SQL
CHEAT SHEET

created by
Tomi Mester

I originally created this cheat sheet for my SQL course and workshop participants.* But I have decided to open-source it and make it available **for everyone who wants to learn SQL**.

It's designed to give you a meaningful structure but also to let you add your own notes (that's why the empty boxes are there). **It starts from the absolute basics** (SELECT * FROM table_name;) and guides you to the intermediate level (JOIN, HAVING, subqueries). **I added everything that you will need as a data analyst/ scientist.**

The ideal use case of this cheat sheet is that you print it in color and keep it next to you while you are learning and practicing SQL on your computer.

Enjoy!

Cheers,
Tomi Mester

*The workshops and courses I mentioned:
Online SQL tutorial (free): data36.com/sql-tutorial
Live SQL workshop: data36.com/sql-workshop
Practice SQL - an online SQL course for practicing: data36.com/practice-sql

BASE QUERY

SELECT * FROM table_name;

This query returns every column and every row of the table called table_name.

SELECT * FROM table_name LIMIT 10;

It returns every column and the first 10 rows from table_name.

SELECTING SPECIFIC COLUMNS

SELECT column1, column2, column3 FROM table_name;

This query returns every row of column1, column2 and column3 from table_name.

[your notes]

DATA TYPES IN SQL

In SQL we have more than 40 different data types. But these seven are the most important ones:

1. **Integer.** A whole number without a fractional part. E.g. 1, 156, 2012412
2. **Decimal.** A number with a fractional part. E.g. 3.14, 3.141592654, 961.1241250
3. **Boolean.** A binary value. It can be either TRUE or FALSE.
4. **Date.** Speaks for itself. You can also choose the format. E.g. 2017-12-31
5. **Time.** You can decide the format of this, as well. E.g. 23:59:59
6. **Timestamp.** The date and the time together. E.g. 2017-12-31 23:59:59
7. **Text.** This is the most general data type. But it can be alphabetical letters only, or a mix of letters and numbers and any other characters. E.g. hello, R2D2, Tomi, 124.56.128.41

FILTERING (the WHERE CLAUSE)

SELECT * FROM table_name WHERE column1 = 'expression';
"Horizontal filtering." This query returns every column from table_name - but only those rows where the value in column1 is 'expression'. Obviously this can be something other than text: a number (integer or decimal), date or any other data format, too.

ADVANCED FILTERING

Comparison operators help you compare two values. (Usually a value that you define in your query and values that exist in your SQL table.) Mostly, they are mathematical symbols, with a few exceptions:

Comparison operator	What does it mean?
=	Equal to
<>	Not equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE '%expression%'	Contains 'expression'
IN ('exp1', 'exp2', 'exp3')	Contains any of 'exp1', 'exp2', or 'exp3'

A few examples:

SELECT * FROM table_name WHERE column1 != 'expression';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression'.

SELECT * FROM table_name WHERE column2 >= 10;

It returns every column from table_name, but only those rows where the value in column2 is greater or equal to 10.

SELECT * FROM table_name WHERE column3 LIKE '%xzy%';

It returns every column from table_name, but only those rows where the value in column3 contains the 'xyz' string.

MULTIPLE CONDITIONS

You can use more than one condition to filter. For that, we have two logical operators: OR, AND.

SELECT * FROM table_name WHERE column1 != 'expression' AND column3 LIKE '%xzy%';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression' AND the value in column3 contains the 'xyz' string.

SELECT * FROM table_name WHERE column1 != 'expression' OR column3 LIKE '%xzy%';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression' OR the value in column3 contains the 'xyz' string.

PROPER FORMATTING

You can use line breaks and indentations for nicer formatting. It won't have any effect on your output. Be careful and put a semicolon at the end of the query though!

```
SELECT *
FROM table_name
WHERE column1 != 'expression'
    AND column3 LIKE '%xzy%'
LIMIT 10;
```

SORTING VALUES

SELECT * FROM table_name ORDER BY column1;
This query returns every row and column from table_name, ordered by column1, in ascending order (by default).

SELECT * FROM table_name ORDER BY column1 DESC;
This query returns every row and column from table_name, ordered by column1, in descending order.

UNIQUE VALUES

SELECT DISTINCT(column1) FROM table_name;
It returns every unique value from column1 from table_name.

CORRECT KEYWORD ORDER

SQL is extremely sensitive to keyword order.
So make sure you keep it right:

1. **SELECT**
2. **FROM**
3. **WHERE**
4. **ORDER BY**
5. **LIMIT**

SQL FUNCTIONS FOR AGGREGATION

In SQL, there are five important aggregate functions for data analysts/scientists:

- **COUNT()**
- **SUM()**
- **AVG()**
- **MIN()**
- **MAX()**

A few examples:
SELECT COUNT(*) FROM table_name WHERE column1 = 'something';
It counts the number of rows in the SQL table in which the value in column1 is 'something'.

