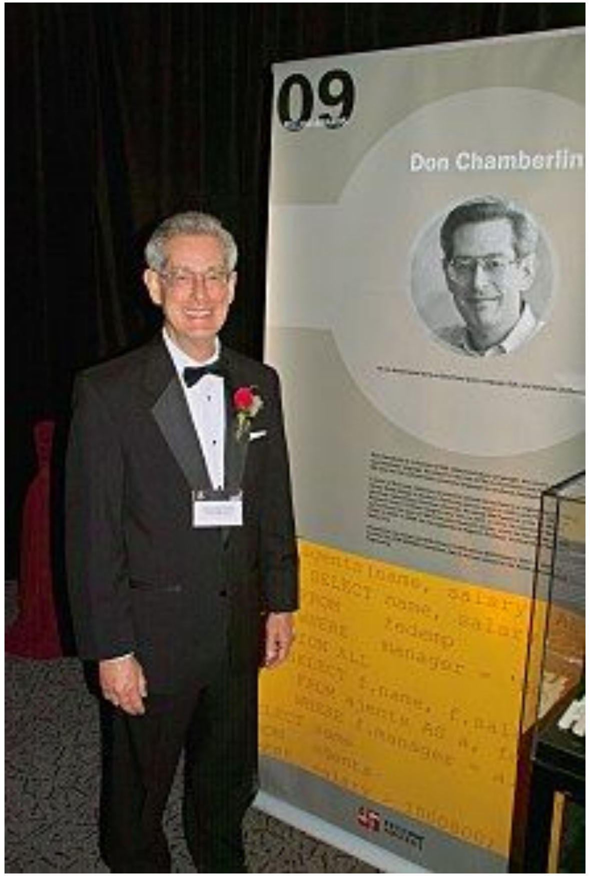
MONTANA
STATE UNIVERSITY

SQL Introduction

History and the SELECT statement

SQL History

- Recall, developed by IBM
- Donald Chamberlin & Raymond Boyce
- Originally called SEQUEL but changed to SQL due to trademark issues



SQL History

- Initially every vendor had its own variant of SQL
- Oracle SQL was not compatible with DB/2 SQL
- This is still true to an extent
  - MySQL & Postgresql have different features and flavors
  - Case sensitivity is a big one!
    - MySQL - not case sensitive

SQL History

- Standardization efforts
- SQL86, SQL89...SQL2016
- SQL99 is a popular standard
  - Standardized majority of the SQL language
  - Was heavily referred to during the dotCom era
  - MySQL kinda sorta implemented it

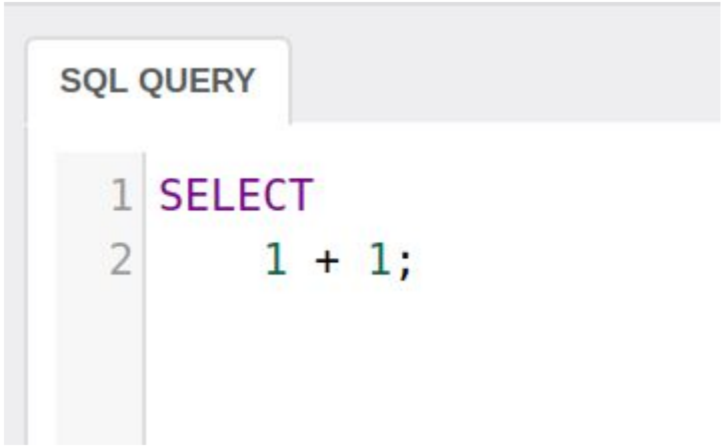
SQL As A Language

- SQL is a *declarative* programming language
  - You tell the computer what you want, not how to get it
- It's a functional language!
- Perhaps the most successful functional language in history
  - Eat it, Haskell nerds

DEALING WITH A FUNCTIONAL LANGUAGE LIKE HASKELL
ENJOYING SOME PLEASANT SQL

SQL As A Language

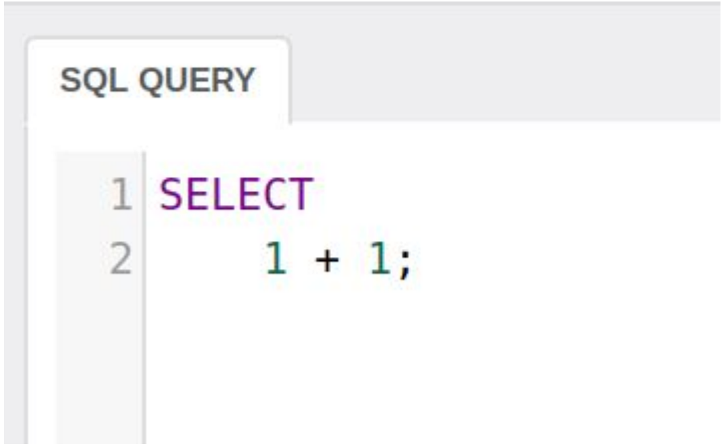
- SQL consists of a variety of statements
- Today we will be talking about the SELECT statement
- As you might suspect, a select statement starts with the word SELECT



```
SELECT
     1 + 1;
```

