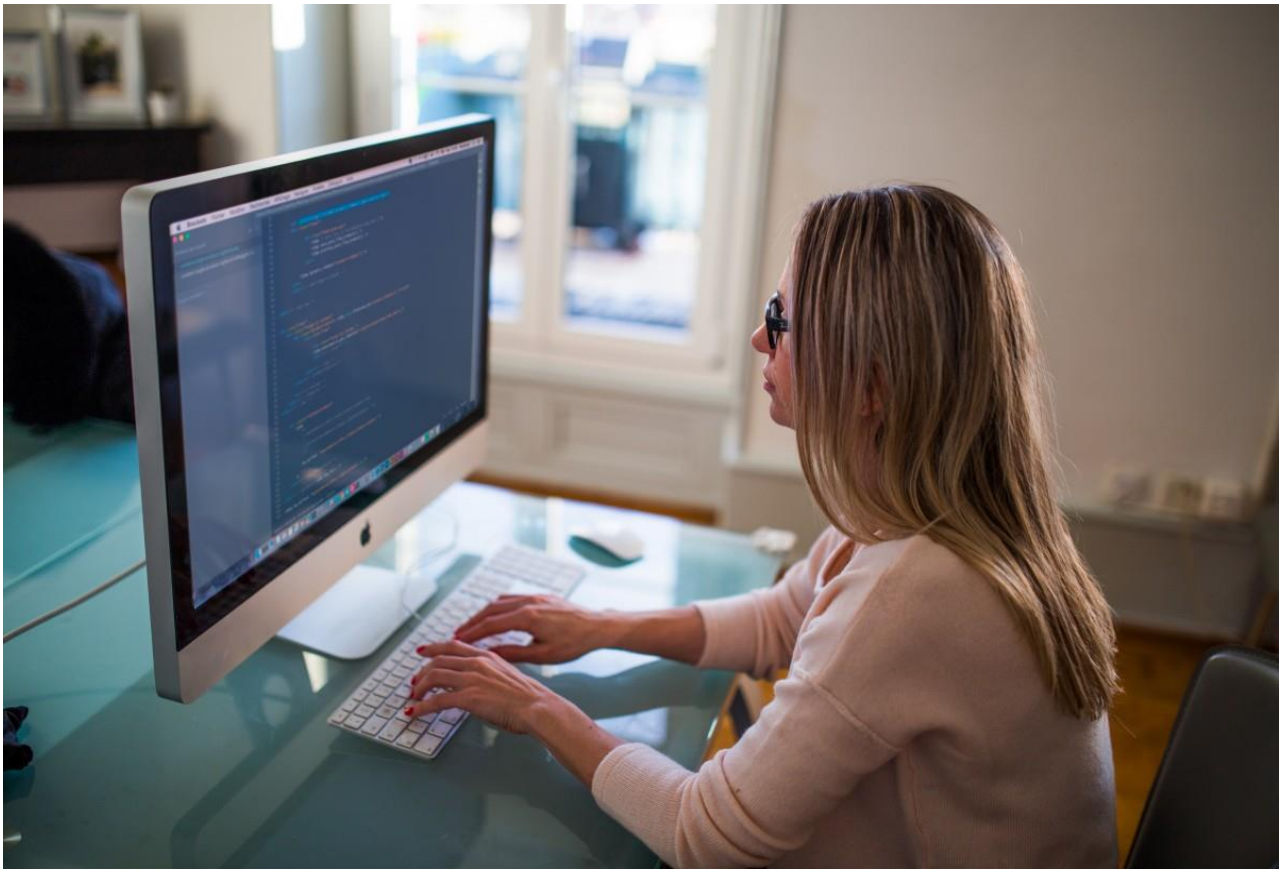
DATABASE PROJECT

Project Title: Freelancer Hustle DB



Introduction:

Freelancer Hustle DB is a relational database system designed to help freelancers **organize, track, and analyze** their entire freelance workflow — from managing clients and project details to logging payments, collecting feedback, and generating invoices.

