INTRODUCTION TO SQL

CS-A1153 - Databases (Summer 2020)

LUKAS AHRENBERG

Aalto University
School of Science

DATABASE OPERATIONS & SQL

- Defining new tables
- Adding rows (a.k.a records) to tables
- Query (retrieving information from) the database tables
- SQL is a language to express such operations

SQL

- Structured Query Language
    - Sometimes pronounced 'sequel'
- Roots in work on relational data models at IBM & Relational Software Inc in the 1970's
- Declarative (we say what we want, not exactly how to do it)
- First commercialized and standardized in the 1980's
    - New standards released since
- Many different 'dialects'
- Understood by most major DB systems
    - Oracle DB
    - MariaDB
    - sqlite
    - PostgreSQL
    - … **And many more**

COURSE SPECIFICS

- [U&W 6:1-6:2]
- Try the examples!
  - You can use SQLiteStudio (free & open source)
  - The database can be created using this file from A+
  - Instructions how to use it with SQLiteStudio

SELECT - MAKING SQL QUERIES

Basic form:

```
SELECT <Attributes> FROM <Relations> WHERE <Conditions>;
```

Attributes

is one or more columns

Relations

is one or more tables

Condition

is a predicate expression controlling tuple selection

(Don't forget the semicolon **;** !)

SELECT EXAMPLES : MOVIESTAR RELATION

`MovieStar(name,address,gender,birthdate)`

name	address	gender	brirthdate
Kate Capshaw	56 Main St., Beverly Hills	F	1953-11-03
Harrison Ford	789 Palm Dr., Beverly Hills	M	1942-07-13
Calista Flockhart	789 Palm Dr., Beverly Hills	F	1964-11-11
Richard Attenborough	Harley St., London	M	1923-08-29
Julia Roberts	Oddstreet 1	F	1967-10-28
Richard Gere	Oddstreet 2	M	1939-08-31
Mike Myers	Oddstreet 12	M	1963-05-25
Sofia Coppola	28 Some St.	F	1971-05-14
Carrie Fisher	123 Maple St., Hollywood	F	1956-10-21
Mark Hamill	456 Oak Rd., Brentwood	M	1951-09-25
Clint Eastwood	4 Maple St., Hollywood	M	1930-05-31
Audrey Hepburn	Tolochenaz	F	1929-05-04

SELECT EXAMPLE 1

Get name and address of all male movie stars in table:

```
SELECT name,address FROM MovieStar WHERE gender = 'M';
```

name	address
Harrison Ford	789 Palm Dr., Beverly Hills
Richard Attenborough	Harley St., London
Richard Gere	Oddstreet 2
Mike Myers	Oddstreet 12
Mark Hamill	456 Oak Rd., Brentwood
Clint Eastwood	4 Maple St., Hollywood

SELECT EXAMPLE 2

Can leave out **WHERE** if we want all rows:

Get name and address of all movie stars in table.

```
SELECT name,address FROM MovieStar;
```

name	address
Kate Capshaw	56 Main St., Beverly Hills
Harrison Ford	789 Palm Dr., Beverly Hills
Calista Flockhart	789 Palm Dr., Beverly Hills
Richard Attenborough	Harley St., London
Julia Roberts	Oddstreet 1
Richard Gere	Oddstreet 2
Mike Myers	Oddstreet 12
Sofia Coppola	28 Some St.
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood
Clint Eastwood	4 Maple St., Hollywood
Audrey Hepburn	Tolochenaz

SELECT EXAMPLE 3

The wild-card * may be used if all columns are wanted. The following basically shows the whole table:

Get all columns of all records.

```
SELECT * FROM MovieStar;
```

name	address	gender	birthdate
Kate Capshaw	56 Main St., Beverly Hills	F	1953-11-03
Harrison Ford	789 Palm Dr., Beverly Hills	M	1942-07-13
Calista Flockhart	789 Palm Dr., Beverly Hills	F	1964-11-11
Richard Attenborough	Harley St., London	M	1923-08-29
Julia Roberts	Oddstreet 1	F	1967-10-28
Richard Gere	Oddstreet 2	M	1939-08-31
Mike Myers	Oddstreet 12	M	1963-05-25
Sofia Coppola	28 Some St.	F	1971-05-14
Carrie Fisher	123 Maple St., Hollywood	F	1956-10-21
Mark Hamill	456 Oak Rd., Brentwood	M	1951-09-25
Clint Eastwood	4 Maple St., Hollywood	M	1930-05-31
Audrey Hepburn	Tolochenaz	F	1929-05-04

SELECTING FROM MULTIPLE TABLES

- Multiple tables may be listed after **FROM**
  - Comma separated
- This allows columns to be selected from the table product
  - Emulating **Cartesian product** and selection

EXAMPLE - NAME AND ADDRESS OF STAR WARS ACTORS

MovieStar(name,address,gender,birthdate)
StarsIn(movieTitle,movieYear,starName)

(Query name and address of Star Wars actors on record. The effect is a Cartesian product of the two relations, followed by selection and projection.)

```
SELECT starName,address FROM MovieStar,StarsIn
WHERE movieTitle = 'Star Wars' AND name = starName;
```

starName	address
Harrison Ford	789 Palm Dr., Beverly Hills
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood

Note: This is a **Cartesian product**, not a join, so it will produce **all** parings unless you write the proper **WHERE** condition. E.g. Leave out `name = starName`: 36 tuples; Leave out conditions completely: 156 tuples. Try it!

A WORD ON SQL STYLE

- SQL keywords are not sensitive to case, but by convention:
  - use **ALL CAPS for KEYWORDS**
- Comments
  - Multi-line comments using C-style `/* This is a comment. */`
  - Single line comments prefixed by `--`
- Statements can be broken up over several lines (ended by `;`)

```
/*
Movie star records for:
+ males
+ born latest 1950.
*/
SELECT name,birthdate -- Only need name and birthdate
       FROM MovieStar
       WHERE gender='M' AND birthdate <= 1950
             ; -- Always end with a semicolon!
```

WRITING A QUERY

Answer the following questions:

1. What tables do you need to get the answer?
    - List them after **FROM**
    - Now imagine that you have every possible combination of rows in these
2. Which combinations will contain the answer?
    - Think of constraints for these combinations and encode them in the **WHERE** clause
3. Which columns are relevant to the answer?
    - Enter them after **SELECT**

WRITING WHERE PREDICATES

- Compare values
  - =, < , <=, > , >=, <>
  - <> (Not equal to.)
    - (Note some dialects also allow for the more familiar !=)
- Arithmetic expressions
  - +, -, *, /.
  - E.g. `PC.price/2 < Laptop.price`
- Logical operators
  - **AND**, **OR**, **NOT**

EXAMPLE - GET SUBSET OF LAPTOPS

Laptop(model,speed,ram,hd,screen,price)

(Get Laptop's with screen larger than 15" and either speed > 2.5 GHz or ram >= 8192.)

```
SELECT * FROM Laptop
        WHERE (speed > 2.5 OR ram >= 8192) AND screen > 15;
```

model	speed	ram	hd	screen	price
2001	2.5	8192	750	17.3	3534
2002	2.3	8192	128	17.3	949
2003	2.3	8192	128	15.6	599
2005	2.6	16384	1000	17.3	2499
2006	2.2	16384	256	15.4	2199
2008	2.5	8192	192	15.6	949
2010	2.5	8192	256	15.6	2352

COMPARING STRINGS

- Strings can also be compared, but when using <, <=, >, >= lexicographic order is imposed
- For more complicated expressions the command **LIKE** can be used when comparing strings
- Basic format `<string attribute>` **LIKE** `<pattern string>`
    - Wildcards may be used
        - `_` matches any single character
        - `%` matches any number of any character