SELECT AVG(column1) FROM table_name WHERE column2 > 1000;
It calculates the average (mean) of the values in column1, only including rows in which the value in column2 is greater than 1000.

SQL GROUP BY

The GROUP BY clause is usually used with an aggregate function (COUNT, SUM, AVG, MIN, MAX). It groups the rows by a given column value (specified after GROUP BY) then calculates the aggregate for each group and returns that to the screen.

SELECT column1, COUNT(column2) FROM table_name GROUP BY column1;
This query counts the number of values in column2 - for each group of unique column1 values.

SELECT column1, SUM(column2) FROM table_name GROUP BY column1;
This query sums the number of values in column2 - for each group of unique column1 values.

SELECT column1, MIN(column2) FROM table_name GROUP BY column1;
This query finds the minimum value in column2 - for each group of unique column1 values.

SELECT column1, MAX(column2) FROM table_name GROUP BY column1;
This query finds the maximum value in column2 - for each group of unique column1 values.

SQL ALIASES

You can rename columns, tables, subqueries, anything.

SELECT column1, COUNT(column2) AS number_of_values FROM table_name GROUP BY column1;
This query counts the number of values in column2 - for each group of unique column1 values. Then it renames the COUNT(column2) column to number_of_values.

SQL JOIN

You can JOIN two (or more) SQL tables based on column values.

```
SELECT *
FROM table1
JOIN table2
ON table1.column1 = table2.column1;
```

This joins table1 and table2 values - for every row where the value of column1 from table1 equals the value of column1 from table2.

Detailed explanation here: https://data36.com/sql-join-data-analysis-tutorial-ep5/

SQL HAVING

The execution order of the different SQL keywords doesn't allow you to filter with the WHERE clause on the result of an aggregate function (COUNT, SUM, etc.). This is because WHERE is executed before the aggregate functions. But that's what HAVING is for:

```
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1
HAVING COUNT(column2) > 100;
```

This query counts the number of values in column2 - for each group of unique column1 values. It returns only those results where the counted value is greater than 100.

Detailed explanation and examples here: https://data36.com/sql-data-analysis-advanced-tutorial-ep6/

CORRECT KEYWORD ORDER AGAIN

SQL is extremely sensitive to keyword order.
So make sure you keep it right:

1. SELECT
2. FROM
3. JOIN (ON)
4. WHERE
5. GROUP BY
6. HAVING
7. ORDER BY
8. LIMIT

SUBQUERIES

You can run SQL queries within SQL queries. (Called subqueries.) Even queries within queries within queries. The point is to use the result of one query as an input value of another query.
Example:

```
SELECT COUNT(*) FROM
  (SELECT column1, COUNT(column2) AS inner_number_of_values
  FROM table_name
  GROUP BY column1) AS inner_query
WHERE inner_number_of_values > 100;
```

The inner query counts the number of values in column2 - for each group of unique column1 values. Then the outer query uses the inner query's results and counts the number of values where inner_number_of_values are greater than 100. (The result is one number.)

Detailed explanation here: https://data36.com/sql-data-analysis-advanced-tutorial-ep6/

CREATED BY

Tomi Mester from Data36.com

Tomi Mester is a data analyst and researcher. He worked for Prezi, iZettle and several smaller companies as an analyst/consultant. He's the author of the Data36 blog where he writes posts and tutorials on a weekly basis about data science, AB-testing, online research and data coding. He's an O'Reilly author and presenter at TEDxYouth, Barcelona E-commerce Summit, Stockholm Analyticsdagarna and more.

WHERE TO GO NEXT

Find company workshops, online tutorials and online video courses on my website: https://data36.com

Subscribe to my Newsletter list for useful stuff like this:
https://data36.com/newsletter

Online SQL tutorial (free): data36.com/sql-tutorial
Live SQL workshop: data36.com/sql-workshop
Practice SQL - an online SQL course for practicing: data36.com/practice-sql

Table of Contents

SQLite Statements

These SQL Statements are organized by their CRUD function on the table or database - Create, Read, Update, or Delete.

