Data Modification & Security Fundamentals

<Practice with DML and security fundamentals>

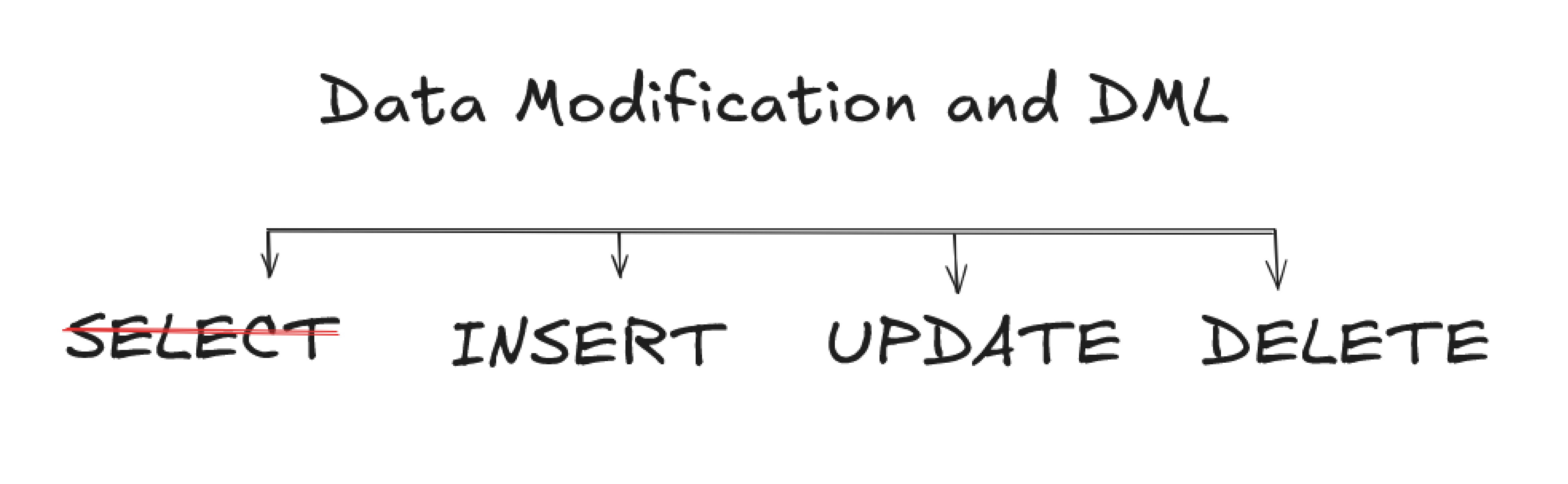
CODEUS_

The task has been prepared by Yevhen Yermolenko

Data Modification

<Let’s explore common data modifications commands>

Data modification



Data modification

INSERT

INSERT operation adds new record to the table

Data modification

INSERT

INSERT operation adds new record to the table

You can specify the columns and values explicitly

Data modification

INSERT

INSERT operation adds new record to the table

You can specify the columns and values explicitly

All constraints might be taken into an account

Data modification

INSERT

Samples:

```
INSERT INTO customers
VALUES (1, ‘Yevhen’, ‘Yermolenko’, ‘mail@gmail.com’,
‘0501234567’,  ‘Vinnytsia, Soborna, 1’);
```

Data modification

INSERT

Samples:

```
INSERT INTO customers
VALUES (1, ‘Yevhen’, ‘Yermolenko’, ‘mail@gmail.com’,
‘0501234567’,  ‘Vinnytsia, Soborna, 1’);
```

```
INSERT INTO customers (first_name, last_name)
VALUES ( ‘Yevhen’, ‘Yermolenko’);
```

Data modification

INSERT

Samples:

```
INSERT INTO customers
VALUES (1, ‘Yevhen’, ‘Yermolenko’, ‘mail@gmail.com’,
‘0501234567’,  ‘Vinnytsia, Soborna, 1’);
```

```
INSERT INTO customers (first_name, last_name)
VALUES ( ‘Yevhen’, ‘Yermolenko’);
```

```
INSERT INTO customers (first_name, last_name)
VALUES
            ( ‘Yevhen’, ‘Yermolenko’),
            (‘Taras’, ‘Shevchenko’),
            (‘Pes’, ‘Patron’);
```

Data modification

INSERT

Samples:

```
INSERT INTO customers
VALUES (1, ‘Yevhen’, ‘Yermolenko’, ‘mail@gmail.com’,
‘0501234567’,  ‘Vinnytsia, Soborna, 1’);
```

```
INSERT INTO customers (first_name, last_name)
VALUES ( ‘Yevhen’, ‘Yermolenko’);
```

```
INSERT INTO customers (first_name, last_name)
VALUES
            ( ‘Yevhen’, ‘Yermolenko’),
            (‘Taras’, ‘Shevchenko’),
            (‘Pes’, ‘Patron’);
```

```
INSERT INTO customers (first_name, last_name)
SELECT e.first_name, e.last_name FROM employees e
WHERE e.last_name = ‘Yermolenko’;
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id           SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)      NOT NULL,
   last_name    VARCHAR(50)      NOT NULL,
   email        VARCHAR(100) UNIQUE NOT NULL,
   phone        VARCHAR(20)  UNIQUE  NOT NULL,
   address      TEXT             NOT NULL,
   created_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id              SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)        NOT NULL,
   last_name    VARCHAR(50)        NOT NULL,
   email           VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT              NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name)
VALUES ('Yevhen', 'Yermolenko');
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id             SERIAL PRIMARY KEY,
   first_name  VARCHAR(50)       NOT NULL,
   last_name   VARCHAR(50)       NOT NULL,
   email          VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT              NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name)
VALUES ('Yevhen', 'Yermolenko');
```

```
[23502] ERROR: null value in column "email" of relation "customers" violates not-null constraint
Detail: Failing row contains (1, Yevhen, Yermolenko, null, null, null, 2025-03-23 16:13:05.15775)
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id             SERIAL PRIMARY KEY,
   first_name  VARCHAR(50)       NOT NULL,
   last_name   VARCHAR(50)       NOT NULL,
   email          VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT              NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id              SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)        NOT NULL,
   last_name    VARCHAR(50)        NOT NULL,
   email           VARCHAR(100) UNIQUE NOT NULL,
   phone           VARCHAR(20)  UNIQUE  NOT NULL,
   address         TEXT              NOT NULL,
   created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');
```

```
INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi');
```

Data modification

The data types smallserial, serial and bigserial are not true types, but merely a notational convenience for creating unique identifier columns (similar to the AUTO_INCREMENT).

INSERT

Samples:

```
CREATE TABLE customers (
   id          SERIAL PRIMARY KEY,
   first_name  VARCHAR(50)       NOT NULL,
   last_name   VARCHAR(50)       NOT NULL,
   email       VARCHAR(100) UNIQUE NOT NULL,
   phone       VARCHAR(20)  UNIQUE  NOT NULL,
   address     TEXT              NOT NULL,
   created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');

INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi');
```

```
[23505] ERROR: duplicate key value violates unique constraint "customers_pkey"
Detail: Key (id)=(1) already exists.
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id             SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)        NOT NULL,
   last_name    VARCHAR(50)        NOT NULL,
   email           VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT              NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');
```

```
INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi’)
ON CONFLICT DO NOTHING;
```

NOT RECOMMENDED

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id             SERIAL PRIMARY KEY,
   first_name     VARCHAR(50)       NOT NULL,
   last_name      VARCHAR(50)       NOT NULL,
   email          VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT              NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');

INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi’)
ON CONFLICT (id)
DO UPDATE SET address = ‘Moryntsi';
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id            SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)       NOT NULL,
   last_name    VARCHAR(50)       NOT NULL,
   email        VARCHAR(100) UNIQUE NOT NULL,
   phone        VARCHAR(20)  UNIQUE  NOT NULL,
   address       TEXT             NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');

INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi’)
ON CONFLICT (id)
DO UPDATE SET address = ‘Moryntsi'
RETURNING id, first_name, last_name;
```

Data modification

INSERT

Samples:

```
CREATE TABLE customers (
   id             SERIAL PRIMARY KEY,
   first_name   VARCHAR(50)       NOT NULL,
   last_name    VARCHAR(50)       NOT NULL,
   email          VARCHAR(100) UNIQUE NOT NULL,
   phone          VARCHAR(20)  UNIQUE  NOT NULL,
   address        TEXT             NOT NULL,
   created_at     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Yevhen', 'Yermolenko', 'email@e.com', '911', 'Vinnytsia');

INSERT INTO customers (id, first_name, last_name, email, phone, address)
VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi’)
ON CONFLICT (id)
DO UPDATE SET first_name = 'Taras', last_name = 'Shevchenko’
RETURNING id INTO conflict_logs;
```

Data modification

INSERT

Samples:

Enterprise insertion script example

```
DO $$
    DECLARE customer_exists INT;
    BEGIN
        SELECT COUNT(*) INTO customer_exists FROM customers WHERE id = 1;

        IF customer_exists > 0 THEN
            RAISE NOTICE 'Conflict detected: Employee ID already exists. Rolling back transaction.';
        ELSE
            INSERT INTO customers (id, first_name, last_name, email, phone, address)
            VALUES (1, 'Taras', 'Shevchenko', 't@e.com', 'n/a', 'Moryntsi');

            RAISE NOTICE 'Insert successful. Committing transaction.';
        END IF;
END $$;
```

Data modification

INSERT

INSERT INTO

VS

SELECT INTO

Data modification

INSERT

INSERT INTO

VS

SELECT INTO

INSERT INTO adds a new row to the existing table

Data modification

INSERT

INSERT INTO

VS

SELECT INTO

INSERT INTO adds a new row to the existing table

SELECT INTO creates a new table and populates it with data from an existing table base on the SELECT query.

Data modification

INSERT

SUMMARY:

- Use DEFAULT values for missing columns
- Validate data types before inserting
- Handle duplicate keys with ON CONFLICT
- Use Transactions for bulk inserts to ensure atomicity

Data modification

INSERT

UPDATE

The UPDATE command allows you to modify data in existing rows based on specific conditions

Data modification

INSERT

UPDATE

The UPDATE command allows you to modify data in existing rows based on specific conditions

The ALTER TABLE allows you to change the structure of an existing table, such as adding new columns or modifying data types

Data modification

The UPDATE command allows you to modify data in existing rows based on specific conditions

The ALTER TABLE allows you to change the structure of an existing table, such as adding new columns or modifying data types

Concurrent updates might result in Lost Updates Phenomena, use optimistic or pessimistic locking if needed

Data modification

INSERT

UPDATE

The UPDATE command allows you to modify data in existing rows based on specific conditions

The ALTER TABLE allows you to change the structure of an existing table, such as adding new columns or modifying data types

Concurrent updates might result in Lost Updates Phenomena, use optimistic or pessimistic locking if needed

Updates on large tables and on heavily indexed columns can decrease performance.

Data modification

INSERT

UPDATE

The UPDATE command allows you to modify data in existing rows based on specific conditions

The ALTER TABLE allows you to change the structure of an existing table, such as adding new columns or modifying data types

Concurrent updates might result in Lost Updates Phenomena, use optimistic or pessimistic locking if needed

Updates on large tables and on heavily indexed columns can decrease performance.

Complex data types must be updated with a proper update.

Data modification

INSERT

UPDATE

Samples:

```
UPDATE customers
SET first_name = ‘YEVHEN’,
      last_name =  ‘YERMOLENKO’
WHERE id = 1;
```

Data modification

INSERT

UPDATE

Samples:

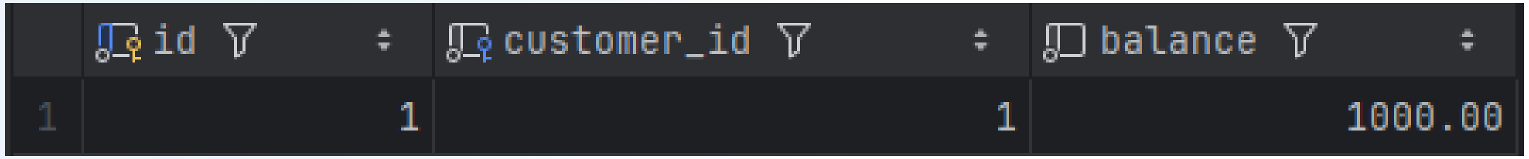
```
CREATE TABLE accounts (
   id                                SERIAL PRIMARY KEY,
   customer_id                       INT    NOT NULL,
   balance                           DECIMAL(15, 2) NOT NULL DEFAULT 0.00,
   FOREIGN KEY (customer_id)
   REFERENCES customers (id) ON DELETE CASCADE
);
```

Data modification

INSERT

UPDATE

Samples:



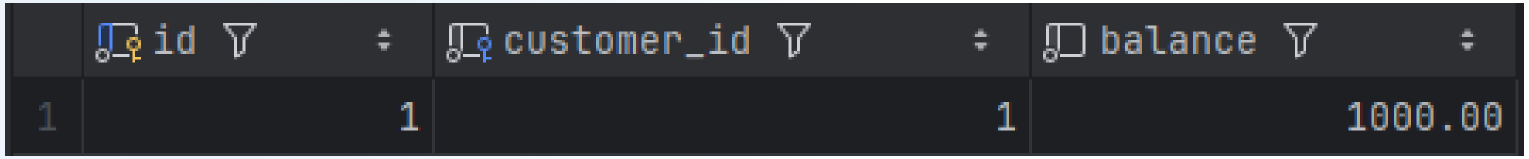
	id	customer_id	balance
1	1	1	1000.00

Data modification

INSERT

UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00

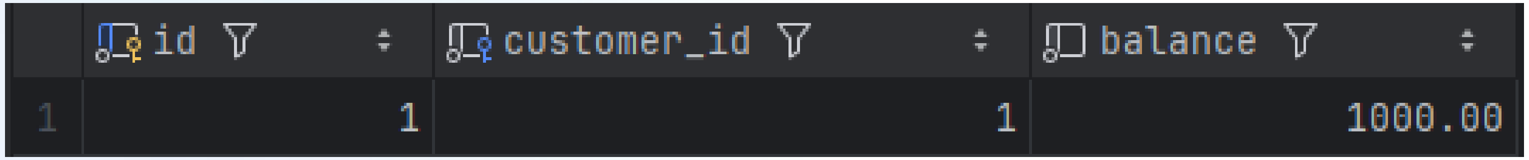
```
UPDATE accounts SET balance = 500
WHERE id = 10;
```

Data modification

INSERT

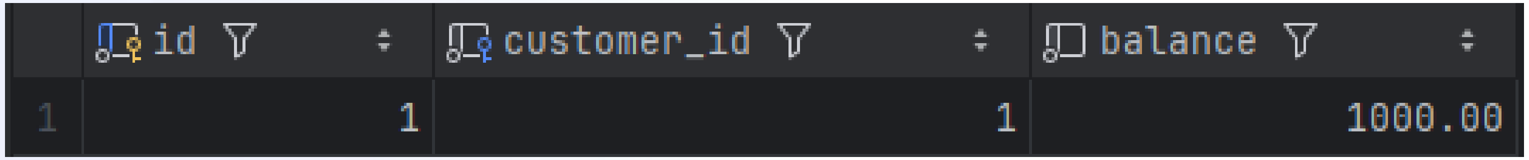
UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00

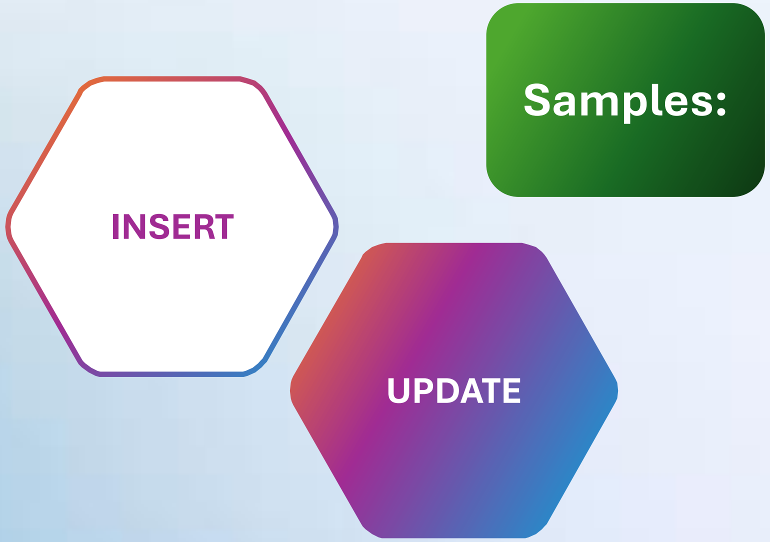
UPDATE accounts SET balance = 500
WHERE id = 10;

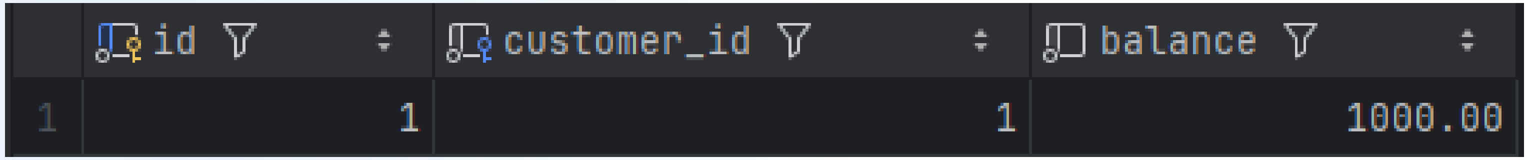


	id	customer_id	balance
1	1	1	1000.00

Update is successful but affects 0 rows. It might be a **silent failure**

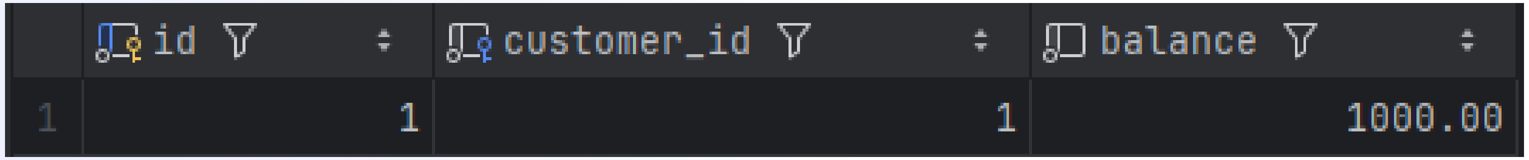
Data modification





	id	customer_id	balance
1	1	1	1000.00

UPDATE accounts SET balance = 500
WHERE id = 10;



	id	customer_id	balance
1	1	1	1000.00

Update is successful but affects 0 rows. It might be a silent failure

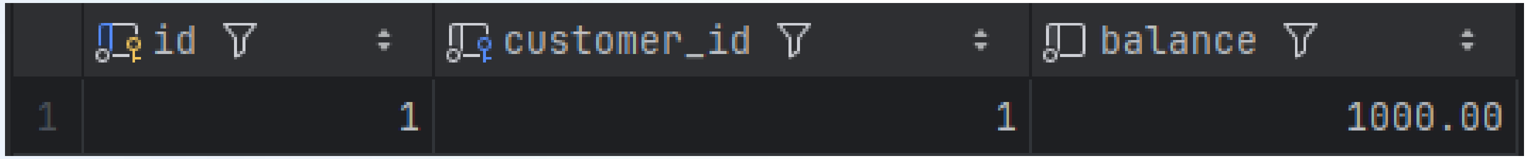
Check row count after update in application logic

Data modification

INSERT

UPDATE

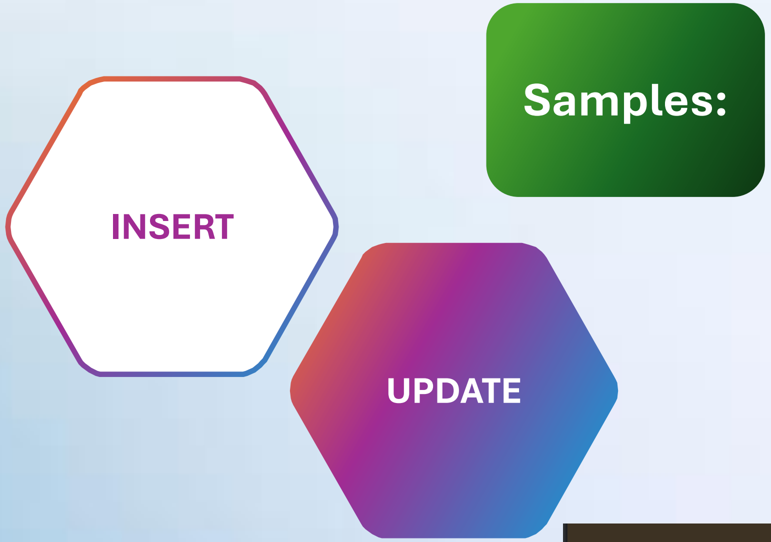
Samples:



	id	customer_id	balance
1	1	1	1000.00

```
UPDATE accounts SET balance = 500;
```

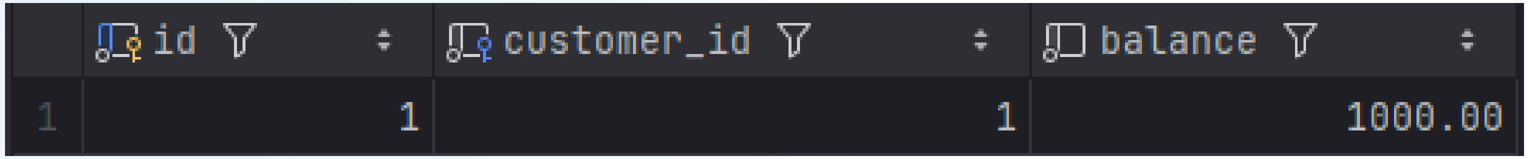
Data modification



INSERT

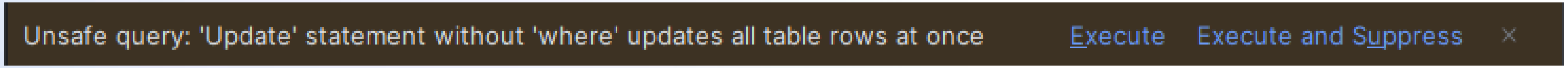
UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00

```
UPDATE accounts SET balance = 500;
```



Unsafe query: 'Update' statement without 'where' updates all table rows at once

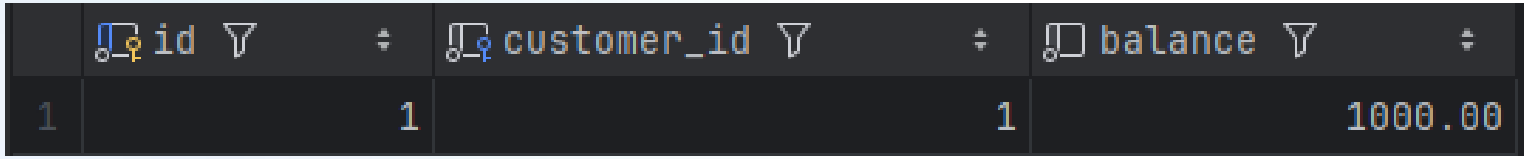
```
[2025-03-23 20:16:56] Unsafe query: 'Update' statement without 'where' updates all table rows at once
```

Data modification

INSERT

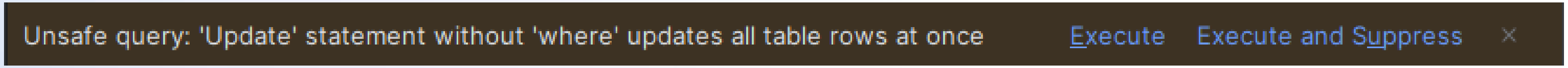
UPDATE

Samples:



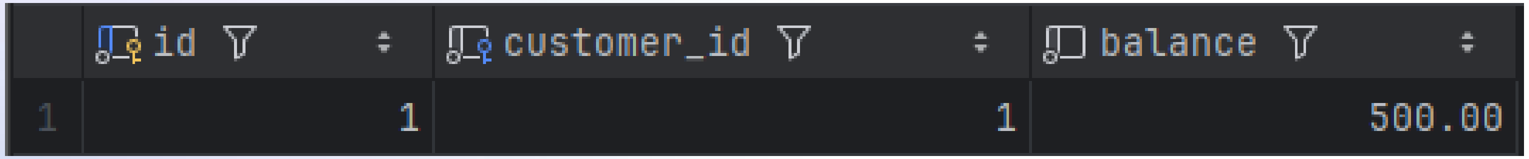
	id	customer_id	balance
1	1	1	1000.00

```
UPDATE accounts SET balance = 500;
```



Unsafe query: 'Update' statement without 'where' updates all table rows at once

```
[2025-03-23 20:16:56] Unsafe query: 'Update' statement without 'where' updates all table rows at once
```



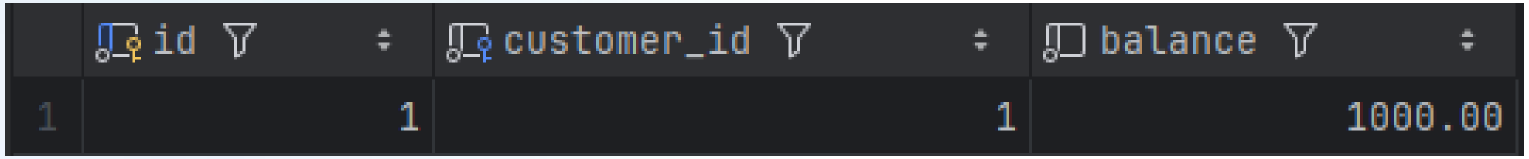
	id	customer_id	balance
1	1	1	500.00

Data modification

INSERT

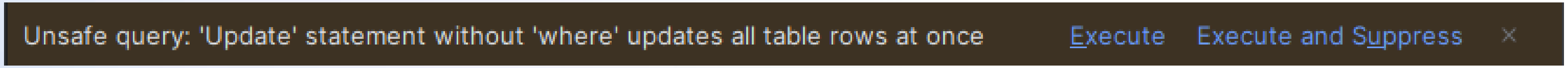
UPDATE

Samples:



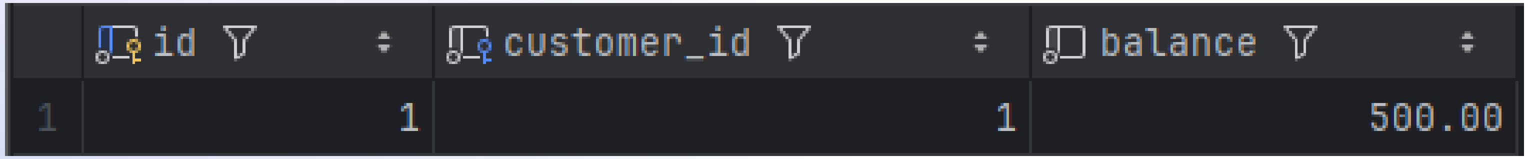
	id	customer_id	balance
1	1	1	1000.00

UPDATE accounts SET balance = 500;



Unsafe query: 'Update' statement without 'where' updates all table rows at once

```
[2025-03-23 20:16:56] Unsafe query: 'Update' statement without 'where' updates all table rows at once
```



	id	customer_id	balance
1	1	1	500.00

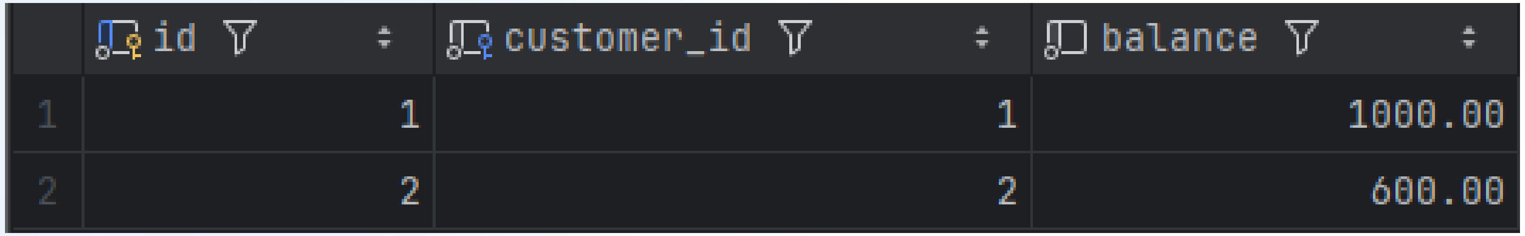
- Use update inside transaction;
- First run SELECT with the same WHERE condition to verify affected rows.

Data modification

INSERT

UPDATE

Samples:



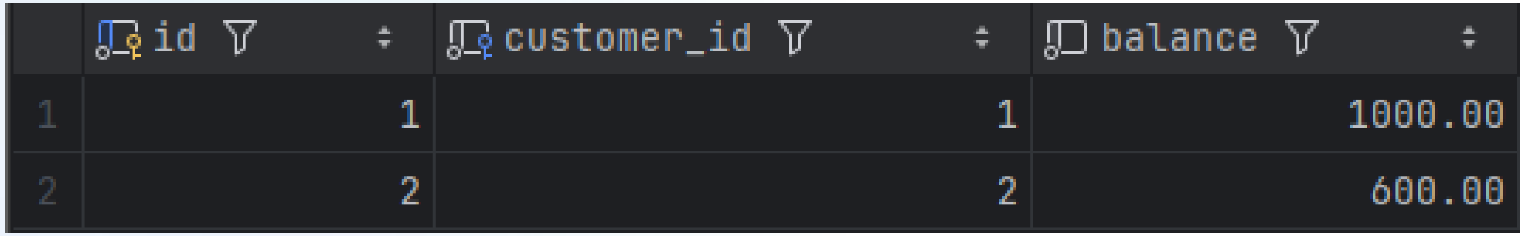
	id	customer_id	balance
1	1	1	1000.00
2	2	2	600.00

Data modification

INSERT

UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00
2	2	2	600.00

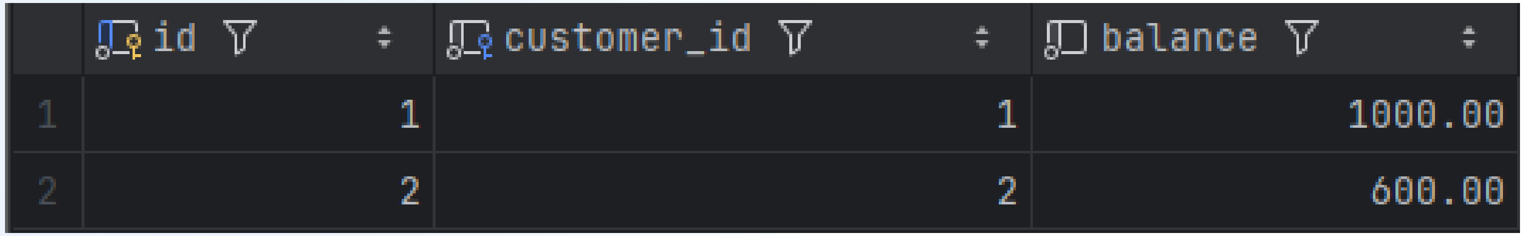
```
UPDATE accounts
SET balance = (SELECT balance FROM accounts WHERE id < 10);
```

Data modification

INSERT

UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00
2	2	2	600.00

```
UPDATE accounts
SET balance = (SELECT balance FROM accounts WHERE id < 10);
```

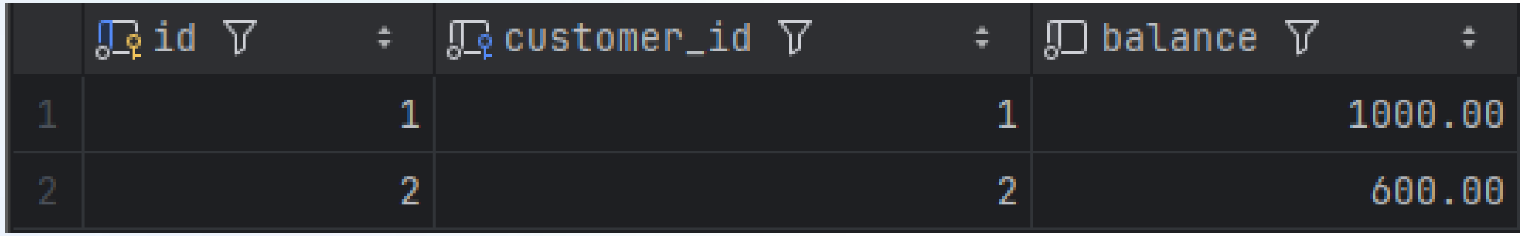
[21000] ERROR: more than one row returned by a subquery used as an expression

Data modification

INSERT

UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00
2	2	2	600.00

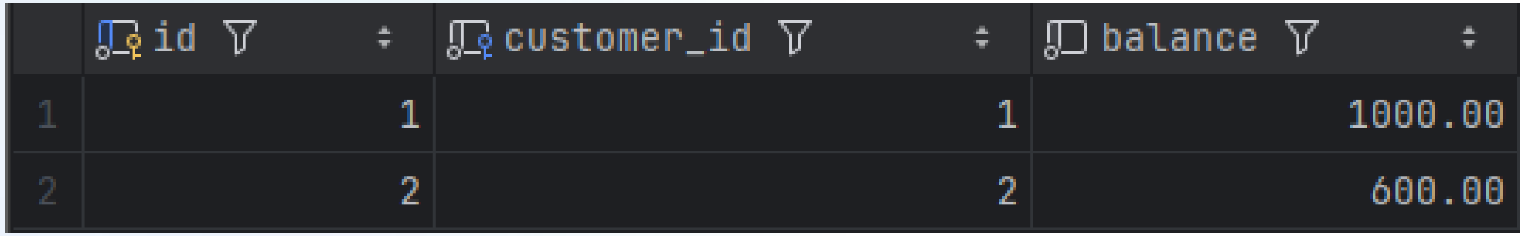
UPDATE accounts
SET balance = (SELECT **AVG**(balance) FROM accounts WHERE id < 10)
WHERE id < 10;

Data modification

INSERT

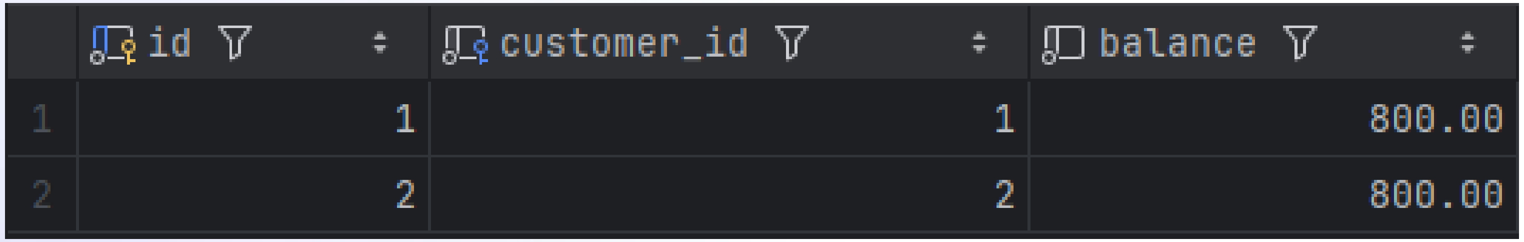
UPDATE

Samples:



	id	customer_id	balance
1	1	1	1000.00
2	2	2	600.00

```
UPDATE accounts
SET balance = (SELECT AVG(balance) FROM accounts WHERE id < 10)
WHERE id < 10;
```



	id	customer_id	balance
1	1	1	800.00
2	2	2	800.00

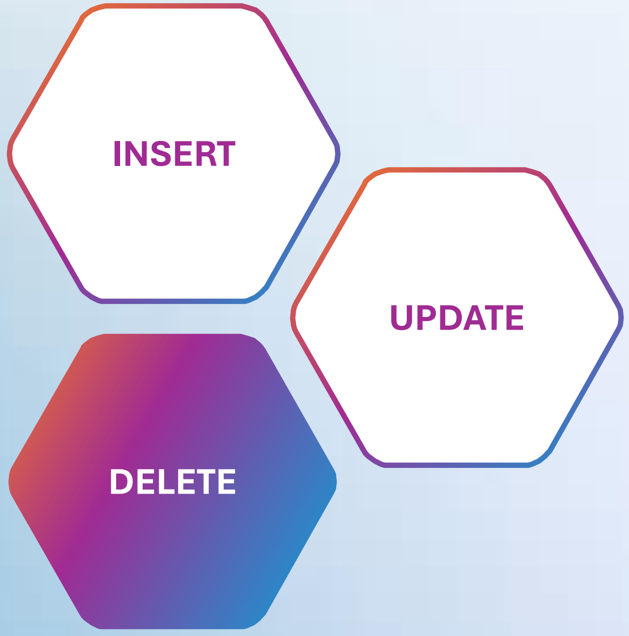
Data modification

SUMMARY

- Always use WHERE with UPDATE
- Do UPDATE inside transactions
- Do updates in batches for large amount of data
- Use SELECT to check condition before executing UPDATE
- Use proper updates for complex data structures:

```
... SET data = jsonb_set(data, ‘{balance}’, ‘100’) ...
```

Data modification



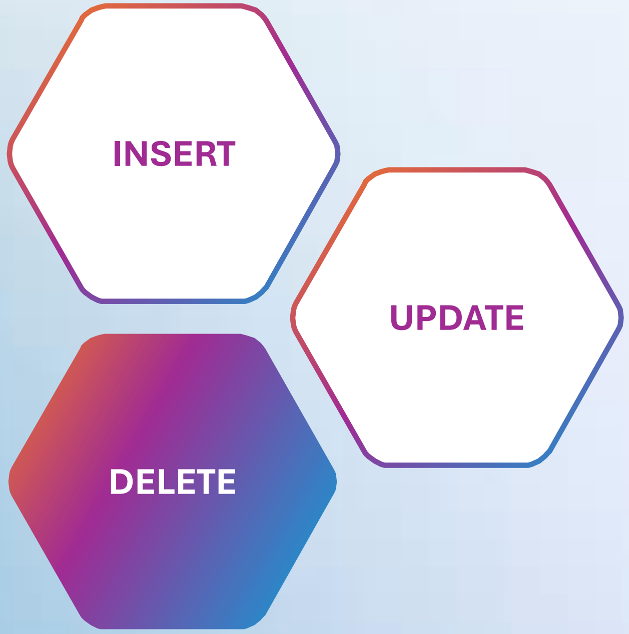
INSERT

UPDATE

DELETE

The DB will locate the rows to be deleted matching the WHERE condition by available indexes or table scans, check constraints violation, place locks

Data modification



INSERT

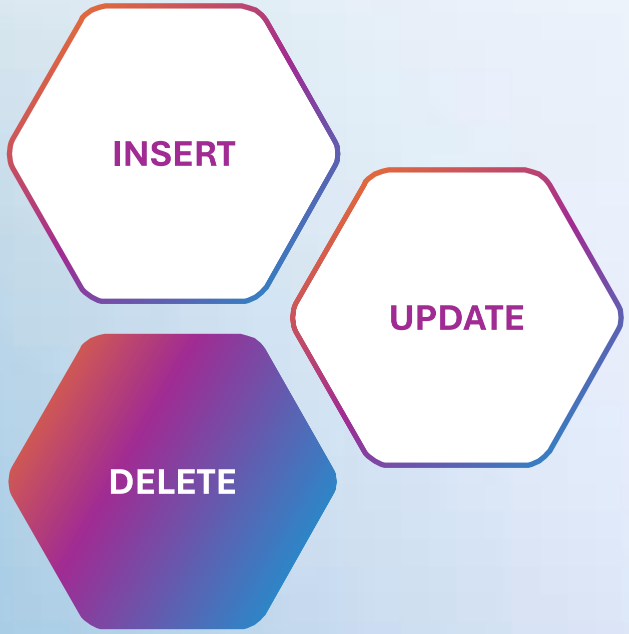
UPDATE

DELETE

The DB will locate the rows to be deleted matching the WHERE condition by available indexes or table scans, check constraints violation, place locks

Every delete operation generates transaction log records to ensure consistency

Data modification



INSERT

UPDATE

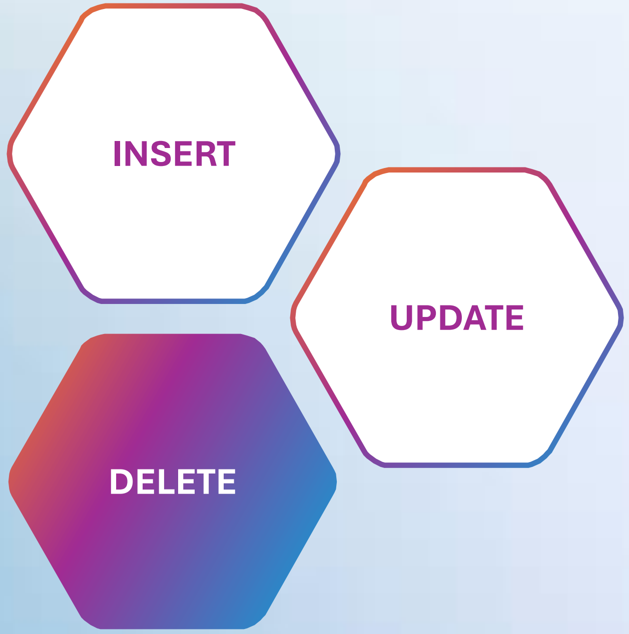
DELETE

The DB will locate the rows to be deleted matching the WHERE condition by available indexes or table scans, check constraints violation, place locks

Every delete operation generates transaction log records to ensure consistency

The DB add different level locks on specific rows to prevent concurrent modifications

Data modification



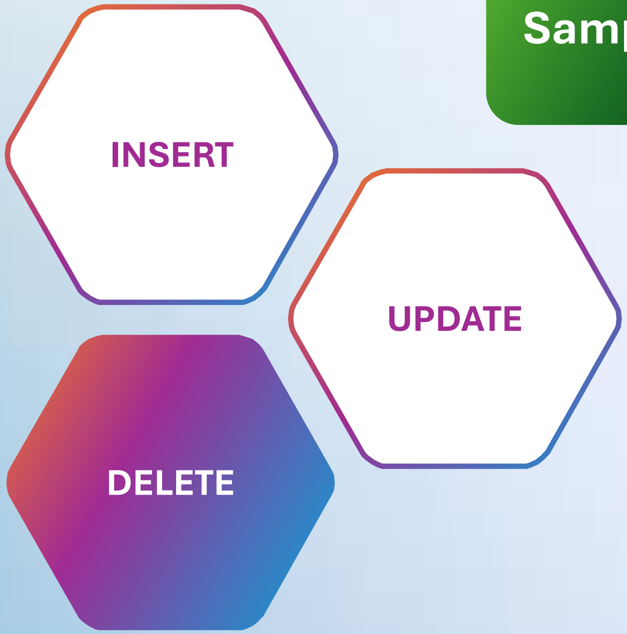
The DB will locate the rows to be deleted matching the WHERE condition by available indexes or table scans, check constraints violation, place locks

Every delete operation generates transaction log records to ensure consistency

The DB add different level locks on specific rows to prevent concurrent modifications

When rows are deleted, each index must be updated

Data modification

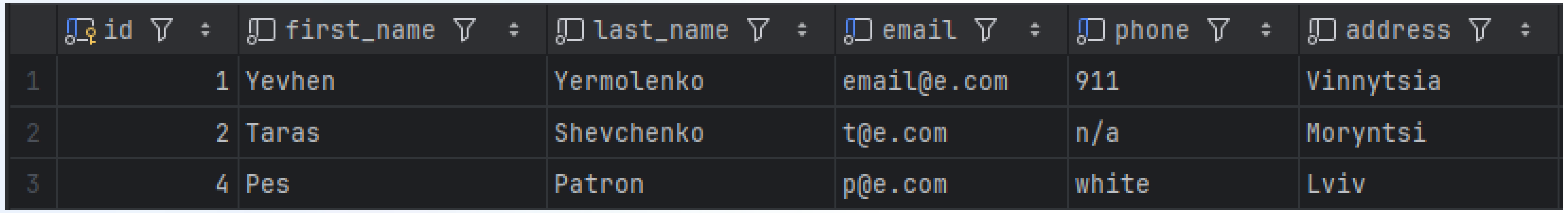


INSERT

UPDATE

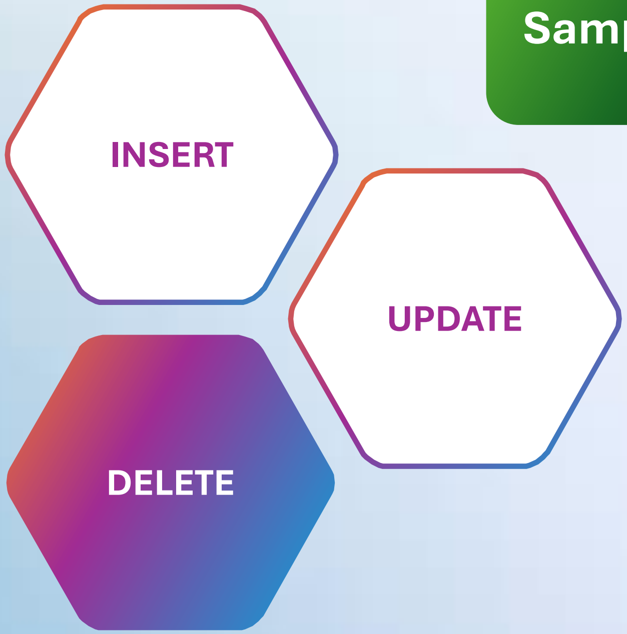
DELETE

Samples:

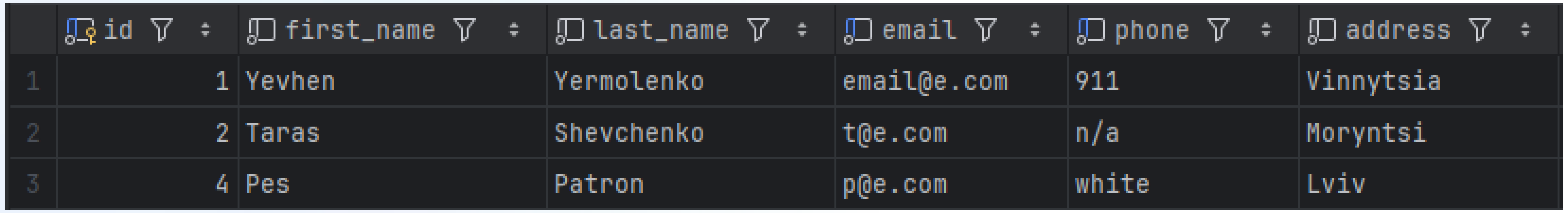


	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

Data modification



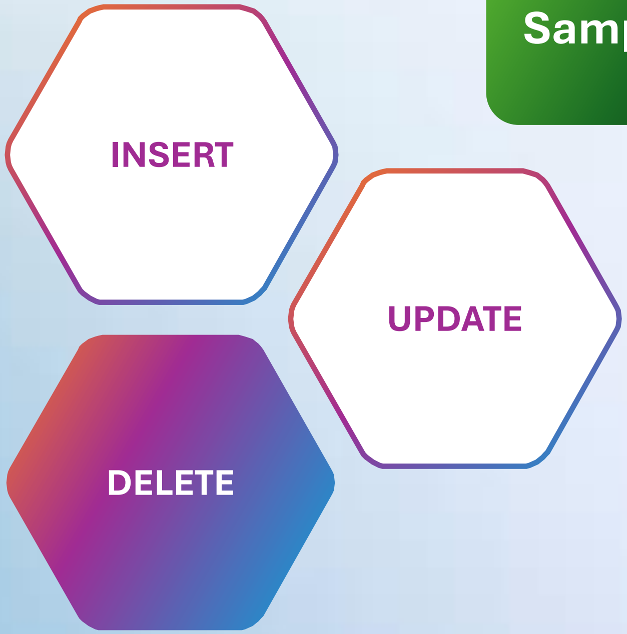
Samples:



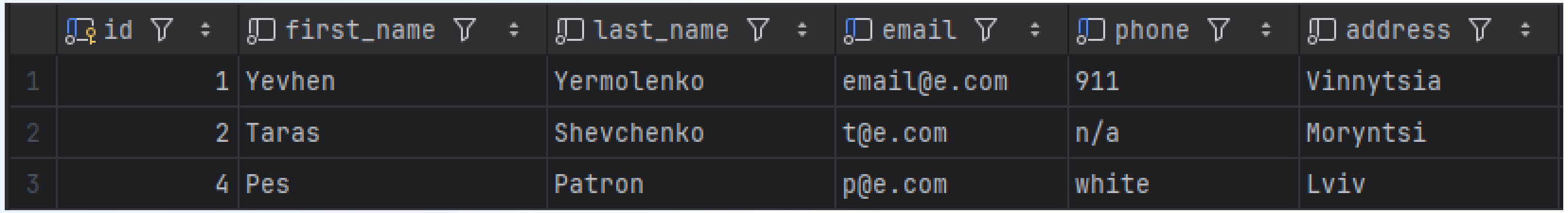
	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

DELETE FROM customers WHERE id = 123;

Data modification



Samples:

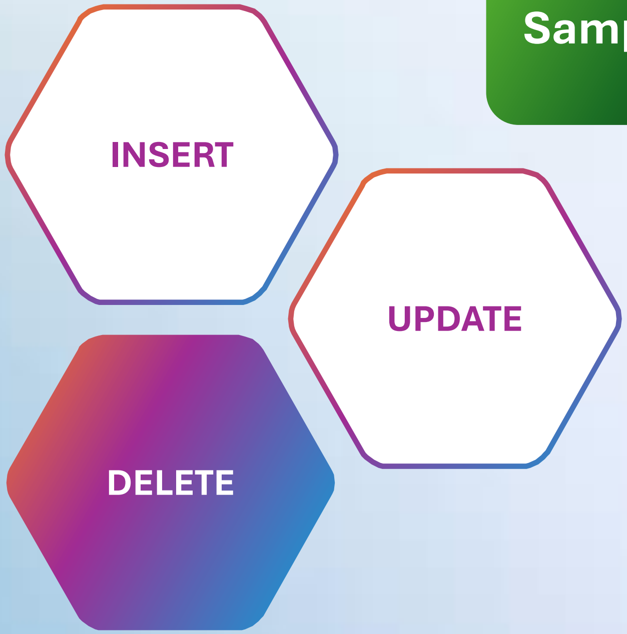


	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

DELETE FROM customers WHERE id = 123;

Delete runs without errors but affects 0 rows. It might be a silent failure

Data modification

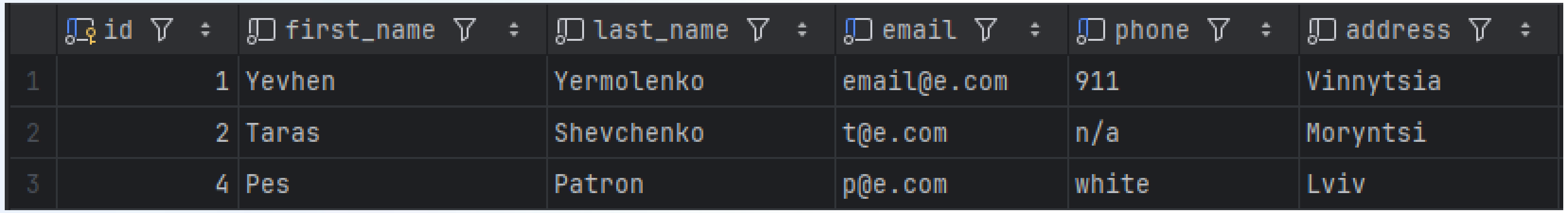


INSERT

UPDATE

DELETE

Samples:



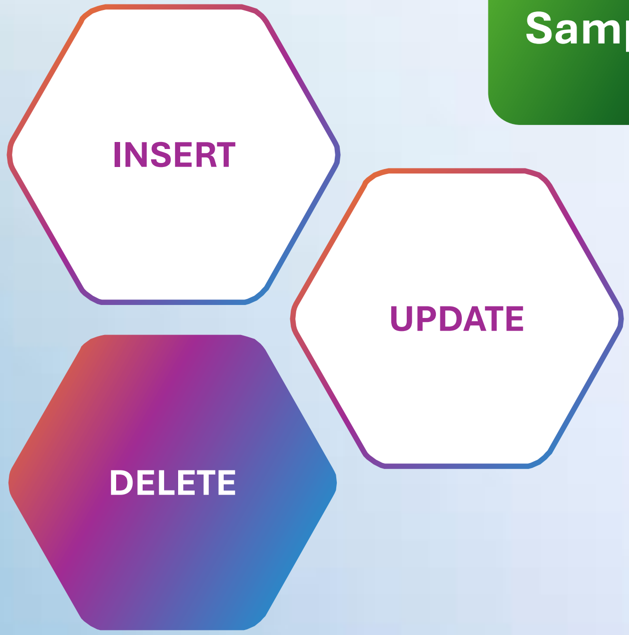
	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

DELETE FROM customers WHERE id = 123;

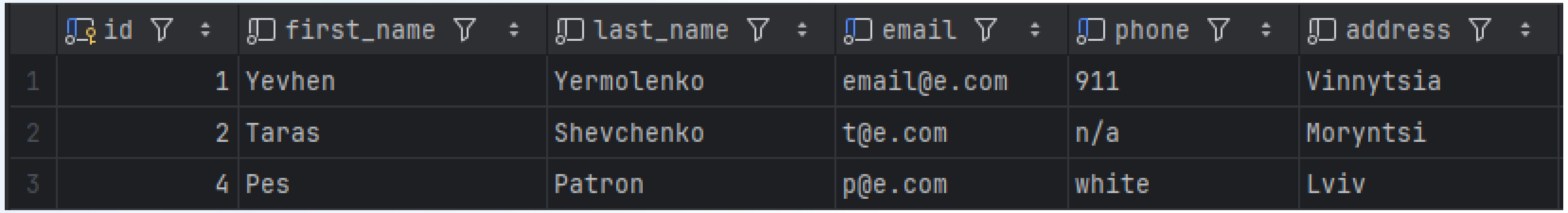
Delete runs without errors but affects 0 rows. It might be a silent failure

Check affected row count or use RETURNING

Data modification



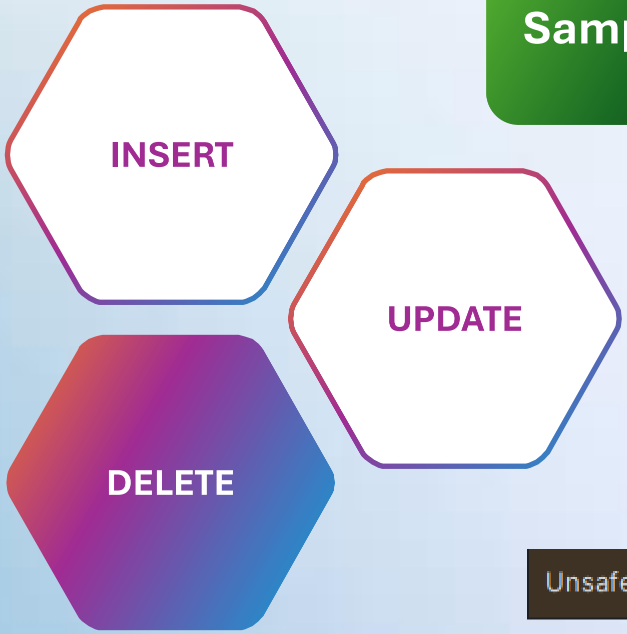
Samples:



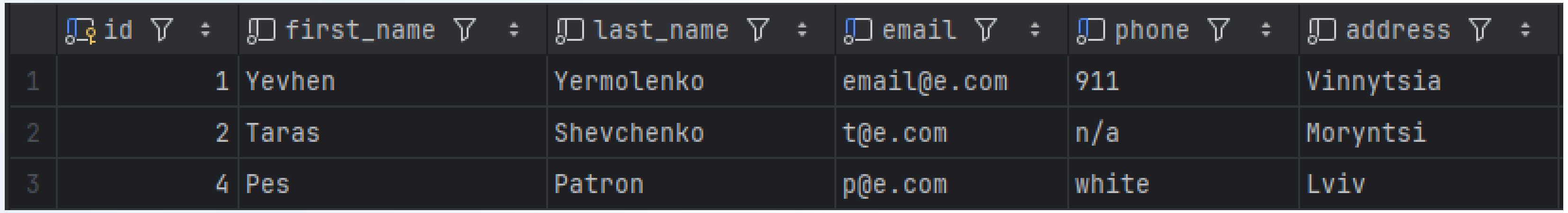
	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

DELETE FROM customers;

Data modification

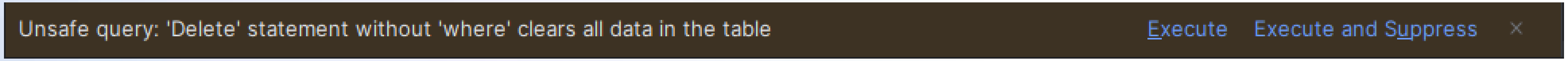


Samples:



	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

