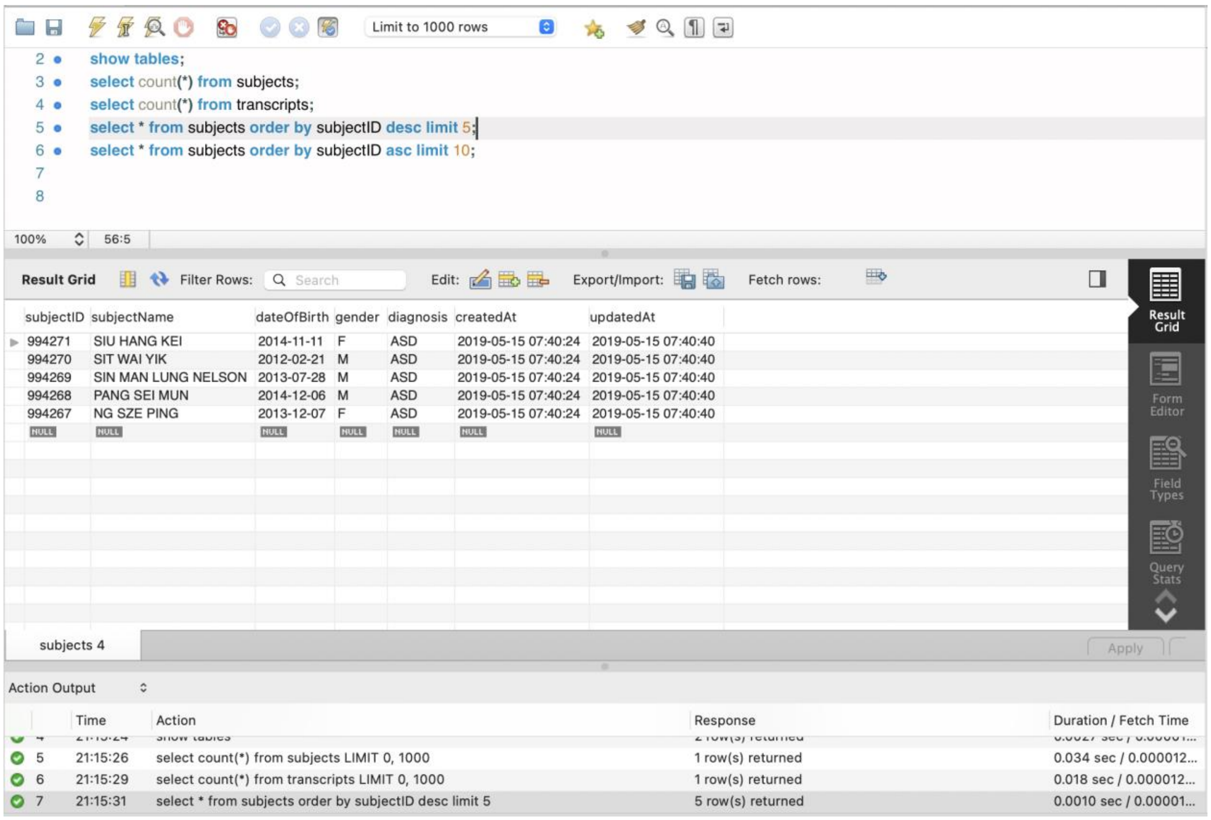
SQL Cheat Sheet

What Is SQL?

SQL is short for **Structured Query Language**. Its chief function is managing structured data on a relational database management system (RDBMS), usually arranged in tables. SQL is case-insensitive, but it's common to capitalize SQL keywords such as `SELECT` and `FROM`.

