SQL - Basics

Davood Rafiei

SQL Overview

- Structured Query Language
  - standard query language for relational system
  - developed in IBM Almaden (system R)
- Some features
  - Declarative: specify the properties that should hold in the result, not how to obtain the result
  - Complex queries have procedural elements
  - Set/Bag semantics
- International Standards
  - SQL1 (1986)
  - SQL2 (SQL-92)
  - SQL3 (SQL-99)
  - Later extensions in 2003, 2006, 2008, 2011, 2016

SQL Overview (cont)

- Data definition component
  - CREATE TABLE table-name (col-defs, constraints)
  - DROP TABLE table-name
  - ALTER TABLE table-name action
    - modifies the definition of a table where action is
      - ADD (col-def)
      - MODIFY (col-def)
      - ADD costraint
      - etc.
- Data update component
  - INSERT INTO table-name …
  - DELETE FROM table-name …
  - UPDATE table-name …
- Query Language

Example Tables

branch (bname, address, city, assets)
customer(cname, street, city)
deposit(accno, cname, bname, balance)
loan(accno, cname, bname, amount)

```
CREATE TABLE branch (
    bname       CHAR(20),
    address     CHAR(30),
    city        CHAR(15),
    asset       INT
);
```

SQLite command

(in SQLite, a semicolon always marks the end of a SQL statement.)

Simple Queries

branch (bname, address, city, assets)

- Find the names of all branches with assets greater than $2,500,000.

```
SELECT bname
FROM   branch
WHERE  assets > 2500000
```

- Find the names of all branches in Edmonton with assets greater than $2,500,000.

```
SELECT bname
FROM   branch
WHERE  assets>2500000 AND city='Edmonton'
```

- Simple predicates can be combined using AND, OR, NOT.

Querying two Relations

customer(cname, street, city)

deposit(accno, cname, bname, balance)

- List the name and the city of every customer who has an account with balance over $2,000.

```
SELECT customer.cname, city
FROM   customer, deposit
WHERE  balance > 2000
AND    customer.cname = deposit.cname
```

Queries With Tuple Variables

loan(accno, cname, bname, amount)
deposit(accno, cname, bname, balance)

- Find customers who have both loans and deposits.

```
SELECT   loan.cname
FROM     loan, deposit
WHERE    loan.cname = deposit.cname
```

- Equivalently using tuple variables:

```
SELECT   l.cname
FROM     loan l, deposit d
WHERE    l.cname = d.cname
```

- Range variables are really needed if the same relation appears twice in the FROM clause.

A Simple Evaluation Alg.

```
SELECT   …
FROM     R1 r1, R2 r2, …
WHERE    C
```

- *Tuple variables* r1, r2, … respectively range over rows of R1, R2, ...
- Evaluation strategy:
  - FROM clause produces Cartesian product of listed tables
  - WHERE clause produces table containing only rows satisfying condition
  - SELECT clause retains listed columns

```
for every tuple r1 in R1, r2 in R2, …
    let r := r1, r2, …
    if C(r) then
        output the desired columns
```

From SQL to Relational Algebra

```
SELECT l.cname
FROM   loan l, deposit d
WHERE  l.cname = d.cname
```

Equivalent to:

$$temp = \rho_{(a1,cn1,bn1,b1,a2,cn2,bn2,a2)}(loan \times deposit)$$

$$\pi_{cn1}\, \sigma_{cn1 = cn2}(temp)$$

This is a simple evaluation algorithm for SELECT.

Queries With Set/Bag Results

- Find all cities of customers.

  -
    ```
    SELECT city
     FROM customer
    ```
- Result?
- To get rid of duplicates, we need

  -
    ```
    SELECT DISTINCT city
    FROM customer
    ```

Duplicates

- Duplicate rows not allowed in a relation
- However, duplicate elimination from query result is costly and not automatically done; it must be explicitly requested:

```
SELECT DISTINCT …..
FROM …..
```

Working with Strings

- Equality and comparison operators apply to strings (based on lexical ordering)
  - E.g. WHERE cn*ame* < ‘P’
- Concatenate operator applies to strings
  - E.g. WHERE *bname* || ‘--’ || *address* = ….
- Expressions can also be used in SELECT clause
  - E.g. SELECT *bname* || ‘--’ || *address* AS NameAdd
    FROM branch

Partial Matching

customer(cname, street, city)

- Find every customer whose address starts with “Computing Science”.

```
SELECT  *
FROM    customer
WHERE   address LIKE ‘Computing Science%’
```

- Expression: col-name [NOT] LIKE pattern
  - Pattern may include wildcard characters ‘%’ matching any string and ‘_’ (underscore) matching any single character.