```
DELETE FROM customers;
```



Unsafe query: 'Delete' statement without 'where' clears all data in the table

Data modification

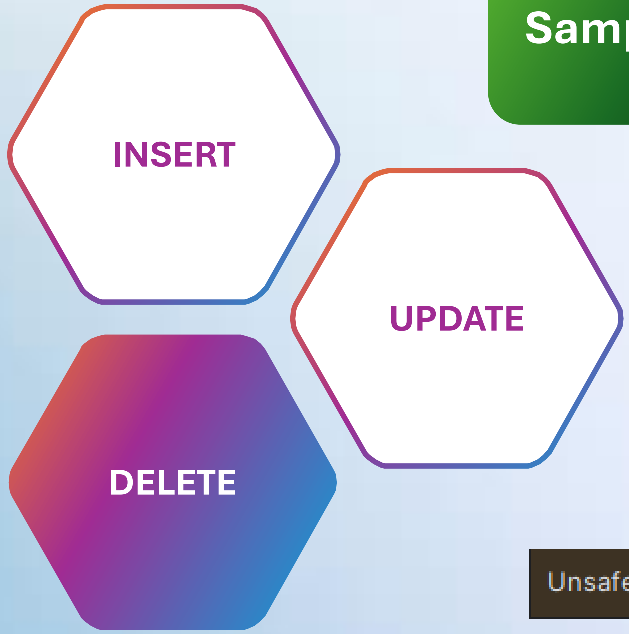
Samples:

INSERT

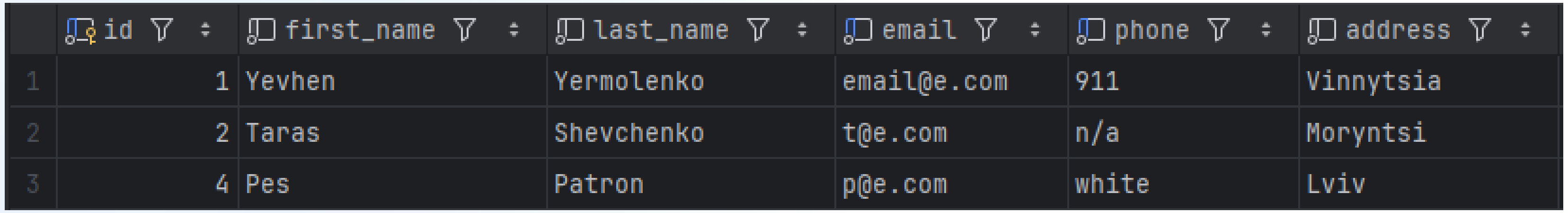
UPDATE

DELETE

Data modification

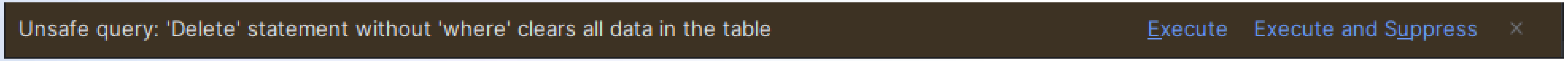


Samples:

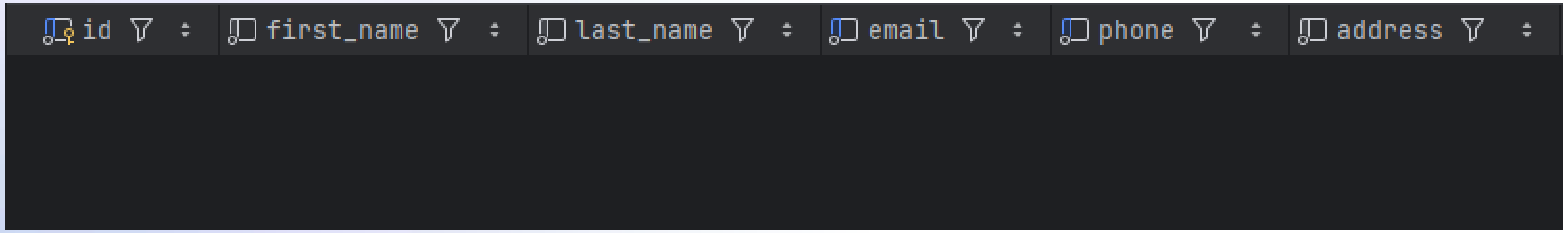


	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

DELETE FROM customers;

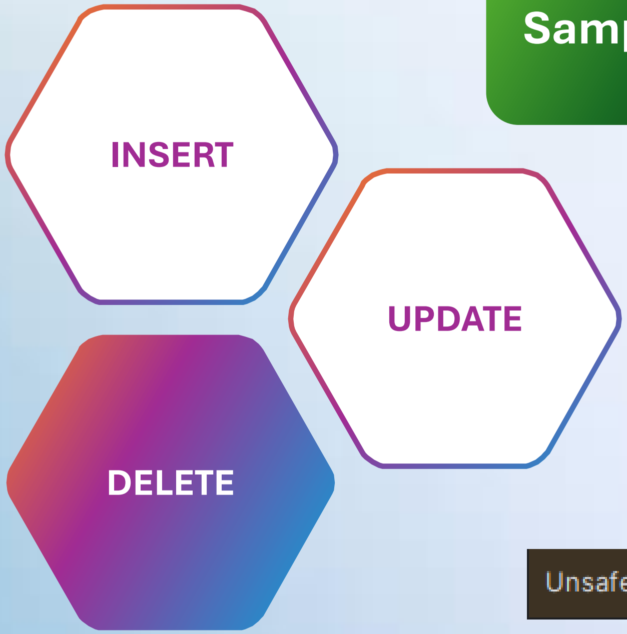


Unsafe query: 'Delete' statement without 'where' clears all data in the table

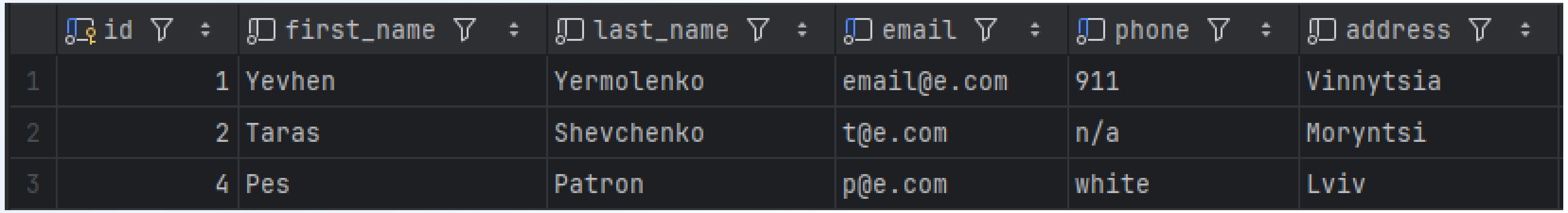


id	first_name	last_name	email	phone	address

Data modification

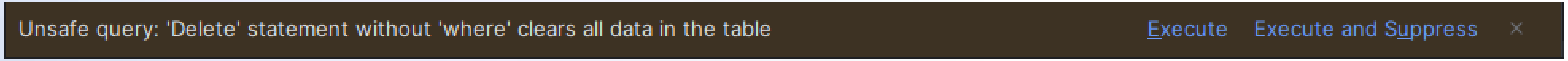


Samples:

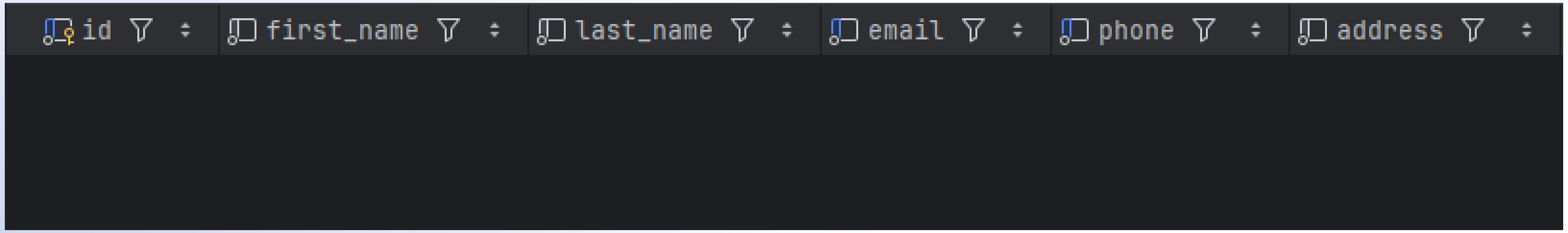


	id	first_name	last_name	email	phone	address
1	1	Yevhen	Yermolenko	email@e.com	911	Vinnytsia
2	2	Taras	Shevchenko	t@e.com	n/a	Moryntsi
3	4	Pes	Patron	p@e.com	white	Lviv

