SQL: A Language for Database Applications

P.J. McBrien

Imperial College London

SQL WHERE expressions in more detail

Testing Strings against a Pattern

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

List customers whose first initial is P, and have one more initial

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

SQL WHERE expressions in more detail

Testing Strings against a Pattern

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

List customers whose first initial is between A and L

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

Processing the result of project

Modifications to data

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

- Many functions proposed in ANSI SQL, *e.g.*
  - ABS(number) returns the absolute value of any number
  - ROUND(value,dp) rounds a numeric value to dp decimal places
  - UPPER(str) returns the string converted to all capitals
- Tends to be an aspect of SQL implementations that is not ANSI SQL compliant, *e.g.*
  - Postgres: LENGTH(object) returns the length of any object (including strings)
  - TrasnsactSQL: LEN(str) returns the length of any string type column

Processing the result of project

Modifications to data

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

- Many functions proposed in ANSI SQL, *e.g.*
  - ABS(number) returns the absolute value of any number
  - ROUND(value,dp) rounds a numeric value to dp decimal places
  - UPPER(str) returns the string converted to all capitals
- Tends to be an aspect of SQL implementations that is not ANSI SQL compliant, *e.g.*
  - Postgres: LENGTH(object) returns the length of any object (including strings)
  - TrasnsactSQL: LEN(str) returns the length of any string type column

Display accounts with just surnames and rounded rates

PostgresSQL

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

Processing the result of project

Modifications to data

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

- Many functions proposed in ANSI SQL, *e.g.*
  - ABS(number) returns the absolute value of any number
  - ROUND(value,dp) rounds a numeric value to dp decimal places
  - UPPER(str) returns the string converted to all capitals
- Tends to be an aspect of SQL implementations that is not ANSI SQL compliant, *e.g.*
  - Postgres: LENGTH(object) returns the length of any object (including strings)
  - TrasnsactSQL: LEN(str) returns the length of any string type column

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
```

Quiz 1: 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.	London SW19
Poulovassilis, A.	13 Haydons Rd

C

cname	area
Poulovassilis, A.	London SW19

D

cname	area
Poulovassilis, A.	13 Haydons Rd

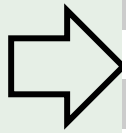
Processing the result of project: CASE statements

CASE statements

A CASE statement may be put in the SELECT clause to process the values being returned.

Display account interest rates

```
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
```

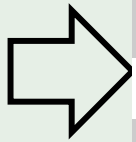


no	rate	interest_class
100	0.00	zero rate
101	5.25	low rate
103	0.00	zero rate
107	0.00	zero rate
119	5.50	high rate
125	0.00	zero rate

Need for yet another type of Join?

Listing of movements 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

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

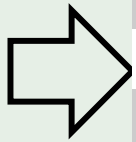
An outer join $R \overset{\text{O}}{\bowtie} S$ returns every row in R, even if no rows in S match, and also returns every row in S even if no row in R matches.

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

Need for yet another type of Join?

Listing any movements for all customers

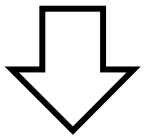
```
SELECT  cname,
        mid
FROM    account NATURAL LEFT JOIN
        movement
```



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

RA equivalent of LEFT JOIN

```
SELECT A1, ..., An
FROM   R1 LEFT JOIN R2 ON O1 AND ... AND Oi
WHERE  P1
AND    ...
AND    Pk
```



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

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

Quiz 2: SQL LEFT JOIN ... ON (1)

```
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

Quiz 3: SQL LEFT JOIN ... ON (2)

```
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

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: 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	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

. . .

GROUP BY

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

SQL OLAP features: GROUP BY

movement

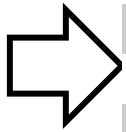
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

```
.
.
.
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

```
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

GROUP BY

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

Quiz 4: 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
```

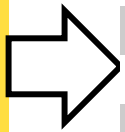
D

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

SQL OLAP features: Aggregate operators

- Normally use GROUP BY on all non aggregated attributes:

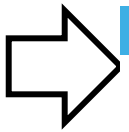
```
SELECT no,
       SUM(amount) AS total,
       COUNT(amount) AS trans
FROM   movement
GROUP BY no
```



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

- Don't forget to choose bag or set semantics for COUNT

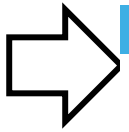
```
SELECT COUNT(DISTINCT no) AS active_accounts
FROM   movement
```



active_accounts
5

- Null attributes don't count!

