

SQL for Data Science: Using Analytical Functions

Doing more by writing less Python code
and instead use SQL Analytical Functions

Brendan Tierney



@brendantierney

Brendan Tierney



ORACLE
ACE Director

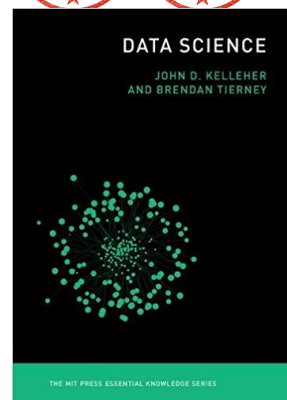


Oracle
Groundbreaker
Ambassador

- Data Warehousing since 1997
- Data Mining since 1998
- Analytics (Data Science) since 1993
- Oracle since 1992

t. @brendantierney

w. www.oralytics.com



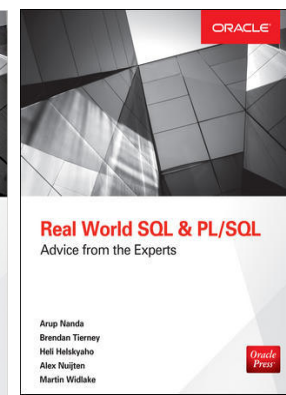
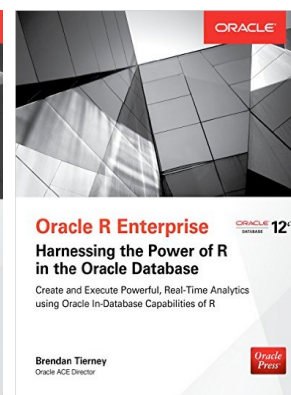
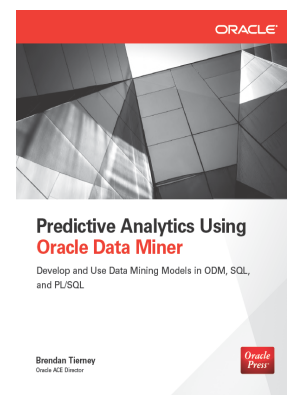
Available in
Chinese, Korean,
Japanese,
Russian, Turkish &
Spanish



97 Things About
Ethics Everyone
in Data Science
Should Know



Edited by Bill Franks

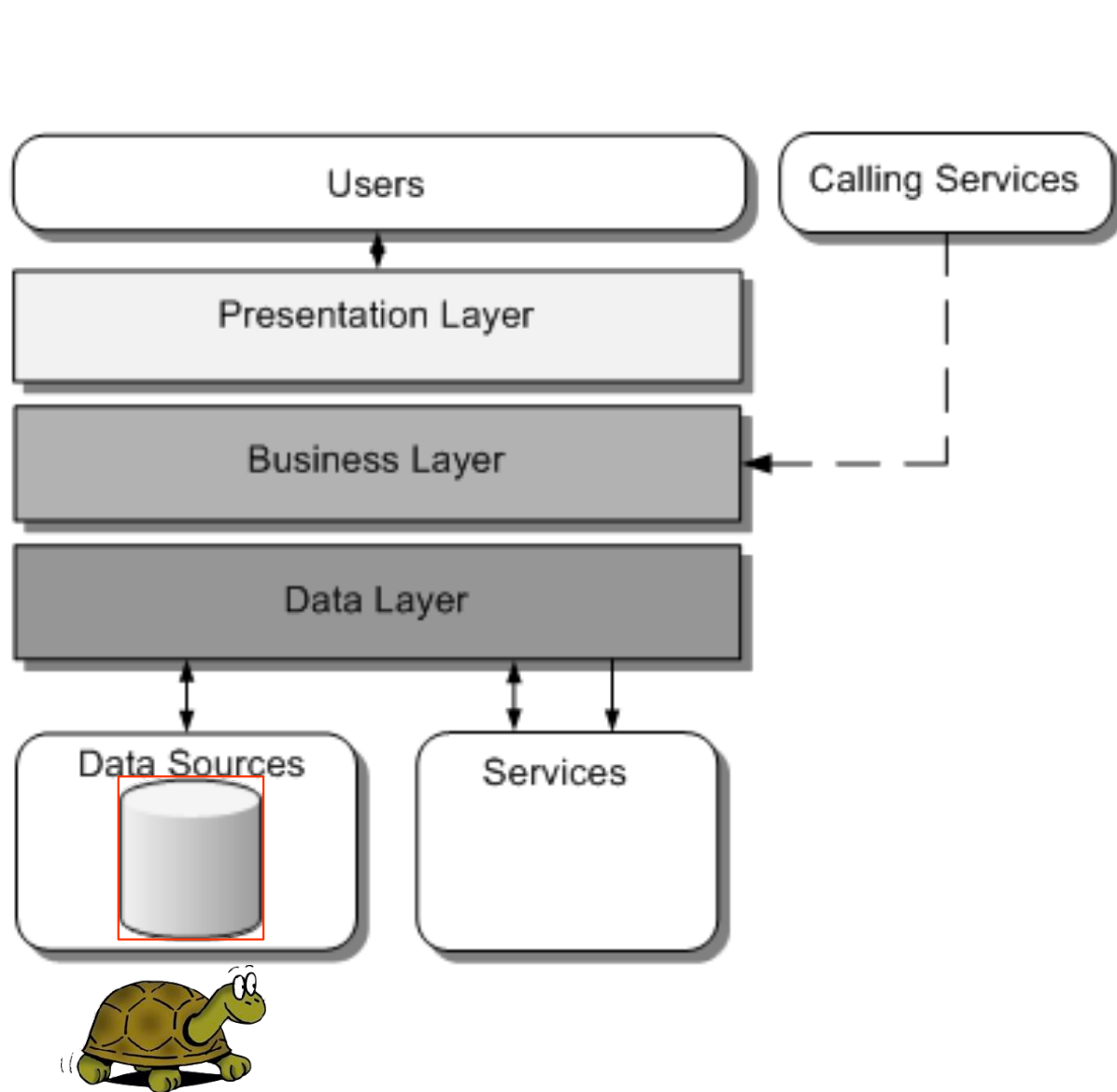




Why







SELECT *
FROM ... ;

1

2

C
R
U
D

→ INSERT / Execute PL/SQL

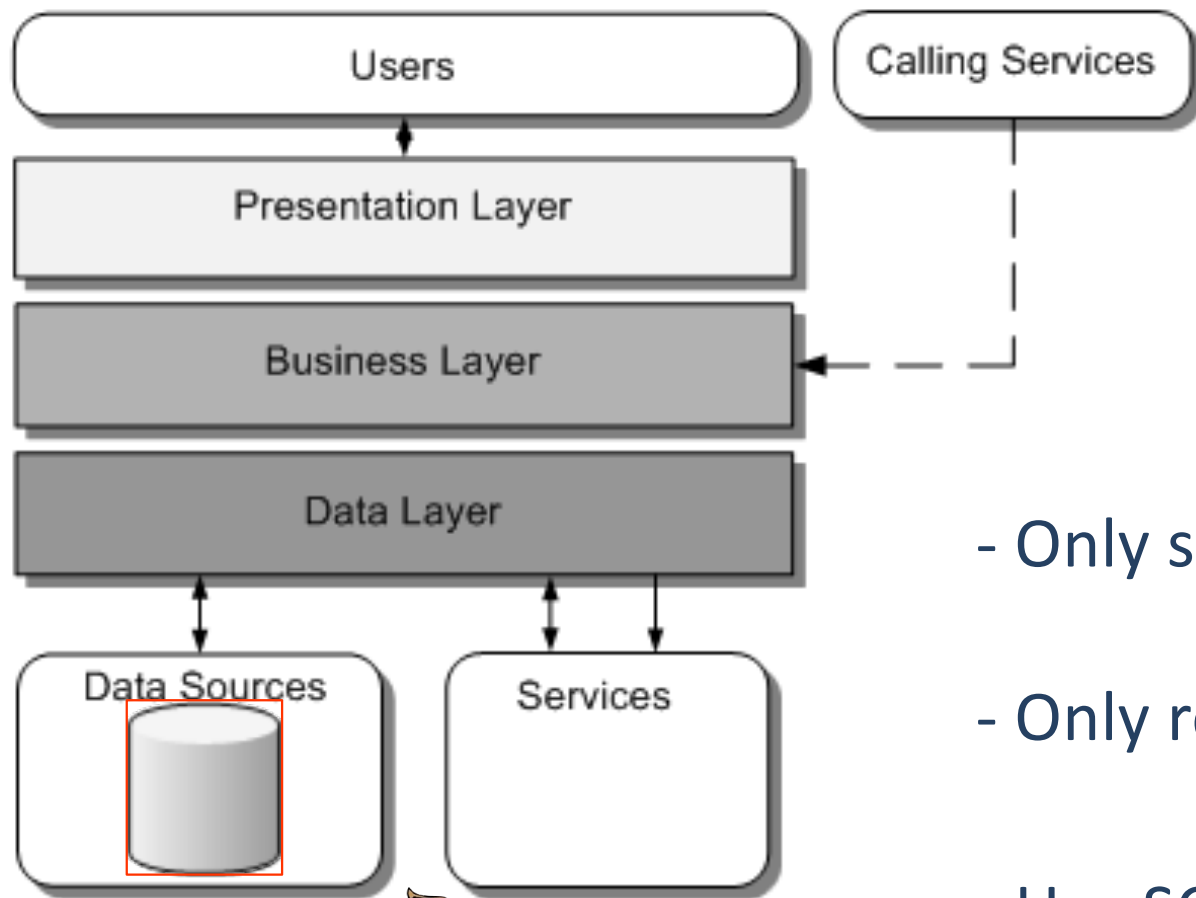
→ SELECT

→ UPDATE

→ DELETE

WHAT IF....?





