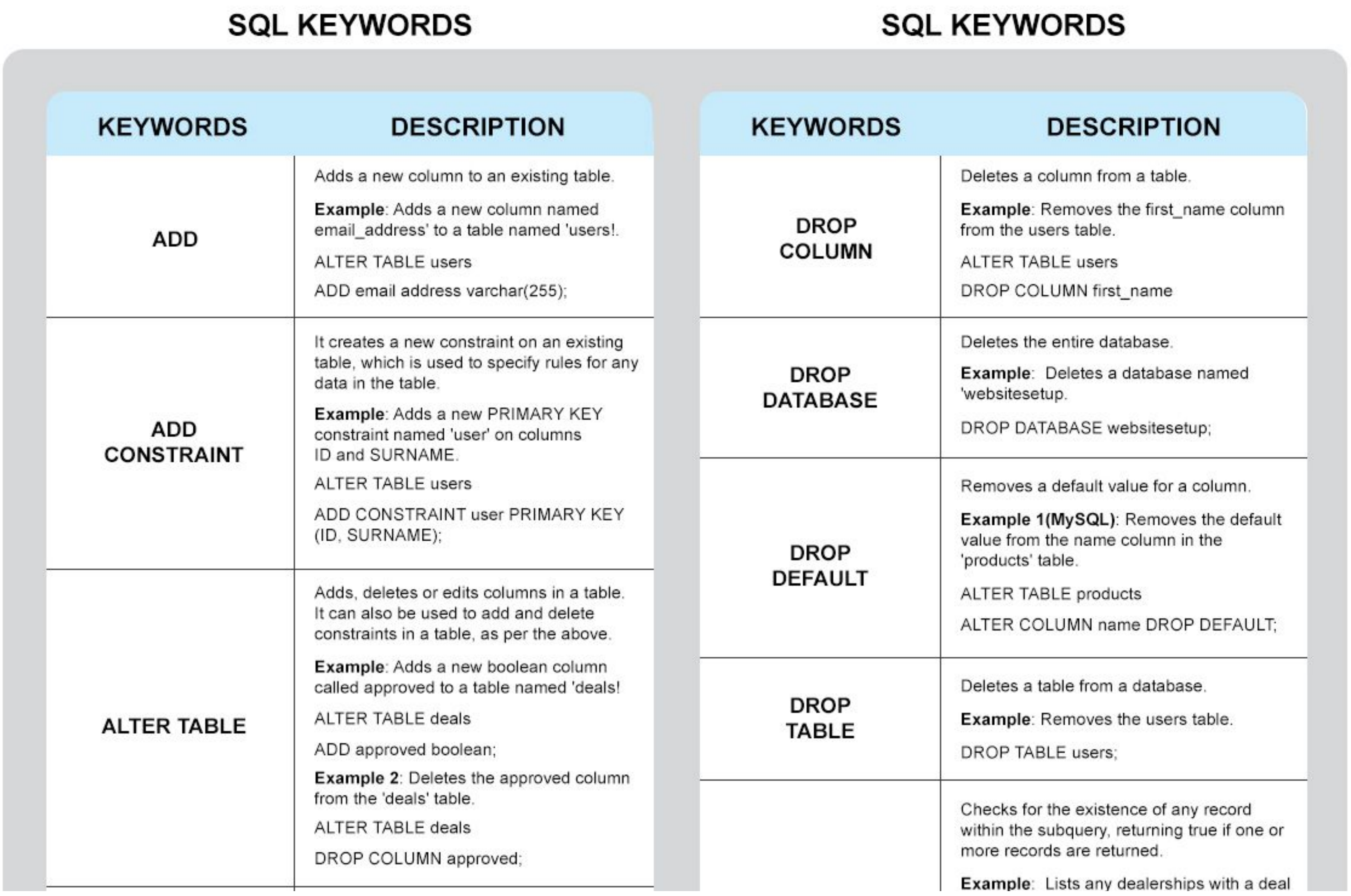
Ultimate SQL Cheat Sheet

```
SELECT * FROM employees;

SELECT * FROM employees
WHERE emp_id = 2;

SELECT * FROM employees
ORDER BY name;
```



SQL KEYWORDS

KEYWORDS	DESCRIPTION
ADD	Adds a new column to an existing table. **Example**: Adds a new column named email_address' to a table named 'users!. ALTER TABLE users ADD email address varchar(255);
ADD CONSTRAINT	It creates a new constraint on an existing table, which is used to specify rules for any data in the table. **Example**: Adds a new PRIMARY KEY constraint named 'user' on columns ID and SURNAME. ALTER TABLE users ADD CONSTRAINT user PRIMARY KEY (ID, SURNAME);
ALTER TABLE	Adds, deletes or edits columns in a table. It can also be used to add and delete constraints in a table, as per the above. **Example**: Adds a new boolean column called approved to a table named 'deals! ALTER TABLE deals ADD approved boolean; **Example 2**: Deletes the approved column from the 'deals' table. ALTER TABLE deals DROP COLUMN approved;

SQL KEYWORDS

KEYWORDS	DESCRIPTION
DROP COLUMN	Deletes a column from a table. **Example**: Removes the first_name column from the users table. ALTER TABLE users DROP COLUMN first_name
DROP DATABASE	Deletes the entire database. **Example**: Deletes a database named 'websitesetup. DROP DATABASE websitesetup;
DROP DEFAULT	Removes a default value for a column. **Example 1(MySQL)**: Removes the default value from the name column in the 'products' table. ALTER TABLE products ALTER COLUMN name DROP DEFAULT;
DROP TABLE	Deletes a table from a database. **Example**: Removes the users table. DROP TABLE users;
	Checks for the existence of any record within the subquery, returning true if one or more records are returned. **Example**: Lists any dealerships with a deal

ALTER COLUMN	Changes the data type of a table's column. Example In the 'users' table, make the column 'incept_date' into a 'datetime' type. ALTER TABLE users ALTER COLUMN incept_date datetime;

ALL	Returns true if all of the subquery values meet the passed condition.
	Example: Returns the users with a higher number of tasks than the user with the highest number of tasks in the HR department (id 2).
	SELECT first_name, surname, tasks_no FROM users
	WHERE tasks_no > ALL (SELECT tasks FROM user WHERE department_id = 2);

AND	Used to join separate conditions within a WHERE clause.
	Example:Returns events located in London, United Kingdom.
	SELECT * FROM events
	WHERE host_country='United Kingdom' AND host_city='London';

ANY	Returns true if any of the subquery values meet the given condition.
	Example:Returns products from the products table which have received orders – stored in the orders table - with a quantity of more than 5.
	SELECT name
	FROM products
	WHERE productId = ANY (SELECT productId FROM orders WHERE Qty >

AS	Renames a table or column with an alias value which only exists for the duration of the query.
	Example:Aliases north_east_user_ subscriptions column.
	SELECT north_east_user_subscriptions AS ne_subs
	FROM users
	WHERE ne_subs > 5;

ASC	Used with ORDER BY to return the data in ascending order. **Example**:Apples, Bananas, Peaches, Raddish.

BETWEEN	Selects values within the given range.
	Example 1:Selects stock with a quantity between 100 and 150.
	SELECT * FROM stock
	WHERE quantity BETWEEN 100 AND 150;
	Example 2:Selects stock with a quantity NOT between 100 and 150. Alternatively, using the NOT keyword here reverses the logic and selects values outside the given range.
	SELECT * FROM stock
	WHERE quantity NOT BETWEEN 100 AND 150;

CASE	Change query output depending on conditions. **Example 1**:Returns users and their subscriptions, along with a new column called activity_levels that makes a judgement based on the number of subscriptions. SELECT first_name, surname, subscriptions CASE WHEN subscriptions > 10 THEN 'Very active'

EXISTS	finance percentage less than 10.
	SELECT dealership_name
	FROM dealerships
	WHERE EXISTS (SELECT deal_name FROM deals WHERE dealership_id = deals.dealership_id AND finance_
	percentage < 10);

FROM	Specifies which table to select or delete data from.
	Example:Selects data from the users table.
	SELECT area_manager
	FROM area_managers
	WHERE EXISTS (SELECT ProductName FROM Products WHERE area_manager_id = deals.area_manager_id AND Price < 20);

IN	Used alongside a WHERE cause as a shorthand for multiple OR conditions.
	So instead of:
	SELECT * FROM users
	WHERE country = 'USA' OR country = 'United Kingdom' OR
	country = 'Russia' OR country = 'Australia';
	You can use:-
	SELECT * FROM users
	WHERE country IN (USA', 'United Kingdom', 'Russia', 'Australia');

INSERT INTO	Add new rows to a table. **Example**: Adds a new vehicle. INSERT INTO cars (make, model, mileage, year) VALUES ('Audi', 'A3', 30000, 2016);

IS NULL	Tests for empty (NULL) values.
	Example: Returns users that haven't given a contact number.
	SELECT * FROM users
	WHERE contact_number IS NULL;

IS NOT NULL	The reverse of NULL Tests for values that aren't empty / NULL.

LIKE	Returns true if the operand value matches a pattern.
	Example: Returns true if the user's first_name ends with 'son'.
	SELECT * FROM users
	WHERE first_name LIKE '%son';

NOT	Retums true if a record DOESN'T meet the condition.
	Example: Returns true if the user's first_name doesn't end with 'son'.
	SELECT * FROM users
	WHERE first_name NOT LIKE 'son';

OR	Used alongside WHERE to include data when either condition is true.
	Example: Returns users that live in either Sheffield or Manchester.
	SELECT * FROM users
	WHERE city = 'Sheffield' OR 'Manchester';

ORDER BY	Used to sort the result data in ascending (default) or descending order through the use of ASC or DESC keywords.
	Example: Returns countries in alphabetical order.
	SELECT * FROM countries
	ORDER BY name;

	Retums results where the row number meets the passed condition.

