Relational data

1 A **relation** is a set of named tuples (with a common set of names) and can be visualized as a table with column headers. The **relational data model** represents data as a collection of **relations**.

2 **Relational algebra** is a collection of mathematical operations that may be be performed on relations:

- **Projection**. Subsetting columns.
- **Restriction**. Subsetting rows based on a condition.
- **Cartesian product**. Forming every possible concatenation of a tuple from one relation with a tuple from a second relation.
- **Sorting**. Ordering tuples according to a condition.
- **Grouping and aggregation**. Applying an aggregation function to the values in a column, potentially after grouping the tuples in the relation (partitioning them according to a condition).
- **Renaming**. Changing the name of one of the fields in a relation (changing a column header, essentially).

SQL Queries (PostgreSQL)

1 **SQL** (Structured Query Language) is the standard language for performing the relational algebra operations on tables stored in a relational database.

2 SQL is **declarative**, meaning that we express the result we want to obtain, not the steps the system is supposed to take to achieve that result.

3 SQL input consists of a sequence of **commands**. A command is composed of a sequence of **tokens** and is terminated by a semicolon.

4 A token can be a *keyword*, an *identifier*, a *literal*, or a *special character symbol*. Tokens are separated by whitespace.

5 **Keywords** are reserved words in the language with special meaning. In the statement `SELECT * FROM birds;`, both `SELECT` and `FROM` are keywords.

6 **Identifiers** specify tables, columns, or other database objects (depending on context). `birds` is an identifier which specifies which table we're selecting from.

7 Identifiers may be surrounded by double quotes to ensure they are not interpreted as keywords and to allow them to use otherwise disallowed characters (like whitespace).

8 String literals in SQL are enclosed in single quotes. Numeric literals can be entered like `4`, `3.2`, or `1.925e-3`.

9 Queries use the `SELECT` keyword. The basic structure of a `SELECT` statement is

```
SELECT [select_list] FROM [table_expression] [sort_specification];
```

The table expression is evaluated and then passed to the select list. The sort specification (if present) then processes the resulting rows before they are returned.

10 The **table expression** is an expression that returns a table, like a table name or another `SELECT` statement enclosed in parentheses.

11 The **select list** is a comma-separated list of *value expressions*, which may consist of column identifiers, constant literals, or expressions involving function calls and operators. In this context, the asterisk is a special character meaning "all columns".

12 Each value expression may be assigned a specific name using the `AS` keyword.

```
SELECT
  common_name,
  LENGTH(common_name) AS name_length
  victory_points + egg_capacity AS total_points,
FROM
  birds;
```

common_name	victory_points	egg_capacity
American Robin	1	4
Cedar Waxwing	3	3
Ash-Throated Flycatcher	4	4
Southern Cassowary	4	4
Common Nightingale	3	4

↓

common_name	name_length	total_points
American Robin	14	5
Cedar Waxwing	13	6
Ash-Throated Flycatcher	23	8
Southern Cassowary	18	8
Common Nightingale	18	7

13 The table expression may be modified by further clauses indicated by keywords like `WHERE` or `GROUP BY` or `HAVING`.

14 The sort specification is a clause of the form `ORDER BY [value_expression] [ASC|DESC]`, where the value indicated by the value expression is evaluated for each row and used to perform the sort:

```
SELECT
    *
FROM
    birds
WHERE
    "set" = 'core' AND wingspan > 25
ORDER BY
    wingspan DESC;
```

common_name	set	wingspan
American Robin	core	43
Cedar Waxwing	core	25
Ash-Throated Flycatcher	core	30
Southern Cassowary	oceania	NULL
Common Nightingale	european	23

↓

common_name	set	wingspan
American Robin	core	43
Ash-Throated Flycatcher	core	30

15 *Grouping* by a value expression partitions the tuples in a relation into groups of equal value. If the table expression in a `SELECT` statement has been grouped, then each entry in the select list must be either a value that was grouped on or a call to an **aggregate** function (like `SUM`, `AVG`, `MAX`, `MIN`, or `COUNT`, which reduces a column of values to a single value).

```
SELECT fruit,
       MAX(LENGTH(common_name)) AS max_name_length
FROM birds
GROUP BY fruit;
```

common_name	fruit
American Robin	1
Cedar Waxwing	2
Ash-Throated Flycatcher	1
Southern Cassowary	2
Common Nightingale	1

↓

common_name	fruit
American Robin	1
Ash-Throated Flycatcher	1
Common Nightingale	1
Cedar Waxwing	2
Southern Cassowary	2

↓

fruit	max_name_length
1	23
2	18

16 Filter results from a grouped and aggregated relation using a `HAVING` clause.

17 Use `LIMIT [limit] OFFSET [offset]` after an `ORDER BY` clause to return at most `limit` records beginning at index `offset`.

18 Name a temporary table using `WITH`. Example: select every card from whichever expansion set has the largest average egg capacity:

```
WITH set_eggs AS (
    SELECT "set",
    AVG(egg_capacity) AS avg_eggs
    FROM birds
    GROUP BY "set"
    ORDER BY avg_eggs DESC LIMIT 1
)
SELECT * FROM birds
WHERE "set" IN (SELECT "set" FROM set_eggs);
```

19 A comma-separated list of two relations denotes their **Cartesian product**. To look at every (bird card, bonus card) combination:

birds		
common_name	set	wingspan
American Robin	core	43
Cedar Waxwing	core	25
Ash-Throated Flycatcher	core	30
Southern Cassowary	oceania	NULL
Common Nightingale	european	23
Sulphur-Crested Cockatoo	oceania	103

bonus_cards	
name	condition
Passerine Specialist	wingspan ≤ 30
Large Bird Specialist	wingspan > 64

↓

`SELECT * FROM birds, bonus_cards;`

common_name	set	wingspan	name	condition
American Robin	core	43	Passerine Specialist	wingspan ≤ 30
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Southern Cassowary	oceania	NULL	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Passerine Specialist	wingspan ≤ 30
American Robin	core	43	Large Bird Specialist	wingspan > 65
Cedar Waxwing	core	25	Large Bird Specialist	wingspan > 65
Ash-Throated Flycatcher	core	30	Large Bird Specialist	wingspan > 65
Southern Cassowary	oceania	NULL	Large Bird Specialist	wingspan > 65
Common Nightingale	european	23	Large Bird Specialist	wingspan > 65
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65

20 Cartesian products are usually combined with a `WHERE` clause. To find which (bird, bonus card) combinations actually yield bonuses:

```
SELECT * FROM birds, bonus_cards
WHERE wingspan <= 30 AND condition = 'wingspan ≤ 30'
   OR wingspan > 65 AND condition = 'wingspan > 65';
```

common_name	set	wingspan	name	condition
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65

21 Cartesian products with restrictions are important enough to warrant their own syntax: `[table1] JOIN [table2] ON [condition]`

```
SELECT * FROM birds JOIN bonus_cards
ON wingspan <= 30 AND condition = 'wingspan ≤ 30'
   OR wingspan > 65 AND condition = 'wingspan > 65';
```

22 Joins come in several flavors:

- `JOIN` or `INNER JOIN`. Cartesian product followed by restriction.
- `LEFT OUTER JOIN`. Inner join followed by adding a single row for each row from the first table completely eliminated by the restriction. Those rows get `NULL` values for second-table fields.

`SELECT * FROM birds LEFT OUTER JOIN bonus_cards;`

common_name	set	wingspan	name	condition
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65
American Robin	core	43	NULL	NULL
Southern Cassowary	oceania	NULL	NULL	NULL

- `RIGHT OUTER JOIN`. Same but for eliminated rows from the *second* table.
- `FULL OUTER JOIN`. Same but for eliminated rows from *either* table.

- `CROSS JOIN`. Cartesian product with no restriction.
- `NATURAL JOIN`. Inner join on equality comparison of all pairs of identically named fields.

23 We can take a union of tuples in two relations (with the same field names) using the `UNION` operator. We can take a set difference using `EXCEPT` and the intersection using `INTERSECT`.

24 The syntax for a table literal is `VALUES (row1), (row2), (row3);` To add two rows manually:

```
(SELECT common_name, "set" FROM birds)
UNION
(VALUES ('Western Tanager', 'core'),
('Scissor-Tailed Flycatcher', 'core'));
```

SQL: Modifying Data

1 To add rows to a database:

```
INSERT INTO
    birds(common_name, "set")
VALUES
    ('Western Tanager', 'core'),
    ('Scissor-Tailed Flycatcher', 'core');
```

2 To update rows to a database:

```
UPDATE
    birds
SET
    wingspan = 0
WHERE
    wingspan IS NULL;
```

3 To delete rows:

```
DELETE FROM
    birds
WHERE
    "set" NOT IN ('core', 'oceania', 'european');
```

SQL: Managing Tables

1 Creating a new table. To make a new table called **`birds`** a text field **`common_name`** which will be used as a primary key, a text field **`set`** which is a foreign key for the **`name`** column in another table called **`expansions`**, and an integer field **`wingspan`** which should not be allowed to be negative:

```
CREATE TABLE birds (
    common_name TEXT PRIMARY KEY,
    "set" TEXT REFERENCES expansions(name),
    wingspan INTEGER CHECK (wingspan >= 0),
);
```

2 PostgreSQL

- `BIGINT/INT8` signed eight-byte integer
- `INTEGER/INT/INT4` signed four-byte integer
- `DOUBLE PRECISION/FLOAT8` double precision floating-point number (8 bytes)
- `REAL/FLOAT4` single precision floating-point number (4 bytes)
- `BOOLEAN/BOOL` logical Boolean (true/false)
- `VARCHAR(n)` variable-length character string (max `n` characters)
- `TEXT` variable-length character string
- `DATE` calendar date (year, month, day)
- `MONEY` currency amount
- `NUMERIC [ (p, s) ]` exact numeric of selectable precision
- `TIMESTAMP` date and time
- `UUID` universally unique identifier

3 To drop a table: `DROP TABLE [table_name];`

4 To remove all data from a table: `TRUNCATE TABLE [table_name];`

5 To add a column: `ALTER TABLE [table_name] ADD [column_name column_type];`

SQL Functions

1 Common SQL operators:

- `AND`, `OR`, `NOT`. Logical operators.
- `<`, `>`, `<=`, `>=`, `=`, `<>` (not equal). Comparison operators.
- `IS NULL`, `IS NOT NULL`. Null checks.
- `LIKE`, `NOT LIKE`. SQL-style pattern matching. Use _ for any single character `%` for any sequence of zero or more characters. `'abc' LIKE '_b_'` returns `TRUE`.
- `~`, `!~`, `~*`, `!~*`. Ordinary regular expression matching. `!` for negation, `*` for case-insensitivity.

2 Arithmetic operators and functions:

Operator or function	Name	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division	4 / 2	2
%	modulo (remainder)	5 % 4	1
^	exponentiation	2.0 ^3.0	8
|/	square root	|/ 25.0	5
!	factorial	5 !	120
@	absolute value	@ -5.0	5
abs(x)	absolute value	abs(-17.4)	17.4
ceil(x)	least integer	ceil(-42.8)	-42
div(y, x)	integer quotient	div(9,4)	2
exp(x)	exponential	exp(1.0)	2.718
floor(x)	greatest integer	floor(-42.8)	-43
ln(x)	natural logarithm	ln(2.0)	0.693
log(x)	base 10 logarithm	log(100.0)	2
log(b, x)	logarithm to base b	log(2.0, 64.0)	6.0
mod(y, x)	remainder of y/x	mod(9,4)	1
pi()	π	pi()	3.14
round(x)	round to nearest integer	round(42.4)	42
round(v, s)	round to s decimal places	round(42.4382, 2)	42.44
sign(x)	signum (-1, 0, +1)	sign(-8.4)	-1
trunc(x)	truncate toward zero	trunc(42.8)	42
trunc(v, s)	truncate to s dec. places	trunc(42.4382, 2)	42.43
width_bucket(x,b1,b2,n)	histogram bucket	width_bucket(1,-3,3,5)	4
cos(x)	inverse cosine	cos(1.05)	0.5
acos(x)	inverse cosine	acos(0.5)	1.05

3 String operators and functions:

Operator or function	Name	Example	Result
string || string	String concatenation	'Post' || 'greSQL'	PostgreSQL
lower(string)	Convert string to lower case	lower('TOM')	tom
overlay(string placing string from int [for int])	Replace substring	overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas
position(substring in string)	Location of specified substring	position('om' in 'Thomas')	3
substring(string [from int] [for int])	Extract substring	substring('Thomas' from 2 for 3)	hom
substring(string from pattern)	Extract substring matching pattern	substring('Thomas' from '...$')	mas
trim([leading | trailing | both] [characters] from string)	Remove characters from ends	trim(both 'x' from 'xTomxx')	Tom
upper(string)	Convert string to upper case	upper('Tom')	TOM
left(string, n)	first n chracters	left('abcde',2)	ab
lpad(string, n, char)	left pad	lpad('5',3,'0')	005
reverse(string)	reverse	reverse('abc')	'cba'

SQL: Setup

1 Easiest way to create a free cloud Postgres instance: Go to **`supabase.io`** > Log in with GitHub > Create an Organization > Create a New Project > [wait a few minutes, and in the meantime add the line **`export DATABASE_PWD="your-pwd-here"`** to your bash profile] > Go into the new project > Settings (gear icon) > Database > Connection String (bottom) > PSQL > Copy.

2 macOS local installation: **`https://postgresapp.com/`**. Instructions on the landing page for finding your connection string. To install locally on Windows: **`https://www.postgresql.org/download/windows/`**.

3 To connect from a Python session, paste the connection string replacing **`[YOUR-PASSWORD]`** with **`{pwd}`**, like this:

```
import sqlalchemy
import os
pwd = os.envget("DATABASE_PWD") # retrieve password from bashrc
connection_string = (
   f"postgresql://postgres:{pwd}@"
    "db.bijsjfasiwdlfkjasdfot.supabase.co:5432/postgres"
) # should be your connection string instead
engine = create_engine(connection_string)
connection = engine.connect()
sql = "SELECT * FROM pg_catalog.pg_tables LIMIT 10;"
connection.execute(sql).fetchall()
```

4 Create a new table in the database from a Pandas dataframe:

```
import pandas as pd
df = pd.read_csv("https://bit.ly/iris-dataset")
df.to_sql("iris", con=engine)
```