CREATE

CREATE a database

`sqlite3 <database_name>.db`	This statement starts the sqlite3 program with the database file specified open. If the file doesn't exist, a new database file with the specified name is automatically created. If no database file is given, a temporary database is created and deleted when the sqlite3 program closes. **Note** this is a SQLite program statement to open the program (different from SQL commands)	`sqlite3 shelter.db`

CREATE a table

`CREATE TABLE <table_name>(`
`<column_name_1> <data_type_1>,`
`<column_name_2> <data_type_2>,`
`...);`
	Create a table with the specified name containing column names of the specified data types.	`CREATE TABLE pets (`
		` _id INTEGER,`
		` name TEXT,`
		` breed TEXT,`
		` gender INTEGER,`
		` weight INTEGER);`

INSERT data in a table

`INSERT INTO <table_name>(`
` <column_name_1>,`
` <column_name_2>,`
` ...)`
`VALUES (`
` <values_1>,`
` <values_2>,`
` ...);`
	Insert into a specific table the listed values at the corresponding column names.	`INSERT INTO pets (`
		`  _id,`
		`  name,`
		`  breed,`
		`  gender,`
		`  weight)`
		`VALUES (`
		`  1,`
		`  "Tommy",`
		`  "Pomeranian",`
		`  1,`
		`  4);`

READ

SELECT data from a table

`SELECT <columns>` `FROM <table_name>;`	Select specific column(s) from a table.	`SELECT name, breed from pets;`
`SELECT * FROM <table_name>;`	Select all columns and all rows from a specific table. (Asterisk here means "all columns and all rows").	`SELECT * FROM pets;`

UPDATE

UPDATE data in a table

`UPDATE <table_name>`
`SET <column_name> = <value>`
`WHERE <condition>;`
	Update information in an existing row in a table.	`UPDATE pets`
		`SET weight = 18`
		`WHERE _id = 5;`

DELETE

DELETE data from a table

`DELETE FROM <table_name> WHERE <condition>;`	Delete data from a table that meet the conditions of the WHERE clause.	`DELETE FROM pets WHERE _id = 1;`

	Different from DROP TABLE because the table definition still remains.	
DROP TABLE		
`DROP TABLE <table_name>;`	Remove a table definition and all its data.	`DROP TABLE pets;`

SQLite Keywords

These SQLite keywords are to be used in conjunction with SQL commands.

PRIMARY KEY		

`CREATE TABLE <table_name> (`
` <column_1> <data_type_1>`
` **PRIMARY KEY**,`
` <column_2> <data_type_2>,`
` ...);`
	Ensure uniqueness. There can only be one primary key per table.	`CREATE TABLE headphones (`
		` _id INTEGER **PRIMARY KEY**,`
		` name TEXT,`
		` price INTEGER,`
		` style INTEGER,`
		` in_stock INTEGER,`
		` description TEXT);`

AUTOINCREMENT		

`CREATE TABLE <table_name> (`
` <column_1> <data_type_1>`
` **AUTOINCREMENT**,`
` <column_2> <data_type_2>,`
` ...);`
	Automatically calculate new integer when row is added. Useful for IDs.	`CREATE TABLE headphones (`
		` _id INTEGER PRIMARY KEY`
		`  **AUTOINCREMENT**,`
		`name TEXT,`
		`price INTEGER,`
		`style INTEGER,`
		`in_stock INTEGER,`
		`description TEXT);`

NOT NULL		

`CREATE TABLE <table_name> (`
` <column_1> <data_type_1>`
`**NOT NULL**,`
` <column_2> <data_type_2>,`
` ...);`
	When a value is inserted into the table, it MUST have a value associated with it.	`CREATE TABLE headphones (`
		` _id INTEGER PRIMARY KEY`
		`  AUTOINCREMENT,`
		` name TEXT **NOT NULL**,`
		` price INTEGER,`
		` style INTEGER,`
		` in_stock INTEGER,`
		` description TEXT);`

DEFAULT <value>		

`CREATE TABLE <table_name> ( <column_1> <data_type_1> DEFAULT <value>, <column_2> <data_type_2>, ...);`	When inserting a new row, if no value is provided, the default value will be used.	`CREATE TABLE headphones ( _id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, price INTEGER, style INTEGER, in_stock INTEGER NOT NULL DEFAULT 0, description TEXT);`
WHERE clause		

Some examples:
 	
`SELECT * FROM pets WHERE <condition>;`
 	
`UPDATE <table_name> SET <column_name> = <value> WHERE <condition>;`
 	
`DELETE FROM <table_name> WHERE <condition>;`
	The WHERE clause ensures that only rows that meet the specified criteria are affected. It can be used in conjunction with SELECT, INSERT, UPDATE, or DELETE statements.	`SELECT * FROM pets WHERE _id = 1;`
		
		`SELECT * FROM pets WHERE weight >= 15;`
		
		`SELECT name, gender FROM pets WHERE breed != "Breed Unknown";`
		
		`DELETE FROM pets WHERE _id = <id_of_pet_to_delete>;`

ORDER BY clause		

`SELECT <column_name> FROM <table_name> ORDER BY <column_name> <ASC|DESC>;`	Sort the data in either ascending (ASC) or descending (DESC) order based on the column(s) listed.	`SELECT * FROM pets ORDER BY name ASC;`
		
		`SELECT weight FROM pets ORDER BY name DESC;`

SQLite Program Dot Commands

These dot commands are specific to the Sqlite Version 3 program(a database library) to be used in the command prompt/terminal. Don't confuse them with Structured Query Language (SQL) commands.
To see a full list of dot commands, check here.