```
DELETE FROM customers;
```



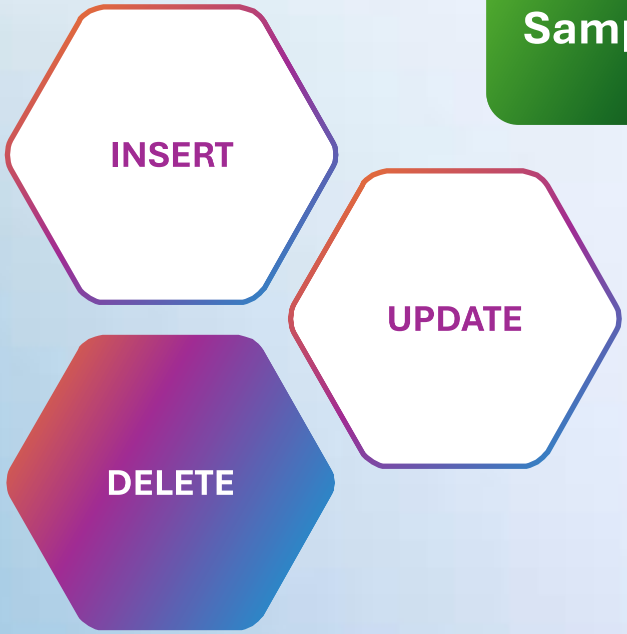
Unsafe query: 'Delete' statement without 'where' clears all data in the table



id	first_name	last_name	email	phone	address

- Use DELETE inside transaction;
- Run SELECT first to check the condition

Data modification



INSERT

UPDATE

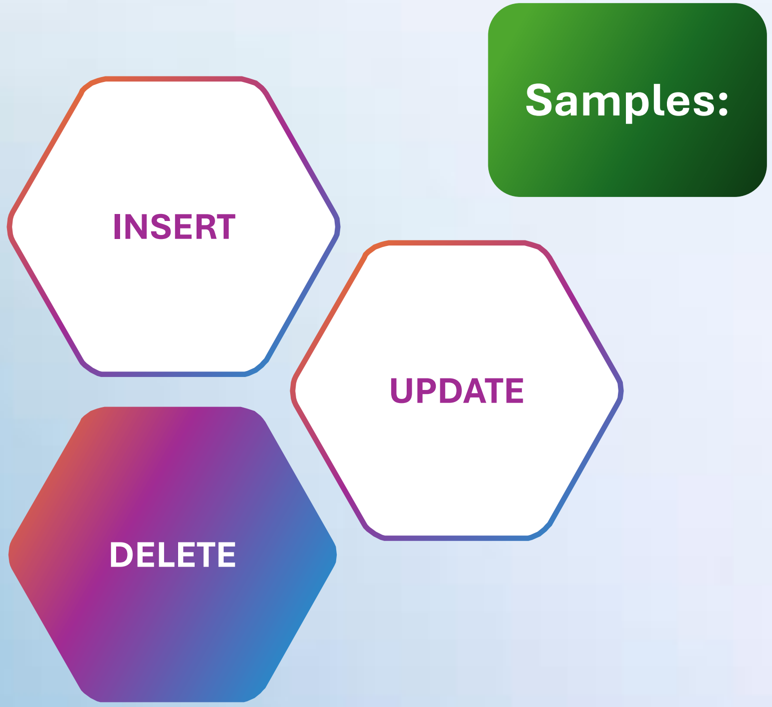
DELETE

Samples:

DELETE on large tables in batches:

DELETE FROM customers WHERE created_at < ‘2024-03-03’ LIMIT 1000;

Data modification



Samples:

INSERT

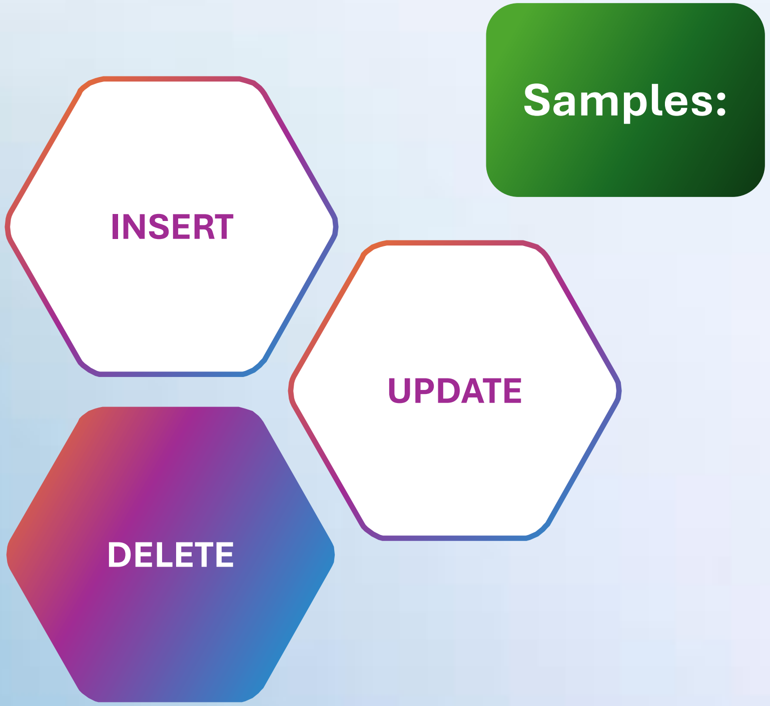
UPDATE

DELETE

DELETE on large tables in batches:
DELETE FROM customers WHERE created_at < ‘2024-03-03’ LIMIT 1000;

DELETE for foreign keys:
ALTER TABLE customers ADD CONSTRAINT fk_user
FOREIGN KEY (user_id) REFERENCES users(id)
ON DELETE SET NULL;

Data modification



INSERT

UPDATE

DELETE

Samples:

DELETE on large tables in batches:

```
DELETE FROM customers WHERE created_at < ‘2024-03-03’ LIMIT 1000;
```

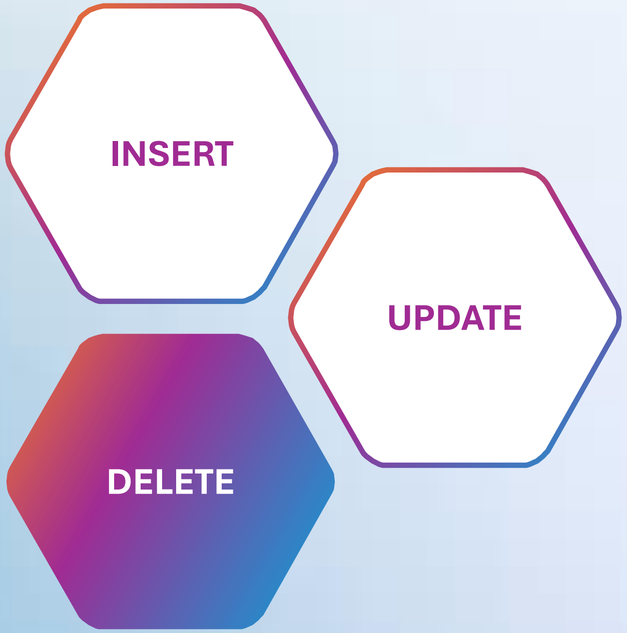
DELETE for foreign keys:

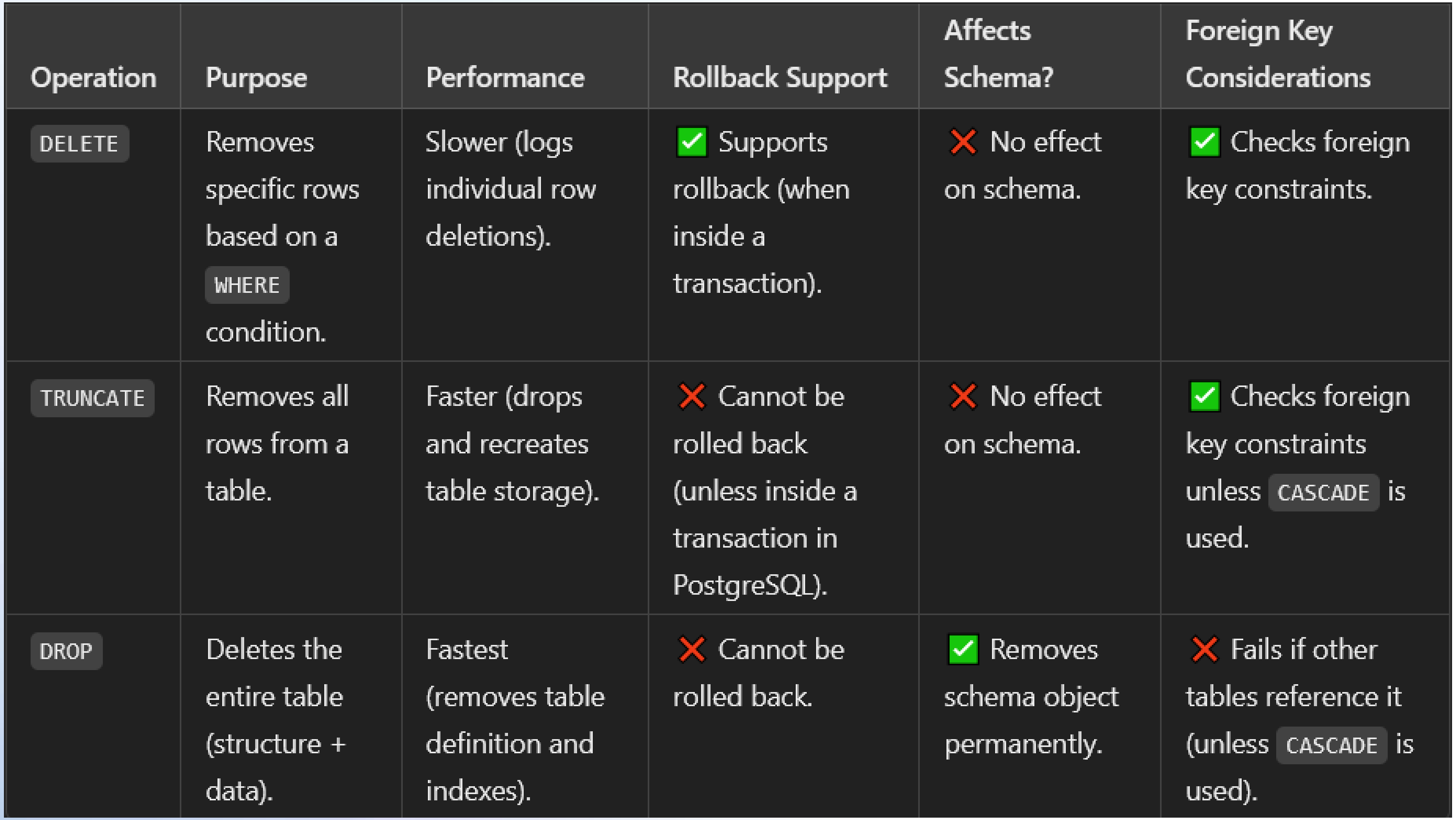
```
ALTER TABLE customers ADD CONSTRAINT fk_user
FOREIGN KEY (user_id) REFERENCES users(id)
ON DELETE SET NULL;
```

Lost Deletes prevention:

```
BEGIN TRANSACTION;
SELECT * FROM customers WHERE first_name = 'Yevhen' FOR UPDATE;
DELETE FROM customers WHERE first_name = 'Yevhen’;
COMMIT;
```

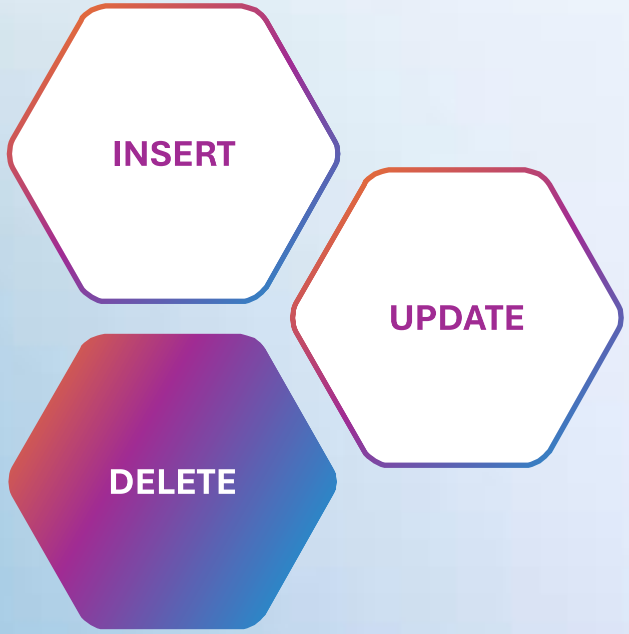
Data modification





Operation	Purpose	Performance	Rollback Support	Affects Schema?	Foreign Key Considerations
`DELETE`	Removes specific rows based on a `WHERE` condition.	Slower (logs individual row deletions).	✅ Supports rollback (when inside a transaction).	❌ No effect on schema.	✅ Checks foreign key constraints.
`TRUNCATE`	Removes all rows from a table.	Faster (drops and recreates table storage).	❌ Cannot be rolled back (unless inside a transaction in PostgreSQL).	❌ No effect on schema.	✅ Checks foreign key constraints unless `CASCADE` is used.
`DROP`	Deletes the entire table (structure + data).	Fastest (removes table definition and indexes).	❌ Cannot be rolled back.	✅ Removes schema object permanently.	❌ Fails if other tables reference it (unless `CASCADE` is used).

Data modification



SUMMARY

- Always use WHERE with DELETE
- Do DELETE inside transactions
- Do deletes in batches for large amount of data
- Use SELECT to check condition before executing DELETE
- Handle concurrency issues by locking rows before deletion
- Use SOFT DELETE instead of physical deletion

Security Fundamentals

CODEUS_

Security Fundamentals

Business requirements:

GDPR - General Data Protection Regulation

CODEUS_

Security Fundamentals

Business requirements:

GDPR - General Data Protection Regulation

HIPAA - Health Insurance Portability and Accountability Act

Security Fundamentals

Business requirements:

GDPR - General Data Protection Regulation

HIPAA - Health Insurance Portability and Accountability Act

PCI-DSS - Payment Card Industry Data Security Standard

Security Fundamentals

PostgreSQL:

Authentication – who can access (password-based, Kerberos, certificate-based)

Security Fundamentals

PostgreSQL:

Authentication – who can access (password-based, Kerberos, certificate-based)

Authorization – what authenticated users can do (use least privilege principle)

Security Fundamentals

PostgreSQL:

Authentication – who can access (password-based, Kerberos, certificate-based)

Authorization – what authenticated users can do (use least privilege principle)

RBAC – role-based access control (grand roles to users or groups)

Security Fundamentals

PostgreSQL logging capabilities:

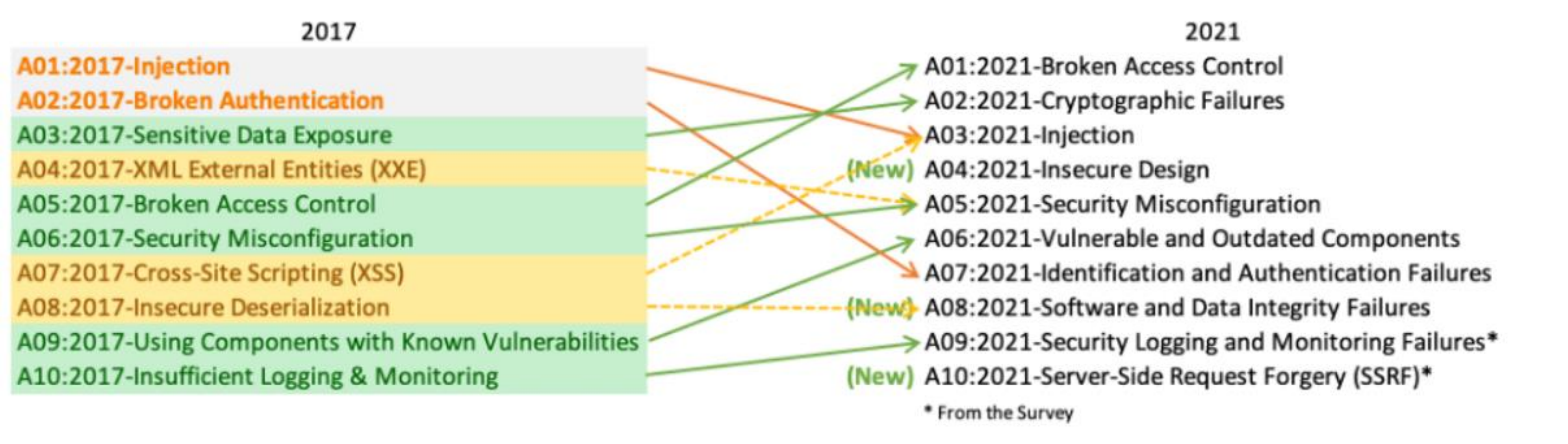
- Connection attempts
- SQL statements executed
- Db object modifications (table creation, data inserts)
- Administrative actions (role changes, config updates)
- Error messages and warnings

Security Fundamentals

HAVE A BACKUP AND DISASTER RECOVERY PLAN

Security Fundamentals

OWASP top 10



2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

https://owasp.org/www-project-top-ten/

CODEUS_

Security Fundamentals

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

Security Fundamentals

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

```
pass' OR 1 = 1
```

Security Fundamentals

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

```
pass' OR 1 = 1
```

```
sql = "SELECT id FROM users WHERE username = 'user' AND password = 'pass' OR 1= 1"
```

Security Fundamentals

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

```
pass’ OR 1 = 1
```

```
sql = "SELECT id FROM users WHERE username = 'user' AND password = 'pass’ OR 1= 1"
```

Avoid concatenation. Use prepared statements.

Thank you

- Author: Yevhen Yermolenko
- My LinkedIn: https://www.linkedin.com/in/yerm/
- March 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn