Constructs in SQL:

Constructs in SQL refer to the various elements or components used to construct and manipulate databases and their data.

These constructs include:

Data Definition Language (DDL):

DDL is used to define, modify, and delete database objects such as tables, indexes, and views. Common DDL commands include CREATE, ALTER, and DROP.

Data Manipulation Language (DML):

DML is used to manipulate data within the database. It includes commands like SELECT, INSERT, UPDATE, and DELETE, which allow users to retrieve, insert, modify, and delete data from tables.

Data Control Language (DCL):

DCL is used to control access to data within the database. It includes commands like GRANT and REVOKE, which grant or revoke permissions to users or roles.

Transaction Control Language (TCL):

TCL is used to manage transactions within the database. It includes commands like COMMIT, ROLLBACK, and SAVEPOINT, which control the beginning and ending of transactions and manage transaction boundaries.

Data Query Language (DQL):

DQL is used to retrieve data from the database. It primarily consists of the SELECT statement, which allows users to query the database and retrieve specific data based on specified criteria.

Constraints:

Constraints are rules defined on columns or tables to enforce data integrity and maintain consistency. Common constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK constraints.

Functions and Operators:

SQL provides a wide range of built-in functions and operators for performing operations on data. Functions include aggregate functions (e.g., SUM, AVG, COUNT), string functions (e.g., CONCAT, SUBSTRING), mathematical functions (e.g., ROUND, ABS), and date/time functions (e.g., DATEADD, DATEDIFF).

```
-- Concatenate first name and last name of employees
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees;

-- Extract substring from a string
SELECT SUBSTRING(product_name, 1, 5) AS short_name FROM products;

-- Convert string to uppercase
SELECT UPPER(product_name) AS uppercase_name FROM products;
```

Joins:

Joins are used to combine data from multiple tables based on related columns. Common join types include INNER JOIN, LEFT JOIN (or LEFT OUTER JOIN), RIGHT JOIN (or RIGHT OUTER JOIN), and FULL JOIN (or FULL OUTER JOIN).

Subqueries: Subqueries are queries nested within another query. They can be used to retrieve data or perform calculations based on the results of another query.

```
SELECT product_id, product_name, price
FROM products
WHERE price = (
    SELECT MAX(price)
    FROM products
);
```

```
SELECT *
FROM products
WHERE price > (SELECT AVG(price) FROM products);
```

```
SELECT *
FROM employees e
WHERE salary = (
   SELECT MAX(salary)
   FROM employees
   WHERE department_id = e.department_id
);
```

```
SELECT *
FROM departments d
WHERE (
   SELECT COUNT(*)
   FROM employees
   WHERE department_id = d.department_id
) > 5;
```

Views: Views are virtual tables generated by queries. They allow users to simplify complex queries, encapsulate business logic, and provide a layer of abstraction over the underlying data.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
CREATE VIEW sales_employees AS
SELECT employee_id, first_name, last_name
FROM employees
WHERE department_id = 1;
-----------------------------------------------------------------------

SELECT * FROM sales_employees;
-----------------------------------------------------------------------
DROP VIEW view_name;
-----------------------------------------------------------------------

