Applied Databases

Lecture 7
Simple SQL Queries

Sebastian Maneth

University of Edinburgh - February 6st, 2017

Outline

1. Structured Querying Language (SQL)
2. Creating Tables
3. Simple SQL queries

SQL

→ Developed in the 1970's at IBM by Chamberlin and Boyce
(originally "SEQUEL" = Structured English Query Language")

→ June 1979 first commercial version by Relational Software Inc (later Oracle)

→ ANSI standard in 1986

→ ISO standard in 1987, important releases 1992, 1999, 2003, 2008, 2011

→ MySQL attempts to comply with 2008 standard

→ Despite the existence of the SQL standards, most SQL code
is *not completely portable among different database systems*
without adjustments :-(

2. Creating Tables

```
CREATE TABLE <name> (attr1 type, …, attrN type);

CREATE TABLE Movies (title char(20),director char(10),actor char(10));
```

Types

→ char(*n*) - fixed length string of exactly *n* characters

→ varchar(*n*) - variable length string of at most *n* characters

→ int - signed integer (4 bytes)

→ smallint - signed integer (2 bytes, i.e., from -32768 to 32767)

→ float(M,D) / double(M,D) - floating-point (approximate value) types (4 / 8 bytes)

D digits after decimal point

up to M digits in total

Rounding!
insert 999.00009 into
float(7,4) gives 999.0001

2. Creating Tables

```
mysql> CREATE TABLE test (a float(3,3), b float(4,2), c float(5,1));
mysql> INSERT INTO test VALUES (100.999, 100.999, 100.999);
Query OK, 1 row affected, 2 warnings (0.01 sec)
mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1264 | Out of range value for column 'a' at row 1 |
| Warning | 1264 | Out of range value for column 'b' at row 1 |
+---------+------+----------------------------------------+
mysql> INSERT INTO test VALUES (2/3,2/3,2/3);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test;
+-------+-------+-------+
| a     | b     | c     |
+-------+-------+-------+
| 0.999 | 99.99 | 101.0 |
| 0.667 |  0.67 |   0.7 |
+-------+-------+-------+
2 rows in set (0.00 sec)
```

2. Creating Tables

```
CREATE TABLE <name> (attr1 type, …, attrN type);

CREATE TABLE Movies (title char(20),director char(10),actor char(10));
```

Types

→ char(*n*) - fixed length string of exactly *n* characters

→ varchar(*n*) - variable length string of at most *n* characters

→ int - signed integer (4 bytes)

→ smallint - signed integer (2 bytes, i.e., from -32768 to 32767)

→ float(M,D) / double(M,D) - floating-point (approximate value) types (4 / 8 bytes)

→ text / blob - text and binary strings (length <= 2^16 = 65536)

Types cont'd

→ **date** - date type

1 warning →

```
> CREATE TABLE T1 (col1 date PRIMARY KEY);
> INSERT INTO T1 VALUES ("2005-12-24");
> INSERT INTO T1 VALUES ("01-01-01");
> INSERT INTO T1 VALUES ("05-05-2010");
> SELECT * FROM T1;
+------------+
| col1       |
+------------+
| 2001-01-01 |
| 2005-12-24 |
| 0000-00-00 |
+------------+
```

MySQL

```
> CREATE TABLE T1 (col1 date PRIMARY KEY);
> INSERT INTO T1 VALUES ("01-01-2001");
> INSERT INTO T1 VALUES ("2001-01-01");
> INSERT INTO T1 VALUES ("Feb-2005");
> INSERT INTO T1 VALUES ("2005");
> SELECT * FROM T1;
2005
01-01-2001
2001-01-01
Feb-2005
```

sqlite3

Types cont'd

→ **time** - time type

```
> CREATE TABLE T1 (col1 time PRIMARY KEY);
> INSERT INTO T1 VALUES ("20:15");
> INSERT INTO T1 VALUES ("20:15:59");
> INSERT INTO T1 VALUES ("20:15:59:99");
ERROR 1062: Duplicate entry '20:15:59'
> SELECT * FROM T1;
+----------+
| col1     |
+----------+
| 20:15:00 |
| 20:15:59 |
+----------+
```

