SQL Programming

P.J. McBrien

Imperial College London

Topic 10: SQL Extensions to RA Select, Project and Join

P.J. McBrien

Imperial College London

Bank Branch Database

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

key branch(sortcode)
key branch(bname)
key movement(mid)
key account(no)

movement(no) $\stackrel{fk}{\Rightarrow}$ account(no)

account(sortcode) $\stackrel{fk}{\Rightarrow}$ branch(sortcode)

SQL Pattern Matching

Testing Strings against a Pattern in ANSI SQL

```
WHERE column LIKE pattern ESCAPE escape_char
```

Will return TRUE where pattern matches column. The escape_char may be used before any of the special characters below to allow them to be treated as normal text.

- _ to match a single character
- % to match any number (including zero) of characters

SQL Pattern Matching

Testing Strings against a Pattern in ANSI SQL

```
WHERE column LIKE pattern ESCAPE escape_char
```

Will return TRUE where pattern matches column. The escape_char may be used before any of the special characters below to allow them to be treated as normal text.

- _ to match a single character
- % to match any number (including zero) of characters

Testing Strings against a Pattern in Transact SQL

In addition to ANSI SQL patterns:

- TransactSQL: [A–Z] to match a character between A and Z
- TransactSQL: [ABC] to match a characters A, B and C

SQL Pattern Matching

Testing Strings against a Pattern in ANSI SQL

```
WHERE column LIKE pattern ESCAPE escape_char
```

Will return TRUE where pattern matches column. The escape_char may be used before any of the special characters below to allow them to be treated as normal text.

- _ to match a single character
- % to match any number (including zero) of characters

Testing Strings against a Pattern in Transact SQL

In addition to ANSI SQL patterns:

- TransactSQL: [A–Z] to match a character between A and Z
- TransactSQL: [ABC] to match a characters A, B and C

List customers whose first initial is P, and have one more initial (ANSI SQL)

```
SELECT DISTINCT cname
FROM   account
WHERE  cname LIKE '%, P._.'
```

SQL Pattern Matching

Testing Strings against a Pattern in ANSI SQL

```
WHERE column LIKE pattern ESCAPE escape_char
```

Will return TRUE where pattern matches column. The escape_char may be used before any of the special characters below to allow them to be treated as normal text.

- _ to match a single character
- % to match any number (including zero) of characters

Testing Strings against a Pattern in Transact SQL

In addition to ANSI SQL patterns:

- TransactSQL: [A–Z] to match a character between A and Z
- TransactSQL: [ABC] to match a characters A, B and C

List customers whose first initial is between A and L (Transact SQL)

```
SELECT DISTINCT cname
FROM   account
WHERE  cname LIKE '%, [A–L].%'
```

SQL Functions to Process Data Results

Modifications to data in ANSI SQL

Many functions proposed in ANSI SQL.

- Any processing of data to appear in a result set must be placed in the SELECT clause.
- Any processing of data to filter data must be placed in the WHERE clause.

SQL general functions	
function	semantics
COALESCE(val1,val2,...)	first non-NULL value from val1, val2, ...
GREATEST(val1,val2,...)	greatest value from val1, val2, ..., or null if one is NULL
LEAST(val1,val2 ,...)	least value from val1, val2, ..., or null if one is NULL
CAST(val AS type)	converts a value to specified datatype

SQL functions

```
SELECT  no,
        COALESCE(rate,0.00)
          AS current_rate,
        GREATEST(COALESCE(rate,0.00),5.30)
          AS new_rate
FROM    account
```

no	current_rate	new_rate
100	0.00	5.30
101	5.25	5.30
103	0.00	5.30
107	0.00	5.30
119	5.50	5.50
125	0.00	5.30

SQL String Processing

SQL string functions	
function	semantics
CHAR_LENGTH(str)	returns the number of characters in a string
TRIM(LEADING pattern FROM str)	Removes the leading characters in pattern from a string
TRIM(TRAILING pattern FROM str)	Removes the trailing characters in pattern from a string
TRIM(str)	Removes leading and trailing spaces from a string
POSITION(substr IN str)	Finds the position (counting from 1) of the substr in the str
SUBSTRING(str FROM start FOR no)	Extract from str no characaters, starting from position start
str1 || str2	Concatenate two strings together
UPPER(str)	returns the string converted to all capitals
LOWER(str)	returns the string converted to all small letters

SQL String Processing

SQL string functions	
function	semantics
CHAR_LENGTH(str)	returns the number of characters in a string
TRIM(LEADING pattern FROM str)	Removes the leading characters in pattern from a string
TRIM(TRAILING pattern FROM str)	Removes the trailing characters in pattern from a string
TRIM(str)	Removes leading and trailing spaces from a string
POSITION(substr IN str)	Finds the position (counting from 1) of the substr in the str
SUBSTRING(str FROM start FOR no)	Extract from str no characaters, starting from position start
str1 || str2	Concatenate two strings together
UPPER(str)	returns the string converted to all capitals
LOWER(str)	returns the string converted to all small letters