	WHEN Quantity BETWEEN 3 AND 10 THEN 'Active'
	ELSE 'Inactive'
	END AS activity levels
	FROM users;

CHECK	Adds a constraint that limits the value which can be added to a column. **Example 1(MySQL)**: Makes sure any users added to the users table are 18 or over. CREATE TABLE users first_name varchar(255), age int, CHECK (age=18)); **Example 2(MySQL)**: Adds a check after the table has already been created. ALTER TABLE users ALTER TABLE users
CREATE DATABASE	Creates a new database. **Example 1(MySQL)**: Creates a new database named websiteset. CREATE DATABASE websitesetup;
CREATE TABLE	Creates a new table. **Example**: Creates a new table called 'users' in the websitesetup database. CREATE TABLE users id int, first_name varchar(255), surname varchar(255), address varchar(255), contact number int);
DEFAULT	Sets a default value for a column; **Example 1(MySQL)**: Creates a new table called Products which has a name column with a default value of 'Placeholder Name' and an available_from column with a default value of today's date. CREATE TABLE products id int, name varchar(255) DEFAULT 'Placeholder Name', available from date DEFAULT GETDATE()); **Example 2(MySQL)**: The same as above, but editing an existing table. ALTER TABLE products ALTER name SET DEFAULT 'Placeholder Name' ALTER available_from SET DEFAULT GETDATE();
DELETE	Delete data from a table. **Example**: Removes a user with a user_id of 674. DELETE FROM users WHERE user_id = 674;
DESC	Used with ORDER BY to retum the data in descending order. **Example**: Raddish, Peaches, Bananas, Apples.

ROWNUM	**Example** : Returns the top 10 countries from the countries table.
	SELECT * FROM countries
	WHERE ROWNUM C= 10;

SELECT	Used to select data from a database, which is then returned in a results set. **Example** : Selects all columns from all users. SELECT * FROM users; Example 2: Selects the first_name and surname columns from all users.XX SELECT first_name, surname FROM users;

SELECT DISTINCT	Sames as SELECT, except duplicate values are excluded.
	Example : Creates a backup table using data from the users table.
	SELECT * INTO usersBackup2020
	FROM users;

SELECT INTO	Copies data from one table and inserts it into another. **Example**: Returns all countries from the users table, removing any duplicate values (which would be highly likely). SELECT DISTINCT country from users;
SELECT TOP	Allows you to return a set number of records to retum from a table. **Example**: Returns the top 3 cars from the cars table. SELECT TOP 3 * FROM cars;

SET	Used alongside UPDATE to update existing data in a table.
	Example: Updates the value and quantity values for an order with an id of 642 in the orders table.
	UPDATE orders
	SET value = 19.49, quantity = 2
	WHERE id = 642;

SOME	Identical to ANY
TOP	Used alongside SELECT to retum a set number of records from a table. **Example**: Returns the top 5 users from the users table. SELECT TOP 5 * FROM users;
TRUNCATE TABLE	Similar to DROP, but instead of deleting the table and its data, this deletes only the data. **Example**: Empties the sessions table, but leaves the table itself intact. TRUNCATE TABLE sessions;
UNION	Combines the results from 2 or more SELECT statements and returns only distinct values. **Example**: Returns the cities from the events and subscribers tables. SELECT city FROM events UNION SELECT city from subscribers;
UNION ALL	The same as UNION but includes duplicate values.
	This constraint ensures all values in a column are unique. **Example 1 (MySQL)**: Adds a unique constraint to the id column when creating a new users table. CREATE TABLE users id int NOT NULL,

UNIQUE	name varchar(255) NOT NULL, UNIQUE (id)); **Example 2 (MySQL)**: Alters an existing column to add a UNIQUE constraint. ALTER TABLE users ADD UNIQUE (id);

UPDATE	Updates existing data in a table.
	Example: Updates the mileage and service Due values for a vehicle with an
	UPDATE cars
	SET mileage = 23500, serviceDue = 8
	WHERE id = 45;

VALUES	Used alongside the INSERT INTO keyword to add new values to a table.
	Example: Adds a new car to the cars table.
	SET mileage = 23500, serviceDue = 8
	INSERT INTO cars (name, model, year)
	VALUES ('Ford', 'Fiesta', 2010);

WHERE	Filters results to only include data which meets the given condition.
	Example: Returns orders with a quantity of more than 1 item.
	SELECT * FROM orders
	WHERE quantity > 1;

COMMENTS

SINGLE LINE COMMENTS

Single line comments start with -.Any text after these 2 characters to the end of the line will be ignored

```
-- My Select query
SELECT * FROM users;
```

MULTILINE COMMENTS

Multiline comments start with /* and end with */. They stretch across multiple lines until the closing characters have been found.

```
/*
This is my select query.
It grabs all rows of data from the users table
SELECT * FROM users;
This is another select query, which I don't
want to execute yet
SELECT * FROM tasks;
*/
```

MySQL Data Types

```
CREATE TABLE users (
id int,
first_name varchar(255)
);
```

STRING DATA TYPES

DATA TYPE	DESCRIPTION
CHAR(SIZE)	Fixed length string which can contain letters, numbers and special characters. The size parameter sets the maximum

NUMERIC DATA TYPES

DATA TYPE	DESCRIPTION
BIT(SIZE)	A bit-value type with a default of 1. The allowed number of bits in a value is set via the size parameter, which can hold values

	string length, from 0 - 255 with a default of 1.
VARCHAR(SIZE)	Variable length string similar to CHARQ, but with a maximum string length range from 0 to 65535.
BINARY(SIZE)	Similar to CHARO but stores binary byte strings.
VARCHAR(SIZE)	Similar to CHARO but stores binary byte strings.
TINYBLOB	Holds Binary Large Objects (BLOBs) with a maxlength of 255 bytes.
TINYTEXT	Holds a string with a maximum length of 255 characters. Use VARCHAR() instead, as it's fetched much faster.
TEXT(size)	Holds a string with a maximum length of 65535 bytes. Again, better to use VARCHARO
BLOB(size)	Holds Binary Large Objects (BLOBS) with a max length of 65535 bytes.
MEDIUMBLOB	Holds Binary Large Objects (BLOBs) with a max length of 16,777,215 bytes
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters.
LONGBLOB	Holds Binary Large Objects (BLOBS) with a max length of 4,294,967,295 bytes
ENUM(a, b, c, etc...)	A string object that only has one value, which is chosen from a list of values which you define, up to a maximum of 65535 values. If a value is added which isn't on this list, it's replaced with a blank value instead. Think of ENUM being similar to HTML radio boxes in this regard. CREATE TABLE tshirts (color ENUM 'red', 'green', 'blue', 'yellow', 'purple'));
SET (a, b, c, etc...)	A string object that can have 0 or more values, which is chosen from a list of values which you define, up to a maximum of 64 values. Think of SET being similar to HTML checkboxes in this regard

DATE/TIME DATA TYPES

DATA TYPE	DESCRIPTION
DATE	A simple date in YYYY-MM-DD format, with a supported range from "1000-01-01' to '9999-12-31.
DATETIME(fsp)	A date time in YYYY-MM-DD hh:mmess format, with a supported range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.' By adding DEFAULT and ON UPDATE to the column definition, it automatically sets to the current date/time.
TIMESTAMP(fsp)	A Unix Timestamp, which is a value relative to the number of seconds since the Unix epoch (1970-01-01 00:00:00' UTC). This has a supported range from '1970-01-01 00:00:01 UTC to 2038-01-09 03:14:07' UTC By adding DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT TIMESTAMP to the column definition, it automatically sets to current date/time.
TIME(fsp)	A time in hh:mmess format, with a supported range from '838:59:59 to '838:59:59.

	from 1 to 64.
TINYINT(SIZE)	A very small integer with a signed range of -128 to 127, and an unsigned range of 0 to 255. Here, the size parameter specifies the maximum allowed display width, which is 255.
BOOLEAN	Same as BOOL
SMALLINT(size)	A small integer with a signed range of -32768 to 32767, and an unsigned range from 0 to 65535. Here, the size parameter specifies the maximum allowed display width, which is 255.
MEDIUMINT(size)	A medium integer with a signed range of -8388608 to 8388607,and an unsigned range from 0 to 16777215. Here, the size parameter specifies the maximum allowed display width, which is 255.
INT(size)	A medium integer with a signed range of -2147483648 to 2147483647, and an unsigned range from 0 to 4294967295. Here, the size parameter specifies the maximum allowed display width, which is 255.
INTEGER(size)	Same as INT.
BIGINT(size)	A medium integer with a signed range of -9223372036854775808 to 9223372036854775807, and an unsigned range from 0 to18446744073709551615. Here, the size parameter specifies the maximum allowed display width, which is 255.
FLOAT(p)	A floating point number value. If the precision (p) parameter is between O to 24, then the data type is set to FLOAT(), whilst if its from 25 to 53,the data type is set to DOUBLE(). This behaviour is to make the storage of values more efficient.
DOUBLE(size, d)	A floating point number value where the total digits are set by the size parameter, and the number of digits after the decimal point is set by the d parameter
DECIMAL(size, d)	An exact fixed point number where the total number of digits is set by the size parameters, and the total number of digits after the decimal point is set by the d parameter. For size, the maximum number is 65 and the default is 10, whilst ford,the maximum number is 30 and the default is 10.
DEC(sze, d)	Same as DECIMAL.

YEAR	A year, with a supported range of '1901' to '2155.

OPERATORS

ARITHMETIC OPERATORS

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

BITWISE OPERATORS

OPERATOR	DESCRIPTION
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

COMPARISON OPERATORS

OPERATOR	DESCRIPTION
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

COMPOUND OPERATORS

OPERATOR	DESCRIPTION
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

FUNCTIONS

STRING FUNCTIONS

DATA TYPE	DESCRIPTION
ASCIIA	Returns the equivalent ASCII value for a specific character.
CHAR_LENGTH	Returns the character length of a string.
CHARACTER LENGTH	Same as CHAR_LENGTH.
CONCAT	Adds expressions together, with a minimum of 2.
CONCAT_WS	Adds expressions together, but with a separator between each value.
	Returns an index value relative to the

NUMERIC FUNCTIONS

DATA TYPE	DESCRIPTION
ABS	Returns the absolute value of the given number.
ACOS	Returns the arc cosine of the given number.
ASIN	Returns the arc sine of the given number.
ATAN	Returns the arc tangent of one or 2 given numbers
ATAN2	Return the arc tangent of 2 given numbers.
AVG	Returns the average value of the given

FIELD	position of a value within a list of values.
FORMAT	When passed a number, returns that number formatted to include commas (eg 3,400,000).
INSERT	Allows you to insert one string into another at a certain point, for a certain number of characters.
INSTR	Returns the position of the first time one string appears within another.
LCASE	Convert a string to lowercase.
LEFT	Starting from the left, extract the given number of characters from a string and return them as another.
LENGTH	Returns the length of a string, but in bytes.
LOCATE	Returns the first occurrence of one string within another,
LOWER	Same as LCASE
LPAD	Left pads one string with another, to a specific length.
LTRIM	Remove any leading spaces from the given string.
MID	Extracts one string from another, starting from any position.
POSITION	Returns the position of the first time one substring appears within another.
REPEAT	Allows you to repeat a string.
REPLACE	Allows you to replace any instances of a substring within a string, with a new substring.
REVERSE	Reverses the string.
RIGHT	Starting from the right, extract the given number of characters from a string and return them as another.
RPAD	Right pads one string with another, to a specific length.
RTRIM	Removes any trailing spaces from the given string.
SPACE	Returns a string full of spaces equal to the amount you pass it.
STRCMP	Compares 2 strings for differences.
SUBSTR	Extracts one substring from another, starting from any position.
SUBSTRING	Same as SUBSTR
SUBSTRING_ INDEX	Returns a substring from a string before the passed substring is found the number of times equals to the passed number.
TRIM	Removes trailing and leading spaces from the given string Same as if you were to run LTRIM and RTRIM together.
UCASE	Convert a string to uppercase.

AVG	expression.
CEIL	Returns the closest whole number (integer) upwards from a given decimal point number.
CEILING	Same as CEIL
COS	Returns the cosine of a given number.
COT	Returns the cotangent of a given number.
COUNT	Returns the amount of records that are returned by a SELECT query.
DEGREES	Converts a radians value to degrees.
DIV	Allows you to divide integers.
EXP	Returns to the power of the given number.
FLOOR	Returns the closest whole number (integer) downwards from a given decimal point number.
GREATEST	Returns the highest value in a list of arguments.
LEAST	Returns the smallest value in a list of arguments.
LN	Returns the natural logarithm of the given number.
LOG	Returns the natural logarithm of the given number, or the logarithm of the given number to the given base.
LOG10	Does the same as LOG, but to base 10
LOG2	Does the same as LOG, but to base 2.
MAX	Returns the highest value from a set of values.
MIN	Returns the lowest value from a set of values.
MOD	Returns the remainder of the given number divided by the other given number.
PI	Returns PI.
POW	Returns the value of the given number raised to the power of the other given number.
POWER	Same as POW.
RADIANS	Converts a degrees value to radians.
RAND	Returns a random number.
SIGN	Returns the sign of the given number.
SIN	Returns the sine of the given number.
SQRT	Returns the square root of the given number.
SUM	Returns the value of the given set of values combined.
TAN	Returns the tangent of the given number.

UPPER	Same as UCASE.

TRUNCATE	Returns number truncated to the given number of decimal places.

NUMERIC FUNCTIONS

DATA TYPE	DESCRIPTION
ADDDATE	Add a date interval (eg 10 DAY) to a date (eg 20/01/20) and return the result (eg: 20/01/30).
ADDTIME	Add a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (07:00).
CURDATE	Get the current date.
CURRENT_DATE	Same as CURDATE.
CURRENT_ TIMESTAMP	Get the current date and time.
CURTIME	Same as CURRENT_TIME.
DATE	Extracts the date from a datetime expression.
DATEDIFF	Returns the number of days between the 2 given dates.
DATE_ADD	Same as ADDDATE.
DATE_FORMAT	Formats the date to the given pattern.
DATE_SUB	Subtract a date interval (eg: 10 DAY) to a date (eg 20/01/20) and return the result (eg: 20/01/10).
DAY	Returns the day for the given date.
DAYNAME	Returns the weekday name for the given date.
DAYOFWEEK	Returns the index for the weekday for the given date.
DAYOFYEAR	Returns the day of the year for the given date.
EXTRACT	Extract from the date the given part (eg MONTH for 20/01/20=01).
FROM DAYS	Return the date from the given numeric date value. HOUR.
HOUR	Return the hour from the given date.
LAST DAY	Get the last day of the month for the given date.
LOCALTIME	Gets the current local date and time.
LOCALTIMESTAMP	Same as LOCALTIME.
MAKEDATE	Creates a date and returns it, based on the given year and number of days values.
MICROSECOND	Returns the microsecond of a given time or datetime.
MINUTE	Returns the minute of the given time or datetime.

NUMERIC FUNCTIONS

DATA TYPE	DESCRIPTION
IN	Returns the given number in binary.
BINARY	Returns the given value as a binary string
CAST	Convert one type into another.
COALESCE	From a list of values, return the first non-null value.
CONNECTION_ID	For the current connection, return the unique connection ID.
CONV	Convert the given number from one numeric base system into another.
CONVERT	Convert the given value into the given datatype or character set.
CURRENT_USER	Return the user and hostname which was used to authenticate with the server.
DATABASE	Get the name of the current database.
GROUP BY	Used alongside aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the results. **Example**: Lists the number of users with active orders. SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders;
HAVING	It's used in the place of WHERE with aggregate functions. **Example**: Lists the number of users with active orders, but only include users with more than 3 active orders. SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders HAVING COUNT(user_id) > 3;
IF	If the condition is true return a value, otherwise return another value.
IFNULL	If the expression is null, return 1 otherwise return O.
LAST_INSERT_ID	For the last row which was added or updated in a table, return the auto increment.
NULLIF	Compares the 2 given expressions. If they are equal, NULL is returned, otherwise the first expression is returned.
SESSION_USER	Return the current user and hostnames.
SYSTEM_USER	Same as SESSION_USER.
VERSION	Returns the current version of the MySQL powering the database.

MONTH	Returns the month of the given date.
MONTHNAME	Returns the name of the month of the given date.
NOW	Same as LOCALTIME.
PERIOD_ADD	Adds the given number of months to the given period.
PERIOD_DIFF	Returns the difference between 2 given periods.
QUARTER	Returns the year quarter for the given date.
SECOND	Returns the second of a given time or datetime.
SEC_TO_TIME	Returns a time based on the given seconds.
STR_TO_DATE	Creates a date and returns it based on the given string and format.
SUBDATE	Same as DATE_SUB.
SUBTIME	Subtracts a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (03:00).
SYSDATE	Same as LOCALTIME.
TIME	Returns the time from a given time or datetime.
TIME_FORMAT	Returns the given time in the given format.
TIME_TO_SEC	Converts and returns a time into seconds.
TIMEDIFF	Returns the difference between 2 given time/datetime expressions.
TIMESTAMP	Returns the datetime value of the given date or datetime.
TO_DAYS	Returns the total number of days that have passed from '00-00- 0000' to the given date.
WEEK	Returns the week number for the given date.
WEEKDAY	Returns the weekday number for the given date.
WEEKOFYEAR	Returns the week number for the given date.
YEAR	Returns the year from the given date.
YEARWEEK	Returns the year and week number for the given date.

WILDCARDS

NAME	DESCRIPTION

%	Equates to zero or more characters.
	Example 1: Find all users with surnames ending in 'son'.
	SELECT * FROM users
	WHERE surname LIKE 'xson';
	Example 2: Find all users with surnames ending in 'son'.
	SELECT * FROM users
	WHERE city LIKE '%che%';

-	Equates to any single character.
	Example 1: Find all users living in cities beginning with any 3 characters, followedby 'chester!
	SELECT * FROM users
	WHERE city LIKE '___chester'

(CHADIST)	Equates to any single character in the list.
	Example 1: Find all users with first names beginning with J, H or M.
	SELECT * FROM users
	WHERE first_name LIKE '(jhm]>';
	Example 2: Find all users with first names beginning letters between A-L.
	SELECT * FROM users
	WHERE first_name LIKE '[a-1]>';
	Example 3: Find all users with first names not ending with letters between n-s.
	SELECT * FROM users
	WHERE first_name LIKE '%[!n-s]);

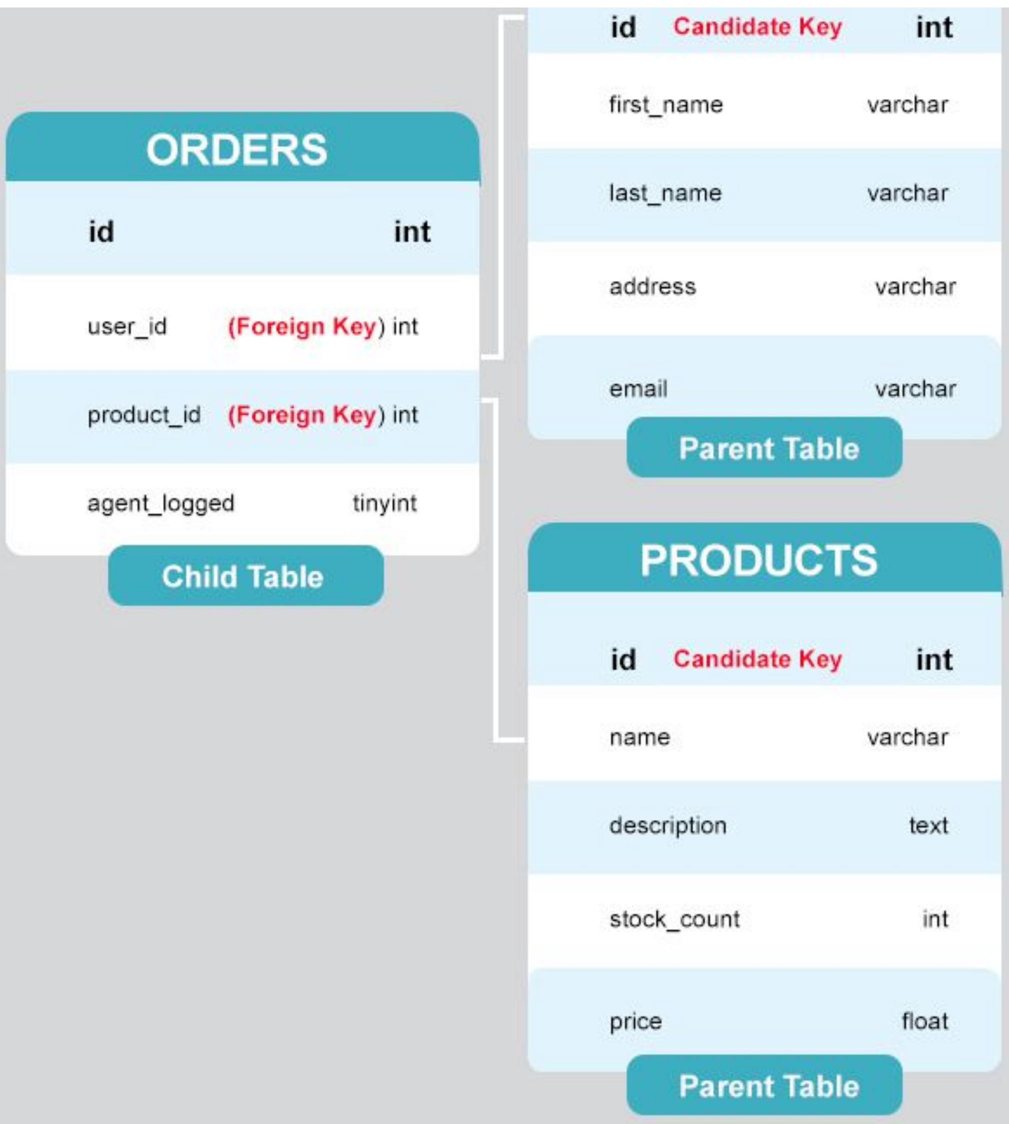
KEYS

FOREIGN KEY

USERS

EXAMPLE 1 (MYSQL)