EXAMPLE 1 - COMPARING STRINGS WITH LIKE

(Get titles and studio for all Movies made by a studio beginning with the letter 'D'.)

```
SELECT title,studioName FROM Movies
       WHERE studioName LIKE 'D%';
```

title	studioName
Pretty Woman	Disney
Mary Poppins	Disney
Prince Caspian	Disney
Letters from Iwo Jima	DreamWorks

EXAMPLE 2 - COMPARING STRINGS WITH LIKE

(Get titles and studio for all Movies made by a studio having the letter 'a' second in name.)

```
SELECT title,studioName FROM Movies
       WHERE studioName LIKE '_a%';
```

title	studioName
Gone with the Wind	Warner
Waynes World	Paramount
Sabrina	Paramount
Love Story	Paramount
The Godfather, Part III	Paramount
The Colour Purple	Warner
Sabrina	Paramount
Raiders of the Lost Ark	Paramount
Million Dollar Baby	Warner

SELECTING DISTINCT ROWS

- The result of a query can contain multiple copies of a tuple
  - This is because **SELECT** returns all rows which matches the query
- To get only unique rows add **DISTINCT**

```
SELECT DISTINCT <Attributes>
       FROM <Relations>
       WHERE <Conditions>;
```

EXAMPLE - SELECT DISTINCT

Laptop(model,speed,ram,hd,screen,price)

(Query available clock speed and RAM combinations in Laptop.)

```
SELECT speed,ram
       FROM Laptop;
```

speed	ram
2.5	8192
2.3	8192
2.3	8192
2.7	8192
2.6	16384
2.2	16384
1.6	2048
2.5	8192
2.5	8192
2.5	8192
2.0	4096
2.0	4096

```
SELECT DISTINCT speed,ram
       FROM Laptop;
```

speed	ram
2.5	8192
2.3	8192
2.7	8192
2.6	16384
2.2	16384
1.6	2048
2.0	4096

DUPLICATE ATTRIBUTE NAMES

- When selecting from multiple tables it can happen that these share attribute names
- Use 'dot' notation to indicate which one you are referring to
  - `<table>.<attribute>`
- E.g. `Laptop.speed > PC.speed` if relations PC and `Laptop` both have attribute `speed`

EXAMPLE DOT NOTATION

Laptop(model,speed,ram,hd,screen,price)
PC(model,speed,ram,hd,price)

(Pick out model and price for Laptop's which are faster than at least one PC while costing the same or less. Remove duplicates.)

```
SELECT DISTINCT Laptop.model,Laptop.price
       FROM Laptop,PC
       WHERE Laptop.speed > PC.speed
       AND Laptop.price <= PC.price;
```

model	price
2002	949
2003	599
2007	249
2008	949
2009	599
2011	499
2012	499

RENAMING

- Sometimes it is useful to list the same relation more than once (creating a product with itself)
- In those cases the tables need to be **renamed** for us to use the dot notation
- This is done by the expression `<relation>` **`AS`** `<newName>`
- Or simply by leaving out **`AS`**: `<relation>` `<newName>`
  - But this can sometimes be less readable

```
SELECT <Attributes>
       FROM <Relation1> AS <newName1>,
            <Relation2> AS <newName2>,
            ...
            <RelationN> AS <newNameN>,
       WHERE <Conditions>;
```

RENAMING EXAMPLE

PC(<u>model</u>,speed,ram,hd,price)

(Get a model-by-model comparison of PC's costing the same.)

```
SELECT *
       FROM PC AS P1,
            PC AS P2
       WHERE P1.price = P2.price
       AND P1.model < P2.model ;
```

model	speed	ram	hd	price	model	speed	ram	hd	price
1001	2.7	8192	1000	999	1011	3.3	8192	700	999
1003	3.0	4096	500	499	1005	1.4	4096	500	499
1003	3.0	4096	500	499	1008	3.2	8192	1000	499
1004	1.6	8192	1000	1249	1009	2.8	8192	1000	1249
1005	1.4	4096	500	499	1008	3.2	8192	1000	499