Non-ANSI SQL Functions

String functions tend to be an aspect of SQL implementations that is not ANSI SQL compliant, *e.g.* for ANSI CHAR_LENGTH

- Postgres: LENGTH(str), but also supports CHAR_LENGTH
- TrasnsactSQL: LEN(str)

SQL String Processing

SQL string functions	
function	semantics
CHAR_LENGTH(str)	returns the number of characters in a string
TRIM(LEADING pattern FROM str)	Removes the leading characters in pattern from a string
TRIM(TRAILING pattern FROM str)	Removes the trailing characters in pattern from a string
TRIM(str)	Removes leading and trailing spaces from a string
POSITION(substr IN str)	Finds the position (counting from 1) of the substr in the str
SUBSTRING(str FROM start FOR no)	Extract from str no characaters, starting from position start
str1 || str2	Concatenate two strings together
UPPER(str)	returns the string converted to all capitals
LOWER(str)	returns the string converted to all small letters

Display accounts with just surnames and rounded rates (ANSI SQL)

```
SELECT no,
       ROUND(rate,1) AS rate_1dp,
       SUBSTRING(cname FROM 1 FOR POSITION(',' IN cname)-1) AS surname
FROM   account
```

SQL String Processing

SQL string functions	
function	semantics
CHAR_LENGTH(str)	returns the number of characters in a string
TRIM(LEADING pattern FROM str)	Removes the leading characters in pattern from a string
TRIM(TRAILING pattern FROM str)	Removes the trailing characters in pattern from a string
TRIM(str)	Removes leading and trailing spaces from a string
POSITION(substr IN str)	Finds the position (counting from 1) of the substr in the str
SUBSTRING(str FROM start FOR no)	Extract from str no characaters, starting from position start
str1 || str2	Concatenate two strings together
UPPER(str)	returns the string converted to all capitals
LOWER(str)	returns the string converted to all small letters

Display accounts with just surnames and rounded rates (Transact SQL)

```
SELECT  no,
        ROUND(rate,1) AS rate_1dp,
        SUBSTRING(cname,1,CHARINDEX(',',cname)-1) AS surname
FROM    account
```

SQL Maths Functions

SQL maths functions	
function	semantics
SIN(num)	the sine of angle num (measured in radians)
ASIN(num)	an angle measured in radians that has sine value num
LN(num)	the natural logarithm of num
LOG10(num)	log to base ten of num
SQRT(num)	square root of num
POWER(num,exp)	num raised to the power of exp
EXP(num)	num raised to the power e
ABS(num)	absolute value of num
ROUND(num,dp)	rounds num to dp decimal places

Quiz 10.1: Transact SQL extensions to RA Select and Project

customer				
cname	phone	address	joined	salary
'McBrien, P.'	'02077651234'	'123 Strand, London WC1A'	1999-01-03	30000
'Boyd, M.'	'02077656666'	'33 Aldwych, London'	1999-01-05	NULL
'Poulovassilis, A.'	'02089474321'	'13 Haydons Rd, London SW19'	1999-01-05	40000
'Bailey, J.'	'02089461111'	'22 Queens Rd, London SW19'	1999-01-07	45000

```
SELECT  cname,
        SUBSTRING(address,CHARINDEX(',',address)+2,LEN(address)) AS area
FROM    customer
WHERE   phone LIKE '02089[4-7]%';
```

What is the result of the TransactSQL query?

A

cname	area
Bailey, J.	London SW19
Poulovassilis, A.	London SW19

B

cname	area
Bailey, J.	22 Queens Rd
Poulovassilis, A.	13 Haydons Rd

C

cname	area
Poulovassilis, A.	London SW19

D

cname	area
Poulovassilis, A.	13 Haydons Rd

Processing the result of project: CASE statements

account

no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

A CASE statement may be used to return alternative values depending on condition. Two forms

CASE expr WHEN v1 THEN ... WHEN v2 THEN ... END

Chooses which statement to return based on value of expr

CASE WHEN cond1 THEN ... WHEN cond2 THEN ... END

Choose which statement to return based on which cond1, cond2 is first TRUE

```
SELECT no,
       COALESCE(rate,0.00) AS rate,
       CASE
       WHEN rate>0 AND rate<5.5
       THEN 'low rate'
       WHEN rate>=5.5
       THEN 'high rate'
       ELSE 'zero rate'
       END AS interest_class
FROM   account
```

no	rate	interest_class
100	0.00	zero rate
101	5.25	low rate
103	0.00	zero rate
107	0.00	zero rate
119	5.50	high rate
125	0.00	zero rate

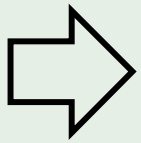
Need for yet another type of Join?