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



no_rates
2

Quiz 5: 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
119	2	45.00
101	2	1230.00
107	1	-100.00
100	3	-223.45
103	1	145.50

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 6: 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

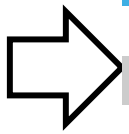
no	balance
119	5645.00
101	5230.00
100	2086.78

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 $\pi\,\sigma_(\ldots\times\ldots)$
- To execute a σ_P *outside* the group by, you must place the predicates P in a HAVING clause

```
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

Quiz 7: 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

SQL OLAP features: PARTITION

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

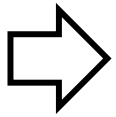
SQL OLAP features: PARTITION

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
```



query			
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	145.50
1007	107	345.56	245.56
1005	103	145.50	245.56
1009	119	5600.00	5600.00

PARTITION BY

- One row output per input row
- Aggregates apply to partition

Relationally Complete SQL

Relational Completeness

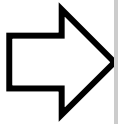
- Relational completeness in SQL means being able to fully support the RA in SQL
- 'pure' RA can be fully supported by SQL
- Aggregates require 'relationally complete' SQL
  - Temporary tables
  - SELECT statements in FROM clause

Relationally Complete SQL

Relational Completeness

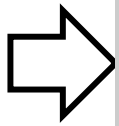
- Relational completeness in SQL means being able to fully support the RA in SQL
- 'pure' RA can be fully supported by SQL
- Aggregates require 'relationally complete' SQL
  - Temporary tables
  - SELECT statements in FROM clause

```
SELECT SUM(amount) AS total
INTO   #total_balance
FROM   movement
```



#total_balance
total
13307.84

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



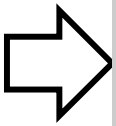
no	balance	pc
100	2086.78	15.7
101	5230.00	39.3
103	145.50	1.1
107	245.56	1.8
119	5600.00	42.1

Relationally Complete SQL

Relational Completeness

- Relational completeness in SQL means being able to fully support the RA in SQL
- 'pure' RA can be fully supported by SQL
- Aggregates require 'relationally complete' SQL
  - Temporary tables
  - SELECT statements in FROM clause

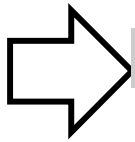
```
SELECT   movement.no,
         SUM(movement.amount) AS balance,
         ROUND(100*SUM(movement.amount)/total_balance.total,1) AS pc
FROM     movement,
         (SELECT SUM(amount) AS total FROM movement) total_balance
GROUP BY movement.no, total_balance.total
ORDER BY movement.no
```



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: Ordering Rows

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

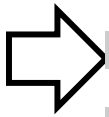


order

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: Ranking Rows

```
SELECT mid,
       tdate,
       amount,
       RANK() OVER (ORDER BY amount DESC) AS rank
FROM   movement;
```

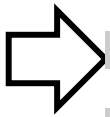


rank

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

SQL OLAP features: Ranking Rows Without RANK()

```
SELECT   movement.mid,
         movement.tdate,
         movement.amount,
         COUNT(higher.mid) AS rank
FROM     movement, movement higher
WHERE    movement.amount<higher.amount
OR       movement.mid=higher.mid
GROUP BY movement.mid,
         movement.tdate,
         movement.amount
ORDER BY COUNT(higher.mid)
```

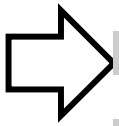


rank			
mid	tdate	amount	rank
1009	1999-01-18	5600.00	1
1001	1999-01-05	4000.00	2
1000	1999-01-05	2300.00	3
1008	1999-01-15	1230.00	4
1007	1999-01-15	345.56	5
1005	1999-01-12	145.50	6
1006	1999-01-15	10.23	7
1004	1999-01-11	-100.00	8
1002	1999-01-08	-223.45	9

OLAP: Pivot

- for presentation purposes, useful to change layout of table
- information spread over rows is instead spread over columns

