**Chapter 6 – Team Project: Normalizing the Relational Model for the Team Project and Creating a Normalized Oracle Database**

**Step 6.1 - Begin with the list of the tables that the entities and relationships from the E-R diagram mapped to naturally, from the sample project section at the end of chapter 4.**

**Tables Created at the end of the Chapter 4:**

1. Client (clientId, firstName, lastName, street, city, state, zip, areaCode, phoneNumber)
2. Meeting (clientId, meetDate, meetTime, repName)
3. Photographer (empId, firstName, lastName, street, city, state, zip, areaCode, number)
4. Booking (clientId, bookingDate, bookingTime, duration, type, empId1, empId2)
5. PackageMenu (packageNo, numWallet, albumType, albumPages, albumCover, price)
6. Job (contractNo, type, eventName, location, clientId , date, time, duration, cost, empId1, empId2, packageNoChosen, totalCost, amtPaid, amtDue)
7. Proof (contractNo, proofNo, quality)
8. Order (orderNo, dateOrdered, totalAmount, packageNoOrdered, contractNo)
9. OrderItem (orderNo, proofNo, size, quantity, dateDelivered)
10. Payment (contractNo, datePaid, payType, amount)
11. EmergencyContact(Clientid,eventid,Emergencyfirstname,Emergencylastname,Emergencyphonenumber)

To determine the functional dependencies and the normalization process for the above tables, let's go through each table and identify its functional dependencies. Functional dependencies (FDs) describe the relationships between attributes in a table. The normalization process aims to eliminate redundancy and improve data integrity by breaking down the tables into smaller, more manageable forms.

1. Client Table:

* clientId uniquely determines firstName, lastName, street, city, state, zip, areaCode, phoneNumber.
* This table does not contain any partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

clientId → firstName, lastName, street, city, state, zip, areaCode, phoneNumber

zip → city, state

areaCode + phoneNumber → street, city, state

* Normalization to BCNF:

The initial table is not in BCNF because areaCode + phoneNumber determines non-prime attributes (street, city, state).

To achieve BCNF, we need to decompose the table.

1. firstName + lastName → all attributes
2. zip → city, state
3. areaCode + phoneNumber → street, city, state, zip
4. areaCode + phoneNumber → all attributes (but not considered relevant)

The correct recognition, that the areaCode + phoneNumber functional dependency is not practical for determining all attributes due to the possibility of multiple individuals sharing the same phone number at a single location.

To address this, you initially decomposed the Client table into three separate tables:

The table will be:

1. Client1 (clientId, firstName, lastName, zip, areaCode, phoneNumber)
2. Address (zip, city, state)
3. PhoneNumber (areaCode, phoneNumber, street)

While the initial decomposition was sound from a normalization perspective, it would have required multiple joins to retrieve the complete address for a client, which could impact query performance.

To address this issue, you opted for a compromise by keeping the street and zip in the Client1 table, while leaving the Zips table as it is. This compromise allows for more efficient querying while still maintaining normalized data.

Resulting Table will be:

1. Client2 (clientId, firstName, lastName, zip, areaCode, phoneNumber)
2. Address (zip, city, state)
3. PhoneNumber (areaCode, phoneNumber, street)

It's a good practice to use numeric surrogate keys, like the `clientId` mentioned, as primary keys in database tables. Numeric keys are more efficient for indexing, take up less storage space, and reduce the chances of data entry errors due to variations in character data. They also improve query performance and maintain data integrity.

It's common to use auto-incrementing or identity columns in database management systems (DBMS) to automatically generate unique numeric values for these surrogate keys. Oracle uses identity columns or sequences, and Microsoft Access uses the autonumber data type to achieve the same goal.

In this case, we've mentioned that the final tables have been designed with ‘clientId’ as the primary key. This approach ensures that all attributes in the ‘Client’ table are functionally dependent on the ‘clientId’ key, meeting the requirements for a surrogate key. This design is aligned with best practices for database design and will help maintain data consistency and accuracy.

Using numeric surrogate keys is a standard practice in relational database design and is recommended for most scenarios where unique identification of records is required.

2. Meeting Table:

* clientId determines meetDate, meetTime, and repName.
* This table does not contain any partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

clientId → meetDate, meetTime, repName

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

3. Photographer Table:

* empId uniquely determines firstName, lastName, street, city, state, zip, areaCode, and number.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

1. empId → firstName, lastName, street, city, state, zip, areaCode, number
2. zip → city, state
3. areaCode + number → street, city, state

* Normalization to BCNF:
* The initial table is not in BCNF because areaCode + number determines non-prime attributes (street, city, state).
* To achieve BCNF, we need to decompose the table.
* The new tables will be:

1. Photographer1 (empId, firstName, lastName, zip, areaCode, number)
2. Address (zip, city, state)
3. PhoneNumber (areaCode, number, street)

4. Booking Table:

* clientId and bookingDate determine bookingTime, duration, type, empId1, and empId2.
* This table doesn't contain partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

clientId → bookingDate, bookingTime, duration, type, empId1, empId2

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

5. PackageMenu Table:

* packageNo determines numWallet, albumType, albumPages, albumCover, and price.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

packageNo → numWallet, albumType, albumPages, albumCover, price

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

6. Job Table:

* contractNo determines type, eventName, location, clientId, date, time, duration, cost, empId1, empId2, packageNoChosen, totalCost, amtPaid, amtDu`.
* This table seems to contain some redundancy. It might benefit from further normalization to remove partial dependencies and ensure data integrity.
* Functional Dependencies:

contractNo → type, eventName, location, clientId, date, time, duration, cost, empId1, empId2, packageNoChosen, totalCost, amtPaid, amtDue.

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

7. Proof Table:

* contractNo and proofNo determine quality.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

contractNo → proofNo, quality

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

8. Order Table:

* orderNo determines dateOrdered, totalAmount, packageNoOrdered, contractNo.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

orderNo → dateOrdered, totalAmount, packageNoOrdered, contractNo

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

9. OrderItem Table:

* orderNo and proofNo determine size, quantity, and dateDelivered.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

orderNo, proofNo → size, quantity, dateDelivered

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

10. Payment Table:

* contractNo determines datePaid, payType, and amount.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

contractNo → datePaid, payType, amount

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

11. EmergencyContact Table:

* clientId and eventid determine Emergencyfirstname, Emergencylastname, and Emergencyphonenumber.
* This table doesn't have partial or transitive dependencies. It's in 1st Normal Form (1NF) and doesn't need further normalization.
* Functional Dependencies:

1. Clientid,eventid→Emergencyfirstname,Emergencylastname, Emergencyphonenumber

* Normalization to BCNF:

The initial table is already in BCNF. No further decomposition is needed.

To summarize the normalization process:

- Tables 1 to 10 are in 1st Normal Form (1NF) and don't require further normalization, as they do not contain partial or transitive dependencies.

- Table 6 (`Job`) appears to have potential redundancy and may benefit from further normalization to ensure data integrity, possibly leading to 2nd or 3rd Normal Form (2NF or 3NF) forms.

**Step 6.2 – Review and update the data dictionary and list of assumptions (as needed).**

1. Client (clientId, firstName, lastName),
2. Address(zip,street,city,state)
3. Phonenumber(areacode, zip,street,city,state),
4. Meeting (clientId, meetDate, meetTime, repName)
5. Photographer (empId, firstName, lastName, street,zip,)
6. Booking (clientId, bookingDate, bookingTime, duration, type, empId1, empId2)
7. PackageMenu (packageNo, numWallet, albumType, albumPages, albumCover, price)
8. Job (contractNo, type, eventName, location, clientId , date, time, duration, cost, empId1, empId2, packageNoChosen, totalCost, amtPaid, amtDue)
9. Proof (contractNo, proofNo, quality)
10. Order (orderNo, dateOrdered, totalAmount, packageNoOrdered, contractNo)
11. OrderItem (orderNo, proofNo, size, quantity, dateDelivered)
12. Payment (contractNo, datePaid, payType, amount)
13. EmergencyContact(Clientid,eventid,Emergencyfirstname,Emergencylastname,Emergencyphonenumber)

**Step 6.3 - For each table, write the table name and write out the names, data types, and sizes of all the data items.**

**Client Table:**

* clientId (NUMBER, PRIMARY KEY): Stores unique client ID.
* firstName (VARCHAR2(50)): First name of the client.
* lastName (VARCHAR2(50)): Last name of the client.

**Address Table:**

* zip (VARCHAR2(10), PRIMARY KEY): ZIP code of the address.
* city (VARCHAR2(50)): City of the address.
* state (VARCHAR2(2)): State of the address.

**PhoneNumber Table:**

* areaCode (VARCHAR2(3)): Area code of the phone number.
* phoneNumber (VARCHAR2(20)): Phone number.
* street (VARCHAR2(100)): Street address.
* zip (VARCHAR2(10)): ZIP code associated with the address.
* PRIMARY KEY (areaCode, phoneNumber).
* FOREIGN KEY (zip) REFERENCES Address(zip): Reference to the Address table.