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

Listing of movement mid for all customers with movements

```
SELECT  cname,
        mid
FROM    account NATURAL JOIN
        movement
```



cname	mid
McBrien, P.	1000
McBrien, P.	1001
McBrien, P.	1002
Poulovassilis, A.	1004
Boyd, M.	1005
McBrien, P.	1006
Poulovassilis, A.	1007
McBrien, P.	1008
Poulovassilis, A.	1009

Need for yet another type of Join?

account

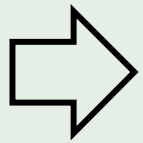
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

Listing any movements for all customers

```
SELECT cname,
       mid
FROM   account NATURAL LEFT JOIN
       movement
```



cname	mid
McBrien, P.	1000
McBrien, P.	1001
McBrien, P.	1002
Poulovassilis, A.	1004
Boyd, M.	1005
McBrien, P.	1006
Poulovassilis, A.	1007
McBrien, P.	1008
Poulovassilis, A.	1009
Bailey, J.	NULL

Left and Right Joins

Left Join

A left join $R \overset{\text{L}}{\bowtie} S$ returns every row in R, even if no rows in S match. In such cases where no row in S matches a row from R, the columns of S are filled with NULL values.

Right Join

A right join $R \overset{\text{R}}{\bowtie} S$ returns every row in S, even if no rows in R match. In such cases where no row in R matches a row from S, the columns of R are filled with NULL values.

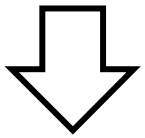
Outer Join

An outer join $R \overset{\text{O}}{\bowtie} S$ returns every row in R, even if no rows in S match, and also returns every row in S even if no row in R matches.

$$R \overset{\text{O}}{\bowtie} S \equiv (R \overset{\text{L}}{\bowtie} S) \cup (R \overset{\text{R}}{\bowtie} S)$$

RA equivalent of LEFT JOIN

SELECT $A_1, ..., A_n$
FROM R_1 LEFT JOIN R_2 ON O_1 AND ... AND O_i
WHERE P_1
AND ...
AND P_k



$$\pi_{A_1,...,A_n}\sigma_{P_1\wedge...\wedge P_k}(\sigma_{O_1\wedge...\wedge O_i}(R_1 \times R_2) \cup (R_1 - \sigma_{O_1\wedge...\wedge O_i}(R_1 \ltimes R_2) \times \omega(R_2)))$$

- $\omega(R_2)$ returns a row of NULLs with the same number of columns as R_2

Quiz 10.2: SQL LEFT JOIN ... ON (1)

```
SELECT  account.no,
        movement.amount
FROM    account LEFT JOIN movement
                ON    account.no=movement.no
                AND   movement.amount<0
```

What is the result of the above query?

A

no	amount

B

no	amount
100	-223.45
107	-100.00

C

no	amount
100	-223.45
101	NULL
103	NULL
107	-100.00
119	NULL
125	NULL

D

no	amount
100	-223.45
101	0.00
103	0.00
107	-100.00
119	0.00
125	0.00

Quiz 10.3: SQL LEFT JOIN ... ON (2)

```
SELECT  account.no,
        movement.amount
FROM    account LEFT JOIN movement
                ON    account.no=movement.no
WHERE   movement.amount<0
```

What is the result of the above query?

A

no	amount

B

no	amount
100	-223.45
107	-100.00

C

no	amount
100	-223.45
101	NULL
103	NULL
107	-100.00
119	NULL
125	NULL

D

no	amount
100	-223.45
101	0.00
103	0.00
107	-100.00
119	0.00
125	0.00

Worksheet: Left, Right and Outer Joins

bank_branch_null database

movement			
mid	no?	amount?	tdate?
0999	119	45.00	null
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999
1010	100	null	20/1/1999
1011	null	null	20/1/1999
1012	null	600.00	20/1/1999
1013	null	-46.00	20/1/1999

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	null	67
101	'deposit'	'McBrien, P.'	5.25	67
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	null	56

Topic 11: SQL OLAP

P.J. McBrien

Imperial College London

OLTP and OLAP

OLTP

- online transactional processing
- reads and writes to a few rows
- 'standard' data processing

```
BEGIN TRANSACTION T1
  UPDATE branch
  SET    cash=cash-10000.00
  WHERE  sortcode=56

  UPDATE branch
  SET    cash=cash+10000.00
  WHERE  sortcode=34
COMMIT TRANSACTION T1
```

OLAP

- online analytical processing
- reads many rows
- management information

```
BEGIN TRANSACTION T4
  SELECT SUM(cash)
  FROM   branch
COMMIT TRANSACTION T4
```