Whether you're a solo freelancer or part of a growing freelance team, this system provides a structured way to keep your hustle efficient, transparent, and data-driven.

It’s especially useful for:

- Tracking income and deliverables
- Managing multiple clients and deadlines
- Monitoring project status and milestones
- Recording client ratings and feedback

The project is implemented using **SQL** with a fully normalized schema and supports **advanced query operations**, enabling powerful insights like:

"Who are my highest-paying clients?",
"Which projects are unpaid?",
"What’s my average rating this month?"

Core Functionalities:

1. Freelancer Profile Management

- Store freelancer details (name, expertise, email)
- Extendable to support multi-user environments

2. Client Tracking

- Store contact and company info for each client
- Useful for organizing repeat work or outreach

3. Project Management

- Link projects with freelancers and clients

- Track start/end dates, project descriptions, and status (e.g., "Ongoing", "Completed")

4. Milestones

- Divide projects into smaller goals with due dates
- Track which milestones are complete/incomplete

5. Proposals

- Keep a record of submitted project proposals
- Track proposal status (Pending, Accepted, Rejected)
- Helps monitor pre-contract communication

6. Payments

- Record payment transactions per project
- Store amount, date paid, and payment method
- Identify which projects are still unpaid

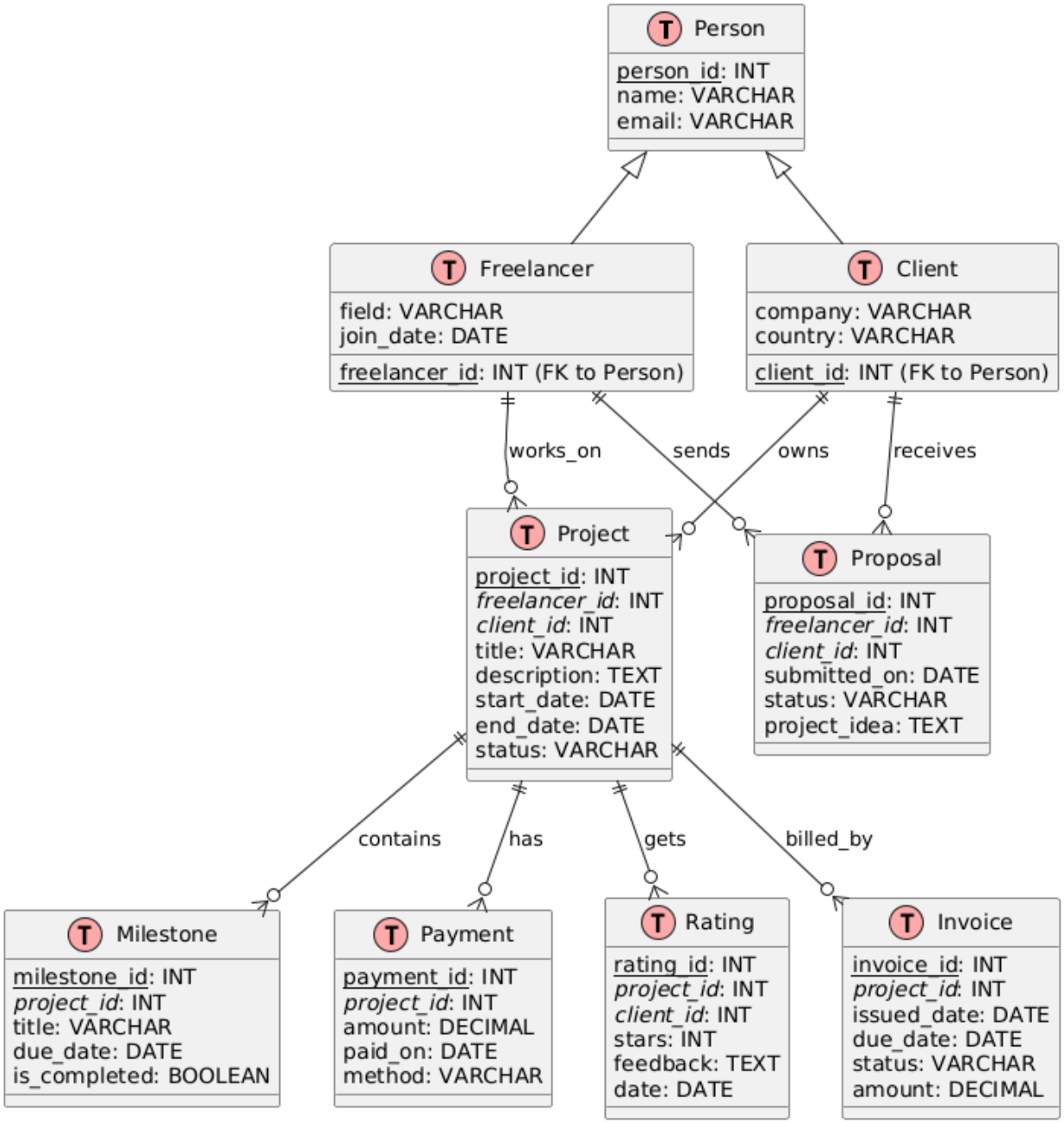
7. Ratings and Feedback

- Collect client reviews (1–5 stars) and optional feedback per project
- Calculate average ratings per freelancer

8. Invoices

- Create invoice records for delivered work
- Store issue date, due date, status (Paid/Unpaid), and total amount

Entity-Relationship Diagram (ERD):



Relationships Between Entities:

Relationship	Type	Description
Freelancer — Project	One-to-Many	A freelancer can work on many projects
