Advanced SQL

SQL, the forgotten workhorse

Robert Ivens - ROCLASI Software Solutions

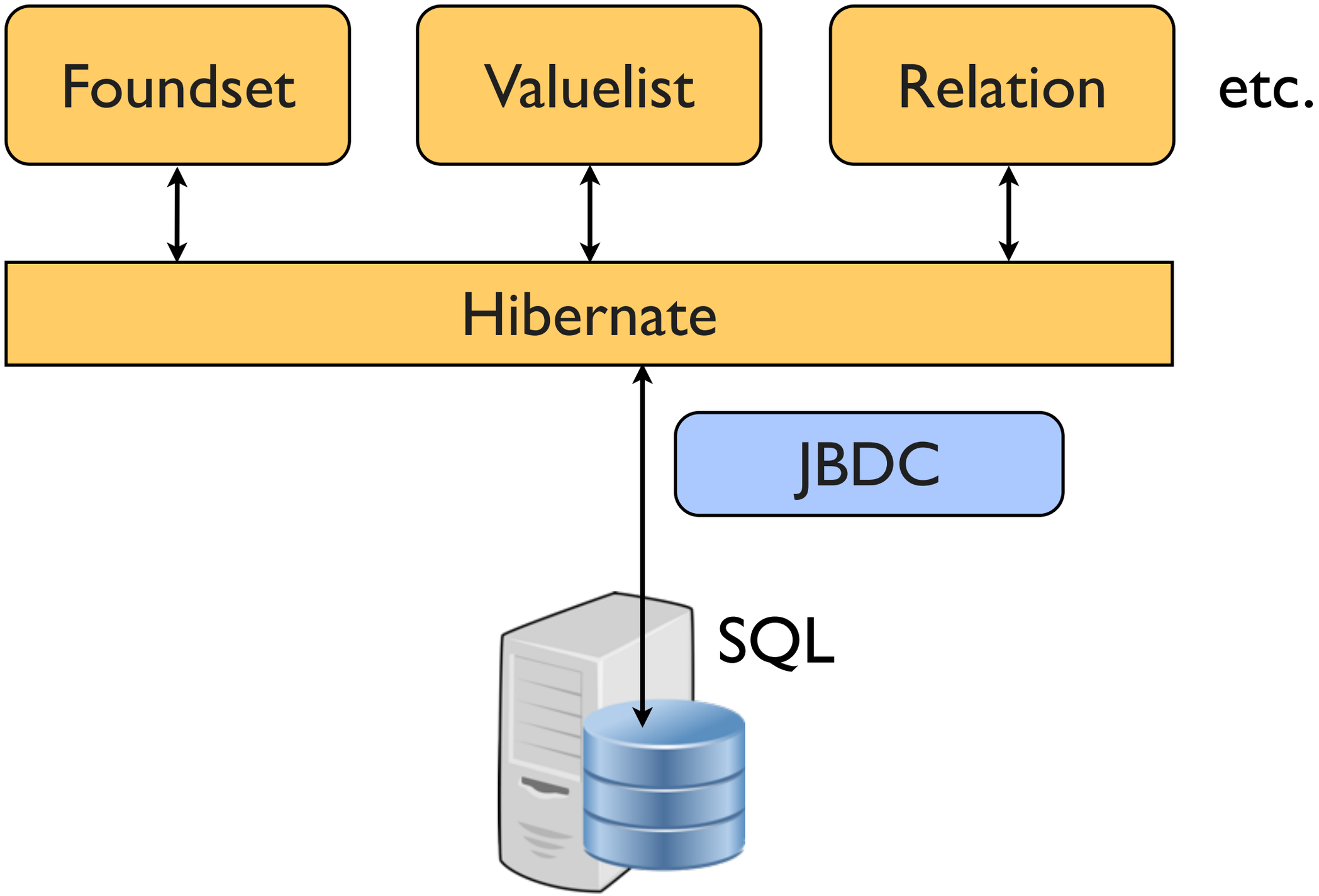
JBS Group, Partner

First,

lets talk about ORM's

"If you don't know SQL you shouldn't be allowed to use an ORM" - @jasonlotito (architect - MeetMe.com)

ORM's and SQL



ORM's and SQL

- SQL = Structured Query Language
  - It's a programming language!
- ORM's generate SQL
  - SQL is parsed by the RDBMS
- Your ORM is as good as the SQL it generates
  - which is not always great....
  - also ORM implementations do things you don't expect

