Databases

Structured Query Language (SQL) I

Dr Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30
See Duo for the Zoom link

Database languages

- A good database language should allow users to:
  - create the database, define relation structures
  - perform basic data management:
    insert, modify, delete data from relations
  - perform simple and complex queries
- All these tasks with minimal user effort!
  - syntax / command structure must be easy to learn
  - users should concentrate on which queries to make
    (not how they are implemented)
- The language should be portable:
  - conform to a recognized standard
  - we can use the same language with many DBMS's

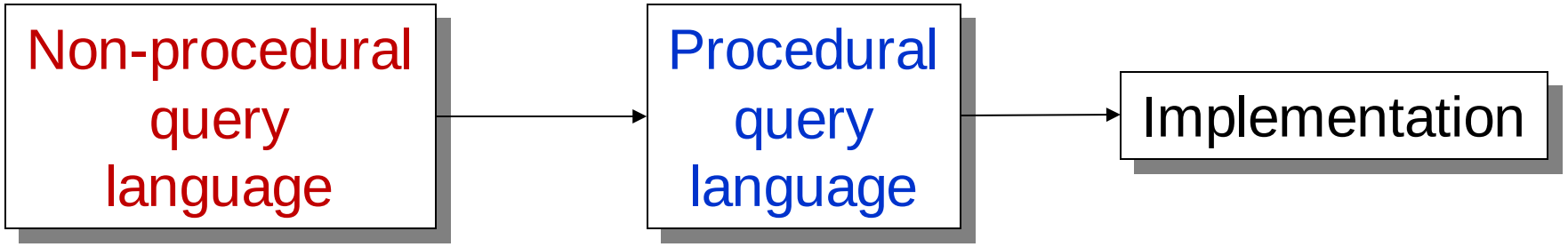
Database languages

- SQL (Structured Query Language)
  - the most common database language
  - simple syntax / easy to learn and use
  - it has two components: DDL & DML
- Data Definition Language (DDL)
  - allows users to define the database
  - define the schema for each relation (attributes / types)
  - define the domain of each attribute
  - specify integrity constraints
- Data Manipulation Language (DML)
  - allows users to insert / update / delete / retrieve data from the DB
  - Query Language: the part of DML that involves data retrieval

Two types of query languages

- SQL: formal *definition* of a new relation
  from existing relations in the DB
- Relational algebra: specifies *how* to build a new relation
  from existing relations in the DB

- Their place in the big picture:



SQL,
relational calculus

relational algebra

specifies which data are to be retrieved

specifies how to retrieve the required data

Writing SQL statements

- SQL statements consist of:
  - reserved words: a fixed part of SQL
    - must be spelt exactly as required
    - cannot be split across lines
  - user-defined words: made up by user
    - represent names of relations, attributes, cell entries, etc.
- SQL statements:
  - case insensitive (i.e. both upper / lower case)
  - except for literal character data (i.e. data entries)
    e.g. a surname ‘SMITH’ is different than ‘Smith’
- SQL is free-format:
  - parts of statements do not have to be written
    in specific locations on the screen (i.e. arbitrary indentation)

Writing SQL statements

- However:
  - more readable with systematic indentation and lineation
  - each clause should begin on a new line
  - start of a clause should line up with start of other clauses
  - if a clause has several parts, they should each appear on separate lines and be indented under the start of clause
- Mostly standard (intuitive) English words:
  - CREATE TABLE Staff (staffNo VARCHAR(5),
    lName VARCHAR(15),
    salary DECIMAL(7,2));
  - INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
  - SELECT staffNo, lName, salary
    FROM Staff
    WHERE salary > 10000;