CREATE TABLE orders (



id	Candidate Key	int
first_name		varchar
last_name		varchar
address		varchar
email		varchar

Parent Table

ORDERS

id		int
user_id	(Foreign Key)	int
product_id	(Foreign Key)	int
agent_logged		tinyint

Child Table

PRODUCTS

id	Candidate Key	int
name		varchar
description		text
stock_count		int
price		float

Parent Table

```
id int NOT NULL,
user_id int,
product_id int,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES users(id),
FOREIGN KEY (product_id) REFERENCES products(id)
);
```

EXAMPLE 2 (MYSQL)

```
ALTER TABLE orders
ADD FOREIGN KEY (user_id) REFERENCES users(id);
```

PRIMARY KEY

EXAMPLE 1 (MYSQL)

```
CREATE TABLE users (
id int NOT NULL AUTO_INCREMENT,
first_name varchar(255),
last_name varchar(255) NOT NULL,
address varchar(255),
email varchar(255),
PRIMARY KEY (id)
);
```

EXAMPLE 2 (MYSQL)

```
ALTER TABLE users
ADD PRIMARY KEY (first_name);
```

WILDCARDS

DATA TYPE	DESCRIPTION
CREATE INDEX	Creates an index named 'idx_test' on the first_name and surname columns of the users table. In this instance, duplicate values are allowed. CREATE INDEX idx_test ON users (first_name, surname);
CREATE UNIQUE INDEX	Creates an index named "idx_test' on the first name and surname columns of the users table. In this instance, duplicate values are allowed. CREATE UNIQUE INDEX idx_test ON users (first_name, surname);

VIEWS

CREATING VIEWS

```
CREATE VIEW priority_users AS
SELECT * FROM users
WHERE country = United Kingdom;
```

```
SELECT * FROM [priority_users];
```

REPLACING VIEWS

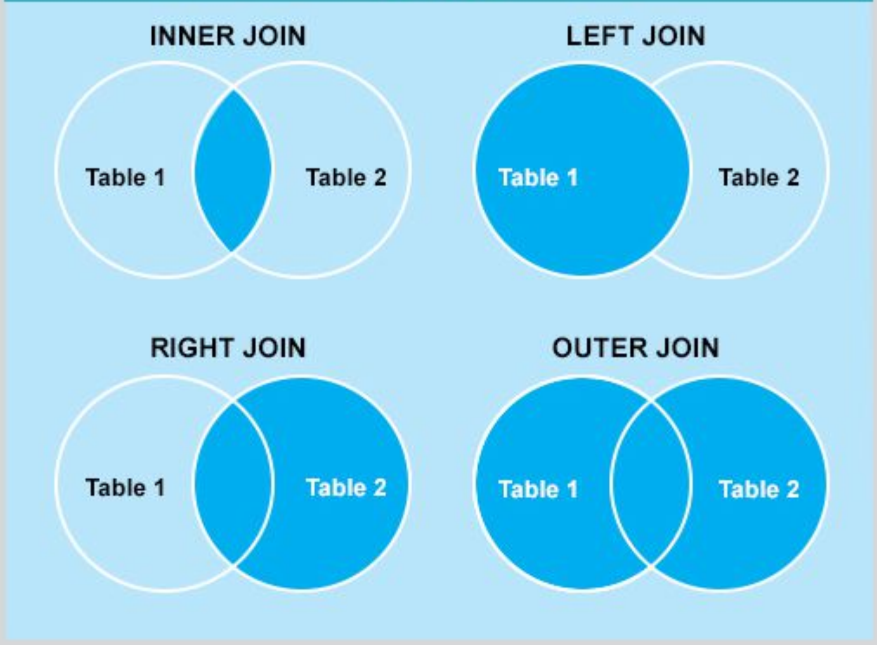
```
CREATE OR REPLACE VIEW [priority_users] AS
SELECT * FROM users
WHERE country = 'United Kingdom' OR country="USA';
```

DROP INDEX	Creates an index named "idx_test' on the first name and surname columns of the users table. In this instance, duplicate values are allowed. ALTER TABLE users DROP INDEX idx_test;

DELETING VIEWS

```
DROP VIEW priority_users;
```

JOINS



ORDERS

id	first_name	Last_name	address	email
1	Lube	Harison	1640, Kjetil Homme ...	Lube@153....
2	Healer	Reynolds	742, Norway, Denmark.	heal132@hot......
3	Simpson	Chekson	7, Nova Scotia.....	Simpson76@....
4	Chekson	Simpson	15, Santo Domingo....	Checkson80@....
5	Oliver	Harison	1640, San Salvador..	oliver5715@....
6	jones	Gabet	598, Caracas, 1010 ...	jones547@....
7	Micheal	Johnson	12, Western Michigan..	Micheal0017@....
8	Thomes	Smith	342, Mary Jones Station	smith098@....
9	Robyn	Gabet	598, Monte, Trigo..	robyn65478@....
10	Byony	Brown	165, First Ave. Usa..	byony8754@....

ORDERS

id	user_id	product_id	agent_logged
1	5	196	0
2	4	32	1
3	6	310	0
4	10	196	1
5	1	67	1
6	1	341	1
7	1	875	0
8	9	3	1
9	5	23	1
10	8	196	1

PROUDCTS

id	name	description	stock_count	price
102	Cartono..	Why	0	14,99
23	Cardbor..	Dec	1	3.49
3	Smart..	NULL	1	24.99
32	Troast 33..	Die	4	09.50
275	A4..	berta	5	4.99
436	Pack of 50.	you	5	12.99
341	Set of 25..	Modo	2	4.99
67	Large Car..	Pack	10	12.99
196	10, XP..	Hay que	10	15.99
310	Set of 35 ..	Cetta	10	2.99

PROUDCTS

id	name	description	stock_count	price
102	Cartono..	Why	0	14,99
23	Cardbor..	Dec	1	3.49
3	Smart..	NULL	1	24.99
32	Troast 33..	Die	4	09.50
275	A4..	berta	5	4.99
436	Pack of 50.	you	5	12.99
341	Set of 25..	Modo	2	4.99
67	Large Car..	Pack	10	12.99
196	10, XP..	Hay que	10	15.99
310	Set of 35 ..	Cetta	10	2.99

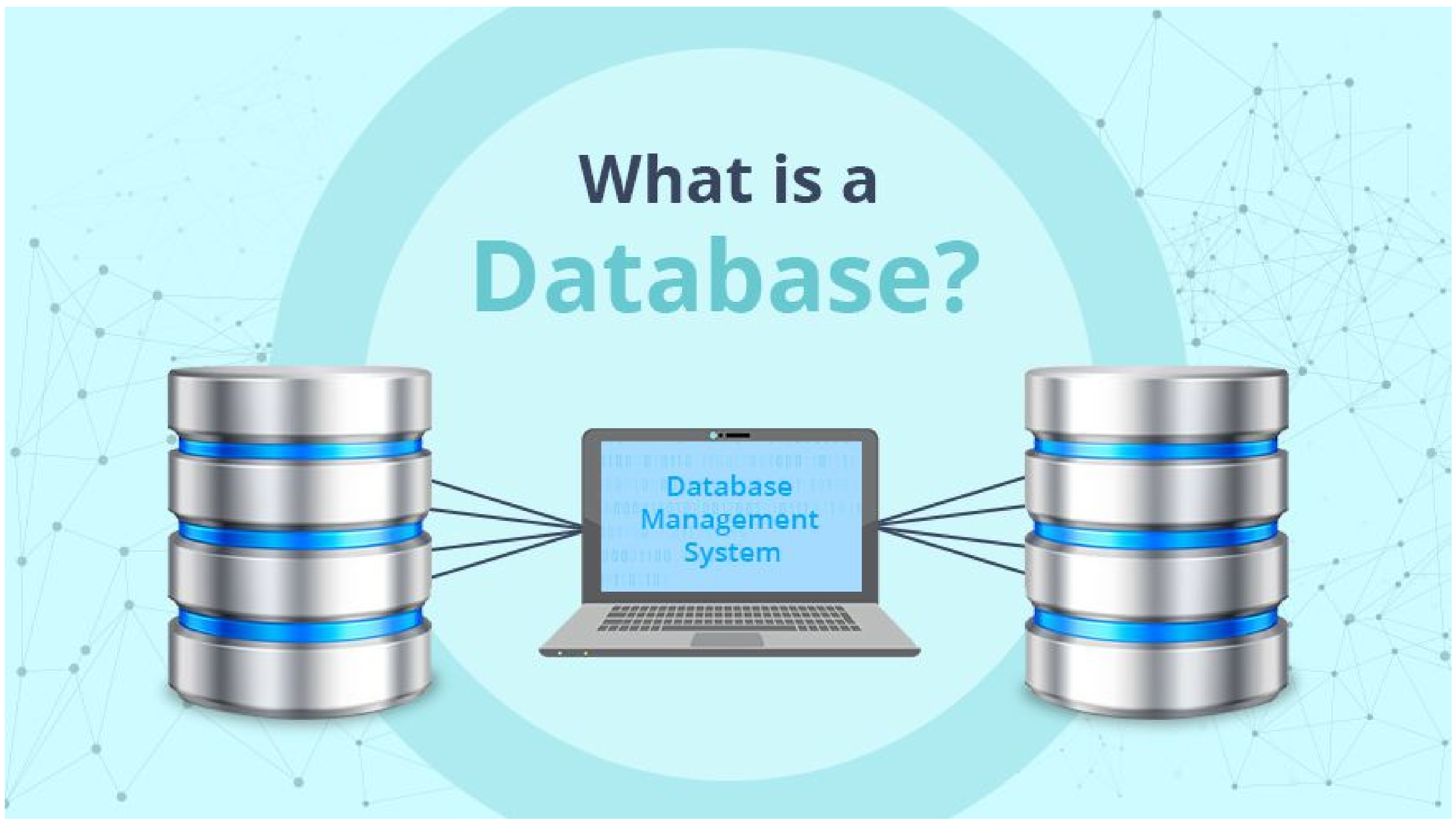
```
SELECT orders.id, users.first_name, users.surname, products.name as
'product name
FROM orders
INNER JOIN users on orders.user_id = users.id
INNER JOIN products on orders.product_id = products.id;
```