SQL Select

SQL Select

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]

where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] |
column_definition [, ...] ) ]
    [ LATERAL ] function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
    from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```

From PostgreSQL documentation

http://www.postgresql.org/docs/9.3/static/sql-select.html

JOIN types

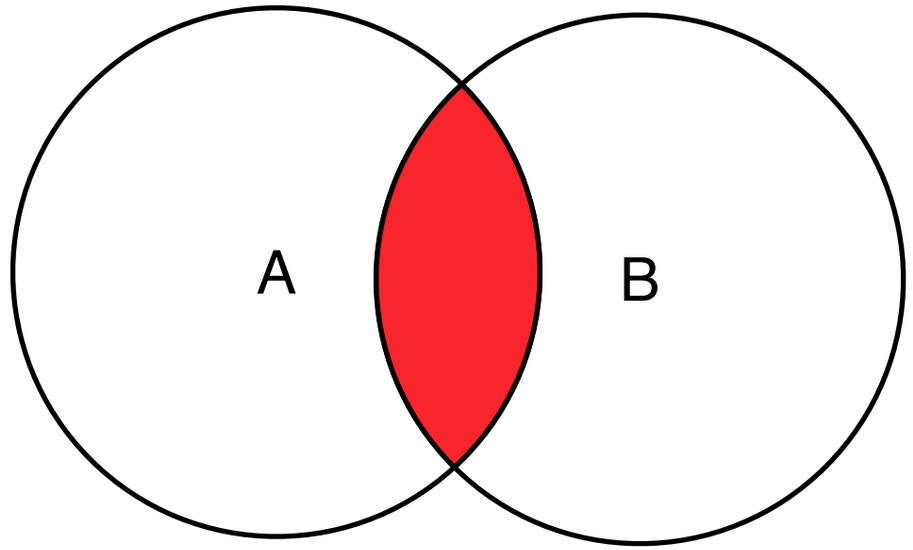
Join types

- (INNER) JOIN
- LEFT (OUTER) JOIN
- RIGHT (OUTER) JOIN
- FULL (OUTER) JOIN
- CROSS JOIN
- syntax:
  - **FROM** tbl1 *join_type* tbl2 **ON (** *join_condition* **)**
  - **FROM** tbl1 *join_type* tbl2 **USING (** *join_column*, [...] **)**

Join types

INNER JOIN

```
SELECT <select_list>
FROM Table_A A
INNER JOIN Table_B B
ON (A.Key = B.Key)
```

