SQL: Structured Query Language

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Some slides are adopted from D. Maier, S. Davidson and A. Parameswaran

Learning Objectives

After this lecture, you should be able to:

- Write **single-relation SQL** queries
- Write SQL queries with **complex WHERE conditions**
- Write **INNER, NATURAL and OUTER JOIN SQL** queries
- Evaluate the effect of Null on SQL queries

SQL: Structured Query Language

The standard language for relational data

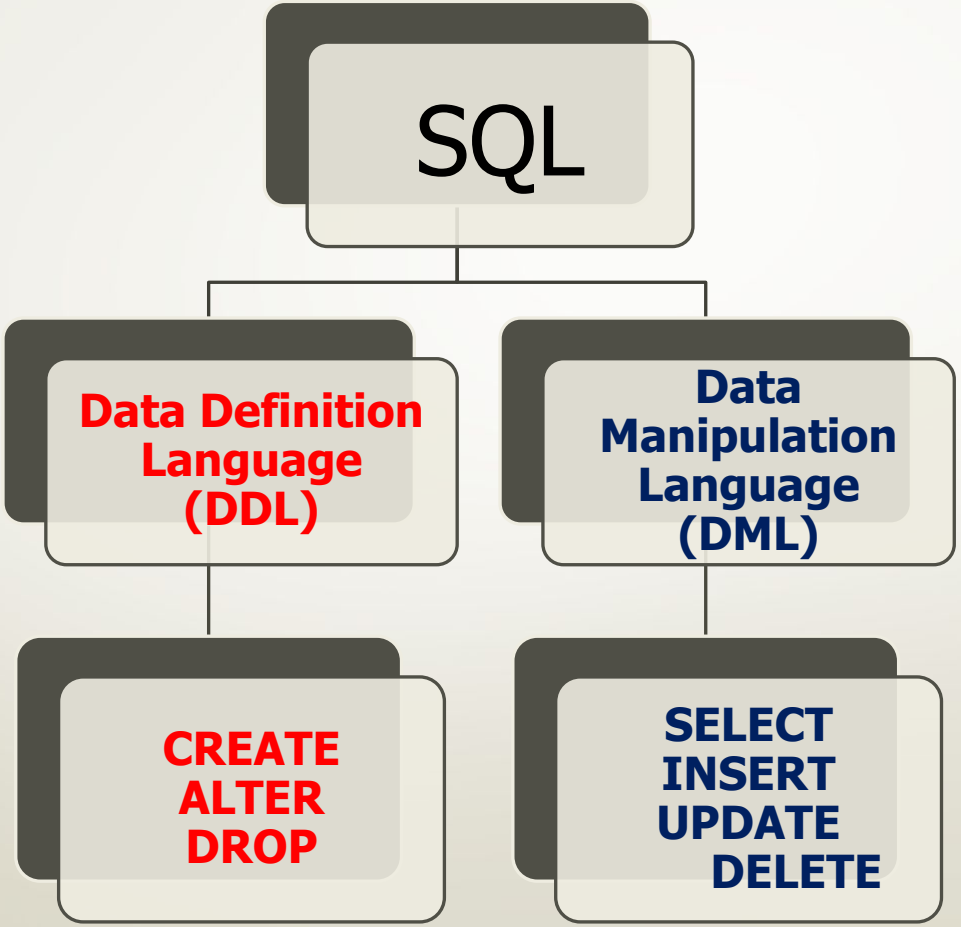
- Invented by folks at IBM, esp. Don Chamberlin

Separated into

- DDL (data definition language)
- DML (data manipulation language)

DML based on relational calculus, which we discuss later

SQL – the language we use to talk to the Database Management System



SQL (cont.)

SQL is a standard…
and there have been a series of SQL standards: 1986, 1989, 1992 (SQL2), 1999 (SQL3), …, SQL:2011

But DBMS products differ in how much of the standard they support … and how many extra features they have.

Single-Relation Query Form

- The principal form of a single-relation query is:

SELECT desired attributes
FROM one table (relation)
WHERE condition about tuples of the table

Example Query

Account

Number	Name	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

```
SELECT    Number, Owner
FROM      Account
WHERE     Type = "savings";
```

Number	Owner
103	J. Smith

How a Single-Relation SQL query is evaluated

Third, the SELECT clause tells us which columns to keep in the query answer.

```
SELECT   Number, Owner
FROM     Account
WHERE    Type = "savings"
```

First, the FROM clause tells us the input tables.