INNER JOIN RESULT SET

id	first_name	surname	product name
1	Oliver	Harison	10 X Plastic.....
2	Claire	Simpson	TripLast 33...
3	James	Gilbert	StorePac 5...
4	Bryony	Brown	10 X Plasti....
5	Luke	Harison	Large Care....
6	Luke	Harison	Set of 2 S...
7	Luke	Harison	A4 Storage...
8	Robyn	Gilbert	SmartMo...
9	Oliver	Harrison	Cardboar...
10	Thomas	Smith	10 X Plasti...

What is a Database?

Before we get started with SQL Cheat Sheet, we need to understand what is a database and why do we need SQL. If you want to learn this offline, you can download the SQL basics cheat sheet any time.



Data is a collection of facts related to any object. For example: - Your name, number, birthday, phone number, email address, etc. is a collection of facts about you.

Therefore, a database is a systematic collection of small units of information (data). For example: - An organized list of all the students of a school along with their data (Name, Phone Number, Birthday, etc.) is referred to as a database.

What is an RDBMS(Relational Database Management System)?

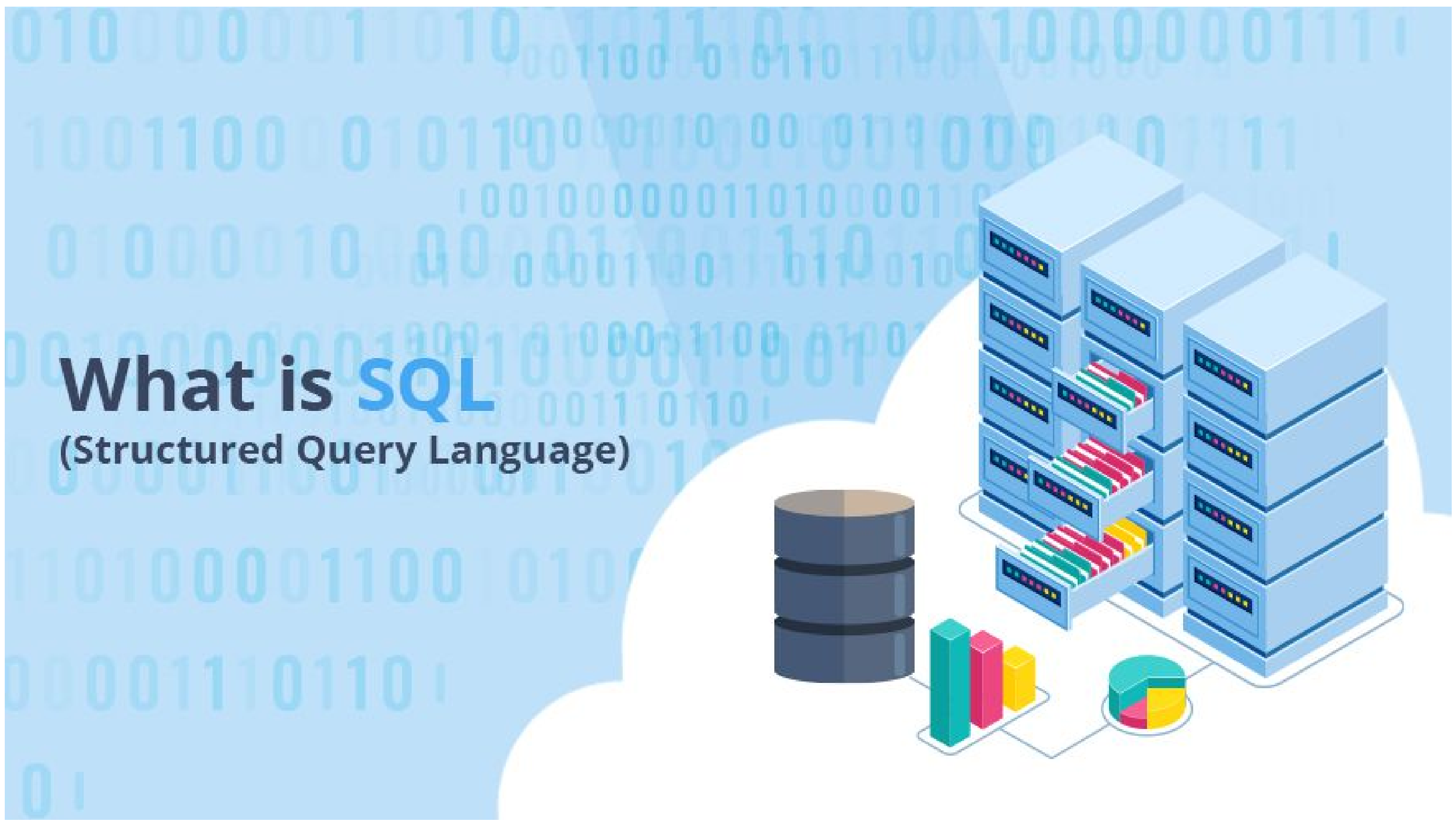
RDBMS Stands for **Relational DataBase Management System** and is a collection of tools that allow users to organize, manipulate, and visualize databases. RDMBS follows some standards that allow for the fastest response from a database and make it easier for humans to interact with a database.

Think of an RDBMS as a tool that allows you to play with your data and generate insights or value from the database.

What is SQL(Structured Query Language)?

Now that you understand what a database is and what a DBMS is, let's understand what SQL is.

To recap, a database is a collection of data and an RDBMS is a tool that allows you to interact with your data. Therefore, you would need a "language" to communicate with the database that humans and computers can understand, and that language is known as SQL.



SQL stands for Structured Query Language. As the name suggests, it is a structured language via which you can query the database for performing various tasks such as Storing, Manipulating, and retrieving data from a database.

SQL is the standard language when it comes to communicating with powerful relational databases such as Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Anything related to data in relational databases such as creating tables, limiting access to data, sorting, filtering, grouping, etc. is achieved using SQL.

In this SQL cheatsheet, we will be learning everything there is to learn about SQL.

SQL v/s MySQL: Difference Between Both

Most beginners usually get confused between the two terms - SQL and MySQL, and sometimes use the two interchangeably. However, there is a clear and vast difference between the two.

As defined earlier, SQL is a language that allows one to communicate with the database. MySQL on the other hand is an RDBMS in which you can type SQL commands to interact with the database.

SQL is the language/protocol that is used by relational database management systems to allow users to manipulate data in the database.

MySQL is a database management system that provides users with an interface to connect with databases. MySQL provides users with the ability to create various databases, tables, stored procedures, functions in their database servers. SQL is the language that is used to perform powerful operations on the database.

In this SQL Cheat Sheet, we would be looking at both SQL and MySQL which would help clarify the difference.

What Can SQL do?

SQL is a powerful programming language that allows you to communicate with the database. Almost all companies use databases to store and retrieve data in some form or the other.

Using SQL, you create Databases, and inside a database, you create various TABLES in which you can add all your data. Using SQL, you can:

1. Create / Delete Databases
2. Create / Delete Table(s) in a database
3. **SELECT** particular data from table(s)
4. **INSERT** data into tables
5. **UPDATE** data in tables
6. **DELETE** data from tables
7. Create **Views** in the database
8. Execute various aggregate functions

and tons of other cool stuff. So, let's see how we can harness this power.

How to get started with SQL?

To get started with writing SQL on your computer, you would need to install a Database Management Server. The RDBMS would then give you all the necessary tools to interact with your database.

There are various RDBMS that you can use, and it doesn't matter (much) what system you choose as long as it's working for you. Some of the most common RDBMS are:

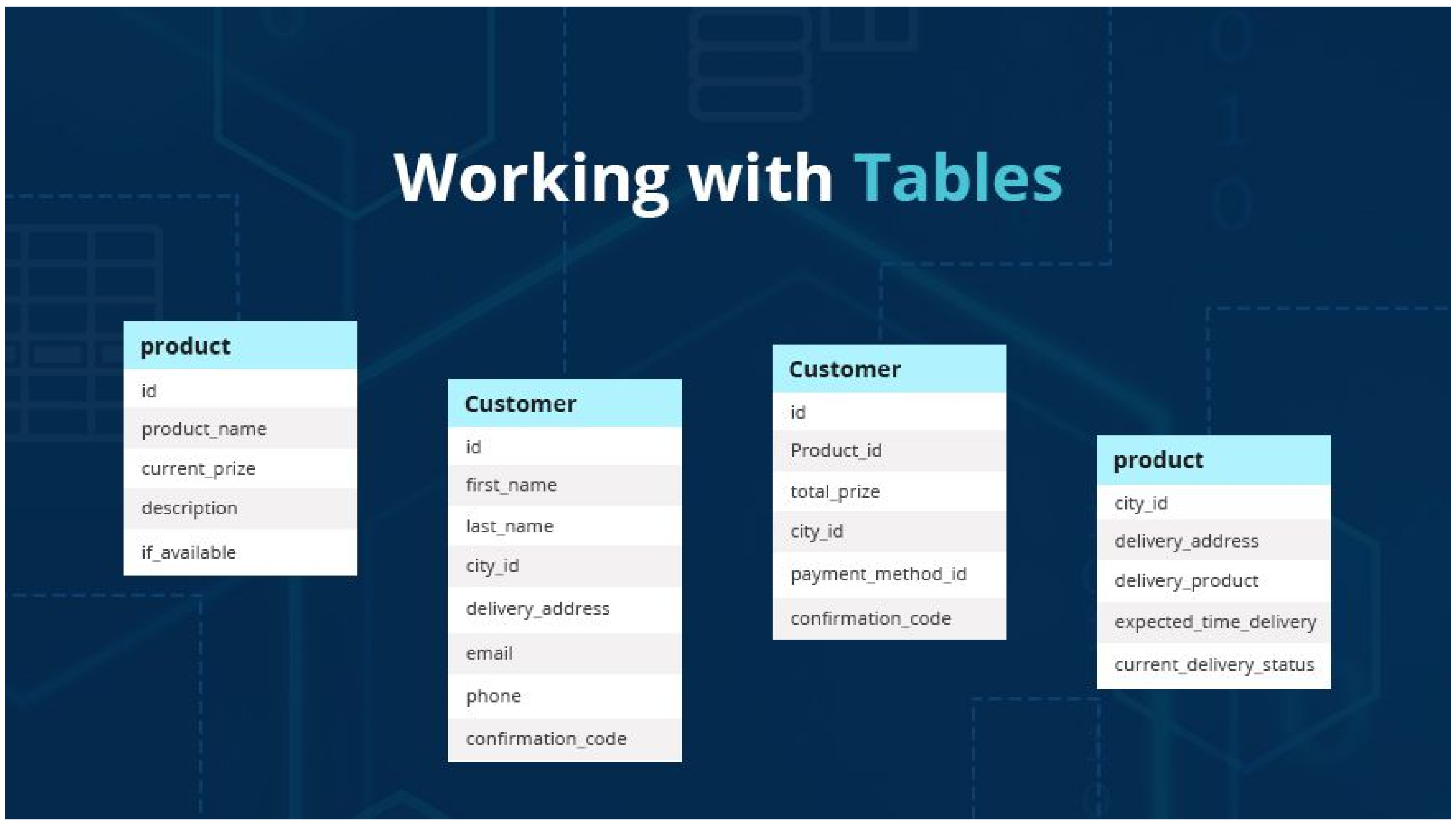
1. MySQL
2. Oracle
3. Microsoft SQL Server
4. PostgreSQL
5. Heidi SQL

Just install any of the RDBMS that you like from their official website and you should be able to create a database server by simply following their instructions. Once you have a database server ready, you can get access to a Query Editor where you can type all your SQL queries.

Now, let's get started with our cheat sheet and learn some SQL basics and SQL syntax to get the ball rolling.

Working with Tables

To work with SQL, you need to understand that data is organized into tables. One database would contain all the data for a single application (in most cases). A single database would have multiple tables that store values.



For example:- If you have a restaurant management application or implemented E-Delivery solution for smooth operations, you would have a database that contains tables such as:

1. Customers
2. Orders
3. Menu Items
4. Receipts
5. Combox

etc. Each table would contain a specific type of data and various tables could have different types of relations. Using SQL relations, we can combine values from different tables to fetch the data that we require. (More on relations in a later section)

To create a table, we would require two things. Firstly, we would need all the fields that we want to store in a table. Secondly, we want to define the type of data that would enter into a table.

Let's take the restaurant management application's Customers table as an example. We want to store some information about each of our customers such as their name, Phone Number, and Postal Code. Now that we are done with the first step, we need to define the data types of these values.

The name of a customer would be of character data type because we need to store alphabetical characters. Similarly, the phone number would be characters again because we would need to store country code, and special characters such as '+', '()', etc. The postal code would be of type integer because we need to store numbers. Here is how the table would look like:

Name	Phone	Postal Code
varchar(50)	varchar(15)	integer

To identify each customer uniquely, we add an ID to them so that we can use this ID to connect data from various tables. So, the final table structure could look something like this.

ID	Name	Phone	Postal Code
INTEGER	VARCHAR(50)	VARCHAR(15)	INTEGER

To create this table, we would use the CREATE SQL Command followed by the fields as follows:

CREATE TABLE customers(

ID INT NOT NULL,

name varchar(50),

phone varchar(15),

postalCode INT

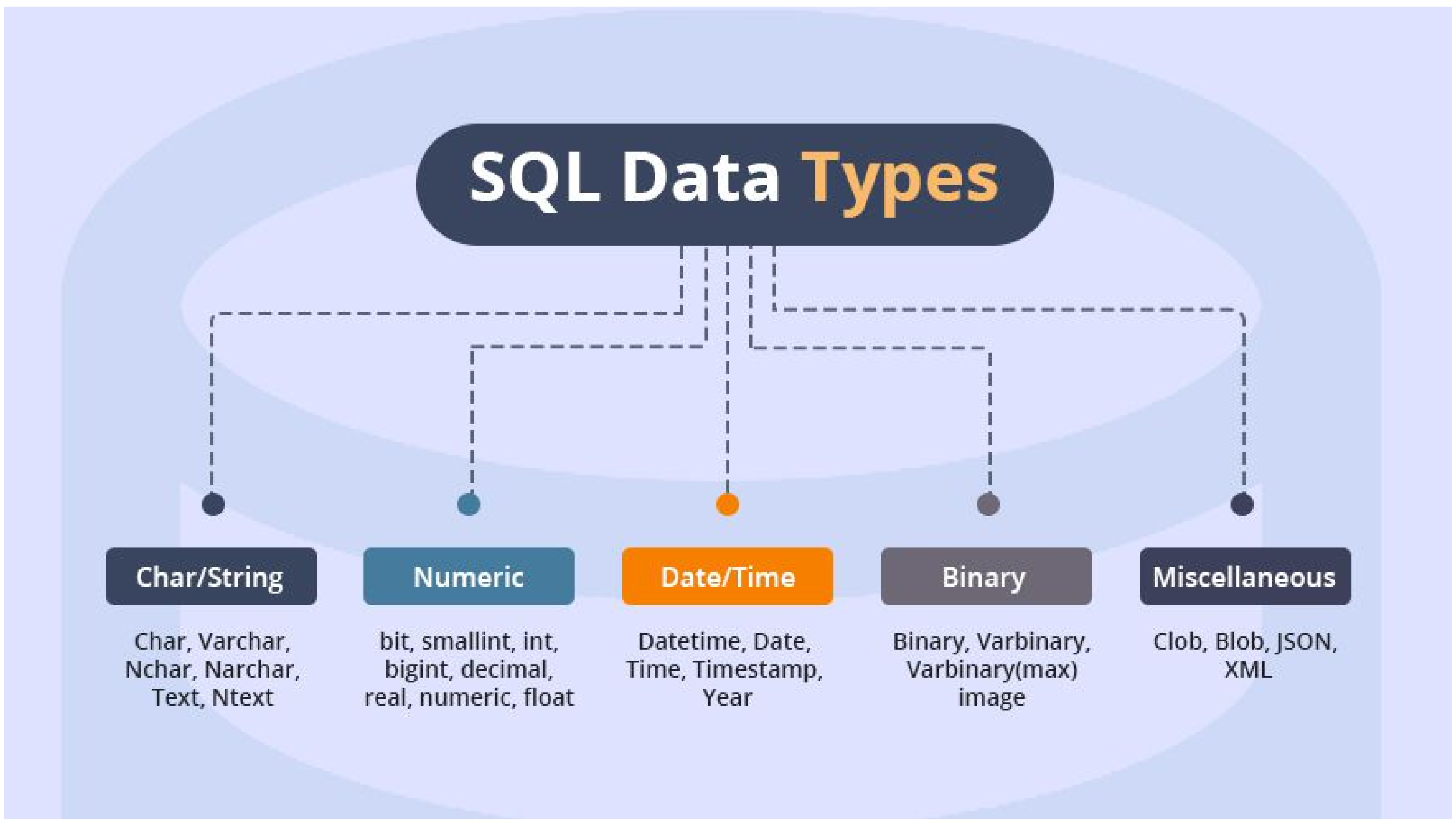
);

To delete this table, we would use the **DROP** command as follows:

DROP TABLE customers;

SQL Data Types for Server

To create tables for manipulating data effectively, we need to work with the correct data type. Let's say we want to work with dates, it would be easier to create a column for holding DATE type values instead of storing them as a string and writing logic to manipulate the values.


SQL Data Types
Char/String
Char, Varchar, Nchar, Narchar, Text, Ntext
Numeric
bit, smallint, int, bigint, decimal, real, numeric, float
Date/Time
Datetime, Date, Time, Timestamp, Year
Binary
Binary, Varbinary, Varbinary(max) image
Miscellaneous
Clob, Blob, JSON, XML

Every RDBMS is different and each RDBMS might have a different data type for working with certain values. The following sections of this SQL Cheat Sheet contain the various types of MySQL data types.