Client — Project	One-to-Many	A client can give many projects
Project — Payment	One-to-Many	Each project can have one or more payments
Project — Rating	One-to-One	One rating per project by the client
Project — Milestone	One-to-Many	Each project can have multiple milestones
Project — Invoice	One-to-One	One invoice per project
Freelancer — Proposal	One-to-Many	A freelancer can send multiple proposals
Client — Proposal	One-to-Many	A client can receive multiple proposals

Construction of Relational Schema:

Top-Down Approach (Entity to Table):

PERSON	PERSON_ID (PK)	NAME	EMAIL

FREELANCER	FREELANCER_ID (PK)	FK → PERSON_ID)	FIELD	JOIN_DATE

CLIENT	CLIENT_ID (PK	FK → PERSON_ID)	COMPANY	COUNTRY

PROJECT	PROJECT _ID (PK)	FREELANCER _ID (FK)	CLIENT _ID (FK)	TITLE	DESCRIPTION	START _DATE	END_ DATE	STATUS

MILESTONE	MILESTONE_ID (PK)	PROJECT_ID (FK)	TITLE	DUE_DATE	IS_COMPLETED

PROPOSAL	PROPOSAL _ ID (PK)	FREELANCER_ ID (FK)	CLIENT_ ID (FK)	SUBMITTED_ ON	STATUS	PROJECT_IDEA

PAYMENT Table	PAYMENT_ID (PK)	PROJECT_ID (FK)	AMOUNT	PAID_ON	METHOD

RATING	RATING_ID (PK)	PROJECT_ID (FK)	CLIENT_ID (FK)	STARS	FEEDBACK	DATE

INVOICE	INVOICE_ID (PK)	PROJECT_ID (FK)	ISSUED_DATE	DUE_DATE	STATUS	AMOUNT

BOTTOM-UP APPROACH:

1NF (Unnormalized to First Normal Form)

Remove repeating groups & ensure atomic values.

1. FREELANCER (Freelancer_ID, Name, Email, Field, Join_Date)
2. CLIENT (Client_ID, Name, Email, Company, Country)
3. PROJECT (Project_ID, Title, Description, Start_Date, End_Date, Status, Freelancer_ID, Client_ID)
4. PAYMENT (Payment_ID, Project_ID, Amount, Paid_On, Method)
5. RATING (Rating_ID, Project_ID, Client_ID, Stars, Feedback, Date)
6. MILESTONE (Milestone_ID, Project_ID, Title, Due_Date, Is_Completed)
7. INVOICE (Invoice_ID, Project_ID, Issued_Date, Due_Date, Amount, Status)
8. PROPOSAL (Proposal_ID, Freelancer_ID, Client_ID, Submitted_On, Status, Project_Idea)

2NF (Partial Dependency Removed)

All non-key attributes depend on the full primary key i.e no partial dependancy.