CREATE VIEW expensive_products AS
SELECT product_id, product_name, price
FROM products
WHERE price > 10000;
-----------------------------------------------------------------------------
SELECT * FROM expensive_products;
-----------------------------------------------------------------------------
```

Data collection:

In a Database Management System (DBMS), data collection refers to the process of gathering, storing, and organizing data within the database. Here's how data collection works in a DBMS:

Data Sources: Data can originate from various sources, including manual entry by users, external files, sensors, application logs, web forms, and other databases.

Data Entry: Users or automated processes enter data into the database. This can involve creating new records, updating existing ones, or deleting outdated information.

Data Validation: Before data is stored in the database, it undergoes validation to ensure accuracy, completeness, and consistency. Validation rules are applied to verify that the data meets specified criteria and constraints.

Data Storage: Once validated, the data is stored in tables within the database. Each table consists of rows and columns, with each row representing a record and each column representing a data attribute or field.

Data Indexing: To facilitate efficient retrieval and manipulation of data, indexes are created on certain columns in the tables. Indexes speed up query performance by allowing the DBMS to quickly locate records based on specific criteria.

Data Maintenance: Regular maintenance tasks such as data backup, data archiving, and database optimization are performed to ensure data integrity, availability, and performance.

Data Retrieval: Users can retrieve data from the database using Structured Query Language (SQL) queries. SQL allows users to specify the data they want to retrieve and the criteria for selecting it.

Data Analysis: Once retrieved, data can be analyzed using various techniques such as reporting, data mining, machine learning, and statistical analysis. This analysis helps derive insights, make informed decisions, and identify patterns or trends in the data.

Data Security: Measures are implemented to protect the data from unauthorized access, alteration, or disclosure. This includes user authentication, encryption, access control, and auditing mechanisms.

