CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#6: Fun with SQL (part1)

CMU SCS

General Overview - Rel. Model

- Formal query languages
  - rel algebra and calculi
- Commercial query languages
  - SQL
  - Datalog
  - LINQ
  - Xquery
  - Pig (Hadoop)

"Intergalactic Standard"

CMU SCS 15-415/615 2

CMU SCS

Relational Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- User only needs to specify the answer that they want, not how to compute it.
- The DBMS is responsible for efficient evaluation of the query.
  - Query optimizer: re-orders operations and generates query plan

CMU SCS 15-415/615 3

CMU SCS

Relational Languages

- Standardized **DML**/**DDL**
  - **DML** → Data Manipulation Language
  - **DDL** → Data Definition Language
- Also includes:
  - View definition
  - Integrity & Referential Constraints
  - Transactions

CMU SCS

History

- Originally "SEQUEL" from IBM's System R prototype.
  - **S**tructured **E**nglish **Que**ry **L**anguage
  - Adopted by Oracle in the 1970s.
- ANSI Standard in 1986, ISO in 1987
  - **S**tructured **Q**uery **L**anguage

CMU SCS

History

- Current standard is SQL:2011
  - **SQL:2008** → TRUNCATE, Fancy ORDER
  - **SQL:2003** → XML, windows, sequences, auto-generated IDs.
  - **SQL:1999** → Regex, triggers, OO
- Most DBMSs at least support **SQL-92**
- System Comparison:
  - http://troels.arvin.dk/db/rdbms/

CMU SCS

Overview

- **DML**
  - select, from, where, renaming
  - set operations
  - ordering
  - aggregate functions
  - nested subqueries
- Other parts: **DDL**, embedded SQL, auth etc

CMU SCS

Intro to SQL

- SELECT
- FROM
- WHERE
- Formal Semantics

CMU SCS

Example Database

CUSTOMER

cname	acctno
Georg Hegel	A-123
Friedrich Engels	A-456
Max Stirner	A-789

ACCOUNT

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Perry	1500
A-456	Downtown	1000

CMU SCS

First SQL Example

```
SELECT bname
  FROM account
 WHERE amt > 1000
```

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700
Redwood	L-450	3000

Similar to…

$\pi_{bname}(\sigma_{amt>1000}(account))$

bname
Downtown
Redwood
Perry

But not quite….

bname
Downtown
Redwood
Perry
Redwood

Duplicates

CMU SCS

First SQL Example

```
SELECT DISTINCT bname
  FROM loan
 WHERE amt > 1000
```

bname	lno	amt
Downtown	L-170	3000
Redwood	L-230	4000
Perry	L-260	1700
Redwood	L-450	3000

Now we get the same result as the relational algebra

bname
Downtown
Redwood
Perry

Why preserve duplicates?

- Eliminating them is costly
- Users often don't care.

CMU SCS

Multi-Relation Queries

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```

cname	acctno
Georg Hegel	A-123
Friedrich Engels	A-456
Max Stirner	A-789

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Perry	1500
A-456	Downtown	1000

Same as

$\pi_{cname,\ amt}(\sigma_{amt>1000}(customer \bowtie account))$

cname	amt
Georg Hegel	1800
Max Stirner	2000
Georg Hegel	1500

CMU SCS

Basic SQL Query Grammar

```
SELECT [DISTINCT|ALL] target-list
  FROM relation-list
[WHERE qualification]
```

- **Relation-List**: A list of relation names
- **Target-List**: A list of attributes from the tables referenced in relation-list
- **Qualification**: Comparison of attributes or constants using operators =, ≠, <, >, ≤, and ≥.