SQL OLAP features: Ordering Rows

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- ORDER BY presents data to a user in specified order, but data is not changed
- Default is ASCending order
- Can specify DESCending order
- To limit the number of rows returned, use FETCH FIRST or FETCH NEXT

```
SELECT    mid,
          tdate,
          amount
FROM      movement
ORDER BY  mid
```

mid	tdate	amount
1000	1999-01-05	2300.00
1001	1999-01-05	4000.00
1002	1999-01-08	-223.45
1004	1999-01-11	-100.00
1005	1999-01-12	145.50
1006	1999-01-15	10.23
1007	1999-01-15	345.56
1008	1999-01-15	1230.00
1009	1999-01-18	5600.00

SQL OLAP features: Ordering Rows

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- ORDER BY presents data to a user in specified order, but data is not changed
- Default is ASCending order
- Can specify DESCending order
- To limit the number of rows returned, use FETCH FIRST or FETCH NEXT

```
SELECT    mid,
          tdate,
          amount
FROM      movement
ORDER BY  amount DESC
```

mid	tdate	amount
1009	1999-01-18	5600.00
1001	1999-01-05	4000.00
1000	1999-01-05	2300.00
1008	1999-01-15	1230.00
1007	1999-01-15	345.56
1005	1999-01-12	145.50
1006	1999-01-15	10.23
1004	1999-01-11	-100.00
1002	1999-01-08	-223.45

SQL OLAP features: Ordering Rows

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- ORDER BY presents data to a user in specified order, but data is not changed
- Default is ASCending order
- Can specify DESCending order
- To limit the number of rows returned, use FETCH FIRST or FETCH NEXT

```
SELECT    mid,
          tdate,
          amount
FROM      movement
ORDER BY  amount DESC
FETCH     FIRST 3 ROWS ONLY
```

mid	tdate	amount
1009	1999-01-18	5600.00
1001	1999-01-05	4000.00
1000	1999-01-05	2300.00

SQL OLAP features: Ordering Rows

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- ORDER BY presents data to a user in specified order, but data is not changed
- Default is ASCending order
- Can specify DESCending order
- To limit the number of rows returned, use FETCH FIRST or FETCH NEXT

```
SELECT    mid,
          tdate,
          amount
FROM      movement
ORDER BY  amount DESC
OFFSET    3
FETCH     NEXT 4 ROWS ONLY
```

mid	tdate	amount
1008	1999-01-15	1230.00
1007	1999-01-15	345.56
1005	1999-01-12	145.50
1006	1999-01-15	10.23

SQL OLAP features: GROUP BY

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
.
.
.
FROM movement
.
.
.
GROUP BY no
```

SQL OLAP features: GROUP BY

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
.
.
.
FROM movement
.
.
.
GROUP BY no
```

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1002		-223.45	8/1/1999
1006		10.23	15/1/1999
1001	101	4000.00	5/1/1999
1008		1230.00	15/1/1999
1004	107	-100.00	11/1/1999
1007		345.56	15/1/1999
1005	103	145.50	12/1/1999
1009	119	5600.00	18/1/1999

Aggregate Functions

Aggregate	Semantics
SUM	Sum the values of all rows in the group
COUNT	Count the number of non-NULL rows in the group
AVG	Average of the non-NULL values in the group
MIN	Minimum value in the group
MAX	Maximum value in the group
ANY_VALUE	A random non-NULL value from the group (new in SQL:2023)
ARRAY_AGG	Generate an array containing all the values of the group
STDDEV_POP	Calculated the population standard deviation

GROUP BY rules

- Only one row output per group
- *ANSI SQL says must apply aggregate function to non grouped columns*

SQL OLAP features: GROUP BY

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
.
.
.
FROM movement
.
.
.
GROUP BY no
```

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1002		-223.45	8/1/1999
1006		10.23	15/1/1999
1001	101	4000.00	5/1/1999
1008		1230.00	15/1/1999
1004	107	-100.00	11/1/1999
1007		345.56	15/1/1999
1005	103	145.50	12/1/1999
1009	119	5600.00	18/1/1999

Example of Aggregate Functions

```
SELECT no,
       SUM(amount) AS balance,
       COUNT(amount) AS trans
       FROM movement
       GROUP BY no
```

no	balance	trans
101	5230.00	2
103	145.50	1
119	5600.00	1
107	245.56	2
100	2086.78	3

GROUP BY rules

- Only one row output per group
- *ANSI SQL says must apply aggregate function to non grouped columns*

Quiz 11.1: GROUP BY in ANSI SQL

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

Which SQL query is not permitted in ANSI SQL?

A

```
SELECT   no,
         cname,
         AVG(rate)
FROM     account
GROUP BY no
```

B

```
SELECT   no,
         MIN(cname),
         AVG(rate)
FROM     account
GROUP BY no
```

C

```
SELECT   no,
         MIN(rate),
         MAX(rate)
FROM     account
GROUP BY no
```

D

```
SELECT AVG(rate)
FROM   account
```

SQL OLAP features: Aggregate operators

Use GROUP BY on all non aggregated columns