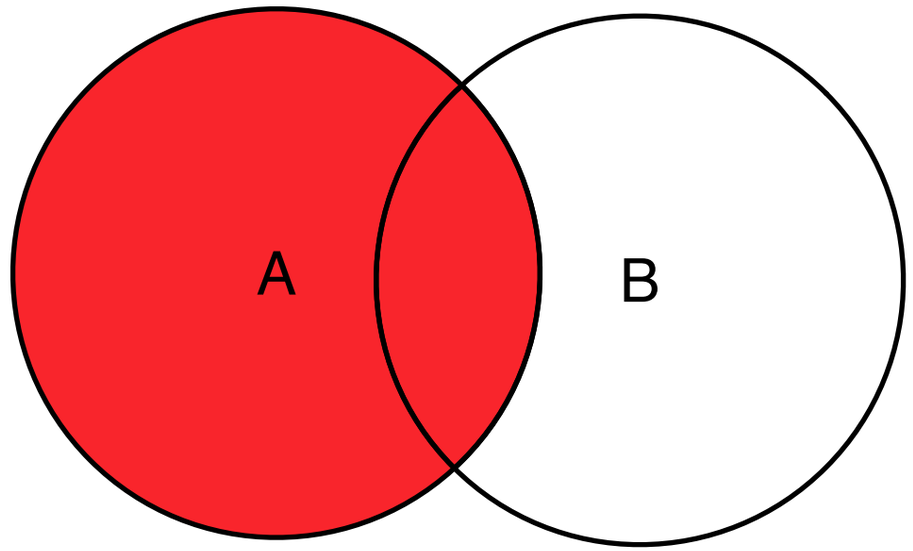


C.L. Moffatt, 2008

Join types

LEFT JOIN

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON (A.Key = B.Key)
```

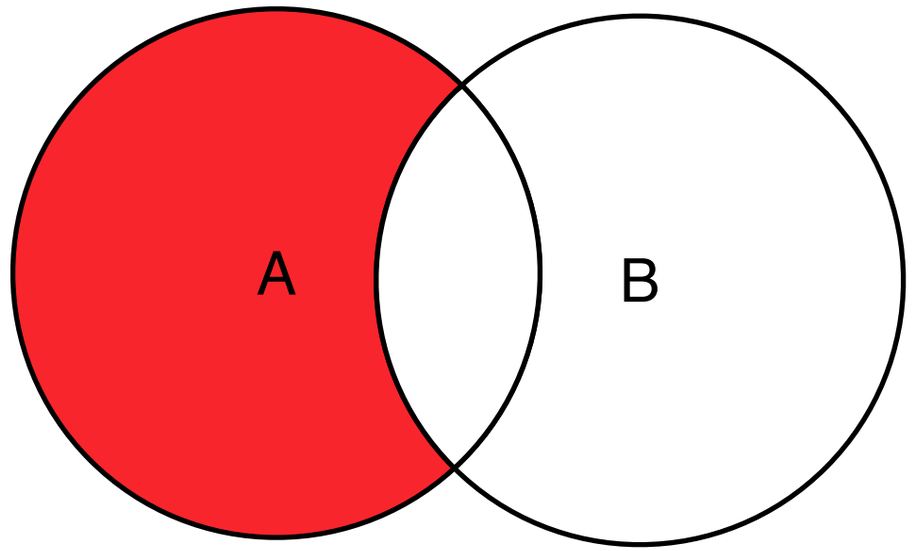


C.L. Moffatt, 2008

Join types

LEFT JOIN

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON (A.Key = B.Key)
WHERE B.Key IS NULL
```

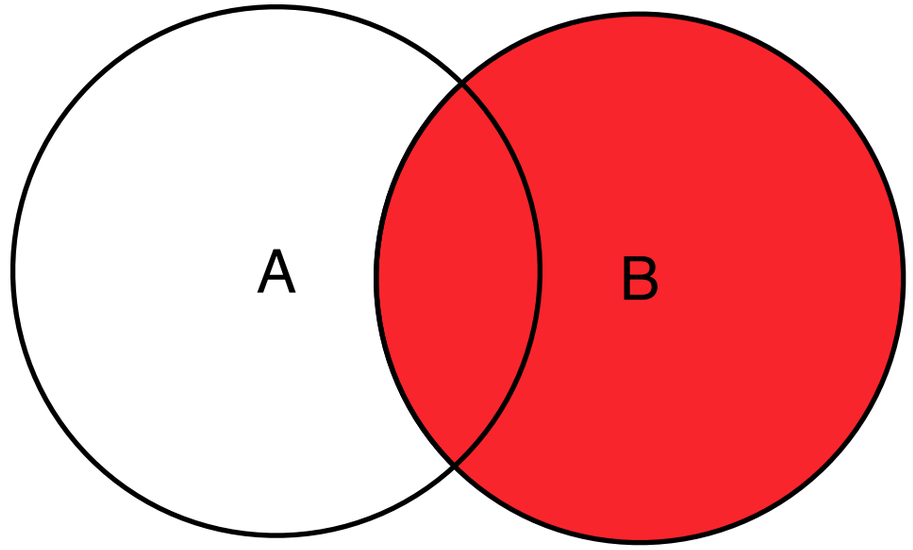


C.L. Moffatt, 2008

Join types

RIGHT JOIN

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON (A.Key = B.Key)
```

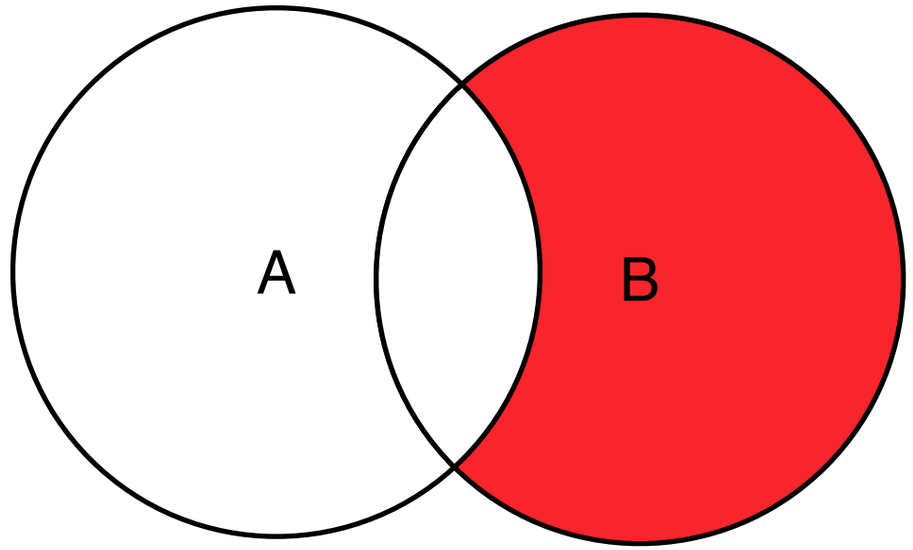


C.L. Moffatt, 2008

Join types

RIGHT JOIN

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON (A.Key = B.Key)
WHERE A.Key IS NULL
```

