Introduction to SQL-Part2
(Structured Query Language)

Introduction to Databases

Sina Meraji

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom
and Renee Miller for material in these slides

OTHER CONCEPTS

NULL values in SQL

- Values allowed to be NULL
  - Explicitly stored in relations
  - Result of outer joins
- Possible meanings
  - Not present (homeless man's address)
  - Unknown (Julian Assange's address)
- Effect: "poison"
  - Arithmetic: unknown value takes over expression
  - Conditionals: ternary logic (TRUE, FALSE, UNKNOWN)
  - Grouping: "not present"

Effect of NULL in expressions

- Arithmetic: NaN (Not a Number)
  - NULL*0 → NULL
  - NULL – NULL → NULL
- Logic: TRUE, FALSE, NULL
  - NULL OR FALSE → NULL
  - NULL OR TRUE → TRUE
  - NULL AND TRUE → NULL
  - NULL AND FALSE → FALSE
  - NOT NULL → NULL

Ternary logic tricks:

TRUE = 1
FALSE = 0
NULL = ½

AND = min(…)
OR = max(…)
NOT = 1-x

Effects of NULL on grouping

- Short version: complicated
  - Usually, “not present”
- COUNT
  - COUNT(R.*) = 2 COUNT(R.x) = 1
  - COUNT(S.*) = 1 COUNT(S.x) = 0
  - COUNT(T.*) = 0 COUNT(T.x) = 0
- Other aggregations (e.g. MIN/MAX)
  - MIN(R.x) = 1 MAX(R.x) = 1
  - MIN(S.x) = NULL MAX(S.x) = NULL
  - MIN(T.x) = NULL MAX(T.x) = NULL

R

x
⊥
1

S

x
⊥

T

x

SET Queries: Union, Intersection, Difference

- Operations on pairs of subqueries
- Expressed by the following forms
  - (<subquery>) **UNION [ALL]** (<subquery>)
  - (<subquery>) **INTERSECT [ALL]** (<subquery>)
  - (<subquery>) **EXCEPT [ALL]** (<subquery>)
- All three operators are **set-based**
  - Adding ‘**ALL**’ keyword forces **bag semantics** (duplicates allowed)
- Another solution to the join selectivity problem!

```
(SELECT R.x FROM R JOIN S ON R.x=S.x)
 UNION
(SELECT R.x FROM R JOIN T ON R.x=T.x)
```

Example: Union

➔“Find all first names and surnames of employees”

SELECT FirstName **AS** Name **FROM** Employee

UNION

SELECT Surname **AS** Name **FROM** Employee

Duplicates are removed, unless the **ALL** option is used:

SELECT FirstName **AS** Name **FROM** Employee

UNION ALL

SELECT Surname **AS** Name **FROM** Employee

Example: Intersection

➔“Find surnames of employees that are also first names”

SELECT FirstName **AS** Name **FROM** Employee

INTERSECT

SELECT Surname **AS** Name **FROM** Employee

equivalent to:

SELECT E1.FirstName **AS** Name

FROM Employee E1, Employee E2

WHERE E1.FirstName = E2.Surname

Example: Difference

➔“Find the surnames of employees that are not first names”

SELECT SurName **AS** Name **FROM** Employee

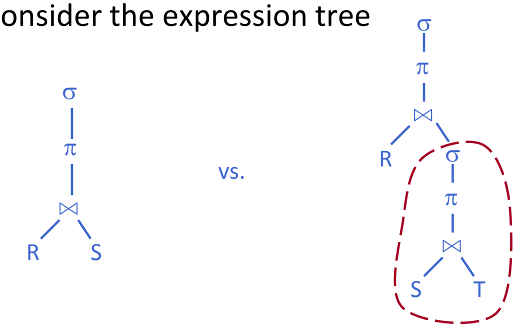
EXCEPT

SELECT FirstName **AS** Name **FROM** Employee

(Can also be represented with a nested query. See later)

Nested queries

- Scary-looking syntax, simple concept
  - Treat one query’s output as input to another query
  - Inner schema determined by inner SELECT clause
- Consider the expression tree



Nested queries – uses

- Explicit join ordering
  - `FROM (A join B)` is a (very simple) query to run first
- Input relation for a set operation
  - Union, intersect, difference
- Input relation for a larger query
  - Appears in FROM clause
  - Usually joined with other tables (or other nested queries)

  => `FROM A, (SELECT …) B WHERE …`

  => Explicit join ordering is a degenerate case

Nested queries – more uses

- Conditional relation expression
  - Dynamic list for [NOT] IN operator

```
=> WHERE (E.id,S.name)
    IN (SELECT id,name FROM …)
```

  - Special [NOT] EXISTS operator

```
=> WHERE NOT EXISTS (SELECT * FROM …)
```

- Scalar expression
  - Must return single tuple (usually containing a single attribute)

```
=> 0.13*(SELECT sum(value)
          FROM Sales WHERE taxable)
=> S.value > (SELECT average(S.value)
               FROM Sales S)
```

List comparisons: ANY, ALL, [NOT] IN

- Compares a value against many others
  - List of literals
  - Result of nested query

Let op be any comparator (>, <=, !=, etc.)

