L2: SQL Basics

CS1106/CS6503: Intro to Relational Databases

Dr Kieran T. Herley

Semester One, 2023-24

School of Computer Science & Information Technology
University College Cork

Summary

Review of relation model. Simple SELECT-FROM and SIMPLE-FROM-WHERE queries. SQL's operators. Queries involving dates and text.

Setting The Scene

Our Running Example

students

id_number	*first_name*	*last_name*	*date_of_birth*	*hometown*	*course*	*points*
112345678	Aoife	Ahern	1993-01-25	Cork	ck401	500
112467389	Barry	Barry	1980-06-30	Tralee	ck402	450
112356489	Ciara	Callaghan	1993-03-14	Limerick	ck401	425
112986347	Declan	Duffy	1993-11-03	Cork	ck407	550
112561728	Eimear	Early	1993-07-18	Thurles	ck406	475
112836467	Fionn	Fitzgerald	1994-06-13	Bandon	ck405	485

DB terminology: databases. tables. attributes. domains

Brief Note on Naming Conventions

SQL Rules Names for databases, tables and attributes:

- start with letter
- composed of: letters, digits and underscores
- NB: no internal spaces etc.

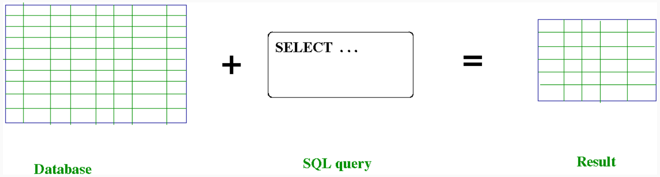
[1]

Conventions

- Use lower-case letters ('a'-'z') for names
- (Use upper-case for keywords, SELECT *etc.*)
- Names should be concise but *meaningful*, i.e. suggestive of what they represent
    - Good: `id_number`
    - Bad: `x`, `y16id`, `id_of_student_in_question`

[1] Actual SQL rules more lenient, but we we will stick to above

SQL Queries



- SQL query:
  - Specifies what info. we require from database table(s)
  - Expressed in SQL's fussy rules (syntax)
- Result:
  - Result is itself a table
  - Simple SELECT queries leave database unchanged

SELECT-FROM Queries

SELECT-FROM Queries

Template

```
SELECT list-of-attributes
FROM table-name;
```

list-of-attributes list of columns of interest; comma separated

table-name specifies table

semicolon we will terminate each query with one

Some Examples

```
SELECT id_number
FROM students;
```

Produces one-column result table with id numbers

```
SELECT *
FROM students;
```

$*$ specifies all columns

```
SELECT first_name, last_name
FROM students;
```

Produces two-column result; Note comma!

Some Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students;
```

Some Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students;
```

```
SELECT fisrt_name, last_name
FROM students;
```

Wrong! Don't misspell keywords or names

Some Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students;
```

```
SELECT fisrt_name, last_name
FROM students;
```

Wrong! Don't misspell keywords or names

```
SELECT id_number, points,
FROM students;
```

Wrong! Commas *between* attribute names

Some Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students;
```

```
SELECT fisrt_name, last_name
FROM students;
```

Wrong! Don't misspell keywords or names

```
SELECT id_number, points,
FROM students;
```

Wrong! Commas *between* attribute names

```
SELECTid_number, points
FROM students;
```

Wrong! Need space between words/names

Some Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students;
```

```
SELECT fisrt_name, last_name
FROM students;
```

Wrong! Don't misspell keywords or names

```
SELECT id_number, points,
FROM students;
```

Wrong! Commas *between* attribute names

```
SELECTid_number, points
FROM students;
```

Wrong! Need space between words/names

```
FROM students
SELECT id_number, points;
```

Wrong! SELECT clause first, then FROM

Distinctness

```
SELECT course
FROM students;
```

course
ck401
ck402
ck401
ck407
ck406
ck405

- Tuples in relations should be distinct . . .
- But result table contains duplicates
- (Irritating discrepancy between relation model and SQL implementations)

Distinctness

```
SELECT course
FROM students;
```

course
ck401
ck402
ck401
ck407
ck406
ck405

- Tuples in relations should be distinct . . .
- But result table contains duplicates
- (Irritating discrepancy between relation model and SQL implementations)

```
SELECT DISTINCT course
FROM students;
```

course
ck401
ck402
ck407
ck406
ck405

- Include keyword DISTINCT to suppress duplicates

SELECT-FROM-WHERE Queries

SELECT-FROM-WHERE Queries

Template

```
SELECT list-of-attributes
FROM table-name
WHERE condition;
```

list-of-attributes, table as before

condition Test of form *attribute-name op. value* to filter rows of interest (*op* is an operator e.g. $=$ or $<$).

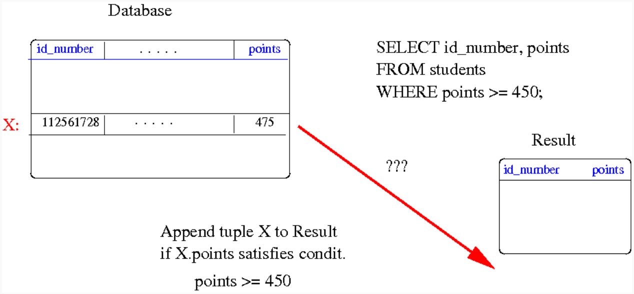
Some Examples