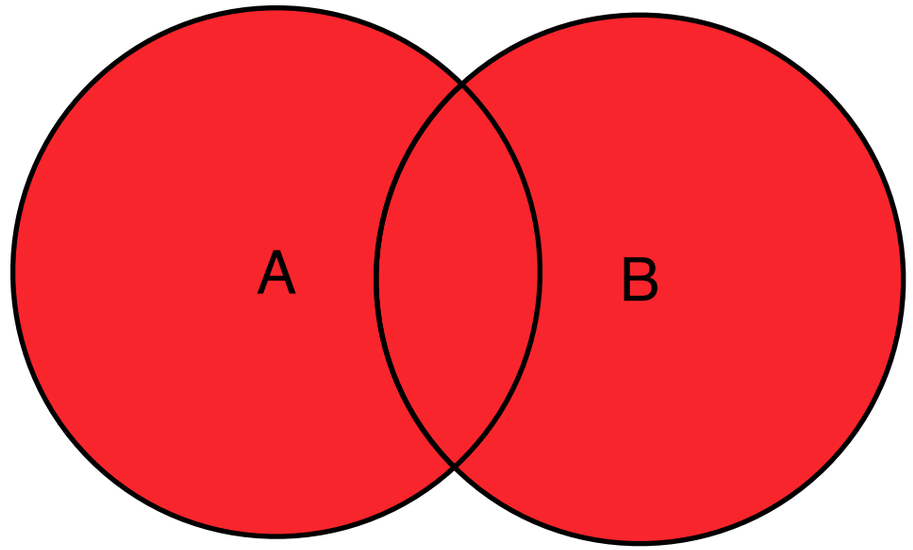


C.L. Moffatt, 2008

Join types

FULL OUTER JOIN

```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON (A.Key = B.Key)
```

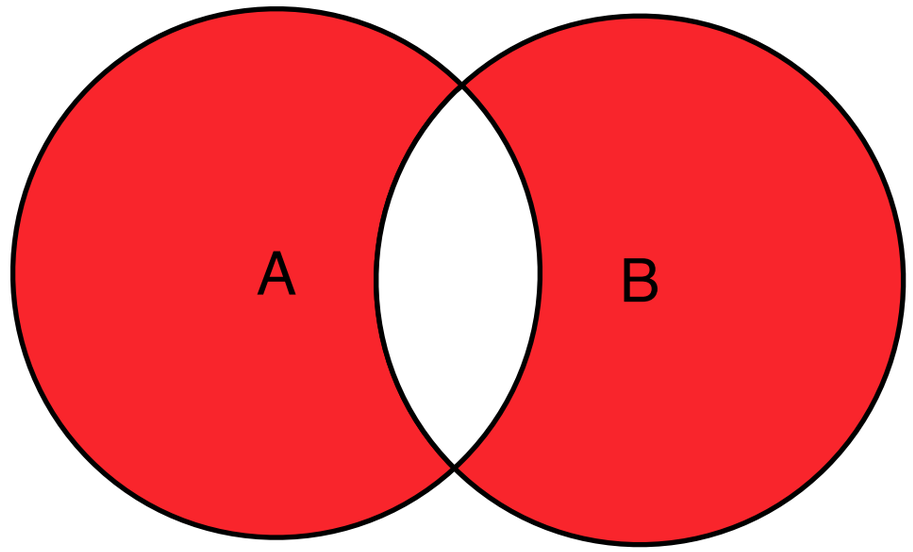


C.L. Moffatt, 2008

Join types

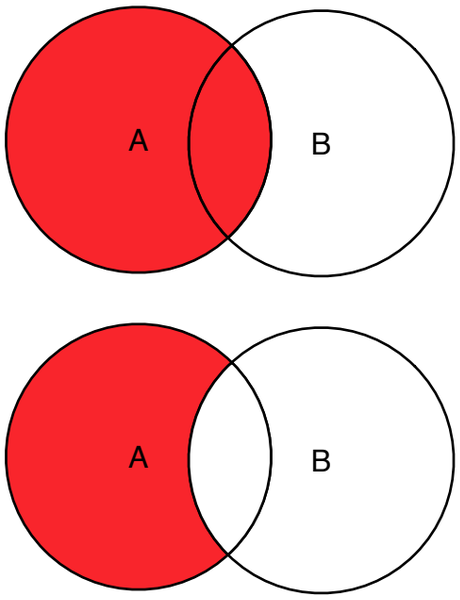
FULL OUTER JOIN

```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON (A.Key = B.Key)
WHERE A.Key IS NULL OR
B.Key IS NULL
```