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



branch account types			
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

SQL OLAP: Pivot using CASE statements

```
SELECT    branch.sortcode, branch.bname,
          COUNT(CASE WHEN type='current' THEN no ELSE null END) AS current,
          COUNT(CASE WHEN type='deposit' THEN no ELSE null END) AS deposit,
          COUNT(CASE WHEN type NOT IN ('current','deposit') THEN no
          ELSE null END) AS other
FROM      account, branch
WHERE     account.sortcode=branch.sortcode
GROUP BY  branch.sortcode, branch.bname
ORDER BY  branch.sortcode, branch.bname
```

branch account types pivot

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

- use CASE statements to filter values from column being pivoted
- one case for each value
- wise to have a default case

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 $\longrightarrow$ account.no

SQL Functions

FUNCTION

- Most SQL implementations support some variant of ANSI SQL FUNCTION
- Details vary ...

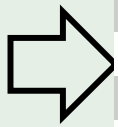
TransactSQL function to return cnames reformatted

```
CREATE FUNCTION cname_to_initial_first(@cname VARCHAR(20))
       RETURNS VARCHAR(20) AS
BEGIN
   DECLARE @ifcname VARCHAR(20)

   SELECT @ifcname=
            SUBSTRING(@cname,CHARINDEX(' ,' ,@cname)+2,LEN(@cname))+
            SUBSTRING(@cname,1,CHARINDEX(' ,' ,@cname)-1)

   RETURN @ifcname
END


SELECT no,
       dbo.cname_to_initial_first(
            account.cname) AS cname
FROM   account
```



no	cname
100	P.McBrien
101	P.McBrien
103	M.Boyd
107	A.Poulovassilis
119	A.Poulovassilis
125	J.Bailey

SQL Procedures

PROCEDURE

- No specific PROCEDURE construct in Postgres
- TransactSQL supports PROCEDURE definition, and generally refers to them a **stored procedure**

TransactSQL Procedure to move cash between branches

```
CREATE PROCEDURE move_cash
(   @from_branch INTEGER,
    @to_branch INTEGER,
    @total DECIMAL(10,2)
) AS
BEGIN
    UPDATE branch
    SET cash=cash-@total
    WHERE sortcode=@from_branch

    UPDATE branch
    SET cash=cash+@total
    WHERE sortcode=@to_branch
END
```

SQL Views

PROCEDURE

- Basic ANSI SQL CREATE VIEW well supported accross platforms
- Variations in details

Views defining current account and depost account

```
CREATE VIEW current_account AS
SELECT no,
       cname,
       sortcode
FROM   account
WHERE  type='current'

CREATE VIEW deposit_account AS
SELECT no,
       cname,
       rate,
       sortcode
FROM   account
WHERE  type='deposit'
```

Quiz 8: Updates to SQL Views

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

```
CREATE VIEW current_account AS
SELECT   no,
         cname,
         sortcode
FROM     account
WHERE    type='current'
```

Which SQL view update does not work?

A

```
UPDATE  current_account
SET     sortcode=56
WHERE   no=125
```

B

```
UPDATE  current_account
SET     sortcode=56
```

C

```
DELETE FROM current_account
WHERE   no=125
```

D

```
INSERT INTO current_account
VALUES (129,'Jones, F.',34)
```

SQL View Updates

SQL restricts View updates to view definitions

- on just one table
- containing no aggregates
- no computed columns
- for INSERT: all non-nullable columns without defaults being included in view

Ambiguous view updates

```
CREATE VIEW active_account AS
SELECT no,
       cname,
       sortcode
FROM   account JOIN movement

DELETE FROM active_account
WHERE  no=100
```

The DELETE could be fulfilled by either (a) deleting account 100 or (b) deleting all movements for account 100

Active Databases: Cascades

- Can cascade updates on one column to other columns that reference that column

```
CONSTRAINT account_fk FOREIGN KEY (sortcode)
REFERENCES branch ON UPDATE CASCADE
```

- Can cascade deletes on one column to other columns that reference that column

```
CONSTRAINT movement_fk FOREIGN KEY (no)
REFERENCES account ON DELETE CASCADE
```