SELECT *
FROM ... ;

1

2

- Only select the Attributes we need
- Only return the Records we need
- Use SQL to Analyse our data





Year	Name	Alias	Example of new features or enhancements
1986	SQL-86	SQL-87	This is the first version standardized by ANSI.
1989	SQL-89		This version includes minor revisions that add integrity constraints.
1992	SQL-92	SQL2	This version includes major revisions on the SQL language. Many database systems use this standard for their specification language.
1999	SQL:1999	SQL3	This version introduces many new features, such as regex matching, triggers, some object-oriented features, and OLAP capability. SQL:1999 also adds BOOLEAN data type, but many commercial database servers do not support it as a column type. In the meanwhile, SQL:1999 is deprecated.
2003	SQL:2003		This version makes minor modifications to all parts of SQL:1999. This SQL:2003 version also introduces new features such as: - Window functions, which are very useful for data analysis - Columns with auto-generated value and identity specification.
2006	SQL:2006		This version defines ways of importing, storing, and manipulating XML data. Additionally, it lets applications use XQuery to query data in XML format.
2008	SQL:2008		This version adds features such as TRUNCATE TABLE statement, FETCH clause, and INSTEAD OF triggers.
2011	SQL:2011		This version adds enhancements for window functions and FETCH clause.
2016	SQL:2016		This version adds various functions to deal with JSON data.
2019	SQL:2019		This version specifies a multidimensional arrays data type (MDarray).

SQL 2020+ : Graph Data & Query Language

ISO/IEC 9075 Database Language SQL

SQL-87 – Transactions, Create, Read, Update, Delete

SQL-89 – Referential Integrity

SQL-92 – Internationalization, etc.

SQL:1999 – User Defined Types

SQL:2003 – XML

SQL:2008 – Expansions and corrections

SQL:2011 – Temporal

SQL:2016 – JSON, Polymorphic Table functions

SQL:2019 – Multi-dimensional Arrays

```
# define the login details
p_username = "*****"
p_password = "*****"
p_host = "*****"
p_service = "*****"
p_port = "1521"

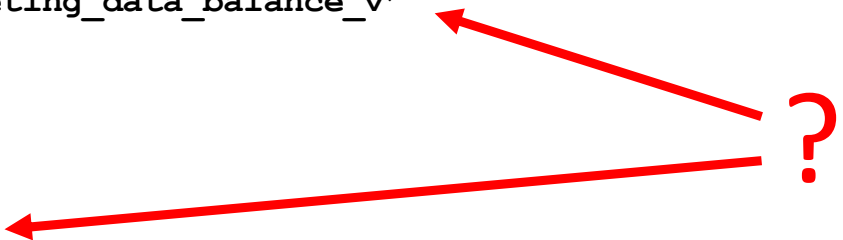
# create the connection
con = cx_Oracle.connect(user=p_username, password=p_password, dsn=p_host+"/"+p_service+": "+p_port)

sql = 'select * from banking_marketing_data_balance_v'

df2 = pd.read_sql(sql, con)

or

df = pd.read_sql_query(sql, con)
...
```

A large red question mark is positioned to the right of the code. Two red arrows originate from the question mark: one points to the SQL query string 'select * from banking_marketing_data_balance_v' and the other points to the pd.read_sql function call.

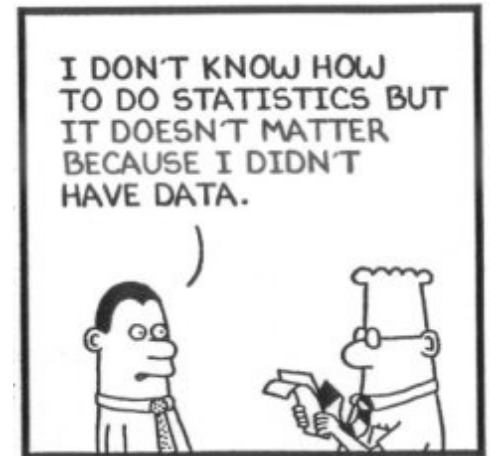
Stats and More Stats

COUNT SUM
MIN ? AVG
CORR MAX MODE

Stats and More Stats

- Correlations (Spearman's, Kendall)
- Linear Regression
- T-Test
- F-Test
- Hypothesis testing
- Anova
- Ranking
- Window Aggregate functions
- Lead / Lag
- Cross Tabulation
- PIVOT / UNPIVOT
- ...

300+ Statistical Functions



ABS	LENGTH2	REGR_AVGX
ACOS	LENGTH4	REGR_ACGY
Aggregate functions	LENGTHB	REGR_COUNT
Analytic functions	LENGTHC	REGR_INTERCEPT
Arithmetic operators	LN	REGR_R2
ASIN	LNNVL	REGR_SLOPE
ATAN	LOG	REGR_SXX
ATAN2	LOWER	REGR_SXY
AVG	LPAD	REGR_SYY
CAST	LTRIM	ROLLUP clause
Comparison functions	MAX	ROUND
CONCAT	MEDIAN	SAMPLE
CORR	MIN	SIN
CORR_K	MOD	SINH
CORR_S	MODEL clause	SQRT
COS	NTH_VALUE	STATS_BINOMIAL_TEST
COSH	Numeric Functions	STATS_CROSSTAB





There's Just Not
ENOUGH
Time!

A woman with short blonde hair, wearing a black long-sleeved top and a grey skirt, stands in a green field with her arms outstretched. In the background are large, rugged mountains with patches of snow under a clear blue sky. The text "A FEW OF MY FAVORITE THINGS" is overlaid on the right side of the image.

A FEW OF MY FAVORITE THINGS

- Analytic Functions
- Advanced Analytic Functions
- Window Functions



```

select deptno,
       name,
       job,
       salary
from employee
order by deptno;

```

DEPTNO	NAME	JOB	SALARY	AVG_SAL
-----	-----	-----	-----	-----
10	CLARK	MANAGER	2450	2073.21429
10	MILLER	CLERK	1300	2073.21429
10	KING	PRESIDENT	5000	2073.21429
20	FORD	ANALYST	3000	2073.21429
20	ADAMS	CLERK	1100	2073.21429
20	SCOTT	ANALYST	3000	2073.21429
20	JONES	MANAGER	2975	2073.21429
20	SMITH	CLERK	800	2073.21429
30	JAMES	CLERK	950	2073.21429
30	TURNER	SALESMAN	1500	2073.21429
30	MARTIN	SALESMAN	1250	2073.21429
30	WARD	SALESMAN	1250	2073.21429
30	ALLEN	SALESMAN	1600	2073.21429
30	BLAKE	MANAGER	2850	2073.21429


```

select deptno,
       name,
       job,
       salary,
       ? < some analytic function>
from employee
order by deptno;

```

```

select deptno,
       job,
       avg(salary)
from employee
group by deptno, job
order by deptno;

```

	DEPTNO	NAME	JOB	SALARY	AVG_SAL
	-----	-----	-----	-----	-----
	10	CLARK	MANAGER	2450	2073.21429
	10	MILLER	CLERK	1300	2073.21429
	10	KING	PRESIDENT	5000	2073.21429
	20	FORD	ANALYST	3000	2073.21429
	20	ADAMS	CLERK	1100	2073.21429
	20	SCOTT	ANALYST	3000	2073.21429
	10	CLERK	20 JONES 1300	2975	2073.21429
	10	MANAGER	20 SMITH 2450	800	2073.21429
	10	PRESIDENT	30 JAMES 5000	950	2073.21429
	20	ANALYST	30 TURNER 3000	1500	2073.21429
	20	CLERK	30 MARTIN 950	1250	2073.21429
	20	MANAGER	30 WARD 2975	1250	2073.21429
	30	CLERK	30 ALLEN 950	1600	2073.21429
	30	MANAGER	30 BLAKE 2850	2850	2073.21429
	30	SALESMAN	1400		

```

select deptno,
       name,
       job,
       salary,
       (select avg(salary) from employee) avg_sal
from employee
order by deptno;