```
SELECT no,
       SUM(amount) AS balance,
       COUNT(amount) AS trans
       FROM movement
       GROUP BY no
```

no	balance	trans
101	5230.00	2
103	145.50	1
119	5600.00	1
107	245.56	2
100	2086.78	3

SQL OLAP features: Aggregate operators

Use GROUP BY on all non aggregated columns

```
SELECT no,
       SUM(amount) AS balance,
       COUNT(amount) AS trans
       FROM movement
       GROUP BY no
```

no	balance	trans
101	5230.00	2
103	145.50	1
119	5600.00	1
107	245.56	2
100	2086.78	3

Choose bag or set semantics for COUNT

```
SELECT COUNT(DISTINCT no)
                  AS active_accounts,
       COUNT(no) AS no_movements
FROM   movement
```

active_accounts	no_movements
5	9

SQL OLAP features: Aggregate operators

Use GROUP BY on all non aggregated columns

```
SELECT no,
       SUM(amount) AS balance,
       COUNT(amount) AS trans
       FROM movement
       GROUP BY no
```

no	balance	trans
101	5230.00	2
103	145.50	1
119	5600.00	1
107	245.56	2
100	2086.78	3

Choose bag or set semantics for COUNT

```
SELECT COUNT(DISTINCT no)
                    AS active_accounts,
       COUNT(no) AS no_movements
FROM   movement
```

active_accounts	no_movements
5	9

NULL attributes don't count!

```
SELECT COUNT(rate) AS no_rates
FROM   account
```

no_rates
2

Quiz 11.2: GROUP BY over NULL values (1)

movement

mid	no	amount	tdate
0999	119	45.00	NULL
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1006	100	10.23	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999
1010	100	NULL	20/1/1999
1011	NULL	NULL	20/1/1999
1012	NULL	600.00	20/1/1999
1013	NULL	-46.00	20/1/1999

account

no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
SELECT  movement.no,
        COUNT(movement.amount) AS no_trans,
        MIN(movement.amount) AS min_value
FROM    movement NATURAL JOIN account
GROUP BY movement.no
```

What is the result of the above query?

A

no	no_trans	min_value
101	2	1230.00
100	3	NULL
119	1	45.00

B

no	no_trans	min_value
101	2	1230.00
100	4	-223.45
119	2	45.00

C

no	no_trans	min_value
101	2	1230.00
100	4	NULL
119	2	45.00

D

no	no_trans	min_value
101	2	1230.00
100	3	-223.45
119	2	45.00

Quiz 11.3: GROUP BY over NULL values (2)

movement

mid	no	amount	tdate
0999	119	45.00	NULL
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1006	100	10.23	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999
1010	100	NULL	20/1/1999
1011	NULL	NULL	20/1/1999
1012	NULL	600.00	20/1/1999
1013	NULL	-46.00	20/1/1999

account

no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
SELECT  movement.no,
        SUM(movement.amount)  AS  balance
FROM    movement
GROUP  BY  movement.no
```

What is the result of the above query?

A

no	balance
NULL	NULL
NULL	600.00
NULL	-46.00
119	5645.00
101	5230.00
100	2086.78

B

no	balance
NULL	600.00
NULL	-46.00
119	5645.00
101	5230.00
100	2086.78

C

no	balance
NULL	554.00
119	5645.00
101	5230.00
100	2086.78

D

no	balance
119	5645.00
101	5230.00
100	2086.78

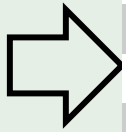
Selecting results from aggregates: HAVING

GROUP BY in the RA

- An extension to the RA includes a group by operator
- In SQL, the GROUP BY operator is applied *outside* the $\sigma_P(\ldots \times \ldots)$
- To execute a σ_P *outside* the GROUP BY, you must place the predicates P in a HAVING clause

Transaction analysis of bank_branch

```
SELECT    no,
          SUM(amount) AS balance,
          COUNT(amount) AS no_trans
FROM      movement
GROUP BY  no
```



no	balance	no_trans
100	2086.78	3
101	5230.00	2
103	145.50	1
107	245.56	2
119	5600.00	1

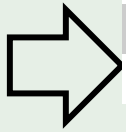
Selecting results from aggregates: HAVING

GROUP BY in the RA

- An extension to the RA includes a group by operator
- In SQL, the GROUP BY operator is applied *outside* the $\sigma_P(\ldots \times \ldots)$
- To execute a σ_P *outside* the GROUP BY, you must place the predicates P in a HAVING clause

Transaction analysis of bank_branch

```
SELECT   no,
         SUM(amount) AS balance,
         COUNT(amount) AS no_trans
FROM     movement
GROUP BY no
HAVING   SUM(amount)>2000
```



no	balance	no_trans
100	2086.78	3
101	5230.00	2
119	5600.00	1

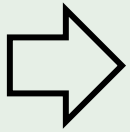
Selecting results from aggregates: HAVING

GROUP BY in the RA

- An extension to the RA includes a group by operator
- In SQL, the GROUP BY operator is applied *outside* the $\sigma_P(\ldots \times \ldots)$
- To execute a σ_P *outside* the GROUP BY, you must place the predicates P in a HAVING clause

Transaction analysis of bank_branch

