

Database Project

Pharmacy

This database is designed to track and manage various economic aspects of the pharmacy's operations, including customer information, medication inventory, employee information, and financial transactions. Let's take a look at some of the more important tables.

The **Medication** table stores detailed information about the pharmacy's medications, including the name, dosage, and instructions for use. This information is used to accurately fill prescriptions and track medication inventory.

The **Prescriptions** table stores information about prescriptions written by doctors and filled by the pharmacy, including the patient's name and medical history, the medication prescribed, and the dosage. This information is used to track the prescriptions filled by the pharmacy and ensure that patients receive the correct medication.

The pharmacy has a coupon system in place to offer discounts to customers. The **Coupons** table stores information about the available coupons, including the discount amount or expiration dates. The **Promotions** table stores information about current promotions, including the products or services being promoted and any applicable discounts.

The **Employees** and **Departments** tables store information about the pharmacy's employees and departments, respectively. This information is used for employee management and to track the performance of different departments.

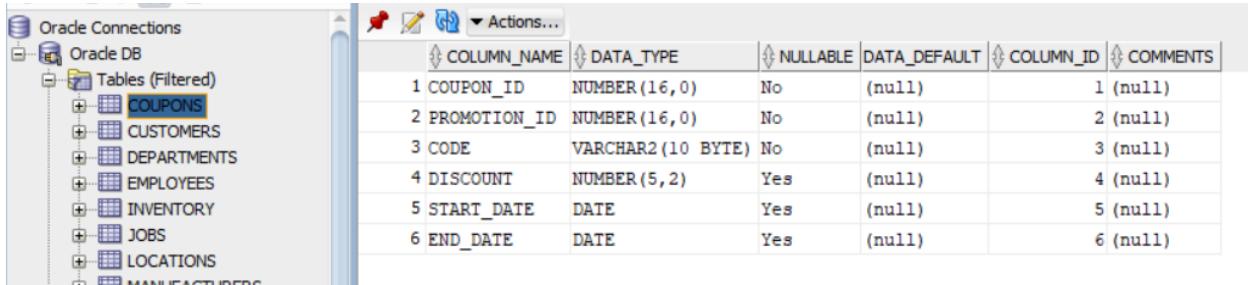
The **Payments** and **Orders** tables store information about payment methods and orders placed by customers, respectively. This information is used to process payments and track customer orders. The **Transactions** table stores information about completed transactions, including the payment method and the total cost of the transaction.

Overall, the pharmacy database is an important tool for managing the economic aspects of the pharmacy's operations and ensuring the smooth and efficient running of the business.

Let's first take a look at the database as a whole, after which we can see how all of this was accomplished.

Constructing the database: How it all came to be

When I first created the table most of it was done in a hurry, and I didn't think all the data through, of course the tables that I ended up with were very superficial, let me show some examples on how those tables would have looked like:



The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view of 'Oracle Connections' and 'Tables (Filtered)'. Under 'Tables (Filtered)', the 'COUPONS' table is selected and highlighted with a blue border. To the right is a detailed view of the 'COUPONS' table structure, including columns: COUPON_ID, PROMOTION_ID, CODE, DISCOUNT, START_DATE, and END_DATE. Each column has its data type, whether it's nullable, and a default value.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 COUPON_ID	NUMBER(16,0)	No	(null)	1	(null)
2 PROMOTION_ID	NUMBER(16,0)	No	(null)	2	(null)
3 CODE	VARCHAR2(10 BYTE)	No	(null)	3	(null)
4 DISCOUNT	NUMBER(5,2)	Yes	(null)	4	(null)
5 START_DATE	DATE	Yes	(null)	5	(null)
6 END_DATE	DATE	Yes	(null)	6	(null)

The “coupons” table was done poorly, I created it using this command:

```
CREATE TABLE Coupons (
    coupon_id number(16) PRIMARY KEY,
    promotion_id number(16) NOT NULL,
    code varchar2(10) NOT NULL,
    FOREIGN KEY (promotion_id) REFERENCES Promotions(promotion_id)
);
```

After realizing that my first table was a bit too slim I decided on adding a few more important things to it.

```
ALTER TABLE Coupons
ADD (discount NUMBER(5,2),
     start_date date,
     end_date date);
```

I also created a few more tables like “transactions” and modified the default “departments” table adding a few more things like, the budget, number of employees and contact info

```
CREATE TABLE Transactions (
    transaction_id number(16) PRIMARY KEY,
    employee_id number(16) NOT NULL,
    patient_id number(16) NOT NULL,
    prescription_id number(16) NOT NULL,
    date_of_transaction date NOT NULL,
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id),
    FOREIGN KEY (patient_id) REFERENCES Patient(patient_id),
    FOREIGN KEY (prescription_id) REFERENCES Prescription(prescription_id)
);
```

```
ALTER TABLE Departments
ADD (budget NUMBER(20,2),
     number_of_employees NUMBER(5),
     contact_info VARCHAR2(255));
```

5 BUDGET	NUMBER(20,2)	Yes	(null)	5 (null)
6 NUMBER_OF_EMPLOYEES	NUMBER(5,0)	Yes	(null)	6 (null)
7 CONTACT_INFO	VARCHAR2(255 BYTE)	Yes	(null)	7 (null)

After all of that I realized that one of my tables became obsolete, since it was mostly a duplicate of the “Transactions” table so I decided to drop it using the “Drop table” command, unfortunately, I forgot the name of the table, but I remember that it was something related to the orders, its only purpose was being a “link table” in order to avoid a many-to-many relationship between the “payments” and “orders” tables, something that the new “transactions” table does much better.