CONTROLLING OUTPUT ORDER - ORDER BY

- By default ordering of the resulting rows is arbitrary
- Output can be sorted by adding **ORDER BY** and a list of columns to the **SELECT** statement
- By default ordering is ascending
  - Can be changed by appending **DESC** after the attributes which should be ordered descending

```
SELECT <Attributes>
       FROM <Relations>
       WHERE <Conditions>
       ORDER BY <Attributes>;
```

ORDER BY EXAMPLE

Courses(code,name,credits)

```
SELECT name, credits
       FROM Courses
       WHERE credits > 3
       ORDER BY credits,
                name;
```

name	credits
Sustainable Product and Service Design	4.0
Basic Course in Programming Y1	5.0
Databases	5.0
Databases	5.0
Programming 1	5.0
Programming 1	5.0
Programming 2	5.0
Programming 2	5.0
Programming Studio 1	5.0
Business and Industrial Policy	6.0

```
SELECT name, credits
       FROM Courses
       WHERE credits > 3
       ORDER BY credits DESC,
                name;
```

name	credits
Business and Industrial Policy	6.0
Basic Course in Programming Y1	5.0
Databases	5.0
Databases	5.0
Programming 1	5.0
Programming 1	5.0
Programming 2	5.0
Programming 2	5.0
Programming Studio 1	5.0
Sustainable Product and Service Design	4.0

CALCULATIONS AND RENAMES IN THE RESULT

- It is possible to transform the resulting values of a query
  - Done by inserting expressions in the attribute list after the **SELECT** statement
- It is also possible to rename the columns of the output
  - Done by using an **AS** statement after the attribute

EXAMPLE - CHANGING THE RESULT

```
SELECT title,length/60.0 AS hours
       FROM Movies WHERE length > 120;
```

title	hours
Gone with the Wind	3.85
Star Wars	2.06666666666667
Mary Poppins	2.31666666666667
Prince Caspian	2.31666666666667
The Godfather, Part III	2.83333333333333
Red Dragon	2.06666666666667
The Colour Purple	2.55
Tuntematon sotilas	2.81666666666667
Sabrina	2.11666666666667
Sabrina	2.11666666666667
Tuntematon sotilas	3.28333333333333
Letters from Iwo Jima	2.35
Notting Hill	2.06666666666667
Gandhi	3.13333333333333
Jurassic Park	2.11666666666667
Schindler's List	3.28333333333333
Million Dollar Baby	2.2
Apocalypse Now	2.55

SET OPERATIONS

- The result from two or more **SELECT** statements can be combined column by column by set operations

```
SELECT ...
<operation>
SELECT ...
;
```

- Where `<operation>` is one of
  - **UNION**
    - The union of the two tables
  - **INTERSECT**
    - Only those results which occurs in both tables.
  - **EXCEPT**
    - Results occurring in the first result, but **not** in the second
- **Note**
  - Tables must have the same number of columns
    - And the columns should have the same type
  - Only column order and type matters

EXAMPLE - UNION

(Get a list of all PC models with clock speed >= 3.2 GHz, and laptops with more than 1TB of hard drive space.)

```
SELECT model FROM PC
       WHERE speed >= 3.2
UNION
SELECT model FROM Laptop
       WHERE hd >= 1000
;
```

model
1006
1007
1008
1010
1011
2005

EXAMPLE - INTERSECT

(Get speed/ram configurations available in both PC and `Laptop` computers.)

```
SELECT speed,ram FROM PC
INTERSECT
SELECT speed,ram FROM Laptop
;
```

speed	ram
2.7	8192

EXAMPLE - EXCEPT

(Get the `Laptop`'s with best clock speed or RAM.)

```
SELECT model,speed,ram,price FROM Laptop
EXCEPT
SELECT L1.model,L1.speed,L1.ram,L1.price
       FROM Laptop AS L1, Laptop AS L2
       WHERE L1.speed < L2.speed
       AND L1.ram < L2.ram
       AND L1.model <> L2.model
;
```

model	speed	ram	price
2004	2.7	8192	1649
2005	2.6	16384	2499
2006	2.2	16384	2199