CMU SCS 15-415/615 13

CMU SCS

SELECT Clause

- Use * to get all attributes

```
SELECT * FROM account
```

```
SELECT account.* FROM account
```

- Use DISTINCT to eliminate dupes

```
SELECT DISTINCT bname FROM account
```

- Target list can include expressions

```
SELECT bname, amt*1.05 FROM account
```

CMU SCS 15-415/615 14

CMU SCS

FROM Clause

- Binds tuples to variable names

```
SELECT * FROM depositor, account
WHERE depositor.acctno = account.acctno
```

- Define what kind of join to use

```
SELECT depositor.*, account.amt
 FROM depositor LEFT OUTER JOIN account
WHERE depositor.acctno = account.acctno
```

CMU SCS 15-415/615 15

CMU SCS

WHERE Clause

- Complex expressions using **AND**, **OR**, and **NOT**

```
SELECT * FROM account
 WHERE amt > 1000
   AND (bname = “Downtown” OR
        NOT bname = “Perry”)
```

- Special operators **BETWEEN**, **IN**:

```
SELECT * FROM account
 WHERE (amt BETWEEN 100 AND 200)
   AND bname IN (“Leon”, “Perry”)
```


CMU SCS

Renaming

- The **AS** keyword can also be used to rename tables and columns in **SELECT** queries.
- Allows you to target a specific table instance when you reference the same table multiple times.

CMU SCS

Renaming – Table Variables

- Find customers with an account in the “Downtown” branch with more than $100.

```
SELECT customer.cname, account.amt
  FROM customer, account
 WHERE customer.acctno = account.acctno
   AND account.bname = “Downtown”
   AND account.amt > 1000
```

CMU SCS

Renaming – Table Variables

- Find customers with an account in the "Downtown" branch with more than $100.

```
SELECT C.cname, A.amt AS camt
  FROM customer AS C, account AS A
 WHERE C.acctno = A.acctno
   AND A.bname = "Downtown"
   AND A.amt > 1000
```


CMU SCS

Renaming – Self-Join

- Find all unique accounts that are open at more than one branch.

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Perry	1500
A-456	Downtown	1000

```
SELECT DISTINCT a1.acctno
  FROM account AS a1, account AS a2
 WHERE a1.acctno = a2.acctno
   AND a1.bname != a2.bname
```


CMU SCS

Renaming – Theta-Join

- Find all unique accounts that are open at more than one branch and have an amount greater than $1600.

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Perry	1500
A-456	Downtown	1000

```
SELECT DISTINCT a1.acctno
  FROM account AS a1, account AS a2
 WHERE a1.acctno = a2.acctno
   AND a1.bname != a2.bname
   AND a1.amt > 1600
```

CMU SCS

Formal Semantics of SQL

- To express SQL, must extend to a bag algebra:
  - A bag is like a set, but can have duplicates
  - Example: {4, 5, 4, 6}

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Redwood	1800
A-456	Downtown	1000

CMU SCS

Formal Semantics of SQL

- A SQL query is defined in terms of the following evaluation strategy:
  1. **Execute `FROM` clause**
     Compute cross-product of all tables
  2. **Execute `WHERE` clause**
     Check conditions, discard tuples
  3. **Execute `SELECT` clause**
     Delete unwanted columns.
- *Probably the worst way to compute!*

CMU SCS

Execution Example

- Find the students that got a "D" grade in any course.

```
SELECT S.name, E.cid
  FROM students AS S, enrolled AS E
 WHERE S.sid = E.sid AND E.grade="D"
```

Step 1 – Cross Product

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	1.8
53688	Bieber	jbieber@cs	21	3.9

sid	cid	grade
53831	Pilates101	C
53832	Reggae203	D
53650	Topology112	A
53666	Massage105	D

X

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Faloutsos	christos@cs	45	1.8	53831	Pilates101	C
53666	Faloutsos	christos@cs	45	1.8	53832	Reggae203	D
53666	Faloutsos	christos@cs	45	1.8	53650	Topology112	A
53666	Faloutsos	christos@cs	45	1.8	53666	Massage105	D
53688	Bieber	jbieber@cs	21	3.9	53831	Pilates101	C
53688	Bieber	jbieber@cs	21	3.9	53831	Reggae203	D
53688	Bieber	jbieber@cs	21	3.9	53650	Topology112	A
53688	Bieber	jbieber@cs	21	3.9	53666	Massage105	D