Dropping the table was easily accomplished by:

```
DROP TABLE name_of_the_table_that_i_forgot;
```

I also created other tables without much problems and added things that I may have forgotten, like some foreign keys:

```
CREATE TABLE Patient (
    patient_id number(16) PRIMARY KEY,
    first_name varchar2(255) NOT NULL,
    last_name varchar2(255) NOT NULL,
    address varchar2(255) NOT NULL,
    date_of_birth date NOT NULL,
    medical_history TEXT
);

CREATE TABLE Prescription (
    prescription_id number(16) PRIMARY KEY,
    patient_id number(16) NOT NULL,
    medication_id number(16) NOT NULL,
    dosage varchar2(255) NOT NULL,
    frequency varchar2(255) NOT NULL,
    FOREIGN KEY (patient_id) REFERENCES Patient(patient_id),
    FOREIGN KEY (medication_id) REFERENCES Medication(medication_id)
);

ALTER TABLE Jobs
ADD FOREIGN KEY (department_id) REFERENCES Departments(department_id);

ALTER TABLE Medication
ADD (manufacturer_id number(16),
    FOREIGN KEY (manufacturer_id) REFERENCES Manufacturers(manufacturer_id));
```

All of these commands worked flawlessly, but I unfortunately forgot to take screenshots when I first ran them, running them now on my database will most likely cause problems, but since all of the modifications are present in the database schema there’s no doubt that they work.

Oracle DB × PATIENT ×

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL Actions...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 PATIENT_ID	NUMBER(16,0)	No	(null)	1	(null)
2 FIRST_NAME	VARCHAR2(20 BYTE)	No	(null)	2	(null)
3 LAST_NAME	VARCHAR2(20 BYTE)	No	(null)	3	(null)
4 ADDRESS	VARCHAR2(255 BYTE)	No	(null)	4	(null)
5 DATE_OF_BIRTH	DATE	No	(null)	5	(null)
6 MEDICAL_HISTORY	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)

Oracle DB × PRESCRIPTION ×

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL Actions...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 PRESCRIPTION_ID	NUMBER(16,0)	No	(null)	1	(null)
2 PATIENT_ID	NUMBER(16,0)	No	(null)	2	(null)
3 PRODUCT_ID	NUMBER(16,0)	No	(null)	3	(null)
4 DOSAGE	VARCHAR2(255 BYTE)	No	(null)	4	(null)
5 FREQUENCY	VARCHAR2(255 BYTE)	No	(null)	5	(null)

Oracle DB × JOBS ×

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL Actions...

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1 EMPLOYEE_ID_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED N
2 SYS_C00760157	Check	"EMPLOYEE_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED N
3 SYS_C00760158	Check	"START_DATE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED N
4 SYS_C00760159	Check	"END_DATE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED N
5 SYS_C00760160	Check	"JOB_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED N
6 SYS_C00760168	Foreign_Key	(null)	CHITAI_65	DEPARTMENTS	DEPT_ID_PK	NO ACTION	ENABLED N

Oracle DB × MEDICATION ×

Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL Actions...

R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE	VALIDATED	GENERATED
1 (null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	USER NAME
2 (null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	USER NAME
3 CHITAI_65	MANUFACTURERS	MANUFACTURERS_PK	NO ACTION	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME
4 CHITAI_65	PROMOTIONS	SYS_C00759944	NO ACTION	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME
5 CHITAI_65	INVENTORY	SYS_C00675205	NO ACTION	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME
6 CHITAI_65	ORDERS	SYS_C00760382	NO ACTION	ENABLED	NOT DEFERRABLE	VALIDATED	GENERATED NAME
7 (null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALIDATED	USER NAME

Using DML Elements: Now that we created most of the tables, we also have to put it to use, first I'll create a table called Students, in which I'll insert my name using this command:

The screenshot shows two side-by-side panes in Oracle SQL Developer. The left pane contains the DDL and DML code for creating the Students table and inserting a single row:

```

CREATE TABLE Students (
    student_id number(1) PRIMARY KEY,
    student_name varchar2(20) NOT NULL,
    group_number number(4) NOT NULL
);

INSERT INTO Students (student_id, student_name, group_number)
VALUES (1, 'CHITA Ionut-Cristian', 1065);
  
```

The right pane shows the results of the SELECT query and the output of the inserted row:

```

SELECT * from students
  
```

Script Output X | Task completed in 0.037 seconds

Table STUDENTS created.

1 row inserted.

STUDENT_ID	STUDENT_NAME	GROUP_NUMBER
1	CHITA Ionut-Cristian	1065

It worked! Let's play around some more, next let's insert some data into the “Patients” table:

```

INSERT INTO patients (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS,
DATE_OF_BIRTH, MEDICAL_HISTORY)
VALUES (1, 'John', 'Doe', '123 Main Street', TO_DATE('1985-01-01', 'YYYY-MM-DD'), 'Allergic to penicillin');

INSERT INTO patients (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS,
DATE_OF_BIRTH, MEDICAL_HISTORY)
VALUES (2, 'Jane', 'Smith', '456 Park Avenue', TO_DATE('1990-05-20', 'YYYY-MM-DD'), 'Diabetes');
  
```

Note: I'm using the “to date” function because in Oracle, the default date format is DD-MON-YY. This lets me change it to whatever date format I wish.