Suppose you want to execute multiple SQL statements in the same server call. In that case, some database administration tools, such as MySQL Workbench, require a semicolon (`;`) at the end of each SQL statement to separate them.



```
show tables;
select count(*) from subjects;
select count(*) from transcripts;
select * from subjects order by subjectID desc limit 5;
select * from subjects order by subjectID asc limit 10;
```

subjectID	subjectName	dateOfBirth	gender	diagnosis	createdAt	updatedAt
994271	SIU HANG KEI	2014-11-11	F	ASD	2019-05-15 07:40:24	2019-05-15 07:40:40
994270	SIT WAI YIK	2012-02-21	M	ASD	2019-05-15 07:40:24	2019-05-15 07:40:40
994269	SIN MAN LUNG NELSON	2013-07-28	M	ASD	2019-05-15 07:40:24	2019-05-15 07:40:40
994268	PANG SEI MUN	2014-12-06	M	ASD	2019-05-15 07:40:24	2019-05-15 07:40:40
994267	NG SZE PING	2013-12-07	F	ASD	2019-05-15 07:40:24	2019-05-15 07:40:40
NULL	NULL	NULL	NULL	NULL	NULL	NULL

	Time	Action	Response	Duration / Fetch Time
4	21:15:24	show tables	2 row(s) returned	0.0027 sec / 0.00001...
5	21:15:26	select count(*) from subjects LIMIT 0, 1000	1 row(s) returned	0.034 sec / 0.000012...
6	21:15:29	select count(*) from transcripts LIMIT 0, 1000	1 row(s) returned	0.018 sec / 0.000012...
7	21:15:31	select * from subjects order by subjectID desc limit 5	5 row(s) returned	0.0010 sec / 0.00001...

Screenshot of MySQL Workbench in action

Basic SQL Syntax

This section is the essential SQL syntax cheat sheet. If you're short on time, read this section first.

Command	Syntax	Description
`ALTER TABLE`	`ALTER TABLE table_name ADD column_name datatype;`	Add columns of a specified `datatype` to a table in a database
`AS`	`SELECT column_name AS 'Alias' FROM table_name;`	A keyword in SQL to rename a column or table using an alias name
`CASE`	`SELECT column_name, CASE WHEN condition THEN 'Result_1' WHEN condition THEN 'Result_2' ELSE 'Result_3' END FROM table_name;`	Create different outputs inside a `SELECT` statement
`CREATE TABLE`	`CREATE TABLE table_name (column_1 datatype, column_2 datatype, column_3 datatype);`	Create a new table in a database and specify the name of the table and columns of a specified `datatype` inside it

DELETE	DELETE FROM table_name WHERE some_column = some_value;	Remove the rows from a table
HAVING	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name HAVING COUNT(*) > value;	Use it like the WHERE keyword in aggregating functions such as GROUP BY
INSERT	INSERT INTO table_name (column_1, column_2, column_3) VALUES (value_1, 'value_2', value_3);	Add new rows to a table with specified values
SELECT	SELECT column_name FROM table_name;	Fetch data from a database; the column_name can be a function applied to an existing column
SELECT DISTINCT	SELECT DISTINCT column_name FROM table_name;	Return unique, non-repeating values in specified columns
UPDATE	UPDATE table_name SET some_column = some_value WHERE some_column = some_value;	Edit rows in a table
WITH	WITH temporary_name AS (SELECT * FROM table_name) SELECT * FROM temporary_name WHERE column_name operator value;	Process the result of a query (SELECT * FROM table_name) stored in a temporary table referenced by the alias temporary_name
/* */ --	/* multi-line comment explaining the code */ --single-line comment	Enclose comments: ● For comments spanning several lines: /* */ ● For comments on the same line as the command: --

Data Types in SQL

The data type of a SQL column identifies how SQL will interact with the stored data. SQL is a strongly typed language, so it’s important to tell apart various data types.

Strongly Typed Languages vs Weakly Typed Languages

In computer programming, a programming language is strongly typed if it demands the specification of data types.

In strongly typed languages, once a type is assigned to a variable at runtime or compile time, it retains that type and can't be intermingled in expressions with other types easily. You cannot assign an integer to a string variable in a strongly typed language. Boolean variables can only hold Boolean values, and writing any other value to it may throw errors.