```

BUT

DEPTNO	NAME	JOB	SALARY	AVG_SAL
10	CLARK	MANAGER	2450	2073.21429
10	MILLER	CLERK	1300	2073.21429
10	KING	PRESIDENT	5000	2073.21429
20	FORD	ANALYST	3000	2073.21429
20	ADAMS	CLERK	1100	2073.21429
20	SCOTT	ANALYST	3000	2073.21429
20	JONES	MANAGER	2975	2073.21429
20	SMITH	CLERK	800	2073.21429
30	JAMES	CLERK	950	2073.21429
30	TURNER	SALESMAN	1500	2073.21429
30	MARTIN	SALESMAN	1250	2073.21429
30	WARD	SALESMAN	1250	2073.21429
30	ALLEN	SALESMAN	1600	2073.21429
30	BLAKE	MANAGER	2850	2073.21429

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	5 (20)	00:00:01
1	SORT AGGREGATE		1	13		
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	13	2 (0)	00:00:01
3	SORT ORDER BY		1	39	5 (20)	00:00:01
4	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2 (0)	00:00:01

```

select deptno,
       name,
       job,
       salary,
       avg (salary) over () avg_sal
from employee
order by deptno;

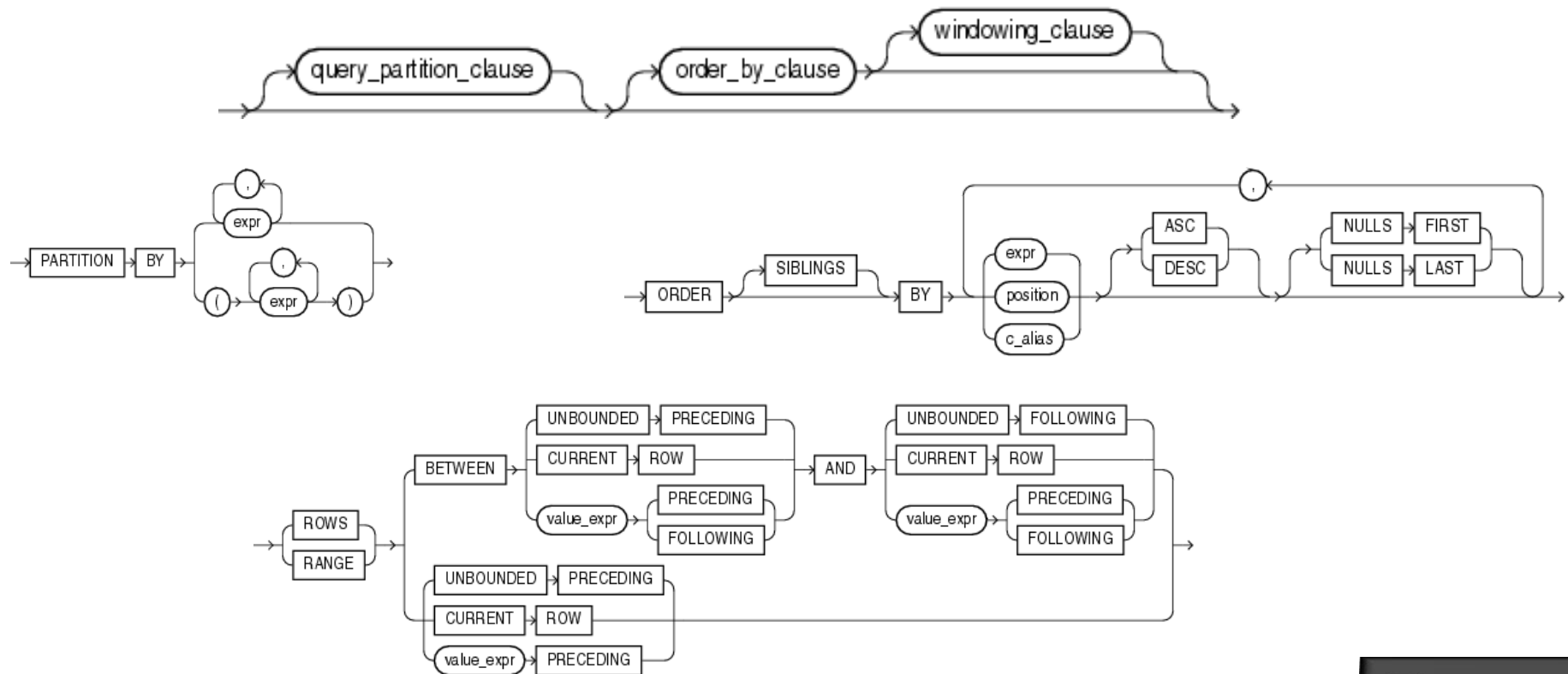
```



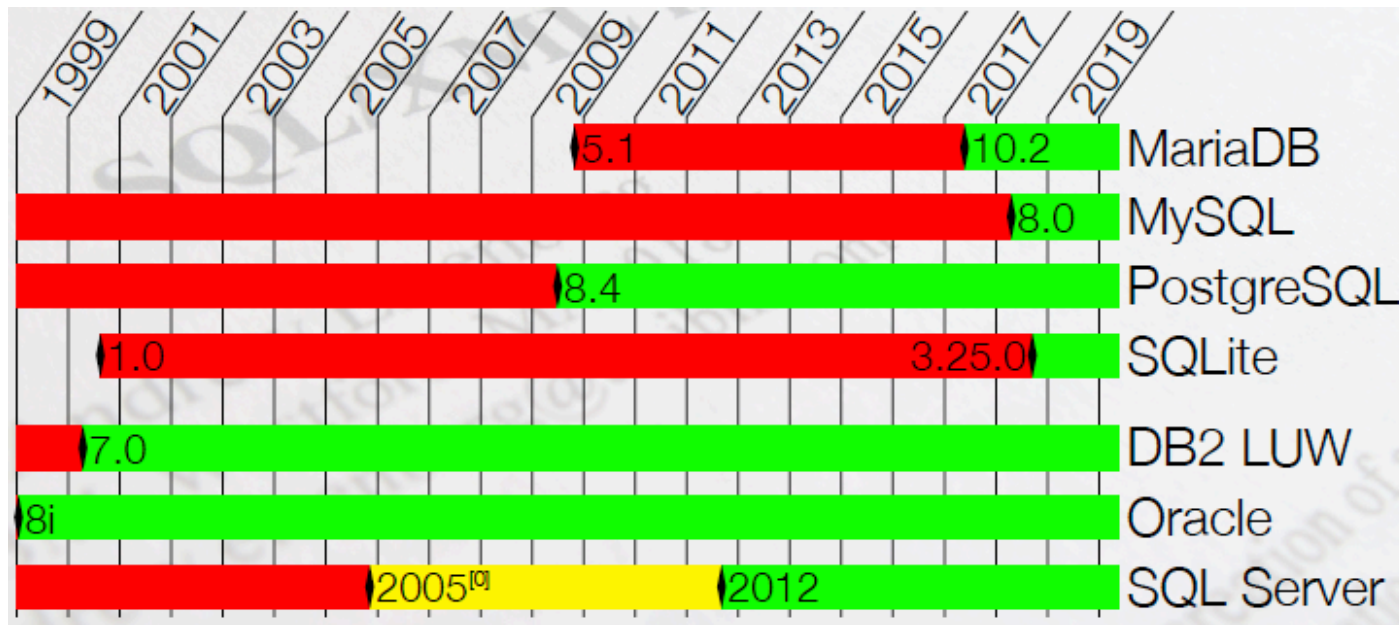
DEPTNO	NAME	JOB	SALARY	AVG_SAL
10	CLARK	MANAGER	2450	2073.21429
10	MILLER	CLERK	1300	2073.21429
10	KING	PRESIDENT	5000	2073.21429
20	FORD	ANALYST	3000	2073.21429
20	ADAMS	CLERK	1100	2073.21429
20	SCOTT	ANALYST	3000	2073.21429
20	JONES	MANAGER	2975	2073.21429
20	SMITH	CLERK	800	2073.21429
30	JAMES	CLERK	950	2073.21429
30	TURNER	SALESMAN	1500	2073.21429
30	MARTIN	SALESMAN	1250	2073.21429
30	WARD	SALESMAN	1250	2073.21429
30	ALLEN	SALESMAN	1600	2073.21429
30	BLAKE	MANAGER	2850	2073.21429



- The analytic clause is broken down as follows.