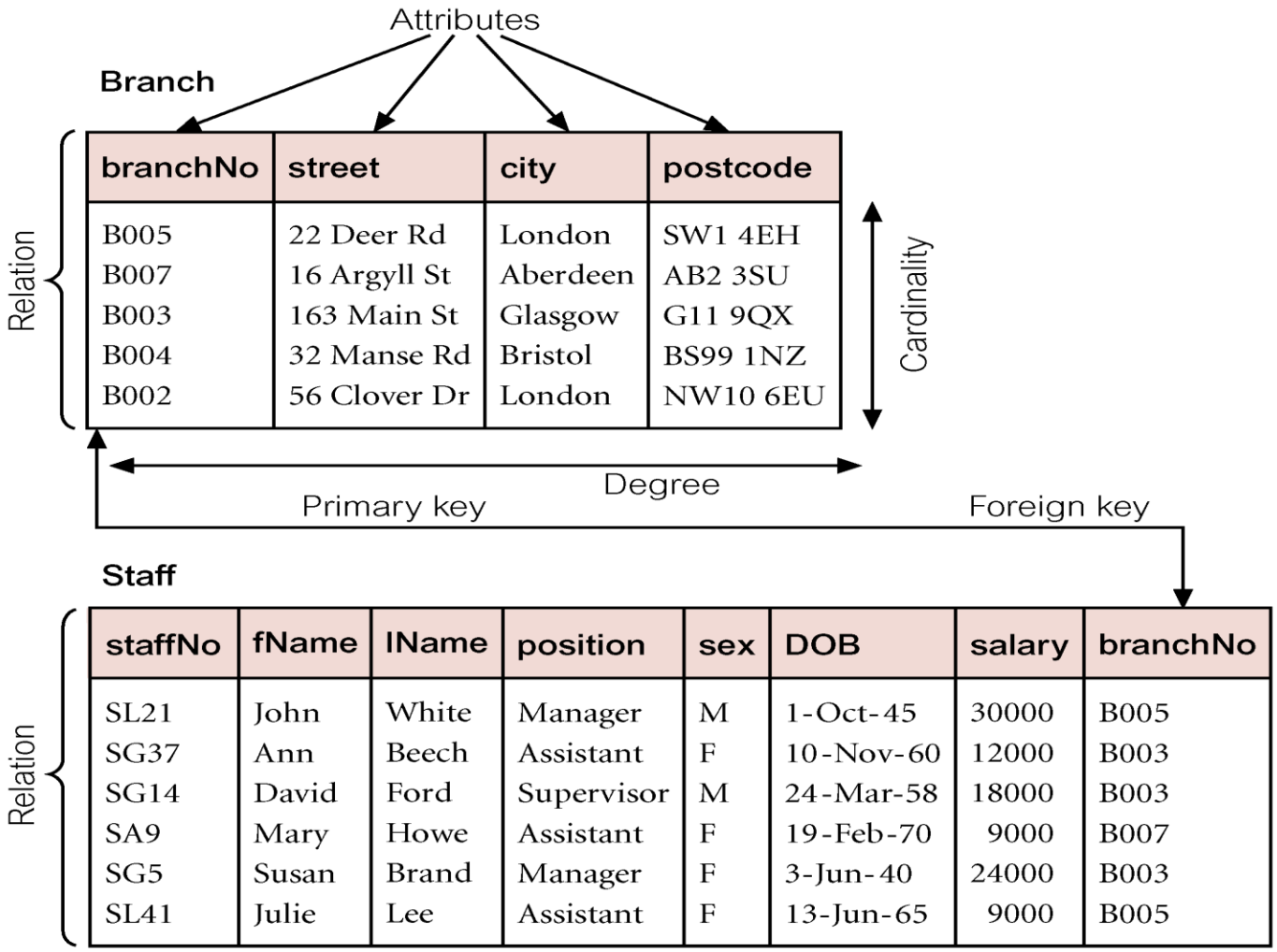
Data Manipulation Language (DML)

- We mainly look at DML aspects of SQL
- To create a database in MySQL:
  - either use DDL commands
  - or just use the interactive tools of *phpMyAdmin*
- Main statements of interest in DML:
  - SELECT – to query data in the database
  - INSERT – to insert new data into an existing table
  - UPDATE – to update data in an existing table
  - DELETE – to delete data from an existing table

