Informatics 1: Data & Analysis

Lecture 8: SQL Queries

Ian Stark

School of Informatics
The University of Edinburgh

Friday 5 February 2016
Semester 2 Week 4

THE UNIVERSITY of EDINBURGH

Lecture Plan for Weeks 1–4

Data Representation

This first course section starts by presenting two common data representation models.

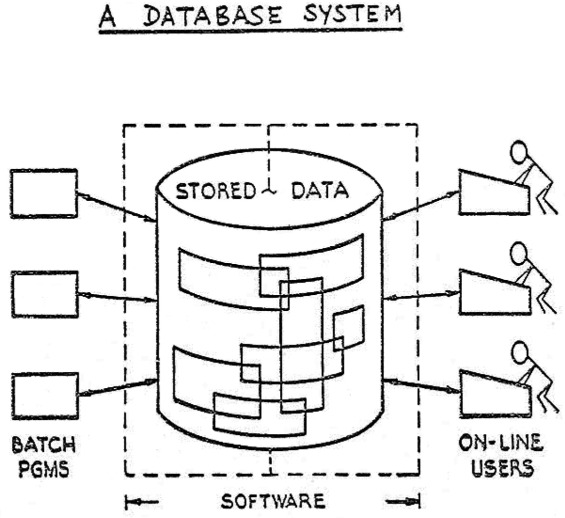
- The *entity-relationship (ER)* model
- The *relational* model

Data Manipulation

This is followed by some methods for manipulating data in the relational model and using it to extract information.

- *Relational algebra*
- The *tuple-relational calculus*
- The query language *SQL*

Remember Codd's Diagram?



ACID Transactions for Reliable Multiuser Databases

The idea of a *transaction* identifies a single coherent operation on a database. This might involve substantial amounts of data, or take considerable computation; but is intended to be an all-or-nothing action.

The features that characterise a reliable implementation of transactions are standardly initialized as the ACID properties.

- **Atomicity** All-or-nothing: a transaction either runs to completion, or fails and leaves the database unchanged.
- **Consistency** Applying a transaction in a valid state of the database will always give a valid result state.
- **Isolation** Concurrent transactions have the same effect as sequential ones: the outcome is as if they were done in order.
- **Durability** Once a transaction is committed, it will not be rolled back.

Implementation of these is especially challenging for databases that are widely distributed and with multiple simultaneous users.

NoSQL

Not every database uses or needs SQL and the relational model.

For these, there is NoSQL — or, less dogmatically, *Not Only SQL*.

NoSQL databases can be highly effective in some application domains: with certain kinds of data, or needing high performance for one operation.

Sometimes the strong guarantees and powerful language of RDBMS simply aren't needed, and alternatives do the job better.

Example NoSQL approaches

Key-value   Column-oriented   Document-oriented   Graph databases

Some of these weaken the ACID requirements – for example, offering only *eventual consistency* in exchange for greater decentralisation.

Balancing the tradeoffs here can be hard to assess, especially at extremes of size and speed. Strange things happen at scale.   e.g. Twitter *Snowflake*

Database Popularity

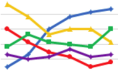
Ranking > Complete Ranking

RSS RSS Feed

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.



