Here's a step-by-step breakdown of the SQL project for an Inventory Management System, along with comments explaining each part of the code. This project involves creating tables, inserting data, and using PL/SQL for functions, cursors, and procedures.

Step 1: Creating Tables

1.1 Create `brands` Table

```
CREATE TABLE brands(
  bid NUMBER(5),         -- Brand ID, numeric type with a max of 5 digits
  bname VARCHAR(20)      -- Brand name, string type with a max length of 20 characters
);
```

The `brands` table stores brand information, including a unique ID (`bid`) and the brand's name (`bname`).

1.2 Set Primary Key for `brands`

```
ALTER TABLE brands
ADD PRIMARY KEY(bid);
```

This command sets the primary key for the `brands` table to ensure that `bid` is unique and not null.

1.3 Create `inv_user` Table

```
CREATE TABLE inv_user(
  user_id VARCHAR(20),   -- Unique user ID, string type
  name VARCHAR(20),      -- User's name, string type
  password VARCHAR(20),  -- User's password, string type
  last_login TIMESTAMP,  -- Last login timestamp
  user_type VARCHAR(10)  -- Type of user (e.g., admin, manager)
);
```

The `inv_user` table holds user credentials and information related to inventory system users.

1.4 Set Primary Key for `inv_user`

```
ALTER TABLE inv_user
ADD PRIMARY KEY(user_id);
```

This command sets the `user_id` as the primary key for the `inv_user` table.

1.5 Create `categories` Table

```
CREATE TABLE categories(
  cid NUMBER(5),         -- Category ID, numeric type
```

```
  category_name VARCHAR(20) -- Name of the category, string type
);
```

The `categories` table defines product categories with a unique ID and name.

1.6 Set Primary Key for `categories`

```
ALTER TABLE categories
ADD PRIMARY KEY(cid);
```
This sets the `cid` as the primary key for the `categories` table.

1.7 Create `product` Table

```
CREATE TABLE product(
  pid NUMBER(5) PRIMARY KEY,    -- Product ID, primary key
  cid NUMBER(5) REFERENCES categories(cid),  -- Foreign key referencing categories
  bid NUMBER(5) REFERENCES brands(bid),      -- Foreign key referencing brands
  sid NUMBER(5),                -- Store ID
  pname VARCHAR(20),            -- Product name
  p_stock NUMBER(5),            -- Product stock quantity
  price NUMBER(5),              -- Product price
  added_date DATE               -- Date when the product was added
);
```

The `product` table contains details of the products in the inventory, linking to both `categories` and `brands` via foreign keys.

1.8 Create `stores` Table

```
CREATE TABLE stores(
  sid NUMBER(5),          -- Store ID, numeric type
  sname VARCHAR(20),      -- Store name, string type
  address VARCHAR(20),    -- Store address, string type
  mobno NUMBER(10)        -- Store mobile number
);
```

The `stores` table contains information about different stores that hold inventory.

1.9 Set Primary Key for `stores`

```
ALTER TABLE stores
ADD PRIMARY KEY(sid);
```

This sets the primary key for the `stores` table, ensuring `sid` is unique.

1.10 Set Foreign Key for `product`

```
ALTER TABLE product
ADD FOREIGN KEY(sid) REFERENCES stores(sid);
```

This establishes a foreign key relationship from `product` to `stores`, linking products to their respective stores.

1.11 Create `provides` Table

```
CREATE TABLE provides(
  bid NUMBER(5) REFERENCES brands(bid),  -- Foreign key referencing brands
  sid NUMBER(5) REFERENCES stores(sid),   -- Foreign key referencing stores
  discount NUMBER(5)                    -- Discount offered by the store on the brand
);
```

The `provides` table links brands and stores, showing what discounts stores provide on different brands.

1.12 Create `customer_cart` Table

```
CREATE TABLE customer_cart(
  cust_id NUMBER(5) PRIMARY KEY, -- Customer ID, primary key
  name VARCHAR(20),              -- Customer name
  mobno NUMBER(10)               -- Customer mobile number
);
```

This table represents customer carts, linking customers to their shopping cart details.