So, the relation is in 2NF.

3NF (Transitive Dependencies Removed)

Non-key attributes should not depend on other non-key attributes

No transitive dependencies found

So, the relation is already in 3NF.

DATATYPES AND DESCRIPTION:

PERSON (Superclass):

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Person_id	VARCHAR2	10	PK
Name	VARCHAR2	30	NOT NULL
Email	VARCHAR2	40	UNIQUE, NOT NULL

FREELANCER:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Freelancer_id	VARCHAR2	10	PK, FK (References Person)
Field	VARCHAR2	30	
Join_date	DATE	–	DEFAULT SYSDATE

CLIENT:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Client_id	VARCHAR2	10	PK, FK (References Person)
Company	VARCHAR2	30	
Country	VARCHAR2	20	

PROJECT:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Project_id	VARCHAR2	10	PK
Freelancer_id	VARCHAR2	10	FK (References Freelancer)
Client_id	VARCHAR2	10	FK (References Client)
Title	VARCHAR2	50	NOT NULL
Description	VARCHAR2	200	
Start_date	DATE	–	
End_date	DATE	–	
Status	VARCHAR2	20	

MILESTONE:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Milestone_id	VARCHAR2	10	PK
Project_id	VARCHAR2	10	FK (References Project)
Title	VARCHAR2	50	NOT NULL
Due_date	DATE	–	
Is_completed	NUMBER	1	CHECK ('1', '0')

PROPOSAL:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Proposal_id	VARCHAR2	10	PK
Freelancer_id	VARCHAR2	10	FK (References Freelancer)
Client_id	VARCHAR2	10	FK (References Client)
Submitted_on	DATE	–	
Status	VARCHAR2	15	CHECK ('Pending', 'Accepted', 'Rejected')
Project_idea	VARCHAR2	200	

PAYMENT:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Payment_id	VARCHAR2	10	PK
Project_id	VARCHAR2	10	FK (References Project)
Amount	NUMBER(10,2)	–	NOT NULL
Paid_on	DATE	–	
Method	VARCHAR2	20	

RATING:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Rating_id	VARCHAR2	10	PK
Project_id	VARCHAR2	10	FK (References Project)
Client_id	VARCHAR2	10	FK (References Client)
Stars	NUMBER	1	CHECK (1–5)
Feedback	VARCHAR2	200	
Date	DATE	–	

INVOICE:

ATTRIBUTES	DATA TYPE	SIZE	CONSTRAINTS
Invoice_id	VARCHAR2	10	PK
Project_id	VARCHAR2	10	FK (References Project)
Issued_date	DATE	–	
Due_date	DATE	–	
Status	VARCHAR2	10	CHECK ('Paid', 'Unpaid')
Amount	NUMBER(10,2)	–	

CREATE TABLE STATEMENTS:

1. PERSON Table (Superclass)

```
CREATE TABLE PERSON (
  person_id NUMBER PRIMARY KEY,
  name VARCHAR2(100),
  email VARCHAR2(100)
);
```

2. FREELANCER Table (Subclass of PERSON)

```
CREATE TABLE FREELANCER (
  freelancer_id NUMBER PRIMARY KEY,
  field VARCHAR2(100),
  join_date DATE,
  CONSTRAINT fk_freelancer_person FOREIGN KEY (freelancer_id) REFERENCES PERSON(person_id)
);
```

3. CLIENT Table (Subclass of PERSON)

```
CREATE TABLE CLIENT (
  client_id NUMBER PRIMARY KEY,
  company VARCHAR2(100),
  country VARCHAR2(50),
  CONSTRAINT fk_client_person FOREIGN KEY (client_id) REFERENCES PERSON(person_id)
);
```

