Data Storage and Retrieval
Lecture 8
Structured Query Language (SQL)

Dr. Graham McDonald

Graham.McDonald@glasgow.ac.uk

University of Glasgow

Relational Algebra (RA)

Three basic types of Relational Algebra operators:

1. **Applying to one relation**

    projection ∏ , selection σ *(conditions)*

2. **Applying to two relations of identical structure**

    union ∪, intersection ∩, difference - *(no conditions)*

3. **Applying to two relations of different structure**

    product × *(no conditions)*

    joins ⋈ *(conditions)*

- RA is used directly in NoSQL
- RA is used internally by SQL

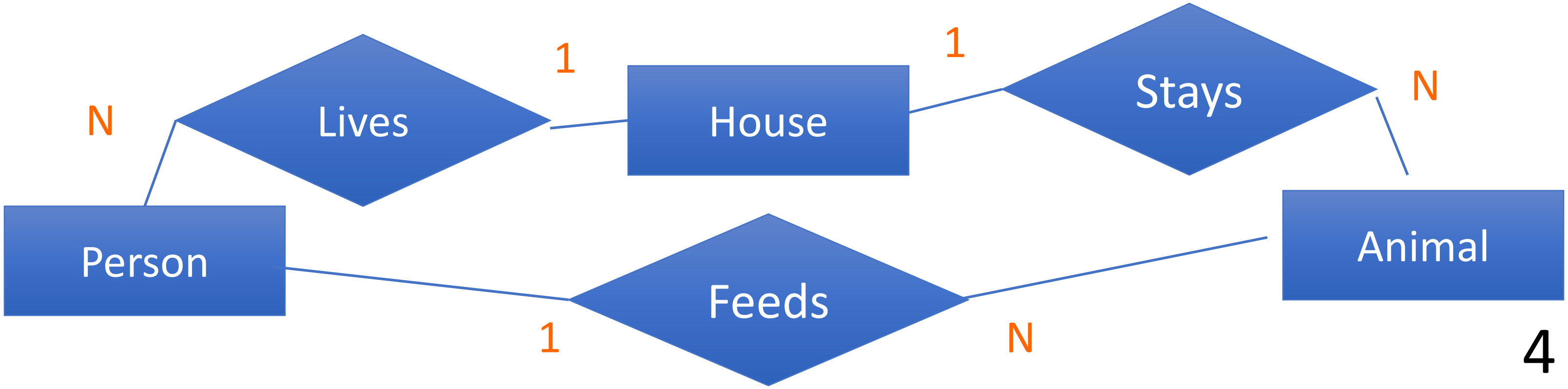
SQL – "seequel"

A database language, which allows the user to:

- create database and relation structures
- perform management tasks (insert, modify, delete)
- perform simple and complex queries

- SQL is non-procedural, you specify:
  - WHAT information you need
  - *Not* HOW to get it
- "Intergalactic dataspeak"- an ISO standard
- Various DBMS products (e.g.: MS Access, even MySQL) do not wholly conform to the ISO SQL standards

The January Census



N

Lives

1

House

1

Stays

N

Person

Animal

1

Feeds

N

The January Census

House

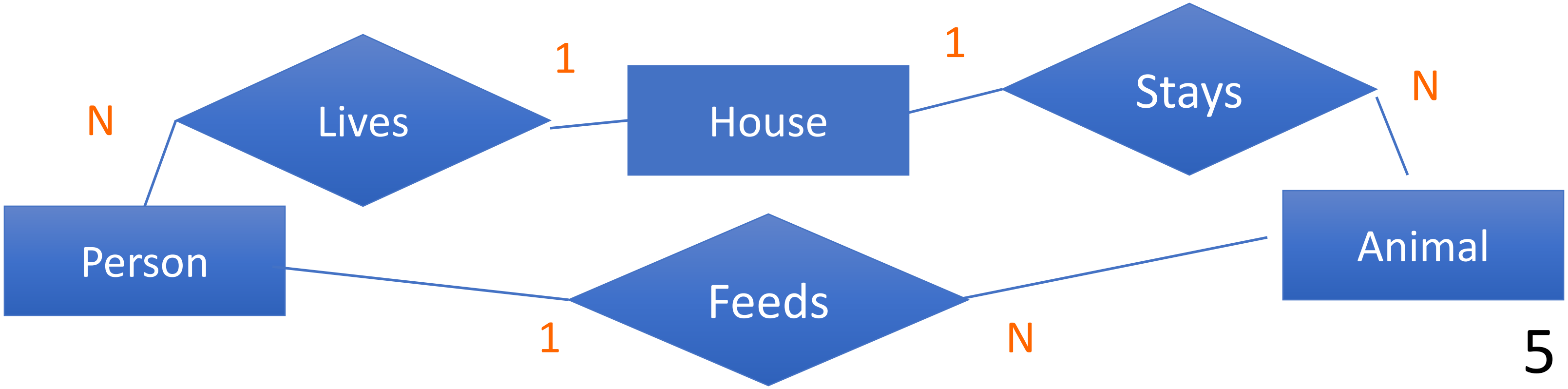
houseNum
34
38
42
48

Person

name	age	houseNum
Jim	15	34
Jo	23	38
Pete	20	38
Jenny	10	42
James	15	48
Paul	15	

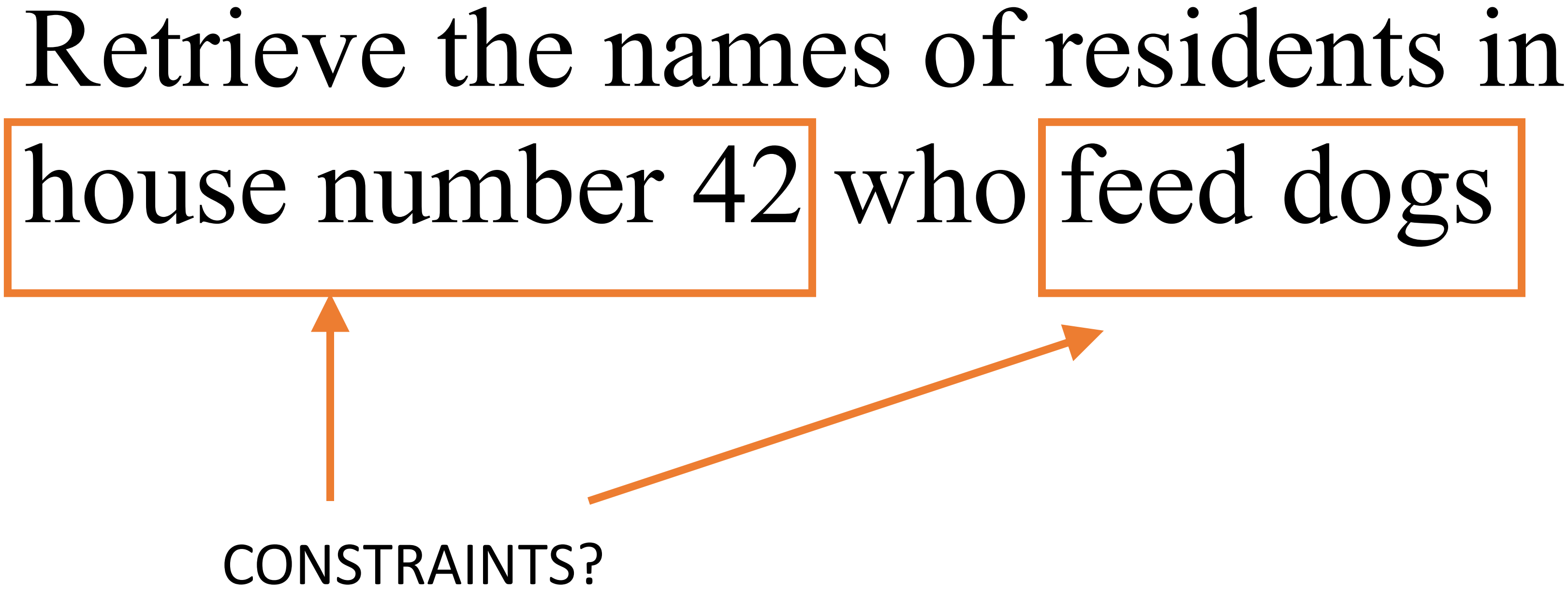
Animal

aname	type	houseNum	fedBy
Fluffy	dog	34	Jim
Splodge	cat	34	Jim
Tinky	dog	38	
Robin	dog	42	Jenny
Red	dog	42	Jim
Dusty	snake		Jim



WHAT?

