Informatics 1: Data & Analysis

Lecture 8: SQL Queries

Ian Stark

School of Informatics
The University of Edinburgh

Thursday 7 February 2019
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*

Outline

1. Basic SQL Queries
2. Transactions and Database Integrity
3. More Complex SQL Queries

Students and Courses

Student

uun	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

uun	code	mark
s0456782	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 only some of its rows.

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

uun	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.uun = Takes.uun
   AND Takes.code = Course.code
   AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once,

Student

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

Takes

uun	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.uun = Takes.uun
   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

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

Takes

uun	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.uun = Takes.uun
   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

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

Takes

uun	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.uun = Takes.uun
   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

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

Takes

uun	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.uun = Takes.uun
   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.uun = Takes.uun
   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:

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

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.uun = Takes.uun
   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,

$$\pi_{\text{name,email}}\left(\sigma_{\left(\begin{array}{c}\text{Student.uun} = \text{Takes.uun} \\ \wedge\ \text{Takes.code} = \text{Course.code} \\ \wedge\ \text{Course.name} = \text{"Mathematics 1"}\end{array}\right)}(\text{Student} \times \text{Takes} \times \text{Course})\right)$$

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.uun = Takes.uun
   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:

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

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

Query Evaluation

SQL **SELECT** queries are very close to a programming-language form for the expressions of the tuple relational calculus, describing the information desired but not dictating how it should be computed.

To do that computation, we need something more like relational algebra. A single **SELECT** statement combines the operations of join, selection and projection. This immediately suggests one possible strategy:

- Compute the complete cross product of all the **FROM** tables;
- Select all the rows which match the **WHERE** condition;
- Project out only the columns named on the **SELECT** line.

Real database engines don't do that. Instead, they use relational algebra to rewrite that procedure into a range of different possible *query plans*, estimate the cost of each — looking at indexes, table sizes, selectivity, potential parallelism — and then execute one of them.

Explicit Join in SQL

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

```
SELECT Student.name, Student.email
FROM Student  JOIN Takes ON Student.uun=Takes.uun
              JOIN Course ON Takes.code = Course.code
WHERE Course.title = 'Mathematics 1'
```

This is explicit **JOIN** syntax.

It has exactly the same effect as implicit **JOIN** syntax:

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

Summary: Starting SQL

SQL: Structured Query Language

A declarative language for interacting with relational databases. SQL provides facilities to define tables; to add, update, and remove tuples; and to query tables in complex ways.

Writing Queries

Queries can be used to extract individual items of data or simple lists; to build large tables combining several others; and to generate *views* on these.

SQL queries take a standard form: **SELECT** … **FROM** … **WHERE** … to identify the fields returned, the tables used, and which records to pick.

Executing Queries

Database engines prepare multiple *query plans* and estimate their cost (in memory space, disk I/O, time) before choosing one to execute.

Outline

1 Basic SQL Queries

2 Transactions and Database Integrity

3 More Complex SQL Queries

Homework from Tuesday

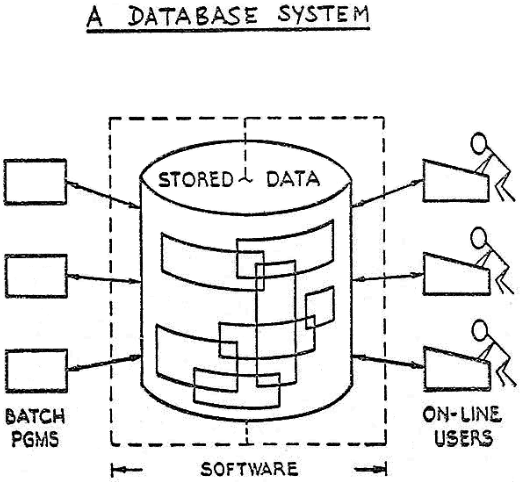
A *transaction* is a single coherent operation on a database. This might involve substantial amounts of data, or take considerable computation; but is meant to be an all-or-nothing action.

