SQL: Part 1

History

- Structured (English) Query Language - now SQL
  - based in Relational Calculus and Algebra
- Developed for IBM's prototype RDBMS - System R
- Early 1970's with first commercial version in 1979 or so.
- First standardized in 1986 by ANSI and then ISO
- New versions released periodically, latest 2016

Tables

sp

name	item	price
S2	P3	100
S1	P2	20
S1	P1	10
S3	P4	1000
S2	P1	11
S4	P1	9

sa

name	addr
S1	123 Any St
S3	10 State St
S2	1 Main St
S4	11 State St

Looking at a table

- Table (relation) *sp*
- The * implies all columns of the table

```
select * from sp;
 name | item | price
------+------+-------
 S2   | P3   |   100
 S1   | P2   |    20
 S1   | P1   |    10
 S3   | P4   |  1000
 S2   | P1   |    11
 S4   | P1   |     9
(6 rows)
```

“Selecting” a few columns

- Get name, item from table *sp*
- π name, item(sp)

```
select name, item
from sp;
 name | item
------+------
 S2   | P3
 S1   | P2
 S1   | P1
 S3   | P4
 S2   | P1
 S4   | P1
(6 rows)
```

Projection vs. SQL Select

- Get name from table *sp*
- π $_{\text{name}}$(sp) — not quite
- No duplicate elimination unless explicitly (or implicitly) specified

```
select name from sp;
 name
------
 S2
 S1
 S1
 S3
 S2
 S4
(6 rows)
```

Projection vs. SQL Select

- Get (unique) name from table *sp*
- π name(sp)
- No duplicate elimination unless explicitly (or implicitly) specified
- No ordering either ….

```
select distinct name
from sp;
 name
------
 S1
 S2
 S4
 S3
(4 rows)
```

Relational σ and WHERE

- Get all rows (tuples) for item P1
- The * implies all columns of the table
- $\sigma_{\text{item='P1'}}(sp)$

```
select *
from sp
where item = 'P1';
 name | item | price
------+------+-------
 S1   | P1   |    10
 S2   | P1   |    11
 S4   | P1   |     9
(3 rows)
```

What kind of conditions

- comparison: =, <, >, <>, >=, <=
  - C1 between v1 and v2 is same as saying C1 >= v1 and C1 <= v2
- string comparison allows “LIKE” with limited pattern matching
- boolean: AND, OR, NOT
- inclusion, exclusion: IN, NOT IN

WHERE clause example

- Get all rows (tuples) for items P1 and P2 where supplier name ends in 1.

```
select *
from sp
where item in ('P1','P2') and
      name like '%1';
 name | item | price
------+------+-------
 S1   | P2   |    20
 S1   | P1   |    10
(2 rows)
```

JOIN

- Tables SP, and, SA were derived from decomposition of the Supplier table: S(name, address, item, price)
- So to get back the original S, we need to join SP and SA on the common column “name”

```
select *
from sa join sp on sa.name = sp.name;
 name |     addr      | name | item | price
------+---------------+------+------+-------
 S2   | 1 Main St     | S2   | P3   |   100
 S1   | 123 Any St    | S1   | P2   |    20
 S1   | 123 Any St    | S1   | P1   |    10
 S3   | 10 State St   | S3   | P4   |  1000
 S2   | 1 Main St     | S2   | P1   |    11
 S4   | 11 State St   | S4   | P1   |     9
(6 rows)
```

JOIN - alternative syntax

- A common way to specify a join is via a condition in the WHERE clause connecting the two (or more) tables
- can refer to columns of just one table using <tablename>.*
- column names when referenced must be uniquely identifiable

```
select sa.*, item, price
from sa, sp
where sa.name = sp.name;
 name |    addr      | item | price
------+--------------+------+-------
 S2   | 1 Main St    | P3   |   100
 S1   | 123 Any St   | P2   |    20
 S1   | 123 Any St   | P1   |    10
 S3   | 10 State St  | P4   |  1000
 S2   | 1 Main St    | P1   |    11
 S4   | 11 State St  | P1   |     9
(6 rows)
```

Example: JOIN + WHERE

- Get name and address of suppliers of item P1

```
select sa.*
from sa, sp
where sa.name = sp.name and
      item = 'P1';
 name |     addr
------+--------------
 S1   | 123 Any St
 S2   | 1 Main St
 S4   | 11 State St
(3 rows)
```

Expressions

```
select sa.name || ', ' || addr as Supplier,
       item, price * 0.9 as DiscountPrice
from sa, sp
where sa.name = sp.name and price * 0.9 > 10;
    supplier      | item | discountprice
------------------+------+---------------
 S1, 123 Any St   | P2   |          18.0
 S3, 10 State St  | P4   |         900.0
 S2, 1 Main St    | P3   |          90.0
(3 rows)
```

- Pretty much any (scalar) function/operator that's applicable to the type of data is available in SQL
- If not, you can create your own function
- WHERE conditions can also have expressions
- "as": alias ("rename operator") to give a new name to a column or expression

Subqueries

- Find suppliers who supplied part P1
- The IN operator checks if the value(s) are in the *subquery*
- Can be powerful, if you can keep track of the meaning
- If the subquery has no references to "outside", then it's called *uncorrelated*

```
select *
from sa
where name in
            (select name
            from sp
            where item = 'P1');
```

```
 name |     addr
------+--------------
 S1   | 123 Any St
 S2   | 1 Main St
 S4   | 11 State St
(3 rows)
```

Subqueries

- Find suppliers who supplied part P1
- Another way to express the same request using EXISTS and a *correlated* subquery
- EXISTS: the relation specified is a non-empty relation
- Many possibilities: use intuition and keep it simple

```
select *
from sa
where exists
          (select name
          from sp
          where sp.name = sa.name
          and item = ‘P1');
```

```
 name  |     addr
-------+--------------
 S1    | 123 Any St
 S2    | 1 Main St
 S4    | 11 State St
(3 rows)
```

Simple Alternative?

- Subqueries can on many occasions be expressed as simple JOIN query

```
select sa.*
from sa, sp
where sa.name = sp.name
      and item = ‘P1';
```

```
 name |    addr
------+-------------
 S1   | 123 Any St
 S2   | 1 Main St
 S4   | 11 State St
(3 rows)
```

Subqueries

- [IN | NOT IN] <subq> - set membership check
- [EXISTS | NOT EXISTS | UNIQUE | NOT UNIQUE] <subq>
- <value> *relop* [ANY | ALL] <subq>
  - *relop*s are >, <, =, etc.
  - <subq> must return 1 row unless ANY / ALL used

Cross Product

- Given a good relational model, need for cross products should be rare
- Mostly for understanding SQL processing
- adding a proper WHERE condition makes a cross product into a join

```
select *
from sa, sp;
 name |    addr     | name | item | price
------+-------------+------+------+-------
 S1   | 123 Any St  | S2   | P3   |   100
 S3   | 10 State St | S2   | P3   |   100
 S2   | 1 Main St   | S2   | P3   |   100
 S4   | 11 State St | S2   | P3   |   100
 S1   | 123 Any St  | S1   | P2   |    20
 S3   | 10 State St | S1   | P2   |    20
 S2   | 1 Main St   | S1   | P2   |    20
 S4   | 11 State St | S1   | P2   |    20
 S1   | 123 Any St  | S1   | P1   |    10
 S3   | 10 State St | S1   | P1   |    10
 S2   | 1 Main St   | S1   | P1   |    10
 S4   | 11 State St | S1   | P1   |    10
 S1   | 123 Any St  | S3   | P4   |  1000
 S3   | 10 State St | S3   | P4   |  1000
 S2   | 1 Main St   | S3   | P4   |  1000
 S4   | 11 State St | S3   | P4   |  1000
 S1   | 123 Any St  | S2   | P1   |    11
 S3   | 10 State St | S2   | P1   |    11
 S2   | 1 Main St   | S2   | P1   |    11
 S4   | 11 State St | S2   | P1   |    11
 S1   | 123 Any St  | S4   | P1   |     9
 S3   | 10 State St | S4   | P1   |     9
 S2   | 1 Main St   | S4   | P1   |     9
 S4   | 11 State St | S4   | P1   |     9
(24 rows)
```

Review

SELECT [options] column_expression_list or "*"
FROM table_expression_list
WHERE condition

- We are SELECTing a set of expressions forming columns of the output
  - RA calls it projection
- The rows that contribute to it come from the tables, or equivalent, in the FROM clause
  - RA cross product (aka cross join)
- But only the rows that meet the WHERE condition
  - RA calls it selection
  - Some conditions may be specifying the RA natural join