1.13 Create `select_product` Table

```
CREATE TABLE select_product(
  cust_id NUMBER(5) REFERENCES customer_cart(cust_id),  -- Foreign key referencing customer_cart
  pid NUMBER(5) REFERENCES product(pid),                -- Foreign key referencing product
  quantity NUMBER(4)                                    -- Quantity of the product selected
);
```

The `select_product` table keeps track of products that customers have selected in their carts.

1.14 Create `transaction` Table

```
CREATE TABLE transaction(
  id NUMBER(5) PRIMARY KEY,     -- Transaction ID, primary key
  total_amount NUMBER(5),       -- Total amount of the transaction
  paid NUMBER(5),               -- Amount paid by the customer
  due NUMBER(5),                -- Amount due
  gst NUMBER(3),                -- Goods and Services Tax
  discount NUMBER(5),           -- Discount applied to the transaction
  payment_method VARCHAR(10),    -- Method of payment (e.g., cash, card)
  cart_id NUMBER(5) REFERENCES customer_cart(cust_id) -- Foreign key referencing customer_cart
);


##  The `transaction` table records all the transactions made, linking to the `customer_cart` to track which cart was used.
```

1.15 Create `invoice` Table

```
CREATE TABLE invoice(
  item_no NUMBER(5),            -- Item number in the invoice
  product_name VARCHAR(20),      -- Name of the product
  quantity NUMBER(5),           -- Quantity of the product
  net_price NUMBER(5),          -- Net price of the product
  transaction_id NUMBER(5) REFERENCES transaction(id) -- Foreign key referencing transaction
);


##  The `invoice` table keeps track of all items in transactions, linking to the corresponding transaction details.
```

Step 2: Inserting Data

2.1 Insert Data into `brands`

```
INSERT INTO brands VALUES(1, 'Apple');
INSERT INTO brands VALUES(2, 'Samsung');
INSERT INTO brands VALUES(3, 'Nike');
INSERT INTO brands VALUES(4, 'Fortune');
```

Inserts various brand entries into the `brands` table.

2.2 Insert Data into `inv_user`

```
INSERT INTO inv_user VALUES('vidit@gmail.com', 'vidit', '1234', '31-oct-18 12:40', 'admin');
INSERT INTO inv_user VALUES('harsh@gmail.com', 'Harsh Khanelwal', '1111', '30-oct-18 10:20', 'Manager');
INSERT INTO inv_user VALUES('prashant@gmail.com', 'Prashant', '0011', '29-oct-18 10:20', 'Accountant');
```

Inserts user entries into the `inv_user` table for system access.

2.3 Insert Data into `categories`

```
INSERT INTO categories VALUES(1, 'Electronics');
INSERT INTO categories VALUES(2, 'Clothing');
INSERT INTO categories VALUES(3, 'Grocery');
```

Adds product categories to the `categories` table.

2.4 Insert Data into `stores`

```
INSERT INTO stores VALUES(1, 'Ram Kumar', 'Katpadi Vellore', 9999999999);
INSERT INTO stores VALUES(2, 'Rakesh Kumar', 'Chennai', 8888555541);
INSERT INTO stores VALUES(3, 'Suraj', 'Haryana', 7777555541);
```

Inserts store details into the `stores` table.

2.5 Insert Data into `product`

```
INSERT INTO product VALUES(1, 1, 1, 1, 'IPHONE', 4, 45000, '31-oct-18');
INSERT INTO product VALUES(2, 1, 1, 1, 'Airpods', 3, 19000, '27-oct-18');
INSERT INTO product VALUES(3, 1, 1, 1, 'Smart Watch', 3, 19000, '27-oct-18');
```

```
INSERT INTO product VALUES(4, 2, 3, 2, 'Air Max', 6, 7000, '27-oct-18');
INSERT INTO product VALUES(5, 3, 4, 3, 'REFINED OIL', 6, 750, '25-oct-18');
```

Inserts product information into the `product` table, linking each product to a category, brand, and store.