SQL:2003 - Analytical



See Markus Winand website for more SQL compatibility matrix
<https://modern-sql.com>


```
select deptno,
       name,
       job,
       salary,
       avg (salary) over () avg_sal
from employee
order by deptno;
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	2 (0)	00:00:01
1	WINDOW SORT		1	39	2 (0)	00:00:01
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2 (0)	00:00:01



```
select deptno,
       name,
       job,
       salary,
       (select avg(salary) from employee) avg_sal
from employee
order by deptno;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	5 (20)	00:00:01
1	SORT AGGREGATE		1	13		
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	13	2 (0)	00:00:01
3	SORT ORDER BY		1	39	5 (20)	00:00:01
4	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2 (0)	00:00:01

DEPTNO	NAME	JOB	SALARY	AVG_SAL
10	CLARK	MANAGER	2450	2073.21429
10	MILLER	CLERK	1300	2073.21429
10	KING	PRESIDENT	5000	2073.21429
20	FORD	ANALYST	3000	2073.21429
20	ADAMS	CLERK	1100	2073.21429
20	SCOTT	ANALYST	3000	2073.21429
20	JONES	MANAGER	2975	2073.21429
20	SMITH	CLERK	800	2073.21429
30	JAMES	CLERK	950	2073.21429
30	TURNER	SALESMAN	1500	2073.21429
30	MARTIN	SALESMAN	1250	2073.21429
30	WARD	SALESMAN	1250	2073.21429
30	ALLEN	SALESMAN	1600	2073.21429
30	BLAKE	MANAGER	2850	2073.21429

No parameters

```
select deptno,  
       name,  
       job,  
       salary,  
       avg (salary) over () avg_sal  
from employee  
order by deptno;
```

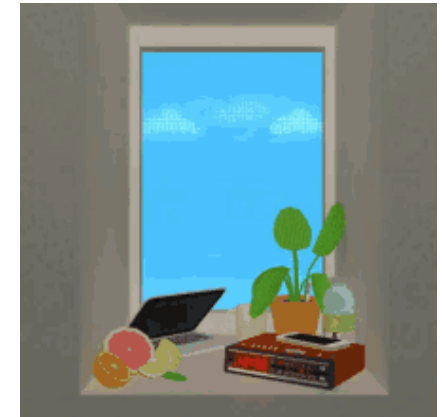
What-if ?
AVG per group
(e.g. DEPTNO)

DEPTNO	NAME	JOB	SALARY	AVG_SAL
10	CLARK	MANAGER	2450	2073.21429
10	MILLER	CLERK	1300	2073.21429
10	KING	PRESIDENT	5000	2073.21429
20	FORD	ANALYST	3000	2073.21429
20	ADAMS	CLERK	1100	2073.21429
20	SCOTT	ANALYST	3000	2073.21429
20	JONES	MANAGER	2975	2073.21429
20	SMITH	CLERK	800	2073.21429
30	JAMES	CLERK	950	2073.21429
30	TURNER	SALESMAN	1500	2073.21429
30	MARTIN	SALESMAN	1250	2073.21429
30	WARD	SALESMAN	1250	2073.21429
30	ALLEN	SALESMAN	1600	2073.21429
30	BLAKE	MANAGER	2850	2073.21429

```

select deptno,
       name,
       job,
       salary,
       avg (salary) over (partition by DEPTNO) avg_sal
from employee
order by deptno;

```



Window <= Partition
Default : Window = Partition

**What-if
AVG per group
(e.g. DEPTNO)**

DEPTNO	NAME	JOB	SALARY	AVG_SAL
10	CLARK	MANAGER	2450	2916.66667
10	MILLER	CLERK	1300	2916.66667
10	KING	PRESIDENT	5000	2916.66667
20	FORD	ANALYST	3000	2175
20	ADAMS	CLERK	1100	2175
20	SCOTT	ANALYST	3000	2175
20	JONES	MANAGER	2975	2175
20	SMITH	CLERK	800	2175
30	JAMES	CLERK	950	1566.66667
30	TURNER	SALESMAN	1500	1566.66667
30	MARTIN	SALESMAN	1250	1566.66667
30	WARD	SALESMAN	1250	1566.66667
30	ALLEN	SALESMAN	1600	1566.66667
30	BLAKE	MANAGER	2850	1566.66667

Partition

Partition

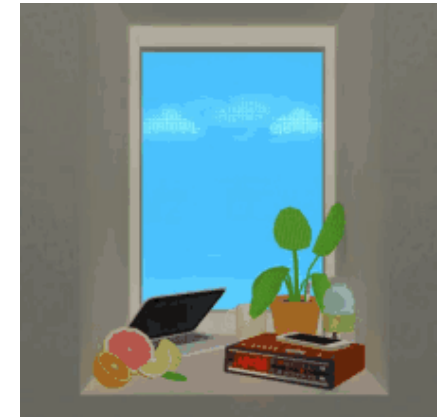
Partition

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	39	2 (0)	00:00:01
1	WINDOW SORT		1	39	2 (0)	00:00:01
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2 (0)	00:00:01

```

select deptno,
       name,
       job,
       salary,
       avg (salary) over (partition by DEPTNO) avg_sal
from employee
order by deptno;

```



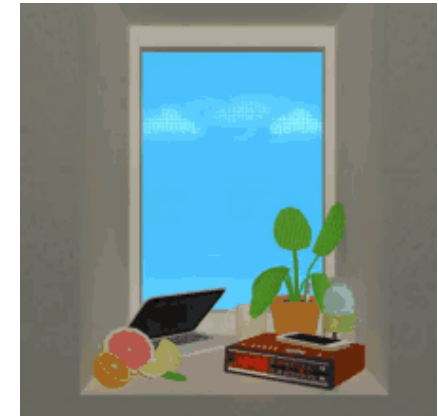
Window <= Partition
Default : Window = Partition

	DEPTNO	NAME	JOB	SALARY	AVG_SAL	
Window	10	CLARK	MANAGER	2450	2916.66667	Partition
	10	MILLER	CLERK	1300	2916.66667	
	10	KING	PRESIDENT	5000	2916.66667	
	20	FORD	ANALYST	3000	2175	Partition
	20	ADAMS	CLERK	1100	2175	
	20	SCOTT	ANALYST	3000	2175	
	20	JONES	MANAGER	2975	2175	
	20	SMITH	CLERK	800	2175	Partition
	30	JAMES	CLERK	950	1566.66667	
	30	TURNER	SALESMAN	1500	1566.66667	
	30	MARTIN	SALESMAN	1250	1566.66667	
	30	WARD	SALESMAN	1250	1566.66667	
	30	ALLEN	SALESMAN	1600	1566.66667	
	30	BLAKE	MANAGER	2850	1566.66667	

```

select deptno,
       name,
       job,
       salary,
       avg (salary) over (partition by DEPTNO) avg_sal,
       sum (salary) over (partition by DEPTNO
                          order by NAME) run_total
from employee
order by deptno;

```



Window <= Partition
Default : Window = Partition

Window

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		1	39	2	(0)	00:00:01
1	WINDOW SORT		1	39	2	(0)	00:00:01
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2	(0)	00:00:01

Still the same plan & Only one read of the table!

DEPTNO	NAME	JOB	SALARY	AVG_SAL	RUN_TOTAL
10	CLARK	MANAGER	2450	2916.66667	2450
10	KING	PRESIDENT	5000	2916.66667	7450
10	MILLER	CLERK	1300	2916.66667	8750
20	ADAMS	CLERK	1100	2175	1100
20	FORD	ANALYST	3000	2175	4100
20	JONES	MANAGER	2975	2175	7075
20	SCOTT	ANALYST	3000	2175	10075
20	SMITH	CLERK	800	2175	10875
30	ALLEN	SALESMAN	1600	1566.66667	1600
30	BLAKE	MANAGER	2850	1566.66667	4450
30	JAMES	CLERK	950	1566.66667	5400
30	MARTIN	SALESMAN	1250	1566.66667	6650
30	TURNER	SALESMAN	1500	1566.66667	8150
30	WARD	SALESMAN	1250	1566.66667	9400

What Range/Window of Records?

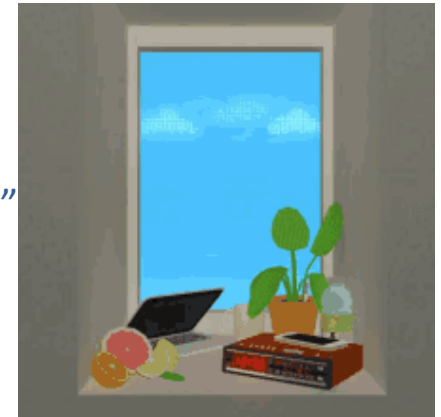
```
select deptno,
       name,
       job,
       salary,
```

```
avg (salary) over (partition by DEPTNO) avg_sal,
sum (salary) over (partition by DEPTNO
                  order by NAME) run_total
```

```
from employee
order by deptno;
```

Default is:

"range between Unbounded Preceding and Current Row"



Window <= Partition

Default : Window = Partition

Start and end points.

UNBOUNDED PRECEDING : Window starts at first row of partition.

UNBOUNDED FOLLOWING : Window ends at last row of partition

CURRENT ROW : Window starts or ends at the current row

value expr PRECEDING : Physical/logical offset before current row.

value expr FOLLOWING : Physical/logical offset after current row.