Step 2 – Discard Tuples

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
[redacted]							
53666	Faloutsos	christos@cs	45	1.8	53666	Massage105	D
[redacted]							

```
SELECT S.name, E.cid
  FROM Students AS S, Enrolled AS E
 WHERE S.sid = E.sid AND E.grade = "D"
```


Step 3 – Discard Columns

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
[redacted]							
53666	Faloutsos	christos@			6	Massage105	D
[redacted]							

S.name	E.cid
Faloutsos	Massage105

```
SELECT S.name, E.cid
  FROM Students AS S, Enrolled AS E
 WHERE S.sid = E.sid AND E.grade = "D"
```

CMU SCS

More SQL

- INSERT
- UPDATE
- DELETE
- TRUNCATE

CMU SCS

INSERT

- Provide target table, columns, and values for new tuples:

```
INSERT INTO account
  (acctno, bname, amt)
VALUES
  ("A-999", "Pittsburgh", 1000);
```

- Short-hand version:

```
INSERT INTO account VALUES
  ("A-999", "Pittsburgh", 1000);
```


CMU SCS

UPDATE

- **UPDATE** must list what columns to update and their new values (separated by commas).
- Can only update one table at a time.
- **WHERE** clause allows query to target multiple tuples at a time.

```
UPDATE account
   SET bname = "Compton",
       amt = amt + 100
 WHERE acctno = "A-999"
   AND bname = "Pittsburgh"
```

CMU SCS

DELETE

- Similar to single-table **SELECT** statements.
- The **WHERE** clause specifies which tuples will deleted from the target table.
- The delete may cascade to children tables.

```
DELETE FROM account WHERE amt < 0
```

CMU SCS 15-415/615 31

CMU SCS

TRUNCATE

- Remove all tuples from a table.
- This is usually faster than **DELETE**, unless it needs to check foreign key constraints.

```
TRUNCATE account
```

CMU SCS 15-415/615 32

CMU SCS

Even More SQL

- NULLs
- String Operations
- Output Redirection
- Set/Bag Operations
- Output Control
- Aggregates

CMU SCS 15-415/615 33

CMU SCS

NULLs

- The "dirty little secret" of SQL, since it can be a value for any attribute.

bname	city	assets
Oakland	Pittsburgh	$9,000,000
Compton	Los Angeles	NULL
Long Beach	Los Angeles	$400,000
Harlem	New York	$1,700,000

- What does this mean?
  - We don't know Compton assets?
  - Compton has no assets?

CMU SCS 15-415/615 34

CMU SCS

NULLs

- Find all branches that have null assets.

bname	city	assets
Oakland	Pittsburgh	$9,000,000
Compton	Los Angeles	NULL
Long Beach	Los Angeles	$400,000
Harlem	New York	$1,700,000

```
SELECT * FROM branch WHERE assets = NULL
```

bname	city	assets

CMU SCS 15-415/615 35

CMU SCS

NULLs

- Find all branches that have null assets.

bname	city	assets
Oakland	Pittsburgh	$9,000,000
Compton	Los Angeles	NULL
Long Beach	Los Angeles	$400,000
Harlem	New York	$1,700,000

```
SELECT * FROM branch WHERE assets IS NULL
```

bname	city	assets
Compton	Los Angeles	NULL

CMU SCS 15-415/615 36

CMU SCS

NULLs

- Arithmetic operations with **NULL** values is always **NULL**.

```
SELECT 1+NULL AS add_null,
       1-NULL AS sub_null,
       1*NULL AS mul_null,
       1/NULL AS div_null;
```

