Informatics 1: Data & Analysis

Lecture 7: SQL

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 4 February 2014
Semester 2 Week 4

Careers in IT

Job Fair

Wednesday 5 February 2014

Informatics Forum

1300–1600

http://is.gd/it_careers

Careers advice and stalls from 35+ local, national and international employers

NB Problems? !

Some students have reported the following symptoms with NB:

- Drag mouse over document
- Enter comment or question
- Hit “Submit” and **nothing happens**.

If this is you, mail me and I will pass to the developer to fix.

- Send me email Ian.Stark@ed.ac.uk
- Tell me what document you were trying to annotate.
- Were you viewing as “Guest”, or logged in? With what email address?

If you also mail me the question you wanted to ask, I’ll answer that too.

Lecture Plan

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*

SQL: Structured Query Language

- SQL is the standard language for interacting with relational database management systems
- Substantial parts of SQL are declarative: code states what should be done, not necessarily how to do it.
- When actually querying a large database, database systems take advantage of this to plan, rearrange, and optimize the execution of queries.
- Procedural parts of SQL do contain imperative code to make changes to the database.
- While SQL is an international standard (ISO 9075), individual implementations have notable idiosyncrasies, and code may not be entirely portable.

MySQL : PostgreSQL : Oracle : SQL Server : DB2 : SQLite : Sybase

SQL Data Manipulation Language

In an earlier lecture we saw the SQL Data Definition Language (DDL), used to declare the schema of relations and create new tables.

This lecture introduces the Data Manipulation Language (DML) which allows us to:

- Insert, delete and update rows in existing tables;
- Query the database.

Note that "query" here covers many different scales: from extracting a single statistic or a simple list, to building large tables that combine several others, or creating *views* on existing data.

SQL is a large and complex language. Here we shall only see some of the basic and most important parts. For a much more extensive coverage of the topic, sign up for the *Database Systems* course in Year 3.

Inserting Data into a Table

```
CREATE TABLE Student (
    matric  VARCHAR(8),
    name    VARCHAR(20),
    age     INTEGER,
    email   VARCHAR(25),
    PRIMARY KEY (matric) )
```

The following adds a single record to this table:

```
INSERT
    INTO Student (matric, name, age, email)
    VALUES ('s1428751', 'Bob', 19, 'bob@sms.ed.ac.uk')
```

For multiple records, repeat; or consult your RDBMS manual.

Strictly, SQL allows omission of the field names; but if we include them, then the compiler will check them against the schema for us.

Update and Delete Rows in a Table

Update

This command changes the name recorded for one student:

```
UPDATE Student
    SET name = 'Bobby'
    WHERE matric = 's1428571'
```

Delete

This deletes from the table all records for students named "Bobby":

```
DELETE
    FROM Students
    WHERE name = 'Bobby'
```

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.

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.

Student

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

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.

Student

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

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.

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

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.

Tuple-Relational Calculus

SQL is similar in form to the comprehensions of tuple-relational calculus:

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

Working out how to implement this efficiently through relational algebra operations is the job of an SQL compiler and database query engine.

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.

Variations

We can explicitly name the selected fields.

SELECT matric, name, age, email
FROM Student
WHERE age $>$ 18

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.

Variations

We can identify which table the fields are from.

SELECT Student.matric, Student.name, Student.age, Student.email
FROM Student
WHERE Student.age $>$ 18

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.

Variations

We can locally abbreviate the table name with an *alias*.

SELECT S.matric, S.name, S.age, S.email
FROM Student **AS** S
WHERE S.age $>$ 18

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.

Variations

We can save ourselves a very small amount of typing.

SELECT S.matric, S.name, S.age, S.email
FROM Student S
WHERE S.age $>$ 18

Anatomy of an SQL Query

SELECT *field-list*
FROM *table-list*
[**WHERE** *qualification*]

- The **SELECT** keyword starts the query.
- The list of fields specifies *projection*: what columns should be retained in the result. Using $*$ means all fields.
- The **FROM** clause lists one or more tables from which to take data.
- An optional **WHERE** clause specifies *selection*: which records to pick out and return from those tables.

Anatomy of an SQL Query

SELECT *field-list*
FROM *table-list*
[**WHERE** *qualification*]

The *table-list* in the **FROM** clause is a comma-separated list of tables to be used in the query:

```
...
FROM Student, Takes, Course
...
```

Each table can be followed by an alias Course **AS** C, or even just Course C.

Anatomy of an SQL Query

SELECT *field-list*
FROM *table-list*
[**WHERE** *qualification*]

The *field-list* after **SELECT** is a comma-separated list of (expressions involving) names of fields from the tables in **FROM**.

SELECT name, age
...
...

Field names can be referred to explicitly using table names or aliases: such as Student.name or C.title.

Anatomy of an SQL Query

SELECT *field-list*
FROM *table-list*
[**WHERE** *qualification*]

The *qualification* in the **WHERE** clause is a logical expression built from tests involving field names, constants and arithmetic expressions.

...
...
WHERE age $>$ 18 **AND** age $<$ 65

Expressions can involve a range of numeric, string and date operations.

Simple Query with Multiset Result

Extract all student ages.

```
SELECT age
FROM Student
```

Returns a new table, similar to Student, but containing only some of its columns.

Simple Query with Multiset Result

Extract all student ages.

```
SELECT age
FROM Student
```

Returns a new table, similar to Student, but containing only some of its columns.