In weakly typed languages, once a type is assigned to a variable at runtime or compile time, it can be intermingled in expressions with other types easily. Here, an integer assigned to a string variable may get converted into the character(s) representing the integer. You can also assign a string or integer to a variable previously used to hold a Boolean value.

The same name may map to different data types in other SQL implementations. Therefore, **always consult the relevant documentation** (MySQL, PostgreSQL).

MySQL Data Types (Version 8.0)

MySQL has three main data types: string, numeric, and date and time:

String

Data type	Description
`CHAR(size)`	A **fixed**-length string: can contain letters, numbers, and special characters. The `size` parameter specifies the column length in characters, from 0 to 255. The default is 1.
`VARCHAR(size)`	A **variable**-length string: can contain letters, numbers, and special characters. The `size` parameter specifies the maximum string length in characters, from 0 to 65535.
`BINARY(size)`	Equal to `CHAR()` but stores binary byte strings. The `size` parameter specifies the column length in bytes. The default is 1.
`VARBINARY(size)`	Equal to `VARCHAR()` but stores binary byte strings. The `size` parameter specifies the maximum column length in bytes.
`TINYBLOB`	For BLOBs (Binary Large Objects). Max length: 255 bytes.
`TINYTEXT`	Hold a string with a maximum length of 255 characters.
`TEXT(size)`	Hold a string with a maximum length of 65,535 bytes.
`BLOB(size)`	For BLOBs (Binary Large Objects). Hold up to 65,535 bytes of data.
`MEDIUMTEXT`	Hold a string with a maximum length of 16,777,215 characters.
`MEDIUMBLOB`	For BLOBs (Binary Large Objects). Hold up to 16,777,215 bytes of data.

`LONGTEXT`	Hold a string with a maximum length of 4,294,967,295 characters.
`LONGBLOB`	For BLOBs (Binary Large Objects). Hold up to 4,294,967,295 bytes of data.
`ENUM(val1, val2, val3, ...)`	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If you insert a value that is not in the list, you insert a blank value. SQL sorts the values in the order you enter them.
`SET(val1, val2, val3, ...)`	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list.

Numeric

We leave the "Alias" field blank if a data type has no alias.

Data type	Alias	Description
`BIT(size)`		A bit-value type. The `size` parameter specifies the number of bits per value and can hold a value from 1 to 64. The default value for size is 1.
`TINYINT(size)`		A tiny integer. The signed is from -128 to 127. The unsigned range is from 0 to 255. The `size` parameter specifies the maximum display width (which is 255).
`BOOLEAN`	`BOOL`	Zero = false, nonzero values = true.
`SMALLINT(size)`		A small integer. The signed range is from -32768 to 32767. The unsigned range is from 0 to 65535. The `size` parameter specifies the maximum display width (which is 255).
`MEDIUMINT(size)`		A medium-sized integer. The signed range is from -8388608 to 8388607. The unsigned range is from 0 to 16777215. The `size` parameter specifies the maximum display width (which is 255).
`INTEGER(size)`	`INT(size)`	A medium-sized integer. The signed range is from -2147483648 to 2147483647. The unsigned range is from 0 to 4294967295. The `size` parameter specifies the maximum display width (which is 255).
`BIGINT(size)`		A large integer. The signed range is from -9223372036854775808 to 9223372036854775807. The unsigned range is from 0 to 18446744073709551615. The `size` parameter specifies the maximum display width (which is 255).