```
SELECT    no,
          SUM(amount) AS balance,
          COUNT(amount) AS no_trans
FROM      movement
GROUP BY  no
HAVING    balance >2000
```



column balance does not exist

Ordering of SQL clauses

- HAVING is executed after GROUP BY, but before SELECT
- Can be used to avoid divide by zero errors

```
SELECT    no,
          MAX(amount)/MIN(amount) AS variance_ratio
FROM      movement
GROUP BY  movement.no
HAVING    MIN(amount)<>0
```

Quiz 11.4: HAVING

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account

no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
SELECT   account.no,
         account.cname,
         SUM(movement.amount) AS balance
FROM     account NATURAL JOIN movement
WHERE    movement.amount>200
GROUP BY account.no,
         account.cname
HAVING   COUNT(movement.no)>1
AND      SUM(movement.amount)>1000
```

What is the result of the above query?

A

no	cname	balance
101	McBrien, P.	5230.00

B

no	cname	balance
101	McBrien, P.	5230.00
119	Poulovassilis, A.	5600.00

C

no	cname	balance
100	McBrien, P.	2086.78
101	McBrien, P.	5230.00

D

no	cname	balance
100	McBrien, P.	2086.78
101	McBrien, P.	5230.00
119	Poulovassilis, A.	5600.00

Quiz 11.5: Execution order of SQL clauses

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

What order are the SQL clauses executed in?

A

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

B

FROM
WHERE
SELECT
GROUP BY
HAVING
ORDER BY

C

FROM
WHERE
GROUP BY
HAVING
SELECT
ORDER BY

D

ORDER BY
HAVING
GROUP BY
WHERE
FROM
SELECT

Relationally Complete SQL

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- relational completeness in SQL means being able to fully support the RA in SQL
- the five primitive operators of the RA can be fully supported by SQL
- SQL Aggregates require **relationally complete SQL**: allow SELECT statements in FROM clause

```
SELECT    no,
          SUM(amount) AS balance,
          ROUND(100*SUM(amount)/total,1) AS pc
FROM      movement,
          (SELECT SUM(amount) AS total
           FROM   movement) AS total_balance
GROUP BY  no,
          total
ORDER BY  no
```

no	balance	pc
100	2086.78	15.7
101	5230.00	39.3
103	145.50	1.1
107	245.56	1.8
119	5600.00	42.1

SQL OLAP features: Windows and PARTITION BY

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

.
.
.
OVER (PARTITION BY no)
FROM movement
.
.
.

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1002	100	-223.45	8/1/1999
1006	100	10.23	15/1/1999
1001	101	4000.00	5/1/1999
1008	101	1230.00	15/1/1999
1004	107	-100.00	11/1/1999
1007	107	345.56	15/1/1999
1005	103	145.50	12/1/1999
1009	119	5600.00	18/1/1999

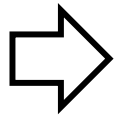
SQL OLAP features: Windows and PARTITION BY

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

.
.
.
OVER (PARTITION BY no)
FROM movement
.
.
.

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1002	100	-223.45	8/1/1999
1006	100	10.23	15/1/1999
1001	101	4000.00	5/1/1999
1008	101	1230.00	15/1/1999
1004	107	-100.00	11/1/1999
1007	107	345.56	15/1/1999
1005	103	145.50	12/1/1999
1009	119	5600.00	18/1/1999

```
SELECT  mid,
        no,
        amount,
        SUM(amount) OVER (PARTITION BY no) AS balance
FROM    movement
```



mid	no	amount	balance
1000	100	2300.00	2086.78
1002	100	-223.45	2086.78
1006	100	10.23	2086.78
1001	101	4000.00	5230.00
1008	101	1230.00	5230.00
1004	107	-100.00	245.56
1007	107	345.56	245.56
1005	103	145.50	145.50
1009	119	5600.00	5600.00

PARTITION BY

- One row output per input row
- Aggregates apply to window defined by PARTITION BY

Window Functions Replacing Subquery in FROM clause

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

- some cases using subqueries in FROM clause may be replaced by window queries

```
SELECT    DISTINCT no,
          SUM(amount) OVER (PARTITION BY no) AS balance,
          ROUND(100*SUM(amount) OVER (PARTITION BY no)/
              SUM(amount) OVER (),1) AS pc
FROM      movement
ORDER BY  no
```

no	balance	pc
100	2086.78	15.7
101	5230.00	39.3
103	145.50	1.1
107	245.56	1.8
119	5600.00	42.1