Retrieve the names of residents in house number 42 who feed dogs



Retrieve the names of residents in house number 42 who feed dogs

CONSTRAINTS?

SELECT

- ***SELECT*** is the SQL command used to ***retrieve*** data
    - It takes a set of relations (tables) and describes the constraints that must be placed on them to return a given set of rows

Basic SQL Query Syntax

```
SELECT [DISTINCT] target-list
    FROM   relation-list
        WHERE  qualification
```

- *relation-list* is a list of relation names (possibly with "aliases")
- *target-list* is a list of attributes of relations in *relation-list*
- *qualification* is a Boolean expression

Basic SQL Query Syntax

```
SELECT [DISTINCT] target-list
    FROM   relation-list
        WHERE  qualification
```

WHERE is equivalent to relational **selection** operator

Basic SQL Query Syntax

```
SELECT [DISTINCT] target-list
    FROM   relation-list
        WHERE  qualification
```

FROM forms a **Cartesian product** of the tables/relations

Basic SQL Query Syntax

```
SELECT [DISTINCT] target-list
    FROM   relation-list
        WHERE   qualification
```

SELECT makes a **projection** of certain attributes

GOTCHA: Relational Algebra vs. SQL

Project ∏ => SELECT

Select σ => WHERE

Projection $\prod$

Retrieving required **columns** from a relation

SELECT name
FROM Person;

$\prod_{\text{name}}$ [Person]

name
Jim
Jo
Pete
Jenny
James
Paul

SELECT name, age
FROM Person;

$\prod_{\text{name, age}}$ [Person]

name	age
Jim	15
Jo	23
Pete	20
Jenny	10
James	15
Paul	15

Projection Over All Columns

SELECT *

FROM Person

WHERE age = 15;

name	age	houseNum
Jim	15	34
James	15	48
Paul	15	

$\sigma_{(age=15)}$ [Person]

Selection σ

Retrieving required **rows** from a relation

```
SELECT name, houseNum
FROM Person
WHERE age = 15;
```

name	houseNum
Jim	34
James	48
Paul	

$$\Pi_{\text{name, houseNum}} (\sigma_{(\text{age}=15)} [\text{Person}])$$

```
SELECT type, fedBy
FROM Animal
WHERE fedBy = ‘Jim’;
```

type	fedBy
dog	Jim
cat	Jim
dog	Jim
snake	Jim

$$\Pi_{\text{type, fedBy}} (\sigma_{(\text{fedBy=“Jim”})} [\text{Animal}])$$

Unlike RA, there are duplicate rows!

Selecting Unique Rows

```
SELECT type, fedBy
FROM Animal
WHERE fedBy = ‘Jim’;
```

type	fedBy
dog	Jim
cat	Jim
dog	Jim
snake	Jim

```
SELECT DISTINCT type, fedBy
FROM Animal
WHERE fedBy = ‘Jim’;
```

type	fedBy
dog	Jim
cat	Jim
snake	Jim

$$\Pi_{\text{type, fedBy}} (\sigma_{(\text{fedBy=‘Jim’})} \text{ Animal})$$

Unlike RA, SQL does **not** automatically remove duplicates. We need to request this explicitly

Other Operators from Relational Algebra

- The SQL standard has equivalents to the RA operators ∪, ∩ and -
  - UNION
  - INTERSECT
  - MINUS/EXCEPT
    - (In fact, MySQL only implements UNION)

- The Cartesian product X and *joins* are very important

Set Union U

Remember
Union Compatibility:
Same number of columns,
with matching datatypes.

```
(SELECT name FROM Person WHERE age=15)
        UNION
(SELECT name FROM Person WHERE age=20);
```

name
Jim
Pete
James
Paul

$$\text{AllPeople} := \Pi_{\text{name, age}} [\text{Person}] \cup \Pi_{\text{name,age}} [\text{Person}]$$

$$\text{People15-20} := \Pi_{\text{name}} (\sigma_{(\text{age=15 or age=20})} [\text{AllPeople}])$$