SQL As A Language

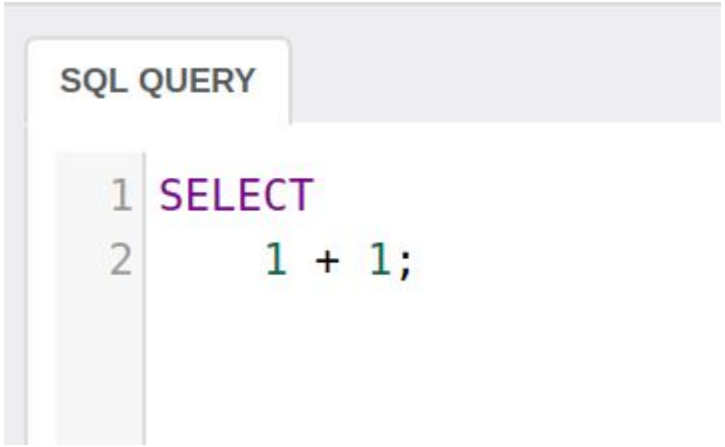
- SQL is NOT case sensitive
- However, there are case conventions
    - Key words are all caps
    - Tables are *often* capitalized
    - Columns are *usually* lower case
        - Sometimes camel case, sometimes underscore separated



```
SELECT
    1 + 1;
```

SQL - Select

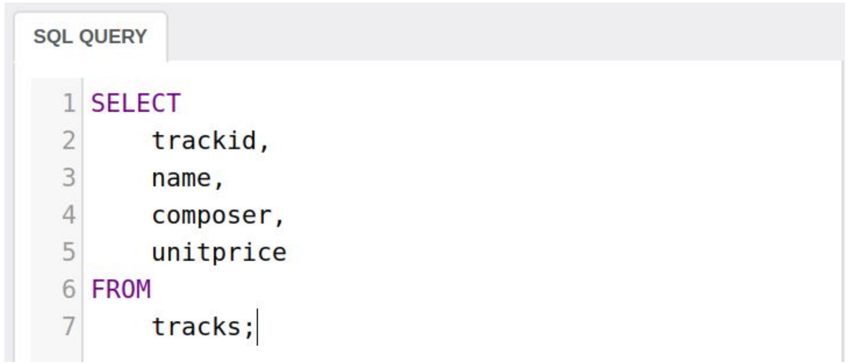
- The SELECT statement is the most complex statement in SQL
- As you can see to the right, mathematical statements are possible
- The result of this select statement is the value 2
    - Earth shattering, I know



```
SELECT
     1 + 1;
```

SQL - Select

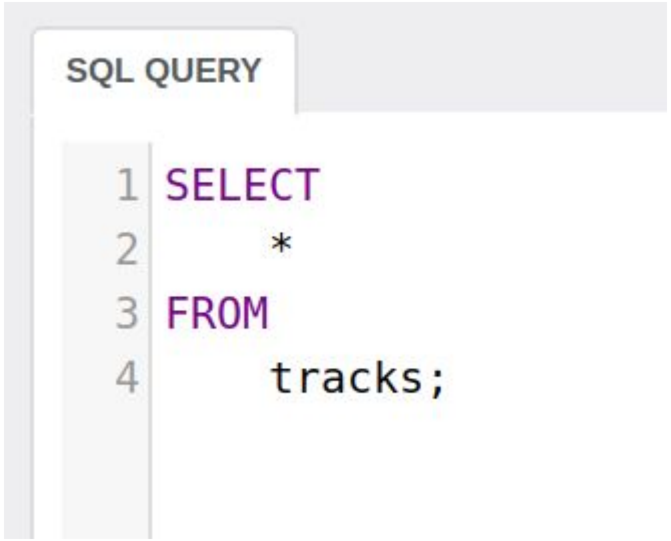
- That's not very interesting, let's actually select some data from our database
- Here you see selecting specific columns **FROM** a specific table
- Returns these column values for all rows


SQL QUERY

```
SELECT
    trackid,
    name,
    composer,
    unitprice
FROM
    tracks;
```