Data Integration: In some cases, data from multiple sources or databases may need to be integrated or consolidated to provide a unified view. This process involves mapping, transforming, and loading data from different sources into a single database or data warehouse.

Overall, data collection in a DBMS involves managing the entire lifecycle of data, from its initial capture to its eventual storage, retrieval, analysis, and utilization. It plays a crucial role in supporting business operations, decision-making processes, and strategic initiatives.

Designing a database schema:

Requirement Analysis: The first step is to gather and analyze the requirements of the system or application for which the database will be designed. This involves understanding the data entities, relationships between them, data attributes, and constraints.

Conceptual Design: In this phase, a conceptual schema is created that represents the high-level view of the database structure, independent of any specific DBMS. Entity-Relationship Diagrams (ERDs) are commonly used to model the entities, their attributes, and the relationships between them.

Normalization: The conceptual schema is normalized to remove redundancy and improve data integrity. Normalization involves decomposing tables into smaller, more manageable entities and defining relationships between them to minimize data duplication.

Logical Design: Based on the normalized conceptual schema, a logical schema is developed that defines the structure of the database using the data model supported by the chosen DBMS (e.g., relational model, hierarchical model, network model). Tables, columns, primary keys, foreign keys, and indexes are defined in the logical schema.

Physical Design: The physical schema specifies how the logical database schema is implemented in the underlying storage structures of the DBMS. This includes decisions such as data types, storage allocation, indexing strategies, partitioning, and clustering.

Denormalization (Optional): In some cases, denormalization may be applied to optimize performance by reintroducing redundancy into the database schema. This can improve query performance by reducing the need for complex joins.

Schema Refinement: The database schema is refined based on feedback from stakeholders, performance testing, and ongoing maintenance requirements. This may involve fine-tuning indexes, optimizing queries, and adjusting the schema to accommodate changing business needs.

Documentation: Comprehensive documentation of the database schema is created, including data dictionary, entity-relationship diagrams, schema diagrams, and any additional documentation required for database administration and development.

Implementation: Finally, the database schema is implemented by creating the tables, indexes, views, and other database objects within the DBMS. Data migration may also be required to populate the database with initial data from existing sources.

Testing and Validation: The database schema is thoroughly tested to ensure that it meets the requirements, performs efficiently, and maintains data integrity. This includes unit testing, integration testing, and validation against real-world scenarios.

Normal Forms (NF):

Normal forms are rules used to minimize data redundancy and dependency in relational databases. They help ensure that the database is organized efficiently, reducing the risk of anomalies and inconsistencies in the data. There are several normal forms, each representing a different level of normalization:

First Normal Form (1NF): Ensures that each table has a primary key and that each column contains atomic values, i.e., no repeating groups or arrays.

1. **First Normal Form (1NF):**
   - 1NF ensures that each column in a table contains atomic values, meaning it holds only single values and does not contain repeating groups or arrays. Each cell should have a single value.
   - Example: Consider a table storing student information. Each student can have multiple phone numbers. In 1NF, instead of having a single column for phone numbers where multiple numbers are stored together (e.g., "123-456-7890, 987-654-3210"), you would create a separate row for each phone number associated with a student, like this:

Student ID	Phone Number
1	123-456-7890
1	987-654-3210
2	555-555-5555

Second Normal Form (2NF): Builds upon 1NF and ensures that all non-key attributes are fully dependent on the primary key. This eliminates partial dependencies.

Student ID	Course ID	Course Name	Course Instructor
1	101	Mathematics 101	Prof. Smith
1	102	Physics 101	Prof. Johnson
2	101	Mathematics 101	Prof. Smith

Third Normal Form (3NF): Builds upon 2NF and ensures that there are no transitive dependencies, meaning that non-key attributes are not dependent on other non-key attributes.