Window

RANGE BETWEEN start_point AND end_point

ROWS BETWEEN start_point AND end_point



DEPTNO	NAME	JOB	SALARY	AVG_SAL	RUN_TOTAL
10	CLARK	MANAGER	2450	2916.66667	2450
10	KING	PRESIDENT	5000	2916.66667	7450
10	MILLER	CLERK	1300	2916.66667	8750
20	ADAMS	CLERK	1100	2175	1100
20	FORD	ANALYST	3000	2175	4100
20	JONES	MANAGER	2975	2175	7075
20	SCOTT	ANALYST	3000	2175	10075
20	SMITH	CLERK	800	2175	10875
30	ALLEN	SALESMAN	1600	1566.66667	1600
30	BLAKE	MANAGER	2850	1566.66667	4450
30	JAMES	CLERK	950	1566.66667	5400
30	MARTIN	SALESMAN	1250	1566.66667	6650
30	TURNER	SALESMAN	1500	1566.66667	8150
30	WARD	SALESMAN	1250	1566.66667	9400

What Range/Window of Records?

Default is:

"range between Unbounded Preceding and Current Row"

```
SELECT empno,  
       deptno,  
       salary,
```

```
      AVG(salary) OVER (PARTITION BY deptno ORDER BY salary) AS avg_dept_sal,
```

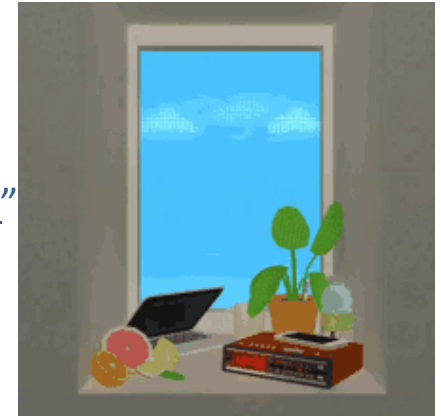
```
      AVG(salary) OVER (PARTITION BY deptno ORDER BY salary
```

```
                      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS avg_dept_sal,
```

```
      AVG(salary) OVER (PARTITION BY deptno ORDER BY salary
```

```
                      ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS avg_dept_sal
```

```
FROM   employee;
```



EMPNO	DEPTNO	SALARY	AVG_DEPT_SAL	AVG_DEPT_SAL	AVG_DEPT_SAL
7934	10	1300	1300	1300	2916.66667
7782	10	2450	1875	1875	2916.66667
7839	10	5000	2916.66667	2916.66667	2916.66667
7369	20	800	800	800	2175
7876	20	1100	950	950	2175
7566	20	2975	1625	1625	2175
7788	20	3000	2175	1968.75	2175
7902	20	3000	2175	2175	2175
7900	30	950	950	950	1566.66667
7521	30	1250	1150	1100	1566.66667
7654	30	1250	1150	1150	1566.66667
7844	30	1500	1237.5	1237.5	1566.66667
7499	30	1600	1310	1310	1566.66667
7698	30	2850	1566.66667	1566.66667	1566.66667

Ranking

Row Number



```
SELECT empno,
       deptno,
       salary,
       ROW_NUMBER() over (PARTITION BY deptno ORDER BY salary desc) sal_rank_rnum,
FROM   employee
ORDER BY deptno, salary desc;
```

EMPNO	DEPTNO	SALARY	SAL_RANK_RNUM
7839	10	5000	1
7782	10	2450	2
7934	10	1300	3
7788	20	3000	1
7902	20	3000	2
7566	20	2975	3
7876	20	1100	4
7369	20	800	5
7698	30	2850	1
7499	30	1600	2
7844	30	1500	3
7654	30	1250	4
7521	30	1250	5
7900	30	950	6

Ranking

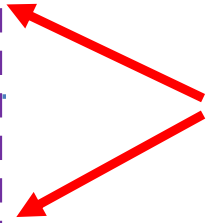
Row Number

Rank



```
SELECT empno,
       deptno,
       salary,
       ROW_NUMBER() over (PARTITION BY deptno ORDER BY salary desc) sal_rank_rnum,
       RANK() OVER (PARTITION BY deptno ORDER BY salary desc) AS sal_rank
FROM   employee
ORDER BY deptno, salary desc;
```

EMPNO	DEPTNO	SALARY	SAL_RANK_RNUM	SAL_RANK
7839	10	5000	1	1
7782	10	2450	2	2
7934	10	1300	3	3
7788	20	3000	1	1
7902	20	3000	2	1
7566	20	2975	3	3
7876	20	1100	4	4
7369	20	800	5	5
7698	30	2850	1	1
7499	30	1600	2	2
7844	30	1500	3	3
7654	30	1250	4	4
7521	30	1250	5	4
7900	30	950	6	6



Ranking

Row Number

Rank

Dense Rank



```
SELECT empno,
       deptno,
       salary,
       ROW_NUMBER() over (PARTITION BY deptno ORDER BY salary desc) sal_rank_rnum,
       RANK() OVER (PARTITION BY deptno ORDER BY salary desc) AS sal_rank,
       DENSE_RANK() OVER (PARTITION BY deptno ORDER BY salary desc) AS sal_dense_rank
FROM   employee
ORDER BY deptno, salary desc;
```

Which one should you use?

It Depends!

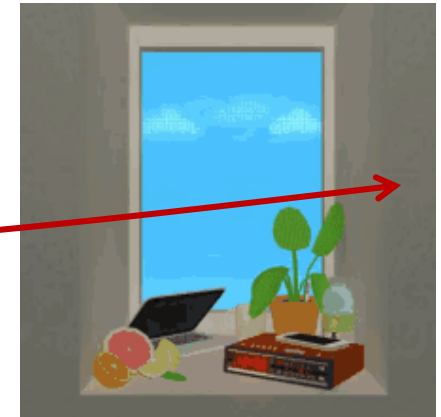
Depends on what the business needs

							No Gaps		
							SAL_RANK_RNUM	SAL_RANK	SAL_DENSE_RANK
							1	1	1
							2	2	2
							3	3	3
							1	1	1
							2	1	1
							3	3	2
							4	4	3
							5	5	4
							1	1	1
							2	2	2
							3	3	3
							4	4	4
							5	4	4
							6	6	5

0	SELECT STATEMENT		1	39	2	(0)	00:00:01
1	WINDOW SORT		1	39	2	(0)	00:00:01
2	TABLE ACCESS STORAGE FULL	EMPLOYEE	1	39	2	(0)	00:00:01

Lead + Lag

```
select empno,
       deptno,
       name,
       job,
       salary,
       lead (name) over (partition by deptno order by salary) lead_name,
       lead (salary) over (partition by deptno order by salary) lead_salary
from employee
order by deptno, salary desc;
```

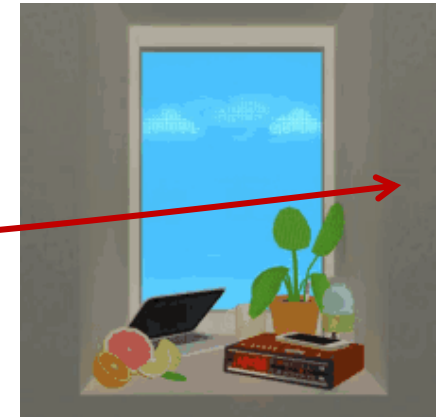


EMPNO	DEPTNO	NAME	JOB	SALARY	LEAD_NAME	LEAD_SALARY
7934	10	MILLER	CLERK	1300	CLARK	2450
7782	10	CLARK	MANAGER	2450	KING	5000
7839	10	KING	PRESIDENT	5000		
7369	20	SMITH	CLERK	800	ADAMS	1100
7876	20	ADAMS	CLERK	1100	JONES	2975
7566	20	JONES	MANAGER	2975	SCOTT	3000
7788	20	SCOTT	ANALYST	3000	FORD	3000
7902	20	FORD	ANALYST	3000		
7900	30	JAMES	CLERK	950	WARD	1250
7521	30	WARD	SALESMAN	1250	MARTIN	1250
7654	30	MARTIN	SALESMAN	1250	TURNER	1500
7844	30	TURNER	SALESMAN	1500	ALLEN	1600
7499	30	ALLEN	SALESMAN	1600	BLAKE	2850
7698	30	BLAKE	MANAGER	2850		

Lead + Lag

```
select empno,
       deptno,
       name,
       job,
       salary,
       lead (name, 2) over (partition by deptno order by salary) lead_name,
       lead (salary, 2) over (partition by deptno order by salary) lead_salary
from employee
order by deptno, salary desc;
```

next value



Change the off-set

EMPNO	DEPTNO	NAME	JOB	SALARY	LEAD_NAME	LEAD_SALARY
7839	10	KING	PRESIDENT	5000		
7782	10	CLARK	MANAGER	2450		
7934	10	MILLER	CLERK	1300	KING	5000
7788	20	SCOTT	ANALYST	3000		
7902	20	FORD	ANALYST	3000		
7566	20	JONES	MANAGER	2975	FORD	3000
7876	20	ADAMS	CLERK	1100	SCOTT	3000
7369	20	SMITH	CLERK	800	JONES	2975
7698	30	BLAKE	MANAGER	2850		
7499	30	ALLEN	SALESMAN	1600		
7844	30	TURNER	SALESMAN	1500	BLAKE	2850
7654	30	MARTIN	SALESMAN	1250	ALLEN	1600
7521	30	WARD	SALESMAN	1250	TURNER	1500
7900	30	JAMES	CLERK	950	MARTIN	1250

