NoSQL db examples:

- Document model: These NoSQL databases replace the familiar rows and columns structure with a document storage model.
  Each document is structured, frequently using the JavaScript Object Notation (JSON) model. The document data model is associated with object - oriented programming where each document is an object (use cases: catalog).

- Graph model: This class of databases uses structures like data modes, edges and properties, making it easier to model relationships between entities in an application (use cases: recommendation engines).

- Key-value model: In this NoSQL database model, a key is required to retrieve and update data. The key-value data model is very simple and therefore scales well. However, this simplicity and scalability come at the cost of query complexity (use cases: shopping cart).

- Wide-column model: Wide-column stores use the typical tables, columns, and rows, but unlike relational databases (RDBs), columnal formatting and names can vary from row to row inside the same table. And each column is stored separately on disk.

CREATE DATABASE ¡database name¿;
SELECT * FROM ¡database name¿;
means select all columns
DROP DATABASE ¡database name¿;
deletes database
BACKUP DATABASE ¡database name¿
TO DISK = '¡path¿';
BACKUP DATABASE ¡database name¿
TO DISK = '¡path¿'
WITH DIFFERENTIAL;
Schema is kinda namespace inside database

```
CREATE SCHEMA <schema name>
AUTHORIZATION <owner name>;

AUTHORIZATION part is not necessary

ALTER SCHEMA <schema name>
TRANSFER <entity type>.<securable name>;

<entity type>. is not necessary

DROP SCHEMA <schema name>;

CREATE TABLE <table_name>(
```

```
    <column name> <column type>,
    <column name> <column type>,
    <column name> <column type>,
    ...
)

DROP TABLE <table name>;

TRUNCATE TABLE <table name>;

removes all rows but not table

ALTER TABLE <table name>
ADD <column name> <type>;

ALTER TABLE <table name>
DROP COLUMN <column name>;

ALTER TABLE <table name>
ALTER COLUMN <column name> <type>;


INSERT INTO <table name>
    (<column name 1>,  <column name 2>, ...)
VALUES
    (<value for col1>, <value for col2>, ...);

INSERT INTO <table name>
VALUES (<value for col1>, ...);

this one is used if you provide values for all columns

UPDATE <table name>
SET <column name> = <column value>, ...
WHERE <condition>;

if where is ommited update is for whole table

DELETE FROM <table name>
WHERE <condition>;

if where is ommited its same as truncate

SELECT <col name 1>, <col name 2>, ...
FROM <table name>;
```

```
SELECT DISTINCT <col name 1>, <col name 2>, ...
FROM <table name>;

only returns distinct values
```

Order of operations:

- from
- on
- join
- where
- group by
- having
- select
- distinct
- order by
- top

```
SELECT <* or coma separated column names>
INTO <new table name>
FROM <old table name>
WHERE <condition>;

copies data from one table to new one
where can be ommited

INSERT INTO <t2>
SELECT <* or names> FROM <t1>
WHERE condition;

copies data from one table to existing one

WHERE <condition>;
where clause is used to filter query
```

where clause operators:

- = eq, e.g. SELECT ¡col1¿, ... FROM ¡table¿ WHERE ¡col2¿ = ¡something¿;

- ¿ gt, example is same as for eq, and actually for other comparison operators
- ¡ lt
- ¿= ge
- ¡= le
- ¡¿ ne
- BETWEEN inclusive range, e.g. WHERE ¡col¿ BETWEEN ¡x0¿ AND ¡x1¿;
- LIKE pattern match, e.g. WHERE ¡col¿ LIKE

```
'a%'; % - matches anything,
_ - matches single char
```

- IN multiple allowed values, e.g. WHERE ¡col¿ IN (¡val1¿, ¡val2¿, ...);

AND, OR, NOT, can be used in WHERE clause
Logical Operators Precedence:

- Parentheses
- Mul div
- Sub Add
- NOT
- AND
- OR

ORDER BY is used to sort output SELECT ¡cols¿ FROM ¡table¿ ORDER BY ¡col1¿ ¡ASC—DESC¿,...;

¡ASC—DESC¿ can be skipped, default order is ASC;

SELECT TOP ¡number¿ PERCENT ¡cols¿ FROM ...;

only ¡number¿% of rows will be returned, if PERCENT is ommited than ¡number¿ of rows will be returned

SELECT ¡cols¿ FROM ¡table¿ ORDER BY ¡cols ASC—DESC¿ OFFSET ¡n¿ ¡ROW—ROWS¿ FETCH ¡FIRST—NEXT¿ ¡m¿ ¡ROW—ROWS¿ ONLY;

OFFSET skips n rows before starting to return FETCH returns m rows after offset

FIRST and NEXT are synonyms

FETCH is optiona OFFSET must be used with order by.

comparison for null: IS NULL IS NOT NULL

SELECT ¡col name¿ AS ¡col alias¿ FROM ¡table name¿ AS ¡table alias¿;

SELECT ¡col1¿ + ',' + ¡col2¿ + ' ' + ¡col3¿ AS ¡col4¿ FROM ¡table¿;

col1, 2 and 3 combined into one with different separators

SELECT ¡t1¿.¡c1¿, ¡t2¿.¡c1¿, ... FROM ¡table1¿ as ¡t1¿, ¡table2¿ as ¡t2¿; – this is a comment /* this is also a comment */