Conceptually

- Compute the cross product of relations in FROM
- Keep the rows that meet the WHERE conditions
- WHERE conditions comprise
  - join conditions (converting cross product to joins)
  - conditions on attributes, including subqueries
  - (subqueries are a kind of join…)
- Compute the expressions using values from rows (and literals and maths and scalar function and …)

DISTINCT option

SELECT [options] column_expression_list or "*"
FROM table_expression_list
WHERE condition

- One of the options is "DISTINCT"
- Applied after all the output rows are formed
- Removes duplicate rows in the output (and converts it into a true *relation*)
- Not used much
  - information loss
  - shouldn't need it much in a well-defined schema
  - aggregation queries don't need it

DISTINCT option

- Get all the distinct items being supplied

```
select distinct item
from sp;

 item
------
 P2
 P1
 P4
 P3
(4 rows)
```

SET operations

- UNION [ALL]
- INTERSECT [ALL]
- EXCEPT [ALL] (set minus)
- The table_expressions have to be compatible: i.e. must have same number of columns and each corresponding column must have compatible type
- without ALL, no duplicates (even if original table_expressions have them)
- with ALL
  - all rows kept in UNION
  - matching number of duplicates kept in INTERSECT
  - matching number of duplicates discarded in EXCEPT

INTERSECT

- Suppliers supplying item P1 and P2
  - Suppliers for P1
  - intersect
  - Suppliers for P2

```
select sa.*
from sa, sp
where sa.name = sp.name
      and item = 'P1'
intersect
select sa.*
from sa, sp
where sa.name = sp.name
      and item = ‘P2’;

 name |    addr
------+-------------
 S1   | 123 Any St
(1 row)
```

EXCEPT - set minus

- Suppliers supplying item P1 but not P2
  - Suppliers for P1
  - except
  - Suppliers for P2

```
select sa.*
from sa, sp
where sa.name = sp.name
      and item = 'P1'
except
select sa.*
from sa, sp
where sa.name = sp.name
      and item = ‘P2’;

 name |    addr
------+-------------
 S2   | 1 Main St
 S4   | 11 State St
(2 rows)
```

UNION

- Suppliers supplying item P1 or P2
  - Suppliers for P1
    - union
  - Suppliers for P2
- Suppliers for either item i.e.
  - item IN (P1, P2) or
  - item = P1 or item = P2.

```
select sa.*
from sa, sp
where sa.name = sp.name
      and item = 'P1'
union
select sa.*
from sa, sp
where sa.name = sp.name
      and item = ‘P2’;
```

```
 name |     addr
------+--------------
 S2   | 1 Main St
 S1   | 123 Any St
 S4   | 11 State St
(3 rows)
```

UNION

- Suppliers supplying item P1 or P2
  - Suppliers for P1
  - union
  - Suppliers for P2
- Suppliers for either item i.e.
  - item IN (P1, P2) or
  - item = P1 or item = P2.

```
select distinct sa.*
from sa, sp
where sa.name = sp.name
      and item in ('P1', 'P2')
;
 name |     addr
------+--------------
 S2   | 1 Main St
 S1   | 123 Any St
 S4   | 11 State St
(3 rows)
```

ORDER BY

- Notice that almost none of the results in our examples are “ordered”
- This stems from the SET foundations of RDBMS
- ORDER BY clause helps us order results in the output

```
select sa.*, item, price
from sa, sp
where sa.name = sp.name;
```

```
 name |     addr      | item | price
------+---------------+------+-------
 S2   | 1 Main St     | P3   |   100
 S1   | 123 Any St    | P2   |    20
 S1   | 123 Any St    | P1   |    10
 S3   | 10 State St   | P4   |  1000
 S2   | 1 Main St     | P1   |    11
 S4   | 11 State St   | P1   |     9
(6 rows)
```

ORDER BY

- ORDER BY clause helps us order results in the output
- Options are ASC (default), DESC, NULLS FIRST, NULLS LAST

```
select sa.*, item, price
from sa, sp
where sa.name = sp.name
order by sa.name, item;
```

```
 name |     addr      | item | price
------+---------------+------+-------
 S1   | 123 Any St    | P1   |    10
 S1   | 123 Any St    | P2   |    20
 S2   | 1 Main St     | P1   |    11
 S2   | 1 Main St     | P3   |   100
 S3   | 10 State St   | P4   |  1000
 S4   | 11 State St   | P1   |     9
(6 rows)
```