SQL II

Review SQL

- SQL
  - DDL
  - DML
  - DCL
  - TCL
  - DQL
  - …

DDL

- CREATE
  - Table
    - CREATE TABLE mytable (
    - id int unsigned NOT NULL auto_increment, username varchar(100) NOT NULL, email varchar(100) NOT NULL, PRIMARY KEY (id));
  - View
    - CREATE VIEW MYVIEW AS SELECT C1, C2 C3 FROM TABLE1
- DROP
  - Table
  - View
- ALTER
  - Table
  - View
    - ADD, DROP, MODIFY
- TRUNCATE

DML

- INSERT
  - INSERT INTO mytable (username, email) VALUES ("myuser", "myuser@example.com");
- UPDATE
  - UPDATE mytable SET username="myuser" WHERE id=8;
- DELETE
  - DELETE FROM mytable WHERE id=8;

DCL

- GRANT
  - GRANT SELECT, UPDATE, DELETE ON TABLEX TO USER1, USER2
- REVOKE
  - REVOKE SELECT, UPDATE, DELETE ON TABLEX FROM USER1

TCL

- Transactions
  - ACID
- COMMIT
  - COMMIT;
- ROLLBACK
  - ROLLBACK;
- SAVEPOINT
  - SAVEPOINT MYSAVEPOINT

DQL

- SELECT
  - SELECT * FROM mytable;
    - `SELECT column1, column2, … FROM table_name;`
  - SELECT * FROM mytable WHERE username = "myuser";
    - `SELECT column1, column2, … FROM table_name WHERE condition;`
  -

Relations

- Types of Relations
- 1:1
- 1:N
- N:M

- How to represent relations?

Transactions

```
app.post('/transfer', async (req, res) => {
  const { fromAccountId, toAccountId, amount } = req.body;

  if (!fromAccountId || !toAccountId || !amount || amount <= 0) {
    return res.status(400).json({ error: 'Invalid transfer details' });
  }
```

```
try {
  const connection = await pool.getConnection(); // Get a connection from the pool

  try {
    await connection.beginTransaction(); // Start the transaction

    // 1. Deduct amount from the 'from' account
    const [fromResult] = await connection.execute(
      'UPDATE accounts SET balance = balance - ? WHERE id = ?',
      [amount, fromAccountId]
    );
```

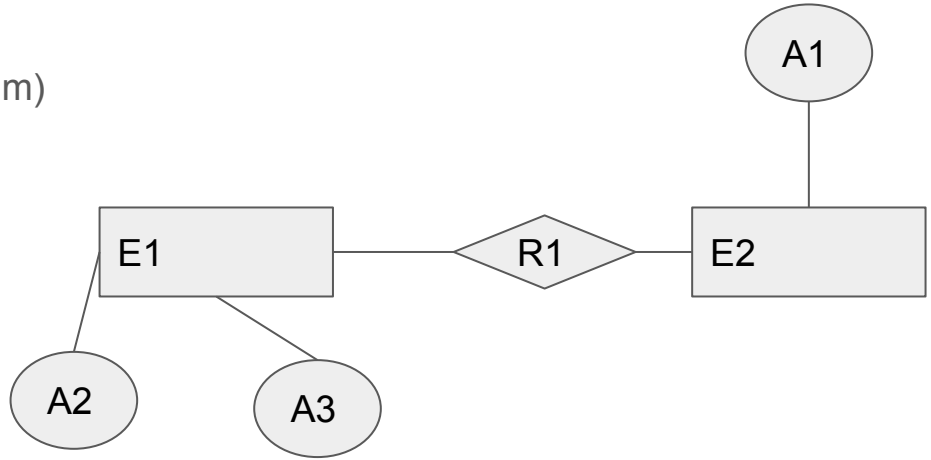
```
if (fromResult.affectedRows === 0) {
  throw new Error('Insufficient balance or invalid from account'); // Rollback if insufficient balance
}

// 2. Add amount to the 'to' account
const [toResult] = await connection.execute(
  'UPDATE accounts SET balance = balance + ? WHERE id = ?',
  [amount, toAccountId]
);

i
```

```
f (toResult.affectedRows === 0) {
      throw new Error('Invalid to account'); // Rollback if to account is invalid
    }

    await connection.commit(); // Commit the transaction if both operations are successful
    res.json({ message: 'Transfer successful' });

  } catch (error) {
    await connection.rollback(); // Rollback on any error
    console.error("Transaction Error:", error);
    res.status(500).json({ error: 'Transfer failed', details: error.message }); // Send error details
  } finally {
    connection.release(); // Release the connection back to the pool (important!)
  }
```

```
  } catch (err) {
    console.error("Database Connection Error:", err);
    res.status(500).json({ error: 'Database error' });
  }
});
```

How to develop tables?

- How to model?
- Many approaches
  - Experience
  - Entity Relationship Diagram (ER Diagram)



- Entity
- Attributes
- Relationship

Relational Model

- Strengths

- Challenges

Alternatives -> Non Standard Databases -> Not Only SQL -> NoSQL

NoSQL Types

- Key Value Store
  - Dynamo, https://hub.docker.com/search?q=dynamo&type=image
  - S3, ……
- Column-oriented Store
  - Flip -> focus on columns
  - Cassandra, https://hub.docker.com/_/cassandra
  - BigTable, …..
- Document-oriented Store
  - CouchDB, https://hub.docker.com/search?q=couchdb&type=image
  - MongoDB, https://hub.docker.com/search?q=mongodb&type=image
  - …..
- Graph-oriented Store
  - Neo4J, https://hub.docker.com/_/neo4j
  - ….
- …..