SQL OLAP features: Window Functions

OVER

The SQL OVER operator produces a **window** which may have an ORDER BY applied to each window.
By default (without PARTITION BY), entire dataset in window

SQL functions dealing with ORDER BY B	
function	semantics
LEAD(A)	The next value of column A in the window when data ordered by column B
LAG(A)	The previous value of column A in the window when data ordered by column B
RANK()	The rank position of the row in the window when data ordered by column B, tied value receive same rank, and next rank value skipped
DENSE_RANK()	The dense rank position of the row in the window when data order by column B (like rank, but no numbers are skipped)
ROW_NUMBER()	The position of row when ordered by B, numbering rows sequentially (with tied rows given different numbers)
FIRST_VALUE(A)	The first value of A in the window when data ordered by column B
LAST_VALUE(A)	The first value of A in the window when data ordered by column B

SQL OLAP features: Ranking Rows

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
SELECT  mid,
        tdate,
        amount,
        RANK() OVER
          (ORDER BY tdate),
        DENSE_RANK() OVER
          (ORDER BY tdate),
        ROW_NUMBER() OVER
          (ORDER BY tdate)
FROM    movement
```

mid	tdate	amount	rank	dense_rank	row_number
1000	1999-01-05	2300.00	1	1	1
1001	1999-01-05	4000.00	1	1	2
1002	1999-01-08	-223.45	3	2	3
1004	1999-01-11	-100.00	4	3	4
1005	1999-01-12	145.50	5	4	5
1006	1999-01-15	10.23	6	5	6
1007	1999-01-15	345.56	6	5	7
1008	1999-01-15	1230.00	6	5	8
1009	1999-01-18	5600.00	9	6	9

SQL OLAP features: Looking at rows in the window

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
SELECT  mid,
        amount,
        LEAD(amount) OVER
          (ORDER BY amount DESC) AS next,
        LAG(amount) OVER
          (ORDER BY amount DESC) AS previous,
        ROUND(AVG(amount) OVER
          (ORDER BY amount ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING),2) AS trend
FROM    movement
```

mid	amount	next	previous	trend
1002	-223.45	NULL	-100.00	-161.73
1004	-100.00	-223.45	10.23	-104.41
1006	10.23	-100.00	145.50	18.58
1005	145.50	10.23	345.56	167.10
1007	345.56	145.50	1230.00	573.69
1008	1230.00	345.56	2300.00	1291.85
1000	2300.00	1230.00	4000.00	2510.00
1001	4000.00	2300.00	5600.00	3966.67
1009	5600.00	4000.00	NULL	4800.00

OLAP: Pivot

Pivoting tabular data

- for presentation purposes, useful to change layout of table
- information spread over rows is instead spread over columns
- we can pivot the value of one or more columns on the values of one or more other columns

```
SELECT    branch.sortcode,
          branch.bname,
          account.type,
          COUNT(no) AS qty
FROM      account JOIN branch
          ON account.sortcode=
                branch.sortcode
GROUP BY branch.sortcode,
          branch.bname,
          account.type
ORDER BY branch.sortcode,
          branch.bname
```

sortcode	bname	type	qty
34	Goodge St	current	1
56	Wimbledon	current	2
56	Wimbledon	deposit	1
67	Strand	current	1
67	Strand	deposit	1

OLAP: Pivot

Pivoting tabular data

- for presentation purposes, useful to change layout of table
- information spread over rows is instead spread over columns
- we can pivot the value of one or more columns on the values of one or more other columns

```
SELECT    branch.sortcode,
          branch.bname,
          account.type,
          COUNT(no) AS qty
FROM      account JOIN branch
          ON account.sortcode=
                branch.sortcode
GROUP BY branch.sortcode,
          branch.bname,
          account.type
ORDER BY branch.sortcode,
          branch.bname
```

sortcode	bname	type	qty
34	Goodge St	current	1
56	Wimbledon	current	2
56	Wimbledon	deposit	1
67	Strand	current	1
67	Strand	deposit	1

pivot qty on type

sortcode	bname	type	qty
34	Goodge St	current	1
56	Wimbledon	current	2
56	Wimbledon	deposit	1
67	Strand	current	1
67	Strand	deposit	1

OLAP: Pivot

Pivoting tabular data

- for presentation purposes, useful to change layout of table
- information spread over rows is instead spread over columns
- we can pivot the value of one or more columns on the values of one or more other columns

```
SELECT    branch.sortcode,
          branch.bname,
          account.type,
          COUNT(no) AS qty
FROM      account JOIN branch
          ON account.sortcode=
                branch.sortcode
GROUP BY branch.sortcode,
          branch.bname,
          account.type
ORDER BY branch.sortcode,
          branch.bname
```

sortcode	bname	type	qty
34	Goodge St	current	1
56	Wimbledon	current	2
56	Wimbledon	deposit	1
67	Strand	current	1
67	Strand	deposit	1

