SQL Querying

Springboard

Download Demo SQL Code

Intro

Goals

- Learn core querying
- Learn insertion, updating, and deletion

SQL DML

Data Manipulation Language

DML is a subset of SQL that involves querying and manipulating records in existing tables.

Most of the DML you'll be doing will be related to **CRUD** operations on rows.

What's CRUD?

Letter	Verb	SQL Commands
C	Create	***INSERT INTO***
R	Read	***SELECT ... FROM***
U	Update	***UPDATE ... SET***
D	Delete	***DELETE FROM***

SELECT

SELECT is the most flexible and powerful command in SQL

It selects rows (included summary data, roll-up data, etc) from table(s)

SELECT statements have subclauses, which are performed in this order:

#	Clause	Purpose	Required?
1	FROM	Select and join together tables where data is	No
2	WHERE	Decide which rows to use	No
3	GROUP BY	Place rows into groups	No
4	SELECT	Determine values of result	Yes
5	HAVING	Determine which grouped results to keep	No
6	ORDER BY	Sort output data	No
7	LIMIT	Limit output to *n* rows	No
8	OFFSET	Skip *n* rows at start of output	No

FROM

Determine which table(s) to use to get data:

all info from books table

```
SELECT *
  FROM books;
```

You can get data from more than one table by "joining" them — we'll discuss this later

WHERE

Filter which rows get included:

only books with price over $10

```
SELECT *
  FROM books
  WHERE price > 10;
```

GROUP BY

Reduce the amount of rows returned by grouping rows together:

group by author, show name & # books

```
SELECT author, COUNT(*)
  FROM books
  GROUP BY author;
```

SELECT

Only at this point do the ***SELECT*** values get calculated:

return only author name & count-of-books fields

```
SELECT author, COUNT(*)
  FROM books
  GROUP BY author;
```

HAVING

Decide which group(s), if grouped, to keep:

only show groups with more than 2 books

```
SELECT author, COUNT(*)
  FROM books
  GROUP BY author
  HAVING COUNT(*) > 2;
```

ORDER BY

Arrange output in order (sort):

order results by author name (A → Z)

```
SELECT title, author, price
  FROM books
  ORDER BY author;
```

LIMIT

Only show *n* number of rows:

only show first 5 rows

```
SELECT title, author, price
  FROM books
  ORDER BY price
  LIMIT 5;
```

OFFSET

Skip *n* number of rows. Used in combination with ***LIMIT*** to **paginate** results.

skip first row

```
SELECT title, author, price
  FROM books
  ORDER BY price
  OFFSET 1;
```

Some Example SELECTs

```
-- Select all rows and all columns from the books table
SELECT * FROM books;

-- Select all rows and two columns from the books table
SELECT title, author FROM books;

-- Select ten rows and two columns from the books table
SELECT title, author FROM books LIMIT 10;

-- Select all columns from short books
SELECT * FROM books WHERE page_count < 150;
```

SQL Operators

Operators are used to build more complicated queries They involve reserved SQL keywords

These include ***IN***, ***NOT IN***, ***BETWEEN***, ***AND***, and ***OR***

An example

```
-- basic WHERE clause
SELECT title FROM books WHERE author = 'J. K. Rowling';

-- basic IN clause
SELECT * FROM books WHERE id IN (1, 12, 30);

-- grab books of moderate length
SELECT title, author
  FROM books
  WHERE page_count BETWEEN 300 AND 500;
```

Another example

```
-- short, cheap books
SELECT title, author
  FROM books
  WHERE price < 10
    AND page_count < 150;

-- new books or expensive books
SELECT title, author, publication_date, price
  FROM books
  WHERE publicatton_date > '01-01-2017'
    OR price > 100;

-- books ordered by price
SELECT title, price
  FROM books
  ORDER BY price;
```

SQL Aggregates

Aggregates are used to combine multiple rows together to extract data

Common aggregate functions include ***COUNT***, ***AVG***, ***SUM***, ***MIN***, and ***MAX***

An Example

```
-- count all books
SELECT COUNT(*) FROM books;

-- count all Kyle Simpson books
SELECT COUNT(*) FROM books WHERE author = 'Kyle Simpson';

-- find page count for longest book
SELECT MAX(page_count) FROM books;

-- find cheapest price
SELECT MIN(price) FROM books;

-- find total number of pages
SELECT SUM(page_count) FROM books;

-- find average price
SELECT AVG(price) FROM books;
```

GROUP BY

The ***GROUP BY*** and ***HAVING*** clauses are often used with aggregate functions

An example

```
-- how many books did each author write?
SELECT author, COUNT(*)
  FROM books
  GROUP BY author;

-- let's only consider authors with at least 2 books
SELECT author, COUNT(*)
  FROM books
  GROUP BY author
  HAVING COUNT(*) > 1;

-- let's order our authors from most to least prolific
SELECT author, SUM(page_count) AS total
  FROM books
  GROUP BY author
  ORDER BY total DESC;
```

Modifying Data

Creating data with INSERT

```
-- Inserting a new book with title and author
INSERT INTO books (title, author)
  VALUES ('The Iliad', 'Homer');

-- Inserting several books with titles only
INSERT INTO books (title) VALUES
  ('War and Peace'),
  ('Emma'),
  ('Treasure Island');
```

Note: INSERT via SELECT

You can combine INSERT and SELECT to copy data from another table.

```
INSERT INTO books (title, author)
  SELECT title, author
    FROM some_other_table
    WHERE price < 10;
```

Updating data with UPDATE

```
-- Matt is a prolific writer
UPDATE books SET author = 'Matt';

-- OK, not that prolific!
UPDATE books SET author = 'Jane Austen' WHERE title = 'Emma';
```

Deleting data with DELETE

```
-- delete Emma
DELETE FROM books WHERE title = 'Emma';

-- delete long books
DELETE FROM books WHERE num_pages > 200;

-- delete all books!
DELETE FROM books;
```