`FLOAT(size, d)`, `DOUBLE(size, d)`, `DOUBLE PRECISION(size, d)`		Floating point number. The parameter `size` specifies the total number of digits. The `d` parameter sets the number of digits. Future MySQL versions (beyond MySQL 8.0.17) will remove this syntax.
`FLOAT(p)`		Floating point number. MySQL uses the `p` value to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If p is from 0 to 24, the data type becomes `FLOAT()`. If p is from 25 to 53, the data type becomes `DOUBLE()`.
`DECIMAL(size, d)`	`DEC(size, d)`	Fixed-point number. The parameter `size` specifies the total number of digits. The `d` parameter sets the number of digits after the decimal point. The maximum number for `size` is 65. The maximum number for `d` is 30. The default value for size is 10. The default value for `d` is 0.

Note: All the numeric data types may have an extra option: `UNSIGNED` or `ZEROFILL`. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the `ZEROFILL` option, MySQL automatically adds the `UNSIGNED` attribute to the column.

Date and time

Adding `DEFAULT` and `ON UPDATE` in the column definition helps you get automatic initialization and updating to the current date and time.

Below, the `fsp` (fractional seconds precision, in microseconds) value must be 0–6. For example, set `fsp` to `1` to encapsulate 0.1–0.9 seconds and `2` for 0.01–0.99 seconds. A value of 0 indicates the absence of a fractional part. If omitted, the default precision is 0.

Data type	**Description**
`DATE`	Date. Format: `YYYY-MM-DD`. The supported range is from '`1000-01-01`' to `'9999-12-31`'
`DATETIME(fsp)`	Date and time combination. Format: `YYYY-MM-DD hh:mm:ss`.
`TIMESTAMP(fsp)`	Timestamp. MySQL stores `TIMESTAMP` values as the number of seconds since the Unix epoch ('`1970-01-01 00:00:00`' UTC). Format: `YYYY-MM-DD hh:mm:ss`.
`TIME(fsp)`	Time. Format: `hh:mm:ss`.
`YEAR`	Year in four-digit format. MySQL 8.0 does not support a two-digit year format.

PostGreSQL Data Types (Version 15)

We leave the “Aliases” field blank if a data type has no alias.

Data type	Aliases	Description
`BIGINT`	`INT8`	Signed eight-byte integer
`BIGSERIAL`	`SERIAL8`	Auto-incrementing eight-byte integer
`BIT [ (n) ]`		Fixed-length bit string
`BIT VARYING [ (n) ]`	`VARBIT [ (n) ]`	Variable-length bit string
`BOOLEAN`	`BOOL`	Logical Boolean (true/false)
`BOX`		Rectangular box on a plane
`BYTEA`		Binary data (“byte array”)
`CHARACTER [ (n) ]`	`CHAR [ (n) ]`	Fixed-length character string
`CHARACTER VARYING [ (n) ]`	`VARCHAR [ (n) ]`	Variable-length character string
`CIDR`		IPv4 or IPv6 network address
`CIRCLE`		Circle on a plane
`DATE`		Calendar date (year, month, day)
`DOUBLE PRECISION`	`FLOAT8`	Double precision floating-point number (eight bytes)
`INET`		IPv4 or IPv6 host address
`INTEGER`	`INT, INT4`	Signed four-byte integer
`INTERVAL [ fields ] [ (p) ]`		Time span
`JSON`		Textual JSON data
`JSONB`		Binary JSON data, decomposed
`LINE`		Infinite line on a plane
`LSEG`		Line segment on a plane
`MACADDR`		MAC (Media Access Control) address
`MACADDR8`		MAC (Media Access Control) address (EUI-64 format)
`MONEY`		Currency amount
`NUMERIC [ (p, s) ]`	`DECIMAL [ (p, s) ]`	Exact numeric of selectable precision
`PATH`		Geometric path on a plane