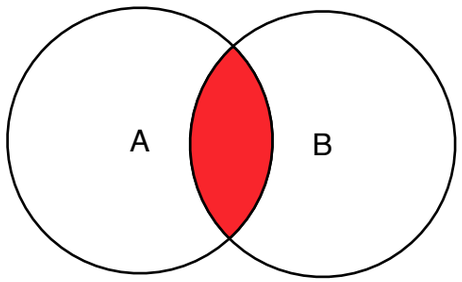


C.L. Moffatt, 2008

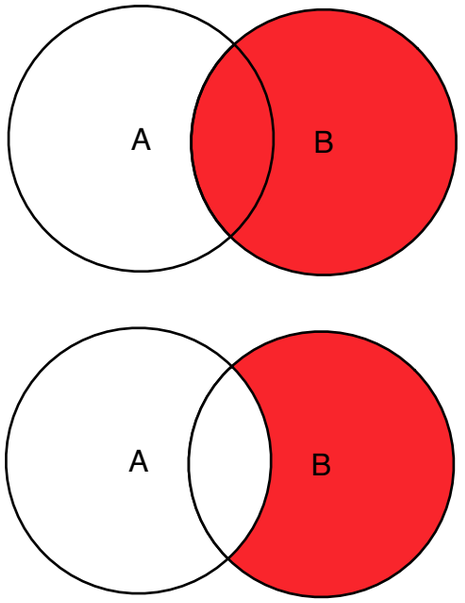
Join types



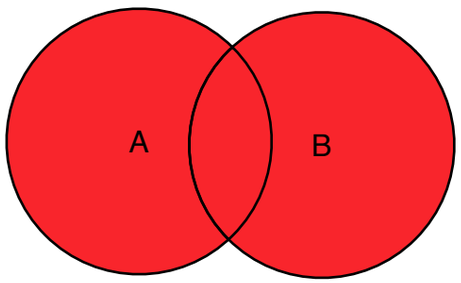
LEFT JOIN

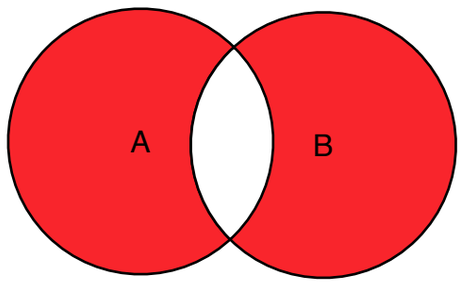


INNER JOIN



RIGHT JOIN





FULL OUTER JOIN

C.L. Moffatt, 2008

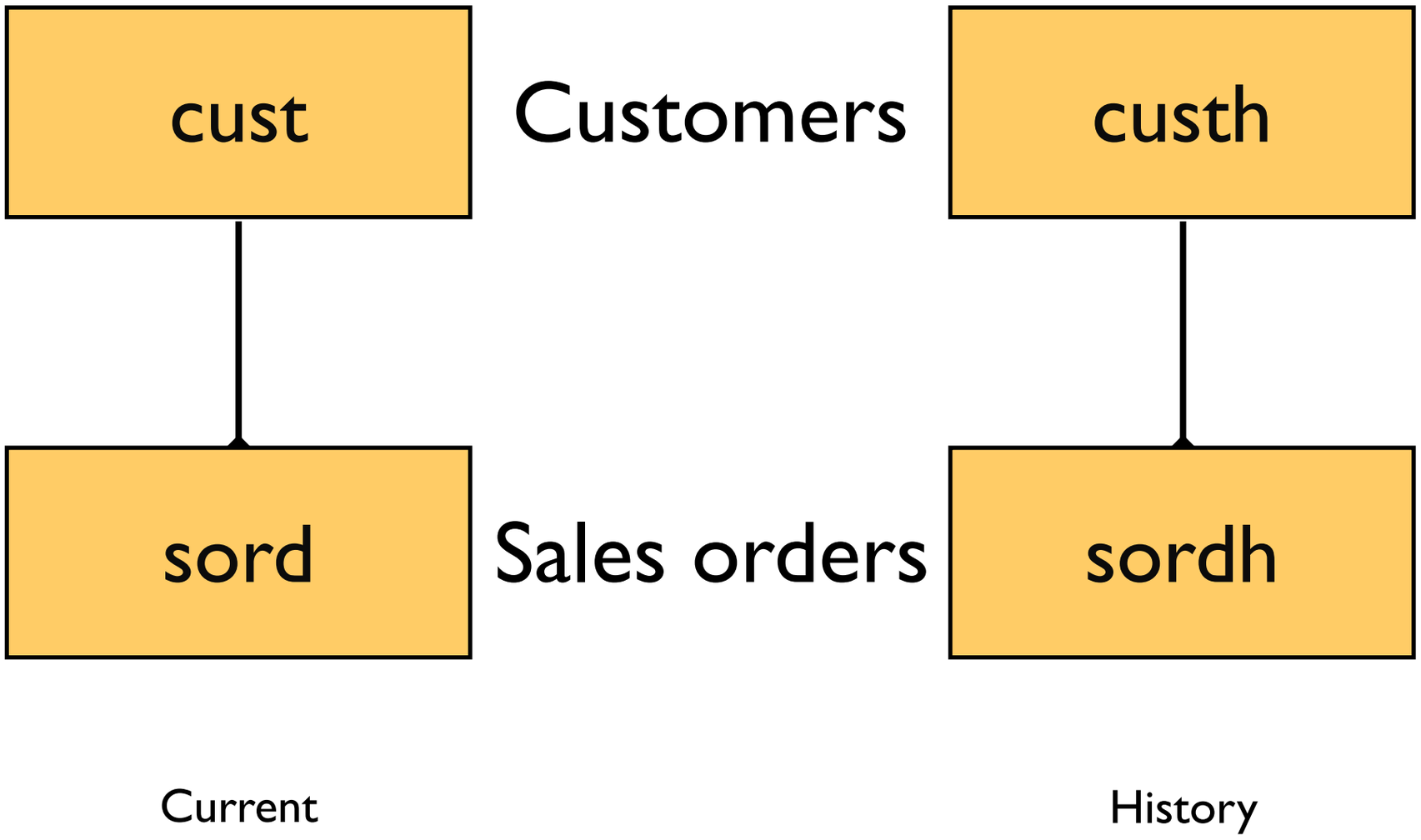
JOIN types

Common Table Expressions

Common Table Expressions

- CTE's
- WITH [RECURSION] clause
- Supported by DB2, MSSQL, Oracle, PostgreSQL, Sybase ASA/ASE and even SQLite
- Not supported by MySQL

