SQL – Subqueries in WHERE Clause and Quantifiers

CS 4750
Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.5.3]

Subqueries in WHERE

Return a single value or a relation that can be compared to another value in a `WHERE` clause

Find the name(s) of the employee(s) who earn the highest salary for each job

`HW_emp(empno, ename, job, sal)`

```
SELECT E1.ename
FROM HW_emp E1
WHERE E1.sal = (SELECT MAX(E2.sal)
                FROM HW_emp E2
                WHERE E1.job = E2.job);
```

"Correlated" query

[Refer to slide 14, https://www.cs.virginia.edu/~up3f/cs4750/slides/4750meet16-17-SQL-subqueries.pdf]

Subqueries in WHERE

Return a single value or a relation that can be compared to another value in a `WHERE` clause

Find the name(s) of the instructor(s) who earn the highest salary

`Instructor (dept_name, ID, name, salary)`

```
SELECT name
FROM   instructor
WHERE  salary = (SELECT MAX(salary)
                 FROM instructor);
```

"Uncorrelated" query

Subqueries in WHERE

Can be used to evaluate existential or universal quantifiers

At least one element

All elements

Any relational operators

Values of a column

ANY (∃)	`SELECT … WHERE constant > ANY (subquery);`
ALL (∀)	`SELECT … WHERE constant > ALL (subquery);`
IN	`SELECT … WHERE attribute IN (subquery);`
NOT IN	`SELECT … WHERE attribute NOT IN (subquery);`
EXISTS	`SELECT … WHERE EXISTS (subquery);`
NOT EXISTS	`SELECT … WHERE NOT EXISTS (subquery);`

Rows

Existential Quantifier (EXISTS)

```
account (account_number, branch_name, balance)
borrower (customer_name, loan_number)
branch (branch_name, branch_city, assets)
depositor (customer_name, account_number)
loan (loan_number, branch_name, amount)
```

Find the names of the branches that have some customers who have both loan(s) and account(s) from the bank

```
SELECT T1.branch_name
FROM account T1
WHERE EXISTS
      (SELECT *
       FROM depositor D NATURAL JOIN borrower B
       WHERE D.account_number = T1.account_number);
```

tuple(s)

Correlated

Existential Quantifier (IN)

```
account (account_number, branch_name, balance)
borrower (customer_name, loan_number)
branch (branch_name, branch_city, assets)
depositor (customer_name, account_number)
loan (loan_number, branch_name, amount)
```

Find the names of the branches that have some customers who have both loan(s) and account(s) from the bank

```
SELECT T1.branch_name
FROM account T1
WHERE T1.account_number IN
      (SELECT D.account_number
       FROM depositor D NATURAL JOIN borrower B);
```

attribute

attribute

Decorrelated

IN and Set Membership

takes (ID, course_id, sec_id, semester, year, grade)

I need to know who takes the following courses:
CS-101, CS-315, BIO-101

```
SELECT *
FROM takes
WHERE course_id IN
      ('CS-101', 'CS-315', 'BIO-101');
```

attribute

Set of literal values

```
SELECT *
FROM takes
WHERE course_id='CS-101' OR course_id='CS-315' OR
      course_id='BIO-101';
```

Existential Quantifier

```
Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)
```

Find the names of the customers who made some purchases that are > $1000

```
SELECT DISTINCT T1.name
FROM Customer T1 NATURAL JOIN purchase T2
WHERE 1000 < T2.price
```

Existential Quantifier (ANY)

```
Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)
```

Find the names of the customers who made some purchases that are > $1000

```
SELECT DISTINCT T1.name
FROM Customer T1
WHERE 1000 < ANY
      (SELECT price
       FROM purchase T2
       WHERE T1.custId = T2.custId)
```

constant

Correlated

Note: another way to solve (with subquery)

Universal Quantifier

```
Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)
```

Find the names of the customers who made purchases that are > $1000 only

Find the names of the customers such that all their purchases are > $1000

There does not exist any purchases the customer made where price <= $1000

Universal Quantifier (NOT IN)

Solution #1

Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)

Find the names of the customers who made purchases that are > $1000 only

Step 1: Find the customers who make some purchases <= 1000

```
SELECT DISTINCT T1.name
FROM Customer T1
WHERE T1.custId IN (SELECT T2.custId
                    FROM Purchase T2
                    WHERE T2.price <= 1000)
```

Universal Quantifier (NOT IN)

Find the names of the customers who made purchases that are > $1000 only

Step 1: Find the customers who make some purchases <= 1000

```
SELECT DISTINCT T1.name
FROM Customer T1
WHERE T1.custId IN (SELECT T2.custId
                    FROM Purchase T2
                    WHERE T2.price <= 1000)
```

Step 2: Find all customers who make purchase > 1000

```
SELECT DISTINCT T1.name
FROM Customer T1 NATURAL JOIN Purchase P
WHERE T1.custId NOT IN (SELECT T2.custId
                        FROM Purchase T2
                        WHERE T2.price <= 1000)
```

Universal Quantifier (NOT EXISTS)

Solution #2

Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)

Find the names of the customers who made purchases that are > $1000 only

```
SELECT DISTINCT T1.name
FROM Customer T1 NATURAL JOIN purchase
WHERE NOT EXISTS (SELECT *
                  FROM purchase T2
                  WHERE T1.custId = T2.custId AND
                        T2.price <= 1000)
```

Universal Quantifier (ALL)

Solution #3

Product(pid, name, cid)
Company(cid, cname, city)
Customer(custId, name, city)
Purchase(purchase_date, pid, custId, quantity, price)

Find the names of the customers who made purchases that are > $1000 only

```
SELECT DISTINCT T1.name
FROM Customer T1 NATURAL JOIN Purchase
WHERE 1000 < ALL (SELECT T2.price
                  FROM purchase T2
                  WHERE T1.custId = T2.custId)
```

Let's Try 1: NOT EXISTS

```
account (account_number, branch_name, balance)
borrower (customer_name, loan_number)
branch (branch_name, branch_city, assets)
depositor (customer_name, account_number)
loan (loan_number, branch_name, amount)
```

Find the name(s) of the customer(s) who has a loan but does not have an account. Do not repeat the customer name.

```
SELECT DISTINCT B.customer_name
FROM borrower AS B
WHERE NOT EXISTS
      (SELECT *
       FROM depositor AS D
       WHERE B.customer_name = D.customer_name);
```

tuple(s)

Correlated

Let's Try 2: NOT IN

```
account (account_number, branch_name, balance)
borrower (customer_name, loan_number)
branch (branch_name, branch_city, assets)
depositor (customer_name, account_number)
loan (loan_number, branch_name, amount)
```

Find the name(s) of the customer(s) who has a loan but does not have an account. Do not repeat the customer name.

```
SELECT DISTINCT B.customer_name
FROM borrower AS B
WHERE B.customer_name NOT IN
    (SELECT D.customer_name
     FROM depositor AS D);
```

attribute

attribute

Decorrelated

Note: another way to solve (equivalent to previous slide)

Wrap-Up

- Subqueries in WHERE
- Internal interpretation of nested queries
- Many ways to express queries

Note:

- Avoid nested queries – if aiming for speed
- Be careful of semantics of nested queries
  - Correlated vs. Uncorrelated

What's next?

- Advanced SQL