String Data Types

Data Type	Description
CHAR(size)	A fixed-length string that can contain numbers, letters, and special characters. The string length is from 0 - 255
VARCHAR(size)	Variable-length string similar to CHAR(), but with a length from 0 to 65535.
TEXT(size)	Holds a string of a maximum of 65,536 bytes.
TINY TEXT	Holds a string of a maximum of 255 characters.
MEDIUM TEXT	Holds a string with a maximum length of 16,777,215 characters.
LONG TEXT	Holds a string with a maximum length of 4,294,967,295 characters.
BINARY(size)	Similar to CHAR() but stores binary byte strings.
VARBINARY(size)	Similar to VARCHAR() but for binary byte strings.
BLOB(size)	For holding blobs of up to 65,536 bytes.
TINYBLOB	Used for BLOBs (Binary Large Objects). Has a max length of 255 bytes.
MEDIUMBLOB	Holds blobs of up to 16,777,215 bytes.
LONGBLOB	Holds blobs of size up to 4,294,967,295 bytes.
ENUM(val1,val2,…)	A string object that can have only one value is chosen from a list of possible values of up to 65535 values in an ENUM list. If the value inserted is not in the list, a blank value will be inserted.
SET(val1,val2,…)	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 Data Types

Data Type	Description
BIT(size)	A bit-value type. The size parameter can hold a value from 1 to 64. The default value for size is 1.
INT(size)	A medium integer with a signed range of -2147483648 to 2147483647, and an unsigned range from 0 to 4294967295.
TINYINT(size)	A very small integer. The signed range is from -128 to 127. The unsigned range is from 0 to 255.
SMALLINT(size)	A small integer. The signed range is from -32768 to 32767. The unsigned range is from 0 to 65535.
MEDIUMINT(size)	A medium integer. The signed range is from -8388608 to 8388607. The unsigned range is from 0 to 16777215.
BIGINT(size)	A large integer. The signed range is from -9223372036854775808 to 9223372036854775807. The unsigned range is from 0 to 18446744073709551615.
BOOL/BOOLEAN	Zero values are considered as FALSE and non-zero values are considered as TRUE.
FLOAT(p)	A floating-point value. If the precision parameter(p) is between 0 to 24, the data type is set to FLOAT(), and if it's from 25 to 53, the data type is set to DOUBLE(). This makes the storage of values more efficient.
DOUBLE(size,d)	A floating-point number value where the total digits are set by the size parameter, and the number of digits after the decimal point is set by the d parameter.
DEC(size,d)/ DECIMAL(size,d)	An exact fixed-point number with the total number of digits set by the size parameters, and the total number of digits after the decimal point set by the d parameter. For size, the maximum number is 65 and the default is 10, while for d, the maximum number is 30 and the default being 10.

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

Date/Time Data Types

Data Type	Description
DATE	A simple date in YYYY-MM–DD format, supporting a range from '1000-01-01' to '9999-12-31'.
TIME(fsp)	A time in hh:mm:ss format, with a supported range from '-838:59:59' to '838:59:59'
DATETIME(fsp)	A date and time combination in YYYY-MM-DD hh:mm:ss format. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP(fsp)	A Unix Timestamp, which is a value relative to the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). This has a supported range from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
YEAR	A year in four-digit format with the range as - 1901 to 2155

CRUD Operations with SQL

Now that we have our table(s) ready, let's see how we can store and retrieve data from the tables in our database. In the following sections of this SQL Basics Cheat Sheet, we would look at the most basic SQL operations.

CRUD is an acronym that stands for Create, Read, Update, and Delete. These are the most fundamental operations that one can perform on any database. For creating any application, these 4 types of operations are crucial. They are:-

1. INSERT (Create)
2. SELECT (Read)
3. UPDATE (Update)
4. DELETE (Delete)

INSERT

To insert data into any table, we use the **INSERT INTO** statement. The general syntax for insert is:

INSERT INTO table_name(column1,column2,...)

VALUES(val1,val2,...);

To insert data into our customer's table, we would use the following statement:

INSERT INTO customers(**ID**,name,phone,postalCode)

VALUES(1,'Alice','+123456789',123456);

SELECT

To read data from a table, we would use the Select statement where we define the columns that we want to fetch. The general syntax is:

SELECT column1,column2,.. **FROM** table_name;

If we wanted to select the name and phone number of a customer from our table, we would use:

SELECT name, phone **FROM** customers;

Also, to read all the columns from our table, we can replace the column names with * as follows:

SELECT * **FROM** customers;

UPDATE

To update specific column(s) of specific row(s), we make use of the Update statement. The general syntax for an update statement is:

```
UPDATE table_name
 SET column1=value1,column2=value2,...
 WHERE conditions...;
```

For example, if we wanted to update the phone number of a customer that has an ID of 2, we would write our query as:

```
UPDATE customers
 SET phone='+2223334445'
 WHEREID=2;
```

We can update multiple columns by adding them to the SET statement and we can target multiple rows by adding them to the WHERE statement. We will look at WHERE in detail in later sections of this SQL commands cheat sheet.

DELETE

If we wanted to remove some rows from a table, we would use the delete statement. The general syntax is:

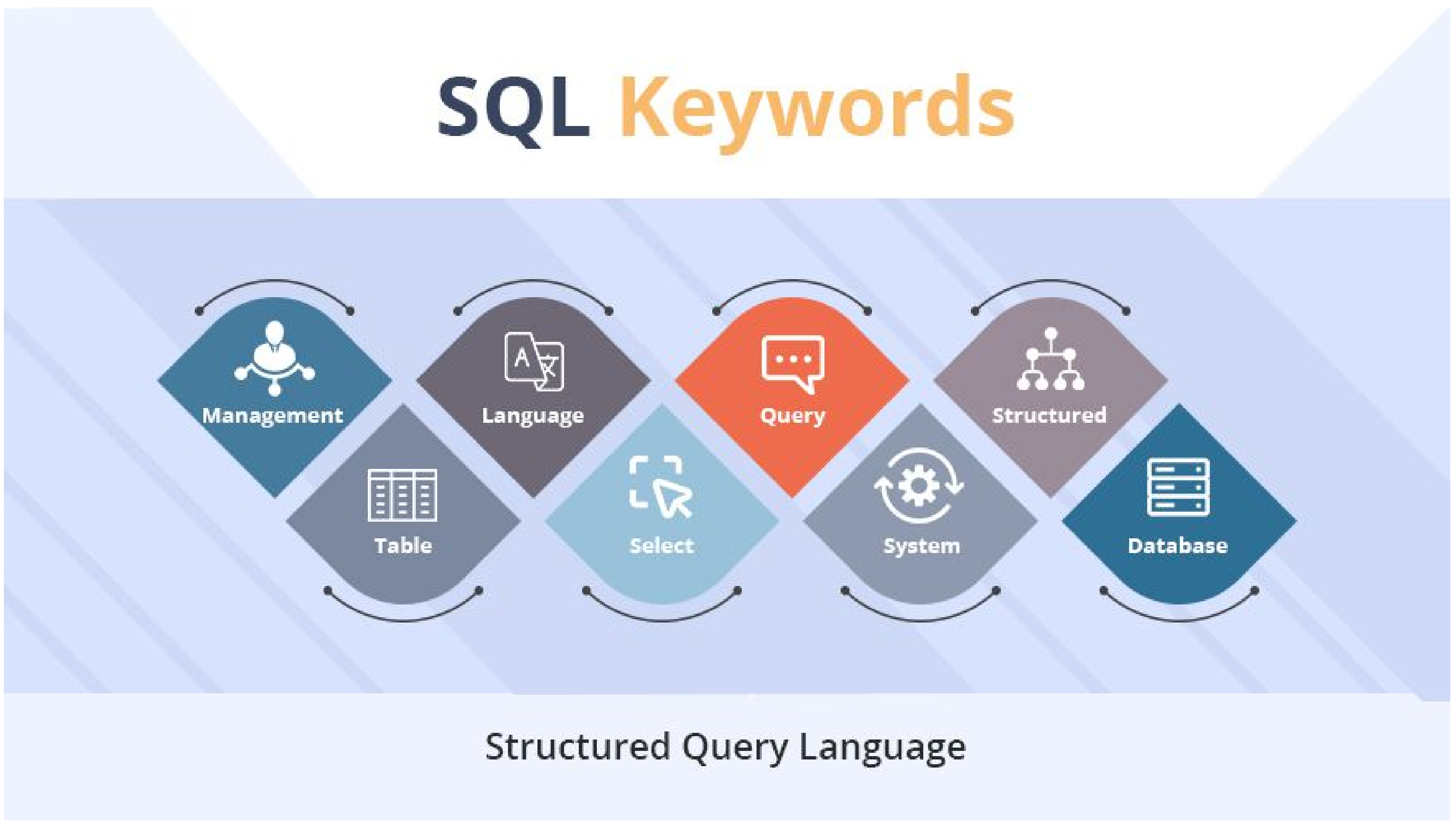
```
DELETE FROM table_name
  WHERE condition...;
```

Let's say we want to remove all the customers who live in a particular area. So, we simply delete those rows that have a specific area code:

```
DELETE FROM customers
  WHERE postalCode=223344;
```

List of useful SQL Keywords

In the following section of this SQL Server Cheat Sheet, we will have a look at all the commands/keywords that we can use in SQL to work with data, tables, and databases.



Keyword	Description
ADD	Add a new column to an existing table. Eg: ALTER TABLE customers ADD email_address VARCHAR(255);
ALTER TABLE	Adds, deletes, or edits columns/constraints in a table. Eg: ALTER TABLE customers DROP COLUMN email_address;
ALTER COLUMN	Changes the data type of a table's column. Eg: ALTER TABLE customers ALTER COLUMN phone varchar(50)
AS	Renames a table or column with an alias value that only exists for the duration of the query. Eg: SELECT name AS customer_name, phone, postalCode FROM customers;
ASC	Used with ORDER BY to return the data in ascending order.
CHECK	Adds a constraint that limits the value which can be added to a column. Eg: CREATE TABLE Users(firstName varchar(255),age INT, CHECK(age>10));
CREATE DATABASE	Creates a new database. Eg: CREATE DATABASE my website;
CREATE TABLE	Creates a new table. Eg: CREATE TABLE users (id int,firsr_name varchar(255), surname varchar(255), address varchar(255), contact_number int);
DEFAULT	Set the default value for a column. Eg: CREATE TABLE products(ID int, name varchar(255) DEFAULT 'Username', from date DEFAULT GETDATE());
DELETE	Delete values from a table. DELETE FROM users WHERE user_id= 674;
DESC	Used with ORDER BY to return the data in descending order.
DROP COLUMN	Deletes a column from a table. ALTER TABLE users DROP COLUMN first_name;

DROP DATABASE	Deletes a complete database along with all the tables and data inside. Eg: DROP DATABASE my website;
DROP DEFAULT	Removes a default value for a column. Eg: ALTER TABLE products ALTER COLUMN name DROP DEFAULT;
DROP TABLE	Delete a table from a database. Eg: DROP TABLE customers;
FROM	Specifies which table to select or delete data from. Eg: SELECT * FROM customers;
IN	Used with a WHERE clause as a shorthand for multiple OR conditions. Eg: SELECT * FROM users WHERE country IN('USA', 'United Kingdom','Russia');
IS NULL	Tests for empty (NULL) values. Eg: SELECT * FROM users WHERE phone IS NULL;
IS NOT NULL	Opposite of IS NULL. Tests for values that are not null.
LIKE	Returns true if the operand value matches a pattern. SELECT * FROM users WHERE first_name LIKE '%son';
ORDER BY	Used to sort the resultant data in ascending (default) or descending order.
SELECT DISTINCT	Same as SELECT, except duplicate values are excluded. Eg: SELECT DISTINCT postalCode from customers;
TOP	Used alongside SELECT to return a set number of records from a table. Eg: SELECT TOP 5 * FROM customers;
VALUES	Used alongside the INSERT INTO keyword to add new values to a table. Eg: INSERT INTO cars (name, model, year) VALUES ('Ford', 'Fiesta', 2010);
WHERE	Filters result only includes data that meets the given condition. SELECT * FROM orders WHERE quantity > 1;

Operators in SQL

SQL has various operators that allow you to manipulate and compare multiple values. These are very useful and handy while creating queries.

