**NOTE: Oracle does not allow ON UPDATE and we understand foreign keys need to be updated**

1. This project represents the schema of a logistics and transportation company. The database models the real-world operation of managing trucks, drivers, dispatchers, customer orders, and geographic locations such as garages, company offices and client buildings. The whole database is built around managing orders and each entity serves a purpose to provide detailed information on each order. The schema incorporates ISA through drivers and dispatchers, aggregation via driver-truck-order relationships, and interlinked tables like connecting which truck and drivers are out on an order or where from and to its going. The system also simulates how trucks are dispatched and parked, and how employees are organized across various work locations. It ensures integrity across all entities and enforces realistic constraints to reflect actual business logic.

2. Our final schema is mostly the same but just changed around what details would be efficient to be stored within an order. Before we had whole entities like payments and maintenance which are important to a trucking company but are irrelevant for a system which is meant to simulate dispatching orders. Instead we changed those to invoices and customers and that is more important and relevant information to be connected with an order. To make our routes more efficient instead of having attributes which give start and end locations, we made our branches ISA hierarchy to a general locations ISA relationship so that it doesn't just contain company office and garage locations but also clients buildings. With this update the routes start and end attributes reference to coordinates from the locations table. And one final update was making the driver-trucks relationship to an aggregation which was then used in a relationship with orders to create a table which contains which truck and driver is carrying out the order. And there are also some minor changes in the attributes in all the entities, again just adding and removing some based on what is needed to best model “dispatching”.

3 / 4.

```
// QUERY 1: Insert  (appService.js LINE 100)
async function insertRouteTable(routeId, origin, destination, distance) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `INSERT INTO ROUTETABLE (routeId, origin, destination, distance) VALUES
(:routeId, :origin, :destination, :distance)`,
            [routeId, origin, destination, distance],
            { autoCommit: true }
        );

        return result.rowsAffected && result.rowsAffected > 0;
    }).catch(() => {
        return false;
    });
}

// QUERY 2: Update (appService.js LINE 206)
async function updateOrderTable(orderId, attribute, newValue) {
    return await withOracleDB(async (connection) => {
        let result;
        if (attribute === "orderDate") {
            result = await connection.execute(
                `UPDATE ORDERTABLE SET ${attribute}=TO_DATE(:newValue,
'YYYY-MM-DD') WHERE orderId=:orderId`,
                [newValue, orderId],
                { autoCommit: true }
            );
        } else {
            result = await connection.execute(
                `UPDATE ORDERTABLE SET ${attribute}=:newValue WHERE
orderId=:orderId`,
                [newValue, orderId],
                { autoCommit: true }
            );
        }
        return result.rowsAffected && result.rowsAffected > 0;
    }).catch((err) => {
        console.log(err);
        return false;
    });
}
// QUERY 3: Delete (appService.js LINE 115)
async function deleteRouteTable(routeId) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `DELETE FROM ROUTETABLE
            WHERE routeId=:routeId`,
            [routeId],
            { autoCommit: true }
        );

        return result.rowsAffected && result.rowsAffected > 0;
    }).catch(() => {
        return false;
    });
}
```

```
// QUERY 4: Select (appService.js LINE 230)
async function selectOrderTable(selectQuery) {
    return await withOracleDB(async (connection) => {
        const result = await connection.execute(
            `SELECT orderId, customerId, weight, routeId, TO_CHAR(orderDate,
'YYYY-MM-DD'), departureTime, arrivalTime FROM ORDERTABLE WHERE
${selectQuery}`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}


// QUERY 5: Projection (appService.js LINE 242)
async function projectOrderTable(projectQuery) {
    return await withOracleDB(async (connection) => {
        if (projectQuery.includes("orderDate")) {
            projectQuery = projectQuery.replace(/\borderDate\b/g, "TO_CHAR(orderDate,
'YYYY-MM-DD')");
        }

        console.log(projectQuery);
        const result = await connection.execute(
            `SELECT DISTINCT ${projectQuery} FROM ORDERTABLE`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}

// QUERY 6: Join (appService.js LINE 131)
async function joinRouteTable(selectQuery) {
    return await withOracleDB(async (connection) => {
        // console.log(selectQuery);
        const result = await connection.execute(
            `SELECT o.orderId, o.customerId, o.weight, o.routeId, TO_CHAR(o.orderDate,
'YYYY-MM-DD'), l1.address, l2.address, r.distance
```

```
            FROM ORDERTABLE o, ROUTETABLE r, LOCATIONTABLE l1,
LOCATIONTABLE l2
            WHERE o.routeId = r.routeId AND r.origin = l1.coordinate AND r.destination =
l2.coordinate AND r.distance >= ${selectQuery}`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}
```

**// QUERY 7: Group By (appService.js LINE 259)**

```
async function countCustomerOrderTable() {
    return await withOracleDB(async (connection) => {
        // console.log(selectQuery);
        const result = await connection.execute(
            `SELECT customerId, COUNT(*) AS orderCount
            FROM ORDERTABLE
            GROUP BY customerId
            ORDER BY orderCount DESC`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}
```

This query counts how many orders each customer has placed. It helps identify the most active customers by number of orders by ordering the count from highest to lowest.

**// QUERY 8: Having (appService.js LINE 275)**

```
async function findDateOrderTable() {
    return await withOracleDB(async (connection) => {
        // console.log(selectQuery);
        const result = await connection.execute(
            `SELECT TO_CHAR(orderDate, 'YYYY-MM-DD'), MIN(departureTime)
            FROM ORDERTABLE
            GROUP BY orderDate
            HAVING COUNT(*) > 1`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}
```

This query finds order dates where at least 2 orders were placed and shows the earliest departure time for those dates.

```
// QUERY 9: Nested Group By (appService.js LINE 291)
async function findWeightOrderTable() {
    return await withOracleDB(async (connection) => {
        // console.log(selectQuery);
        const result = await connection.execute(
            `SELECT o.orderId, o.weight
            FROM ORDERTABLE o
            WHERE o.weight >= ALL (SELECT AVG(o2.weight) FROM ORDERTABLE o2 GROUP BY o2.orderDate)`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}
```

This query returns orders that are heavier than or equal to the average order weight for each day. It helps highlight the heaviest orders across the dataset.

```
// QUERY 10: Division (appService.js LINE 146)
async function findRouteDateRouteTable() {
    return await withOracleDB(async (connection) => {
        // console.log(selectQuery);
        const result = await connection.execute(
            `SELECT r.routeId
            FROM ROUTETABLE r
            WHERE NOT EXISTS (
                SELECT o1.orderDate FROM ORDERTABLE o1
                MINUS (
                    SELECT o2.orderDate FROM ORDERTABLE o2
                    WHERE r.routeId=o2.routeId
                )
            )`
        );
        return result.rows;
    }).catch(() => {
        return [];
    });
}
```

This query finds routes that were used on every order date