PG_LSN		PostgreSQL Log Sequence Number
PG_SNAPSHOT		User-level transaction ID snapshot
POINT		Geometric point on a plane
POLYGON		Closed geometric path on a plane
REAL	FLOAT4	Single precision floating-point number (four bytes)
SMALLINT	INT2	Signed two-byte integer
SMALLSERIAL	SERIAL2	Auto-incrementing two-byte integer
SERIAL	SERIAL4	Auto-incrementing four-byte integer
TEXT		Variable-length character string
TIME [(p)] [without time zone]		Time of day (no time zone)
TIME [(p)] WITH TIME ZONE	TIMETZ	Time of day, including time zone
TIMESTAMP [(p)] [without time zone]		Date and time (no time zone)
TIMESTAMP [(p)] WITH TIME ZONE	TIMESTAMPTZ	Date and time, including time zone
TSQUERY		Text search query
TSVECTOR		Text search document
UUID		Universally unique identifier
XML		XML data

SQL Operators

This subsection is a basic SQL operators cheat sheet, where you learn to create complex Boolean expressions in SQL queries.

Command	Syntax	Description
AND	SELECT column_name(s) FROM table_name WHERE column_1 = value_1	Combine two conditions

	`AND column_2 = value_2;`	
`BETWEEN`	`SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value_1 AND value_2;`	Filter the result within a certain range
`IS NULL`	`SELECT column_name(s) FROM table_name WHERE column_name IS NULL;`	Check for empty values in conjunction with the `WHERE` clause
`IS NOT NULL`	`SELECT column_name(s) FROM table_name WHERE column_name IS NOT NULL;`	Check for the absence of empty values in conjunction with the `WHERE` clause
`LIKE`	`SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;`	Search for a specific pattern in a column in conjunction with the `WHERE` clause
`OR`	`SELECT column_name FROM table_name WHERE column_name = value_1 OR column_name = value_2;`	Filter the result set to contain only the rows where either condition is `TRUE`
`UNION`	`SELECT column_name(s) FROM table1 UNION SELECT column_name(s) FROM table2;`	Combine the results of two or more `SELECT` statements and select only distinct values
`UNION ALL`	`SELECT column_name(s) FROM table1 UNION ALL SELECT column_name(s) FROM table2;`	Combine the results of two or more `SELECT` statements, allowing duplicate values

SQL Functions

SQL functions help you compute and analyze the contents of database tables.

Here are some common SQL functions:

Command	Syntax	Description
`AVG()`	`SELECT AVG(column_name) FROM table_name;`	Aggregate a numeric column and return its arithmetic mean, ignoring `NULL` values
`CASE()`	`CASE WHEN condition1 THEN result1 WHEN condition2 THEN result2 WHEN conditionN THEN resultN ELSE result END;`	The `CASE` expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). Once a condition is true, it will stop reading and return the result. If no conditions

		are `TRUE`, it returns the value in the `ELSE` clause. Without an `ELSE` part and with all conditions `FALSE`, it returns `NULL`.
`CAST()`	`SELECT CAST(value AS datatype);`	Convert a `value` (of any type) into the specified `datatype`.
`CHAR_LENGTH()`	`SELECT CHAR_LENGTH(string) AS LengthOfString;`	(MySQL) Return the length of a `string` in characters
`COALESCE()`	`SELECT COALESCE([list of values including NULL separated by commas]);`	Return the first non-null value in a list
`COUNT()`	`SELECT COUNT(column_name) FROM table_name;`	Take the name of a column as an argument and count the number of rows when the column is not `NULL`
`FIRST()`	`SELECT FIRST(column_name) FROM table_name;`	Return the first value of the selected column
`LAST()`	`SELECT LAST(column_name) FROM table_name;`	Return the last value of the selected column
`LCASE()`	`SELECT LCASE(column_name) FROM table_name;`	Convert string values in the selected column to lowercase
`LEN()`	`SELECT LEN(string);`	(SQL Server) Return the length of a `string`
`MAX()`	`SELECT MAX(column_name) FROM table_name;`	Take at least one column as an argument and return the largest value among them
`MIN()`	`SELECT MIN(column_name) FROM table_name;`	Take at least one column as an argument and return the smallest value among them
`NULLIF()`	`SELECT NULLIF(expr1, expr2);`	Return `NULL` if two expressions `expr1`, `expr2` are equal. Otherwise, it returns the first expression.
`ROUND()`	`SELECT ROUND(column_name, integer) FROM table_name;`	Take the column name and an integer as an argument, and round the values in a column to the number of decimal places specified by an integer
`SUBSTRING()`	`SELECT SUBSTRING(string, start, length) AS ExtractString;`	Extract some characters from a string, where `start` is the starting position (one-indexed) and `length` is the

		number of characters to extract. Aliases: `MID()`, `SUBSTR()`
`SUM()`	`SELECT SUM(column_name) FROM table_name;`	Return the sum of values from a particular column
`UCASE()`	`SELECT UCASE(column_name) FROM table_name;`	Convert string values in the selected column to uppercase
`VAR()`	`SELECT VAR(column_name) FROM table_name;`	Return the statistical variance

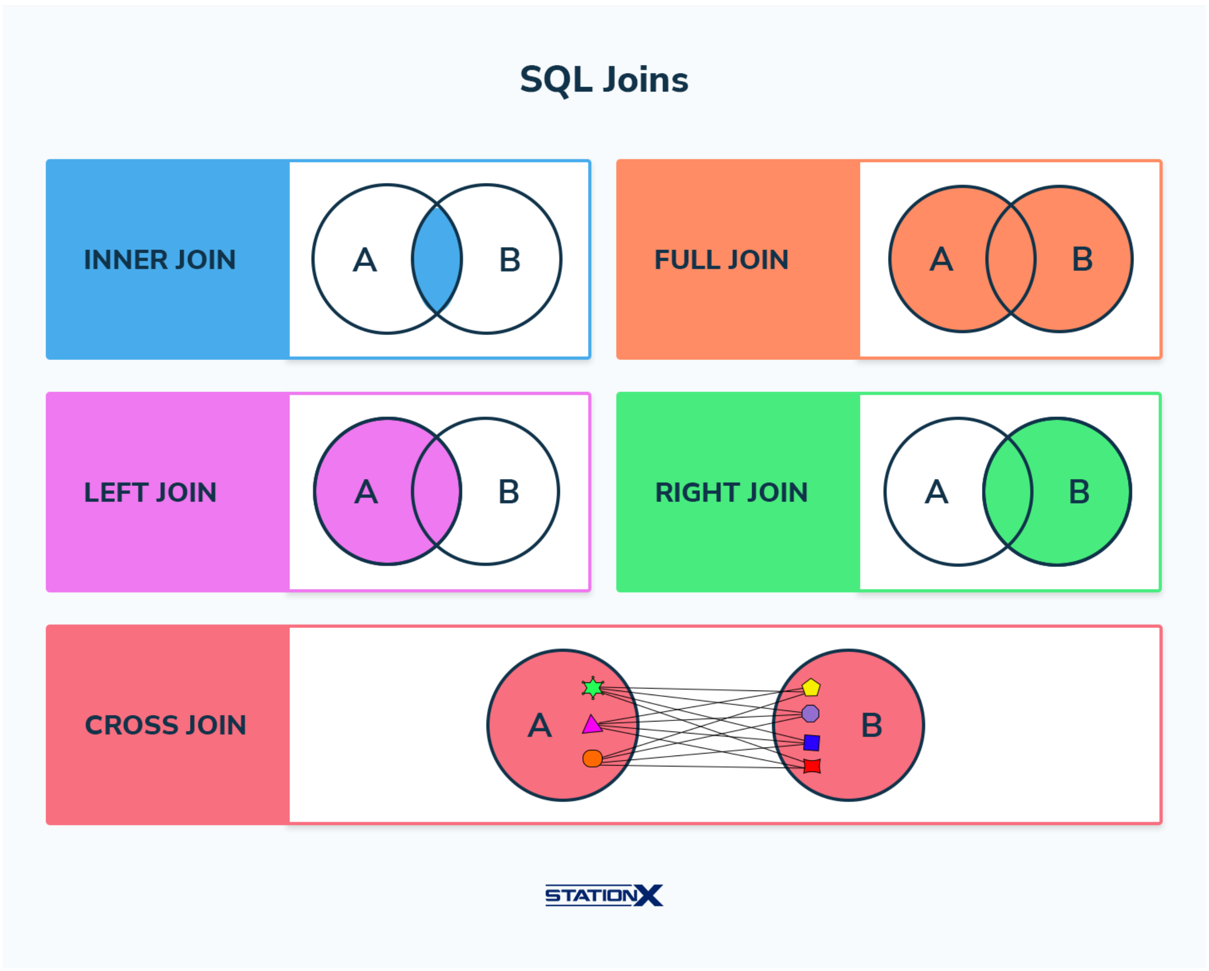
SQL Clauses

A SQL clause presents the results of a SQL query in a way you specify.

Command	Syntax	Description
`LIMIT`	`SELECT column_name(s) FROM table_name LIMIT number;`	Specify the maximum number of rows the result set must have. Some SQL implementations have `SELECT TOP` playing a similar role.
`GROUP BY`	`SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;`	Used for aggregate functions in collaboration with the `SELECT` statement
`ORDER BY`	`SELECT column_name FROM table_name ORDER BY column_name ASC | DESC;`	Sort the result set by a particular column either numerically or alphabetically. `ASC` means “in ascending order;” `DESC`, “descending.”
`WHERE`	`SELECT column_name(s) FROM table_name WHERE column_name operator value;`	Filter the result set to include the rows where the condition is `TRUE`

SQL Joins

Combining two tables in SQL is easy:



Command	Syntax	Description
`INNER JOIN`	`SELECT column_name(s) FROM table_1 JOIN table_2 ON table_1.column_name = table_2.column_name;`	Select records that have matching values in both tables
`LEFT JOIN`	`SELECT column_name(s) FROM table_1 LEFT JOIN table_2 ON table_1.column_name = table_2.column_name;`	Combine all records from the left side and any matching rows from the right table. `LEFT OUTER JOIN` and `LEFT JOIN` are the same.
