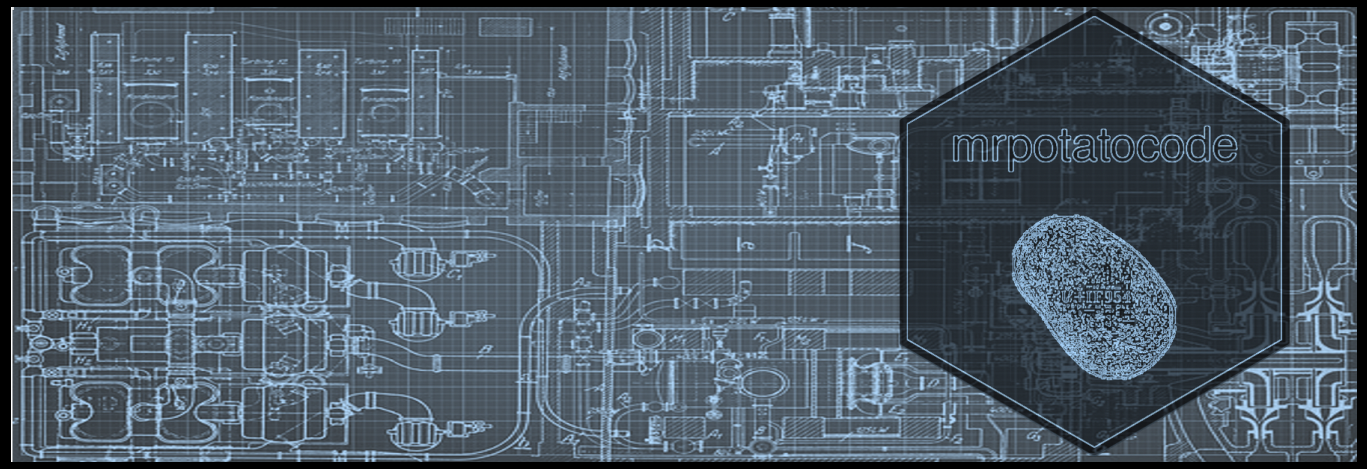


Introduction to Data Access and Storage

Thomas Rosenthal - DSI @ UofT

Module 04

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

String Manipulation

LTRIM & RTRIM

REPLACE

UPPER & LOWER

Concatenation

(...)

String Manipulation (continued)

SUBSTR

LENGTH

CHAR & UNICODE

REGEXP

String Manipulation

LTRIM & RTRIM

REPLACE

UPPER & LOWER

Concatenation

LTRIM & RTRIM

LTRIM & RTRIM

- `LTRIM` and `RTRIM` serve two purposes in SQLite:
  - Their main function is to remove leading or trailing white spaces from strings
  - This is surprisingly common — many SQL DBs are populated by human input, and this is a frequently overlooked input error
  - e.g. ' Thomas Rosenthal '
  - Alternatively, they act similarly to `REPLACE` (coming up next), but within their specific context:
  - `LTRIM` removes any specified set of characters from the *left*
  - `RTRIM` removes any specified set of characters from the *right*
    - The usefulness of this is going to be very case specific:
    - e.g. wanting to remove a prefix/suffix of an ID:
      - `LTRIM("A189A", 'A')` would result in '189A'
      - `RTRIM("A189A", 'A')` would result in 'A189'
      - `REPLACE` would remove both A's: '189'

LTRIM & RTRIM

(`LTRIM` & `RTRIM` live coding)

String Manipulation

LTRIM & RTRIM

REPLACE

UPPER & LOWER

Concatenation

REPLACE

- `REPLACE` is likely going to be one of your most commonly used string manipulations
- It substitutes a character or set of characters with another
  - We specify which string (or set of strings within a column in many cases), what we want to replace, and the replacement value'
  - e.g. `REPLACE('A is an excellent instructor','instructor','TA')` results in 'A is an excellent TA'
  - You can also replace a character with nothing, using an empty string ''
  - e.g. `REPLACE('flour','u','')` results in 'flow'

- `REPLACE` statements can be strung together — the innermost function will be executed first
  - e.g. `REPLACE(REPLACE(REPLACE(REPLACE('A?lot-of,punctuation.','.',''),',',''),'-',''),'?','')` results in 'A lot of punctuation'

REPLACE

(REPLACE live coding)

String Manipulation

LTRIM & RTRIM

REPLACE

UPPER & LOWER

Concatenation

UPPER & LOWER

- `UPPER` forces all string characters to be uppercase
- `LOWER` does the opposite
- Both `UPPER` and `LOWER` are essential for filtering tables based on strings
  - It always best to assume that there is some string variety
  - Sometimes a `%LIKE` statement will not be an option

annoying_string_column
WORD
Word
word
wOrD
DifferentWord

- We can always use `UPPER` or `LOWER` in a `WHERE` clause, even without using the commands in the `SELECT` statement

```
SELECT annoying_string_column
FROM table
WHERE LOWER(annoying_string_column) = 'word'
```

UPPER & LOWER

(`UPPER` & `LOWER` live coding)

String Manipulation

LTRIM & RTRIM

REPLACE

UPPER & LOWER

Concatenation

Concatenation (sometimes CONCAT, flavour dependent)

- String concatenation combines two or more columns into another
- Concatenation can handle non-column values too
  - e.g. `first_name || ' ' || last_name as full_name`
  - or `last_name || ', ' first_name AS full_name`
- In SQLite, `CONCAT` is replaced by two vertical bar characters `||`
  - most other flavours use `CONCAT`
- By default, spaces are not included between columns
  - i.e., you need to add a blank space between quotes

Concatenation

(Concatenation live coding)

String Manipulation (continued)

SUBSTR

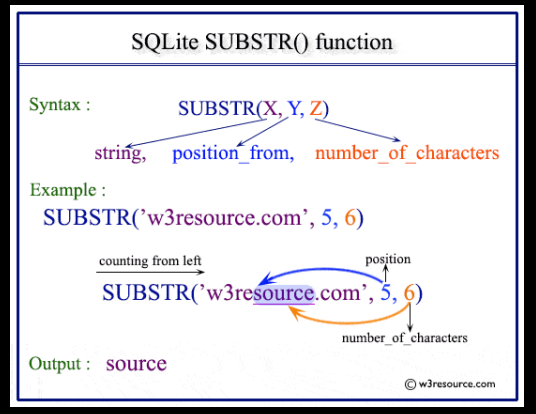
LENGTH

CHAR & UNICODE

REGEXP

SUBSTR ("substring")

- `SUBSTR` specifies any section of a string to return, based on:
  - which string (i.e. column)
  - where to begin the section (i.e. the string position to start, as an integer)
  - the (optional) number of characters to return (i.e. how far to go, as an integer)
- `SUBSTR` replaces flavour specific functions like `LEFT` or `RIGHT`
  - by default `SUBSTR` counts to the left
    - e.g. `substr('a long string', 3, 4)` will return "long"
  - to count from the right, specify a negative number for start
    - e.g. `substr('a long string', -6, 6)` will return "string"



SUBSTR

(SUBSTR live coding)

String Manipulation (continued)

SUBSTR

LENGTH

CHAR & UNICODE

REGEXP

LENGTH

- `LENGTH` returns the number of characters in a given string (or set of strings in a column)
  - `LENGTH` also works on integers
- `LENGTH` is perhaps less of a string manipulation in of itself, but is useful in debugging
  - Combined with `MAX`, `LENGTH` can be a useful, especially when adding string length constraints to a column
  - Combined with `SUBSTR`, `LENGTH` can cut strings within a column by a dynamic value
  - e.g. **What happens when we apply `SELECT SUBSTR(CanadianMusicians, 0, LENGTH(CanadianMusicians)-6)` to the table below?**