`.header <on|off>`	Turn display headers on or off
`.help`	Display the help menu listing dot commands
`.mode <mode>`	Set the output mode to one of these options - ascii, csv, column, html, insert, line, list, tabs, tcl

`.open <filename>`	Close the existing database and open the file name given
`.quit`	Exit the program
`.schema <table_name>`	Show the CREATE statement used to generate the table listed
`.tables`	List names of tables

This is used as part of the Udacity Android Basics Nanodegree by Google.

Code samples and descriptions are licensed under the Apache 2.0 License.
All other content of this page is licensed under the Creative Commons Attribution 3.0 License.

SQL Facts

- SQL stands for Structured Query Language
- SQL is pronounced "sequel"
- SQL is declarative language
- SQL is used to access & manipulate data in databases
- Top SQL DBs are MS SQL Server, Oracle, DB2, and MySQL

Database Definitions

- **RDBMS** (Relational Database Management System) – Software that stores and manipulates data arranged in relational database tables.
- **Table** – A set of data arranged in columns and rows. The columns represent characteristics of stored data and the rows represent actual data entries.

SQL Commands Categories

Data Query Language (DQL)

- SELECT - Retrieve data from table(s)

Data Manipulation Language (DML)

- INSERT - Insert data into db table
- UPDATE - Update data in db table
- DELETE - Delete data from table

Data Definition Language (DDL)

- CREATE - Create db object (table, view, etc.)
- ALTER - Modify db object (table, view, etc.)
- DROP - Delete db object (table, view, etc.)

Data Control Language (DCL)

- GRANT - Assign privilege
- REVOKE - remove privilege

How to select data from a table

```
SELECT <Column List>
FROM <Table Name>
WHERE <Search Condition>
```

Example:

```
SELECT FirstName, LastName, OrderDate
FROM Orders WHERE OrderDate > '10/10/2010'
```

How to insert data in a table

```
INSERT INTO <Table Name>
(<Column List>) VALUES (<Values>)
```

Example:

```
INSERT INTO Orders
(FirstName, LastName, OrderDate) VALUES
('John', 'Smith', '10/10/2010')
```

How to update data in a table

```
UPDATE <Table Name>
SET <Column1> = <Value1>, <Column2> = <Value2>, …
WHERE <Search Condition>
```

Example:

```
UPDATE Orders
SET FirstName = 'John', LastName = 'Who' WHERE LastName='Wo'
```

How to delete data from a table

```
DELETE FROM <Table Name>
WHERE <Search Condition>
```

Example:

```
DELETE FROM Orders
WHERE OrderDate < '10/10/2010'
```

How to group data and use aggregates

```
SELECT <Column List>, <Aggregate Function>(<Column Name>)
FROM <Table Name>
WHERE <Search Condition>
GROUP BY <Column List>
```

Example:

```
SELECT LastName, SUM(OrderValue)
FROM Orders
WHERE OrderDate > '10/10/2010'
GROUP BY LastName
```

How to order data

```
SELECT <Column List>
FROM <Table Name>
WHERE <Search Condition>
ORDER BY <Column List>
```

Example:

```
SELECT FirstName, LastName, OrderDate
FROM Orders
WHERE OrderDate > '10/10/2010'
ORDER BY OrderDate
```

How to select data from more than one table

```
SELECT <Column List>
FROM <Table1> JOIN <Table2>
ON <Table1>.<Column1> = <Table2>.<Column1>
```

Example:

```
SELECT Orders.LastName, Countries.CountryName
FROM Orders JOIN Countries ON
Orders.CountryID = Countries.ID
```

Using UNION

```
SELECT <Column List> FROM <Table1>
UNION
SELECT <Column List> FROM <Table2>
```

Example:

```
SELECT FirstName, LastName FROM Orders2010
UNION
SELECT FirstName, LastName FROM Orders2011
```

CREATE TABLE

```
CREATE TABLE <Table Name>
(   Column1 DataType,
    Column2 DataType,
    Column3 DataType,
    ....    )
```

```
CREATE TABLE Orders
(   FirstName CHAR(100),
    LastName CHAR(100),
    OrderDate DATE,
    OrderValue Currency    )
```

© http://www.SQL-Tutorial.net

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t ;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger