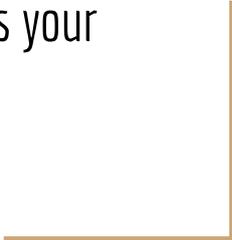




# SQL WHERE clause

Where everybody knows your  
name



# Boolean expressions

In SQL, an expression has a value and a type (or it's `null`, representing no value and no type).

When we want to make a selection of rows according to some criteria, we use a Boolean expression. Such an expression is a claim about the world which is either True or False.

In SQLite3 True is represented by 1 (one) and False is represented by 0 (zero).

This expression is typically used in the “WHERE clause” of an SQL statement.

# Criteria for a SELECT

A simple form of a SELECT statement can be expressed as:

```
SELECT <* or comma separated list of column names>
```

```
FROM <tablename> [WHERE <Boolean expression>];
```

Example:

```
SELECT title, author FROM book WHERE publisher_id = 3;
```

column list: **title, author**

table name: **book**

Boolean expression: **publisher\_id = 3**

# Compound Boolean expressions

The Boolean expression for a WHERE clause can be a simple “predicate” (Boolean expression), but often is a compound expression with logical operators:

```
sqlite> SELECT make, color, license_number FROM cars
.....> WHERE color IN ('Blue', 'Black', 'Brown')
.....>   AND make = 'Dodge'
.....>   AND license_number LIKE 'N%';
Dodge|Brown|NGR 230
Dodge|Blue|NCT 331
Dodge|Brown|NZN 420
Dodge|Brown|NQL 251
Dodge|Black|NEZ 751
```

# Compound Boolean expressions

The basic Boolean operators are:

- AND (both operands need to evaluate to True)
- OR (*at least one* operand needs to evaluate to True)

```
sqlite> SELECT make, color, license_number FROM cars
.....> WHERE make = 'Dodge' AND license_number LIKE 'N%';
sqlite> SELECT make, color, license_number FROM cars
.....> WHERE make = 'Dodge' OR make = 'Volvo';
```

Tests (operands) always need to be a complete Boolean Expression:

```
sqlite> SELECT make, color, license_number FROM cars
.....> WHERE make = 'Dodge' OR 'Volvo';
```

# Boolean column type

You can use the Boolean type for a column:

```
CREATE TABLE author(author_id INTEGER PRIMARY KEY NOT NULL,  
                    name TEXT,  
                    got_nobel_prize BOOLEAN DEFAULT 0 NOT NULL);
```

```
sqlite> SELECT name FROM author WHERE got_nobel_prize;
```

```
Selma Lagerlöf
```

```
sqlite> SELECT name FROM author WHERE NOT got_nobel_prize;
```

```
Henrik and Rikard
```

As you see, you don't need to compare a BOOLEAN value to True or False

In SQLite3, the BOOLEAN type will be treated as NUMERIC  
(0 used for false, 1 used for true)

# Operators which produce Boolean values

The following operators produce a `BOOLEAN` value:

- `=` `<` `>` `<=` `>=` `IS` `<>` `!=`
- `IN`
- `BETWEEN`
- `AND` `OR`
- `NOT`
- `LIKE` `GLOB`
- `CASE-WHEN-THEN-ELSE-END`
- `HAVING`

# Some examples - CASE-WHEN-ELSE-END

```
sqlite> SELECT name ||  
.....> CASE WHEN got_nobel_prize THEN  
.....> ' (winner) '  
.....> ELSE  
.....> ' (loser) '  
.....> END  
.....> FROM author;  
Henrik and Rikard (loser)  
Selma Lagerlöf (winner)
```

The || operator concatenates text (in SQLite).

# Some examples - BETWEEN

```
sqlite> SELECT title FROM book WHERE title BETWEEN 'D' AND 'K';  
Java direkt med Swing  
Databasteknik
```

```
sqlite> SELECT title FROM book WHERE  
...> title BETWEEN 'Databasteknik' AND 'Java direkt med Swing';  
Java direkt med Swing  
Databasteknik
```

```
sqlite> SELECT publisher_id, name FROM publisher WHERE  
...> publisher_id BETWEEN 2 AND 4;  
2|Juneday  
3|Mayday! Mayday!  
4|Oh Really
```

# Some examples - HAVING

```
sqlite> SELECT count(*) AS number_of_titles, name AS publisher FROM book  
NATURAL JOIN publisher GROUP BY publisher;
```

number_of_titles	publisher
1	Juneday
2	Studentlitteratur

```
sqlite> SELECT count(*) AS number_of_titles, name AS publisher FROM book  
NATURAL JOIN publisher GROUP BY publisher HAVING number_of_titles > 1;
```

number_of_titles	publisher
2	Studentlitteratur

You cannot use WHERE on an aggregate value.