Naming the Results

deposit(accno, cname, bname, balance)

- For every deposit holder who has over $1000, find the customer name and the balance over $1000.

```
SELECT   cname, (balance - 1000) as bal
FROM     deposit
WHERE    balance > 1000
```

Ordering the Results

branch (bname, address, city, assets)

- Find the names and assets of all branches with assets greater than $2,500,000 and order the result in ascending order of asset values.

```
SELECT   bname, assets
FROM     branch
WHERE    assets > 2500000
ORDER BY  assets
```

- Default is ascending order; a descending order can be specified by the DESC keyword.

Queries Involving Set Operators

- set union : Q1 UNION Q2
  - the set of tuples in Q1 or Q2
- set difference : Q1 EXCEPT Q2
  - the set of tuples in Q1 but not in Q2
- set intersection : Q1 INTERSECT Q2
  - the set of tuples in both Q1 and Q2

- Q1 and Q2 must be union-compatibles
  - same number/types of attributes.

Queries With Set Operators

- List deposit-holders who have no loans.

```
(SELECT   cname
 FROM     deposit)

 EXCEPT

 (SELECT  cname
  FROM    loan)
```

- List cities where there is either a customer or a branch.

```
(SELECT city
FROM   customer)

UNION

(SELECT city
FROM   branch)
```

Queries With Set Operators

- Find all cities that have both customers and branches in.

```
(SELECT city
FROM    customer)

INTERSECT

(SELECT city
FROM    branch)
```

- Find every city that has a branch but no customer.

Queries Over Multiple Relations

branch (bname, address, city, assets)
customer(cname, street, city)
deposit(accno, cname, bname, balance)

```
SELECT branch.bname, assets
FROM  branch,customer,deposit
WHERE customer.city = ‘Jasper’
AND   customer.cname = deposit.cname
AND   deposit.bname = branch.bname
```

- What does the query do?

Find the name and asset of every branch that has a deposit account holders who lives in Jasper.

Queries With Nested Structures

- Queries within the WHERE clause of an outer query

```
    SELECT
    FROM
    WHERE   OPERATOR(SELECT … FROM … WHERE)
```

- There can be multiple levels of nesting
- Operators: IN, (NOT) EXISTS, < ALL, …
- Avoid nesting as much as possible.

Nested Structures Using "IN"

```
SELECT
FROM
WHERE expr|(expr list) IN
                       (set of values)
```

- E.g.

  ```
  … WHERE province IN ('AB','BC')
  … WHERE province NOT IN ('AB','BC')
  ```

Example

deposit(accno, cname, bname, balance)

```
SELECT cname
FROM   deposit
WHERE  bname   IN
              (SELECT    bname
               FROM  deposit
               WHERE cname = ‘John Doe’)
```

- What does the query do?

  Find every customer who has a deposit in some branch at which John Doe has a deposit.

The Same Example Without Nesting

deposit(accno, cname, bname, balance)

```
SELECT   d1.cname
 FROM    deposit d1, deposit d2
 WHERE   d2.cname = 'John Doe'
 AND     d1.bname = d2.bname
```

NOTE: avoid nesting as much as possible.

Nested Structures Using “< ALL”,…

```
SELECT
FROM
WHERE expr < ALL (set of values)
```

- Other forms: “<= ALL”, “= ALL”, “>= ALL”, “> ALL”
- Op ALL (set of values) evaluates to **true** iff the comparison evaluates to true for every value in the set.
- Op SOME (set of values) evaluates to **true** iff the the comparison evaluates to true for at least one value in the set.

Nested Structures Using “>ALL”, …

branch (bname, address, city, assets)

- Find branches that have assets greater than the assets of all branches in Calgary.

```
SELECT   bname
FROM     branch
WHERE    assets   > ALL
               (SELECT  assets
                FROM    branch
                WHERE   city = ‘Calgary’)
```

Nested Structures Using "EXISTS"

```
SELECT
FROM
WHERE  (NOT)  EXISTS(SELECT  …  )
```

- EXISTS (SELECT …) evaluates to true iff the result of the subquery contains at least one row.
- The expression is evaluated for every iteration of the outer query.

“EXISTS” Construct Example

branch (bname, address, city, assets)
customer(cname, street, city)

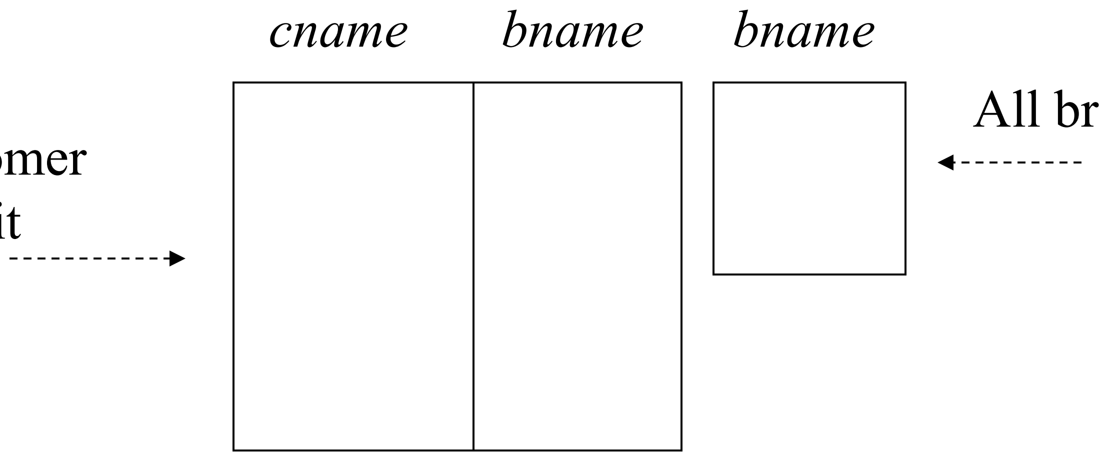
- Find the names of customers who live in a city with no bank branches.

```
SELECT   cname
FROM     customer
WHERE NOT EXISTS (SELECT *
                  FROM   branch
                  WHERE  customer.city=branch.city)
```

- Find the names of customers who live in a city which has a bank branch.
  - Change NOT EXISTS to EXISTS.
    (can also write it using join)

Division

- *Query type*: Find the subset of items in one set that are related to *all* items in another set
- *Example*: Find customers who have deposit accounts in *all* branches.



Contains row <*c*,*b*> if customer *c* has a deposit account in branch *b*

All branch names

$$\pi_{\text{cname, bname}}(\text{deposit}) \ / \ \pi_{\text{bname}}(\text{branch})$$

Division

- *Strategy for implementing division in SQL*:
    - Find set, A, of all branches in which a particular customer, *c*, has a deposit account.
    - Find set, B, of all branches.
    - Output *c* if A ⊇ B, or, equivalently, if B–A is empty

Division – SQL Solution

branch (bname, address, city, assets)

deposit(accno, cname, bname, balance)

```
SELECT c.cname
FROM customer c
WHERE NOT EXISTS
   (SELECT b.bname              -- set B of all branches
    FROM branch b
        EXCEPT
    SELECT d.bname               -- set A of branches in which
                                 -- customer c has a deposit account
    FROM deposit d
    WHERE d.cname=c.cname        -- global variable
)
```

Division – 2nd SQL Solution

- Find customers who have deposit accounts in all branches.
  - Same as “find all customers such that there is no branch where they do not have deposits in.”

```
SELECT c.cname
FROM   customer c
WHERE NOT EXISTS
      (SELECT    bname
       FROM      branch b
       WHERE NOT EXISTS
         (SELECT *
          FROM    deposit
          WHERE   b.bname=deposit.bname
          AND     c.cname = deposit.cname))
```

branches where c has no deposits in

Deposits of c in branch b

Division – Another Example

- Find professors who have taught courses in *all* departments.

- *Strategy for implementing division in SQL*:
  - Find set, A, of all departments in which a particular professor, *p*, has taught a course
  - Find set, B, of all departments
  - Output *p* if A $\supseteq$ B, or, equivalently, if B–A is empty

Division – SQL Solution

Professor (*Id, Name, DeptId*)
Department (*DeptId, Name*)
Course (*DeptId, CrsCode, CrsName, Descr*)
Teaching (*ProfId, CrsCode, Semester*)

```
SELECT P.Id
FROM Professor P
WHERE NOT EXISTS
    (SELECT D.DeptId              -- set B of all dept Ids
     FROM Department D
         EXCEPT
     SELECT C.DeptId              -- set A of dept Ids of depts in
                                  -- which P has taught a course
     FROM Teaching T, Course C
     WHERE T.ProfId=P.Id          -- global variable
          AND T.CrsCode=C.CrsCode)
```

SQL and SQLite

- SQLite: pretty modern but light language
  - Released in 2000 (Oracle was 1977, DB2 was 1983)
  - It is light (intended for small devices)
  - Tightly integrated with applications (software library rather than a stand-alone system/process to communicate)
  - Locks the whole file/database during writes (not a good choice for write-intensive/concurrent transactions)
  - Supports both in-memory and on-disk databases
- Differences with SQL-92
  - Does not support op ALL in nested queries
  - Does not support RIGHT OUTER JOIN and FULL OUTER JOIN
  - Views are read-only
  - ALTER TABLE command is very limited
  - Foreign keys constraints are not enforced by default
  - More left to be explored!

Basic Data Types in SQLite

- INTEGER 1, 2, 3, 4, 6, 8 bytes integer (depending on the magnitude)
- REAL 8 bytes floating point number
- TEXT stored using database encoding (e.g. UTF-8)
- BLOB stored as is
- NULL null value

- NUMERIC can store all other types; stored value is converted to INTEGER or REAL if the conversion is lossless and reversible

Other Types in SQLite

▪ INT,SMALLINT	INTEGER
▪ CHAR(n), VARCHAR(n)	TEXT
▪ DOUBLE, FLOAT	REAL
▪ DECIMAL	NUMERIC
▪ DATE, DATETIME	NUMERIC