Common Table Expressions



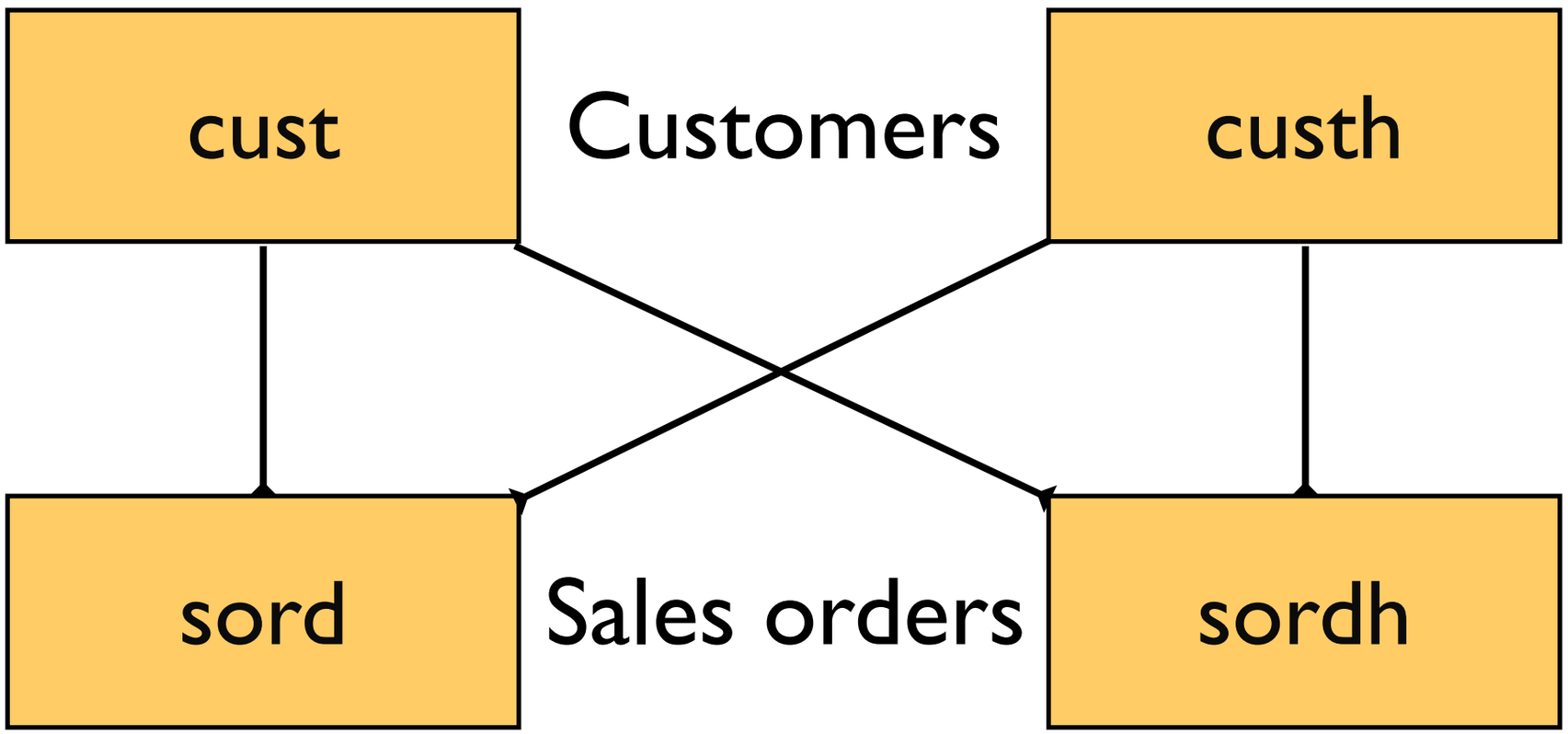
Common Table Expressions

```
SELECT territory, sum(salesamt)
FROM (
   SELECT a.territory, SUM(o.salesamt) as salesamt
   FROM sord o
   JOIN cust a ON (o.custno=a.custno)
   WHERE o.orderdate>=? AND o.orderdate<=?
   GROUP BY a.territory

   UNION ALL

   SELECT a.territory, SUM(o.salesamt) as salesamt
   FROM sordh o
   JOIN custh a ON (o.custno=a.custno)
   WHERE o.orderdate>=? AND o.orderdate<=?
   GROUP BY a.territory
) t
GROUP BY territory
```

Common Table Expressions



Current

History

Users actually pushed salesorders and/or customers independantly to the history table as well

Common Table Expressions

```
SELECT territory, sum(salesamt)
FROM (
    SELECT a.territory, SUM(o.salesamt) as salesamt
    FROM sord o
    JOIN cust a ON (o.custno=a.custno)
    WHERE o.orderdate>=? AND o.orderdate<=?
    GROUP BY a.territory

    UNION ALL

    SELECT a.territory, SUM(o.salesamt) as salesamt
    FROM sordh o
    JOIN custh a ON (o.custno=a.custno)
    WHERE o.orderdate>=? AND o.orderdate<=?
    GROUP BY a.territory

    UNION ALL

    SELECT a.territory, SUM(o.salesamt) as salesamt
    FROM sord o
    JOIN custh a ON (o.custno=a.custno)
    WHERE o.orderdate>=? AND o.orderdate<=?
    GROUP BY a.territory

    UNION ALL

    SELECT a.territory, SUM(o.salesamt) as salesamt
    FROM sordh o
    JOIN cust a ON (o.custno=a.custno)
    WHERE o.orderdate>=? AND o.orderdate<=?
    GROUP BY a.territory
) t
GROUP BY territory
```