Student ID	Course ID	Course Instructor	Instructor Department
1	101	Prof. Smith	Mathematics
1	102	Prof. Johnson	Physics
2	101	Prof. Smith	Mathematics

InstructorDepartment (new table):

Instructor	Department
Prof. Smith	Mathematics
Prof. Johnson	Physics

Boyce-Codd Normal Form (BCNF): A stricter version of 3NF, which further eliminates anomalies by ensuring that every determinant is a candidate key.

Employee ID	Project ID	Project Name	Employee Name
1	101	Project A	John
1	102	Project B	John
2	101	Project A	Alice

Employee Projects Table:

Employee ID	Project ID
1	101
1	102
2	101

Employee Information Table:

Employee ID	Employee Name
1	John
2	Alice

Fourth Normal Form (4NF): Addresses multi-valued dependencies, ensuring that each attribute is fully dependent on the primary key.

Course ID	Textbook ID	Textbook Title	Author
101	1	Physics	John Smith
101	2	Chemistry	Jane Doe
102	1	Physics	John Smith
102	3	Biology	Mark Johnson

Courses Table:

Course ID	Textbook ID
101	1
101	2
102	1
102	3

Textbooks Table:

Textbook ID	Textbook Title	Author
1	Physics	John Smith
2	Chemistry	Jane Doe
3	Biology	Mark Johnson

Fifth Normal Form (5NF) and beyond: Address additional complex dependencies beyond 4NF.

Entity-Relationship (ER) Diagram:

ER diagrams are graphical representations of the logical structure of a database. They depict entities (things of interest) and their relationships, along with attributes associated with entities. Key components of an ER diagram include:

Entity: Represents a real-world object or concept, such as a person, place, or thing. Entities are depicted as rectangles in the ER diagram.

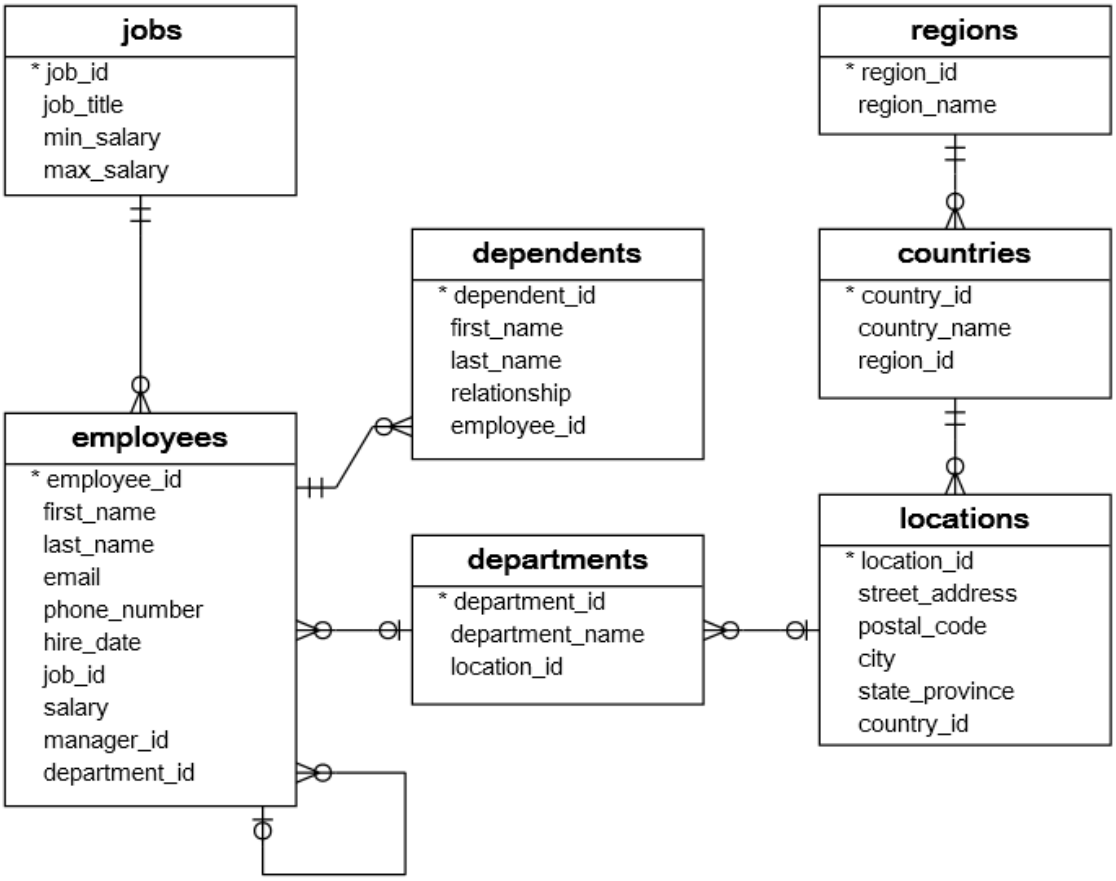
Attribute: Describes a characteristic or property of an entity. Attributes are represented within ovals connected to their respective entities.

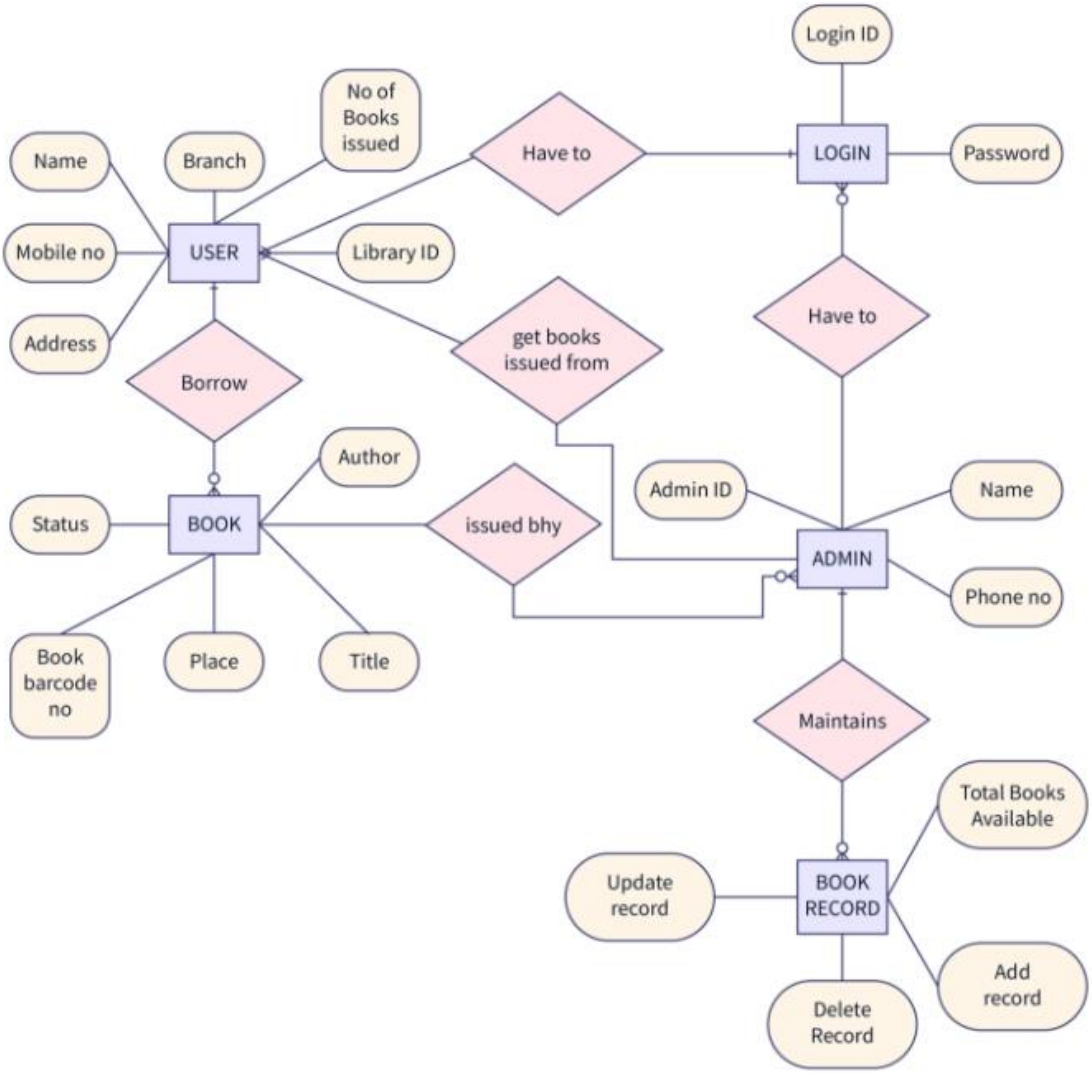
Relationship: Describes how entities are related to each other. Relationships are depicted as lines connecting entities, with cardinality (the number of instances of one entity that are associated with a single instance of another entity) and participation constraints (whether participation in the relationship is mandatory or optional) indicated on each end.

Key Attribute: An attribute or combination of attributes that uniquely identifies each instance of an entity. Key attributes are underlined in the ER diagram.