`RIGHT JOIN`	`SELECT column_name(s) FROM table_1 RIGHT JOIN table_2 ON table_1.column_name = table_2.column_name;`	Combine all rows from the right side and any matching rows from the left table. `RIGHT OUTER JOIN` and `RIGHT JOIN` are the same.
`FULL JOIN`	`SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name =`	Return all records whether the records in the left (`table1`) and right (`table2`) tables match.

	`table2.column_name WHERE condition;`	`FULL OUTER JOIN` and `FULL JOIN` are the same.
`CROSS JOIN`	`SELECT * FROM table1 CROSS JOIN table2`	Combine each row of the first table (`table1`) with each row of the second table (`table2`).

SQL Views

In SQL, a view is a virtual table based on the results of an SQL query. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from a single table.

Here are the most important functions for manipulating SQL views:

Command	Syntax	Description
`CREATE VIEW`	`CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	Create a view from the SQL query beginning with `SELECT`
`CREATE OR REPLACE VIEW`	`CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	Update a view created from the SQL query beginning with `SELECT`
`DROP VIEW`	`DROP VIEW view_name;`	Delete a view

SQL Indexes

Indexes are for speeding up data retrieval from a database. The users cannot see the indexes. Updating a table with indexes takes longer than updating a table without (because the indexes also need an update). So, only create indexes on the columns against which users frequently search.

Command	Syntax	Description
`CREATE INDEX`	`CREATE INDEX index_name ON table_name (column1, column2, ...);`	Creates a unique index on a table, allowing duplicate values
`CREATE UNIQUE INDEX`	`CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);`	Create a unique index on a table, forbidding duplicate values
`DROP INDEX`	`/* MS Access */ DROP INDEX index_name`	Delete an index in a table

	`ON table_name;`
	
	`/* SQL Server */`
	`DROP INDEX`
	`table_name.index_name`
	`;`
	`/* DB2/Oracle */`
	`DROP INDEX`
	`index_name;`
	
	`/* MySQL */`
	`ALTER TABLE`
	`table_name DROP INDEX`
	`index_name;`	

SQL Constraints

Constraints are for specifying rules for data in a table. Use them with the `CREATE TABLE` statement for a new table or the `ALTER TABLE` statement for an existing table.

The syntax is:

```
[CREATE|ALTER] TABLE table_name (
      column1 datatype constraint,
      column2 datatype constraint,
      column3 datatype constraint,
      ...
);
```

The table below lists common constraints in SQL:

Command	Description
`NOT NULL`	Ensure that a column cannot have a `NULL` value
`UNIQUE`	Ensure that all values in a column are different
`PRIMARY KEY`	A combination of `NOT NULL` and `UNIQUE`: uniquely identifies each row in a table.
`FOREIGN KEY`	Prevent actions that would destroy links between tables
`CHECK`	Ensure that the values in a column satisfy a specific condition
`DEFAULT`	Set a default value for a column that contains no specified value
`AUTO_INCREMENT`	Allow the automatic generation of a unique number when inserting a new record into a table.

SQL Transactions

A transaction is the propagation of one or more changes to the database. For example, you perform a transaction if you perform create, update, and delete operations on a table.

Below we list the top SQL transactional commands:

Command	Syntax	Description
`COMMIT`	`COMMIT;`	Save changes invoked by a transaction to the database

`ROLLBACK`	`ROLLBACK;`
	
	`/* Roll back a given SAVEPOINT */`
	`ROLLBACK TO SAVEPOINT_NAME;`
		Undo transactions not yet saved to the database

`SAVEPOINT`	`SAVEPOINT SAVEPOINT_NAME;`
	
	`/* remove a SAVEPOINT that you have created */`
	`RELEASE SAVEPOINT SAVEPOINT_NAME;`
		Roll the transaction back to a certain point without rolling back the entire transaction

`SET TRANSACTION`	`SET TRANSACTION [ READ WRITE | READ ONLY ];`	Initiate a database transaction, and specify characteristics for the transaction that follows. For example, you can specify a transaction to be `READ ONLY` or `READ WRITE`.

SQL Performance Tuning Tips

As this article is a SQL basics cheat sheet, we present the following SQL performance optimization tips without elaboration.

- ✓ Add missing indexes and check for unused indexes
- ✓ Use `SELECT` fields instead of `SELECT *`
- ✓ Avoid `SELECT DISTINCT`
- ✓ Avoid using multiple `OR` in the `FILTER` predicate
- ✓ Create joins with `INNER JOIN` (not `WHERE`)
- ✓ Avoid too many `JOIN`s
- ✓ Use `WHERE` instead of `HAVING` to define filters
- ✓ Use wildcards at the end of a phrase only
- ✓ Use `TOP` and `LIMIT` to sample query results
- ✓ Minimize the usage of query hints

✓ Minimize large write operations
✓ Run the query during off-peak hours and analyze wait statistics

Conclusion

This SQL command cheat sheet covers most SQL database tasks. We hope it has helped you solve your problems at hand. Bookmark the documentation links for your SQL implementation, such as MySQL or PostgreSQL. Remember to check out our articles on SQL and our beginner-friendly cyber security courses, which cover SQL injection attacks:

- Complete Python 3 Ethical Hacking Course: Zero To Mastery
  - https://courses.stationx.net/p/complete-python-3-ethical-hacking-course-zero-to-mastery
- Learn Ethical Hacking From Scratch
  - https://courses.stationx.net/p/learn-ethical-hacking-from-scratch
- Learn Website Hacking / Penetration Testing From Scratch
  - https://courses.stationx.net/p/learn-website-hacking-penetration-testing-from-scratch