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

Task

Find out what each letter A C I D stands for here, and what those four terms mean.

Remember Codd's Diagram?



Homework from Tuesday

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

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

Initials

A — Atomicity

C — Consistency

I — Isolation

D — Durability

ACID Transactions for Reliable Multiuser Databases

Atomicity All-or-nothing: a transaction either runs to completion, or fails and leaves the database unchanged.

> This may involve a *rollback* mechanism to undo a partially-complete transaction.

Consistency Applying a transaction in a valid state of the database will always give a valid result state.

> This requires maintaining constraints and cascades; and rolling back a transaction if it will break any of these.

ACID Transactions for Reliable Multiuser Databases

Isolation Concurrent transactions have the same effect as sequential ones: the outcome is as if they were done in order.

> Transactions may, in fact, run at the same time: but should never see each other's intermediate state. In concurrent programming languages this is known as *sequential consistency*.

Durability Once a transaction is committed, it will not be rolled back.

> May need many levels: non-volatile memory; uninterruptible power; distributed commit protocols.

ACID Transactions for Reliable Multiuser Databases

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-store   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 Feed

DB-Engines Ranking

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

trend chart

Read more about the method of calculating the scores.

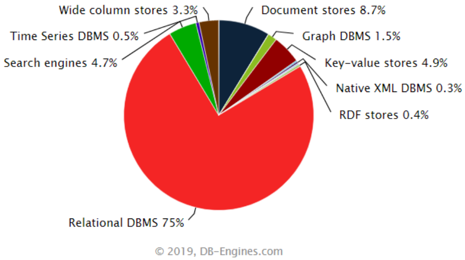
343 systems in ranking, February 2019

Rank Feb 2019	Rank Jan 2019	Rank Feb 2018	DBMS	Database Model	Score Feb 2019	Score Jan 2019	Score Feb 2018
1.	1.	1.	Oracle	Relational DBMS	1264.02	-4.82	-39.26
2.	2.	2.	MySQL	Relational DBMS	1167.29	+13.02	-85.18
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1040.05	-0.21	-81.98
4.	4.	4.	PostgreSQL	Relational DBMS	473.56	+7.45	+85.18
5.	5.	5.	MongoDB	Document store	395.09	+7.91	+58.67
6.	6.	6.	IBM Db2	Relational DBMS	179.42	-0.43	-10.55
7.	7.	↑ 8.	Redis	Key-value store	149.45	+0.43	+22.43
8.	8.	↑ 9.	Elasticsearch	Search engine	145.25	+1.81	+19.93
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	144.02	+2.41	+13.95
10.	10.	↑ 11.	SQLite	Relational DBMS	126.17	-0.63	+8.89

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

Popularity of Different Kinds of Database

Ranking scores per category in percent, February 2019



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%.

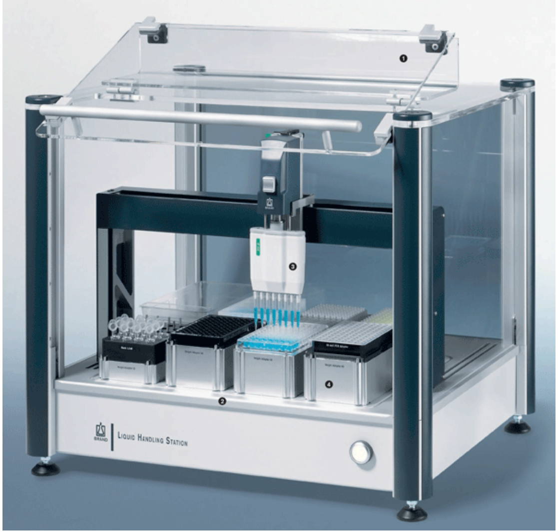
http://db-engines.com/en/ranking_categories