4. PROJECT Table

```
CREATE TABLE PROJECT (
  project_id NUMBER PRIMARY KEY,
  freelancer_id NUMBER,
  client_id NUMBER,
  title VARCHAR2(150),
  description CLOB,
  start_date DATE,
  end_date DATE,
  status VARCHAR2(20),
  CONSTRAINT fk_project_freelancer FOREIGN KEY (freelancer_id) REFERENCES FREELANCER(freelancer_id),
  CONSTRAINT fk_project_client FOREIGN KEY (client_id) REFERENCES CLIENT(client_id)
);
```

5. MILESTONES Table

```
CREATE TABLE milestones (
  milestone_id     INT PRIMARY KEY,
  project_id       INT REFERENCES project(project_id),
  title            VARCHAR2(100),
  due_date         DATE,
  is_completed     NUMBER(1) CHECK (is_completed IN (0, 1))
);
```

6. PROPOSAL Table

```
CREATE TABLE PROPOSAL (
  proposal_id NUMBER PRIMARY KEY,
  freelancer_id NUMBER,
  client_id NUMBER,
  submitted_on DATE,
  status VARCHAR2(20),
  project_idea CLOB,
  CONSTRAINT fk_proposal_freelancer FOREIGN KEY (freelancer_id) REFERENCES FREELANCER(freelancer_id),
  CONSTRAINT fk_proposal_client FOREIGN KEY (client_id) REFERENCES CLIENT(client_id)
);
```

7. PAYMENT Table

```
CREATE TABLE PAYMENT (
  payment_id NUMBER PRIMARY KEY,
  project_id NUMBER,
  amount NUMBER(10, 2),
  paid_on DATE,
  method VARCHAR2(50),
  CONSTRAINT fk_payment_project FOREIGN KEY (project_id) REFERENCES PROJECT(project_id)
);
```

8. RATING Table

```
CREATE TABLE RATING (
  rating_id NUMBER PRIMARY KEY,
  project_id NUMBER,
  client_id NUMBER,
  stars NUMBER(1) CHECK (stars BETWEEN 1 AND 5),
  feedback CLOB,
  rating_date DATE,
  CONSTRAINT fk_rating_project FOREIGN KEY (project_id) REFERENCES PROJECT(project_id),
  CONSTRAINT fk_rating_client FOREIGN KEY (client_id) REFERENCES CLIENT(client_id)
);
```

9. INVOICE Table

```
CREATE TABLE INVOICE (
  invoice_id NUMBER PRIMARY KEY,
  project_id NUMBER,
  issued_date DATE,
  due_date DATE,
  status VARCHAR2(20),
  amount NUMBER(10, 2),
  CONSTRAINT fk_invoice_project FOREIGN KEY (project_id) REFERENCES PROJECT(project_id)
);
```

TWO IMPORTANT VIEWS:

View1: freelancer_project_summary:

```
CREATE OR REPLACE VIEW freelancer_project_summary AS
SELECT
  f.freelancer_id,
  p.name AS freelancer_name,
  f.field,
  COUNT(DISTINCT pr.project_id) AS total_projects,
  NVL(AVG(r.stars), 0) AS average_rating,
  NVL(SUM(pay.amount), 0) AS total_earnings
FROM
  freelancer f
JOIN person p ON f.freelancer_id = p.person_id
LEFT JOIN project pr ON f.freelancer_id = pr.freelancer_id
LEFT JOIN payment pay ON pr.project_id = pay.project_id
LEFT JOIN rating r ON pr.project_id = r.project_id
GROUP BY
  f.freelancer_id, p.name, f.field;
```

```
select * from freelancer_project_summary
```

Results Explain Describe Saved SQL History

FREELANCER_ID	FREELANCER_NAME	FIELD	TOTAL_PROJECTS	AVERAGE_RATING	TOTAL_EARNINGS
2	Ali Khan	Graphic Design	0	0	0
1	Zoha Shabbir	Web Development	1	5	60000
3	Sara Malik	Graphic Design	1	0	40000

3 rows returned in 0.01 seconds Download

VIEW2. top_earning_clients:

```
CREATE OR REPLACE VIEW top_earning_clients_view AS
SELECT
  c.client_id,
  p.name AS client_name,
  SUM(pay.amount) AS total_paid
FROM
  client c
JOIN person p ON c.client_id = p.person_id
JOIN project pr ON c.client_id = pr.client_id
JOIN payment pay ON pr.project_id = pay.project_id
GROUP BY c.client_id, p.name
ORDER BY total_paid DESC
FETCH FIRST 5 ROWS ONLY;
```

```
select * from top_earning_clients_view
```

Results Explain Describe Saved SQL History

CLIENT_ID	CLIENT_NAME	TOTAL_PAID
2	Ali Khan	60000
4	Umer Raza	40000

2 rows returned in 0.01 seconds Download

Important Reports to be generated by system:

Report 1: Clients Who Haven't Paid for Any Project

```
SELECT
  c.client_id,
  p.name AS client_name
FROM
  client c
JOIN person p ON c.client_id = p.person_id
WHERE
  NOT EXISTS (
    SELECT 1
    FROM project pr
    JOIN payment pay ON pr.project_id = pay.project_id
    WHERE pr.client_id = c.client_id
  );
```

Results Explain Describe Saved SQL History

CLIENT_ID	CLIENT_NAME
202	Fatima Sheikh
201	Ahmad Khan
204	Sara Ali
203	Zeeshan Tariq
205	Tariq Javed

5 rows returned in 0.01 seconds Download

Report 2: Freelancers Without Any Rating Yet

```
SELECT
  f.freelancer_id,
  p.name AS freelancer_name
FROM
  freelancer f
JOIN person p ON f.freelancer_id = p.person_id
WHERE
  NOT EXISTS (
    SELECT 1
    FROM project pr
    JOIN rating r ON pr.project_id = r.project_id
    WHERE pr.freelancer_id = f.freelancer_id
  );
```

Results Explain Describe Saved SQL History

FREELANCER_ID	FREELANCER_NAME
3	Sara Malik
2	Ali Khan

2 rows returned in 0.00 seconds Download

Report 3: Total Income Per Client (via Subquery)

```
SELECT
  c.client_id,
  p.name AS client_name,
  (
    SELECT SUM(pay.amount)
    FROM project pr
    JOIN payment pay ON pr.project_id = pay.project_id
    WHERE pr.client_id = c.client_id
  ) AS total_paid
FROM
  client c
JOIN person p ON c.client_id = p.person_id;
```

Results Explain Describe Saved SQL History

CLIENT_ID	CLIENT_NAME	TOTAL_PAID
2	Ali Khan	60000
4	Umer Raza	40000
201	Ahmad Khan	-
202	Fatima Sheikh	-
203	Zeeshan Tariq	-
204	Sara Ali	-
205	Tariq Javed	-

7 rows returned in 0.01 seconds Download

Report 4: Proposal Conversion Rate by Freelancer

```
SELECT
  f.freelancer_id,
  p.name AS freelancer_name,
  COUNT(*) AS total_proposals,
  SUM(CASE WHEN prop.status = 'Accepted' THEN 1 ELSE 0 END) AS accepted_count,
  ROUND(
    (SUM(CASE WHEN prop.status = 'Accepted' THEN 1 ELSE 0 END) / COUNT(*)) * 100,
    2
  ) AS acceptance_rate
FROM
  freelancer f
JOIN person p ON f.freelancer_id = p.person_id
JOIN proposal prop ON f.freelancer_id = prop.freelancer_id
GROUP BY
  f.freelancer_id, p.name;
```

Results Explain Describe Saved SQL History

FREELANCER_ID	FREELANCER_NAME	TOTAL_PROPOSALS	ACCEPTED_COUNT	ACCEPTANCE_RATE
3	Sara Malik	1	1	100
1	Zoha Shabbir	1	1	100

2 rows returned in 0.00 seconds Download

Report 5: Monthly Earnings Summary

```
SELECT
  TO_CHAR(paid_on, 'YYYY-MM') AS month,
  SUM(amount) AS total_earnings
FROM
  payment
GROUP BY
  TO_CHAR(paid_on, 'YYYY-MM')
ORDER BY
  month;
```