MySQL

```
> CREATE TABLE T1 (col1 time PRIMARY KEY);
> INSERT INTO T1 VALUES ("20:15");
> INSERT INTO T1 VALUES ("20:15:59");
> INSERT INTO T1 VALUES ("20:15:59:00");
> INSERT INTO T1 VALUES ("20:15:59:00:00");
> SELECT * FROM T1 WHERE col1>"20:15:59";
20:15:59:00
20:15:59:00:000
```

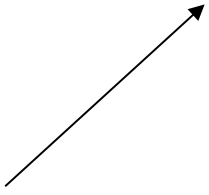
sqlite3

Types cont'd

→ **timestamp**

```
> CREATE TABLE T1 (col1 timestamp PRIMARY KEY);
> INSERT INTO T1 VALUES ("2001-12-24 11:18:00");
> INSERT INTO T1 VALUES ("2001-12-24 23:18:00");
> SELECT * FROM T1;
+---------------------+
| col1                |
+---------------------+
| 2001-12-24 11:18:00 |
| 2001-12-24 23:18:00 |
+---------------------+
```

MySQL



date / time / timestamp

→ can be compared for equality and less than (<)
→ if date1 < date2, then date 1 is earlier than date2

Tables

→ you can use queries for **insertion**

INSERT INTO T1 (SELECT … FROM … WHERE)

attributes of the result of the query must be same as those of T1.

→ MySQL is quite relaxed about this, it will often do the insertion….

→ sometimes unexpected behavior!

```
> CREATE TABLE T1 (a int, b int, c text);
> CREATE TABLE T2 (c1 text, c2 int, c3 int);
> INSERT INTO T1 VALUES (1,1,"a5c");
> INSERT INTO T1 VALUES (2,3,"7de");
> INSERT INTO T2 (SELECT * FROM T1);
> SELECT * FROM T2;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
| 1    |    1 |    0 |
| 2    |    3 |    7 |
+------+------+------+
```

Tables

→ you can use queries for **insertion**

INSERT INTO T1 (SELECT … FROM … WHERE)

attributes of the result of the query must be same as those of T1.

→ MySQL is quite relaxed about this, it will often do the insertion….

→ sometimes unexpected behavior!

```
> CREATE TABLE T1 (a int, b int, c text);
> CREATE TABLE T2 (c1 text, c2 int, c3 int);
> INSERT INTO T1 VALUES (1,1,"abc");
> INSERT INTO T1 VALUES (2,3,"7de");
> INSERT INTO T2 (SELECT c,b,a FROM T1);
> SELECT * FROM T2;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
| abc  |    1 |    1 |
| 7de  |    3 |    2 |
+------+------+------+
```

Tables

→ `DELETE FROM T1;` - delete all rows from table T1

→ `DELETE FROM T1 where c3=1;` - delete rows with c3-value equals 1

→ `DROP TABLE T1;` - remove table T1

→ `ALTER TABLE T1 ADD COLUMN col1 int;` - adds a column to table T1

→ `ALTER TABLE T1 DROP COLUMN col1;` - removes a column from table T1

→ `DESCRIBE T1;` - lists the fields and types of table t1 (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".schema t1" or "PRAGMA table_info(t1)")

→ `SHOW tables;` - lists tables of your database (**MySQL**, not SQL!)
(in **sqlite3** this is done via ".tables")

Tables

→ default values for some attributes:

```
CREATE TABLE T1 (  <attribute> <type> DEFAULT <value>  )
```

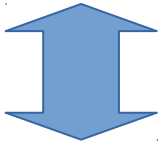
```
> CREATE TABLE T1 (col1 int DEFAULT 0, col2 int);
> INSERT INTO T1 VALUES (1,2);
> INSERT INTO T1 (col2) VALUES (5);
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    0 |    5 |
+------+------+
```