Lead

+

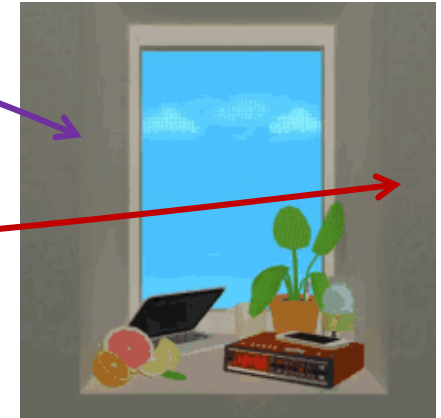
Lag

previous value

next value

```
select empno,  
       deptno,  
       name,  
       job,  
       salary,  
       lead (name) over (partition by deptno order by salary) lead_name,  
       lead (salary) over (partition by deptno order by salary) lead_salary,  
       lag (name) over (partition by deptno order by salary) lag_name,  
       lag (salary) over (partition by deptno order by salary) lag_salary  
from employee  
order by deptno, salary desc;
```

EMPNO	DEPTNO	NAME	JOB	SALARY	LEAD_NAME	LEAD_SALARY
7934	10	MILLER	CLERK	1300	CLARK	2450
7782	10	CLARK	MANAGER	2450	KING	5000
7839	10	KING	PRESIDENT	5000		
7369	20	SMITH	CLERK	800	ADAMS	1100
7876	20	ADAMS	CLERK	1100	JONES	2975
7566	20	JONES	MANAGER	2975	SCOTT	3000
7788	20	SCOTT	ANALYST	3000	FORD	3000
7902	20	FORD	ANALYST	3000		
7900	30	JAMES	CLERK	950	WARD	1250
7521	30	WARD	SALESMAN	1250	MARTIN	1250
7654	30	MARTIN	SALESMAN	1250	TURNER	1500
7844	30	TURNER	SALESMAN	1500	ALLEN	1600
7499	30	ALLEN	SALESMAN	1600	BLAKE	2850
7698	30	BLAKE	MANAGER	2850		



Lead

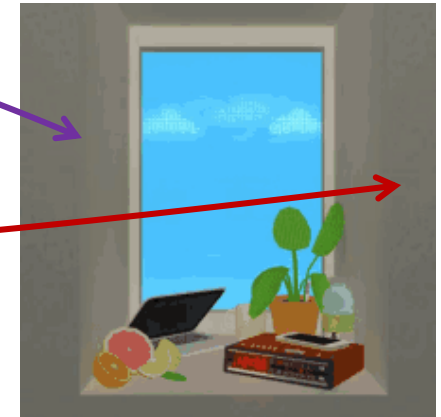
+

Lag

previous value

next value

```
select empno,  
       deptno,  
       name,  
       job,  
       salary,  
       lead (name) over (partition by deptno order by salary) lead_name,  
       lead (salary) over (partition by deptno order by salary) lead_salary,  
       lag (name) over (partition by deptno order by salary) lag_name,  
       lag (salary) over (partition by deptno order by salary) lag_salary  
from employee  
order by deptno, salary desc;
```



EMPNO	DEPTNO	NAME	JOB	SALARY	LEAD_NAME	LEAD_SALARY	LAG_NAME	LAG_SALARY
7934	10	MILLER	CLERK	1300	CLARK	2450		
7782	10	CLARK	MANAGER	2450	KING	5000	MILLER	1300
7839	10	KING	PRESIDENT	5000			CLARK	2450
7369	20	SMITH	CLERK	800	ADAMS	1100		
7876	20	ADAMS	CLERK	1100	JONES	2975	SMITH	800
7566	20	JONES	MANAGER	2975	SCOTT	3000	ADAMS	1100
7788	20	SCOTT	ANALYST	3000	FORD	3000	JONES	2975
7902	20	FORD	ANALYST	3000			SCOTT	3000
7900	30	JAMES	CLERK	950	WARD	1250		
7521	30	WARD	SALESMAN	1250	MARTIN	1250	JAMES	950
7654	30	MARTIN	SALESMAN	1250	TURNER	1500	WARD	1250
7844	30	TURNER	SALESMAN	1500	ALLEN	1600	MARTIN	1250
7499	30	ALLEN	SALESMAN	1600	BLAKE	2850	TURNER	1500
7698	30	BLAKE	MANAGER	2850			ALLEN	1600

Calculating Moving Average

```
select month,  
       SUM(amount) AS month_amount,  
       AVG(SUM(amount))  
         OVER (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg  
from sales  
group by month  
order by month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	58704.52	58704.52
2	28289.3	43496.91
3	20167.83	35720.55
4	50082.9	39311.1375
5	17212.66	28938.1725
6	31128.92	29648.0775
7	78299.47	44180.9875
8	42869.64	42377.6725
9	35299.22	46899.3125
10	43028.38	49874.1775
11	26053.46	36812.675
12	20067.28	31112.085

Calculating Moving Average

```
select month,  
       SUM(amount) AS month_amount,  
       AVG(SUM(amount))  
         OVER (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg  
from sales  
group by month  
order by month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	58704.52	58704.52
2	28289.3	43496.91
3	20167.83	35720.55
4	50082.9	39311.1375
5	17212.66	28938.1725
6	31128.92	29648.0775
7	78299.47	44180.9875
8	42869.64	42377.6725
9	35299.22	46899.3125
10	43028.38	49874.1775
11	26053.46	36812.675
12	20067.28	31112.085

Calculating Moving Average

```
select month,  
       SUM(amount) AS month_amount,  
       AVG(SUM(amount))  
         OVER (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg  
from sales  
group by month  
order by month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	58704.52	58704.52
2	28289.3	43496.91
3	20167.83	35720.55
4	50082.9	39311.1375
5	17212.66	28938.1725
6	31128.92	29648.0775
7	78299.47	44180.9875
8	42869.64	42377.6725
9	35299.22	46899.3125
10	43028.38	49874.1775
11	26053.46	36812.675
12	20067.28	31112.085

Calculating Moving Average

```
select month,  
       SUM(amount) AS month_amount,  
       AVG(SUM(amount))  
         OVER (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg  
from sales  
group by month  
order by month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	58704.52	58704.52
2	28289.3	43496.91
3	20167.83	35720.55
4	50082.9	39311.1375
5	17212.66	28938.1725
6	31128.92	29648.0775
7	78299.47	44180.9875
8	42869.64	42377.6725
9	35299.22	46899.3125
10	43028.38	49874.1775
11	26053.46	36812.675
12	20067.28	31112.085

Calculating Moving Average

```
select month,  
       SUM(amount) AS month_amount,  
       AVG(SUM(amount))  
         OVER (ORDER BY month ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg  
from sales  
group by month  
order by month;
```

MONTH	MONTH_AMOUNT	MOVING_AVERAGE
1	58704.52	58704.52
2	28289.3	43496.91
3	20167.83	35720.55
4	50082.9	39311.1375
5	17212.66	28938.1725
6	31128.92	29648.0775
7	78299.47	44180.9875
8	42869.64	42377.6725
9	35299.22	46899.3125
10	43028.38	49874.1775
11	26053.46	36812.675
12	20067.28	31112.085

- List Agg
- Pivot
- UnPivot
- Rollup
- Cube
- Pagination / Top N-Queries

Listing values : LISTAGG

```
select deptno,  
       name  
from employee  
order by deptno, name;
```

```
select deptno,  
       LISTAGG(name, ',') WITHIN GROUP (ORDER BY name) AS emp  
from   employee  
group by deptno;
```

DEPTNO	NAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

DEPTNO	EMP
10	CLARK,KING,MILLER
20	ADAMS,FORD,JONES,SCOTT,SMITH
30	ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD

Listing values : LISTAGG

```
select deptno,  
       name  
from employee  
order by deptno, name;
```

DEPTNO	NAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

```
select deptno,  
       LISTAGG(name, ',') WITHIN GROUP (ORDER BY salary desc) AS emp  
from employee  
group by deptno;
```

DEPTNO	EMP
10	CLARK, KING, MILLER
20	ADAMS, FORD, JONES, SCOTT, SMITH
30	ALLEN, BLAKE, JAMES, MARTIN, TURNER, WARD

DEPTNO	EMP
10	KING, CLARK, MILLER
20	FORD, SCOTT, JONES, ADAMS, SMITH
30	BLAKE, ALLEN, TURNER, MARTIN, WARD, JAMES

Listing values : LISTAGG

```
select deptno,
       name
from employee
order by deptno, name;
```

DEPTNO	NAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

```
select deptno,
       LISTAGG(name, ',') WITHIN GROUP (ORDER BY salary desc)
       OVER (PARTITION BY deptno) AS emp
from employee
group by deptno;
```

DEPTNO	EMP
10	CLARK,KING,MILLER
20	ADAMS,FORD,JONES,SCOTT,SMITH
30	ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD

DEPTNO	EMP
10	KING,CLARK,MILLER
20	FORD,SCOTT,JONES,ADAMS,SMITH
30	BLAKE,ALLEN,TURNER,MARTIN,WARD,JAMES

DEPTNO	EMP
10	KING,CLARK,MILLER
10	KING,CLARK,MILLER
10	KING,CLARK,MILLER
20	SCOTT,FORD,JONES,ADAMS,SMITH
20	SCOTT,FORD,JONES,ADAMS,SMITH
20	SCOTT,FORD,JONES,ADAMS,SMITH
20	SCOTT,FORD,JONES,ADAMS,SMITH
20	SCOTT,FORD,JONES,ADAMS,SMITH
30	BLAKE,ALLEN,TURNER,WARD,MARTIN,JAMES
30	BLAKE,ALLEN,TURNER,WARD,MARTIN,JAMES
30	BLAKE,ALLEN,TURNER,WARD,MARTIN,JAMES
30	BLAKE,ALLEN,TURNER,WARD,MARTIN,JAMES
30	BLAKE,ALLEN,TURNER,WARD,MARTIN,JAMES

Listing values : LISTAGG

Some DBs have a restriction on size of returned string of approx. 4K bytes or 32K depending on DB setting.