Cartesian Product ×

person × animal

Combination of all rows and columns

Dot operator

```
SELECT name, age, person.houseNum,
        aname, type, animal.houseNum, fedBy
FROM Person, Animal;


SELECT *
FROM Person, Animal;
```

Cartesian product of all relations in the FROM

The January Census

Person

name	age	houseNum
Jim	15	34
Jo	23	38
Pete	20	38
Jenny	10	42
James	15	48
Paul	15	

Animal

aname	type	houseNum	fedBy
Fluffy	dog	34	Jim
Splodge	cat	34	Jim
Tinky	dog	38	
Robin	dog	42	Jenny
Red	dog	42	Jim
Dusty	snake		Jim

SELECT * FROM Person, Animal

name	age	Person.houseNum	aname	type	Animal.houseNum	fedBy
Jim	15	34	Fluffy	dog	34	Jim
Jim	15	34	Splodge	cat	34	Jim
Jim	15	34	Tinky	dog	38	
Jim	15	34	Robin	dog	42	Jenny
Jim	15	34	Red	dog	42	Jim
Jim	15	34	Dusty	snake		Jim
Jo	23	38	Fluffy	dog	34	Jim
Jo	23	38	Splodge	cat	34	Jim
Jo	23	38	Tinky	dog	38	
Jo	23	38	Robin	dog	42	Jenny
Jo	23	38	Red	dog	42	Jim
Jo	23	38	Dusty	snake		Jim
Pete	20	38	Fluffy	dog	34	Jim
Pete	20	38	Splodge	cat	34	Jim
Pete	20	38	Tinky	dog	38	
Pete	20	38	Robin	dog	42	Jenny
Pete	20	38	Red	dog	42	Jim
Pete	20	38	Dusty	snake		Jim
Jenny	10	42	Fluffy	dog	34	Jim
Jenny	10	42	Splodge	cat	34	Jim
Jenny	10	42	Tinky	dog	38	
Jenny	10	42	Robin	dog	42	Jenny
Jenny	10	42	Red	dog	42	Jim
Jenny	10	42	Dusty	snake		Jim
James	15	48	Fluffy	dog	34	Jim
James	15	48	Splodge	cat	34	Jim
James	15	48	Tinky	dog	38	
James	15	48	Robin	dog	42	Jenny
James	15	48	Red	dog	42	Jim
James	15	48	Dusty	snake		Jim
Paul	15		Fluffy	dog	34	Jim
Paul	15		Splodge	cat	34	Jim
Paul	15		Tinky	dog	38	
Paul	15		Robin	dog	42	Jenny
Paul	15		Red	dog	42	Jim
Paul	15		Dusty	snake		Jim

Recall: Cartesian Product

The **Cartesian Product** of two relations A and B, (with attributes A_1 … A_m and B_1 … B_n),

is a relation with **m + n** attributes containing

a row for every pair of rows

(one from A and one from B)

Thus if A has *a* tuples and B has *b* tuples then the result has *a* * *b* tuples, with *m*+*n* attributes

Equi-join - $\bowtie_{A=B}$

An Equi-join is the cartesian product, followed by a selection

```
SELECT Person.name
FROM Person, Animal
WHERE Person.houseNum = Animal.houseNum;
```

Dot operator

$$\Pi_{\text{person.name}}\ (\sigma_{(\text{person.houseNum=animal.HouseNum})}(\text{Person} \times \text{Animal}))$$

name	age	Person. houseNum	aname	type	Animal. houseNum	fedBy
Jim	15	34	Fluffy	dog	34	Jim
Jim	15	34	Splodge	cat	34	Jim
Jo	23	38	Tinky	dog	38	
Pete	20	38	Tinky	dog	38	
Jenny	10	42	Robin	dog	42	Jenny
Jenny	10	42	Red	dog	42	Jim

```
SELECT *
FROM Person, Animal
WHERE Person.houseNum = Animal.houseNum;
```

$$\sigma_{(\text{person.houseNum}=\text{animal.HouseNum})}(\text{Person} \times \text{Animal})$$

Natural Join - ⋈

product + *condition that* all attributes of the **same name** be equated, then only one column for each pair of equated attributes is projected out

```
SELECT *
FROM Person NATURAL JOIN Animal;
```

$$\text{Person} \bowtie \text{Animal}$$

name	age	Person.houseNum	aname	type	fedBy
Jim	15	34	Fluffy	dog	Jim
Jim	15	34	Splodge	cat	Jim
Jo	23	38	Tinky	dog	
Pete	20	38	Tinky	dog	
Jenny	10	42	Robin	dog	Jenny
Jenny	10	42	Red	dog	Jim

```
SELECT *
FROM Person NATURAL JOIN Animal;
```

```
SELECT name, age, Person.houseNum, aname, type, fedBy
FROM Person, Animal
WHERE Person.houseNum = Animal.houseNum;
```

$$\text{person} \bowtie \text{animal}$$

Renaming

The reserved word AS can be used to define **aliases** for attributes or relations:

Relations:

```
SELECT p.name, age, p.houseNum,
          a.aname, type, a.houseNum, fedBy
FROM    Person AS p, Animal AS a
WHERE p.houseNum = a.houseNum;
```

Attributes:

```
SELECT p.name AS pn, age, p.houseNum AS ph
          a.aname AS an, type, a.houseNum AS ah, fedBy
FROM    Person AS p, Animal AS a
WHERE ph = ah;
```

University of Glasgow

Using the Same Table Twice

Renaming is needed when you have to use the same table twice in the same query

```
SELECT E.name, S.name
  FROM Employee AS E, Employee AS S
       WHERE (E.supervisor = S.NI#)
```

Useful for querying across recursive relationships

Employee

NI#	name	DoB	DNo	Supervisor
1001	J Smith	23/2/54	14	null
1002	J Jones	24/5/73	11	1001
1003	J Brown	24/7/80	14	1001
1004	J Smith	24/6/76	14	1002

→

E.name	S.name
J Jones	J Smith
J Brown	J Smith
J Smith	J Jones

Using the Same Table Twice

E			S	
NI#	name	Sup'r	S.NI	name
1001	J Smith	null	1001	J Smith
1001	J Smith	null	1002	J Jones
1001	J Smith	null	1003	J Brown
1001	J Smith	null	1004	J Smith
1002	J Jones	1001	1001	J Smith
1002	J Jones	1001	1002	J Jones
1002	J Jones	1001	1003	J Brown
1002	J Jones	1001	1004	J Smith
1003	J Brown	1001	1001	J Smith
1003	J Brown	1001	1002	J Jones
1003	J Brown	1001	1003	J Brown
1003	J Brown	1001	1004	J Smith
1004	J Smith	1002	1001	J Smith
1004	J Smith	1002	1002	J Jones
1004	J Smith	1002	1003	J Brown
1004	J Smith	1002	1004	J Smith

E x S

Using the Same Table Twice

E			S	
NI#	name	Sup'r	S.NI	name
1001	J Smith	null	1001	J Smith
1001	J Smith	null	1002	J Jones
1001	J Smith	null	1003	J Brown
1001	J Smith	null	1004	J Smith
1002	**J Jones**	**1001**	**1001**	**J Smith**
1002	J Jones	1001	1002	J Jones
1002	J Jones	1001	1003	J Brown
1002	J Jones	1001	1004	J Smith
1003	**J Brown**	**1001**	**1001**	**J Smith**
1003	J Brown	1001	1002	J Jones
1003	J Brown	1001	1003	J Brown
1003	J Brown	1001	1004	J Smith
1004	J Smith	1002	1001	J Smith
1004	**J Smith**	**1002**	**1002**	**J Jones**
1004	J Smith	1002	1003	J Brown
1004	J Smith	1002	1004	J Smith

E x S

Basic SQL - reminder

```
SELECT target attribute(s)
FROM relation(s)
WHERE qualification condition(s)
```