SQL Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater Than
<	Less Than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not Equal to

SQL Compound Operators

Operator	Description
+=	Add Equals
-=	Subtract Equals
*=	Multiply Equals
/=	Divide Equals
%=	Modulo Equals
&=	Bitwise AND Equals
^-=	Bitwise Exclusive Equals
|*=	Bitwise OR Equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

SQL Keys

In a database, different tables store different values and these values are related to each other. To identify each row uniquely, we make use of SQL keys. An SQL key is either a single column (or attribute) or a group of columns that can uniquely identify rows in a table. SQL Keys ensures that there aren't any rows with duplicate values.

However, the most powerful use of keys is to establish relations between multiple tables in a database. To do so, we need to understand Primary Key and Foreign Key. The following sections of this SQL cheatsheet explain both of these concepts.

Primary Key

It is a key that uniquely identifies a single row in a table. For example, in a customer's table, the ID key can be used as a primary key to uniquely identify a single customer. This key can then be used to fetch data from multiple tables that have data related to the customer.

Key Points:

1. There can only be One Primary Key for a Table.
2. Primary Key should be unique for Each Row.
3. Primary key cannot have Null Values.

Typically, the primary key in a table is the ID column and is usually paired with the AUTO_INCREMENT keyword to uniquely identify the row. To mark a column as the primary key, we use the PRIMARY KEY keyword followed by the column/columns that consist of the primary key. For Example: -

```
CREATE TABLE users (
    id int NOT NULL AUTO_INCREMENT,
    first_name varchar(255) NOT NULL,
    last_name varchar(255) NOT NULL,
    address varchar(255),
    email varchar(255) NOT NULL,
    PRIMARY KEY (id)
);
```

Foreign Key

A foreign key is a field in a table that references the PRIMARY KEY of another table. A foreign key is used to link two tables together by establishing a relationship.

The table that contains the foreign key is known as the child table, while the table containing the primary key for the foreign key is known as the parent table.

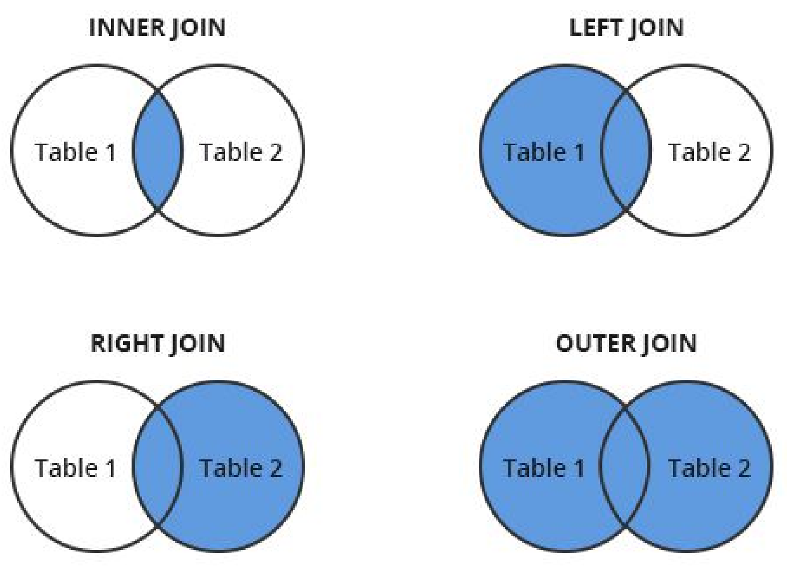
For example: Let's say we have 3 different tables to manage a restaurant - products, users, and orders. In our products table, we list all our products and in the user's table, we have details of all our users. When a user places an order, we save the data in the orders table. But, instead of saving the complete details of the product and all the information of the user, we save their primary keys in the orders table.

```
CREATE TABLE orders(
       order_id INT NOT NULL,
       user_id INT,
       product_id int,
       PRIMARY KEY(order_id),
       FOREIGN KEY(user_id) REFERENCES users(id),
       FOREIGN KEY(product_id) REFERENCES products(id)
    );
```

Here, we create a primary key for the order ID as it uniquely identifies an order. Also, we create two foreign keys that reference different primary keys.

SQL Joins

Once you understand Primary Key and Foreign Key, you can use joins to fetch data by combining multiple tables. Let's take the orders, customers, and products table as an example.



Products:

product_id	product_name	price
1	Burger	10
2	Sandwich	15

Customers:

customer_id	customer_name	email
1	Alice	alice@alice.com
2	Bob	bob@bob.com

Orders:

order_id	customer_id	product_id
1	1	1
2	1	2
3	2	1

We can join the orders table with customers and products table to get only the information that we require. Let's say we want to see all the orders with the customer's name, product name, and product price as follows:

order_id	product_name	customer_name	price
1	Burger	Alice	10
2	Sandwich	Alice	15
3	Burger	Bob	10

To do so, we would use join in SQL as follows:

```
SELECT orders.order_id, products.product_name,customers.customer_name,products.price
FROM orders
INNER JOIN products ON products.product_id = order.product_id
INNER JOIN customers on customers.customer_id = order.customer_id;
```

SQL has Different Types of Joins that Help Achieve Different Results:

1. INNER JOIN - Returns any records which have matching values in both tables.
2. LEFT JOIN - Returns all of the records from the first table, along with any matching records from the second table
3. RIGHT JOIN - Returns all of the records from the second table, along with any matching records from the first
4. FULL JOIN - Returns all records from both tables when there is a match

SQL Cheatsheet for SELECT Queries

1. Retrieve specific columns

SELECT userId,name,age,phone,country FROM Users;

2. Retrieve all Columns

SELECT * FROM Users;

3. Retrieve Filtered Rows

SELECT * FROM Users WHERE age>18;

4. Retrieve Distinct Rows

SELECT DISTINCT country from Users;

5. Count the Filtered Rows

SELECT COUNT(*) FROM users WHERE age>18;

6. Sort Rows Based on Criteria

SELECT * FROM Users ORDER BY userId ASC/DESC;

Note: You can use ASC for Ascending Order or DESC for descending order. If nothing is specified, sorting is done in Ascending order(ASC).

7. Retrieve Limited Rows
SELECT * FROM Users WHERE country='india' LIMIT 20;

8. Retrieve and Skip/Offset Rows

SELECT * FROM Users ORDER BY userId OFFSET 10 ROWS;

9. Get Average, Sum, Max, Min, etc. of Results

SELECT AVG(age) FROM USERS;

10. Get all Values from two Tables

SELECT * FROM Users INNER JOIN Wallets ON Users.walletId = Wallets.walletId;

Note:- We can use any type of join that we want. The condition via which we want to join the tables needs to be specified after the ON keyword.

11. Get selected values from two tables

```
SELECT us.userId,us.name,wall.walletId,wall.balance
FROM Users AS us
INNER JOIN Wallets AS wall
```

Note: - We use the AS keyword to give an alias to a table to make our SELECT statement shorter. We can even eliminate the AS keyword in this case and simply write the Alias after the table name.

SQL Cheatsheet for INSERT Queries

1. Insert All Values in Order a Table

INSERT INTO Users VALUES('**Kanak Infosystems**','sales@kanakinfosystems.com',9876543210);

2. Insert Selected Values in a Table

INSERT INTO Users(userName,email) VALUES('Kanak Infosystems','sales@kanakinfosystems.com');

3. Insert Multiple Rows

```
INSERT INTO User(userName) VALUES
('user1'),
('user2');
```

Note: - We separate each row with a pair of brackets followed by a comma.

SQL Cheatsheet for TABLE Queries

1. Create a New Table

```
CREATE TABLE Users(
        id INT PRIMARY KEY,
        userName VARCHAR(50),
        age INT DEFAULT 10
     );
```

2. Delete a Table

DROP TABLE Users;

3. Remove all Values from a Table

TRUNCATE TABLE Users;

4. Add a Column to the Table

ALTER TABLE Users ADD COLUMN country VARCHAR(20);

5. Remove a Column from a Table

ALTER TABLE Users DROP COLUMN country;

6. Rename a Table

ALTER TABLE Users RENAME TO Customers;

7. Rename a Column

ALTER TABLE Users RENAME userName to name;

SQL Cheat Sheet for UPDATE/DELETE Queries

1. Update Column Value for all Rows

UPDATE Users SET country='india';

2. Update Column Value for Selected Rows

UPDATE Users SET isEligible='true' WHERE age>=18;

3. Delete all Rows

DELETE FROM Users;

4. Delete Specific Rows

DELETE FROM Users WHERE age<18;

SQL Cheat Sheet for SELECT Filters

1. Filter by Multiple Matching Conditions

SELECT * FROM Users WHERE age>=18 AND country=’india’;

2. Filter Rows by Multiple Parallel Conditions

SELECT * FROM Users WHERE country=’india’ OR name LIKE ‘Kan%’;

3. Filter Rows Based on Values in a List

SELECT * FROM Users WHERE age IN (15,18,22,27);

4. Filter Rows with Values in a Range

SELECT * FROM Users WHERE age BETWEEN 25 AND 30

Conclusion

SQL is a definite requirement when you are trying to build an application of any size and scale. Learning SQL might be tough for beginners, but once you get the hang of it, it's just like thinking.

Make sure to bookmark this page and download the SQL cheat sheet pdf if you are working with SQL. If you can remember a particular operation or keyword, you can open up this SQL commands cheat sheet to get all the required information.