A SQL View is a virtual table backed by a SQL query. It carries much of the same functionality that a table has, but it's backed by a query, rather than a table in the database, and depending on how you have it set up, can pull in a wide variety of data.

Create a New View

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Update a View

A view always displays fresh data since the database engine recreates it each time, using the view's SQL statement. To refresh your view use the next code:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Rename a View

If you are dealing with multiple views at a time, it's best to give them distinctive names. Here's how that done:

```
RENAME TABLE view_name TO new_view_name;
```

Show All Views

To call up all current views for all tables from the database, use this snippet:

```
SHOW FULL TABLES
WHERE table_type = 'VIEW';
```

Delete a View

To delete a single view use the DROP command:

```
DROP VIEW [IF EXISTS] view_name;
```

You can also delete multiple views at a time:
Drop Multiple views: `DROP VIEW [IF EXISTS] view1, view2, ...;`

Type of Views

SIMPLE	COMPLEX
CREATED OUT OF SINGLE TABLE	MORE THAN ONE TABLE
NO AGGREGATE FUNCTIONS	CAN HAVE AGGREGATE
DMLS ARE PERMITTED	NOT ALWAYS

What are the advantages?

- It acts like a table, so most things you'd do with a table will work with a view
- if you're doing user level access control, you can give a user access to a view without giving them access to the tables behind it
- It allows you to keep the logic centralised, rather than repeating it in your code
- It can allow for massive performance improvements

What are the disadvantages?

- If done wrong, it can result in performance issues
- You may not be able to update the view, forcing you back to the original tables
- A view is one more moving piece that has to be maintained

1. Data Abstraction: Views provide an abstraction layer to the underlying tables, making it easier to work with complex data structures.
2. Security: Views can be used to restrict access to sensitive data by only allowing users to see a limited portion of a table.
3. Simplification of complex queries: Complex queries can be simplified by using views, making it easier to understand and maintain the code.
4. Reusability: Views can be used across multiple queries, reducing the amount of repetitive code.
5. Improved performance: Views can be optimised for specific use cases, leading to improved query performance.
6. Data consistency: Views can ensure consistent data by encapsulating business logic and enforcing data constraints.