The screenshot shows the DML statements for inserting data into the patient table:

```

INSERT INTO patient (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS, DATE_OF_BIRTH, MEDICAL_HISTORY)
VALUES (1, 'John', 'Doe', '123 Main Street', TO_DATE('1985-01-01', 'YYYY-MM-DD'), 'Allergic to penicillin');

INSERT INTO patient (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS, DATE_OF_BIRTH, MEDICAL_HISTORY)
VALUES (2, 'Jane', 'Smith', '456 Park Avenue', TO_DATE('1990-05-20', 'YYYY-MM-DD'), 'Diabetes');

select * from patient
  
```

Script Output X | Query Result X | All Rows Fetched: 2 in 0.011 seconds

PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS	DATE_OF_BIRTH	MEDICAL_HISTORY
1	John	Doe	123 Main Street	01-JAN-85	Allergic to penicillin
2	Jane	Smith	456 Park Avenue	20-MAY-90	Diabetes

Now, let's say that one of our patients moved out and discovered that they have other ailments too, let's change that too, we'll use the UPDATE statement:

```
UPDATE patients
SET address = '789 Oak Street', medical_history = 'Asthma and Diabetes'
WHERE patient_id = 2;
```

The screenshot shows a SQL query window with the following content:

```
UPDATE patients
SET address = '789 Oak Street', medical_history = 'Asthma and Diabetes'
WHERE patient_id = 2;

select * from patients
```

Below the query window, there is a "Script Output" tab and a "Query Result" tab. The "Query Result" tab shows the following table:

	PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS	DATE_OF_BIRTH	MEDICAL_HISTORY
1	1	John	Doe	123 Main Street	01-JAN-85	Allergic to penicillin
2	2	Jane	Smith	789 Oak Street	20-MAY-90	Asthma and Diabetes

Let's say that one of our patients chose another pharmacy (very unlikely), how would we delete their records ? For this we have the “DELETE” statement, let's try it:

```
DELETE FROM patients WHERE patient_id = 1;
```

This will delete the patient with the id = 1, in our case, Mr. John Doe.

The screenshot shows a SQL query window with the following content:

```
DELETE FROM patients WHERE patient_id = 1;

select * from patients
```

Below the query window, there is a "Script Output" tab and a "Query Result" tab. The "Query Result" tab shows the following table:

	PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS	DATE_OF_BIRTH	MEDICAL_HISTORY
1	2	Jane	Smith	789 Oak Street	20-MAY-90	Asthma and Diabetes

We have quite a lot of data on the employees by default, let's play around with that shall we ? We also have quite a few jobs by default, all jobs have a minimum salary and a maximum salary, each job has it's own id, let's give them all a raise using the “MERGE” statement

In order to do that we'll have to merge the salaries into the employees table, we can accomplish that by using:

```
MERGE INTO employees e
USING jobs j
ON (e.job_id = j.job_id)
WHEN MATCHED THEN
    UPDATE SET e.salary = j.max_salary
```

This will update the salary of all the rows in the employees table with the corresponding max_salary value from the jobs table. Before running it, let's first take a look at our employees table and look at their salaries:

These are some of the lucky guys that will get a raise, Lets run it and see what happens

Error report -

ORA-30926: unable to get a stable set of rows in the source tables

Looks like it didn't work, this happens because the **MERGE** statement returns multiple rows for a single row in the target table. To fix this error, we need to ensure that we have a **USING** clause that returns at most one row for each row in the target table

```
MERGE INTO employees e
USING (SELECT j.max_salary FROM jobs j WHERE j.job_id = e.job_id) s
ON (1=1)
WHEN MATCHED THEN
    UPDATE SET e.salary = s.max_salary
```

As you can see, this time around it worked:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
77	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	12000	0.2	149	80
78	177	Jack	Livingston	JLIVINGS	011.44.1644.429264	23-APR-98	SA_REP	12000	0.2	149	80
79	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	12000	0.15	149	(null)
80	179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04-JAN-00	SA_REP	12000	0.1	149	80
81	180	Winston	Taylor	WTAYLOR	650.507.9876	24-JAN-98	SH_CLERK	3200	(null)	120	50
82	181	Jean	Fleaur	JFLEAUR	650.507.9877	23-FEB-98	SH_CLERK	3100	(null)	120	50
83	182	Martha	Sullivan	MSULLIVA	650.507.9878	21-JUN-99	SH_CLERK	2500	(null)	120	50
84	183	Girard	Geoni	GGEONI	650.507.9879	03-FEB-00	SH_CLERK	2800	(null)	120	50
85	184	Nandita	Sarchand	NSARCHAN	650.509.1876	27-JAN-96	SH_CLERK	4200	(null)	121	50
86	185	Alexis	Bull	ABULL	650.509.2876	20-FEB-97	SH_CLERK	4100	(null)	121	50
87	186	Julia	Dellinger	JDELLING	650.509.3876	24-JUN-98	SH_CLERK	3400	(null)	121	50
88	187	Anthony	Cabrio	ACABRIO	650.509.4876	07-FEB-99	SH_CLERK	3000	(null)	121	50
89	188	Kelly	Chung	KCHUNG	650.505.1876	14-JUN-97	SH_CLERK	3800	(null)	122	50
90	189	Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-97	SH_CLERK	3600	(null)	122	50
91	190	Timothy	Gates	TGATES	650.505.3876	11-JUL-98	SH_CLERK	2900	(null)	122	50
92	191	Randall	Perkins	RPERKINS	650.505.4876	19-DEC-99	SH_CLERK	2500	(null)	122	50
93	192	Sarah	Bell	SBELL	650.501.1876	04-FEB-96	SH_CLERK	4000	(null)	123	50
94	193	Britney	Everett	BEVERETT	650.501.2876	03-MAR-97	SH_CLERK	3900	(null)	123	50
95	194	Samuel	McCain	SMCCAIN	650.501.3876	01-JUL-98	SH_CLERK	3200	(null)	123	50
96	195	Vance	Jones	VJONES	650.501.4876	17-MAR-99	SH_CLERK	2800	(null)	123	50
97	196	Alana	Walsh	AWALSH	650.507.9811	24-APR-98	SH_CLERK	3100	(null)	124	50
98	197	Kevin	Feehey	KFEENEY	650.507.9822	23-MAY-98	SH_CLERK	3000	(null)	124	50
99	198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-99	SH_CLERK	2600	(null)	124	50
100	199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-00	SH_CLERK	2600	(null)	124	50
101	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10
102	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20
103	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK REP	9000	(null)	201	20
104	203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-94	MK REP	6500	(null)	101	40
105	204	Hermann	Baer	HBAER	515.123.8888	07-JUN-94	PR REP	10000	(null)	101	70
106	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110
107	206	William	Gietz	WGIEZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110

Now, let's also get some more patients, in order to get a bunch of people fast I'll get the data from the "customers" table and insert it into the "patients" table using this command:

```
INSERT INTO Patients (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS,
DATE_OF_BIRTH, MEDICAL_HISTORY)
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME, ADDRESS, TO_DATE('1900-01-01',
'YYYY-MM-DD'), NULL
FROM Customers;
```

Note: the reason we use the “**to date**” function this time is because we first created the “**Patients**” table we set the date of birth as **not null**, which means that we have to provide a value, to circumvent that we can either use the “**to date**” function to set a default value that we can change later on. Alternatively, we can alter the table using:

```
ALTER TABLE Patients MODIFY DATE_OF_BIRTH DATE NULL;
```

This removes the constraint and lets us input null data

The screenshot shows the Oracle SQL Developer interface. The Worksheet tab is active, displaying the following SQL code:

```
INSERT INTO Patients (PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS, DATE_OF_BIRTH, MEDICAL_HISTORY)
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME, ADDRESS, TO_DATE('1900-01-01', 'YYYY-MM-DD'), NULL
FROM Customers;
```

Below the code, the Script Output window shows the result of the execution:

```
Script Output x
Task completed in 0.028 seconds
319 rows inserted.
```

As you can see, it was a success, now we have some more data to work with. Let's also insert some products into our inventory:

```
INSERT INTO Inventory (INVENTORY_ID, PRODUCT_ID, NAME, QUANTITY, COST, PRICE)
VALUES (2, 1002, 'Ibuprofen', 1000, 0.75, 1.50);
```

```
INSERT INTO Inventory (INVENTORY_ID, PRODUCT_ID, NAME, QUANTITY, COST, PRICE)
VALUES (3, 1003, 'Aspirin', 1000, 0.65, 1.30);
```

```
INSERT INTO Inventory (INVENTORY_ID, PRODUCT_ID, NAME, QUANTITY, COST, PRICE)
VALUES (4, 1004, 'Naproxen', 1000, 0.85, 1.70);
```

```
INSERT INTO Inventory (INVENTORY_ID, PRODUCT_ID, NAME, QUANTITY, COST, PRICE)
VALUES (5, 1005, 'Acetylsalicylic Acid', 1000, 0.70, 1.40);
```

```
INSERT INTO Inventory (INVENTORY_ID, PRODUCT_ID, NAME, QUANTITY, COST, PRICE)
VALUES (1, 1001, 'Acetaminophen', 1000, 0.50, 1.00);
```

The screenshot shows the Oracle SQL Developer interface with the Data tab selected. The table structure for Inventory is displayed:

INVENTORY_ID	PRODUCT_ID	NAME	QUANTITY	COST	PRICE
1	2	1002 Ibuprofen	1000	0.75	1.5
2	3	1003 Aspirin	1000	0.65	1.3
3	4	1004 Naproxen	1000	0.85	1.7
4	5	1005 Acetylsalicylic Acid	1000	0.7	1.4
5	1	1001 Acetaminophen	1000	0.5	1

Now that we have all of this data, let's play around with the SELECT statement:

1. Let's first Select all of the patients whose first name is "Amanda"

```
SELECT *
FROM Patients
WHERE First_Name = 'Amanda'
```

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is active, displaying the query: `SELECT * FROM Patients WHERE First_Name = 'Amanda'`. Below the query, the 'Script Output' tab shows the results: `no rows selected`. The results grid displays three rows of patient data:

PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS
850	Amanda	Finney	Pfannenstilstr 13
851	Amanda	Brown	Kreuzstr 32
852	Amanda	Tanner	1539 Stripes Rd

2. What about selecting customers based by their postal code, let's select customers with postal code between 4600 and 4700

```
SELECT *
FROM Customers
WHERE Postal_Code BETWEEN '4600' AND '4700'
```

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is active, displaying the query: `SELECT * FROM Customers WHERE Postal_Code BETWEEN '4600' AND '4700'`. Below the query, the 'Script Output' tab shows the results: `no rows selected`. The results grid displays 10 rows of customer data:

CUSTOMER_ID	FIRST_NAME	LAST_NAME	ADDRESS	POSTAL_CODE	PHONE_NUMBER	EMAIL
101	Constantin	Welles	514 W Superior St	46901	+1 317 123 4104	Constantin.Welles@ANHINGA.COM
102	Harrison	Pacino	2515 Floyd Ave	46218	+1 317 123 4111	Harrison.Pacino@ANI.COM
104	Harrison	Sutherland	6445 Bay Harbor Ln	46254	+1 317 123 4126	Harrison.Sutherland@GODWIT.COM
106	Matthias	Hannah	1608 Portage Ave	46616	+1 219 123 4136	Matthias.Hannah@REBEE.COM
107	Matthias	Cruise	23943 Us Highway 33	46517	+1 219 123 4138	Matthias.Cruise@ROSEBARK.COM
108	Meenakshi	Mason	136 E Market St # 800	46204	+1 317 123 4141	Meenakshi.Mason@JACANA.COM
109	Christian	Cage	1905 College St	46628	+1 219 123 4142	Christian.Cage@KINGLET.COM
110	Charlie	Sutherland	3512 Rockville Rd # 137C	46222	+1 317 123 4146	Charlie.Sutherland@LIMPICKIN.COM
249	Hannah	Broderick	6915 Grand Ave	46323	+1 219 123 4145	Hannah.Broderick@SHRIKE.COM
269	Kyle	Martin	2713 N Bendix Dr	46628	+1 219 123 4116	Kyle.Martin@EIDER.COM

10 rows selected.

3. Let's also count how many of our customers own a phone / phone number

```
SELECT COUNT(*) AS "Total Customers"
FROM Customers
WHERE Phone_Number IS NOT NULL
```

```

Worksheet | Query Builder
SELECT COUNT(*) AS "Total Customers"
FROM Customers
WHERE Phone_Number IS NOT NULL

Worksheet | Query Builder
SELECT COUNT(*) AS "Total Customers"
FROM Customers
WHERE Phone_Number IS NULL

```

Script Output | Task completed in 0.018 seconds

Total Customers

319

Script Output | Task completed in 0.023 seconds

Total Customers

0

Looks like all of them own a phone, we can see the opposite by slightly altering the condition, replacing **IS NOT NULL** with **IS NULL**.

4. What about using the **TO_CHAR** function, it is used in order to convert a number or a date into a string, let's extract our patients' date of birth and convert it into a string

```

SELECT First_Name, Last_Name, TO_CHAR(DateOfBirth, 'Month DD, YYYY') AS "DOB"
FROM Patients

```

```

Worksheet | Query Builder
SELECT First_Name, Last_Name, TO_CHAR(DateOfBirth, 'Month DD, YYYY') AS "DOB"
FROM Patients

```

Script Output | Task completed in 0.052 seconds

FIRST_NAME	LAST_NAME	DOB
Max	von Sydow	January 01, 1900
Max	Schell	January 01, 1900
Cynda	Whitcraft	January 01, 1900
Donald	Minnelli	January 01, 1900
Hannah	Broderick	January 01, 1900
Dan	Williams	January 01, 1900
Raul	Wilder	January 01, 1900
Shah Rukh	Field	January 01, 1900
Sally	Bogart	January 01, 1900
Bruce	Bates	January 01, 1900
Brooke	Shepherd	January 01, 1900

5. But what if we only care about the year ? We can use the **EXTRACT** function:

```

SELECT First_Name, EXTRACT(YEAR FROM DateOfBirth) AS "Birth Year"
FROM Patients

```

Worksheet Query Builder

```
SELECT First_Name, EXTRACT(YEAR FROM Date_of_Birth) AS "Birth Year"
FROM Patients
```

Script Output x
 Task completed in 0.057 seconds

FIRST_NAME	Birth Year
Max	1900
Max	1900
Cynda	1900
Donald	1900

6. Let's do something more daring, shall we ? Let's see whose medical history we have on file and whose we don't. We can do that by using this SELECT statement:

```
SELECT p.First_Name, p.Last_Name,
CASE
    WHEN NVL(p.Medical_History, 'N/A') = 'N/A' THEN 'No medical history on
file'
    ELSE 'Medical history on file'
END AS "Medical History"
FROM Patients p
```

Worksheet Query Builder

```
SELECT p.First_Name, p.Last_Name,
CASE
    WHEN NVL(p.Medical_History, 'N/A') = 'N/A' THEN 'No medical history on file'
    ELSE 'Medical history on file'
END AS "Medical History"
FROM Patients p
```

Script Output x
 Task completed in 0.052 seconds

FIRST_NAME	LAST_NAME	Medical History
Bryan	Belushi	No medical history on file
Burt	Spielberg	No medical history on file
Burt	Neeson	No medical history on file
Buster	Jackson	No medical history on file
Buster	Edwards	No medical history on file
Buster	Bogart	No medical history on file
C. Thomas	Nolte	No medical history on file
Daniel	Loren	No medical history on file
Daniel	Gueney	No medical history on file
Jane	Smith	Medical history on file
Ali	Boyer	No medical history on file

7. Let's try some more functions:

```
SELECT *
FROM Patients
WHERE First_Name IN ('John', 'Jane', 'Bob')
```

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is selected. In the main pane, the following SQL code is written:

```
SELECT *
FROM Patients
WHERE First_Name IN ('John', 'Jane', 'Bob')
```

In the 'Script Output' pane below, the results are displayed as a table:

PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS
274	Bob	McCarthy	701 Seneca St
913	Bob	Sharif	1600 Target Crt
2	Jane	Smith	789 Oak Street

The total execution time is listed as 0.02 seconds.