Common Table Expressions

```
WITH customers AS (
   SELECT custno, territory FROM cust
   UNION
   SELECT custno, territory FROM custh
), orders AS (
   SELECT custno, orderdate, salesamt FROM sord
   UNION
   SELECT custno, orderdate, salesamt FROM sordh
)
SELECT a.territory, SUM(o.salesamt)
FROM orders o
JOIN customers a ON (o.custno=a.custno)
WHERE o.orderdate>=? AND o.orderdate<=?
GROUP BY a.territory
```

Common Table Expressions

Window Functions

Window Functions

- Essentially a GROUP BY per column
- You can use multiple Window Functions in the same query
- You can’t use Window Functions in the WHERE clause
  - Wrap the query in another SELECT statement

Window Functions

```
SELECT
   depname,
   empno,
   salary,
   avg(salary) OVER (PARTITION BY depname)
FROM empsalary;
```

depname	empno	salary	avg
develop	11	5200	5020.0000000000000000
develop	7	4200	5020.0000000000000000
develop	9	4500	5020.0000000000000000
develop	8	6000	5020.0000000000000000
develop	10	5200	5020.0000000000000000
personnel	5	3500	3700.0000000000000000
personnel	2	3900	3700.0000000000000000
sales	3	4800	4866.6666666666666667
sales	1	5000	4866.6666666666666667
sales	4	4800	4866.6666666666666667

Window Functions

```
SELECT
   salary,
   sum(salary) OVER (ORDER BY salary)
FROM empsalary;
```

```
 salary |  sum
--------+-------
   3500 |  3500
   3900 |  7400
   4200 | 11600
   4500 | 16100
   4800 | 25700
   4800 | 25700
   5000 | 30700
   5200 | 41100
   5200 | 41100
   6000 | 47100
```

Window Functions

- row_number()
- rank()
- dense_rank()
- percent_rank()
- cume_dist()
- ntile()
- lag()
- lead()
- first_value()
- last_value()
- nth_value()

Window Functions

```
SELECT
  firstname,
  lastname,
  row_number() OVER (ORDER BY pkColumn)
FROM customers
```

```
 firstname  |  lastname   |  row_number
------------+-------------+-------------
 Andrew     | Fuller      |           1
 Anne       | Dodsworth   |           2
 Janet      | Leverling   |           3
 Laura      | Callahan    |           4
 Margaret   | Peacock     |           5
 Michael    | Suyama      |           6
 Nancy      | Davolio     |           7
 Robert     | King        |           8
 Steven     | Buchanan    |           9
```

Window Functions

```
SELECT
   depname,
   empno,
   salary,
   rank() OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
```

depname	empno	salary	rank
develop	8	6000	1
develop	10	5200	2
develop	11	5200	2
develop	9	4500	4
develop	7	4200	5
personnel	2	3900	1
personnel	5	3500	2
sales	1	5000	1
sales	4	4800	2
sales	3	4800	2

Window Functions

Fun with Expressions

Boolean Expressions

Boolean Expressions

- 1 < 3 = **true**, 3 < 1 = **false**
- Use in ORDER BY clause

Boolean Expressions

Cylclic Order example:

Original data: `31, 30, 29, 28, 27, 26, 25`

Wanted result: `28, 27, 26, 25, 31, 30, 29`

Order starting at 28

http://stackoverflow.com/questions/20902129/cyclic-order-by-sorting-in-postgresql/20902431

Boolean Expressions

```
SELECT code > 28, code
FROM myTable
```

TRUE	31
TRUE	30
TRUE	29
FALSE	28
FALSE	27
FALSE	26
FALSE	25

```
SELECT code > 28, code
FROM myTable
ORDER BY code > 28, code DESC
```

FALSE	28
FALSE	27
FALSE	26
FALSE	25
TRUE	31
TRUE	30
TRUE	29

http://stackoverflow.com/questions/20902129/cyclic-order-by-sorting-in-postgresql/20902431

Conditional Expressions

Conditional Expressions

- CASE:

  CASE WHEN condition **THEN** result

  [**WHEN** ...]

  [**ELSE** result]

  END

Conditional Expressions

test
1
2
3

```
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
  FROM test;
```

```
 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

Conditional Expressions

- COALESCE(value [, ...])
- NULLIF(value1, value2)
- GREATEST(value [, ...])
- LEAST(value [, ...])
- GREATEST() and LEAST() not in SQL standard but common extensions

Fun with Expressions

How standard is the SQL Standard?

JDBC

Scalar Functions

JDBC Scalar Functions

- Built in abstraction for many SQL functions
  - though you won't always get the same result
- Syntax:

```
SELECT {fn functionName()} FROM tbl1
```

JDBC

Scalar Functions

Questions