SQL - Select

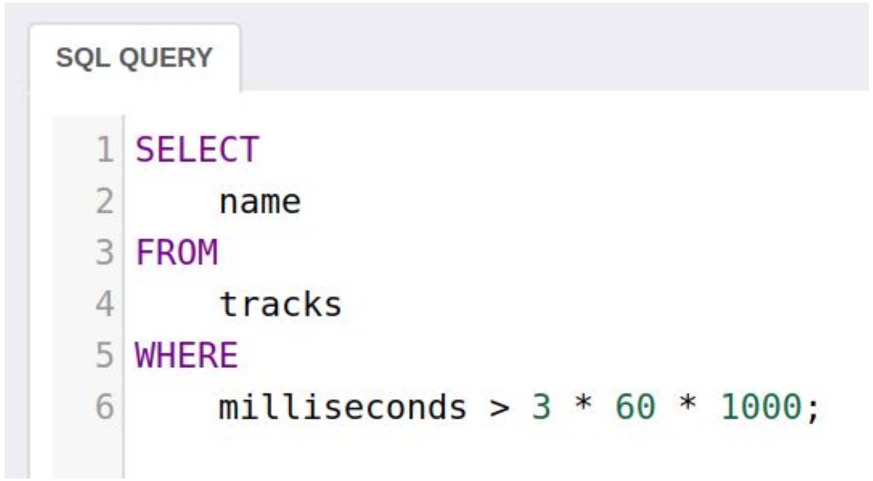
- Perhaps you want to select *all* values from a row
- You can use the asterisk operator to return all columns
- Pros
  - Shorter
  - Easier to get right
- Cons
  - Might bring back unused data
  - Unclear what columns are available



```
SELECT
    *
FROM
    tracks;
```

SQL - Where

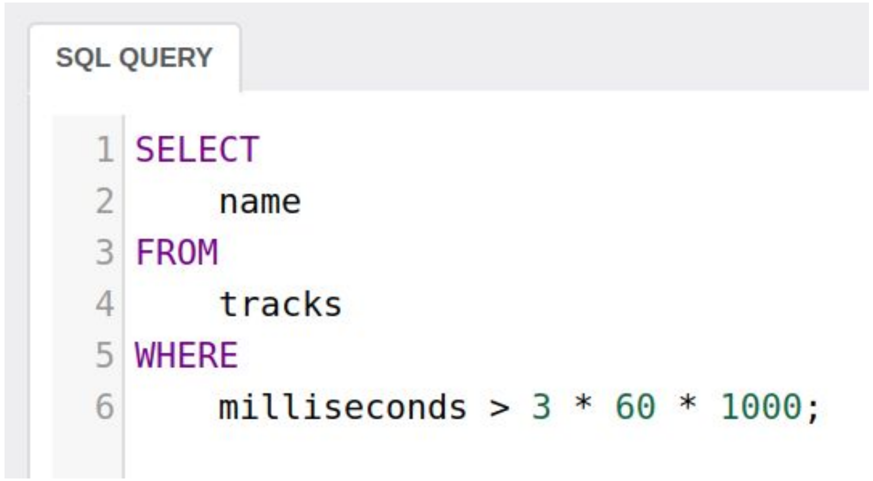
- Typically not useful to bring back all the data of a table
- You often want to find a particular piece of data
- Enter the WHERE clause



```sql
SELECT
    name
FROM
    tracks
WHERE
    milliseconds > 3 * 60 * 1000;
```

SQL - Where

- The WHERE clause allows you to give *predicates* that a row must satisfy in order to be included in the results
- *Show me the name of all tracks that are longer than 3 minutes*



```sql
SELECT
    name
FROM
    tracks
WHERE
    milliseconds > 3 * 60 * 1000;
```

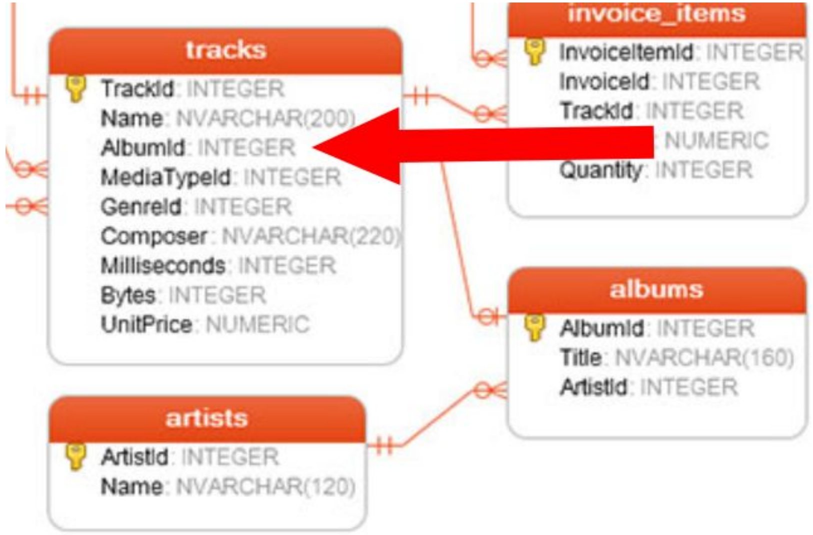
SQL - Select

- This is the general form of SELECT
- Simple, but *powerful*

```
SELECT

        column_list

FROM

        table

WHERE

        search_condition;
```

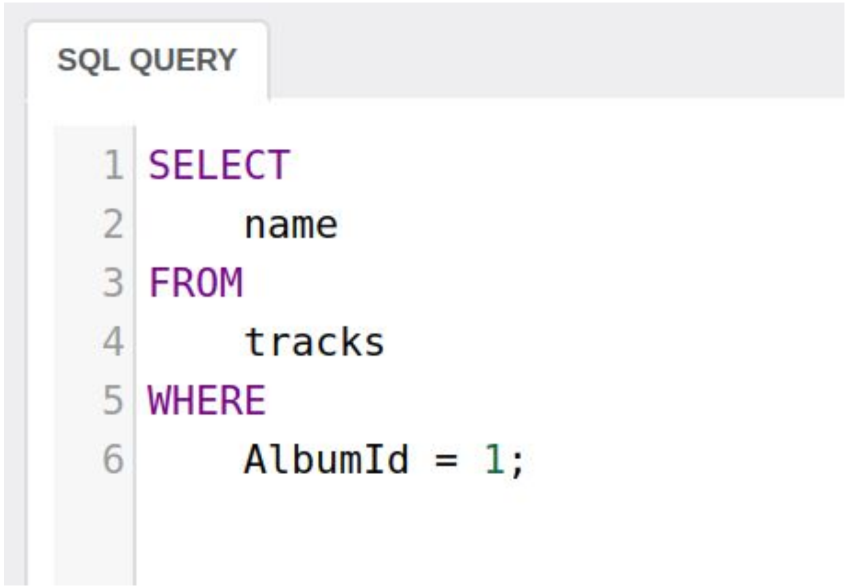
SQL - Where

- Recall foriegn keys
- There is a 1-N relationship between albums and tracks
- The *tracks* table has an *AlbumId* column
- We can use this column in a WHERE



SQL - Where

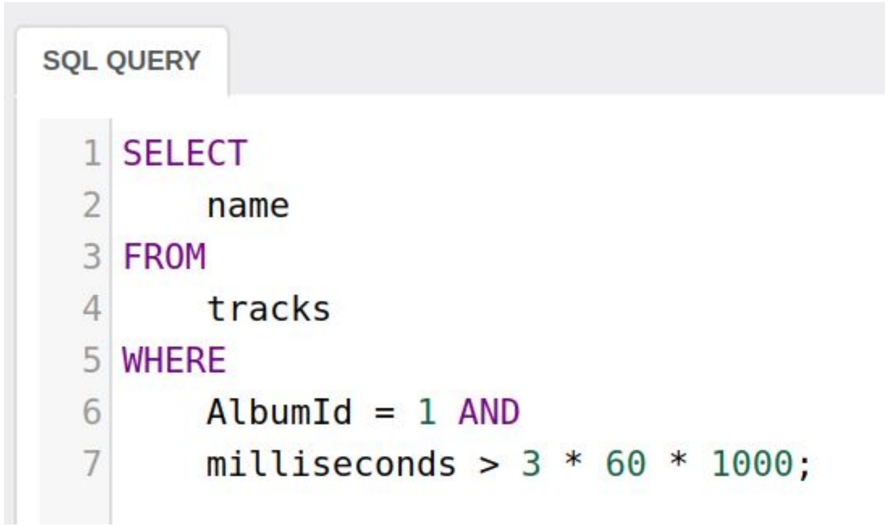
- *Give me the name all the tracks on the album with the AlbumID of 1*
- Note that the = operator is a single character!



```
SELECT
    name
FROM
    tracks
WHERE
    AlbumId = 1;
```

SQL - Combining Predicates

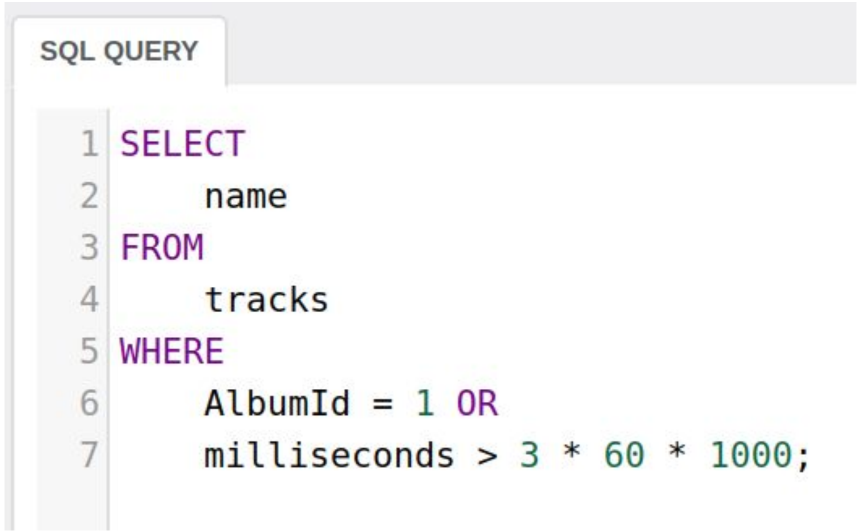
- You can combine predicates using the AND and OR expressions
- *Give me the name all the tracks on the album with the AlbumID of 1 that are also longer than 3 minutes*



```sql
SELECT
    name
FROM
    tracks
WHERE
    AlbumId = 1 AND
    milliseconds > 3 * 60 * 1000;
```

SQL - Combining Predicates

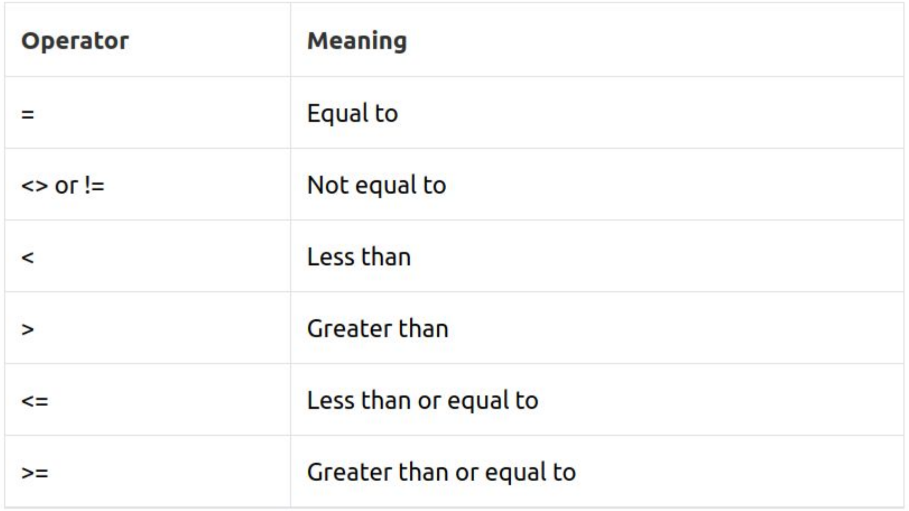
- *Give me the name all the tracks on the album with the AlbumID of 1 OR that are longer than 3 minutes*



```sql
SELECT
    name
FROM
    tracks
WHERE
    AlbumId = 1 OR
    milliseconds > 3 * 60 * 1000;
```

SQL - Comparison Operators

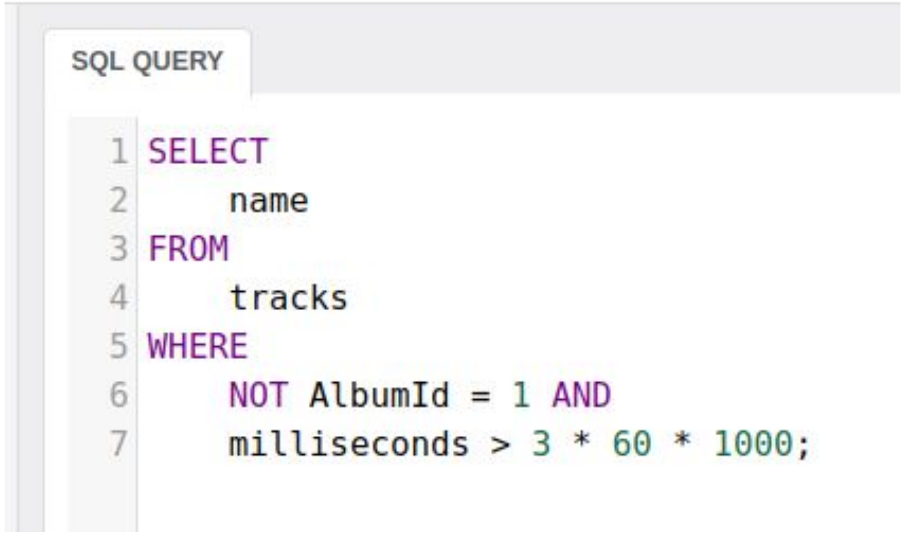
- Most operators will be familiar to you from other programming languages
- The major exceptions:
  - equals (a single = character)
    - double equals usually works too
  - not equals (<>)



Operator	Meaning
=	Equal to
<> or !=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

SQL - Logical Operators

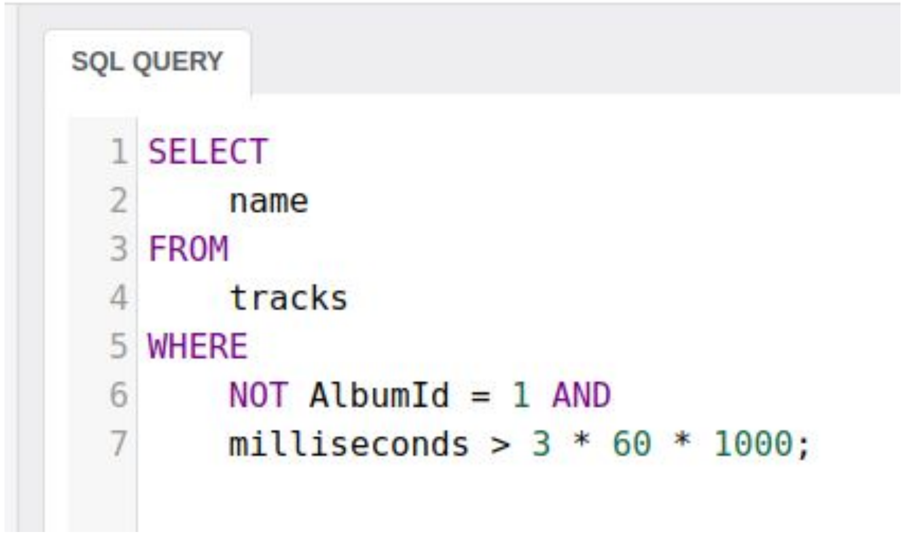
- We have seen AND and OR
- There is also NOT
- What does this query mean?



```sql
SELECT
    name
FROM
    tracks
WHERE
    NOT AlbumId = 1 AND
    milliseconds > 3 * 60 * 1000;
```

SQL - Logical Operators

- Careful with binding!
- *Return the name of all tracks not on album 1 and that are longer than 3 minutes*

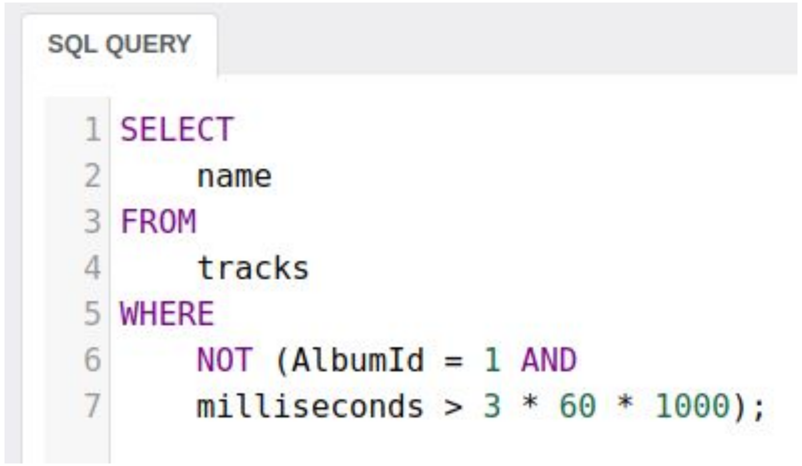


```
SELECT
    name
FROM
    tracks
WHERE
    NOT AlbumId = 1 AND
    milliseconds > 3 * 60 * 1000;
```

SQL - Logical Operators

- Use parentheses to get the right expression

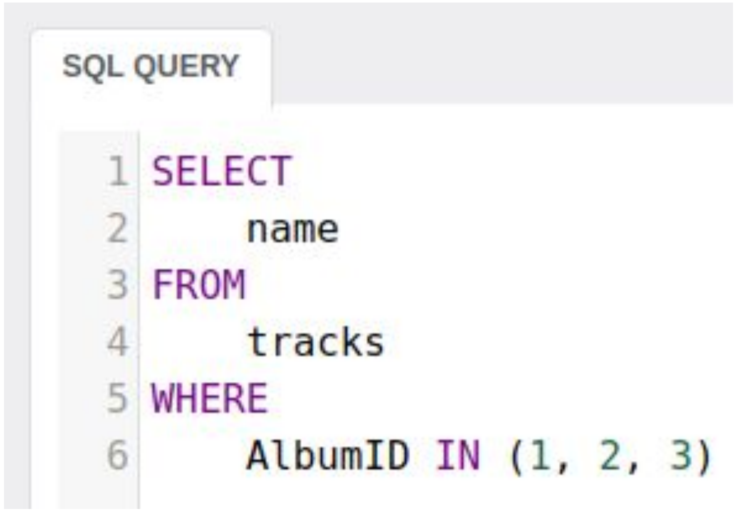
- *Return the name of all tracks not on album 1 and that are not longer than 3 minutes*



```
SELECT
    name
FROM
    tracks
WHERE
    NOT (AlbumId = 1 AND
    milliseconds > 3 * 60 * 1000);
```

SQL - Logical Operators

- The IN operator is extremely useful
- All rows where attribute value falls into a subset
- Can be used with what is called a SubSelect for advanced queries (covered later)

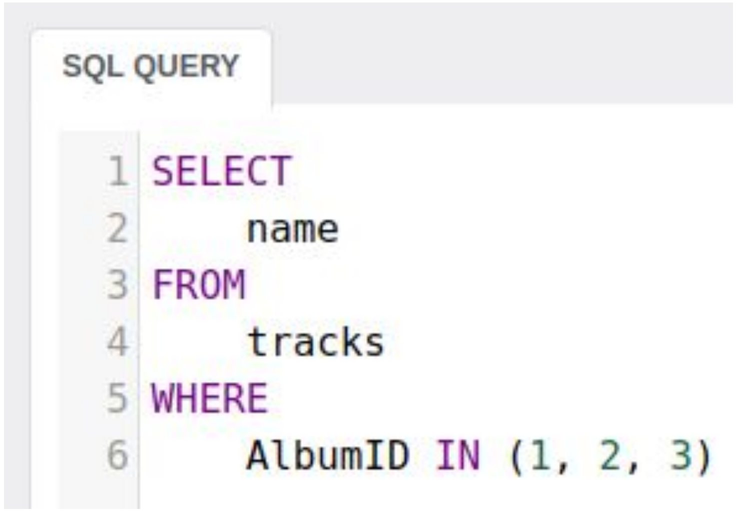


SQL QUERY

```sql
SELECT
    name
FROM
    tracks
WHERE
    AlbumID IN (1, 2, 3)
```

SQL - Logical Operators

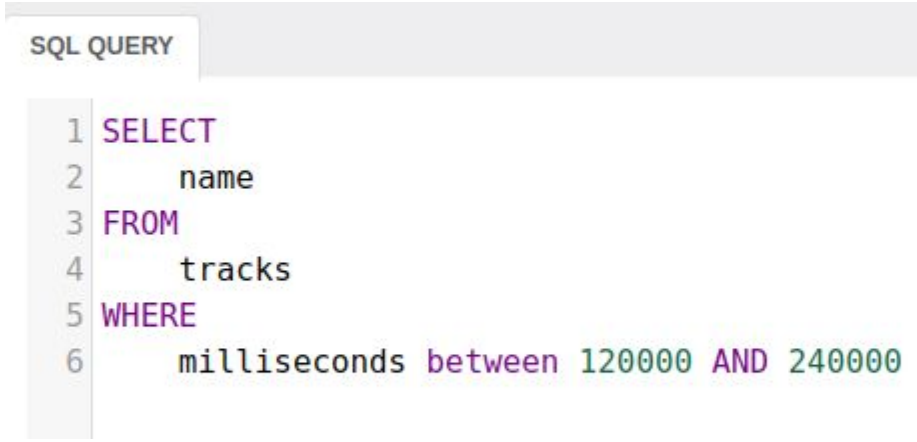
- The IN operator is extremely useful
- All rows where attribute value falls into a subset
- Can be used with what is called a SubSelect for advanced queries (covered later)



```
SELECT
    name
FROM
    tracks
WHERE
    AlbumID IN (1, 2, 3)
```

SQL - Logical Operators

- The BETWEEN operator less widely used
- Can be replaced with two comparison expressions
- Might be clearer in some cases
- *Give me the name of all tracks between 2 and 4 minutes long*

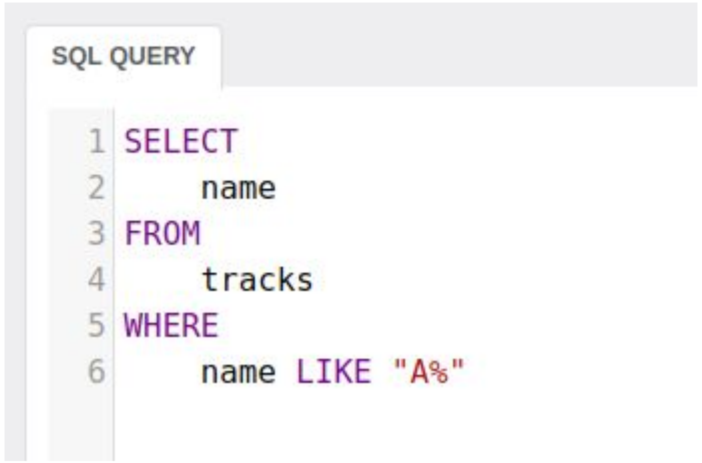


SQL QUERY

```
SELECT
    name
FROM
    tracks
WHERE
    milliseconds between 120000 AND 240000
```

SQL - Logical Operators

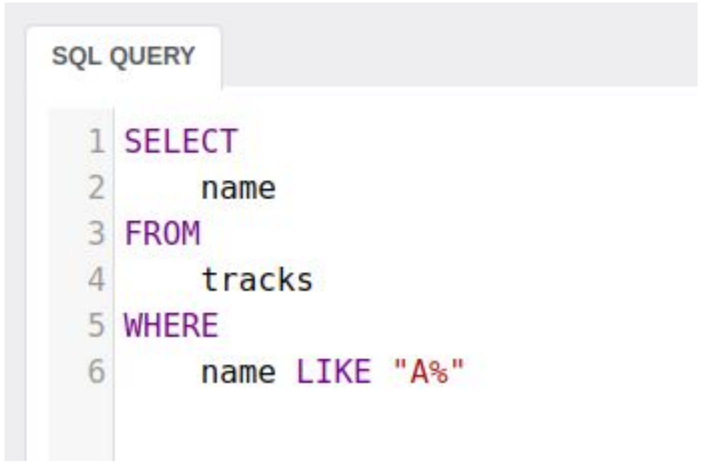
- The LIKE operator can be used for string matching
- Incredibly useful
- Percent (%) is a wildcard
- *Give me the name of all tracks that start with a capital A*



```sql
SELECT
    name
FROM
    tracks
WHERE
    name LIKE "A%"
```

SQL - Logical Operators

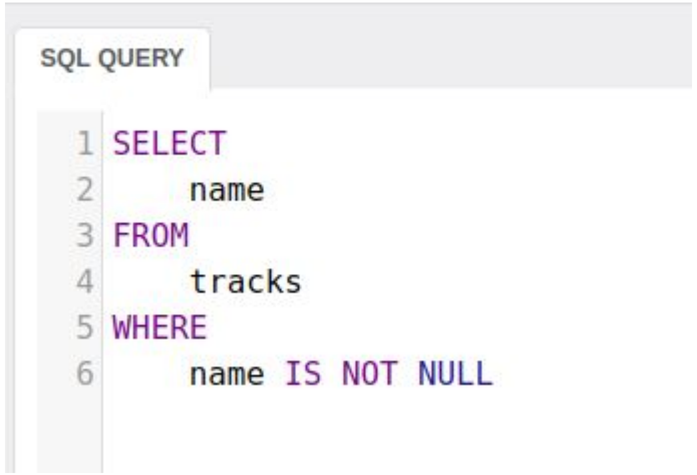
- Incredibly useful…
- Also incredibly expensive
- Difficult to index for in the general cases
- Google doesn't use relational databases for search, for a good reason
- Still, if you don't have google-size data, it's great!



```
SELECT
    name
FROM
    tracks
WHERE
    name LIKE "A%"
```

SQL - Logical Operators

- NULL checks:
  - IS NULL
  - IS NOT NULL
- What does NULL mean?



```sql
SELECT
    name
FROM
    tracks
WHERE
    name IS NOT NULL
```

SQL - The Select Statement

- That’s a lot of stuff, but it’s not too bad I hope
- But you have probably learned about 30% of what you need to be a useful employee in most tech firms
- Next lecture we will discuss sub-queries, which expand on this basic knowledge
- Play around with the IntelliJ
- You can’t hurt the database with a select
  - Except performance ;)

MONTANA
STATE UNIVERSITY