add_null	sub_null	mul_null	div_null
NULL	NULL	NULL	NULL

37

CMU SCS

NULLs

- Comparisons with **NULL** values varies.

```
SELECT true = NULL AS eq_bool,
       true != NULL AS neq_bool,
       true AND NULL AS and_bool,
       NULL = NULL AS eq_null,
       NULL IS NULL AS is_null;
```

eq_bool	neq_bool	and_false	eq_null	is_null
NULL	NULL	NULL	NULL	TRUE

CMU SCS 15-415/615 38

CMU SCS

String Operations

	String Case	String Quotes
SQL-92	Sensitive	Single Only
Postgres	Sensitive	Single Only
MySQL	Insensitive	Single/Double
SQLite	Sensitive	Single/Double
DB2	Sensitive	Single Only
Oracle	Sensitive	Single Only

```
WHERE UPPER(name) = 'EURKEL'
```
SQL-92

```
WHERE name = "EURKEL"
```
MySQL

39

CMU SCS

String Operations

- **LIKE** is used for string matching.
- String-matching operators
  - "%" Matches any substring (incl. empty).
  - "_" Match any one character

```
SELECT * FROM enrolled AS e
 WHERE e.cid LIKE 'Pilates%'
```

```
SELECT * FROM student AS s
 WHERE s.name LIKE '%loutso_'
```

CMU SCS 15-415/615 40

CMU SCS

String Operations

- **SQL-92** defines string functions.
  - Many DBMSs also have their own unique functions
- Can be used in either output and predicates:

```
SELECT SUBSTRING(name,0,5) AS abbrv_name
  FROM student WHERE sid = 53688
```

```
SELECT * FROM student AS s
 WHERE UPPER(e.name) LIKE 'FALOU%'
```

CMU SCS 15-415/615 41

CMU SCS

Output Redirection

- Store query results in another table:
  - Table must not already be defined.
  - Table will have the same # of columns with the same types as the input.

SQL-92
```
SELECT DISTINCT cid INTO CourseIds
  FROM Enrolled;
```

MySQL
```
CREATE TABLE CourseIds (
SELECT DISTINCT cid FROM Enrolled);
```

CMU SCS 15-415/615 42

CMU SCS

Output Redirection

- Insert tuples from query into another table:
  - Inner **SELECT** must generate the same columns as the target table.
  - DBMSs have different options/syntax on what to do with duplicates.

SQL-92

```
INSERT INTO CourseIds
(SELECT DISTINCT cid FROM Enrolled);
```

CMU SCS 15-415/615 43

CMU SCS

Set/Bag Operations

- Set Operations:
  - **UNION**
  - **INTERSECT**
  - **EXCEPT**
- Bag Operations:
  - **UNION ALL**
  - **INTERSECT ALL**
  - **EXCEPT ALL**

CMU SCS 15-415/615 44

CMU SCS

Set Operations

```
(SELECT cname FROM depositor)
              ???
(SELECT cname FROM borrower)
```

UNION
Returns names of customers with saving accts, loans, or both.

INTERSECT
Returns names of customers with saving accts AND loans.

EXCEPT
Returns names of customers with saving accts but NOT loans.

CMU SCS 15-415/615 45

Bag Operations

- There are *m* copies of *a* in table R and *n* copies of *a* in table S.
- How many copies of *a* in…
  - R **UNION ALL** S → $m + n$
  - R **INTERSECT ALL** S → $\min(m, n)$
  - R **EXCEPT ALL** S → $\max(0, m-n)$

Output Control

- **ORDER BY <column*> [ASC|DESC]**
  - Order the output tuples by the values in one or more of their columns.

```
SELECT sid, grade FROM enrolled
 WHERE cid = 'Pilates105'
 ORDER BY grade
```

sid	grade
53123	A
53334	A
53650	B
53666	D

```
SELECT sid FROM enrolled
 WHERE cid = 'Pilates105'
 ORDER BY grade DESC, sid ASC
```

sid
53666
53650
53123
53334

Output Control

- **LIMIT <count> [offset]**
  - Limit the # of tuples returned in output.
  - Can set an offset to return a "range"

```
SELECT sid, name FROM Student
 WHERE login LIKE '%@cs'
 LIMIT 10
```