Second, the WHERE clause is evaluated for all rows from the input table.

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example based on Account(number, owner, balance, type):

```
SELECT Number AS Acc_Num, Owner
 FROM Accounts
 WHERE Type = "savings";
```

Acc_Num	Owner
103	J. Smith

Another Database Schema

- We will be using the following database schema as a running example.
  - Underline indicates key attributes.

Drinks(<u>name</u>, manf)

Cafe(<u>name</u>, addr, license)

Customers(<u>name</u>, addr, phone)

Likes(<u>customer</u>, <u>drink</u>)

Sells(<u>cafe</u>, <u>drink</u>, price)

Frequents(<u>customer</u>, <u>cafe</u>)

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.

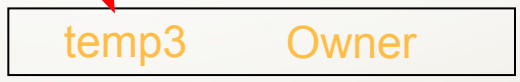
- Example: from Sells (cafe, drink, price):

```
SELECT cafe, drink, price * 120 AS priceInYen
FROM Sells;
```

Creating temporary tables using INTO

We can create a name for the query answer:

```
SELECT    Owner INTO temp3
FROM      Account
WHERE     Type = "checking";
```



temp3	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

temp3 can be used as a table in subsequent queries!

REMEMBER TO DELETE YOUR TEMPORARY TABLES!

Complex Conditions in WHERE Clause

- From Sells (cafe, drink, price), find the price Caffe bene charges for Mocha:

```
SELECT price
FROM Sells
WHERE cafe = 'Caffe bene' AND
      drink = 'Mocha';
```

WHERE Clause Syntax

What you can use in WHERE:

- attribute names of the relation(s) used in the FROM.
- comparison operators: =, <>, <, >, <=, >=
- apply arithmetic operations: stockprice*2
- operations, comparisons on strings
- pattern matching: s LIKE p
- special stuff for comparing dates and times.

See the textbook for more…

Then, create bigger expressions using Boolean connectives

Syntax: Patterns and "LIKE"

- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.

- General form: <Attribute> LIKE <pattern>
  or <Attribute> NOT LIKE <pattern>

- Pattern is a quoted string with
  - % "any string"
  - _ "any character."

Example

- From Customers(name, addr, phone) find the customers that have phone numbers with middle three numbers 555:

```
SELECT name
FROM Customers
WHERE phone LIKE '%555-_ _ _ _';
```

Ordering the results

```
SELECT *
FROM  Account
WHERE Name LIKE
'J%'
ORDER BY Balance
```

- Ordering is ascending, unless you specify the DESC keyword.
- Ties are broken by the second attribute on the ORDER BY list, etc.

Outline

- ✔ Single-Relation SQL Queries
- Multi-Relation SQL Queries
- Join SQL Queries

Multi-relation Queries

- Interesting queries often combine data from more than one relation.

- We can address several relations in one query by listing them all in the FROM clause.

- Distinguish attributes of the same name by "<relation>.<attribute>"

Example of Multi-Relation Query

```
SELECT    A.Owner, A.Balance
FROM      Account A, Deposit D
WHERE     D.AcctNo = A.Number and A.Balance > 1000;
```

How does this work?

Which rows, from which tables,

are evaluated in the WHERE clause?

Example of Multi-Relation Query

```
SELECT    A.Owner, A.Balance
FROM      Account A, Deposit D
WHERE     D.AcctNo = A.Number and A.Balance > 1000;
```

"A" is a correlation name for Account

and

"D" is a correlation name for Deposit.

Correlation names are like local variables – they hold one tuple or row from the corresponding table.

You choose correlation names when you write the query.

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

AcctNo	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

```
SELECT  A.Owner, A.Balance
FROM    Account A, Deposit D
WHERE   D.AcctNo = A.Number and A.Balance > 1000;
```

We must check every combination of one row from Account with one row from Deposit.

Intermediate result

(after processing the FROM & WHERE clauses)

Number	Owner	Balance	Type	AcctNo	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10,000.00

Process the SELECT

```
SELECT A.Owner, A.Balance
FROM   Account A, Deposit D
WHERE  D.AcctNo = A.Number and A.Balance > 1000;
```

Final query answer: (notice that W. Wei appears twice)

Owner	Balance
W. Wei	2000.00
W. Wei	2000.00
H. Martin	10,000.00

Intermediate result
(after processing the FROM & WHERE clauses)

Number	Owner	Balance	Type	AcctNo	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10,000.00

```
SELECT DISTINCT A.Owner, A.Balance
FROM   Account A, Deposit D
WHERE  D.AcctNo = A.Number and A.Balance > 1000;
```

If we use the word DISTINCT, then duplicates are removed from the query answer. W. Wei only appears once.