Writing SQL commands

- Literals (character data / numericals):
  - are constants used in SQL statements
- All non-numeric literals:
  - must be enclosed in single quotes (e.g. ‘London’)
- All numeric literals:
  - must not be enclosed in quotes (e.g. 650.00)
- Notation:
  - UPPER-case letters represent reserved words
  - lower-case letters represent user-defined words
  - a vertical bar (|) indicates a choice among alternatives
  - curly braces {a} indicate a required element
  - square brackets [a] indicate an optional element
  - ellipsis (…) indicates optional repetition (0 or more)

Tables / Relations used in example



Simple Queries

- The sequence of processing in a SELECT-FROM-WHERE statement is:
  - SELECT: specifies which columns are to appear in the output
  - FROM: specifies the table or tables to be used
  - WHERE: filters the rows subject to some condition
    - GROUP BY: forms groups of rows with the same column value
    - HAVING: filters the groups subject to some condition
    - ORDER BY: specifies the order of the output

SELECT and FROM are mandatory

Syntax

SELECT [ALL | DISTINCT] *column1* [,*column2, column3, …*]
FROM *table1* [,*table2, table3, …*]
[WHERE "conditions"]
[GROUP BY "column-list"]
[HAVING "conditions]
[ORDER BY "column-list" [ASC | DESC]] ;

Example: **SELECT** staffNo, fName, lName, position, sex, DOB, salary, branchNo
FROM staff ;

- The above statement will select the (whole) specified columns from the staff table

NOTE: If you want to place more queries at once, remember to put a semicolon (;) at the end of each SQL statement.
The ; indicates that your SQL statement has finished and the next one can start.

SELECT Statement, Example 1

- **List full details of all staff (All columns, all rows)**

```
SELECT staffNo, fName, lName,
       position, sex, DOB, salary, branchNo
FROM Staff
```

- **Alternative: use * as an abbreviation for 'all columns':**

```
SELECT *
FROM Staff
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

SELECT Statement, Example 2

Produce a list of salaries for all staff, showing only:
staff number, first name, last name, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff
```

This command creates a new table (from the table Staff), containing the designated columns (in the specified order)

The rows are NOT ordered

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

SELECT Statement, Example 3

Produce a list of monthly salaries for all staff, showing only staff number, first name, last name, and monthly salary

```
SELECT staffNo, fName, lName, salary/12
FROM Staff
```

Derived field

staffNo	fName	lName	Month Salary
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

We can leave the column name blank or use an "AS" clause:

```
SELECT staffNo, fName, lName, salary/12 AS 'Month Salary'
FROM Staff
```

SELECT & FROM clause review

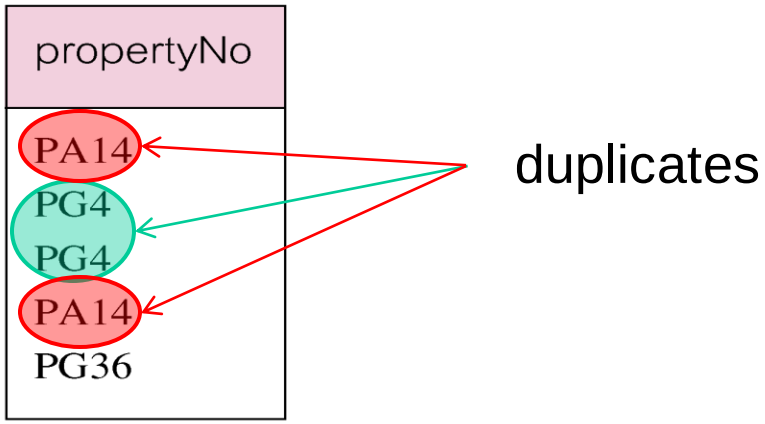
```
SELECT   first_column_name, second_column_name
FROM   table_name
WHERE   first_column_name > 12000
```

- Next to the SELECT keyword:
  - the column name(s) specify which columns will be returned
  - as many columns as we like
  - or * to return all columns
- Next to the FROM keyword:
  - the table name(s) specifies the table that will be queried to retrieve the results
- Next to the (optional) WHERE keyword:
  - the condition(s) specifies which rows will be returned (filtering of the rows)

DISTINCT

- In normalized relational tables:
  - no duplicated rows
- But the result of SELECT:
  - may have duplicate rows

SELECT propertyNo
FROM Viewing



- **Use DISTINCT to eliminate duplicates:**

SELECT DISTINCT propertyNo
FROM Viewing

propertyNo
PA14
PG4
PG36

SELECT Statement, Example 4 (DISTINCT)

List all branch numbers for all branches.

```
SELECT branchNo
FROM Staff
```

branchNo
B005
B003
B003
B007
B003
B005

With use of DISTINCT:

```
SELECT DISTINCT branchNo
FROM Staff
```

branchNo
B005
B003
B007

SELECT statement (WHERE)

- We often need to restrict the rows that are retrieved
- The WHERE clause is followed by search conditions (predicates):
  - Comparisons
    - compare values of two expressions
  - Range
    - BETWEEN / NOT BETWEEN
    - tests whether the value falls within a specified range
  - Set membership
    - IN / NOT IN
  - Pattern matching
    - LIKE / NOT LIKE

Comparison conditions

Comparison operators:

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to
LIKE	String comparison (for pattern matching)

List all staff with a salary greater than 10000

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Range conditions

List all the salaries and staff numbers of all staff with salary at least 20000 and at most 30000

```
SELECT staffNo, salary
FROM staff
WHERE salary BETWEEN 20000 AND 30000
```

- Range condition (BETWEEN) does not add much to SQL's expressive power:

- The above query is equivalent to:

```
SELECT staffNo, salary
FROM staff
WHERE salary >= 20000 AND Salary <= 30000
```

- NOTE: this would include those staff who have salary exactly 20000 or exactly 30000

Set membership conditions

List all managers and supervisors

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor')
```

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

- "IN" does not add much to SQL's expressive power
- The above query is equivalent to:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position='Manager' OR position= 'Supervisor'
```

Pattern matching (LIKE)

- Sometimes we want to search within a string:

  "Find all owners with the string 'Glasgow' in their addresses"

- SQL has two special pattern-matching symbols:
  - % (percent) represents an arbitrary sequence of zero or more characters (called wildcard)
  - _ (underscore) represents an arbitrary single character
- LIKE 'H%' means:
  - first character must be H, but the rest can be anything
- LIKE 'H_ _ _' means:
  - exactly 4 characters, first character must be H
- LIKE '%e' means:
  - any sequence of characters, ending with 'e'
- NOT LIKE 'H%' means:
  - the first character cannot be 'H'

Pattern matching (LIKE)

Find all owners with the string 'Glasgow' in their address

```
SELECT   ownerNo, fName, lName, address, telNo
FROM   PrivateOwner
WHERE   address  LIKE  '%Glasgow%'
```

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Combining conditions and Boolean Operators

- The logical AND operator:
  - both sides of the condition must be true
- The logical OR operator:
  - at least one of the two sides of the condition must be true
- They can be used to join two (or more) conditions in the WHERE clause:

```
SELECT  fName, IName, position, salary
FROM  staff
WHERE  postition = ‘Manager' OR  position = ‘Supervisor'

SELECT  fName, lName, position, salary
FROM  staff
WHERE  salary >= 24000  AND  position = ‘Manager‘
```

- These two operators can also be used combined

Combining conditions and Boolean Operators

List addresses of all branch offices in London or Glasgow

```
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow'
```

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Examples of using WHERE

- SELECT staffNo
  FROM staff
  WHERE salary <= 40000;

Comparison Condition

- SELECT *
  FROM branch
  WHERE city = 'London' OR city = 'Glasgow'

Disjunction (logical OR)

- SELECT staffNo, salary
  FROM staff
  WHERE city = 'London' AND salary > 25000

Conjunction (logical AND)

- SELECT staffNo, salary
  FROM staff
  WHERE salary BETWEEN 20000 AND 30000

Range Condition

Example: NULL Search Condition

- List details of all viewings on property PG4 where no comment has been supplied
- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using the special keyword IS NULL:

```
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo='PG4' AND comment IS NULL
```

ORDER BY clause

- In the resulting table of a SELECT query:
  - the rows are NOT ordered
- ORDER BY can be used to sort the rows
  - according to the values of a particular set of columns
  - can be ascending / descending
  - ordering appears regardless of whether that column appears in the result
- General format:

```
SELECT  column1,
FROM  "list-of-tables"
ORDER BY  "column-list" [ASC | DESC]
```

[…] means “optional”

ORDER BY clause

List the salaries of all staff, arranged in descending order of salary *(i.e. large salaries last)*

```
SELECT  staffNo, fName, lName, salary
FROM  Staff
ORDER BY  salary  DESC
```

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

decreasing salaries

ORDER BY clause

- We can also sort according to multiple columns:
  - first sort according to the first column
  - among rows with the same value in the first column, sort according to the second column, etc.

List the properties arranged in (ascending) order of property type

i.e. ascending

```
SELECT *
FROM PropertyForRent
ORDER BY type
```

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

```
SELECT *
FROM PropertyForRent
ORDER BY type, rent DESC
```

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

Aggregate Functions

- Aggregate functions:
  - operate on a single column
  - return a single (numeric) value

numeric data	SUM	returns the sum of the numeric values in a given column
	AVG	returns the average value of a given column
any data	MIN	returns the smallest value in a given column
	MAX	returns the largest value in a given column
	COUNT	returns the total number of values in a given column
	COUNT(*)	returns the number of rows in a table

```
SELECT  AVG(salary)
FROM  staff
```

```
SELECT  SUM(salary)
FROM  staff;
WHERE  position = 'Manager'
```

```
SELECT  COUNT(*)
FROM  staff
```

Aggregate Functions

How many properties cost more than £350 per month to rent?

```
SELECT COUNT(*) AS myCount
FROM PropertyForRent
WHERE rent > 350
```

myCount
5

How many different properties have been viewed in May '04 ?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-04' AND '31-May-04'
```

Avoid duplications: the same property can be viewed many times !

myCount
2

Aggregate Functions

Find number of Managers and sum of their salaries

```
SELECT  COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM  Staff
WHERE  position = 'Manager'
```

myCount	mySum
2	54000.00

Find minimum, maximum, and average staff salary

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
FROM Staff
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

GROUP BY clause

- Aggregate functions:
  - similar to the "totals" at the bottom of a report
- Often we also need "subtotals" in reports
  - at the bottom of some part of the report (e.g. at the end of every day)
- GROUP BY can be used to:
  - partition the data into groups
  - produce a single summary row (e.g. "subtotal") for each group

---

Find the number of staff working in each branch and the sum of their salaries

one sum for every group (i.e. branchNo)

```
SELECT  branchNo,
        COUNT(staffNo) AS myCount,
        SUM(Salary) AS mySum
FROM staff
GROUP BY branchNo
ORDER BY branchNo
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Summary of Writing SQL commands

- The beginning of each clause should line up with the beginning of other clauses
- If a clause has several parts, each should be on a separate line and indented
- Upper case letters for reserved words e.g. SELECT
- Lower case to represent user-defined words e.g. students
- A vertical bar | indicates a choice e.g. a | b | c;
- Curly braces indicated a required element e.g. {a};
- Square brackets indicate an optional element e.g [a];
- An ellipsis […] indicates optional repetition
  - e.g. {a | b} [,c …] means:
    either a or b followed by zero or more repetitions of c,
    separated by comma