Checking if the new version is compatible with the previous one

By Diya Neupane

Version control is an important software engineering practice. Softwares continuously evolve and checking if the new version contradicts with the previous version is a must since lack of version compatibility can result in issues like crashes and data corruption.
A new version is considered to be compatible with the old one if it ensures:

1. **Schema changes compatibility:**

   A new schema is said to be compatible with the old one if the following actions are not performed:

   Deletion of a column: Deletion of a column results in the queries including the column being obsolete.

   Renaming a column: Renaming a column results in failure in the execution of any query that updates the column.

   Changing data types: Changing the data types also causes failure in updating new values to the specific column.

   Making a nullable column non-nullable: This raises unnecessary errors for the column.

Therefore, these factors should be checked to ensure that the new version of the schema is compatible with the previous version or not.

```
old_schema= {
    "users":{
        "columns":{
            "id": {"type":"integer", "nullable":False},
            "name":{"type":"varchar", "nullable":False},
            "email":{"type":"varchar", "nullable":False},
            "phone":{"type":"varchar", "nullable":True}
        }
    },
    "incidents":{
        "columns":{
            "id":{"type":"integer", "nullable":False},
            "status":{"type":"varchar", "nullable":False},
            "title":{"type":"varchar", "nullable":False}
        }

    }
}
queries=[
    "ALTER TABLE users ADD COLUMN age integer",
    "ALTER TABLE users RENAME COLUMN name to full_name",
    "ALTER TABLE incidents DROP COLUMN title",
    "ALTER TABLE users ALTER COLUMN phone SET NOT NULL"
]

def check_schema_compatibility(old_schema , queries):
    for query in queries:
        query_lower = query.lower()
        if "rename column" or "drop column" or "set not null" or "alter column" in query_lower:
            return False
        else:
            return True
```

2. Dependencies compatibility:

Some new features added might require some new dependencies to be added or the present version of a dependency to be upgraded. A proper dependency management strategy should be followed to ensure that the requirements are checked and necessary actions are taken.

```
required_dependencies={
    "numpy":"1.18.0",
    "pandas":"1.0.0",
    "requests":"2.22.0"
}

current_dependencies={
    "numpy":"1.19.0",
    "pandas":"1.1.0",
    "requests":"2.21.0"
}


def check_dependencies_compatibility(required_dependencies , current_dependencies):
    for package, version in required_dependencies.items():
        current_version = current_dependencies.get(package)
        if current_version is None or current_version < version:
            return False
        else:
            return True
```

References

https://www.atlassian.com/git/tutorials/what-is-version-control

https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html