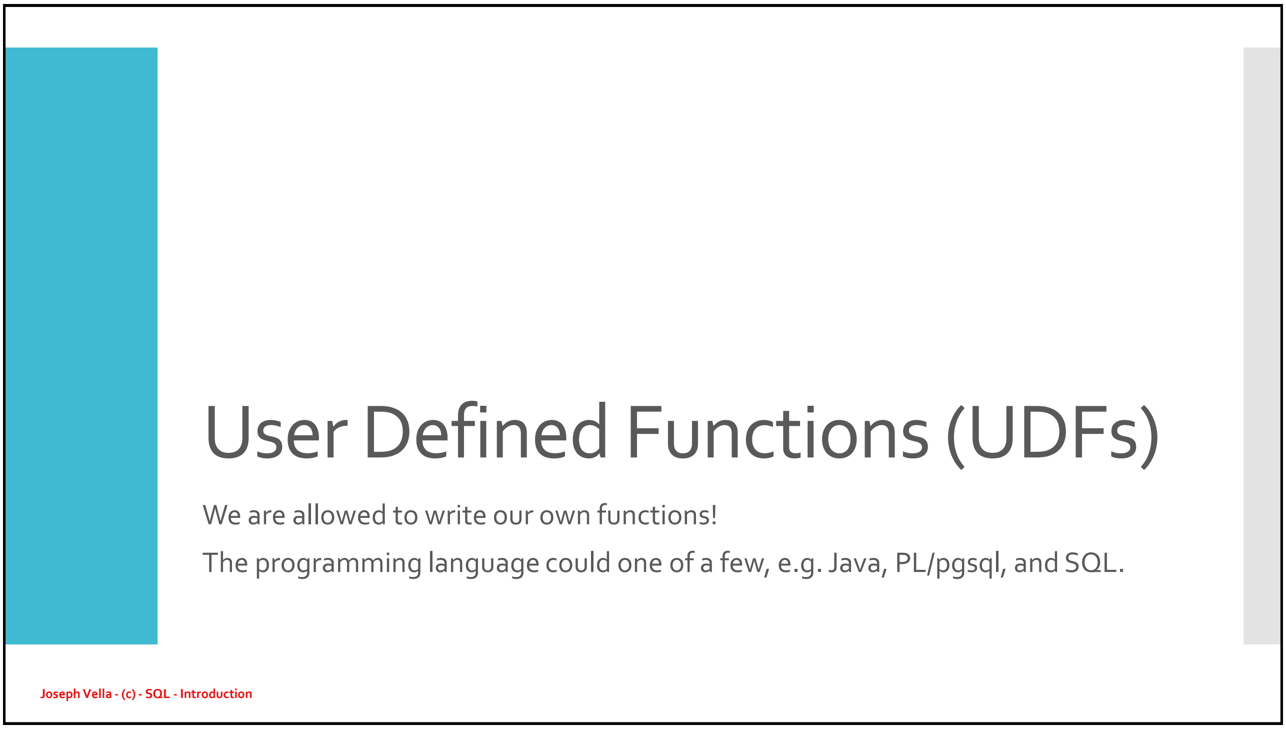


Property	Value
Name	top_s
OID	62481
Owner	postgres
ACL	
Definition	SELECT sup.sname AS topsupplier, sup.city AS hiscity FR
System view?	No
Materialized view?	No
Check Option	
Comment	

```
-- View: date.top_s

-- DROP VIEW date.top_s;

CREATE OR REPLACE VIEW date.top_s AS
 SELECT sup.sname AS topsupplier,
    sup.city AS hiscity
   FROM date.s sup
  WHERE sup.status = 10;

ALTER TABLE date.top_s
  OWNER TO postgres;
```

Retrieving details on view top_s... Done. date on postgres@localhost:5432 17 msec

96



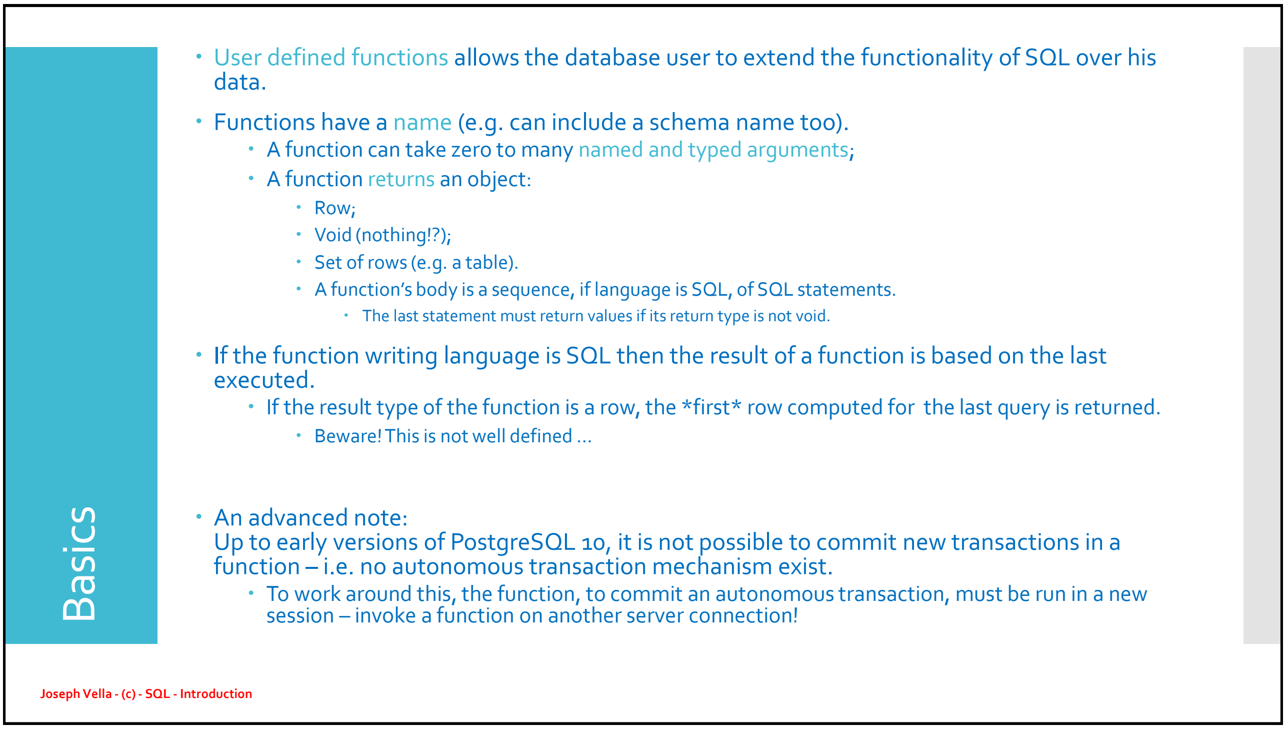
User Defined Functions (UDFs)

We are allowed to write our own functions!

The programming language could one of a few, e.g. Java, PL/pgsql, and SQL.

Joseph Vella - (c) - SQL - Introduction

97



Basics

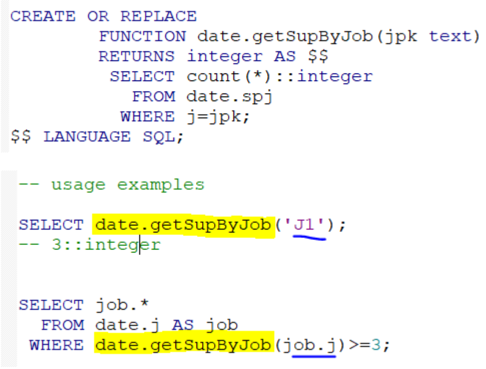
- User defined functions allows the database user to extend the functionality of SQL over his data.
- Functions have a name (e.g. can include a schema name too).
  - A function can take zero to many named and typed arguments;
  - A function returns an object:
    - Row;
    - Void (nothing!?);
    - Set of rows (e.g. a table).
    - A function's body is a sequence, if language is SQL, of SQL statements.
      - The last statement must return values if its return type is not void.
- If the function writing language is SQL then the result of a function is based on the last executed.
  - If the result type of the function is a row, the *first* row computed for the last query is returned.
    - Beware! This is not well defined …
- An advanced note:
  Up to early versions of PostgreSQL 10, it is not possible to commit new transactions in a function – i.e. no autonomous transaction mechanism exist.
  - To work around this, the function, to commit an autonomous transaction, must be run in a new session – invoke a function on another server connection!

Joseph Vella - (c) - SQL - Introduction

98

Define and use a UDF

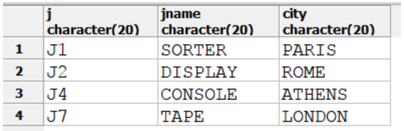
Given a job, as an argument, work out the relative entries in the work schedule.



```
CREATE OR REPLACE
        FUNCTION date.getSupByJob(jpk text)
        RETURNS integer AS $$
         SELECT count(*)::integer
           FROM date.spj
          WHERE j=jpk;
$$ LANGUAGE SQL;
```

Execute this to compile it!

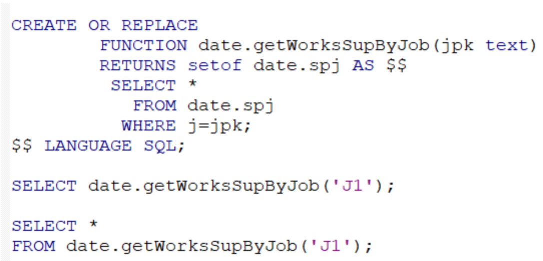
```
-- usage examples

SELECT date.getSupByJob('J1');
-- 3::integer


SELECT job.*
  FROM date.j AS job
 WHERE date.getSupByJob(job.j)>=3;
```



	j character(20)	jname character(20)	city character(20)
1	J1	SORTER	PARIS
2	J2	DISPLAY	ROME
3	J4	CONSOLE	ATHENS
4	J7	TAPE	LONDON

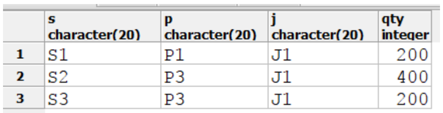
An example

Get all work detail records related to job 'J1'.

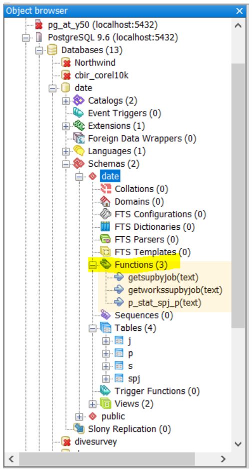


```
CREATE OR REPLACE
        FUNCTION date.getWorksSupByJob(jpk text)
        RETURNS setof date.spj AS $$
         SELECT *
           FROM date.spj
          WHERE j=jpk;
$$ LANGUAGE SQL;

SELECT date.getWorksSupByJob('J1');

SELECT *
FROM date.getWorksSupByJob('J1');
```

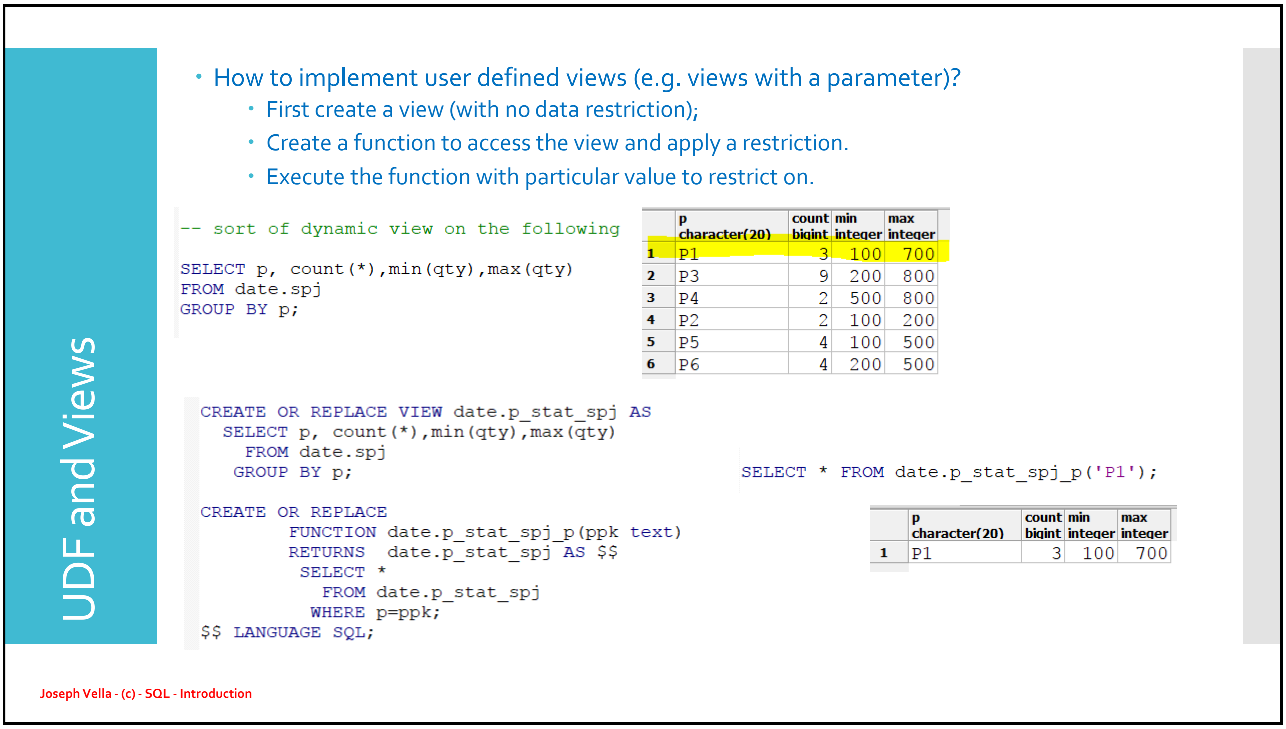
Execute this to compile it!



	s character(20)	p character(20)	j character(20)	qty integer
1	S1	P1	J1	200
2	S2	P3	J1	400
3	S3	P3	J1	200



```
Object browser
  pg_at_y50 (localhost:5432)
  PostgreSQL 9.6 (localhost:5432)
    Databases (13)
      Northwind
      cbir_corel10k
      date
        Catalogs (2)
        Event Triggers (0)
        Extensions (1)
        Foreign Data Wrappers (0)
        Languages (1)
        Schemas (2)
          date
            Collations (0)
            Domains (0)
            FTS Configurations (0)
            FTS Dictionaries (0)
            FTS Parsers (0)
            FTS Templates (0)
            Functions (3)
              getsupbyjob(text)
              getworkssupbyjob(text)
              p_stat_spj_p(text)
            Sequences (0)
            Tables (4)
              j
              p
              s
              spj
            Trigger Functions (0)
            Views (2)
          public
        Slony Replication (0)
      divesurvey
```



UDF and Views

- How to implement user defined views (e.g. views with a parameter)?
  - First create a view (with no data restriction);
  - Create a function to access the view and apply a restriction.
  - Execute the function with particular value to restrict on.

```
-- sort of dynamic view on the following

SELECT p, count(*),min(qty),max(qty)
FROM date.spj
GROUP BY p;
```

	p character(20)	count bigint	min integer	max integer
1	P1	3	100	700
2	P3	9	200	800
3	P4	2	500	800
4	P2	2	100	200
5	P5	4	100	500
6	P6	4	200	500

```
CREATE OR REPLACE VIEW date.p_stat_spj AS
  SELECT p, count(*),min(qty),max(qty)
    FROM date.spj
   GROUP BY p;

CREATE OR REPLACE
        FUNCTION date.p_stat_spj_p(ppk text)
        RETURNS  date.p_stat_spj AS $$
         SELECT *
           FROM date.p_stat_spj
          WHERE p=ppk;
$$ LANGUAGE SQL;
```

```
SELECT * FROM date.p_stat_spj_p('P1');
```

	p character(20)	count bigint	min integer	max integer
1	P1	3	100	700

Joseph Vella - (c) - SQL - Introduction

101