→ `ALTER TABLE Movies ADD COLUMN length int DEFAULT 0;`

Constraints

→ PRIMARY KEY – primary means of accessing a table
→ NOT NULL – specifies that NULL is a forbidden value for the attribute

```
CREATE TABLE Employee (
    employee_id int NOT NULL PRIMARY KEY,
    first_name char(20),
    last_name char(20),
    department char(10),
    salary int default 0
)
```



equivalent

```
CREATE TABLE Employee (
    employee_id int NOT NULL,
    first_name char(20),
    last_name char(20),
    department char(10),
    salary int default 0,
    PRIMARY KEY (employee_id)
)
```

Constraints

→ PRIMARY KEY – primary means of accessing a table
→ NOT NULL – specifies that NULL is a forbidden value for the attribute

```
CREATE TABLE Employee (
    employee_id int NOT NULL PRIMARY KEY,
    first_name char(20),
    last_name char(20),
    department char(10),
    salary int default 0
)
```

```
mysql> INSERT INTO Employee VALUES (NULL, "a", "b", "c", 5);
ERROR 1048 (23000): Column 'employee_id' cannot be null
mysql>
```

Constraints

→ PRIMARY KEY – primary means of accessing a table
→ NOT NULL – specifies that NULL is a forbidden value for the attribute

→ more than one key: UNIQUE

```
CREATE TABLE Employee (
    employee_id int NOT NULL,
    first_name char(20),
    last_name char(20),
    department char(10),
    salary int default 0,
    PRIMARY KEY (employee_id),
    UNIQUE (first_name,last_name)
)
```

→ checked in the same way as primary keys, i.e.,
→ forbids inserting same (first,last)-value more than once.

Checking Functional Dependencies

T = a table in BCNF

→ if X→Y is a nontrivial fd, then X is a superkey

Question

Does **UNIQUE X** enforce the fd X→Y?

Inclusion Dependencies

→ inclusion dependency T1[a1,a2,...,aN] SUBSET T2[b1,b2,...,bN]
means every (a1,...,aN)-projection of T1 is a (b1,...,bN)-projection of T2
"REFERENCES" keyword

→ often as part of a foreign key

```
CREATE TABLE Movies (title char(20),director char(10),actor char(10));

CREATE TABLE Schedule (title char(20) REFERENCES Movies(title),
                       theater char(20));
```

Inclusion Dependencies

→ inclusion dependency T1[a1,a2,...,aN] SUBSET T2[b1,b2,...,bN]
means every (a1,...,aN)-projection of T1 is a (b1,...,bN)-projection of T2
"REFERENCES" keyword

→ often as part of a foreign key

```
CREATE TABLE Person (first_name char(20) NOT NULL,
                     last_name char(20) NOT NULL,
                     PRIMARY KEY (first_name,last_name));

CREATE TABLE Employee (first char(20) NOT NULL,
                       last char(20) NOT NULL,
                       FOREIGN KEY (first,last)
                           REFERENCES Person(first_name,last_name));
```

Duplicates

In relational algebra, duplicate rows are not permitted.

→ in SQL, the are permitted.
→ most queries have Multiset Semantics, i.e., answers contain duplicates.

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+
> SELECT col1 FROM T1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    1 |
|    2 |
+------+
```

Duplicates

→ most queries have Multiset Semantics, i.e., answers contain duplicates.
→ not if you use set operators! E.g., UNION, INTERSECT, DIFFERENCE, etc or the DISTINCT operator

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT DISTINCT col1 FROM T1;
+------+
| col1 |
+------+
|    1 |
|    2 |
+------+
```

Duplicates

→ most queries have Multiset Semantics, i.e., answers contain duplicates.
→ not if you use set operators! E.g., UNION, INTERSECT, DIFFERENCE, etc or the DISTINCT operator

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 FROM T1 UNION SELECT col2 FROM T1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

Set Operations **with** Duplicates

→ add “ALL” keyword

→ e.g. “UNION ALL” instead of “UNION”

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 FROM T1 UNION ALL SELECT col2 FROM T1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    1 |
|    2 |
|    2 |
|    1 |
|    1 |
|    3 |
+------+
```

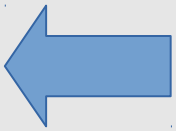
Set Operations **with** Duplicates

→ add “ALL” keyword

→ e.g. “UNION ALL” instead of “UNION”

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 FROM T1 UNION ALL SELECT col2 FROM T1;
+------+
| col1 |
+------+
|    1 |
|    2 |
|    1 |
|    2 |
|    2 |
|    1 |
|    1 |
|    3 |
+------+
```



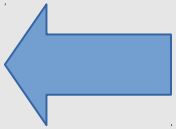
Why??

