Introduction to Database Systems

CSE 444

Lecture #4
Jan 17 2001

Announcements – I

% Special Lecture
- At Sieg 134 on Friday January 19th from 330-450PM
- Topic: Building SQL Applications
- Important For
  - Programming Assignment
  - Course Project

Announcement II

% Homework Due Today
% Programming Assignment available
- Due in a week
- Goal
  - More experience in SQL
  - Building applications using SQL
  - Incentive to build front-end
% Mid Term
- In Class
- All material except Transactions

SQL (Contd.)

Reading:
- Sec 5 (except 5.10)
- Sec 7 (except 7.2 – to be covered later)

Views

% A view is just a relation, but we store a definition (query), rather than a set of tuples.
- Can rename columns
  - CREATE VIEW YoungActiveStudents (Yname, Ygrade) AS SELECT S.name, E.grade FROM Students S, Enrolled E WHERE S.sid = E.sid and S.age<21
- Views can be dropped using the DROP VIEW command.

Uses for Views

% Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s) (security).
% Views also useful for maintaining logical data independence when the conceptual schema changes.
% May be used to precompute results
Views vs. Relations

Logical distinctions:
- Updates not always possible to a view
- View updates must be unambiguously mappable to base relation updates in order to be allowed

Physical distinctions:
- Relations must be physically stored somewhere
- Views are logical entities

Is it possible to rewrite using Views?

Find companies who manufacture products bought by Joe Blow.

```
SELECT Product.Company
FROM Product
WHERE Product.company = "Bazzar"
AND Product.name IN
    (SELECT product
    FROM Purchase
    WHERE buyer = "Joe Blow");
```

Is it possible to rewrite using Views?

Product (pname, price, category, maker)
Find products that are more expensive than all those produced by “Gizmo-Works”

```
SELECT name
FROM Product
WHERE price > ALL (SELECT price
    FROM Purchase
    WHERE maker="Gizmo-Works")
```

Is it possible to rewrite using Views?

Product (pname, price, category, maker, year)
Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT pname, maker
FROM Product AS x
WHERE price > ALL (SELECT price
    FROM Product AS y
    WHERE x.maker = y.maker AND
    y.year < 1972);
```

Null Values

- If x=Null then 4*(3-x)/7 is still NULL
- If x=Null then x="Joe" is UNKNOWN
- Three boolean values:
  - FALSE = 0
  - UNKNOWN = 0.5
  - TRUE = 1

Null Values

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

SELECT *
FROM Person
WHERE (age < 25) AND
    (height > 6 OR weight > 190)

Rule in SQL: include only tuples that yield TRUE
**Null Values**

Unexpected behavior:

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
```

Some Persons are not included!

**Null Values**

Can test for NULL explicitly:

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
OR age IS NULL
```

Now it includes all Persons

---

**Notation for Join in SQL92**

Explicit joins in SQL:
- `Product(name, category)`
- `Purchase(prodName, store)`

```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON
  Product.name = Purchase.prodName
```

Same as:

```sql
SELECT Product.name, Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
```

But products that never sold will be lost!

**Outerjoin**

Left outer joins in SQL:
- `Product(name, category)`
- `Purchase(prodName, store)`

```sql
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase ON
  Product.name = Purchase.prodName
```

**Example of Outerjoin**

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ProdName</td>
</tr>
<tr>
<td>Category</td>
<td>Store</td>
</tr>
<tr>
<td>Game</td>
<td>Gismo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gismo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
<tr>
<td>OneClick</td>
<td>-</td>
</tr>
</tbody>
</table>

---

**Modifying the Database**

Insert a new purchase to the database:

```sql
INSERT INTO Purchase(buyer, seller, product_name, store)
VALUES ("Joe", "Fred", "gizmo", "GizmoStore")
```
Insertion Exploiting Query

\[
\text{INSERT INTO PRODUCT (product\_name, store) SELECT DISTINCT product\_name, store FROM Purchase WHERE product NOT IN (SELECT name FROM Product)}
\]

Schema: Purchase(buyer, seller, product\_name, store)
Product (product\_name, store)

Note the order of querying and inserting.

Deletion

\[
\text{DELETE FROM PURCHASE WHERE seller = "Joe" AND product = "Brooklyn Bridge"}
\]

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Updates

\[
\text{UPDATE PRODUCT SET price = price/2 WHERE Product.name IN (SELECT product FROM Sales WHERE Date = today)}
\]

Updating Views

\*Need to be able to update base relations such that result of view reflects update
\*Formal Definition of "updateable" views is complex
\*Example of "updateable" views
  - Simple selection OK
  - Use of DISTINCT not allowed
  - Self-referential selection condition not allowed

Updating Complex Views

How can I insert a tuple into a table that doesn’t exist?

CREATE VIEW bon-purchase AS
SE\text{SELECT store, seller, product FROM Purchase WHERE store = "The Bon Marche"}

If we make the following insertion:

\[
\text{INSERT INTO bon-purchase VALUES ("the Bon Marche", Joe, "Denby Mug")}
\]

We can simply add a tuple
   - ("the Bon Marche", Joe, NULL, "Denby Mug")
   - ("the Bon Marche", Joe, NULL, "Denby Mug")

to relation Purchase.

Example of Non-Updatable Views

CREATE VIEW Seattle-view AS
\text{SELECT seller, product, store FROM Person, Purchase WHERE Person.city = "Seattle" AND Person.name = Purchase.buyer}

How can we add the following tuple to the view above?
Think about null semantics..

(Joe, "Shoe Model 12345", "Nine West")
Using SQL in Applications

Business logic involves
- Multiple SQL queries
- Application code in a development language (Java, C++, Visual Basic)
- Code may need to be executed
  - At Client/Middle-Tier
  - At server

Using SQL in Applications

(2)

Data Type issues (Mapping of Types)
- Reconcile Explicit iteration in Programming Language with set-oriented processing in SQL (Cursors)
- SQL generated on-the-fly (Dynamic SQL)
- Connectivity of client code to database server

Mapping Types

- char => character (length, char set)
- varchar => character varying (length, char set)
- short => smallint
- Long => integer
- Float => real
- Double => double precision

Getting Data Out

- Application languages deals with a row at a time
  - Not set of rows
- How to consume result of a SQL query?
- SQL supports cursors
  - Like a pointer that traverses a collection of rows one at a time

Cursors

1. Declare the cursor
2. Open the cursor
3. Fetch rows one by one
4. Update/Delete "current" tuples
5. Close the cursor
Declare - Example

Declare cursor1 cursor for
Select current_sales_price, our_cost
From movie_titles
Where current_sales_price > :minprice
Order By current_sales_price

Open/Fetch/Close

Open cursor_name
Fetch [Next| Prior| First | Last | Absolute
<k> | Relative <k> ] cursor_name into
:struct1
Close cursor_name

Update/Delete

Delete from table_name
where current of cursor_name
Update table_name Set set_list
where current of cursor_name
Update movie_titles Set our_cost = our_cost/2
where current of cursor1

Revisiting Declare

 DECLARE cursor-name
 <=[INSENSTIVE] [SCROLL] CURSOR FOR
 Query_expression
 ORDER BY sort_expression
 updatability

 Declare (Contd)

 $%Updatability$
 ☐Read Only – no update/delete on cursor allowed
 ☐Update restricted to specific fields
 ☐For update of column_name [, column_name]
 ☐Declare cursor1 cursor for
 Select current_sales_price, our_cost
 From movie_titles
 For update of current_sales_price

 Declare (Contd)

 $%Insensitive$
 ☐Cursor fetches all movies with cost > x
 ☐Fetch n records
 ☐Reduce cost of all movies by 20%
 ☐What records do you see next?
 ☐Same as above
 $%Indeterminant$
Declare (Contd)

*Scrollable Cursors
- Additional syntax in Fetch enabled
- Otherwise, only “next” tuple is available
- But scroll forces cursor to be read-only!

Connectivity - ODBC

- Client needs to establish a connection to server
  - Generates connection handle - unique identification
- Execute statements
  - Statement Handle with each – unique identification

ODBC Details

- SQLDriverConnect -- opens a connection
- SQLExecDirect -- executes a sql statement
- SQLBindCol -- binds a program variable to a column in the result of a sql statement
- SQLFetch -- fetches the next row in the current result set
- SQLMoreResults -- returns true if more result sets are yet to be consumed (e.g., useful for a batch of queries)
- SQLError -- returns information about the last error (for the specified connection)

Friday’s (Jan 19) Special Lecture

- More on Connectivity
- Building a front-end using ASP
- Relevant for
  - Programming Assignment
  - Project
- Note time and place
  - Sieg 134
  - 3.30-4.50pm
- Please be there!