CanadianMusicians
Neil Young
Leonard Cohen
Shania Twain
Michael Bublé

CanadianMusicians
Neil
Leonard
Shania
Michael

LENGTH

(`LENGTH` live coding)

String Manipulation (continued)

SUBSTR

LENGTH

CHAR & UNICODE

REGEXP

CHAR & UNICODE

CHAR

- When provided an ASCII value, `CHAR` will return the appropriate character from the ASCII table
  - e.g. `CHAR(97)` will result in a
  - pronunciation is split on "char":
  - "char" as in "*char*-broiled"
  - "char" as in "*car*"
  - "char as in "_char_acter"
  - "char" as in "*care*"

- `CHAR` is hugely useful with replace
  - occasionally, line breaks affect SQL column validity, `REPLACE(br_column,CHAR(10),'')` and/or `REPLACE(cr_column,CHAR(13),'')` will be hugely useful
- `CHAR` can help with structure and control on strings as they flow into columns

UNICODE (ASCII in some flavours)

- `UNICODE` provides the ASCII value of any given character
  - i.e. the opposite of `CHAR`

CHAR & UNICODE

(`CHAR` & `UNICODE` live coding)

String Manipulation (continued)

SUBSTR

LENGTH

CHAR & UNICODE

REGEXP

REGEXP (flavour dependent)

- `REGEXP` allows for string filtering based on regular expressions (regex)
- Situated within a `WHERE` clause, very similar to `LIKE`
- Can use either SQL's or regex's Boolean operators
  - e.g. `WHERE book_title REGEXP '(tion|ice)$'`
  - or `WHERE book_title REGEXP 'tion$' OR book_title REGEXP 'ice$'`

REGEXP (flavour dependent)

(Quick `REGEXP` live coding)

> Some people, when confronted with a problem think: "I know, I'll use regular expressions." Now they have two problems - Jamie Zawinski (probably)

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

CROSS JOIN

- `CROSS JOIN` creates an unfiltered Cartesian Product
- They are not joined on any columns
- Recall our deck of cards example in Module 2:

```
SELECT suit, rank
FROM suits
CROSS JOIN ranks
```

- Because tables suits and ranks contain no common columns, we would have no other means to join

- I love CROSS JOINs!
- They can be super useful when used correctly
  - What are some good examples that could be useful?

CROSS JOIN

(`CROSS JOIN` live coding)

> No complex query is complete without at least one `CROSS JOIN` (me, jokingly)

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

Self Joins

- Self Joins are somewhat uncommon, but are the last type of possible joins
- They are useful for comparison:
    - Determine Maximum to-date
    - generating pairings

-They can help with hierarchy

- Child to Parent relationships
- The syntax is as we might expect:

```
SELECT
e.name as employee_name,
m.name as manager_name

FROM people e
LEFT JOIN people m ON e.manager_id = m.id
```

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

UNION & UNION ALL

- `UNION` and `UNION ALL` combine the results of two or more queries vertically (rowwise)
- `UNION ALL` keeps duplicate values, whereas `UNION` removes them
  - the difference between the two is one of the most common interview questions!
- `UNION` and `UNION ALL` require both/all queries to have the same number of columns
  - You can `UNION` unrelated columns if you had a specific use-case for it
    - Column names will come from the first query
  - In situations where you don't have exactly the same columns, but still need to `UNION`, you can pass a NULL (or zero, or blank) column
    - Similarly, you can pass a string character to keep which data is associated to which query

```
SELECT number_of_chips, number_of_tacos, 0 AS number_of_burritos, 'lunch' AS meal
FROM lunch

UNION

SELECT NULL as number_of_chips, number_of_tacos, number_of_burritos, 'dinner' AS meal
FROM dinner
```

UNION & UNION ALL

- If we recall back to SQLite's lack of support for `FULL OUTER JOINS`, `UNION ALL` will allow us to emulate one:

```
SELECT s1.quantity, s1.costume, s2.quantity
FROM store1 s1
LEFT JOIN store2 s2 ON s1.costume = s2.costume

UNION ALL

SELECT s1.quantity, s2.costume, s2.quantity
FROM store2 s2
LEFT JOIN store1 s1 ON s2.costume = s1.costume

WHERE s1.quantity IS NULL
```

UNION & UNION ALL

(`UNION` & `UNION ALL` live coding)

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

INTERSECT & EXCEPT

- Both `INTERSECT` & `EXCEPT` function require all queries to have the same number of columns

INTERSECT

- `INTERSECT` returns data in common to both `SELECT` statements
- Values returned will be distinct
- What's the difference between `INTERSECT` and `INNER JOIN`?

EXCEPT

- `EXCEPT` returns the opposite of an `INTERSECT`:
  - for whatever is returned by first `SELECT` statement, `EXCEPT` will return what is *not* returned by the second `SELECT` statement
- The "direction" of `EXCEPT` matters a lot
  - `EXCEPT` is relative the the first `SELECT` statement, so changing which comes first will always change the results of the query

INTERSECT & EXCEPT

Let's consider an example:

For the following `product` table,

product	product_id
blue bike	1
tiger onesie	2
house plant	3
headphones	4

and `order` table,

order_id	product_id
1	1
2	1
3	1
4	4

`INTERSECT` will find all products with work orders

```
SELECT product_id FROM product
INTERSECT
SELECT product_id FROM orders
```

Resulting in product_id's 1 & 4

`EXCEPT` will find all products will find all products *without* work orders

```
SELECT product_id FROM product
EXCEPT
SELECT product_id FROM orders
```

Resulting in product_id's 2 & 3

OR all work orders *without* products

```
SELECT product_id FROM orders
EXCEPT
SELECT product_id FROM product
```

Resulting in nothing (because no orders have a product_id that is not found in the product table)

INTERSECT & EXCEPT

(`INTERSECT` & `EXCEPT` live coding)

Advanced Techniques:

String Manipulation

CROSS JOIN

Self Joins

UNION & UNION ALL

INTERSECT & EXCEPT

INSERT, UPDATE, DELETE

Prior to this, we've focused solely on retrieving values from tables:

- Tables can also be manipulated with `INSERT`, `UPDATE`, and/or `DELETE`
- *A word of warning...these commands are PERMANENT* 😱
    - Generally, follow a policy that avoids altering data
    - Make backups to tables before you run a query
    - Never hurts to test on a temporary table first!
- But they are useful, and sometimes the correct solution!
- There is no `SELECT` statement for these types of queries

INSERT, UPDATE, DELETE

INSERT

- `INSERT` allows you to add a record
- Specify where you want to add:
  - `INSERT INTO [some_table_name]`
- ...and what you to add:
  - `VALUES(column_one_value, column_two_value)`
- `VALUES` come in the order of the columns within the tables
- `VALUES` must respect table constraints
  - e.g. NULLs, UNIQUE, data types, etc
- `INSERT` can help you create small helper tables
  - Can you think of any scenarios?

INSERT, UPDATE, DELETE

UPDATE

- `UPDATE` allows you to change a record
- Specify where you are making your change:
  - `UPDATE [some_table_name]`
- ...and what you to change:
  - `SET column_one = value1, column_one = value2`
- *SPECIFY A `WHERE` CONDITION*
  - `WHERE condition`
- You can change a single column, a few columns, all the columns, etc
  - (Respecting table constraints)
- **What happens if you don't specify a WHERE condition?**

INSERT, UPDATE, DELETE

DELETE

- `DELETE` allows you to remove a record
- Specify where you want to delete:
  - `DELETE FROM [some_table_name]`
- *SPECIFY A `WHERE` CONDITION*
  - `WHERE condition`
- **What happens if you don't specify a WHERE condition?!?**
- `DELETE` doesn't *remove* a table from a database

INSERT, UPDATE, DELETE

(`INSERT`, `UPDATE`, `DELETE` live coding with a TEMP TABLE)

Lingering Questions?

What happens if you don't specify a `WHERE` condition