Use this to truncate and avoid errors

```
select deptno,
       name
from employee
order by deptno, name;
```

```
select deptno,
       LISTAGG(name, ', ' ON OVERFLOW TRUNCATE '...' ) WITHIN GROUP (ORDER
       BY salary desc)
       OVER (PARTITION BY deptno) AS emp
from employee
group by deptno;
```

DEPTNO	NAME
10	CLARK
10	KING
10	MILLER
20	ADAMS
20	FORD
20	JONES
20	SCOTT
20	SMITH
30	ALLEN
30	BLAKE
30	JAMES
30	MARTIN
30	TURNER
30	WARD

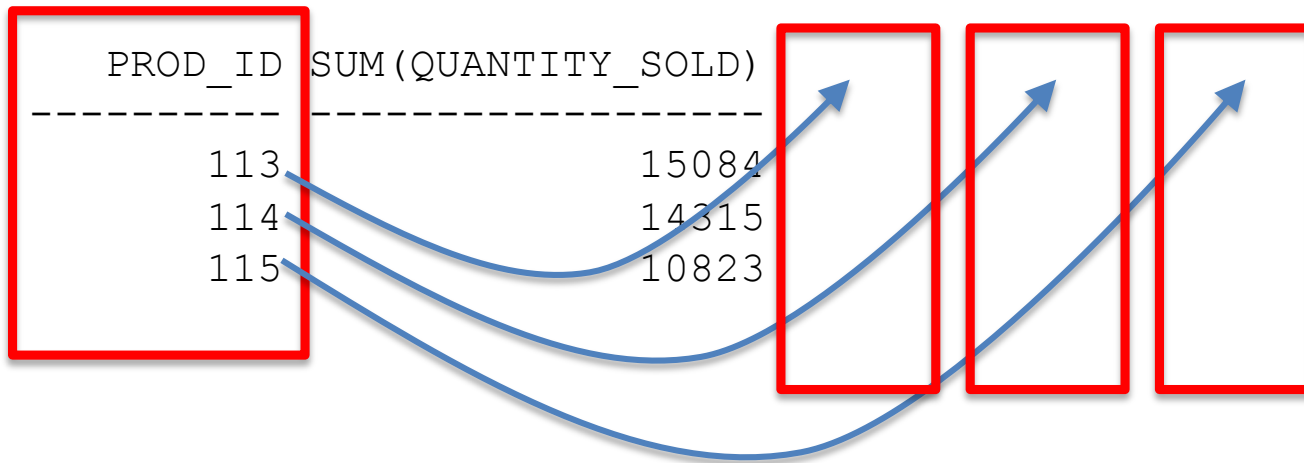
DEPTNO	EMP
10	CLARK, KING, MILLER
20	ADAMS, FORD, JONES, SCOTT, SMITH
30	ALLEN, BLAKE, JAMES, MARTIN, TURNER, WARD

DEPTNO	EMP
10	KING, CLARK, MILLER
20	FORD, SCOTT, JONES, ADAMS, SMITH
30	BLAKE, ALLEN, TURNER, MARTIN, WARD, JAMES

DEPTNO	EMP
10	KING, CLARK, MILLER
10	KING, CLARK, MILLER
10	KING, CLARK, MILLER
20	SCOTT, FORD, JONES, ADAMS, SMITH
20	SCOTT, FORD, JONES, ADAMS, SMITH
20	SCOTT, FORD, JONES, ADAMS, SMITH
20	SCOTT, FORD, JONES, ADAMS, SMITH
20	SCOTT, FORD, JONES, ADAMS, SMITH
30	BLAKE, ALLEN, TURNER, WARD, MARTIN, JAMES
30	BLAKE, ALLEN, TURNER, WARD, MARTIN, JAMES
30	BLAKE, ALLEN, TURNER, WARD, MARTIN, JAMES
30	BLAKE, ALLEN, TURNER, WARD, MARTIN, JAMES
30	BLAKE, ALLEN, TURNER, WARD, MARTIN, JAMES

Pivot

```
select prod_id, sum(quantity_sold)
from   sales
where  prod_id in (113, 114, 115)
group by prod_id
order by prod_id;
```



Pivot

```
select *  
from (select region, prod_id, quantity_sold from sales)  
      PIVOT (sum(quantity_sold) for (prod_id)  
              IN (113 as Prod_113, 114 as Prod_114, 115 as Prod_115));
```

REGION	PROD_113	PROD_114	PROD_115
EAST	15084	14315	10823
WEST	24916	31005	16340
...			



UNPIVOT command

ROLLUP



Use the ROLLUP operation to produce cumulative aggregates, such as subtotals and grand totals.

```
select deptno, job, sum(salary)
from   employee
group by (deptno, job);
```

DEPTNO	JOB	SUM(SALARY)
20	MANAGER	2975
10	PRESIDENT	5000
20	ANALYST	6000
10	CLERK	1300
30	SALESMAN	5600
10	MANAGER	2450
20	CLERK	1900
30	MANAGER	2850
30	CLERK	950

ROLLUP



Use the ROLLUP operation to produce cumulative aggregates, such as subtotals and grand totals.

```
select deptno, job, sum(salary)
from   employee
group by (deptno, job);
```

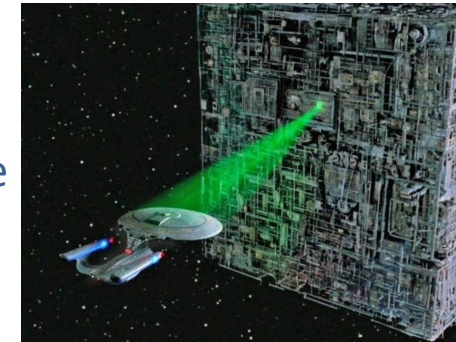
DEPTNO	JOB	SUM(SALARY)
20	MANAGER	2975
10	PRESIDENT	5000
20	ANALYST	6000
10	CLERK	1300
30	SALESMAN	5600
10	MANAGER	2450
20	CLERK	1900
30	MANAGER	2850
30	CLERK	950

```
select deptno, job, sum(salary)
from   employee
group by ROLLUP(deptno, job);
```

DEPTNO	JOB	SUM(SALARY)	
10	CLERK	1300	① Total by DEPTNO, JOB
10	MANAGER	2450	
10	PRESIDENT	5000	
10		8750	② Total by DEPTNO
20	CLERK	1900	
20	ANALYST	6000	
20	MANAGER	2975	
20		10875	
30	CLERK	950	
30	MANAGER	2850	
30	SALESMAN	5600	
30		9400	
		29025	③ Grand Total

CUBE

You can use the CUBE operator to produce cross-tabulation values with a single SELECT statement, including Rollup.



```
select deptno, job, sum(salary)
from   employee
group by (deptno, job);
```

DEPTNO	JOB	SUM(SALARY)
20	MANAGER	2975
10	PRESIDENT	5000
20	ANALYST	6000
10	CLERK	1300
30	SALESMAN	5600
10	MANAGER	2450
20	CLERK	1900
30	MANAGER	2850
30	CLERK	950

```
select deptno, job, sum(salary)
from   employee
group by CUBE(deptno, job);
```

DEPTNO	JOB	SUM(SALARY)	
		29025	① Grand Total
	CLERK	4150	
	ANALYST	6000	
	MANAGER	8275	② Total by JOB
	SALESMAN	5600	
	PRESIDENT	5000	
10		8750	③ Total by DEPTNO
10	CLERK	1300	
10	MANAGER	2450	
10	PRESIDENT	5000	
20		10875	③ Total by DEPTNO
20	CLERK	1900	
20	ANALYST	6000	
20	MANAGER	2975	
30		9400	③ Total by DEPTNO
30	CLERK	950	
30	MANAGER	2850	
30	SALESMAN	5600	

Extend with Grouping Sets e.g. GROUP BY CUBE (
department_id, (job_id, manager_id));