**Meeting Table:**

* clientId (NUMBER): Client ID associated with the meeting.
* meetDate (DATE): Date of the meeting.
* meetTime (TIMESTAMP): Time of the meeting.
* repName (VARCHAR2(50)): Representative name.
* FOREIGN KEY (clientId) REFERENCES Client(clientId): Reference to the Client table.

**Photographer Table:**

* empId (NUMBER, PRIMARY KEY): Unique photographer ID.
* firstName (VARCHAR2(50)): First name of the photographer.
* lastName (VARCHAR2(50)): Last name of the photographer.
* zip (VARCHAR2(10)): ZIP code associated with the photographer.
* street (VARCHAR2(100)): Street address of the photographer.
* FOREIGN KEY (zip) REFERENCES Address(zip): Reference to the Address table.

**Booking Table:**

* clientId (NUMBER): Client ID associated with the booking.
* bookingDate (DATE): Date of the booking.
* bookingTime (TIMESTAMP): Time of the booking.
* duration (NUMBER): Duration of the booking.
* type (VARCHAR2(50)): Type of booking.
* empId1 (NUMBER): First photographer's ID.
* empId2 (NUMBER): Second photographer's ID.
* FOREIGN KEY (clientId) REFERENCES Client(clientId): Reference to the Client table.
* FOREIGN KEY (empId1) REFERENCES Photographer(empId): Reference to the Photographer table.
* FOREIGN KEY (empId2) REFERENCES Photographer(empId): Reference to the Photographer table.

**PackageMenu Table:**

* packageNo (NUMBER, PRIMARY KEY): Unique package number.
* numWallet (NUMBER): Number of wallets in the package.
* albumType (VARCHAR2(50)): Type of album in the package.
* albumPages (NUMBER): Number of pages in the album.
* albumCover (VARCHAR2(50)): Type of cover for the album.
* price (NUMBER): Price of the package.

**Job Table:**

* contractNo (NUMBER, PRIMARY KEY): Unique job contract number.
* type (VARCHAR2(50)): Type of job (e.g., wedding, event).
* eventName (VARCHAR2(100)): Name of the event.
* location (VARCHAR2(100)): Location of the event.
* clientId (NUMBER): Client ID associated with the job.
* jobdate (DATE): Date of the job.
* time (TIMESTAMP): Time of the job.
* duration (NUMBER): Duration of the job.
* cost (NUMBER): Cost of the job.
* empId1 (NUMBER): First photographer's ID.
* empId2 (NUMBER): Second photographer's ID.
* packageNoChosen (NUMBER): Chosen package number.
* totalCost (NUMBER): Total cost of the job.
* amtPaid (NUMBER): Amount paid by the client.
* amtDue (NUMBER): Amount due from the client.

**Proof Table:**

* contractNo (NUMBER): Job contract number associated with proof.
* proofNo (NUMBER): Proof number.
* quality (VARCHAR2(50)): Quality of the proof.
* PRIMARY KEY (contractNo, proofNo).
* FOREIGN KEY (contractNo) REFERENCES Job(contractNo): Reference to the Job table.

**Orders Table:**

* orderNo (NUMBER, PRIMARY KEY): Unique order number.
* dateOrdered (DATE): Date when the order was placed.
* totalAmount (NUMBER): Total amount of the order.
* packageNoOrdered (NUMBER): Chosen package number for the order.
* contractNo (NUMBER): Job contract number associated with the order.
* FOREIGN KEY (packageNoOrdered) REFERENCES PackageMenu(packageNo): Reference to the PackageMenu table.
* FOREIGN KEY (contractNo) REFERENCES Job(contractNo): Reference to the Job table.

**Payment Table:**

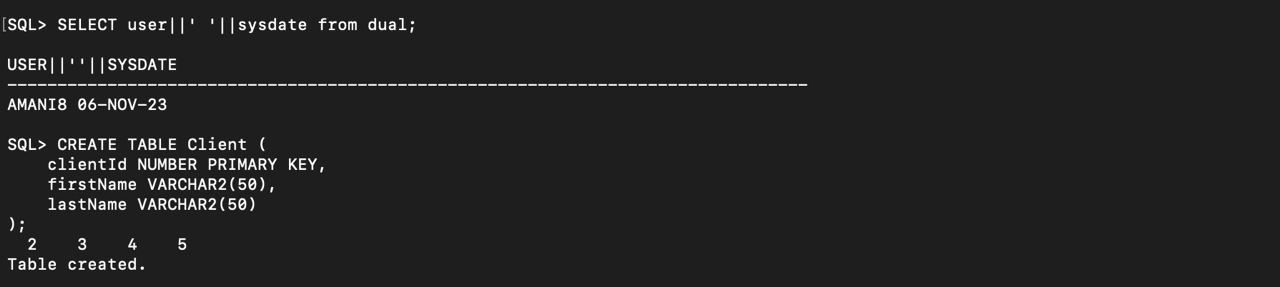
* contractNo (NUMBER): Job contract number associated with payment.
* datePaid (DATE): Date when the payment was made.
* payType (VARCHAR2(50)): Type of payment (e.g., cash, credit).
* amount (NUMBER): Payment amount.
* PRIMARY KEY (contractNo, datePaid).
* FOREIGN KEY (contractNo) REFERENCES Job(contractNo): Reference to the Job table.

**EmergencyContact Table:**

* clientId (NUMBER): Client ID associated with the emergency contact.
* eventid (NUMBER): Event ID associated with the emergency contact.
* Emergencyfirstname (VARCHAR2(50)): First name of the emergency contact.
* Emergencylastname (VARCHAR2(50)): Last name of the emergency contact.
* Emergencyphonenumber (VARCHAR2(20)): Phone number of the emergency contact.
* PRIMARY KEY (clientId, eventid).
* FOREIGN KEY (clientId) REFERENCES Client(clientId): Reference to the Client table.

**Step 6.4 - Design SQL statements to create all tables needed to implement the design. Then create the tables in the database.**

1. **Client**

****

1. **Address**

**A black background with white lines

Description automatically generated**

1. **Phonenumber**

**A black screen with white lines

Description automatically generated**

1. **Meeting**

**A black screen with white lines

Description automatically generated**

1. **Photographer**

**A black screen with a white line

Description automatically generated**

1. **Booking**

**A black screen with a white line

Description automatically generated**

1. **PackageMenu**

**A black screen with white lines

Description automatically generated**

1. **Job**

**A black screen with white text

Description automatically generated**

1. **Proof**

**A computer screen with white text

Description automatically generated**

1. **orders**

**A computer screen with white text

Description automatically generated**

1. **Orderitem**

**A black screen with white lines

Description automatically generated**

1. **Payment**

**A black screen with white text

Description automatically generated**

1. **Emergency Contact**

**A black screen with white text

Description automatically generated**

**Step 6.5 - Design SQL statements to create indexes for foreign keys and for any other columns that will be used most often for queries. Then execute the SQL statements in the database.**

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Step 6.6 - Design SQL statements to insert at least five records in each table, preserving all constraints.** **Then insert the records into the tables.**

**Inserting Records into table:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

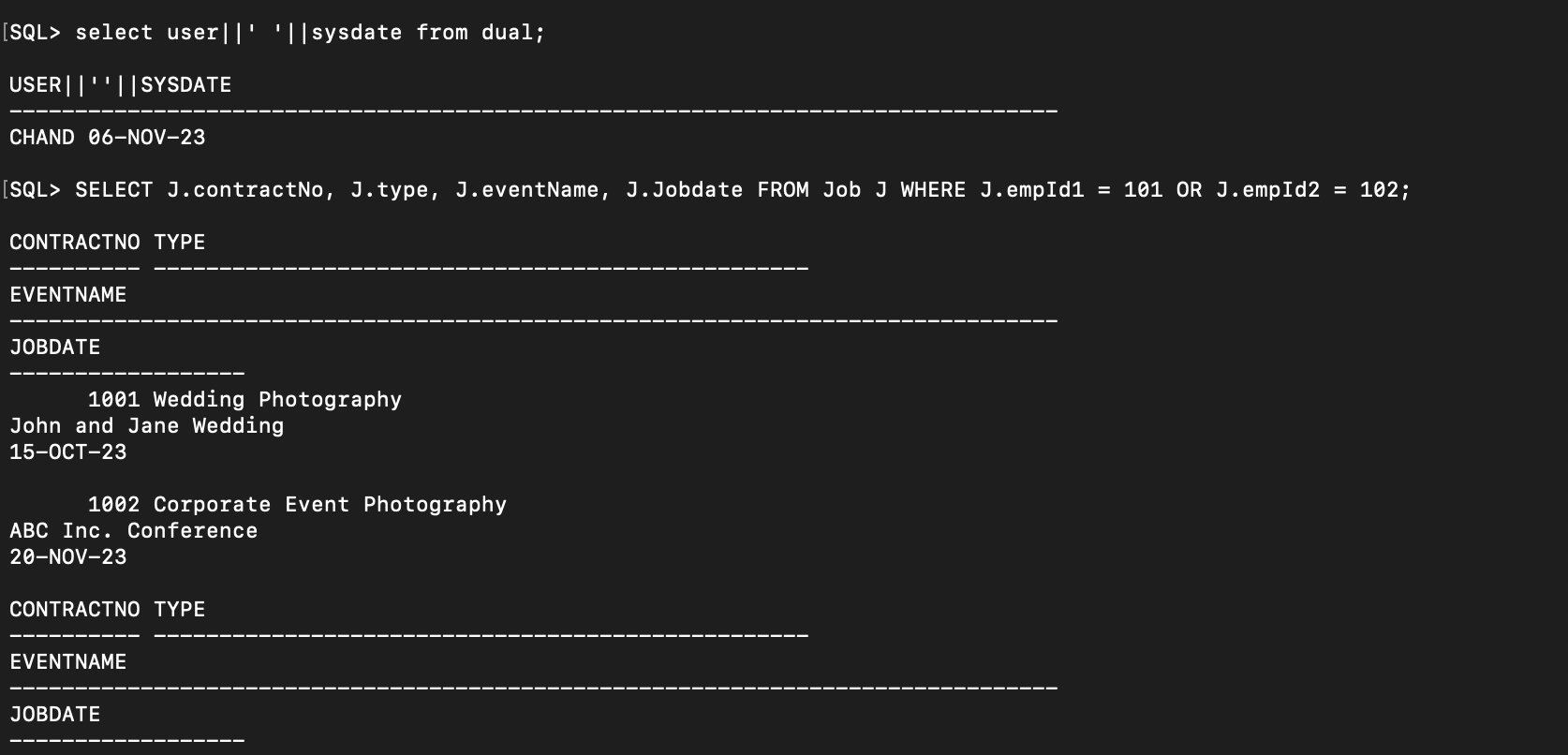
**Step 6.7 - Design SQL statements that will process five non-routine requests for information from the database. Then execute the SQL statements in the database.**

1. List Clients Who Have Booked a Job: To list clients who have booked at least one job.

A screenshot of a computer

Description automatically generated

1. Retrieve Jobs for a Specific Photographer: To retrieve jobs assigned to a specific photographer.



1. Find the Highest-Paying Job Type: To find the job type that generates the highest total revenue.

A black screen with white text

Description automatically generated

1. List Clients Who Have Outstanding Payments and the Corresponding Jobs:

To list clients with outstanding payments and the details of the jobs they've booked.

A screenshot of a computer

Description automatically generated

1. Retrieve Total Payments by Payment Type: To retrieve the total payments made for each payment type.

A screen shot of a computer

Description automatically generated

**Step 6.8 - Design one trigger for your project. Then create the trigger in the database.**

1. **Create the orders2 table:**

A black screen with white text

Description automatically generated

1. **Inserting data into the table:**

**A screenshot of a computer

Description automatically generated**

1. **Creating a trigger on Job:**

**A screenshot of a computer

Description automatically generated**

**Step 6.9 - Design and execute SQL statements to demonstrate that the trigger is working as expected.**

1. **Inserting new ‘3’ records into table Job:**



A black screen with white text

Description automatically generated

1. **Verifying that the trigger updated the "Order2" table:**

**The Order2 table to verify the update:**

**The query executed down in screenshot will show you the records in the "Order2" table, including the one that was automatically inserted or updated by the trigger.**

**A screen shot of a computer

Description automatically generated**

1. **Check the "Job" table to ensure that it contains the newly inserted job record:** A screenshot of a computer

   Description automatically generated

A screenshot of a computer

Description automatically generated

**This query will confirm the existence of the new job record in the "Job" table.**

**The trigger will automatically be updated the "Order2" table whenever a new record was inserted into the "Job" table.**

1. **Updating existing records in the "Job" table to observe the trigger's behavior when updating records.**

**A. -- Update an existing record in the Job table**

**-- Replace the values with your specific data**

**A screen shot of a computer

Description automatically generated**

**B. -- Query the Job table to verify the updated record**

**A screenshot of a computer

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**C. -- Query the Order2 table to verify the corresponding update**

**A screen shot of a computer code

Description automatically generated**

**These queries will allow you to see the updated data in both tables, confirming that the trigger worked as expected when updating existing records in the "Job" table.**