SkyServer

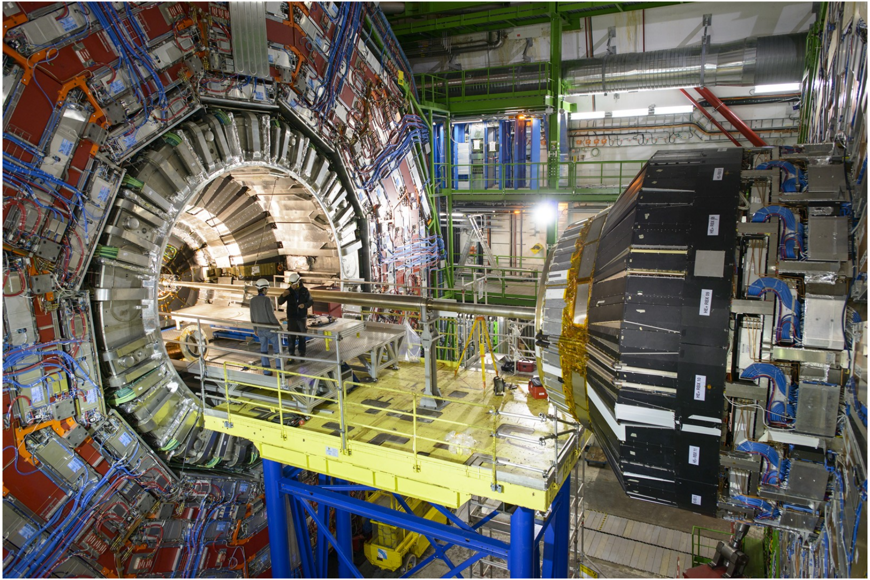
http://skyserver.sdss.org/en
http://skyserver.sdss.org/en/sdss/telescope/telescope.aspx
http://skyserver.sdss.org/en/tools/search
http://skyserver.sdss.org/en/tools/search/sql.aspx

```
-- Search near a spot in the sky
SELECT
s.specObjId,
s.class, s.subclass, s.ra, s.dec
FROM SpecObjAll s, fGetNearbyObjEq(229.329,21.574,30) n
WHERE s.bestobjid=n.objId
```

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



http://www.brand.de



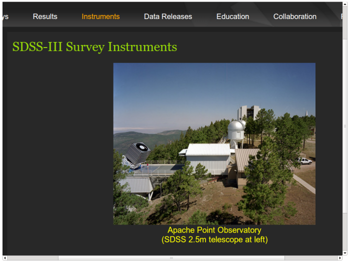
CERN — https://home.cern

Homework

Explore SkyServer, its different types of search, and try some SQL yourself. Work through the SQL Tutorial there up to at least the first set of practice questions.

http://skyserver.sdss.org/en

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



Finally: What about **http://is.gd/locatepluto** ?

Outline

1. Basic SQL Queries
2. Transactions and Database Integrity
3. More Complex SQL Queries

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, MathCourse

WHERE Student.uun=Takes.uun AND Takes.code=MathCourse.code
```

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.uun=Takes.uun 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.

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.uun = T.uun 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.uun = T.uun AND T.code = C.code
   AND C.title = 'Informatics 1'

UNION

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun 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.uun = T1.uun AND T1.code = C1.code
   AND S.uun = T2.uun 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.uun = T.uun AND T.code = C.code
    AND C.title = 'Informatics 1'

INTERSECT

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

EXCEPT

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun 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.uun = T1.uun AND T1.code = C.code
   AND S2.uun = T2.uun 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.uun) AS "Number",
       AVG(T.mark) AS "Mean Mark",
       MAX(T.mark) AS "Highest"
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
  AND C.title = 'Informatics 1'
```

Number	Mean Mark	Highest
263	65.66	100

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,. . .

Summary: More SQL

ACID Properties

Atomicity : Consistency : Isolation : Durability.

Characteristics that enable reliable use of *transactions*. Challenging to implement for widely distributed databases that serve many users.

More Queries

Nested queries: using one result table as input to another query.

Combining tables with **UNION**, **INTERSECT** and **EXCEPT**.

Aggregate Operations

SQL provides arithmetic operations and comparisons to use within queries. Aggregate operators take all the values in a multiset of results and combine them together.