8. We can do the opposite of `IN` by using `NOT IN`

```
SELECT *
FROM Customers
WHERE Customer_ID NOT IN (SELECT Customer_ID FROM Orders)
```

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is selected. In the main pane, the following SQL code is written:

```
SELECT *
FROM Customers
WHERE Customer_ID NOT IN (SELECT Customer_ID FROM Orders)
```

In the 'Script Output' pane below, the results are displayed as a table:

CUSTOMER_ID	FIRST_NAME	LAST_NAME	ADDRESS	POSTAL_CODE	PHONE_NUMBER	EMAIL
841	Ali	Boyer	Bendlein	9062	+41 71 012 3559	Ali.Boyer@WILLET.COM
101	Constantin	Welles	514 W Superior St	46901	+1 317 123 4104	Constantin.Welles@ANHINGA.COM
102	Harrison	Pacino	2515 Bloyd Ave	46218	+1 317 123 4111	Harrison.Pacino@ANI.COM
103	Manisha	Taylor	8768 N State Rd 37	47404	+1 812 123 4115	Manisha.Taylor@AUKLET.COM
104	Harrison	Sutherland	6445 Bay Harbor Ln	46254	+1 317 123 4126	Harrison.Sutherland@GODWIT.COM
105	Matthias	MacGraw	4019 W 3Rd St	47404	+1 812 123 4129	Matthias.MacGraw@GOLDENEYE.COM
106	Matthias	Hannah	1608 Portage Ave	46616	+1 219 123 4136	Matthias.Hannah@GREBE.COM
107	Matthias	Cruise	23943 Us Highway 33	46517	+1 219 123 4138	Matthias.Cruise@GROSBEEK.COM
108	Meenakshi	Mason	136 E Market St # 800	46204	+1 317 123 4141	Meenakshi.Mason@JACANA.COM
109	Christian	Cage	1905 College St	46628	+1 219 123 4142	Christian.Cage@KINGLET.COM
110	Charlie	Sutherland	3512 Rockville Rd # 137C	46222	+1 317 123 4146	Charlie.Sutherland@LIMPKIN.COM

The total execution time is listed as 0.102 seconds.

This selects everything from customers whose id doesn't match the ones from the Orders table

9. A bit earlier we added all of our customers to the patients table, but before that, we added jane, let's see if we can find her using the `MINUS` function

```
SELECT PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS
FROM Patients
MINUS
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME, ADDRESS
FROM Customers
```

Worksheet Query Builder

```

SELECT PATIENT_ID, FIRST_NAME, LAST_NAME, ADDRESS
FROM Patients
MINUS
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME, ADDRESS
FROM Customers

```

Script Output x | Task completed in 0.023 seconds

PATIENT_ID	FIRST_NAME	LAST_NAME	ADDRESS
2	Jane	Smith	789 Oak Street

And here she is! Note: We didn't select the whole tables, that's because they have a different number of columns which will cause error ORA-01789 to occur, to fix this we can simply select only the columns that we need.

10. Some simple and correlated subqueries: Simple:

```

SELECT First_Name, Last_Name
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);

```

Worksheet Query Builder

```

SELECT First_Name, Last_Name
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);

```

Script Output x | Query Result x | Task completed in 1.504 seconds

FIRST_NAME	LAST_NAME
Steven	King
Neena	Kochhar
Lex	De Haan
Alexander	Hunold
Bruce	Ernst
David	Austin

11. Correlated

```

SELECT First_Name, Last_Name
FROM Employees e1
WHERE EXISTS (SELECT 1 FROM Employees e2 WHERE e2.Manager_ID =
e1.Employee_ID);

```

Worksheet Query Builder

```
SELECT First_Name, Last_Name
FROM Employees e1
WHERE EXISTS (SELECT 1 FROM Employees e2 WHERE e2.Manager_ID = e1.Employee_ID);
```

Script Output Query Result | Task completed in 0.018 seconds

FIRST_NAME	LAST_NAME
Steven	King
Lex	De Haan
Alexander	Hunold
Neena	Kochhar
Nancy	Greenberg
Den	Raphaely
Matthew	Weiss
Adam	Fripp
Davram	Wangler

12. Let's also see some inner and outer joints. I'll use an inner join to show what products I have in my inventory and how many:

```
SELECT m.Product_Name, i.Quantity
FROM Medication m
INNER JOIN Inventory i ON m.Product_ID = i.inventory_ID;
```

Worksheet Query Builder

```
SELECT m.Product_Name, i.Quantity
FROM Medication m
INNER JOIN Inventory i ON m.Product_ID = i.inventory_ID;
```

Script Output | Task completed in 0.025 seconds

PRODUCT_NAME	QUANTITY
Ibuprofen	1000
Aspirin	1000
Naproxen	1000
Acetylsalicylic Acid	1000
Acetaminophen	1000

13. Let's see some outer joints too:

The screenshot shows a SQL worksheet interface with a toolbar at the top containing various icons. Below the toolbar, there are two tabs: "Worksheet" and "Query Builder". The "Worksheet" tab is selected. In the main area, a SQL query is written in the code editor:

```
SELECT m.Product_Name, i.Quantity
FROM Medication m
left JOIN Inventory i ON m.Product_ID = i.inventory_ID;
```

Below the code editor is a "Script Output" window. It displays a message: "Task completed in 0.039 seconds". The output results are shown in a table:

PRODUCT_NAME	QUANTITY
LCD Monitor 11/PM	
LCD Monitor 9/PM	
Monitor 17/HR	
Monitor 17/HR/F	
Monitor 17/SD	
Monitor 19/SD	
Monitor 19/SD/M	

14. Using aggregate function `HAVING` and `GROUP BY`

```
SELECT Product_Name, SUM(Quantity) AS "Total Quantity" FROM Inventory GROUP
BY Product_Name HAVING SUM(Quantity) > 500
```

The screenshot shows a SQL worksheet interface with a toolbar at the top containing various icons. Below the toolbar, there are two tabs: "Worksheet" and "Query Builder". The "Worksheet" tab is selected. In the main area, a SQL query is written in the code editor:

```
SELECT product_name, SUM(Quantity) AS "Total Quantity" FROM Inventory GROUP BY Product_name HAVING SUM(Quantity) > 500
```

Below the code editor is a "Script Output" window. It displays a message: "Task completed in 0.02 seconds". The output results are shown in a table:

PRODUCT_NAME	Total Quantity
Acetylsalicylic Acid	1000
Naproxen	1000
Ibuprofen	1000
Aspirin	1000
Acetaminophen	1000

15. `SUBSTR` function

```
SELECT SUBSTR>Last_Name, 1, 3) AS "Last Name Initials", MAX(Salary) FROM
Employees GROUP BY Last_Name
```

Worksheet Query Builder

```
SELECT SUBSTR>Last Name, 1, 3) AS "Last Name Initials", MAX(Salary) FROM Employees GROUP BY Last_Name|
```

Script Output x | Task completed in 0.039 seconds

Last Name Initials	MAX(Salary)
Gre	12000
Mar	5000
Mar	5000
Gee	5000
Phi	5000
Sti	5000
Raj	5000

16. Let's see how many employees have a salary over 5000 and who were hired before 2010:

Worksheet Query Builder

```
SELECT COUNT(*) AS "Number of Employees" FROM Employees WHERE Salary > 5000 AND Hire_Date <= TO_DATE('01-JAN-2010', 'DD-MON-YYYY')|
```

Script Output x | Task completed in 0.018 seconds

Number of Employees
61

17. What about some taxes? This SELECT statement will compute the tax amount based on the price of the product

Worksheet Query Builder

```
SELECT Product_Name, Price, (CASE WHEN Taxing = 'y' THEN Price * 0.15 ELSE 0 END) AS "Tax Amount" FROM Medication|
```

Script Output x | Task completed in 0.049 seconds

PRODUCT_NAME	PRICE	TAX AMOUNT
LCD Monitor 11/PM	259	38.85
LCD Monitor 9/PM	249	37.35
Monitor 17/HR	299	44.85
Monitor 17/HR/F	350	52.5
Monitor 17/SD	369	55.35
Monitor 19/SD	499	74.85
Monitor 19/SD/M	512	76.8
Monitor 21/D	999	149.85
Monitor 21/HR	879	131.85

18. Previously we used both a **join** and a **minus** in order to single out some products, let's try using **intersect** and see how it goes:

Worksheet Query Builder

```
SELECT Product_Name FROM Medication INTERSECT SELECT Product_Name FROM Inventory
```

Script Output Query Result | All Rows Fetched: 5 in 0.009 seconds

PRODUCT_NAME
1 Acetaminophen
2 Acetylsalicylic Acid
3 Aspirin
4 Ibuprofen
5 Naproxen

19. Let's also use `SYSDATE` and an `INNER JOIN` to show orders that happened in the past

Worksheet Query Builder

```
SELECT c.*, o.Order_Date
FROM Customers c
INNER JOIN Orders o ON c.Customer_ID = o.Customer_ID
WHERE o.Order_Date <= SYSDATE - INTERVAL '7' DAY
```

Script Output Query Result | Task completed in 0.022 seconds

CUSTOMER_ID	FIRST_NAME	LAST_NAME	ADDRESS	POSTAL_CODE	PHONE_NUMBER	EMAIL
101	Constantin	Welles	514 W Superior St	46901	+1 317 123 4104	Constan
102	Harrison	Pacino	2515 Bloyd Ave	46218	+1 317 123 4111	Harriso
103	Manisha	Taylor	8768 N State Rd 37	47404	+1 812 123 4115	Manisha
104	Harrison	Sutherland	6445 Bay Harbor Ln	46254	+1 317 123 4126	Harriso
105	Matthias	MacGraw	4019 W 3rd St	47404	+1 812 123 4129	Matthia
106	Matthias	Hannah	1608 Portage Ave	46616	+1 219 123 4136	Matthia
107	Matthias	Cruise	23943 Us Highway 33	46517	+1 219 123 4138	Matthia
108	Meenakshi	Mason	136 E Market St # 800	46204	+1 317 123 4141	Meenaks
109	Christian	Cage	1905 College St	46628	+1 219 123 4142	Christi
110	Chandira	Churchland	2619 Dahlberg Dr # 4 1270	46229	+1 317 123 4142	Chandira

20. Another useful thing to use is `DECODE`, as the name implies it is used to decode my binary y/n system into something more human readable

Worksheet Query Builder

```
SELECT Product_id, Product_Name, DECODE(Taxing, 'y', 'Tax Included', 'n', 'Tax Not Included', 'Other') AS "Tax" FROM Medication
```

Script Output Query Result | Task completed in 0.049 seconds

Product_id	Product_Name	Tax
1770	Cache / RAM	Tax Included
2412	8MB EDO Memory	Tax Included
2378	DIMM - 128 MB	Tax Included
3087	DIMM - 16 MB	Tax Included
2384	DIMM - 1GB	Tax Not Included

PRODUCT_ID	PRODUCT_NAME	Tax
1749	DIMM - 256MB	Tax Not Included
1750	DIMM - 2GB	Tax Not Included
2394	DIMM - 32MB	Tax Not Included
2400	DIMM - 512 MB	Tax Not Included
1763	DIMM - 64MB	Tax Not Included

21. Hierarchical queries:

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'Worksheet' and contains the SQL query: 'SELECT Product_Name, Level FROM Medication CONNECT BY PRIOR Product_ID = Manufacturer_ID;'. Below this is the 'Query Result' tab, which displays the output of the query. The output is a table with two columns: 'PRODUCT_NAME' and 'LEVEL'. The data shows ten rows of monitor models, all assigned to level 1.

PRODUCT_NAME	LEVEL
LCD Monitor 11/PM	1
LCD Monitor 9/PM	1
Monitor 17/HR	1
Monitor 17/HR/F	1
Monitor 17/SD	1
Monitor 19/SD	1
Monitor 19/SD/M	1
Monitor 21/D	1
Monitor 21/HB	1

This query retrieves the **Product_Name** and **Level** values from the **Medication** table, and the **PRIOR** operator is used to specify that the **Manufacturer_ID** column in the **Medication** table should be used to connect rows in the hierarchy.

The **LEVEL** pseudocolumn is used to display the level of each row in the hierarchy.

The query returns a result set with a row for each product in the **Medication** table, and the rows are organized into a hierarchy based on the **Manufacturer_ID** values. The **Level** column shows the level of each product in the hierarchy.

22. Some examples of constructing and manipulating other objects of the database: views, indexes, synonyms and sequences:

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'Worksheet' and contains the SQL script to create a view: 'CREATE VIEW top_coupon_discounts AS SELECT Coupon_ID, Promotion_ID, MAX(Discount) as "Highest Discount" FROM Coupons GROUP BY Coupon_ID, Promotion_ID;'. Below this is the 'Query Result' tab, which displays the message 'View TOP_COUPON_DISCOUNTS created.' indicating the successful creation of the view.

View TOP_COUPON_DISCOUNTS created.

Worksheet Query Builder

```
CREATE SYNONYM customer_info FOR Customers;

CREATE SEQUENCE order_sequence
START WITH 1000
INCREMENT BY 1;
```

Script Output Query Result Task completed in 0.068 seconds

View TOP_COUPON_DISCOUNTS created.

Synonym CUSTOMER_INFO created.

Sequence ORDER_SEQUENCE created.

Worksheet Query Builder

```
CREATE OR REPLACE SYNONYM medication_synonym
FOR Medication;
```

Script Output Query Result Task completed in 0.034 seconds

Synonym MEDICATION_SYNONYM created.

Worksheet Query Builder

```
CREATE INDEX Medication_Index ON Medication(Product_Name);
```

Script Output Query Result Task completed in 0.093 seconds

Synonym MEDICATION_SYNONYM created.

Index MEDICATION_INDEX created.

Worksheet Query Builder

```
ALTER INDEX medication_index RENAME TO medication_index_new;
DROP INDEX medication_index_new;
```

Script Output Query Result Task completed in 0.179 seconds

Synonym MEDICATION_SYNONYM created.

Index MEDICATION_INDEX created.

Index MEDICATION_INDEX altered.

Index MEDICATION_INDEX_NEW dropped.

23. Create/Insert/Update/Delete + Select

Worksheet Query Builder

```
CREATE TABLE ExampleTable (
    id INTEGER PRIMARY KEY,
    name VARCHAR(255)
);

INSERT INTO ExampleTable (id, name) VALUES (1, 'John');
INSERT INTO ExampleTable (id, name) VALUES (2, 'Jane');

UPDATE ExampleTable
SET name = 'Jack'
WHERE id = 1;

DELETE FROM ExampleTable
WHERE id = 2;

SELECT * FROM ExampleTable;
```

Script Output Query Result Task completed in 0.173 seconds

Table EXAMPLETABLE created.

1 row inserted.

1 row inserted.

1 row updated.

1 row deleted.

ID	NAME
1	Jack

24. Let's also use `SYS_CONNECT_BY_PATH` :

The screenshot shows the Oracle SQL Developer interface. In the top worksheet, a query is written:

```
SELECT Product_Name, SYS_CONNECT_BY_PATH(Product_Name, '|') AS "Hierarchy" FROM Medication CONNECT BY PRIOR Product_ID = Manufacturer_ID;
```

In the bottom Query Result tab, the output is displayed in two columns:

PRODUCT_NAME	Hierarchy
Ibuprofen	Ibuprofen
Aspirin	Aspirin
Naproxen	Naproxen

Below this, another section is shown:

PRODUCT_NAME	Hierarchy

This `SELECT` statement retrieves the `Product_Name` column values from the `Medication` table and creates a new column called `Hierarchy`. The `SYS_CONNECT_BY_PATH` function is used to generate a string of `Product_Name` values, separated by the `|` character, for each row in the `Medication` table. The `CONNECT BY PRIOR` clause is used to indicate that the `Product_Name` values should be connected in a hierarchy, with the `Product_ID` column serving as the parent key and the `Manufacturer_ID` column serving as the child key. The resulting hierarchy will show the relationships between the different products in the `Medication` table, with the `Product_Name` values of the parent products appearing before the `Product_Name` values of the child products.

In conclusion, our pharmacy database has proven to be a valuable resource for managing and organizing all aspects of our business. With tables for customers, employees, medication, orders, and more, we have been able to efficiently track and analyze data in order to make informed decisions and improve operations. The use of foreign keys and relationships has ensured the integrity of our data, and the implementation of views, indexes, synonyms, and sequences will allow us to easily access and manipulate the information in our database. Overall, the development and utilization of this database has greatly benefited our pharmacy and we look forward to continuing to use it in the future.