2.6 Insert Data into `provides`

```
INSERT INTO provides VALUES(1, 1, 10);
INSERT INTO provides VALUES(2, 2, 5);
INSERT INTO provides VALUES(3, 3, 20);
```

This establishes what discounts are provided by which store for each brand.

2.7 Insert Data into `customer_cart`

```
INSERT INTO customer_cart VALUES(1, 'Vidit', 9999999999);
INSERT INTO customer_cart VALUES(2, 'Harsh', 8888555541);
INSERT INTO customer_cart VALUES(3, 'Prashant', 7777555541);
```

Adds customer details to the `customer_cart` table.

2.8 Insert Data into `select_product`

```
INSERT INTO select_product VALUES(1, 1, 1);
INSERT INTO select_product VALUES(1, 3, 2);
INSERT INTO select_product VALUES(2, 2, 1);
```

Records products selected by customers in their carts.

2.9 Insert Data into `transaction`

```
INSERT INTO transaction VALUES(1, 20000, 19000, 1000, 1000, 1000, 'Credit Card', 1);
INSERT INTO transaction VALUES(2, 40000, 38000, 2000, 500, 500, 'Cash', 2);
```

Adds transaction records, detailing amounts and payment methods.

2.10 Insert Data into `invoice`

```
INSERT INTO invoice VALUES(1, 'IPHONE', 1, 45000, 1);
INSERT INTO invoice VALUES(2, 'Air Max', 1, 7000, 2);
```

Creates invoice records for the respective transactions.

Step 3: PL/SQL Procedures and Functions

3.1 Create Procedure to Add New Product

```
CREATE OR REPLACE PROCEDURE add_product(
  p_pid IN NUMBER,
  p_cid IN NUMBER,
  p_bid IN NUMBER,
  p_sid IN NUMBER,
  p_pname IN VARCHAR2,
  p_pstock IN NUMBER,
  p_price IN NUMBER,
  p_added_date IN DATE
) AS
BEGIN
  INSERT INTO product VALUES(p_pid, p_cid, p_bid, p_sid, p_pname, p_pstock, p_price, p_added_date);
  COMMIT;
END;
```

This procedure allows users to add a new product to the `product` table with specified attributes.

3.2 Create Function to Get Total Stock of a Product

```
CREATE OR REPLACE FUNCTION total_stock(p_pid IN NUMBER) RETURN NUMBER IS
  v_stock NUMBER;
BEGIN
  SELECT p_stock INTO v_stock FROM product WHERE pid = p_pid;
  RETURN v_stock;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
     RETURN 0;  -- Returns 0 if no product is found
END;
```

This function retrieves the total stock of a specified product by its ID.

3.3 Create Cursor to Display All Products

```
DECLARE
  CURSOR product_cursor IS
     SELECT * FROM product;
  v_product product%ROWTYPE;  -- Declares a variable to hold the row data
BEGIN
  OPEN product_cursor;  -- Opens the cursor
  LOOP
     FETCH product_cursor INTO v_product;  -- Fetches a row into the variable
     EXIT WHEN product_cursor%NOTFOUND;  -- Exits the loop if no more rows
     DBMS_OUTPUT.PUT_LINE('Product ID: ' || v_product.pid || ' - Product Name: ' || v_product.pname);
  END LOOP;
```

```
  CLOSE product_cursor;  -- Closes the cursor
END;
```

This block uses a cursor to display all products in the `product` table.

Summary:

This SQL project for an Inventory Management System covers the following:

1. Database Structure: Creation of tables with relationships and constraints.
2. Data Manipulation: Inserting data into the tables.
3. Stored Procedures and Functions: Using PL/SQL for creating reusable components.

This structure provides a solid foundation for managing inventory and can be expanded with more functionality as needed. Let me know if you need further explanations or modifications!