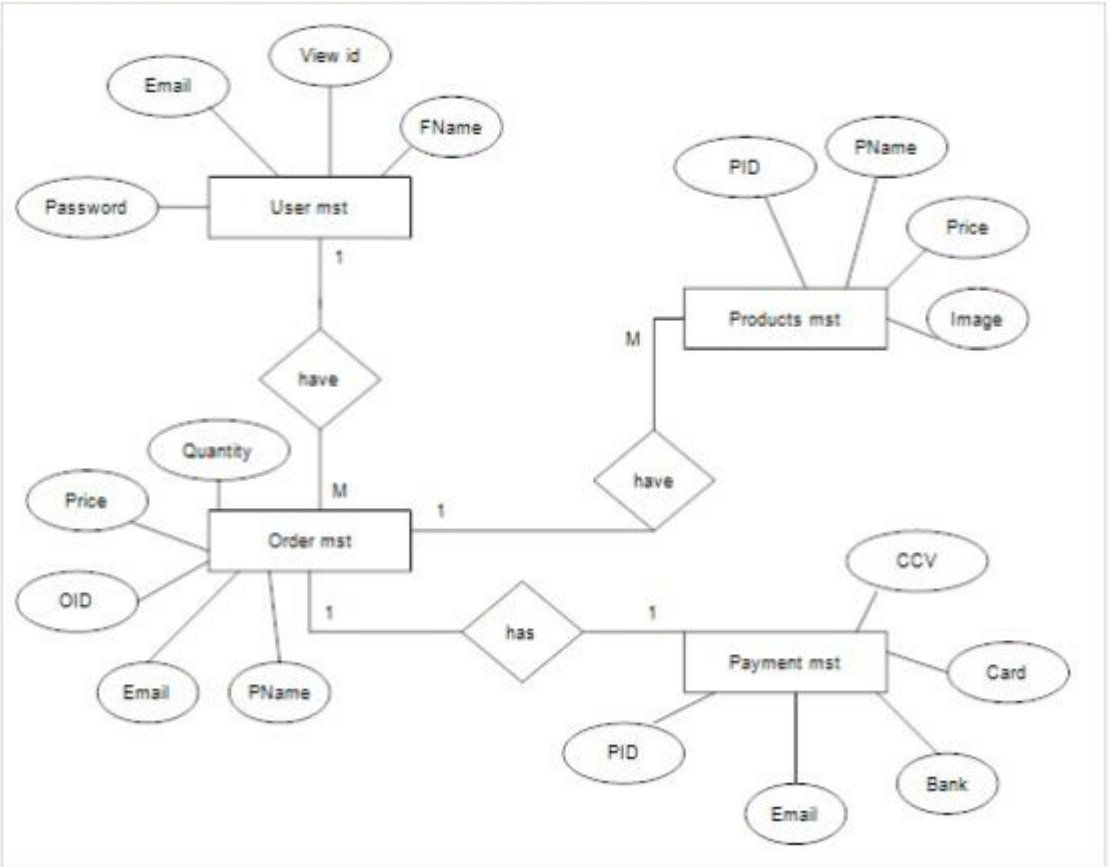
Composite Attribute: An attribute that can be further subdivided into smaller attributes.

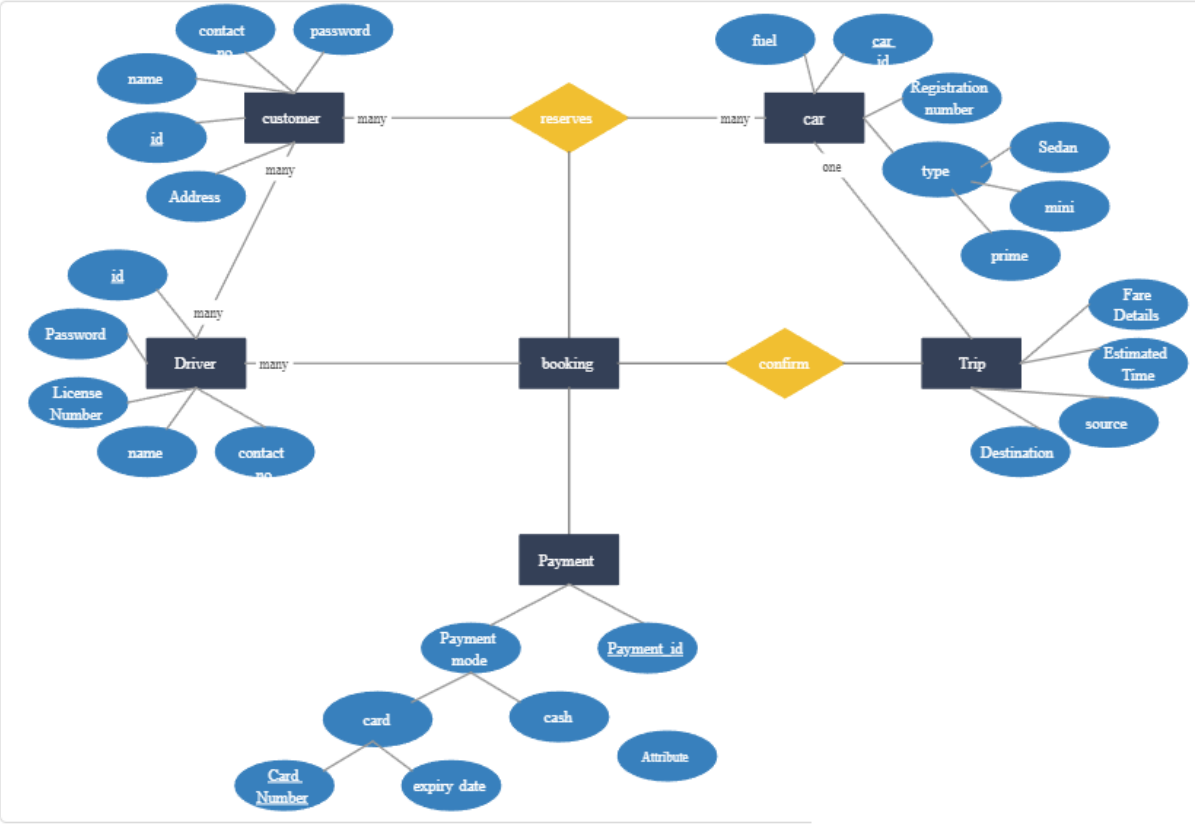
Derived Attribute: An attribute whose value can be derived or calculated from other attributes.


jobs
* job_id
job_title
min_salary
max_salary
regions
* region_id
region_name
dependents
* dependent_id
first_name
last_name
relationship
employee_id
countries
* country_id
country_name
region_id
employees
* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id
departments
* department_id
department_name
location_id
locations
* location_id
street_address
postal_code
city
state_province
country_id


Login ID
No of Books issued
Name
Branch
Have to
LOGIN
Password
Mobile no
USER
Library ID
Have to
Address
get books issued from
Borrow
Author
Admin ID
Name
Status
BOOK
issued bhy
ADMIN
Phone no
Book barcode no
Place
Title
Maintains
Total Books Available
Update record
BOOK RECORD
Add record
Delete Record

E-R Diagram for Online Food System




contact no
password
name
id
customer
many
reserves
many
car
fuel
car_id
Registration number
type
Sedan
mini
prime
one
Address
many
id
many
Password
Driver
many
booking
confirm
Trip
Fare Details
Estimated Time
source
Destination
License Number
name
contact no
Payment
Payment mode
Payment_id
card
cash
Attribute
Card_Number
expiry date

A stored procedure in MySQL is a precompiled collection of SQL statements and procedural logic stored in the database catalog. It is compiled once and can be invoked multiple times, enhancing performance and promoting code modularity.

```
DELIMITER //

CREATE PROCEDURE GetEmployeeCount()
BEGIN
   SELECT COUNT(*) AS TotalEmployees FROM employees;
END //

DELIMITER ;
```

Execution:

```
CALL GetEmployeeCount();
```

Result:

TotalEmployees
10

Creating a Stored Procedure with Parameters:

```
DELIMITER //
```

```
CREATE PROCEDURE GetEmployeeByDept(IN deptName VARCHAR(50))
BEGIN
   SELECT * FROM employees WHERE department = deptName;
END //
DELIMITER ;
```

Execution:

```
CALL GetEmployeeByDept('IT');
```

Result:

```
-----------------------------------------------
| id  |  name    | department | designation |
-----------------------------------------------
| 101 | John     | IT         | Developer   |
| 102 | Jane     | IT         | Analyst     |
```