SQL Queries (iii)

- Sets in SQL
- Bags in SQL
- The IN Operator
- The **EXISTS** Operator
- Quantifiers
- Union, Intersection, Difference
- Division
- Selection with Aggregation

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [0/13]

>>

>>



The relational model is set-based

Set literals are written as $(expr_1, expr_2, ...)$ (each $expr_i$ yields an atomic value)

SQL query results are (more or less) sets of tuples or atomic values

Examples:

```
-- set literals
(1,2,3) ('a','b','c','d')
-- set of atomic values
(select salary from Employees)
-- set of tuple values
(select id, name from Employees)
```

SQL provides a variety of set-based operators: \in , \cup , \cap , -, /, \exists , \forall , ...

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [1/13]



SQL query results are actually bags (multisets), allowing duplicates, e.g.

```
select age from Students;
-- yields (18,18,18,...19,19,19,19,...20,20,20,...)
```

Can convert bag to set (eliminate duplicates) using **DISTINCT**, e.g.

```
select distinct age from Students;
```

SQL set operations UNION, INTERSECT, EXCEPT ...

- yield sets by default (i.e. eliminate duplicates)
- can produce bags with keyword ALL (e.g. UNION ALL)

```
(1,2,3) UNION (2,3,4) yields (1,2,3,4) (1,2,3) UNION ALL (2,3,4) yields (1,2,3,2,3,4)
```

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [2/13]

<< \ \ >>

♦ The IN Operator

Tests whether a specified tuple is contained in a relation (i.e. t∈R)

tuple IN relation is true iff the tuple is contained in the relation.

Conversely for tuple **NOT IN** relation.

Syntax:

```
SELECT *
FROM R
WHERE R.a IN (SELECT x FROM S WHERE Cond)
-- assume multiple results
```

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [3/13]

The IN Operator (cont)

Example: Find the name and brewer of beers that John likes.

```
SELECT name, brewer
FROM
      Beers
WHERE name IN
         (SELECT beer
          FROM Likes
          WHERE drinker = 'John');
                         brewer
       name
80/-
                    | Caledonian
 Bigfoot Barley Wine | Sierra Nevada
 Pale Ale
                     | Sierra Nevada
 Three Sheets
                     | Lord Nelson
```

Subquery = "What are the names of the beers that John likes?"

(This and subsequent beer queries use an older smaller version of the Beer database)

COMP3311 20T3 \$ SQL Queries (iii) \$ [4/13]

The EXISTS Operator

EXISTS (*relation*) is true iff the relation is non-empty.

Example: Find the beers that are the unique beer by their manufacturer.

A subquery that refers to values from a surrounding query is called a correlated subquery.

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [5/13]



ANY and **ALL** behave as existential and universal quantifiers respectively.

Example: Find the beers sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >=
    ALL(SELECT price FROM sells);
```

Beware: in common use, "any" and "all" are often synonyms.

E.g. "I'm better than any of you" vs. "I'm better than all of you".

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [6/13]

<<



Union, Intersection, Difference

SQL implements the standard set operations

R1UNION R2 set of tuples in either R1 or R2

R1 INTERSECT R2 set of tuples in both R1 and R2

R1 EXCEPT R2 set of tuples in R1 but not R2

R1 and R2 must be union-compatible (i.e. same schema)

Union and intersection semantics are straightforward.

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [7/13]

Union, Intersection, Difference (cont)

Example: Find the drinkers and beers such that the drinker likes the beer and frequents a bar that sells it.

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [8/13]

Union, Intersection, Difference (cont)

Set difference is implemented by **EXCEPT**

R	
Α	В
1	'a'
2	'b'
3	'a'

<u>S</u>	
A	В
1	'a'
1	'b'
2	'a'

R except S	
A	В
2	'b'
3	'a'

o except n	
A	В
1	'b'
2	'a'

S avcent R

Semantics of set difference: R except $S = \{x \in R, \text{ where } x \notin S\}$

COMP3311 20T3 ◊ SQL Queries (iii) ◊ [9/13]

Division

Division aims to find values in one table that occur in conjunction with all values in another table:

R	
Α	В
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'

S		
	В	
	'a'	
	'b'	

R/S
Α
1
2

Arises in queries like "Find Xs that are related to all Ys / every Y"

COMP3311 20T3 \diamond SQL Queries (iii) \diamond [10/13]



Not all SQL implementations provide a division operator

But can be achieved by combination of existing operations

Example: Find bars that each sell all of the beers Justin likes.

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [11/13]

Selection with Aggregation

Selection clauses can contain aggregation operations.

Example: What is the average price of New?

- the bag semantics of SQL gives the correct result here
- the price for New in all hotels will be included, even if two hotels sell it at the same price
- if we used set semantics, we'd get the average of all the different prices for New.

COMP3311 20T3 ◊ SQL Queries (iii) ◊ [12/13]

Selection with Aggregation (cont)

If we want set semantics, can force using **DISTINCT**.

Example: How many different bars sell beer?

```
SELECT COUNT(DISTINCT bar)
FROM Sells;

count
-----
6
```

Without **DISTINCT**, counts number of entries in the **Sells** table.

Aggregation operators on numbers: SUM, AVG, MIN, MAX,

COMP3311 20T3 ♦ SQL Queries (iii) ♦ [13/13]

Produced: 4 Oct 2020