pivot qty on type

sortcode	bname	type	qty
34	Goodge St	current	1
56	Wimbledon	current	2
56	Wimbledon	deposit	1
67	Strand	current	1
67	Strand	deposit	1

sortcode	bname	current	deposit
34	Goodge St	1	
56	Wimbledon	2	1
67	Strand	1	1

SQL OLAP: Pivot using FILTER statements

```
SELECT    branch.sortcode,
          branch.bname,
          COUNT(no) FILTER (WHERE type='current') AS current,
          COUNT(no) FILTER (WHERE type='deposit') AS deposit,
          COUNT(no) FILTER (WHERE (type IN ('current','deposit')) IS NOT TRUE) AS other
FROM      account JOIN branch ON account.sortcode=branch.sortcode
GROUP BY branch.sortcode, branch.bname
ORDER BY branch.sortcode, branch.bname
```

sortcode	bname	current	deposit	other
34	Goodge St	1	0	0
56	Wimbledon	2	1	0
67	Strand	1	1	0

- use FILTER statements to filter values from column being pivoted
- generally one FILTER (WHERE A=v) for each value v that appears in column A
- can have a default case for unexpected values

Worksheet: OLAP Queries in SQL

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

movement.no $\overset{fk}{\Rightarrow}$ account.no

Worksheet: OLAP Queries Questions 3 & 4

3 Write an SQL query returning the scheme (cname,current_balance,deposit_balance) that lists one row for each customer (*i.e.* each distinct cname), with a column for the net balance of all current accounts held by the customer, and a column for the net balance of all deposit accounts held by the customer.

4 Write an SQL query returning the scheme (no,cname,type,pc_cust_funds,pc_type_funds) that lists one row for each account, and for each account, lists the no, cname and type of the account, and in pc_cust_funds the percentage of the customer funds held in the account, and in pc_type_funds the percentage of the total funds in this particular type of account. For the current data this should result in:

no	cname	type	pc_cust_funds	pc_type_funds
100	McBrien, P.	current	28.52	84.22
101	McBrien, P.	deposit	71.48	48.29
103	Boyd, M.	current	100.00	5.87
107	Poulovassilis, A.	current	4.20	9.91
119	Poulovassilis, A.	deposit	95.80	51.71
125	Bailey, J.	current	NULL	0.00

Worksheet: OLAP Queries in SQL (3)

```
SELECT    account.cname,
          COALESCE(SUM(amount) FILTER (WHERE type='current'),0.0)
              AS current_balance,
          COALESCE(SUM(amount) FILTER (WHERE type='deposit'),0.0)
              AS deposit_balance
FROM      account LEFT JOIN movement ON account.no=movement.no
GROUP BY  account.cname
```

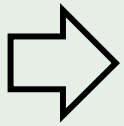
Worksheet: OLAP Queries in SQL (4)

```
SELECT DISTINCT account.no,
       account.cname,
       account.type,
       ROUND(COALESCE(100.0*SUM(movement.amount) OVER (PARTITION BY account.no),0.0)/
                SUM(movement.amount) OVER (PARTITION BY account.cname),2)
       AS pc_cust_funds,
       ROUND(COALESCE(100.0*SUM(movement.amount) OVER (PARTITION BY account.no),0.0)/
                SUM(movement.amount) OVER (PARTITION BY account.type),2
       AS pc_type_funds
FROM   account LEFT JOIN movement ON account.no=movement.no
```

SQL OLAP: Un-pivot using UNION statements

Un-pivot the account table to triple format

```
SELECT no,
       'cname' AS col,
       cname AS value
FROM   account
UNION
SELECT no,
       'type',
       type
FROM   account
UNION
SELECT no,
       'rate',
       CAST(rate AS VARCHAR)
FROM   account
WHERE  rate IS NOT NULL
UNION
SELECT no,
       'sortcode',
       CAST(sortcode AS VARCHAR)
FROM   account
```



no	col	value
100	cname	McBrien, P.
100	sortcode	67
100	type	current
101	cname	McBrien, P.
101	rate	5.25
101	sortcode	67
101	type	deposit
103	cname	Boyd, M.
103	sortcode	34
103	type	current
107	cname	Poulovassilis, A.
107	sortcode	56
107	type	current
119	cname	Poulovassilis, A.
119	rate	5.50
119	sortcode	56
119	type	deposit
125	cname	Bailey, J.
125	sortcode	56
125	type	current

Triple Stores

Critism of RDBMS: row format

Some say that

- Sometimes get tables with many (hundreds) of columns, many of which are null
- Adding/removing a column (with ALTER TABLE) is a relatively slow operation, and can break (badly written) queries.

Solutions

Triple format involves presenting data as key, property, value **triples**

- Adopt triple format in RDBMS: performance can be poor due to multiple joins
- Adopt **column store** RDBMS such as SAP **Hana**
- Adopt a Graph/RDF model database such as **StarDog** or **GraphDB**
- ...