```
SELECT id_number
FROM students
WHERE points = 475;
```

```
SELECT first_name, last_name
FROM students
WHERE points >= 550;
```

```
SELECT first_name, last_name, points
FROM students
WHERE course = 'ck401' ;
```

Meaning



- For each tuple X of table in turn
    - Check whether X's values satisfy condition in query's WHERE condition (Here points $>=$ 450)
    - If they do, append copy (specified columns) of tuple X into result; otherwise ignore.
- Possible result may be empty (say of condition was points $>=$ 750)

The Main SQL Operators

Op.	Meaning	Example
=	Equal to	points = 450
<>	Not equal to	points <> 450
<	Less than	points < 450
<=	Less than or equal to	points <= 450
>	Greater than	points > 450
>=	Greater than or equal to	points >= 450
BETWEEN	Between	points BETWEEN 350 AND 45

- Note two-letter combinations for $\leq$ and $\neq$ etc.
- BETWEEN and AND are keywords

Conditions Involving Dates

- Can also write conditions involving dates, e.g.

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth < '1980—01—01';
```

  Extract names (first, last) of students born before 1980.
- Operators have obvious interpretations:
  - date_of_birth $<$ '1980-01-01' means "born before 1 Jan. 1980"
  - date_of_birth BETWEEN'1980-01-01' AND '1980-12-31' means "born during 1980" (includes first and last)
- Date constant format
  - Format: `YYYY-MM-DD` and wrapped in single quotes
  - Good: `'2012-10-10'`, `'2012-12-25'`, `'2013-01-01'`
  - Bad: `'10/10/2012'`, `'12-10-10'`, `'2013-1-1'`

Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= '1992-10-10';
```

Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= '1992-10-10';
```

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= 1992-10-10;
```

Wrong! Don't omit quotes

Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= '1992-10-10';
```

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= 1992-10-10;
```

Wrong! Don't omit quotes

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth BETWEEN '1992-01-01' AND '1992-12-31';
```

Examples (and Non-Examples)

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= '1992—10—10';
```

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth >= 1992—10—10;
```

Wrong! Don't omit quotes

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth BETWEEN '1992—01—01' AND '1992—12—31';
```

```
SELECT first_name, last_name
FROM students
WHERE date _of_birth > = '1992—10—10';
```

Wrong! No space inside operators

Working With Textual Data

Conditions Involving Text

- Can also use operators with text
- Example:

```
SELECT *
FROM students
WHERE first_name = 'Kieran';
```

- Note quotes around *string* 'Kieran' ("Kieran" also OK)
- Subtleties regarding what = or < etc. mean:
    - SQLite views `'Kieran'`, `'kieran'` and `'KiErAn'` as different
    - but also `'Kieran'`, `'Kieran '` and `' Kieran'`

SQLite and Case-Sensitivity

Case-sensitivity In some computing contexts upper-case letters (‘A’-‘Z’) are treated as being identical to their lower-case counterparts (‘a’-‘z’), but in others they are treated as distinct.

Examples

- Linux file names are case sensitive (so ’mywebpage.html’ and ’MyWebpage.html’ different files)
- Windows file names (generally) are not

SQLite and Case-Sensitivity

Case-sensitivity In some computing contexts upper-case letters (‘A’-‘Z’) are treated as being identical to their lower-case counterparts (‘a’-‘z’), but in others they are treated as distinct.

Examples

- Linux file names are case sensitive (so ’mywebpage.html’ and ’MyWebpage.html’ different files)
- Windows file names (generally) are not

SQLite

- Insensitive for keywords and names
- Sensitive for strings (by default)
- (Other SQL dialects may differ)

SQL and Case-Sensitivity cont'd

	Preferred	Also Legal	cs1106 Convention
Keywords	SELECT	select, SeLeCt	Use upper-case
DB/ Table names	students	DBMS dependent	Use lower-case
Attribute names	id_number		Use lower-case
Strings	'Aoife'		Use "natural" capitalization

Some SQL dialects ignore capitalization in text, you should preserve the "natural" capitalization of the text for readability:

- 'Fred Snodgrass'
- '123 High Street, Cork, Ireland'
- 'Jack and Jill went up the hill . . .'

Comparing Strings

- In everyday life we use "dictionary ordering" to impose an ordering on words based on the natural alphabetical ordering $\sqcup < a < b < c < d \cdots < z$. [2]
  - Words are ordered by their first letter (alphabetically)

    $$\text{aardvark} \; < \cdots < \; \text{baboon} \; \cdots \; < \; \text{cat} \cdots \; < \; \text{zebra}$$

  - Words with the same first letter are ordered by their second letters

    $$\text{aardvark} \; \cdots < \; \text{anaconda} \; \cdots \; < \; \text{armadillo} \cdots$$

  - Words with the same first and second letters are ordered by their third letters and so on; any word is ordered after any strict prefix (so 'computer' $<$ 'computers')
- SQL extends this idea to provide an ordering for text incorporating non-letter symbols (by interpreting such symbols as honorary "letters" in an expanded alphabet)

[2] Symbol $\sqcup$ denotes a space.

Apples and Oranges

- Beware of comparisons involving different types

```
SELECT first_name, last_name
FROM students
WHERE points = 'lots';
```

- Values in points column are integers not strings
- Above query satisfies SQL's rules, but makes no sense; resturns empty results table
- To be avoided– sometimes give unexpected results

One Last Thing

- List the names and points for all students named `O’Reilly`
- Our first attempt

```
SELECT first_name, last_name, points
FROM students
WHERE last_name = 'O'Reilly';
```

One Last Thing

- List the names and points for all students named `O’Reilly`
- Our first attempt

```
SELECT first_name, last_name, points
FROM students
WHERE last_name = 'O'Reilly';
```

- Wrong!SQL complains about “syntax error”
- What’s wrong? Quote within string causes problems
- Use either of the following instead

```
/* use double single quotes */
SELECT . . . WHERE last_name = 'O''Reilly';

/* precede quote with backslash */
SELECT . . . WHERE last_name = 'O\'Reilly';
```