

Advanced SQL



Chapter 8 finally!

Views (okay so this is Ch 7)

- We've been “saving” SQL queries by saving text files
- Isn't there a better way?
- Views! → basically saved SELECT queries
- Syntax →
CREATE VIEW *viewname*
AS *some_select_query*

Okay so why use them?

- They can be thought of as virtual tables
 - A view can be used anywhere a table would be used
 - A mechanism to provide restricted access
 - Can be a very simple form of reporting

It's about time...

Now to Chapter 8

More set operators

- SQL operates on **sets** of data, that is the combination of rows and columns
- Some data sets are **union-compatible** → the column names are the same and data types are “similar”
- Sets must be union-compatible to use the following commands: **union**, **minus**, and **intersect**

Join the union

- A union combines the results of two select queries but excludes any duplicates
- Syntax → *select_query* **UNION** *select_query*
- SELECT a, b FROM foo UNION SELECT a, b FROM bar
- Many tables can be strung together with successive UNION keywords, not just two
- Fig. 8.1

Join all the unions

- This is the same idea as UNION but duplicates are included
- Syntax is the same except **UNION ALL** is used

Stop at the INTERSECTion

- Again, similar syntax to UNION but use **INTERSECT** instead
- In this case you get only the rows that appear in both tables
- Fig. 8.3

MINUS

- Again, similar syntax but use **MINUS**
- Return the rows that appear in the first query but not in the second
- Thus, this query is *order dependent*
- Fig. 8.4

The problem

- Remember the “everyone implements SQL a little differently” problem?
- These commands may not exist in your particular DBMS

A question

- Give me all of the device data for the manufacturer Intel
- What about a join?
- `SELECT * FROM Device, Manufacturer WHERE Device.manid = Man.id`
- Will this work?

Subqueries

- Sometimes you will not need to do a join (as in have results from two tables), but you do need information from another table to write your query
- For this subqueries are used and we've seen them briefly before

Subqueries

- Back to the question...
SELECT * FROM Devices
WHERE Devices.man_id = (SELECT id FROM
Manufacturer WHERE name = 'intel')
- Using the part/vendor DB, what about giving
me a vendor that has no products?

Subqueries

- `SELECT * FROM Vendor
WHERE v_code NOT IN (SELECT v_code
FROM Product)`

Subqueries

- The earlier examples we've seen are all subqueries contained in the WHERE clause, the most common type
- When the inner query returns more than one result there's a problem though
- In this case, make sure you are using an IN inside the WHERE clause, as in the last example
- Subqueries can be used elsewhere, such as the HAVING clause (the WHERE of groups)

ANYthing better than IN at ALL?

- IN is only good for equality comparisons, what if you need inequalities?
- **ANY** and **ALL** are used for these
- `SELECT * FROM Products
WHERE p_price > ALL(SELECT p_price FROM
Products WHERE p_qoh > 0)`

Functions

- Functions in SQL work much like functions in programs
- We have seen some that deal with dates already:
 - YEAR, MONTH, DAY → returns the year/month/day part of a date
 - GETDATE() → return the current date and time

New Date Functions

- DATEADD(part, number, date) → add *number* to *date's part*, where part is day, month, or year
 - SELECT DATEADD(year, 1, GETDATE());
- DATEDIFF(part, start, end) → get *end-start* where you subtract the *part* portion
 - SELECT DATEDIFF(day, '1999-12-31', GETDDATE());
- Oracle has a couple of helpful ones but MS doesn't, like TO_DATE

Numeric Functions

- ABS → absolute value
- ROUND(val, precision) → round the value to the specified number of digits
- CEILING and FLOOR → normal ceiling and floor operations
- SELECT ABS(-4), ROUND(4.89, 1),
ROUND(4.89, 0), CEILING(4.89),
FLOOR(4.89);

String Functions

- + → concatenation
- UPPER/LOWER → change string to all upper or lower case letters
- SUBSTRING(*str*, *start*, *len*) → returns part of a string starting at position *start* (0-based indexing) and return *len* number of characters
- LENGTH → length of a string

Conversions

- CAST and CONVERT (mostly the same, just different syntax)
- Convert number to a string →
`SELECT CAST(1234 AS nvarchar(4)),`
`CONVERT(varchar(4), 1234)`
- Date to string →
`SELECT CAST(GETDATE() AS varchar(11)),`
`CONVERT(varchar(11), GETDATE())`
- String to number →
`SELECT CAST('1,000.12' AS decimal(8,2)),`
`CONVERT(decimal(8,2), '1000.12')`

A CASE for SWITCHing

- SQL can also do a switch-like statement called a CASE
- **SELECT CASE WHEN state='OH' THEN 0.05
WHEN state='KY' THEN 0.1
WHEN state='IN' THEN 0.0**

Pull the TRIGGER

- Triggers allow the DBMS to run code automatically when certain conditions are met when rows are inserted, updated, or deleted
- Triggers can enforce things that may not be enforceable through other means (like CHECKs)
- They are associated with tables and a table can have more than one

Uses

- Oracle's recommendations on when to use triggers
 - Auditing
 - Generation of derived column values
 - Enforcement of business and/or security constraints
 - Creation of table copies for backup purposes

Syntax

- Syntax →
CREATE TRIGGER *trigger_name*
ON *table or view*
FOR UPDATE/INSERT/DELETE
AS
IF UPDATE(*column* **)** [AND/OR
UPDATE(*column***)...**]
some query here

Example

- When inventory gets low automatically add more
- **CREATE TRIGGER** tgr_inv
ON Product
FOR UPDATE, INSERT
AS
IF UPDATE(p_qoh)
UPDATE Product
SET p_backorder = 1
WHERE p_qoh <= 0

Stored Procedures

- Sort of like views, these are ways to store a set of code in the database to be run at different times
- Unlike views, they aren't simply “virtual tables” created by select statements, they can run any SQL code

Syntax

- **CREATE PROCEDURE** *name parameters*
AS
some query
- **CREATE PROCEDURE** sp_getstateprods
@state varchar(2)
AS
SELECT name **FROM** Product
WHERE v_code **IN** (**SELECT** v_id **FROM**
Vendor **WHERE** state = @state)

Using SPs

- To use an SP we call it with EXECUTE giving the name and then data for any parameters
- **EXECUTE** sp_getstateprods 'CA'

Cursors

- These are used for two things:
 - Returning more than one value (triggers and SPs can return values, but only one)
 - Doing something in a more programming-like fashion

How to use them

Don't use them

Homework!

- Review: 3, 4, 5, 6, 14, 15