First 10 rows

```
SELECT sid, name FROM Student
 WHERE login LIKE '%@cs
 LIMIT 20 OFFSET 10
```

Skip first 10 rows, Return the following 20

CMU SCS

Aggregates

- Functions that return a single value from a bag of tuples:
  - `AVG(col)` → Return the average col value.
  - `MIN(col)` → Return minimum col value.
  - `MAX(col)` → Return maximum col value.
  - `SUM(col)` → Return sum of values in col.
  - `COUNT(col)` → Return # of values for col.

CMU SCS

Aggregates

- Functions can only be used in the **SELECT** attribute output list.
- Get the number of students with a @cs login:

```
SELECT COUNT(login) AS cnt
  FROM student WHERE login LIKE '%@cs'
```

cnt
12

CMU SCS

Aggregates

- Can use multiple functions together at the same time.
- Get the number of students and their GPA that have a @cs login.

```
SELECT AVG(gpa), COUNT(sid)
  FROM student WHERE login LIKE '%@cs'
```

AVG(gpa)	COUNT(sid)
3.25	12

CMU SCS

Aggregates

- **COUNT**, **SUM**, **AVG** support **DISTINCT**
- Get the number of unique students that have an @cs login.

```
SELECT COUNT(DISTINCT login)
  FROM student WHERE login LIKE '%@cs'
```

COUNT(DISTINCT login)
10

CMU SCS 15-415/615 52

CMU SCS

Aggregates

- Output of other columns outside of an aggregate is undefined:

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
```

AVG(s.gpa)	e.cid
3.5	???

- *Unless…*

CMU SCS 15-415/615 53

CMU SCS

GROUP BY

- Project tuples into subsets and calc aggregates against each subset.

```
SELECT AVG(s.gpa), e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
```

e.sid	s.sid	s.gpa	e.cid
53435	53435	2.25	Pilates101
53439	53439	2.70	Pilates101
53423	53423	2.98	Topology112
56023	56023	2.75	Reggae203
59439	59439	3.90	Reggae203
53961	53961	3.50	Reggae203
58345	58345	1.89	Massage105

AVG(s.gpa)	e.cid
2.46	Pilates101
3.39	Reggae203
2.98	Topology112
1.89	Massage105

54

CMU SCS

GROUP BY

- Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

```
SELECT AVG(s.gpa), e.cid, s.name
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
```

X

CMU SCS

GROUP BY

- Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

```
SELECT AVG(s.gpa), e.cid, s.name
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid, s.name
```

✓

CMU SCS

HAVING

- Filters output results
- Like a **WHERE** clause for a **GROUP BY**

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid
  FROM enrolled AS e, student AS s
 WHERE e.sid = s.sid
 GROUP BY e.cid
 HAVING avg_gpa > 2.75;
```

AVG(s.gpa)	e.cid
2.46	Pilates101
3.39	Reggae203
2.98	Topology112
1.89	Massage105

→

avg_gpa	e.cid
3.39	Reggae203
2.98	Topology112

CMU SCS

All-in-One Example

- Store the total balance of the cities that have branches with more than \$1m in assets and where the total balance is more than \$700, sorted by city name in descending order.

```
SELECT bcity, SUM(balance) AS totalbalance
  INTO BranchAcctSummary
  FROM branch AS b, account AS a
 WHERE b.bname=a.bname AND assets > 1000000
 GROUP BY bcity
HAVING totalbalance >= 700
 ORDER BY bcity DESC
```

CMU SCS

All-in-One Example

Steps 1,2 : FROM, WHERE

b.bname	b.city	b.assets	a.bname	a.acct_no	a.balance
Downtown	Boston	\$9,000,000	Downtown	A-101	\$500
Compton	Los Angeles	\$2,100,000	Compton	A-215	\$700
Long Beach	Los Angeles	\$1,400,000	Long Beach	A-102	\$400
Harlem	New York	\$7,000,000	Harlem	A-202	\$350
Marcy	New York	\$2,100,000	Marcy	A-305	\$900
Marcy	New York	\$2,100,000	Marcy	A-217	\$750

59

CMU SCS

All-in-One Example

Step 3: GROUP BY
Step 4: SELECT

b.city	totalbalance
Boston	500
Los Angeles	1100
New York	2000

Step 5: HAVING

b.city	totalbalance
Los Angeles	1100
New York	2000

Step 6: ORDER BY

b.city	totalbalance
New York	2000
Los Angeles	1100

Step 7: INTO

< Store in new table >

60

CMU SCS

Summary

Clause	Evaluation Order	Semantics (RA)
SELECT[DISTINCT]	4	p* (or p)
FROM	1	X*
WHERE	2	s*
INTO	7	←
GROUP BY	3	Cannot Express
HAVING	5	s*
ORDER BY	6	Cannot Express

61

CMU SCS

Advantages of SQL

- Write once, run everywhere (in theory…)
  - Different DBMSs
  - Single-node DBMS vs. Distributed DBMS

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno = account.acctno
   AND account.amt > 1000
```

62

CMU SCS

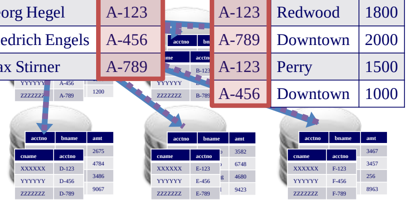
Distributed Execution

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno = account.acctno
   AND account.amt > 1000
```

Query R

cname	acctno
Georg Hegel	A-123
Friedrich Engels	A-456
Max Stirner	A-789

acctno	bname	amt
A-123	Redwood	1800
A-789	Downtown	2000
A-123	Perry	1500
A-456	Downtown	1000



63

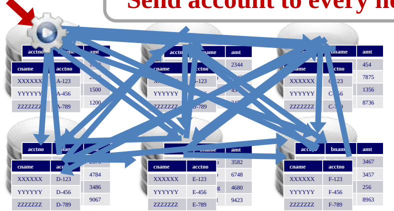
CMU SCS

Stupid Joins Are Stupid

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.cname = account.bname
   AND account.amt
```

Send customer to every node?
Send account to every node?

Query Request



64

CMU SCS

NoSQL

- NoSQL really means non-relational
  - Many NoSQL DBMSs are just key-value stores

key		name	login	age	gpa
53666	→	Faloutsos	christos@cs	45	1.8
53688	→	Bieber	jbieber@cs	21	3.9

  - Queries are often written in procedural code.
- Relax the guarantees of the relational model to gain better horizontal scalability.

mongoDB Redis CouchDB relax HBASE Cassandra

66

CMU SCS

NoSQL: *Not Only SQL!*

- Many NoSQL systems now support a SQL-like dialect.
  - Facebook's Hive (http://bit.ly/qId8np)
  - Cassandra CQL (http://bit.ly/nGJLtX)
- Other systems support declarative languages:
  - Yahoo's Pig + Hadoop (http://bit.ly/pLbhtN)

67

CMU SCS

Additional Information

- Online SQL validators:
  - http://developer.mimer.se/validator/
  - http://format-sql.com
- Links to Postgres, MySQL, and SQLite documentation will be posted.
- *When in doubt, try it out!*

CMU SCS 15-415/615 68

CMU SCS

Next Class

- DDLs
- Complex Joins
- Views
- Nested Subqueries
- Triggers
- Stored Procedures

CMU SCS 15-415/615 69