- x op ANY (a, b, c)

  = x op a OR x op b OR x op c
- x op ALL (a, b, c)

  = x op a AND x op b AND x op c
- [NOT] IN
  - **x NOT IN (…)** equivalent to **x != ALL(…)**
  - **x IN (…)** equivalent to **x = ANY(…)**

ANY is ∃(exist), ALL is ∀(for each) (English usage often different!)

Example: Simple Nested Query

➔“Find the names of employees who work in departments in London”

SELECT FirstName, Surname

FROM Employee

WHERE Dept = **ANY**(

SELECT DeptName

FROM Department

WHERE City = ‘London’)

equivalent to:

SELECT FirstName, Surname

FROM Employee, Department D

WHERE Dept = DeptName **AND** D.City = ‘London’

Example: Another Nested Query

➔“Find employees of the Planning department, having the same first name as a member of the Production department”

SELECT FirstName,Surname

FROM Employee

WHERE Dept = ‘Plan’ **AND** FirstName = **ANY** (

SELECT FirstName **FROM** Employee **WHERE** Dept = ‘Prod’)

equivalent to:

SELECT E1.FirstName,E1.Surname

FROM Employee E1, Employee E2

WHERE E1.FirstName=E2.FirstName **AND** E2.Dept=‘Prod’ **AND** E1.Dept=‘Plan’

Example: Negation with Nested Query

➔“Find departments where there is no employee named Brown”

SELECT DeptName

FROM Department

WHERE DeptName **<> ALL** (

SELECT Dept **FROM** Employee **WHERE** Surname = ‘Brown’)

equivalent to:

SELECT DeptName **FROM** Department

EXCEPT

SELECT Dept **FROM** Employee **WHERE** Surname = ‘Brown’

Operators IN and NOT IN

- Operator **IN** is a shorthand for **= ANY**

```
SELECT FirstName, Surname
FROM Employee
WHERE Dept IN (
      SELECT DeptName FROM Department WHERE City = ‘London’)
```

- Operator **NOT IN** is a shorthand for **<> ALL**

```
SELECT DeptName
FROM Department
WHERE DeptName NOT IN (
      SELECT Dept FROM Employee WHERE Surname = ‘Brown’)
```

max, min as Nested Queries

“Find the department of the employee earning the highest salary”

with max:

```
SELECT Dept FROM Employee
WHERE Salary IN (SELECT max(Salary) FROM Employee)
```

without max:

```
SELECT Dept FROM Employee
WHERE Salary >= ALL (SELECT Salary FROM Employee)
```

Operator: [NOT] EXISTS

- Checks whether a subquery returned results

“Find all persons who have the same first name and surname with someone else (synonymous folks) but different tax codes”

```
SELECT * FROM Person P
WHERE EXISTS (
      SELECT * FROM Person P1
      WHERE P1.FirstName = P.FirstName AND P1.Surname =
                  P.Surname AND P1.TaxCode <> P.TaxCode)
```

Operator: [NOT] EXISTS (cont.)

“Find all persons who have no synonymous persons”

```
SELECT * FROM Person P
WHERE NOT EXISTS (
      SELECT * FROM Person P1
      WHERE P1.FirstName = P.FirstName AND P1.Surname =
                  P.Surname AND P1.TaxCode <> P.TaxCode)
```

Tuple Constructors

- The comparison within a nested query may involve several attributes bundled into a tuple
- A tuple constructor is represented in terms of a pair of angle brackets
  - The previous query can also be expressed as:

```
SELECT * FROM Person P
WHERE <FirstName,Surname> NOT IN (
      SELECT FirstName,Surname
      FROM Person P1
      WHERE P1.TaxCode <> P.TaxCode)
```

Comments on Nested Queries

- Use of nesting

  (-) may produce less declarative queries

  (+) often results in improved readability
- Complex queries can become very difficult to understand
- The use of variables must respect scoping conventions:
  - a variable can be used only within the query where it is defined, OR
  - within a query that is recursively nested within the query where it is defined

What's next?

- The Data Definition Language (DDL)
  - Subset of SQL used to manage schema
  - CREATE, ALTER, RENAME, DROP
  - Data types
- Data Manipulation Language (DML)
  - Subset of SQL used to manipulate data
  - INSERT, UPDATE, DELETE