Results Explain Describe Saved SQL History

MONTH	TOTAL_EARNINGS
2024-07	75000
2025-07	25000

2 rows returned in 0.00 seconds Download

FUNCTIONS:

- **Get Average Rating of a Freelancer**

```
CREATE OR REPLACE FUNCTION get_avg_rating(f_id IN NUMBER)
RETURN NUMBER
IS
  avg_rating NUMBER;
BEGIN
  SELECT NVL(AVG(r.stars), 0)
  INTO avg_rating
  FROM rating r
  JOIN project p ON r.project_id = p.project_id
  WHERE p.freelancer_id = f_id;

  RETURN avg_rating;
END;
/
```

Results Explain Describe Saved SQL History

```
Function created.
```

```
SELECT get_avg_rating(101) FROM dual;
```

Results Explain Describe Saved SQL History

GET_AVG_RATING(101)
0

1 rows returned in 0.00 seconds Download

PROCEDURE:

- **Mark Milestone as Completed**

```
CREATE OR REPLACE PROCEDURE mark_milestone_complete(
  m_id IN NUMBER
)
IS
BEGIN
  UPDATE milestone
  SET is_completed = 'Y'
  WHERE milestone_id = m_id;

  COMMIT;
END;
/
```

Results Explain Describe Saved SQL History

```
Procedure created.
```

0.03 seconds

```
BEGIN
  mark_milestone_complete(301);
END;
```

Results Explain Describe Saved SQL History

```
Statement processed.
```

0.01 seconds

TRIGGER

- **Auto-generate Invoice After Project Completion**

```
CREATE OR REPLACE TRIGGER trg_create_invoice
AFTER UPDATE OF status ON project
FOR EACH ROW
WHEN (NEW.status = 'Completed' AND OLD.status <> 'Completed')
DECLARE
  new_invoice_id invoice.invoice_id%TYPE;
BEGIN
  SELECT NVL(MAX(invoice_id), 0) + 1 INTO new_invoice_id FROM invoice;

  INSERT INTO invoice (
    invoice_id, project_id, issued_date, due_date, status, amount
  )
  VALUES (
    new_invoice_id,
    :NEW.project_id,
    SYSDATE,
    SYSDATE + 15,
    'Unpaid',
    (SELECT NVL(SUM(amount), 0)
     FROM payment
     WHERE project_id = :NEW.project_id)
  );
END;
```

Results Explain Describe Saved SQL History

```
Trigger created.
```

0.08 seconds

Current status:

```
SELECT project_id, title, status FROM project WHERE project_id = 101;
```

Results Explain Describe Saved SQL History

PROJECT_ID	TITLE	STATUS
101	Portfolio Website	Ongoing

1 rows returned in 0.01 seconds Download

After applying trigger:

```
UPDATE project
SET status = 'Completed'
WHERE project_id = 101;
```

Results Explain Describe Saved SQL History

1 row(s) updated.

0.01 seconds

Final result shows the auto-generated invoice:

```
SELECT * FROM invoice WHERE project_id = 101;
```

Results Explain Describe Saved SQL History

INVOICE_ID	PROJECT_ID	ISSUED_DATE	DUE_DATE	STATUS	AMOUNT
501	101	07/14/2024	07/21/2024	Paid	20000
503	101	07/15/2025	07/30/2025	Unpaid	60000
504	101	07/15/2025	07/30/2025	Unpaid	60000

3 rows returned in 0.01 seconds Download

Conclusion:

The *Freelancer Hustle DB* project helped me understand how to design and implement a real-world relational database system using Oracle SQL and PL/SQL. I worked with normalized tables, complex relationships, and implemented useful features like views, stored procedures, and triggers.

This project improved my skills in writing efficient queries, managing business logic in the database, and thinking critically about data organization and integrity. Overall, it was a valuable hands-on learning experience.