SQLAlchemy Sync vs Async Operations

Introduction

SQLAlchemy supports both synchronous (traditional) and asynchronous ORM/database access. While both provide similar APIs,
asynchronous usage is designed to work with Pythons async/await syntax and is backed by an async-compatible database driver like asyncpg (for PostgreSQL).

This document compares commonly used SQLAlchemy ORM methods and constructs between their sync and async counterparts.

1. join vs select.join

Sync:
```
  session.query(User).join(Address).all()
```

Async:
```
  result = await session.execute(select(User).join(Address))
  users = result.scalars().all()
```

Explanation:
- `join` is used to join tables in both sync and async.
- In async, you must use `select()` with `await session.execute(...)` and process results.

2. joinedload vs selectinload

Sync:
```
  from sqlalchemy.orm import joinedload
  session.query(User).options(joinedload(User.addresses)).all()
```

Async:
```
  from sqlalchemy.orm import selectinload
  result = await session.execute(select(User).options(selectinload(User.addresses)))
  users = result.scalars().all()
```

Explanation:
- `joinedload` and `selectinload` are eager loading techniques.
- Both can be used in async and sync, but `selectinload` is more performant with async due to async DB I/O parallelism.

3. filter vs where

SQLAlchemy Sync vs Async Operations

Sync:

```
    session.query(User).filter(User.name == "John").all()
```

Async:

```
    result = await session.execute(select(User).where(User.name == "John"))
    users = result.scalars().all()
```

Explanation:
- In sync, `filter()` is used on `query()`.
- In async, `where()` is used inside `select()`.

4. filter_by vs where with keyword args

Sync:

```
    session.query(User).filter_by(name="John").all()
```

Async:

```
    result = await session.execute(select(User).where(User.name == "John"))
    users = result.scalars().all()
```

Explanation:
- `filter_by` uses keyword args; `filter`/`where` uses expressions.
- No direct async equivalent for `filter_by`; use `where()` instead.

5. add vs add (same for both)

Sync:

```
    session.add(new_user)
    session.commit()
```

Async:

```
    async with async_session() as session:
        session.add(new_user)
        await session.commit()
```

Explanation:
- Both use `.add()`.
- Async requires `await session.commit()` and use of `async with` context.

6. delete vs delete (via execute)

Sync:

```
    session.delete(obj)
    session.commit()
```

Async:

```
    await session.delete(obj)
    await session.commit()
```

Or using SQL expression:

```
    await session.execute(delete(User).where(User.name == "John"))
    await session.commit()
```

Explanation:
- Object deletion with `session.delete(obj)` is similar.
- SQL-level delete uses `delete(...).where(...)` in both, with `await` in async.

7. Query all vs scalar fetch

Sync:

```
    session.query(User).all()
```

Async:

```
    result = await session.execute(select(User))
    users = result.scalars().all()
```

Explanation:
- Async `execute(select(...))` returns a `Result` object.
- Use `scalars().all()` to extract ORM instances.

8. get() vs get()

Sync:

```
    user = session.get(User, 1)
```

Async:

```
    async with async_session() as session:
        user = await session.get(User, 1)
```

Explanation:
- `get()` is available in both sync and async.

- Async requires `await`.

Conclusion

Most sync operations have async equivalents using `select()`, `await session.execute(...)`, and scalars extraction.
Key differences include:

- Sync uses `query()`, async uses `select()`.
- Always use `await` for async DB operations.
- Avoid blocking sync functions (e.g., `.all()`) in async context; use async syntax instead.