ORDER MATTERS. It is ***NOT***

```
FROM relation(s)
WHERE qualification
SELECT attributes
```

🚫

- We have taken care of the following core relational algebra operations in SQL:
  - Projection (columns)
  - Selection (rows)
  - Intersection
  - Difference
  - Union
  - Cartesian products and joins
  - Renaming

Mapping to Relational Algebra

- The simple query

```
SELECT a1, …, an
FROM   R1, …, Rm
WHERE  b;
```

- Can be mapped to the RA expression:

$$\pi_{a_1,\ldots,a_n}(\sigma_b(R_1 \times \ldots \times R_m))$$

(Assuming no duplicates)

Semantics of SQL Queries

- The semantics of SQL query evaluation can be conceptually defined as:
  - Compute the **product** of *relation-list*
  - Discard those tuples that fail the *qualification*
  - Delete attributes that are not in *target-list*
  - If DISTINCT is specified, eliminate duplicate rows

WHERE Options

SQL provides powerful string matching facilities
% denotes zero or more characters
_ denotes any one character

WHERE age > 15

WHERE age < > 18

WHERE age = 15 AND houseNum **IS NOT NULL**

WHERE houseNum **IS NULL**

WHERE age BETWEEN 10 AND 20

WHERE (type = ‘dog’) OR (type = ‘cat’)

Combine using AND, OR; use brackets to denote precedence

WHERE name

LIKE ‘J%’ (names beginning with J)

LIKE ‘J_ _ _ _’ (5 letter names beginning with J)

LIKE ‘_ _ M %’ (names with M as the third letter)

University of Glasgow

Operator Precedence

name	age	houseNum
Jim	15	34
Jo	23	38
Pete	20	38
Jenny	10	42
James	15	48
Paul	15	

```
SELECT name
FROM Person
WHERE age = 23
AND houseNum = 38
OR houseNum = 48
```

(a)

```
SELECT name
FROM Person
WHERE age = 23
AND (houseNum = 38
OR houseNum = 48)
```

name
Jo

(b)

```
SELECT name
FROM Person
WHERE (age = 23
AND houseNum = 38)
OR houseNum = 48
```

name
James
Jo

Operator Precedence (2)

- Different operators (+ - * OR AND) have higher *precedence*
  - *This means they are evaluated before others*
  - *E.g. * before +, so 5*4 + 2 = 22, not 30*
  - *E.g. AND before OR*

```
WHERE (age = 23
AND houseNum = 38)
OR houseNum = 48
```

- If you aren’t sure about which operator has precedence, then use brackets
  - Even if you are, other people reading your SQL might not be!

What is the connection between UNION, INTERSECT & AND, OR?

```
(SELECT name
FROM Person
WHERE houseNum = 34)
UNION
(SELECT name
FROM Person
WHERE houseNum = 38)
```

name
Jim
Jo
Pete

```
SELECT name
FROM Person
WHERE houseNum = 34
OR houseNum = 38
```

name
Jim
Jo
Pete

Calculated Attributes: Extended Projection

Specify a calculation to be performed on the values of an attribute:

```
SELECT name, age*age,
houseNum-2
FROM Person;
```

name	val1	val2
Jim	225	32
Jo	529	36
Pete	400	36
Jenny	100	40
James	225	48
Paul	225	

Example

- Specify a calculation to be performed on values of an attribute (and use renaming)

```
SELECT
    name,
    age*age AS agesquare,
    houseNum-2 AS nextdoor
FROM Person;
```

name	agesquare	nextdoor
Jim	225	32
Jo	529	36
Pete	400	36
Jenny	100	40
James	225	48
Paul	225	

Ordering

- Theoretically, rows in a relation have no order (being a set)
- SQL can order the rows: we can specify the order criteria

```
SELECT *
FROM Person
ORDER BY age;
```

name	age	houseNum
Jenny	10	42
James	15	48
Paul	15	
Jim	15	34
Pete	20	38
Jo	23	38

```
SELECT *
FROM Person
ORDER BY age DESC, name;
```

Age “ties”
broken by name

name	age	houseNum
Jo	23	38
Pete	20	38
James	15	48
Jim	15	34
Paul	15	
Jenny	10	42