Student

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

Simple Query with Multiset Result

Extract all student ages.

```
SELECT age
FROM Student
```

Returns a new table, similar to Student, but containing only some of its columns.

Student

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

Simple Query with Multiset Result

Extract all student ages.

```
SELECT age
FROM Student
```

Returns a new table, similar to Student, but containing only some of its columns.

age
18
20
18
22

Aside: Multisets

The relational model given in earlier lectures has tables as *sets* of rows: so the ordering doesn't matter, and there are no duplicates.

Actual SQL does allow duplicate rows, with a **SELECT DISTINCT** operation to remove duplicates on request.

Thus SQL relations are not sets but *multisets* of rows. A multiset, or *bag*, is like a set but values can appear several times. The number of repetitions of a value is its *multiplicity* in the bag.

The following are distinct multisets:

$$\lbag 2, 3, 5 \rbag \qquad \lbag 2, 3, 3, 5 \rbag \qquad \lbag 2, 3, 3, 5, 5, 5 \rbag \qquad \lbag 2, 2, 2, 3, 5 \rbag$$

Ordering still doesn't matter, so these are all the same multiset:

$$\lbag 2, 2, 3, 5 \rbag \qquad \lbag 2, 3, 2, 5 \rbag \qquad \lbag 5, 2, 3, 2 \rbag \qquad \lbag 3, 2, 2, 5 \rbag$$

Simple Query with Set Result

Extract all student ages.

```
SELECT DISTINCT age
FROM Student
```

Returns a new table, similar to Student, but containing only some elements from some of its columns.

Simple Query with Set Result

Extract all student ages.

```
SELECT DISTINCT age
FROM Student
```

Returns a new table, similar to Student, but containing only some elements from some of its columns.

Student

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

Simple Query with Set Result

Extract all student ages.

```
SELECT DISTINCT age
FROM Student
```

Returns a new table, similar to Student, but containing only some elements from some of its columns.

Student

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

Simple Query with Set Result

Extract all student ages.

```
SELECT DISTINCT age
FROM Student
```

Returns a new table, similar to Student, but containing only some elements from some of its columns.

age
18
20
22

Further Aside: Quotation Marks in SQL Syntax

SQL uses alphanumeric tokens of three kinds:

- Keywords: **SELECT**, **FROM**, **UPDATE**, ...
- Identifiers: Student, matric, age, S, ...
- Strings: 'Bobby', 'Informatics 1', ...

Each of these kinds of token has different rules about case sensitivity, the use of quotation marks, and whether they can contain spaces.

While programmers can use a variety of formats, and SQL compilers should accept them, programs that *generate* SQL code may be very cautious in what they emit and use apparently verbose formats.

Further Aside: Know Your Syntax

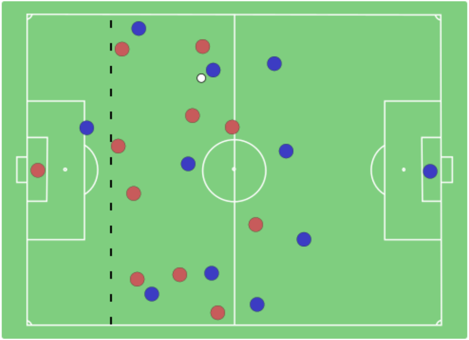
		Case sensitive?	Spaces allowed?	Quotation character?	Quotation Required?
Keywords	**FROM**	No	Never	None	No
Identifiers	Student	Maybe	If quoted	"Double"	If spaces
Strings	'Bob'	It depends	Yes	'Single'	Always

For example:

> **select** matric
> **from** Student **as** "Student Table"
> **where** "Student Table".age $>$ 18 **and** "name" $=$ 'Bobby Tables'

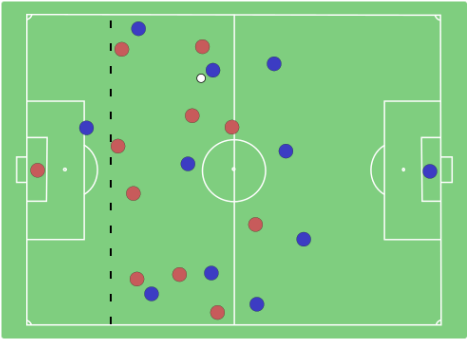
It's always safe to use only uppercase keywords and put quotation marks around all identifiers. Some tools will do this automatically.

Entirely Offside



NielsF, Wikimedia Commons

Entirely Offside



NielsF, Wikimedia Commons

The blue forward on the left of the diagram is in an offside position as he is in front of both the second-to-last defender (marked by the dotted line) and the ball. Note that this does not necessarily mean he is committing an *offside offence*; it only becomes an offence if the ball were to be played to him at this moment, whether or not he is in an offside position when he receives the ball, as he could receive the ball in an *onside position* but he'd still have committed an *offside offence*.

(FIFA guidelines 2003; IFAB Law XI 2005; Clarified 2010; Explained by Wikipedia)

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

Students and Courses

Student

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

Takes

matric	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

Example Query

Find the names of all students who are taking Mathematics 1

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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Takes

matric	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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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
Mary
Peter

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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:

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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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,

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

Example Query

Find the names of all students who are taking Mathematics 1

```
SELECT Student.name
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:

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

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

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, which immediately suggests one 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.

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, which immediately suggests one 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.

Crucially, 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 of all students who are taking Mathematics 1

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