Owner	Balance
W. Wei	2000.00
H. Martin	10,000.00

Operational Semantics of Multi-relation queries

Almost the same as for single-relation queries:

1. Start with the product of all the relations in the FROM clause.
2. Apply the selection condition from the WHERE clause.
3. Project onto the list of attributes and expressions in the SELECT clause.

Queries

Account	Number	Owner	Balance	Type	
	101	J. Smith	1000.00	checking	
	102	W. Wei	2000.00	checking	
	103	J. Smith	5000.00	savings	
	104	M. Jones	1000.00	checking	
	105	H. Martin	10,000.00	checking	

Notice that a query is Expressed against the schema.

But the query runs or executes against the instance (the data)

And may give different answers on different instances

```
SELECT  Owner
FROM    Account
WHERE Type = "checking";
```

	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

Comments on Queries

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Notice that the answer to a query is always a **table**!

It doesn't always have a name (for the table).

The attribute names are deduced from the input tables (or supplied by the query author). It might or Might not have any rows.

	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

Comments on Queries

Because the answer to a relational query is always a table

- ✔ we can use the answer from one query as input to another query.
- ✔ This means that we can create arbitrarily complex queries!
- ✔ A query language is **closed** if it has this property.

Outline

- ✔ Single-Relation SQL Queries
- ✔ Multi-Relation SQL Queries
  - Join SQL Queries

Example Database

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Cross Product

Department × Employee

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

```
SELECT *
FROM Department, Employee
```

Department.DepartmentName	Department.DepartmentID	Employee.LastName	Employee.DepartmentID
Sales	31	Rafferty	31
Sales	31	Jones	33
Sales	31	Steinberg	33
Sales	31	Smith	34
Sales	31	Robinson	34
Sales	31	John	NULL
Engineering	33	Rafferty	31
Engineering	33	Jones	33
Engineering	33	Steinberg	33
Engineering	33	Smith	34
Engineering	33	Robinson	34
Engineering	33	John	NULL
Clerical	34	Rafferty	31
Clerical	34	Jones	33
Clerical	34	Steinberg	33
Clerical	34	Smith	34
Clerical	34	Robinson	34
Clerical	34	John	NULL
Marketing	35	Rafferty	31
Marketing	35	Jones	33
Marketing	35	Steinberg	33
Marketing	35	Smith	34
Marketing	35	Robinson	34
Marketing	35	John	NULL

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Equijoin

Employee ⋈ **Department**

Employee.DeptID = **Department.DeptID**

```
SELECT *
FROM Employee emp JOIN Department dept
      ON emp.DepartmentID = dept.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Natural Join

Employee ⋈ Department

```
SELECT *
FROM Employee emp NATURAL JOIN Department dept
```

DepartmentID	Employee.LastName	Department.DepartmentName
34	Smith	Clerical
33	Jones	Engineering
34	Robinson	Clerical
33	Steinberg	Engineering
31	Rafferty	Sales

Nulls and Joins

- Sometimes need special variations of joins:
  - I want to see all employees and their departments
  - … But what if there's a department with no employees?
  - Or what if an employee has not been assigned to a department?
- Outer join:
  - Most common is *left outer join*

Outer Joins

- Left outer join:
  - Include the left tuple even if there's no match
- Right outer join:
  - Include the right tuple even if there's no match
- Full outer join:
  - Include both the left and right tuples even if there's no match

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Left Outer Join

Employee ⟕ Department

Employee.DepartmentID = Department.DepartmentID

```
SELECT *
FROM Employee emp LEFT OUTER JOIN Department dept
        ON emp.DepartmentID = dept.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Right Outer Join

Employee ⟖ Department

Employee.DepartmentID = Department.DepartmentID

```
SELECT *
FROM Employee emp RIGHT OUTER JOIN Department dept
    ON emp.DepartmentID = dept.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

Full Outer Join

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee ⟗ Department

Employee.DepartmentID = Department.DepartmentID

```
SELECT *
FROM Employee FULL OUTER JOIN Department dept
    ON emp.DepartmentID = dept.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

Outline

- ✔ Single-Relation SQL Queries (Cont.)
- ✔ Multi-Relation SQL Queries
- ✔ Join SQL Queries