trend chart

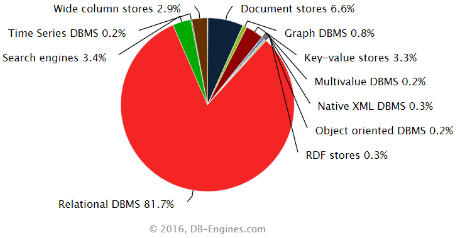
296 systems in ranking, February 2016

Rank			DBMS	Database Model	Score		
Feb 2016	Jan 2016	Feb 2015			Feb 2016	Jan 2016	Feb 2015
1.	1.	1.	Oracle	Relational DBMS	1476.14	-19.94	+36.42
2.	2.	2.	MySQL	Relational DBMS	1321.13	+21.87	+48.67
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1150.23	+6.16	-27.26
4.	4.	4.	MongoDB	Document store	305.60	-0.43	+38.36
5.	5.	5.	PostgreSQL	Relational DBMS	288.66	+6.26	+26.32
6.	6.	6.	DB2	Relational DBMS	194.48	-1.89	-7.94
7.	7.	7.	Microsoft Access	Relational DBMS	133.08	-0.96	-7.47
8.	8.	8.	Cassandra	Wide column store	131.76	+0.81	+24.68
9.	9.	9.	SQLite	Relational DBMS	106.78	+3.04	+7.22
10.	10.	10.	Redis	Key-value store	102.07	+0.92	+2.86

http://db-engines.com/en/ranking

Popularity of Different Kinds of Database

Ranking scores per category in percent, February 2016



This chart shows the popularity of each category. It is calculated with the popularity (i.e. the ranking scores) of all individual systems per category. The sum of all ranking scores is 100%.

http://db-engines.com/en/ranking_categories

SkyServer

http://skyserver.sdss.org/en
http://skyserver.sdss.org/en/sdss/telescope/telescope.aspx
http://skyserver.sdss.org/en/tools/search
http://skyserver.sdss.org/en/tools/search/sql.aspx

```
-- Find some stars near Hoag's Object
SELECT TOP 10
  p.objId,
  p.run, p.rerun, p.camcol, p.field, p.obj,
  p.type, p.ra, p.dec
FROM PhotoTag p, fGetNearbyObjEq(229.32878,21.57426,3) n
WHERE n.objID=p.objID AND p.type=3
```

```
-- How many stars can you see in the sky?
SELECT COUNT(*) FROM star
```

Students and Courses

Student

matric	**name**	**age**	**email**
s0456780	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	**code**	**mark**
s0456780	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	**title**	**year**
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Simple Query

Extract all records for students older than 19.

SELECT $*$
FROM Student
WHERE age $>$ 19

Returns a new table, with the same schema as Student, but containing only some of its rows.

$$\{\, S \mid S \in \text{Student} \wedge S.\text{age} > 19 \,\}$$

matric	name	age	email
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once,

Student

matric	name	age	email
s0456780	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	code	mark
s0456780	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

Student

matric	name	age	email
s0456780	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	code	mark
s0456780	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND  Takes.code = Course.code
   AND  Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

Student

matric	name	age	email
s0456780	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	code	mark
s0456780	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Student

matric	name	age	email
s0456780	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	code	mark
s0456780	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
dbs	Database Systems	3
adbs	Advanced Databases	4

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

name	email
Mary	mary@inf
Peter	peter@math

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Expressed in tuple-relational calculus:

$$\{\ R \mid \exists S \in \mathsf{Student}, T \in \mathsf{Takes}, C \in \mathsf{Course}\ .$$
$$R.\mathsf{name} = S.\mathsf{name} \wedge R.\mathsf{email} = S.\mathsf{email} \wedge S.\mathsf{matric} = T.\mathsf{matric}$$
$$\wedge\, T.\mathsf{code} = C.\mathsf{code} \wedge C.\mathsf{title} = \text{"Mathematics 1"}\ \}$$

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra,

$$\pi_{\text{name,email}}(\sigma_{\substack{\text{Student.matric} = \text{Takes.matric} \\ \wedge\, \text{Takes.code} = \text{Course.code} \\ \wedge\, \text{Course.name} = \text{"Mathematics 1"}}}(\text{Student} \times \text{Takes} \times \text{Course}))$$

Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE  Student.matric = Takes.matric
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra, in several possible ways:

$$\pi_{\text{name,email}}(\sigma_{\text{title="Mathematics 1"}}(\text{Student} \bowtie \text{Takes} \bowtie \text{Course}))$$

$$\pi_{\text{name,email}}(\text{Student} \bowtie (\text{Takes} \bowtie (\sigma_{\text{title="Mathematics 1"}}(\text{Course}))))$$

Nested Query

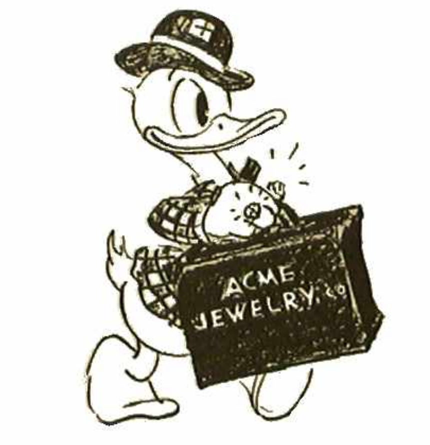
As **SELECT** both takes in and produces tables, we can use the result of one query in building another.

```
SELECT Student.name, Student.email
FROM Student, Takes,
(SELECT code FROM Course WHERE title='Mathematics 1') AS C
WHERE Student.matric=Takes.matric AND Takes.code=C.code
```

Inner query (**SELECT** code ...) **AS** C computes a table of course codes.

Adding nested queries does not change the expressive power of SQL; but it may make some queries more succinct, or easier to understand.

Of course, as usual, the execution plan used by a RDBMS to compute the query is quite independent of whether we use nested queries or not — it will rearrange and rewrite as necessary to reduce computation cost.


ACME
JEWELRY Co


Elgin
Fraserburgh
Peterhead
Inverurie
Aberdeen
Fort William
Montrose
Forfar
Arbroath
Dundee
Perth
St Andrews
Oban
Glenrothes
Kirkcaldy
Dunfermline
Greenock
Glasgow
Edinburgh
Livingston
Hamilton
Kilmarnock
Ayr
B
C
D

Travelling Salesman

Given a table of travel times between all n^2 pairs of towns:

- What is the quickest route to visit them all?
- Is there any route shorter than X?

Checking any route is fast.

However, there a lot of routes to check ($n!$).

Can we do it faster?

Not much, no. The fastest algorithm known takes time related to 2^n.

This exponential growth means that a small problem can quickly become very large.

Travelling Salesman

This travelling salesman problem is more widely applicable than the name suggests, and it also arises in:

- Commercial distribution logistics;
- Optimising chip layout;
- Assembling DNA sequence fragments; *etc.*

Travelling Salesman

The travelling salesman problem is **NP-hard**: one of a large class of equivalent computational challenges where:

- Checking a potential solution is quick...
- ...but there are a lot of potential solutions...
- ...and we don't know how to do it any faster.

There's a bounty on these. You find a fast way to solve travelling salesman, or a proof that there isn't one, and you collect $1M.

Apply to the *Clay Mathematics Institute*, 10 Memorial Boulevard, Providence, Rhode Island, USA.

Travelling Salesman

P.S. If you are an actual salesman:

- Distances based on real space simplify the general problem;
- There are fast algorithms which with very high probability return an answer very close to the optimum.

Other NP-hard problems are not be so easy to work around.

Limits of Computation

Various physical factors set upper bounds on how much computation is possible:

- The speed of light, c;
- Planck's constant, $\hbar$;
- Universal gravitational constant, G;
- Quantisation of energy states;
- Heisenberg uncertainty in observation;
- Mass/energy equivalence.

For a 1kg computer, this sets a limit on computation of 10^{50} operations per second on 10^{31} bits.

Working at 10^9K, "the ultimate laptop looks like a small piece of the big bang".

[Seth Lloyd, *Ultimate Physical Limits to Computation*, Nature 406:1045–154, August 2000]

Limits of Computation

Matter organised to provide the greatest possible computing power is fancifully known as computronium.

In the 1960's Hans-Joachim Bremermann was one of the first people to estimate upper limits to computation.

His *Bremermann limit* is the computation which could be performed using the earth, over the period of its existence so far.

This is around 10^{93} bits of computation.

That's enough to solve the travelling salesman problem for 300 cities.

But just the once.

Disjunction Query

Find the names of all students who are taking *either* Informatics 1 *or* Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.matric = T.matric AND T.code = C.code
   AND (C.title = 'Informatics 1'
          OR C.title = 'Mathematics 1')
```

Disjunction Query

Find the names of all students who are taking *either* Informatics 1 *or* Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.matric = T.matric AND T.code = C.code
    AND C.title = 'Informatics 1'

UNION

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.matric = T.matric AND T.code = C.code
    AND C.title = 'Mathematics 1'
```

Conjunction Query

Find the names of all students who are taking *both* Informatics 1 *and* Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T1, Course C1, Takes T2, Course C2
WHERE  S.matric = T1.matric AND T1.code = C1.code
    AND S.matric = T2.matric AND T2.code = C2.code
    AND C1.title = 'Informatics 1'
    AND C2.title = 'Mathematics 1'
```

Conjunction Query

Find the names of all students who are taking *both* Informatics 1 *and* Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.matric = T.matric AND T.code = C.code
    AND C.title = 'Informatics 1'

INTERSECT

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.matric = T.matric AND T.code = C.code
    AND C.title = 'Mathematics 1'
```

Difference Query

Find the names of all students who are taking Informatics 1 *but not* Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE  S.matric = T.matric AND T.code = C.code
   AND C.title = 'Informatics 1'

EXCEPT

SELECT S.name
FROM Student S, Takes T, Course C
WHERE  S.matric = T.matric AND T.code = C.code
   AND C.title = 'Mathematics 1'
```

Comparison Query

Find the students' names in all cases where one person scored higher than another in Mathematics 1.

```
SELECT S1.name AS "Higher", S2.name AS "Lower"
FROM Student S1, Takes T1, Student S2, Takes T2, Course C
WHERE  S1.matric = T1.matric AND T1.code = C.code
   AND S2.matric = T2.matric AND T2.code = C.code
   AND C.title = 'Informatics 1'
   AND T1.mark > T2.mark
```

Higher	Lower
Mary	Peter

Aggregates: Operations on Multiple Values

SQL includes a range of mathematical operations on individual values, like T1.mark $>$ T2.mark.

SQL also provides operations on whole collections of values, as returned in a **SELECT** query. There are five of these standard aggregate operations:

COUNT(val)	The number of values in the val field
SUM(val)	The total of all values in the val field
AVG(val)	The mean of all values in the val field
MAX(val)	The greatest value in the val field
MIN(val)	The least value in the val field

Particular RDBMS implementations may refine and extend these with other operations.

Aggregates: Operations on Multiple Values

SQL includes a range of mathematical operations on individual values, like T1.mark $>$ T2.mark.

SQL also provides operations on whole collections of values, as returned in a **SELECT** query. There are five of these standard aggregate operations:

COUNT(**DISTINCT** val)	The number of distinct values in the val field
SUM(**DISTINCT** val)	The total of the distinct values in the val field
AVG(**DISTINCT** val)	The mean of the distinct values in the val field
MAX(val)	The greatest value in the val field
MIN(val)	The least value in the val field

Particular RDBMS implementations may refine and extend these with other operations.

Aggregating Query

Find the number of students taking Informatics 1, their mean mark, and the highest mark.

```
SELECT  COUNT(DISTINCT T.matric) AS "Number",
        AVG(T.mark) AS "Mean Mark",
        MAX(T.mark) AS "Highest"
FROM Student S, Takes T, Course C
WHERE  S.matric = T.matric AND T.code = C.code
   AND C.title = 'Informatics 1'
```

Number	Mean Mark	Highest
1	82	82

Who Writes SQL?

SQL is one of the world's most widely used programming languages, but programs in SQL come from many sources. For example:

- Hand-written by a programmer
- Generated by some interactive visual tool
- Generated by an application to fetch an answer for a user
- Generated by one program to request information from another

Most SQL is written by programs, not directly by programmers.

The same is true of HTML, another domain-specific language.

Also XML, Postscript,...

Homework

Explore SkyServer, its different types of search, and try some SQL yourself. Work through the first two pages of the SQL Tutorial there.

http://skyserver.sdss.org/en

http://skyserver.sdss.org/en/help/howto/search