Set Operations **with** Duplicates

→ add “ALL” keyword
→ e.g. “UNION ALL” instead of “UNION”

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 AS d FROM T1 UNION ALL SELECT col2 FROM T1;
+------+
| d    |
+------+
|    1 |
|    2 |
|    1 |
|    2 |
|    2 |
|    1 |
|    1 |
|    3 |
+------+
```



Set Operations **with** Duplicates

→ add “ALL” keyword
→ “UNION ALL” – adds multiplicities
→ “INTERSECT ALL” – keeps **minimum** (non-0) number of occ's of an element

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 AS d FROM T1 INTERSECT ALL SELECT col2 FROM T1;

??
```

Set Operations **with** Duplicates

→ add “ALL” keyword
→ “UNION ALL” – adds multiplicities
→ “INTERSECT ALL” – keeps **minimum** (non-0) number of occ's of an element

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1 AS d FROM T1 INTERSECT ALL SELECT col2 FROM T1;
```

→ MySQL does not support INTERSECT and INTERSECT ALL

→ how can you formulate an “INTERSECT ALL” in MySQL??

Set Operations **with** Duplicates

→ add "ALL" keyword
→ "UNION ALL" – adds multiplicities
→ "INTERSECT ALL" – keeps **minimum** number of occurrences of an element
→ "EXCEPT ALL" – subtracts multiplicities

Empty Set Trap

→ want to compute: R intersect (S union T)
Assume R = S = { 1 } and T is the empty set.

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a OR R.a=T.a;
```

Returns empty! → why??

Set Operations with Duplicates

→ add "ALL" keyword
→ "UNION ALL" – adds multiplicities
→ "INTERSECT ALL" – keeps **minimum** number of occurrences of an element
→ "EXCEPT ALL" – subtracts multiplicities

Empty Set Trap

→ want to compute: R intersect (S union T)
Assume R = S = { 1 } and T is the empty set.

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a OR R.a=T.a;
```

Returns empty! → why??

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a;
```

Returns empty!

Set Operations with Duplicates

→ add "ALL" keyword
→ "UNION ALL" – adds multiplicities
→ "INTERSECT ALL" – keeps **minimum** number of occurrences of an element
→ "EXCEPT ALL" – subtracts multiplicities

Empty Set Trap

→ want to compute: R intersect (S union T)
Assume R = S = { 1 } and T is the empty set.

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a OR R.a=T.a;
```

Returns empty! → why??

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a;
```

Returns empty!

```
SELECT R.a FROM R
WHERE R.a IN (SELECT * FROM S) UNION (SELECT * FROM T);
+------+
| a    |
+------+
|    1 |
+------+
```

Set Operations with Duplicates

→ add "ALL" keyword
→ "UNION ALL" – adds multiplicities
→ "INTERSECT ALL" – keeps **minimum** number of occurrences of an element
→ "EXCEPT ALL" – subtracts multiplicities

Empty Set Trap

→ want to compute: R intersect (S union T)
Assume R = S = { 1 } and T is the empty set.

```
SELECT R.a FROM R, S, T
WHERE R.a=S.a OR R.a=T.a;
```

We select from the
Cartesian product of R, S, and T!
→ empty if one of the tables is empty!

3. Simple SQL Queries

```
SELECT     list, of, attributes
FROM       list of tables
WHERE      conditions
GROUP BY   list of attributes
ORDER BY   attribute ASC | DESC
```

optionally with aggregates

Aggregates: `COUNT, SUM, AVG, MIN, MAX`

Conditions: `AND, OR, NOT, IN, <, =, >, LIKE`

Combine tables (using set-semantics): `UNION, INTERSECT, EXCEPT`

→ query returns a table

→ where a table is allowed,
you can place a *nested* query: `(SELECT * FROM ... )`

SQL is NOT a programming language

- Calculate $2 + 2$ in SQL
- Step 1: there must be a table to operate with:

```
create table foo (a int)
```

- $2 + 2$ itself must go into selection. We also have to give it a name (attribute).
- Try:

```
db2 => select 2+2 as X from foo

X
-----------
  0 record(s) selected.
```

SQL is NOT a programming language cont'd

- Problem: there were no tuples in `foo`.
- Let's put in some:

```
insert into foo values 1
insert into foo values 5

 select 2+2 as X from foo

X
-----------
          4
          4

  2 record(s) selected.
```

```
INSERT INTO foo VALUES (1);
INSERT INTO foo VALUES (5);
SELECT 2+2 FROM foo;
+-----+
| 2+2 |
+-----+
|   4 |
|   4 |
+-----+
```

SQL is NOT a programming language cont'd

- It is also important to eliminate duplicates.
- So finally:

```
db2 => select distinct 2+2 as X from foo

X
-----------
          4

  1 record(s) selected.
```

No problem in MySQL

```
SELECT 2+2;
+-----+
| 2+2 |
+-----+
|   4 |
|   4 |
+-----+
```

More on the WHERE clause

- Once we have types such as strings, numbers, we have type-specific operations, and hence type-specific selection conditions
- ```
  create table finance (title char(20),
                        budget int,
                        gross int)

  insert into finance values ('Shining', 19, 100)
  insert into finance values ('Star wars', 11, 513)
  insert into finance values ('Wild wild west', 170, 80)
  ```
  ```

More on the WHERE clause

- Find movies that lost money:

```
select title
from finance
where gross < budget
```

- Find movies that made at least 10 times as much as they cost:

```
select title
from finance
where gross > 10 * budget
```

- Find profit each movie made:

```
select title, gross - budget as profit
from finance
where gross - budget > 0
```

More on the WHERE clause cont'd

- Is Kubrick spelled with a "k" or "ck" at the end?
- No need to remember.

```
SELECT Title, Director
FROM Movies
WHERE director LIKE 'Kubr%'
```

- Is Polanski spelled with a "y" or with an "i"?

```
SELECT Title, Director
FROM Movies
WHERE director LIKE 'Polansk_'
```

LIKE comparisons

- `attribute LIKE pattern`
- Patterns are built from:

  letters

  _ – stands for any letter

  % – stands for any substring, including empty
- Examples:

  `address LIKE '%Edinburgh%'`

  pattern '_a_b_' matches cacbc, aabba, etc

  pattern '%a%b_' matches ccaccbc, aaaabcbcbbd, aba, etc

Aggregates

→ count the number of tuples in Movies

```
SELECT COUNT(*) FROM Movies;
```

→ add up all movie lengths

```
SELECT SUM(length) FROM Movies;
```

Aggregates

→ find the number of directors

Naive approach:

```
SELECT COUNT(director) FROM Movies;
```

Returns the number of tuples in Movies

→ correct query (remove duplicates!)

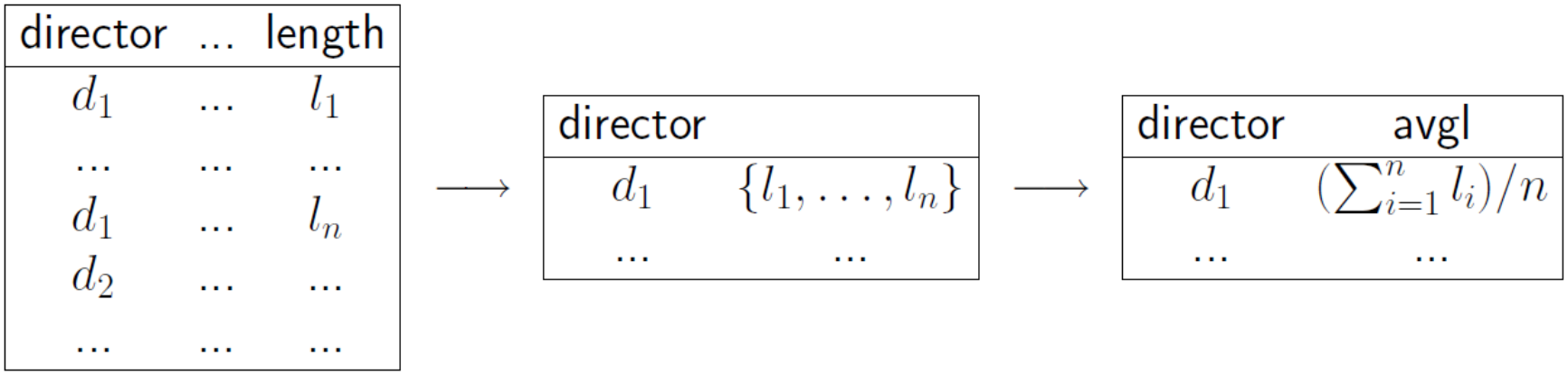
```
SELECT COUNT(DISTINCT director) FROM Movies;
```

Aggregation and Grouping

→ for each director return the average running time of his/her movies

```
SELECT director, AVG(length)
FROM Movies
GROUP BY director;
```

How does grouping work?



Aggregation and Grouping

→ for each director return the number of his/her movies

```
SELECT director, ???
FROM Movies
GROUP BY director;
```

How does grouping work?

director	...	length
d_1	...	l_1
...	...	...
d_1	...	l_n
d_2	...	...
...	...	...

$\longrightarrow$

director	
d_1	$\{l_1, \ldots, l_n\}$
...	...

$\longrightarrow$

director	avgl
d_1	$(\sum_{i=1}^{n} l_i)/n$
...	...

Aggregation and Grouping

→ generating histograms

```
> SELECT * FROM T1;
+------+------+
| col1 | col2 |
+------+------+
|    1 |    2 |
|    2 |    1 |
|    1 |    1 |
|    2 |    3 |
+------+------+

> SELECT col1,COUNT(col1) FROM T1 GROUP BY col1;
+------+-------------+
| col1 | COUNT(col1) |
+------+-------------+
|    1 |           2 |
|    2 |           2 |
+------+-------------+
```

Grouping wo Aggregation?

→ CAVE! Not sensible!!
→ No clearly defined semantics (implementation dependent)!

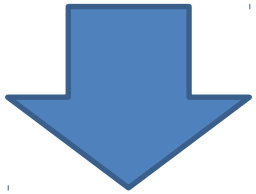
```
sqlite> select * from T;
1|1
2|3
1|5
2|7
3|3
1|5
sqlite> select a,b from T group by a;
1|5
2|7
3|3
sqlite> select * from R;
2|7
1|5
1|5
2|3
3|3
1|1
sqlite> select a,b from R group by a;
1|1
2|3
3|3
```

```
> SELECT col1,COUNT(col1) FROM T1 GROUP BY col1;
+------+-------------+
| col1 | COUNT(col1) |
+------+-------------+
|    1 |           2 |        R1
|    2 |           2 |
+------+-------------+
> SELECT col2,COUNT(col2) FROM T1 GROUP BY col2;
+------+-------------+
| col2 | COUNT(col2) |
+------+-------------+
|    1 |           2 |
|    2 |           1 |        R2
|    3 |           1 |
+------+-------------+
> SELECT Z.a,min(Z.b) FROM (
    SELECT R1.a,R1.b FROM R1 JOIN R2 ON R1.a=R2.a UNION ALL
    SELECT R2.a,R2.b FROM R1 JOIN R2 on R1.a=R2.a ) Z
  GROUP BY Z.a;
+------+----------+
| a    | min(Z.b) |
+------+----------+
|    1 |        2 |
|    2 |        1 |
+------+----------+
```

MySQL often requires a name for nested queries (even if not used elsewhere..)

Sample data: bibliography data (from XML)

```
<article mdate="2011-01-11" key="journals/acta/Milner96">
<author>Robin Milner</author>
<title>Calculi for Interaction.</title>
<year>1996</year>
.
.
</article>
```



AID	NAME
.	
.	
7	Robin Milner
8	
.	
.	

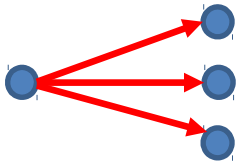
author.csv

PID	NAME	YEAR
.		
.		
13	Calculi for Interaction	1996
14		
.		
.		

paper.csv

PID	AID
.	
.	
13	7
8	
.	
.	

writtenBy.csv



```
> CREATE TABLE Author (aid int PRIMARY KEY,
                       name text);
> CREATE TABLE Paper (pid int PRIMARY KEY,
                      title text,
                      year int);
> CREATE TABLE WrittenBy (pid int,
                          aid int, PRIMARY KEY (pid,aid));
> CREATE INDEX ai ON Author (name);
> CREATE INDEX wi ON WrittenBy (pid, aid);


> .separator ";"
> .import author.csv author
> .import paper.csv paper
> .import writtenBy.csv writtenBy
> .timer on
```

sqlite3

A Join Query

```
-- title and year of papers by Robin Milner

SELECT P.title, P.year
FROM   Paper P, Author A, WrittenBy W
WHERE  A.name = "Robin Milner"
AND    A.aid = W.aid AND W.pid = P.pid;
Calculi for Interaction.;1996
Elements of Interaction - Turing Award Lecture.;1993
An Interview with Robin Milner.;1993
.
.
.
```

Join and Grouping

```
-- number of papers per year by aid=314 (Jeffrey D. Ullman)

SELECT P.year, COUNT(P.year) as count
FROM (Paper p JOIN WrittenBy W ON (P.pid = W.pid))
WHERE W.aid=314
GROUP BY year;

1966|1
1967|4
1968|8
1969|7
1970|6
1971|4
1972|12
.
.
```

Sorting (Ordering)

```
-- most prolific authors??

SELECT   A.name, count(W.pid) as count
FROM     Author A, WrittenBy W
WHERE    A.aid = W.aid
GROUP BY W.aid
ORDER BY count DESC LIMIT 40;
H. Vincent Poor|1114
Wei Wang|1064
Yan Zhang|999
Wei Liu|981
Wen Gao|926
Philip S. Yu|885
Thomas S. Huang|838
Chin-Chen Chang|795
Lajos Hanzo|790
Elisa Bertino|782
Wei Zhang|779
...
```

Co-Author Graph

Question

Give a SQL query that defines the co-author graph G

→ two authors a,b are in the relation G, if and only if
there exists a paper that was written by a, and was also written by b.

Expressive Power of SQL

SQL is not Turing-complete

→ there are many things that you cannot express in SQL

→ can you think of an example?

Expressive Power of SQL

SQL is not Turing-complete

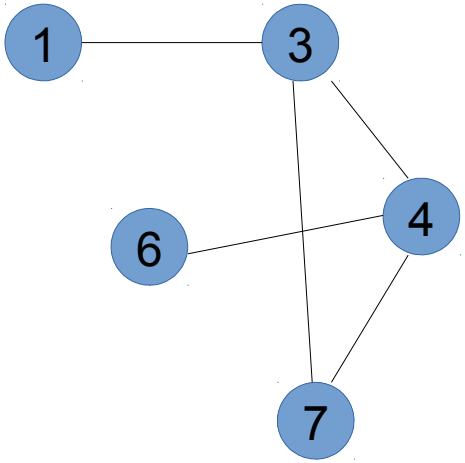
→ there are many things that you cannot express in SQL

→ can you think of an example?

Hint:

R
1 | 3
3 | 4
4 | 6
4 | 7
7 | 3

has a cycle?



Expressive Power of SQL

SQL is not Turing-complete

→ there are many queries you cannot express in SQL

→ why is that wanted?

→ what is the time complexity of evaluating a SQL query?

END
Lecture 7