Pagination / Top N-Queries

Top-N Queries allows for result set limiting and paging

```
SELECT empno, name, salary
FROM   employee
ORDER BY salary ASC
FETCH FIRST 4 ROWS ONLY;
```

EMPNO	NAME	SALARY
7369	SMITH	800
7900	JAMES	950
7876	ADAMS	1100
7521	WARD	1250

```
[ OFFSET offset{ ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { rowcount| percent PERCENT} ]
{ ROW | ROWS } { ONLY | WITH TIES } ]
```

```
SELECT empno, name, salary
FROM   employee
ORDER BY salary ASC
FETCH FIRST 4 ROWS WITH TIES;
```

EMPNO	NAME	SALARY
7369	SMITH	800
7900	JAMES	950
7876	ADAMS	1100
7521	WARD	1250
7654	MARTIN	1250

```
SELECT empno, name, salary
FROM   employee
ORDER BY salary ASC
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

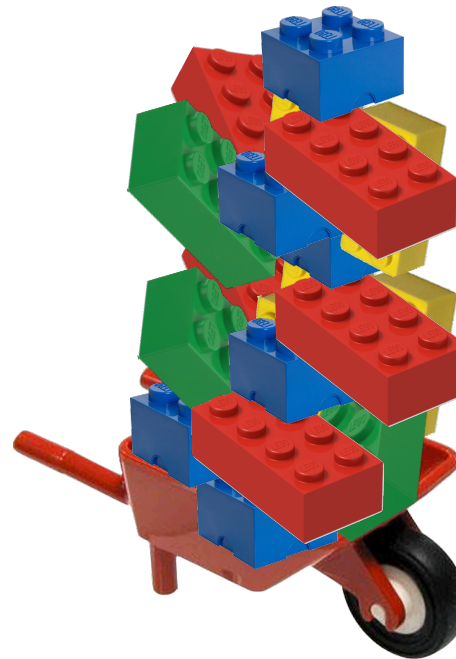
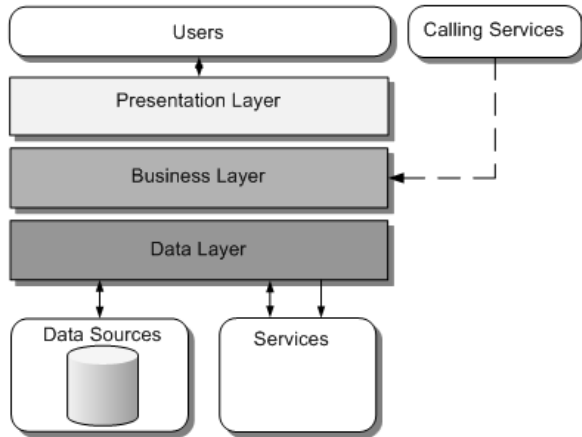
EMPNO	NAME	SALARY
7934	MILLER	1300
7844	TURNER	1500
7499	ALLEN	1600
7782	CLARK	2450
7698	BLAKE	2850

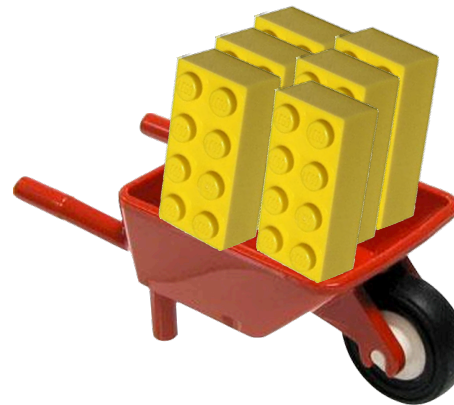
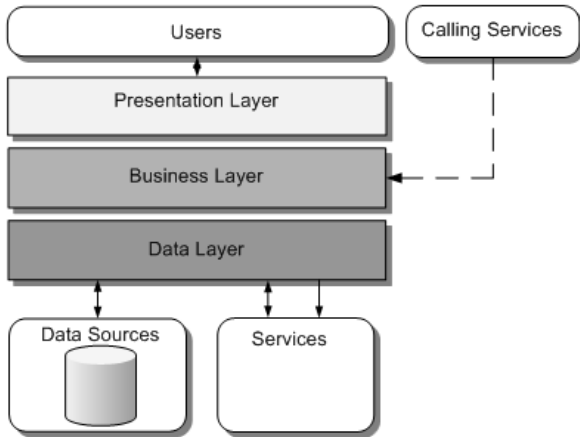
The starting point for the FETCH is OFFSET+1.

PERCENTILE_DISC	PREDICTION_COST	PREDICTION_PROBABILITY			
PERCENTILE_CONT	PREDICTION	PREDICTION_DETAILS		PREDICTION_SET	
PERCENTILE_RANK	VAR_POP	NTH_VALUE	CUME_DIST	CLUSTER_SET	
VAR_SAMP	COVAR_POP	LISTAGG	NTILE	LAG	VARIANCE
STDDEV_SAMP	RANK	COUNT		LEAD	COVAR_SAMP
		MIN	MAX		
MEDIAN	DENSE_RANK	SUM	AVG	LAST_VALUE	FIRST LAST
CLUSTER_DETAILS	ROW_NUMBER	FIRST_VALUE		FEATURE_VALUE	
CLUSTER_DISTANCE		RATIO_TO_REPORT		FEATURE_DETAILS	
CLUSTER_ID	CORR	FEATURE_ID	REGR_AVGX	FEATURE_SET	
REGR_SLOPE	CLUSTER_PROBABILITY	REGR_SXX	REGR_SYY		
REGR_INTERCEPT	REGR_COUNT	REGR_R2	REGR_AVGY	REGR_SXY	

One more thing...







Database Connection Buffer

Array Size



KNOW YOUR DATA

ORALYTICS

Importance of Setting the ARRAYSIZE (correctly)

```
# import the Oracle Python library
import cx_Oracle
import datetime
import pandas as pd
import numpy as np

# define the login details
p_username = "*****"
p_password = "*****"
p_host = "*****"
p_service = "*****"
p_port = "1521"

sql = 'select * from banking_marketing_data_balance_v'

startTime = datetime.datetime.now()
print(' Starting test at : ', startTime)
cur = con.cursor()
cur.arraysize = cur_array_size
results = cur.execute(sql).fetchall()
print(' Fetched ', len(results), ' in ', datetime.datetime.now()-startTime, ' seconds at ',
datetime.datetime.now())

startTime2 = datetime.datetime.now()
col_names = []
for i in range(0, len(cur.description)):
    col_names.append(cur.description[i][0])

print(' Fetched data & Created the list of Column names in ', datetime.datetime.now()-startTime, '
seconds at ', datetime.datetime.now())
```

72,666 records

Array Size	Time
5	36min 31sec
50	5min 32sec
500	40.76sec
1000	14.3sec
2000	10.18sec
3000	7.8sec

Importance of Setting the ARRAYSIZE (correctly)

```
cur_array_size = 3000
sql = 'select * from banking_marketing_data_balance_v'

print(' Will use arraysize = ', cur_array_size)
print('')

startTime = datetime.datetime.now()
print(' Starting test at : ', startTime)
df2 = pd.read_sql(sql, con)
print(' Finished creating Dataframe in ', datetime.datetime.now()-startTime)
# close the connection at end of experiments
con.close()
```

Will use arraysize = 3000

Starting test at : 16:35:08.095189

Finished creating Dataframe in **0:02:03.200411** seconds

2minutes 3seconds vs 7.8seconds

Default Arraysize => Pick value for your data

72,666 records

Array Size	Time
5	36min 31sec
50	5min 32sec
500	40.76sec
1000	14.3sec
2000	10.18sec
3000	7.8sec

Know your Data!

Know your Libraries!

Know your functions!

Test & Prove what works!



Wrapping up

A lot has changed
since SQL-92

Modern SQL is very powerful
Do it in the Database with
SQL

A lot has changed
since we first
learned SQL

If using SQL for CRUD only,
you are doing it wrong

SQL & Databases have
evolved beyond the
relational model
(JSON, XML, Media, Timeseries,
Graph, Spatial, ...)

Do More with Less by using
SQL analytical functions

A close-up of Yoda's head and shoulders. He is looking slightly to the right with a serious expression. His green skin is wrinkled, and he has large, pointed ears. He is wearing his characteristic brown robe. The background is a dark, out-of-focus forest with blueish-green light filtering through the trees.

Use SQL you must

Expert Developer
you'll become

Thank You



Brendan Tierney



ORACLE
ACE Director

t. @brendantierney

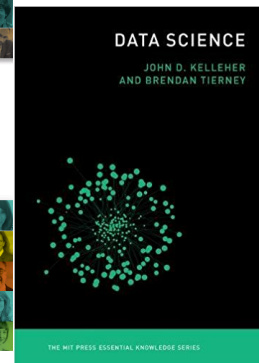
w. www.oralytics.com



97 Things About
Ethics Everyone
in Data Science
Should Know



Edited by Bill Franks



Available in
Chinese, Korean,
Japanese, Russian,
Turkish & Spanish

