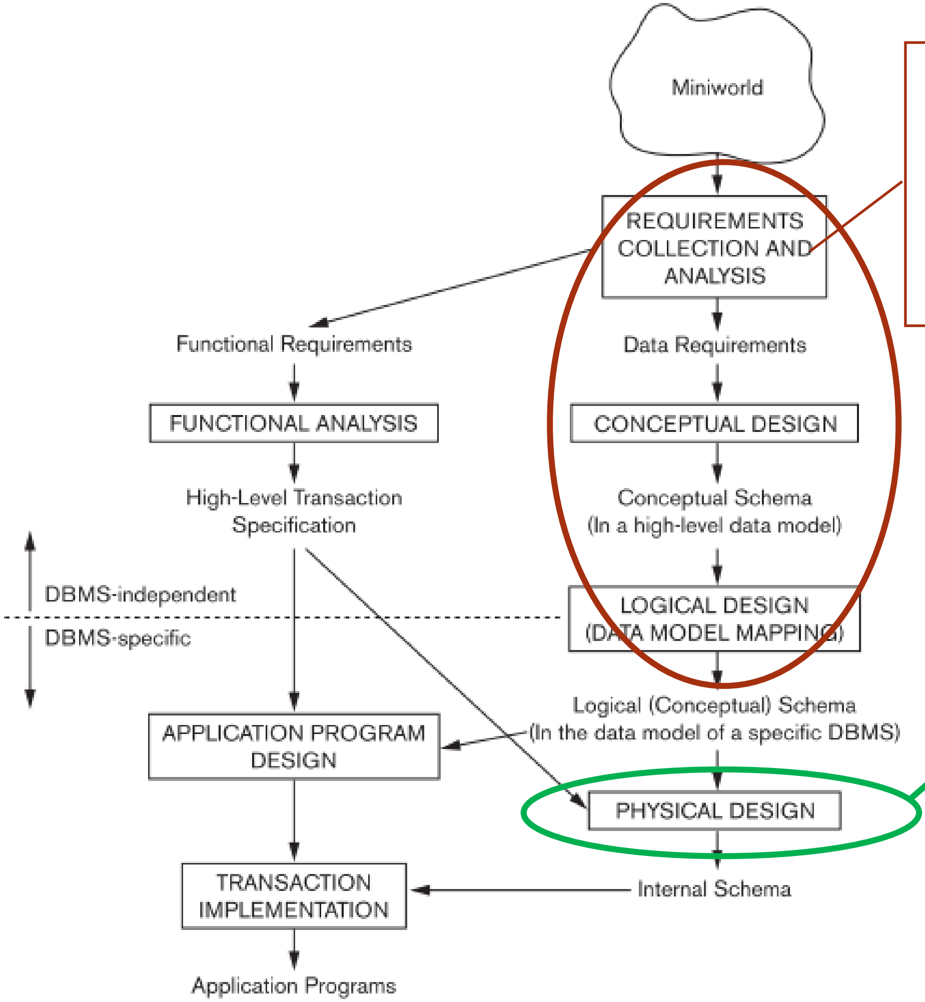
SQL Workshop

Structured Query Language

sjapit@uow.edu.au

22 March 2024


Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

For this workshop, we do a quick revision on these topics.

More exercises on this topic.

What is SQL?

- SQL is a **non-procedural** language, consisting of standard English words such as SELECT, INSERT, DELETE, that can be used by professionals and non-professionals alike to **define** and **manipulate** relational databases.

Non-procedural means you specify **what** information you require, rather than **how** to get it.

What is SQL?

SQL statements are categorized into:

Data definition statements – These statements are used to create database objects, such as tables, views, domains, indexes, roles, etc.

Data manipulation statements – These statements are used to manipulate and query database objects.

Access control statements – These statements are used to grant and revoke access and privileges.

System Administration statements – These statements are used to perform action on any object in the database.

Data Definition Language (DDL)

CREATE	ALTER	DROP
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE VIEW		DROP VIEW
CREATE INDEX		DROP INDEX

GRANT	REVOKE
GRANT privileges	REVOKE privileges

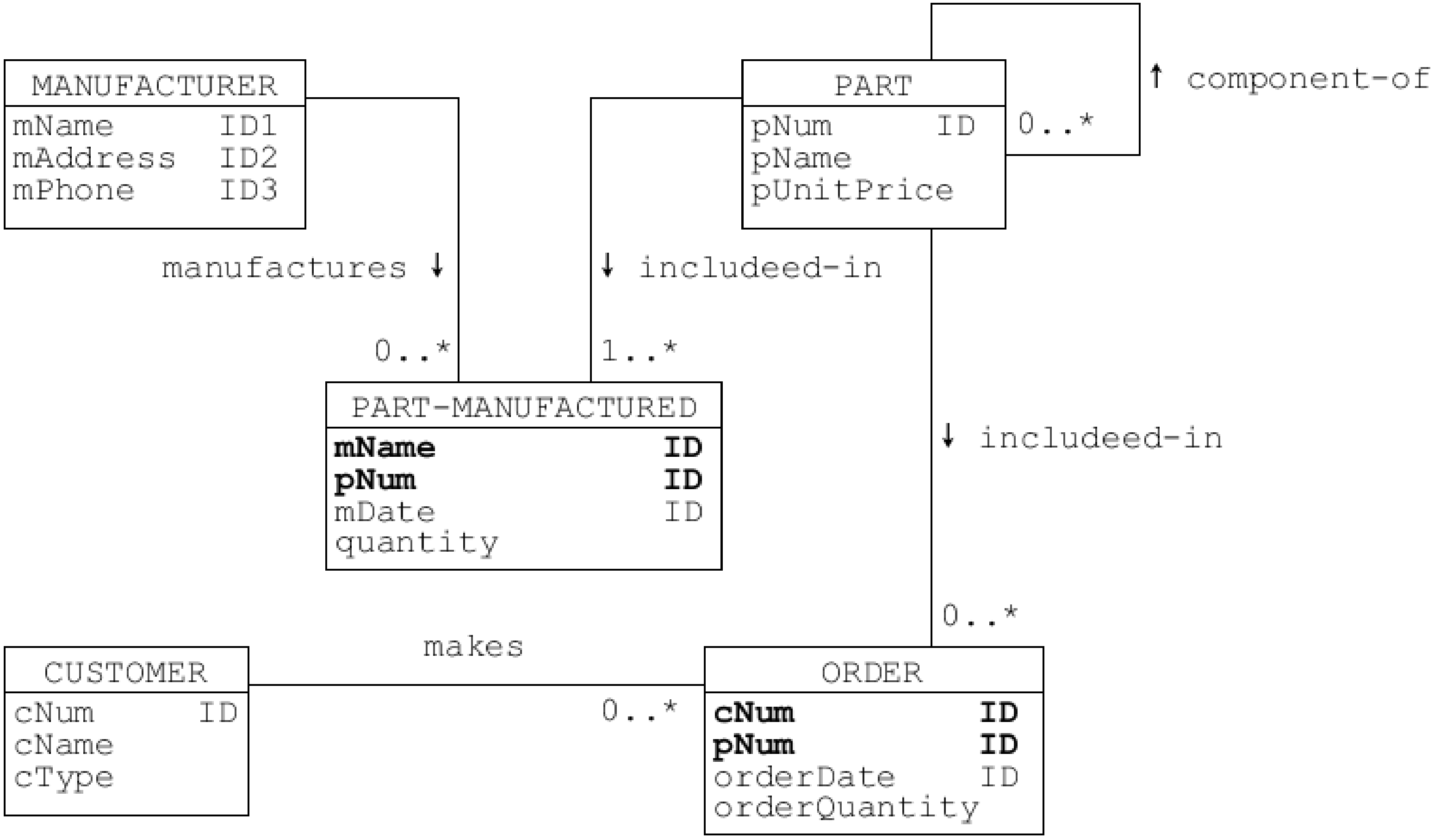
Data Manipulation Language (DML)

Command	Description
INSERT	INSERT statement inserts a new row into a relational table and automatically verifies the consistency constraints.
DELETE	DELETE statement deletes all rows that satisfy a given condition and automatically verifies the consistency constraints.
UPDATE	UPDATE statement modifies all rows that satisfy a given condition and automatically verifies the consistency constraints.
SELECT	SELECT statement retrieves data from a relational database.

Transforming from Conceptual Model to Relational Model

Part-manufactured as example

Sample Conceptual Schema



MANUFACTURER
mName ID1
mAddress ID2
mPhone ID3

manufactures ↓

0..*

PART
pNum ID
pName
pUnitPrice

0..*

↑ component-of

↓ includeed-in

1..*

PART-MANUFACTURED
mName **ID**
pNum **ID**
mDate ID
quantity

↓ includeed-in

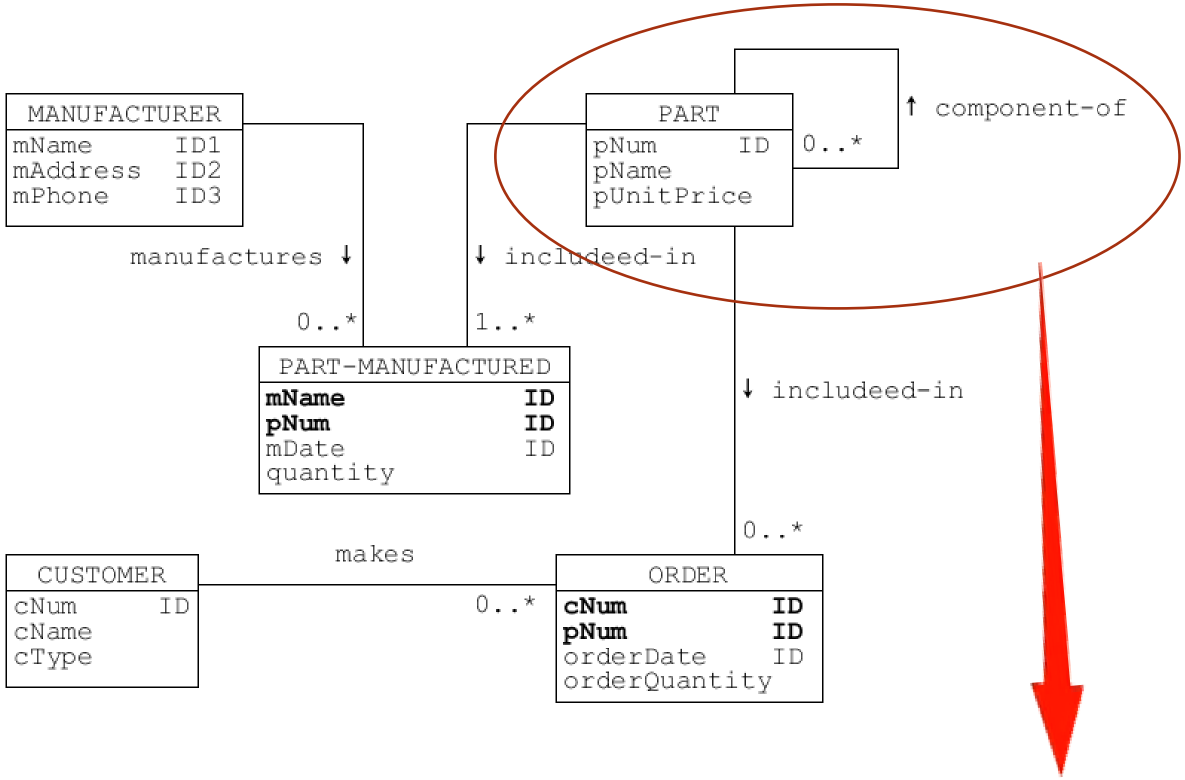
0..*

CUSTOMER
cNum ID
cName
cType

makes

0..*

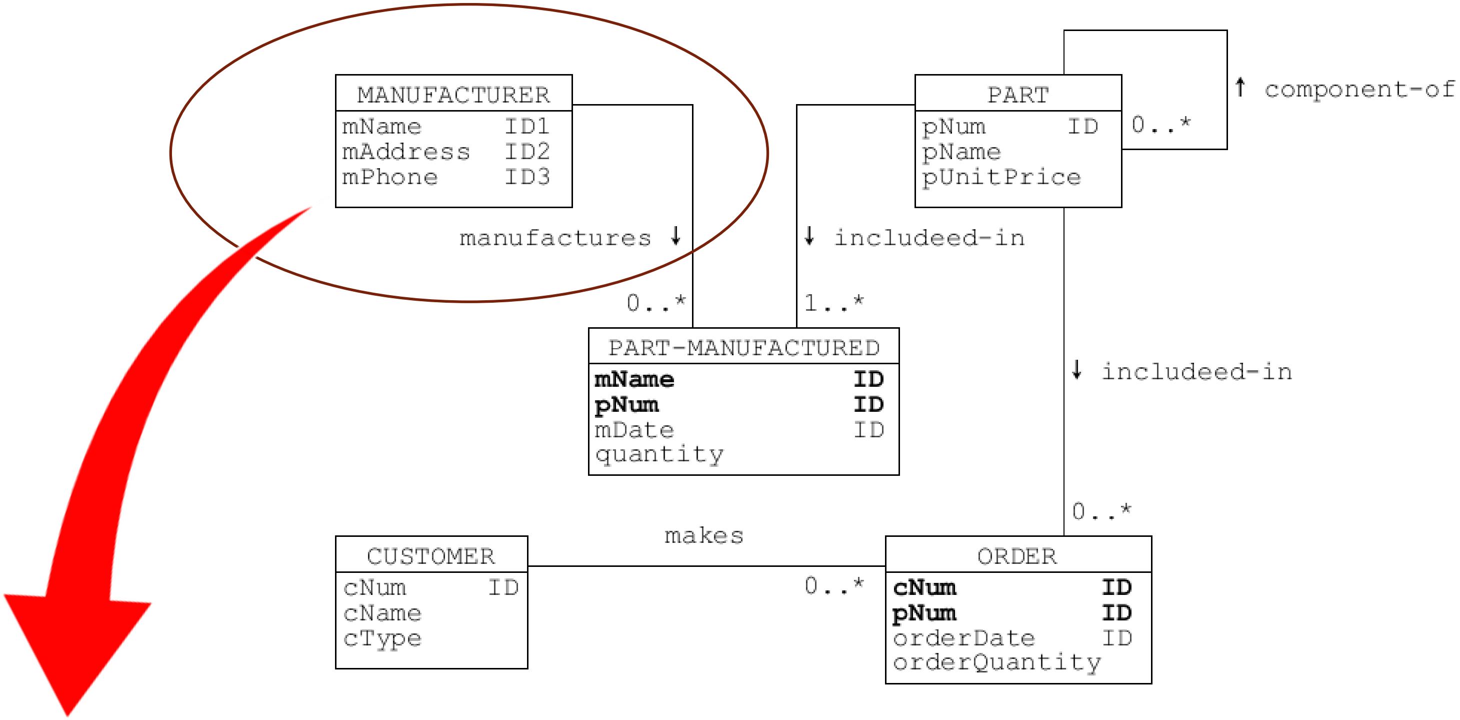
ORDER
cNum **ID**
pNum **ID**
orderDate ID
orderQuantity



PART(PNum, PName, PUnitPrice, ComponentOf)

primary key (PNum)

foreign key (ComponentOf) references PART(PNum)

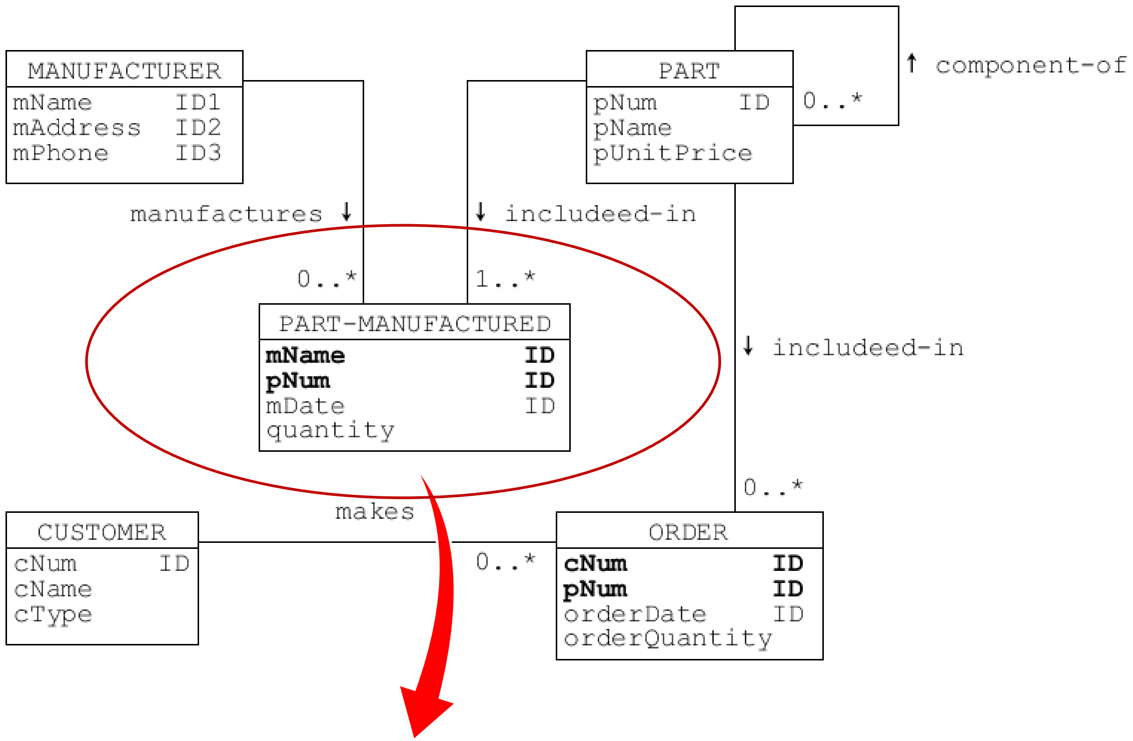


MANUFACTURER(MName, MAddress, MPhone)

primary key (MName)

candidate key (MPhone)

candidate key (MAddress)

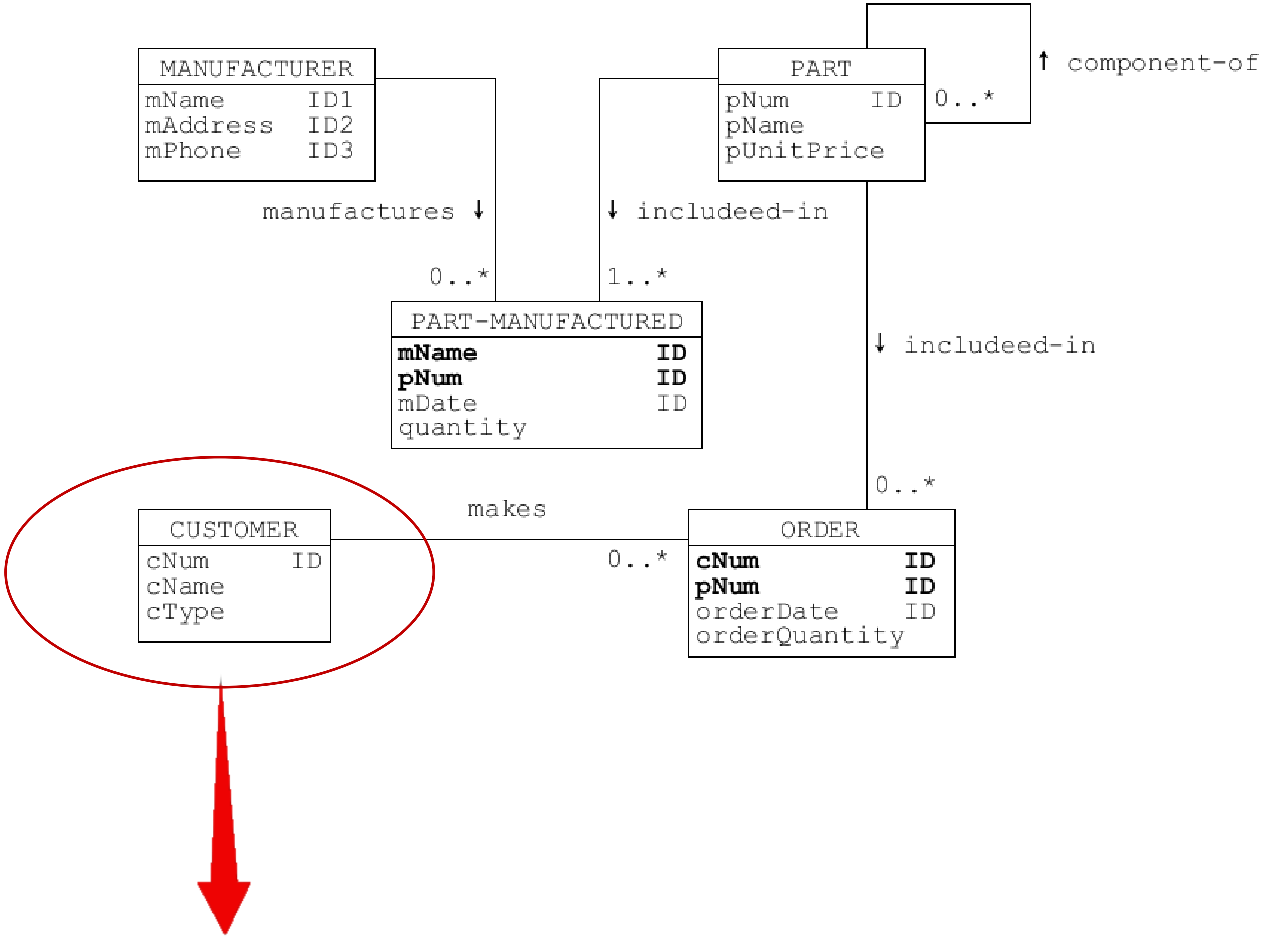


PART-MANUFACTURED(MDate, PNum, MName, Quantity)

primary key (MName, PNum, MDate)

foreign key (PNum) references PART(PNum)

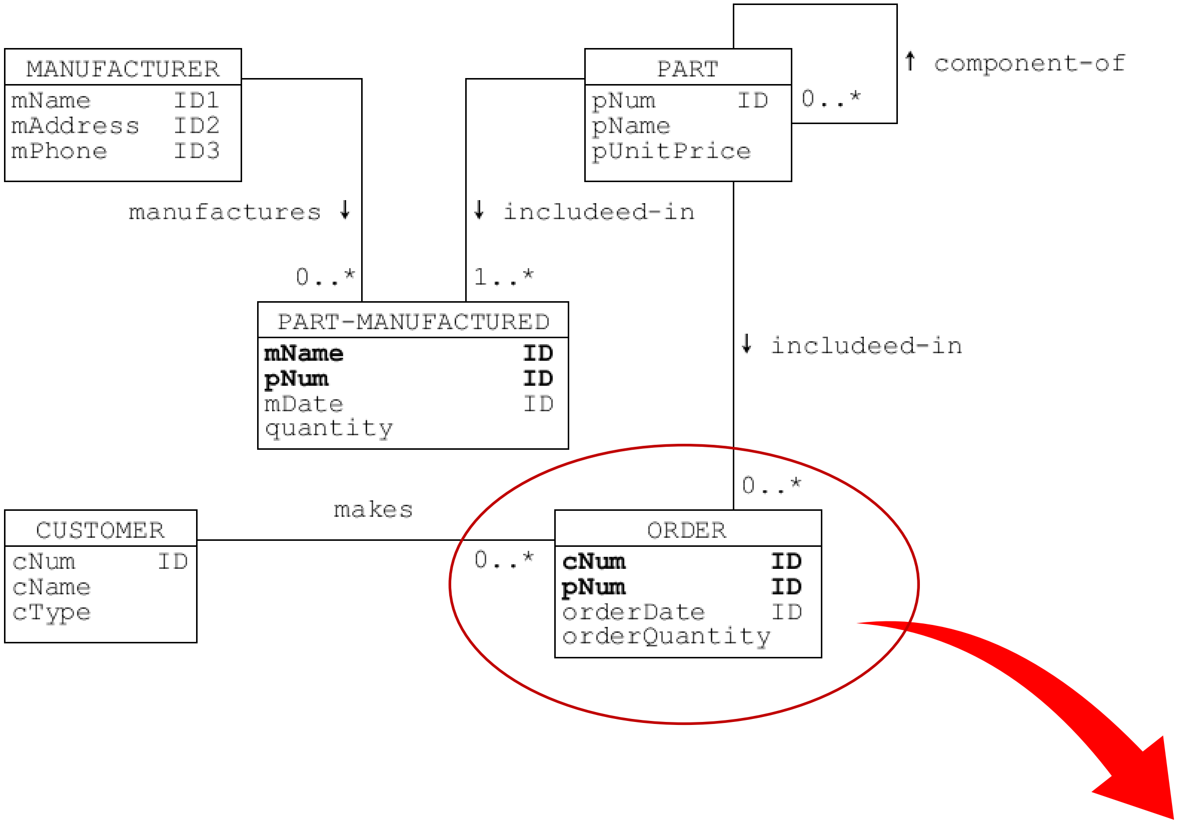
foreign key (MName) references MANUFACTURER(MName)



CUSTOMER(CNum, CName, CType)

primary key (CNum)

domain constraint ctype in ('INDIVIDUAL', 'INSTITUTION')



ORDERS(CNum, PNum, OrderDate, OrderQuantity)
primary key (CNum, PNum, OrderDate)
foreign key (CNum) references CUSTOMER(CNum)
foreign key (PNum) references PART(PNum

Datatype of attributes

We assume the following data type and NULL/NOT NULL constraints are used for each of the attributes:

PART

PNum	VARCHAR(25), not null
PName	VARCHAR(75), not null
PUnitPrice	Number(7,2), not null
ComponentOf	VARCHAR(25)

Datatype of attributes

We assume the following data type and NULL/NOT NULL constraints are used for each of the attributes:

MANUFACTURER

MName	VARCHAR(50), not null
MAddress	VARCAHR(100), not null
MPhone	VARCHAR(25), not null

Datatype of attributes

We assume the following data type and NULL/NOT NULL constraints are used for each of the attributes:

PART-MANUFACTURED

MDate	Date, not null
PNum	VARCHAR(25), not null
MName	VARCHAR(50), not null
Quantity	Number(10), not null

Datatype of attributes

We assume the following data type and NULL/NOT NULL constraints are used for each of the attributes:

CUSTOMER

CNum	VARCHAR(25), not null
CName	VARCHAR(75), not null
CType	VARCHAR(20), not null

Datatype of attributes

We assume the following data type and NULL/NOT NULL constraints are used for each of the attributes:

ORDERS

CNum	VARCHAR(25), not null
PNum	VARCHAR(25), not null
OrderDate	Date, not null
OrderQuantity	NUMBER(7,2), not null

CREATE TABLE statement

CREATE TABLE Statement

```
CREATE TABLE table_name (
  { column_name   data_type            [NOT NULL] [UNIQUE]
          [DEFAULT default_option] [CHECK (search_condition)] [, …] }

    [PRIMARY KEY (list_of_columns),]

  { [UNIQUE (list_of_columns),] [, …] }

  { [FOREIGN KEY (list_of_foreign_key_columns)
          REFERENCES parent_table_name [
          (list_of_candidate_key_columns)],
                    [ON DELETE referential_action] ]
                    [, …] }

  { [CHECK (search_condition)] [, …] }
);
```

- Bolded terms are SQL keywords
- Square brackets indicate an optional element.
- Curly braces indicates a required element.
- An ellipsis (…) is used to indicate optional repetition of an item 0 or more times.
- A vertical bar ‘|’ indicates a choice among alternative

CREATE TABLE Statement

- For example, create the table PART with the following specification:

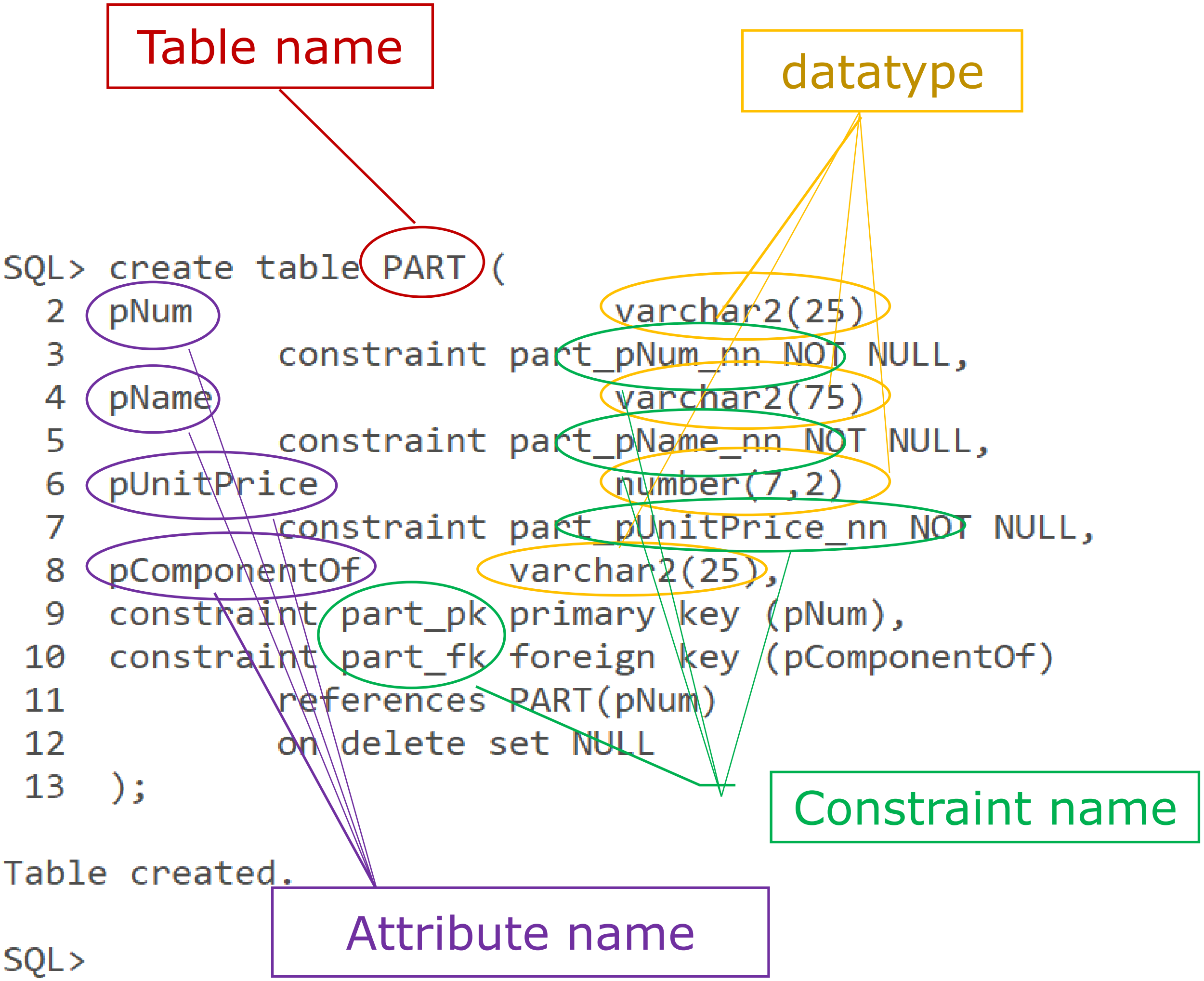
PART(PNum, PName, PUnitPrice, ComponentOf)
primary key (PNum)
foreign key (ComponentOf) references PART(PNum)

PART

PNum	VARCHAR(25), not null
PName	VARCHAR(75), not null
PUnitPrice	Number(7,2), not null
ComponentOf	VARCHAR(25)

PART(PNum, PName, PUnitPrice, ComponentOf)
primary key (PNum)
foreign key (ComponentOf) references PART(PNum)

PART	
PNum	VARCHAR(25), not null
PName	VARCHAR(75), not null
PUnitPrice	Number(7,2), not null
ComponentOf	VARCHAR(25)

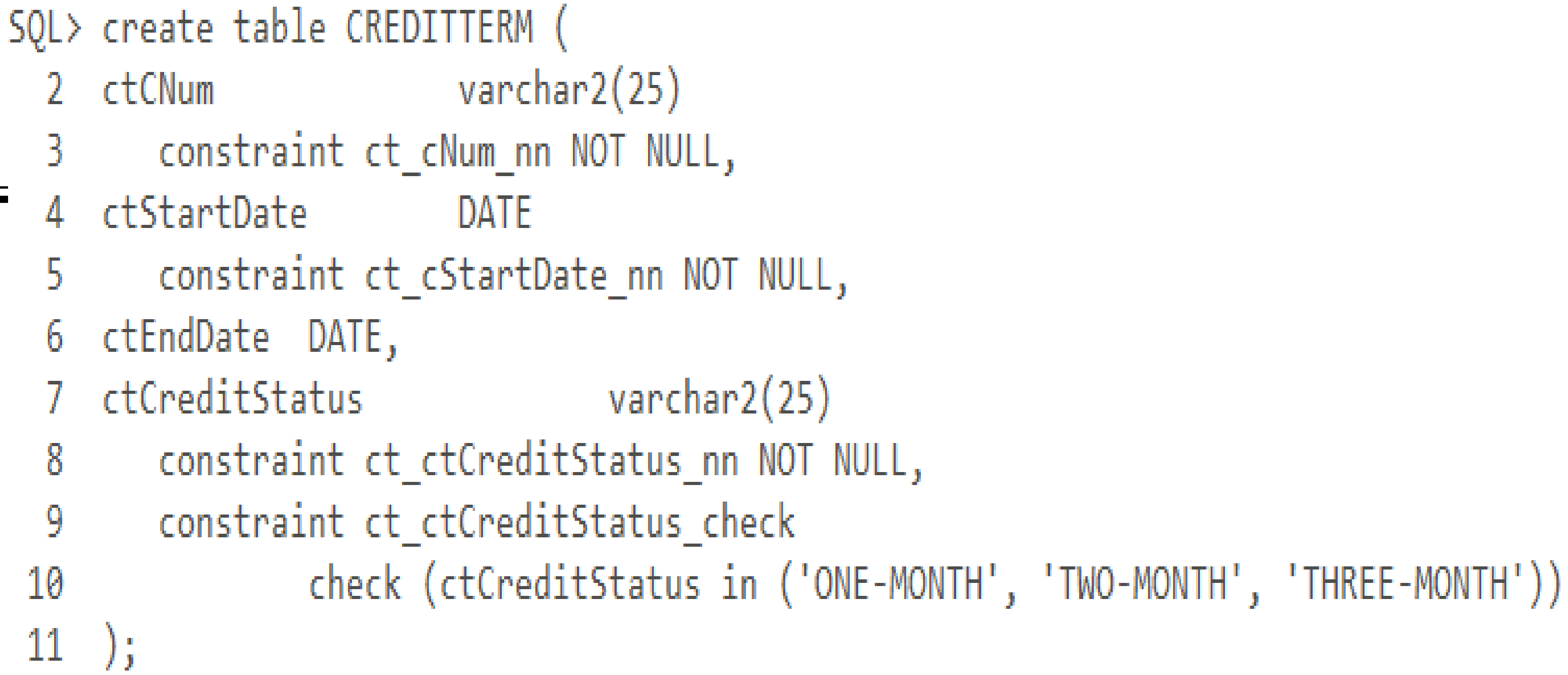


```
SQL> create table PART (
  2  pNum                         varchar2(25)
  3          constraint part_pNum_nn NOT NULL,
  4  pName                        varchar2(75)
  5          constraint part_pName_nn NOT NULL,
  6  pUnitPrice                   number(7,2)
  7          constraint part_pUnitPrice_nn NOT NULL,
  8  pComponentOf          varchar2(25),
  9  constraint part_pk primary key (pNum),
 10  constraint part_fk foreign key (pComponentOf)
 11          references PART(pNum)
 12          on delete set NULL
 13  );

Table created.

SQL>
```

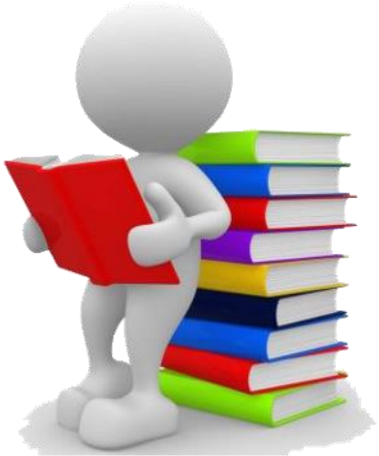
Exercise (15 minutes)

As an example, the management decided to give a credit status to each customer. The credit status can be a one-month credit term, a two-month credit term, or a three-month credit term. This credit status, given to the customers, valid for a period of one year with a specified start and end date. The credit status may be change at the end of the one-year validity period or earlier depending on the performance of the customer. Each customer is given one credit term at any one time, and a credit term may be given to many customers. The credit term is described as a string of not more than 20 characters.

Write an SQL statement to create the required table.



```
SQL> create table CREDITTERM (
  2  ctCNum                 varchar2(25)
  3     constraint ct_cNum_nn NOT NULL,
  4  ctStartDate        DATE
  5     constraint ct_cStartDate_nn NOT NULL,
  6  ctEndDate  DATE,
  7  ctCreditStatus              varchar2(25)
  8     constraint ct_ctCreditStatus_nn NOT NULL,
  9     constraint ct_ctCreditStatus_check
 10             check (ctCreditStatus in ('ONE-MONTH', 'TWO-MONTH', 'THREE-MONTH'))
 11  );

Table created.

SQL>
```



ALTER TABLE statement

Coloring

ALTER TABLE Statement

An existing database table can be altered at any time by means of the ALTER TABLE command. The definition of the ALTER TABLE command consists of six options to:

- Add a new column
- Define a new default for an existing column, replacing the previous one, if any
- Delete an existing column default
- Delete an existing column
- Specify a new database table integrity constraint
- Delete an existing database table integrity constraints.

ALTER TABLE Statement

```
ALTER TABLE table_name
[ADD [COLUMN] column_name data_type [NOT NULL] [UNIQUE]
[DEFAULT default_option] [CHECK (search_condition)]]
[DROP COLUMN column_name [RESTRICT | CASCADE]]
[MODIFY [COLUMN] column_name data_type [NOT NULL] [UNIQUE]
[ADD [CONSTRAINT [constraint_name]] table_constraint_definition]
[DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT default_option]
[ALTER [COLUMN] DROP DEFAULT]
```

- Square brackets indicate an optional element.
- Curly braces indicates a required element.
- An ellipsis (…) is used to indicate optional repetition of an item 0 or more times.
- A vertical bar ‘|’ indicates a choice among alternative.

ALTER TABLE Statement

- Assuming we want to add an attribute email to store the email of a customer. The attribute email is of type variable character length not exceeding 50 characters.

```
Alter table CUSTOMER
          add (
                    email varchar2(50)
          );
```

ALTER TABLE Statement

- For demonstration purposes, we will insert two customers information into the CUSTOMER table.
    i. Describe the structure of the CUSTOMER table.
    ii. Insert two customer records.
- Next, we add a new attribute SEX of type 1 character. The valid values are ‘M’ and ‘F’. The attribute is to be made mandatory.

ALTER TABLE Statement

```
Alter table CUSTOMER
        add (
                sex char(1)
                        constraint customer_sex_nn
                                NOT NULL,
                        constraint customer_sex_check
                                check (sex in (‘M’,’F’))
        );
```

Note: We should not be able to do this if the relational table is not empty. Instead, we should do it in 3 steps as shown next.

ALTER TABLE Statement

- We should do it in three steps:

```
ALTER TABLE CUSTOMER
        ADD (
                sex                     CHAR(1),
                CONSTRAINT cust_sex_check CHECK sex in (‘F’, ‘M’)
        );
```

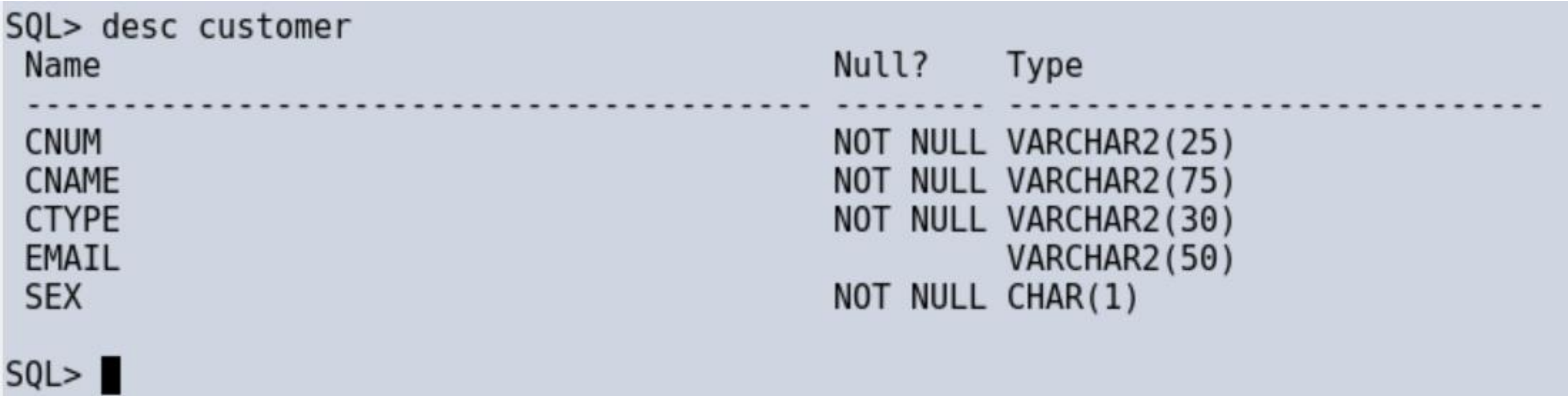
```
UPDATE CUSTOMER
        SET sex = ‘M’;
```

```
ALTER TABLE CUSTOMER
        MODIFY sex CHAR(1) NOT NULL;
```

EXERCISE (10 minutes)

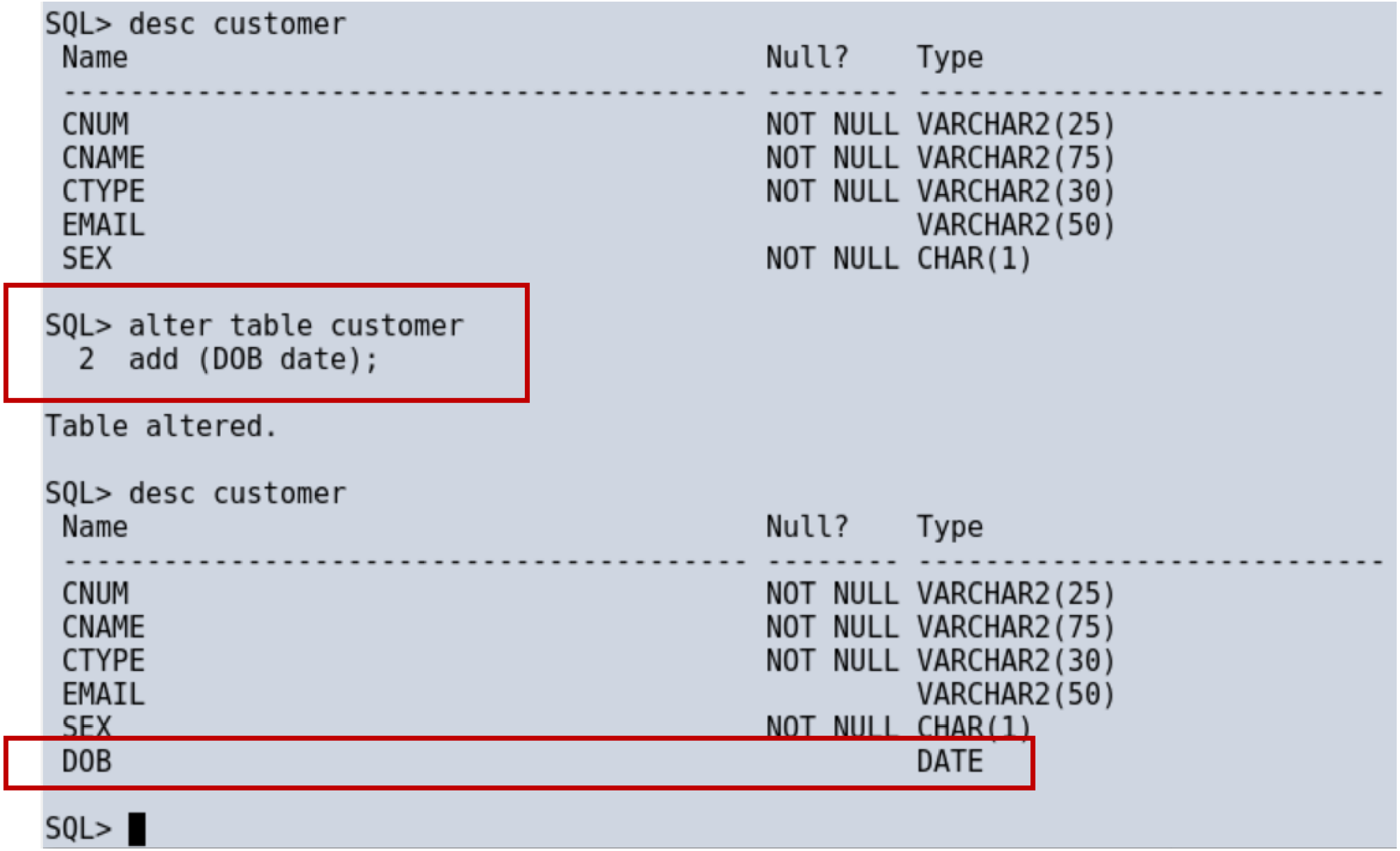
- Alter the table CUSTOMER to add an attribute DOB of type DATE.

CUSTOMER(CNum, CName, Ctype, email , sex)
primary key (CNum)
domain constraint ctype in ('INDIVIDUAL', 'INSTITUTION')



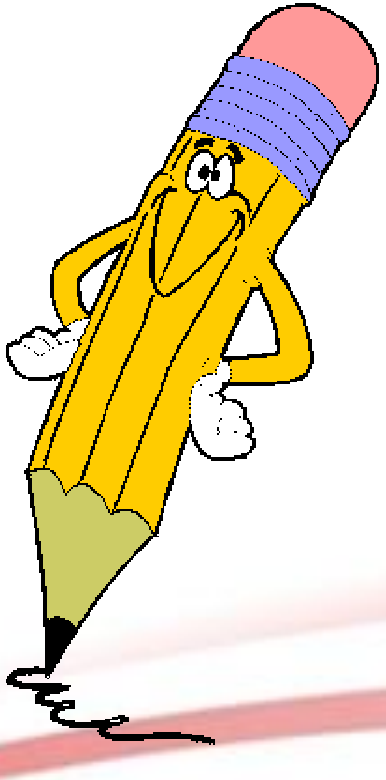
```
SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CNUM                                      NOT NULL VARCHAR2(25)
 CNAME                                     NOT NULL VARCHAR2(75)
 CTYPE                                     NOT NULL VARCHAR2(30)
 EMAIL                                              VARCHAR2(50)
 SEX                                       NOT NULL CHAR(1)

SQL> █
```

Sample answer



```
SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CNUM                                      NOT NULL VARCHAR2(25)
 CNAME                                     NOT NULL VARCHAR2(75)
 CTYPE                                     NOT NULL VARCHAR2(30)
 EMAIL                                              VARCHAR2(50)
 SEX                                       NOT NULL CHAR(1)

SQL> alter table customer
  2  add (DOB date);

Table altered.

SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CNUM                                      NOT NULL VARCHAR2(25)
 CNAME                                     NOT NULL VARCHAR2(75)
 CTYPE                                     NOT NULL VARCHAR2(30)
 EMAIL                                              VARCHAR2(50)
 SEX                                       NOT NULL CHAR(1)
 DOB                                                DATE

SQL> █
```

ALTER USER statement



ALTER user statement

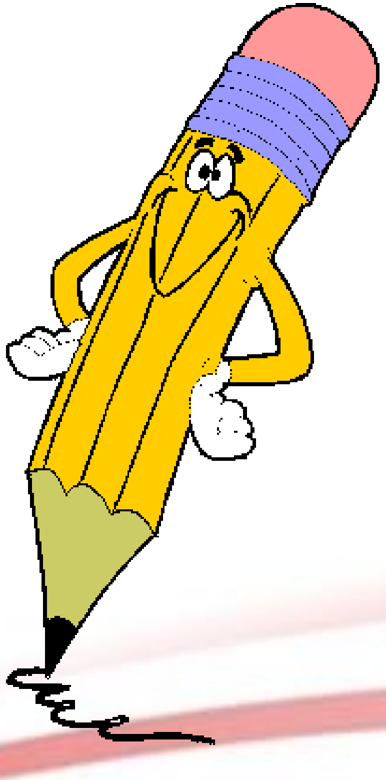
- ALTER USER statement changes the properties associated to the user’s account.
- For example, to change user’s account password:

```
ALTER USER student235 IDENTIFIED BY test235;
```

- If user’s account is locked due to some valid reason, we can unlock the user’s account as follow:

```
ALTER USER student235 ACCOUNT unlock;
```

DROP TABLE statement



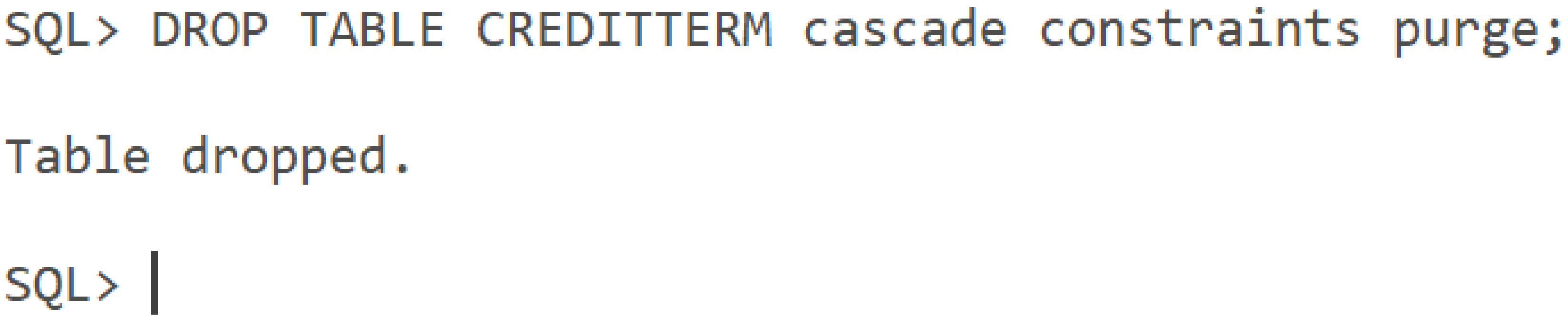
DROP TABLE statement

- Over times, some of the table becomes obsolete. We can DROP obsoleted tables using the following command:

```
DROP TABLE table_name;
DROP TABLE table_name cascade constraints;
DROP TABLE table_name purge;
DROP TABLE table_name casecade constraints purge;
```

Care must be taken when dropping a table. DO NOT simply drop table. Missing database table may corrupt your database.

- Drop the table on credit term that was created earlier.



```
SQL> DROP TABLE CREDITTERM cascade constraints purge;

Table dropped.

SQL> |
```

CREATE VIEW statement

CREATE VIEW

- A view is the dynamic result of one or more relational operations operating on the base relational tables to produce another set of data (result).
- A view is a virtual relational table that does not actually exist in the database but is produced upon request by a particular user, at the time of request.
- A view is commonly created to satisfy access requirement to specific set of data for operations when a user, for example, does not own the data but needs the data for processing.

CREATE VIEW

- The syntax of CREATE VIEW statement:

CREATE VIEW view_name [(column_name [, …])]
AS subselect [**WITH** [**CASCADED | LOCAL**] **CHECK OPTION**]

CREATE VIEW

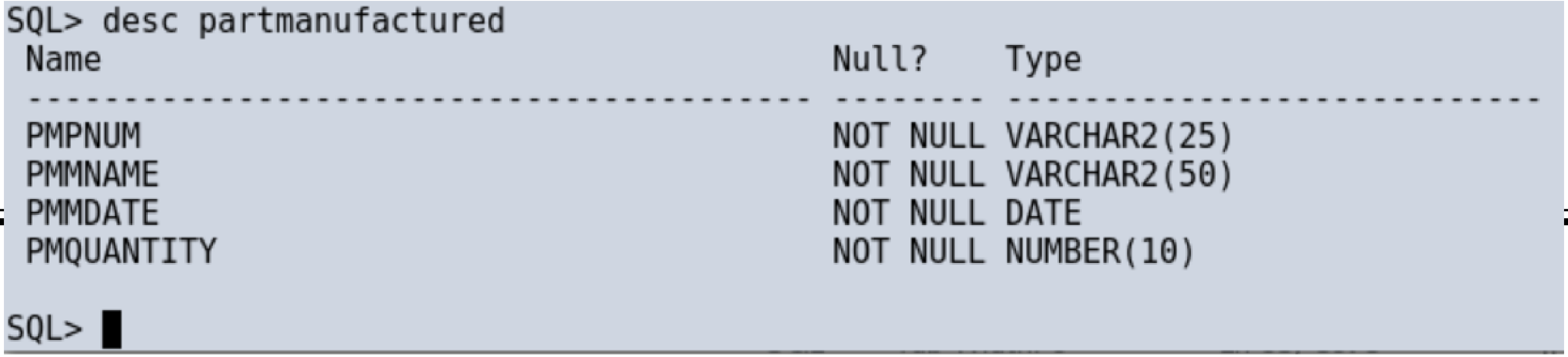
- For example, a user requested a read access to the total quantities of each parts manufactured by manufacturers. The user is interested to know only the name of manufacturers, the part number and total quantities.

PART-MANUFACTURED(MDate, PNum, MName, Quantity)

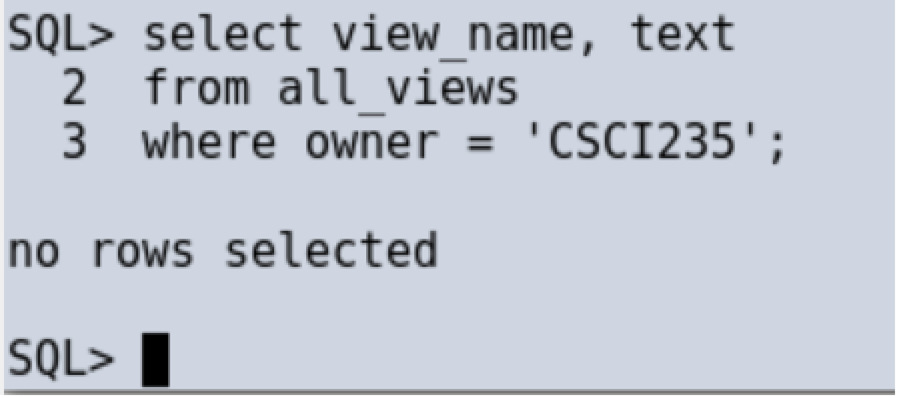
primary key (MName, PNum, MDate)

foreign key (PNum) references PART(PNum)

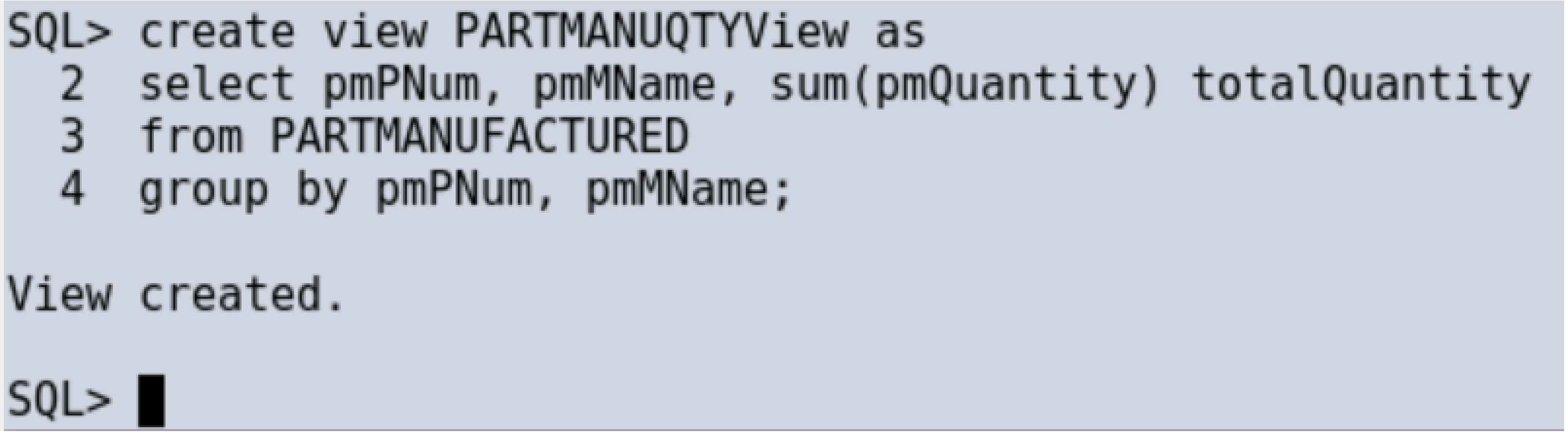
foreign key (MName) references MANUFACTURER(MName)



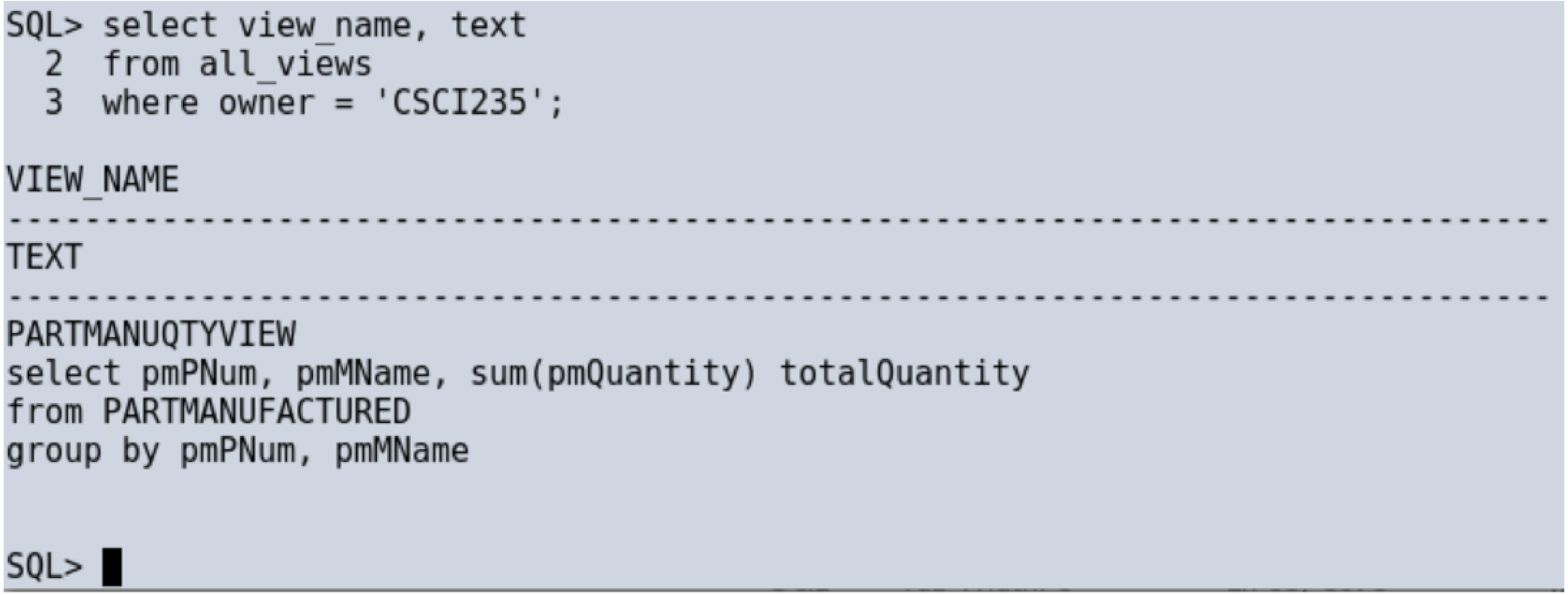
```
SQL> desc partmanufactured
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PMPNUM                                    NOT NULL VARCHAR2(25)
 PMMNAME                                   NOT NULL VARCHAR2(50)
 PMMDATE                                   NOT NULL DATE
 PMQUANTITY                                NOT NULL NUMBER(10)

SQL> 
```



```
SQL> select view_name, text
  2  from all_views
  3  where owner = 'CSCI235';

no rows selected

SQL> 
```



```
SQL> create view PARTMANUQTYView as
  2  select pmPNum, pmMName, sum(pmQuantity) totalQuantity
  3  from PARTMANUFACTURED
  4  group by pmPNum, pmMName;

View created.

SQL> 
```



```
SQL> select view_name, text
  2  from all_views
  3  where owner = 'CSCI235';

VIEW_NAME
--------------------------------------------------------------------------------
TEXT
--------------------------------------------------------------------------------
PARTMANUQTYVIEW
select pmPNum, pmMName, sum(pmQuantity) totalQuantity
from PARTMANUFACTURED
group by pmPNum, pmMName


SQL> █
```

CREATE INDEX statement

Coloring

CREATE INDEX

- An index is a structure that provides rapid access to the rows of a database table on the values of one or more columns.
- The presence of an index can significantly improve the performance of a query. However, since indexes may be updated by the system every time the underlying tables are updated, it incurred additional overheads.
- Indexes are usually created to satisfy particular search criteria after the table has been in use for some time and has grown in size.

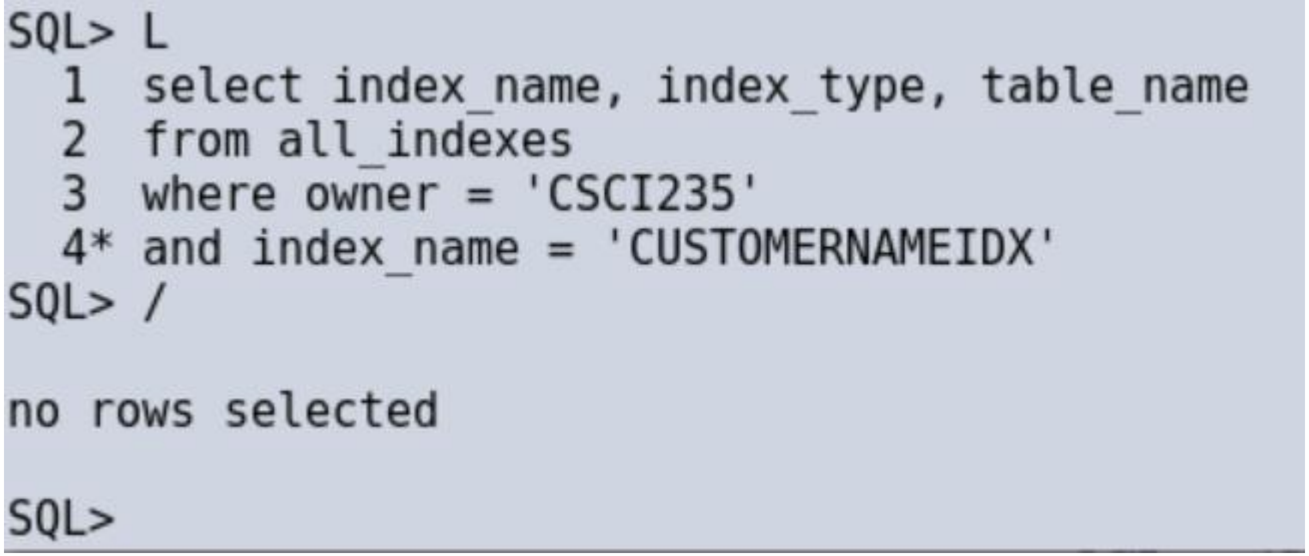
CREATE INDEX

- The syntax of CREATE INDEX statement:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column [ASC | DESC] [, …])
```

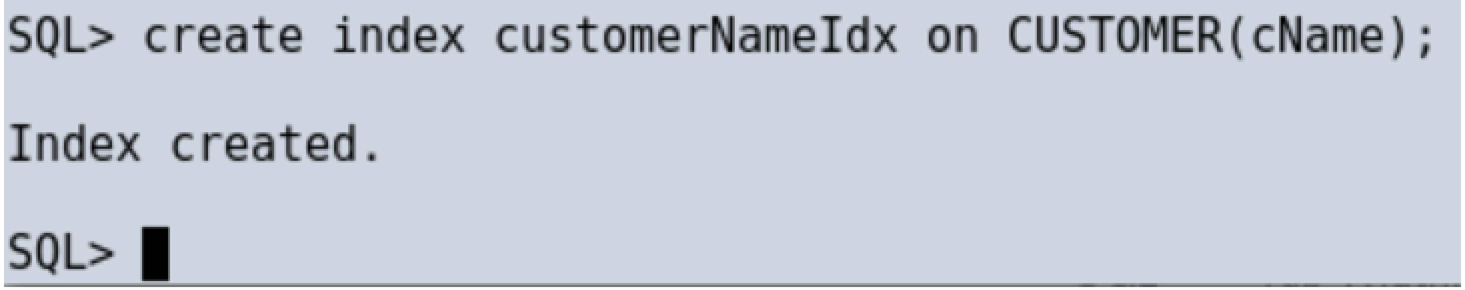
CREATE INDEX

- For example, we want a to create additional index on customer name in CUSTOMER relational table so that the user can speed up the query process using customer name.

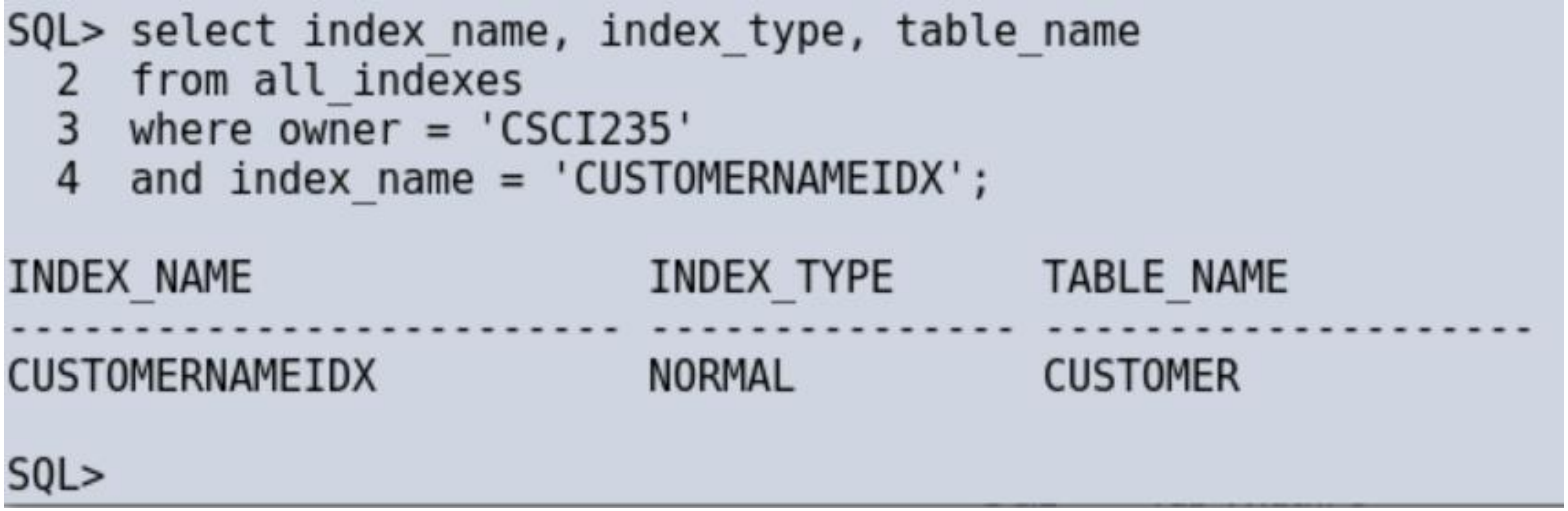


```
SQL> L
  1  select index_name, index_type, table_name
  2  from all_indexes
  3  where owner = 'CSCI235'
  4* and index_name = 'CUSTOMERNAMEIDX'
SQL> /

no rows selected

SQL>
```

CREATE INDEX



```
SQL> create index customerNameIdx on CUSTOMER(cName);

Index created.

SQL> 
```



```
SQL> select index_name, index_type, table_name
  2  from all_indexes
  3  where owner = 'CSCI235'
  4  and index_name = 'CUSTOMERNAMEIDX';

INDEX_NAME                INDEX_TYPE      TABLE_NAME
------------------------- --------------- --------------------
CUSTOMERNAMEIDX           NORMAL          CUSTOMER

SQL>
```

Grant database privileges

Coloring

Grant database privileges

- GRANT statement is applied by a user to authorise various kinds of access to his/her relational tables/views by another user or class of users

Note: in the following examples, we would assume that the users granting the privileges must own the relational tables or must have the grant option privileges (will explain in more detail later) to the relational tables in order to grant the said privileges to other users.

Grant database privileges

- The syntax for granting privileges on database objects in Oracle is as follow:

```
GRANT privileges ON object TO user;
```

Grant database privileges

The privileges can be any of the following:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

GRANT database objects privileges

The following are some example of GRANT statement:

```
GRANT SELECT ON Department TO sjapit;
GRANT ALL ON Course_View TO scott;
GRANT DELETE ON Course TO PUBLIC;
GRANT REFERENCES name ON Department TO sjapit;
GRANT SELECT ON Course TO scott WITH GRANT OPTION;
```

Revoke database privileges

- REVOKE statement is applied by a user to revoke the access rights granted to another user or class of users.
- The syntax for granting privileges on database objects in Oracle is as follow:

REVOKE privileges **ON** object **FROM** user;

Revoke database privileges

- The following are some example of REVOKE statement:

```
REVOKE SELECT ON CUSTOMER FROM sjapit;
REVOKE ALL ON PARTMANUQTYView FROM sjapit;
REVOKE DELETE ON PART FROM PUBLIC;
REVOKE REFERENCES pNum ON PART FROM sjapit;
REVOKE SELECT ON MANUFACTURER FROM sjapit
WITH GRANT OPTION;
```

Data Manipulation Language (DML)

Data Manipulation Language (DML)

- Data manipulation statements (DML) are statements used to manipulate and query database objects. Manipulating means insert, delete, or update the database objects. Oracle provides the following four data manipulation languages.

Data Manipulation Language (DML)

Command	Description
INSERT	INSERT statement inserts a new row into a relational table and automatically verifies the consistency constraints.
DELETE	DELETE statement deletes all rows that satisfy a given condition and automatically verifies the consistency constraints.
UPDATE	UPDATE statement modifies all rows that satisfy a given condition and automatically verifies the consistency constraints.
SELECT	SELECT statement retrieves data from a relational database.

Data Manipulation Language (DML)

INSERT statement

INSERT statement

- INSERT statement has two forms of the syntax:

INSERT INTO table-name **VALUES** (<list of values satisfying the domains separated by commas>);

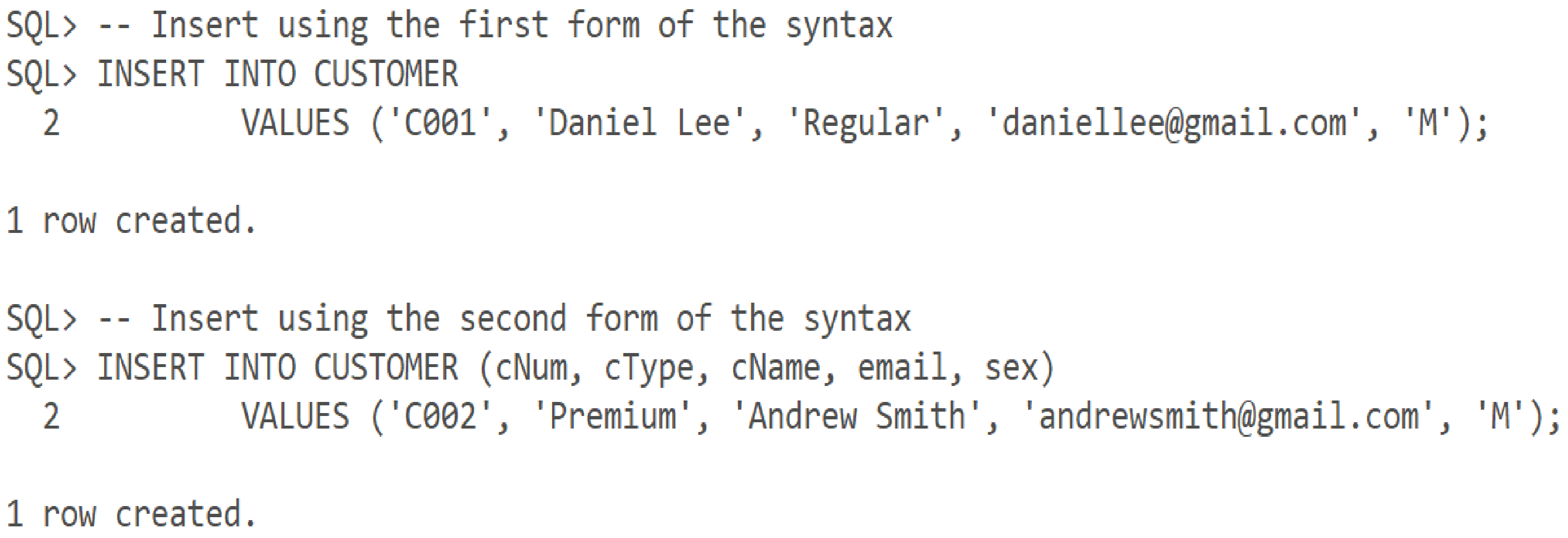
OR

INSERT INTO table-name (<list of attributes name separated by commas>)

VALUES (<list of values satisfying the domains of the attributes specified, separated by commas>);

INSERT statement

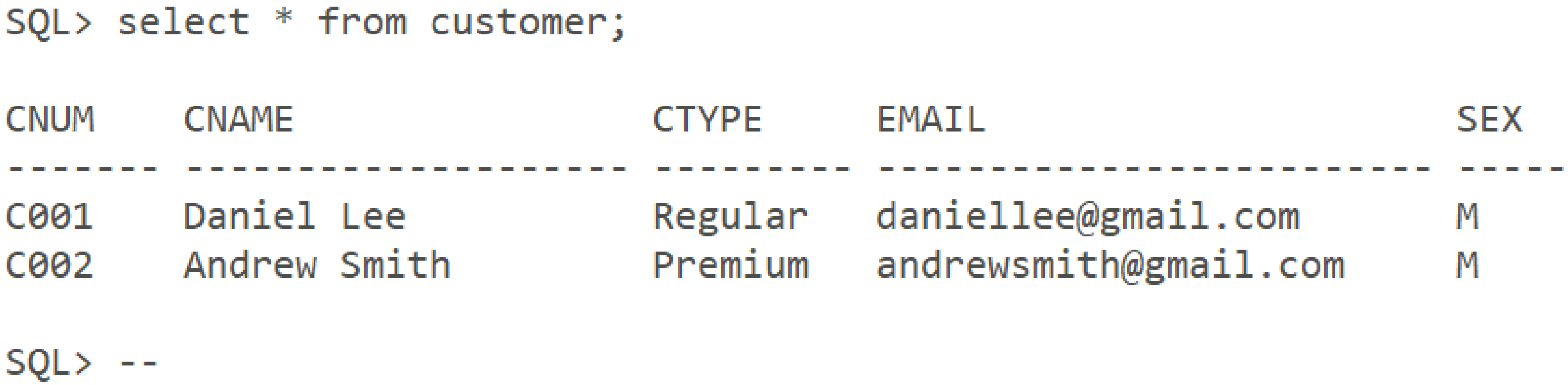
Inserting two customer records into CUSTOMER table.



```
SQL> -- Insert using the first form of the syntax
SQL> INSERT INTO CUSTOMER
  2          VALUES ('C001', 'Daniel Lee', 'Regular', 'daniellee@gmail.com', 'M');

1 row created.

SQL> -- Insert using the second form of the syntax
SQL> INSERT INTO CUSTOMER (cNum, cType, cName, email, sex)
  2          VALUES ('C002', 'Premium', 'Andrew Smith', 'andrewsmith@gmail.com', 'M');

1 row created.
```

INSERT statement

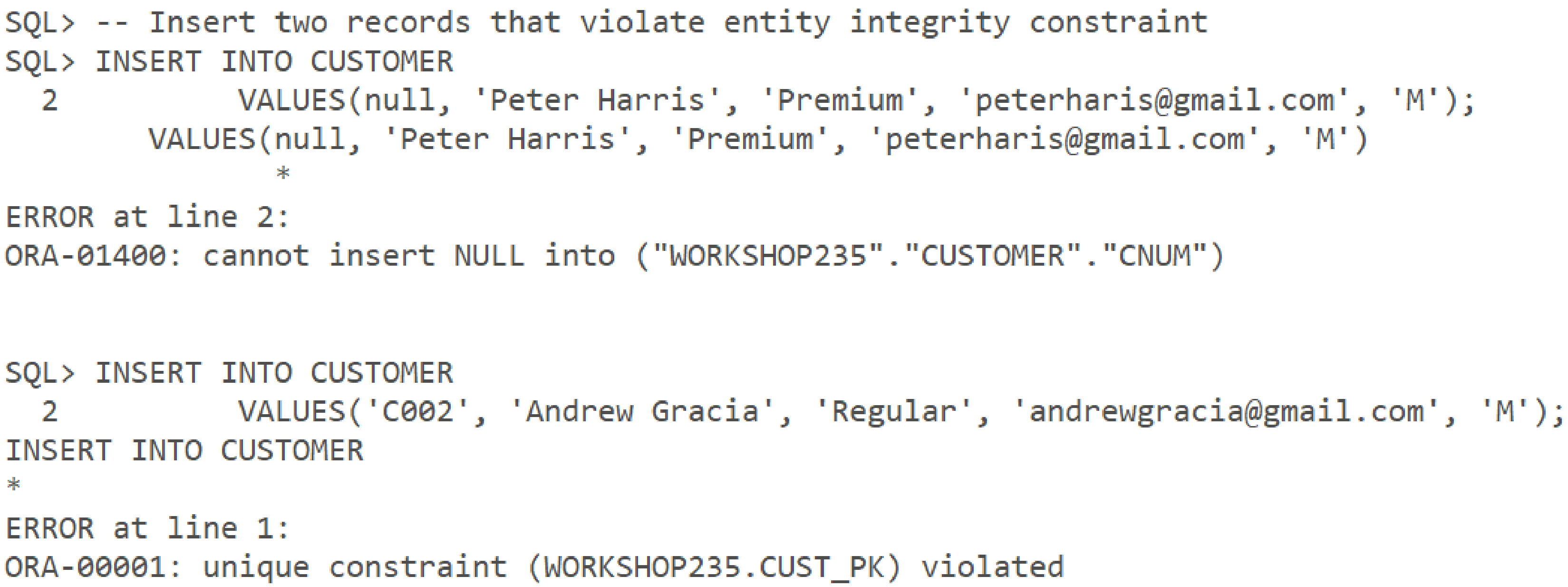
- Verify the content of the relational table CUSTOMER after the insertion.



```
SQL> select * from customer;

CNUM    CNAME                CTYPE     EMAIL                     SEX
------- -------------------- --------- ------------------------- -----
C001    Daniel Lee           Regular   daniellee@gmail.com       M
C002    Andrew Smith         Premium   andrewsmith@gmail.com     M

SQL> --
```

INSERT statement

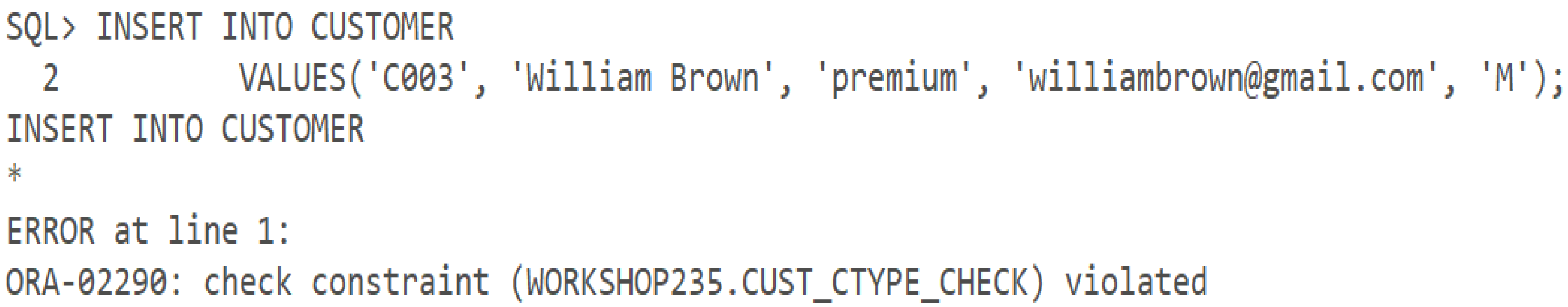
- Insert two records into CUSTOMER relational table that violate entity integrity rule.



```
SQL> -- Insert two records that violate entity integrity constraint
SQL> INSERT INTO CUSTOMER
  2          VALUES(null, 'Peter Harris', 'Premium', 'peterharis@gmail.com', 'M');
        VALUES(null, 'Peter Harris', 'Premium', 'peterharis@gmail.com', 'M')
               *
ERROR at line 2:
ORA-01400: cannot insert NULL into ("WORKSHOP235"."CUSTOMER"."CNUM")


SQL> INSERT INTO CUSTOMER
  2          VALUES('C002', 'Andrew Gracia', 'Regular', 'andrewgracia@gmail.com', 'M');
INSERT INTO CUSTOMER
*
ERROR at line 1:
ORA-00001: unique constraint (WORKSHOP235.CUST_PK) violated
```

INSERT statement

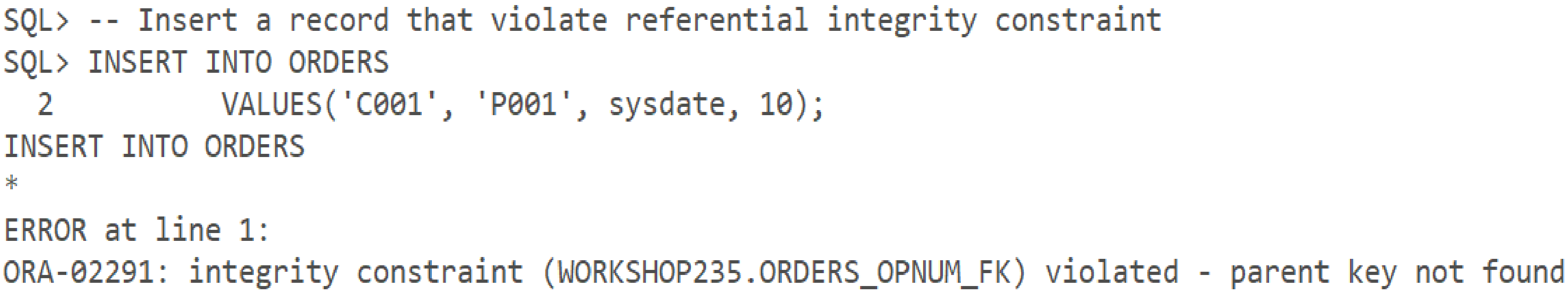
Insert a record into CUSTOMER relational table that violate semantic constraint (CHECK constraint)



```
SQL> INSERT INTO CUSTOMER
  2          VALUES('C003', 'William Brown', 'premium', 'williambrown@gmail.com', 'M');
INSERT INTO CUSTOMER
*
ERROR at line 1:
ORA-02290: check constraint (WORKSHOP235.CUST_CTYPE_CHECK) violated
```

INSERT statement

Insert a record into ORDERS relational table that violate referential integrity constraint.



```
SQL> -- Insert a record that violate referential integrity constraint
SQL> INSERT INTO ORDERS
  2          VALUES('C001', 'P001', sysdate, 10);
INSERT INTO ORDERS
*
ERROR at line 1:
ORA-02291: integrity constraint (WORKSHOP235.ORDERS_OPNUM_FK) violated - parent key not found
```

Data Manipulation Language (DML)

DELETE statement

DELETE statement

- DELETE statement deletes all rows that satisfy a given condition.
- The rows deleted by DELETE statement CAN be restored by ROLLBACK statement unless DELETE has been committed by COMMIT statement.
- DELETE statement DOES NOT delete a table.
- DELETE statement DOES NOT release disk storage occupied by the deleted rows.

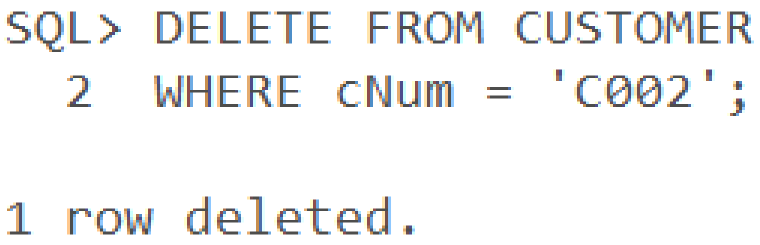
DELETE statement

- The syntax of a DELETE statement:

```
DELETE FROM table_name
[WHERE <condition>];
```

DELETE statement

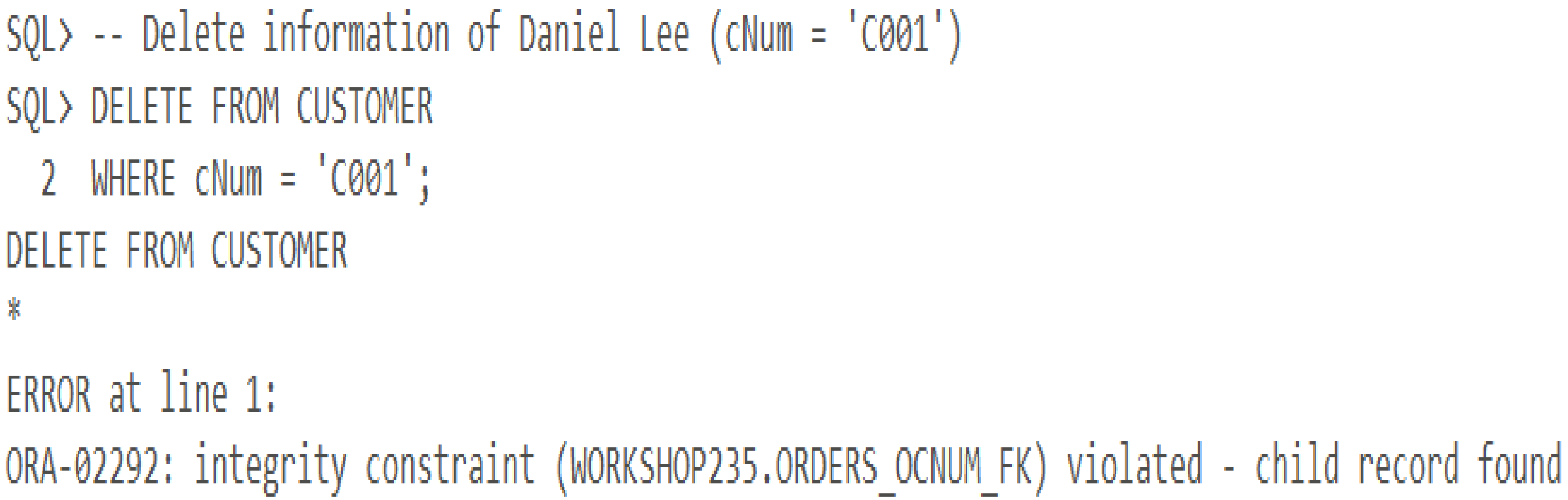
- For example, to delete the information of customer Andrew Smith from the customer table.



```
SQL> DELETE FROM CUSTOMER
  2  WHERE cNum = 'C002';

1 row deleted.
```

DELETE statement

- Unsuccessful deletion of customer due to violation of referential constraint.



```
SQL> -- Delete information of Daniel Lee (cNum = 'C001')
SQL> DELETE FROM CUSTOMER
  2  WHERE cNum = 'C001';
DELETE FROM CUSTOMER
*

ERROR at line 1:
ORA-02292: integrity constraint (WORKSHOP235.ORDERS_OCNUM_FK) violated - child record found
```

DELETE statement

- If ON DELETE CASCADE clause is not used in a specification of foreign key in ORDERS table, then an order in which the rows are deleted is important.

Data Manipulation Language (DML)

UPDATE statement

UPDATE statement

- UPDATE statement modifies all rows that satisfy a given condition.
- The values of attributes modified by UPDATE statements CAN be restored by ROLLBACK statement unless the UPDATE process has been committed by COMMIT statement.

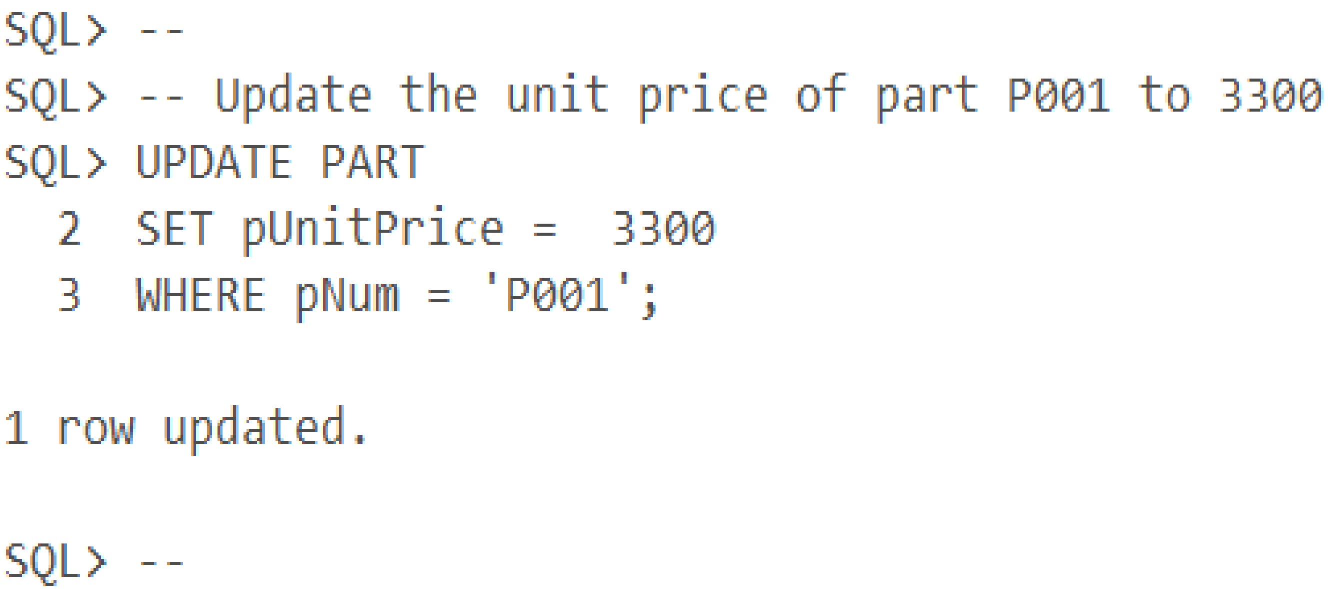
UPDATE statement

- The syntax of an UPDATE statement:

```
UPDATE <TABLE>
SET <ASSIGNMENTS>
WHERE <CONDITION>;
```

UPDATE statement

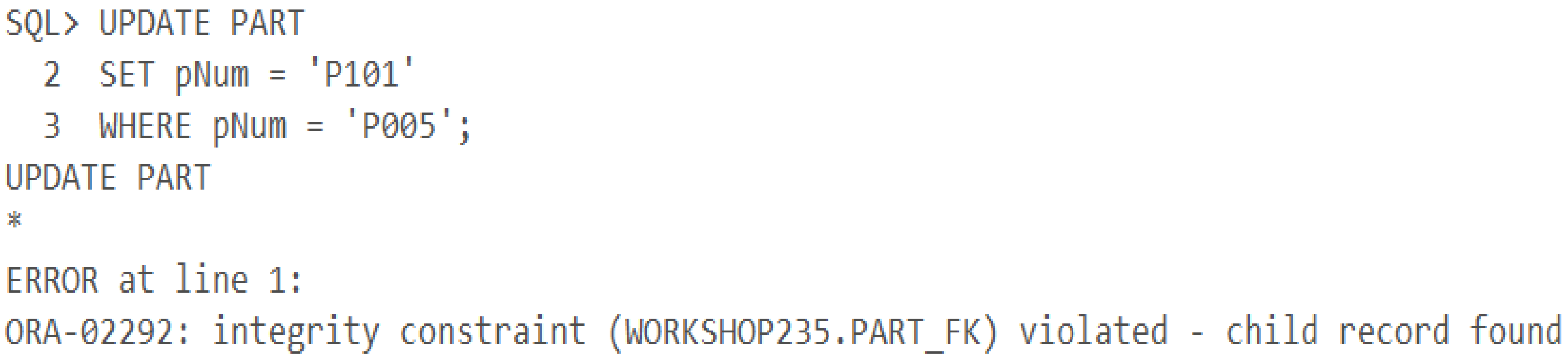
- For example, a user of the database wants to change the unit price of the part P001 to 3300.



```
SQL> --
SQL> -- Update the unit price of part P001 to 3300
SQL> UPDATE PART
  2  SET pUnitPrice =  3300
  3  WHERE pNum = 'P001';

1 row updated.

SQL> --
```

UPDATE statement

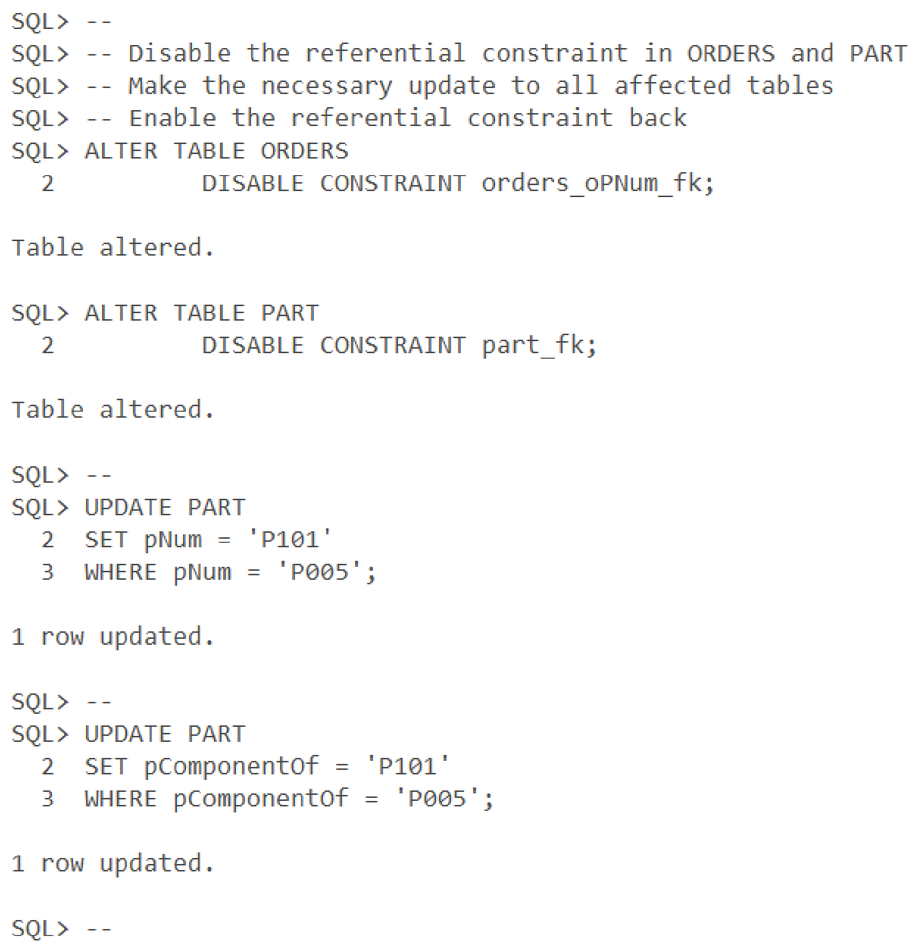
- An example of unsuccessful update operation because it violates referential integrity constraint.



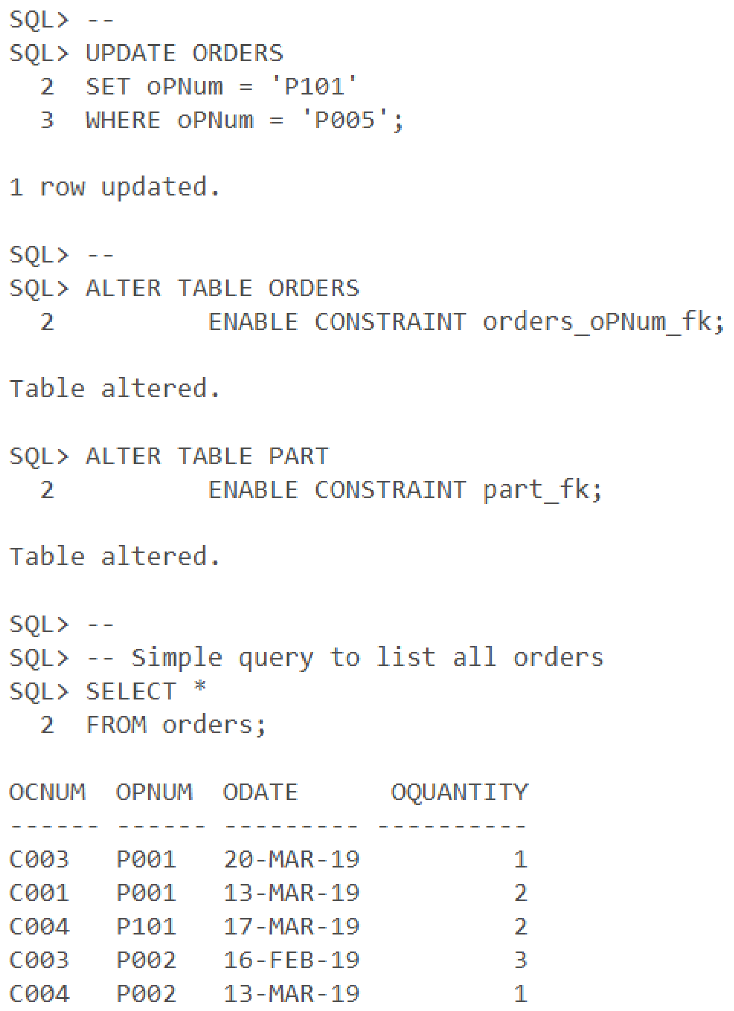
```
SQL> UPDATE PART
  2  SET pNum = 'P101'
  3  WHERE pNum = 'P005';
UPDATE PART
*
ERROR at line 1:
ORA-02292: integrity constraint (WORKSHOP235.PART_FK) violated - child record found
```

UPDATE statement

- The violation occurred in the previous example is because we try to change a primary key that has been referenced by some other relational tables.
- To resolve this problem, we need to perform a series to operations:
    i. Disable the referential constraint from the referencing table.
    ii. Make the necessary update to all affected tables.
    iii. Enable the referential constraint in the referencing table.



```
SQL> --
SQL> -- Disable the referential constraint in ORDERS and PART
SQL> -- Make the necessary update to all affected tables
SQL> -- Enable the referential constraint back
SQL> ALTER TABLE ORDERS
  2          DISABLE CONSTRAINT orders_oPNum_fk;

Table altered.

SQL> ALTER TABLE PART
  2          DISABLE CONSTRAINT part_fk;

Table altered.

SQL> --
SQL> UPDATE PART
  2  SET pNum = 'P101'
  3  WHERE pNum = 'P005';

1 row updated.

SQL> --
SQL> UPDATE PART
  2  SET pComponentOf = 'P101'
  3  WHERE pComponentOf = 'P005';

1 row updated.

SQL> --
```



```
SQL> --
SQL> UPDATE ORDERS
  2  SET oPNum = 'P101'
  3  WHERE oPNum = 'P005';

1 row updated.

SQL> --
SQL> ALTER TABLE ORDERS
  2          ENABLE CONSTRAINT orders_oPNum_fk;

Table altered.

SQL> ALTER TABLE PART
  2          ENABLE CONSTRAINT part_fk;

Table altered.

SQL> --
SQL> -- Simple query to list all orders
SQL> SELECT *
  2  FROM orders;

OCNUM  OPNUM  ODATE      OQUANTITY
------ ------ --------- ----------
C003   P001   20-MAR-19          1
C001   P001   13-MAR-19          2
C004   P101   17-MAR-19          2
C003   P002   16-FEB-19          3
C004   P002   13-MAR-19          1
```

Data Manipulation Language (DML)

SELECT statement

SELECT statement

- SELECT statement retrieves data from a relational database.
- The results of SELECT statement can be considered as a transient relational table.
- The results of SELECT statement can be saved in a persistent relational table.
- SELECT statement does not change the content of the database table; it just retrieves information to display as requested.

SELECT statement

- The syntax of typical SELECT statement.

SELECT [**DISTINCT** | **ALL**]

{ * | [column_expression [**AS** new_name]] [, …] }

FROM table_name [alias] [, …]

[**WHERE** condition]

[**GROUP BY** column_list] [**HAVING** condition]

[**ORDER BY** column_list]

- Bold words are keywords
- Square brackets indicate an optional element.
- Curly braces indicate a required element.
- An ellipsis (…) is used to indicate optional repetition of an item 0 or more times.
- A vertical bar '|' indicates a choice among alternative.

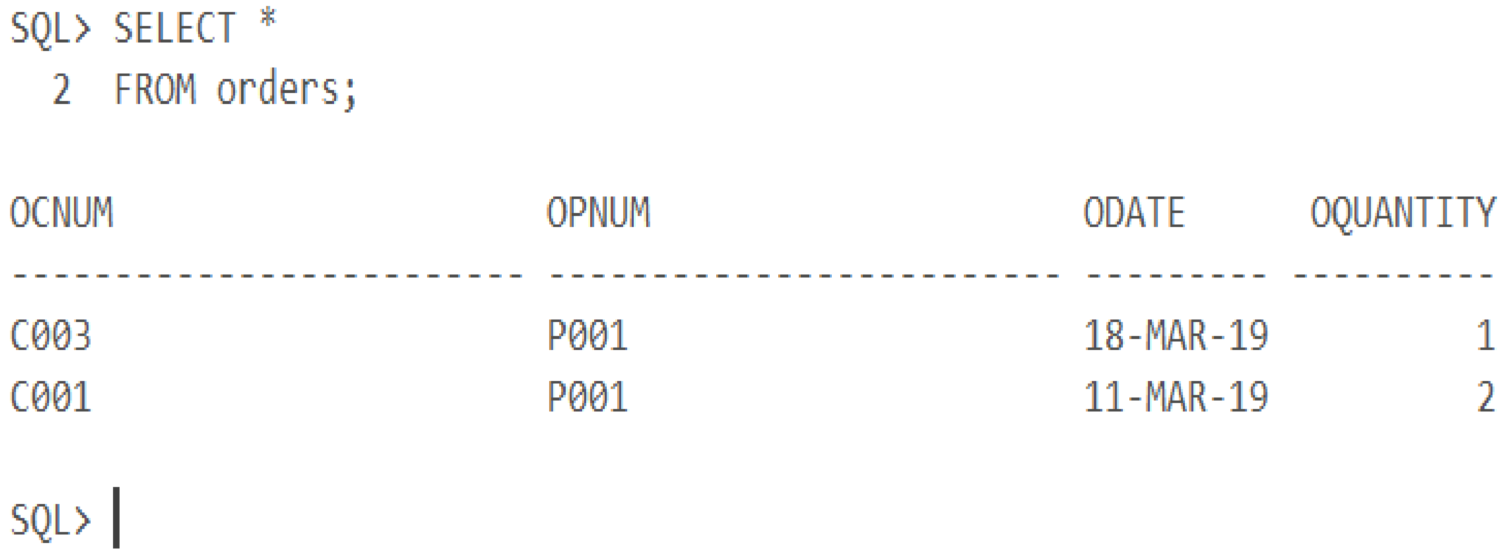
SELECT statement

- When a query (SELECT statement) is sent to the dbms, the query processor will process the query in the following sequence:

FROM	Specifies the table or tables to be used.
WHERE	Filters the rows subject to some condition.
GROUP BY	Forms groups of rows with the same column value.
HAVING	Filters the groups subject to some condition.
SELECT	Specifies which columns are to appear in.
ORDER BY	Sorted the output in ascending or descending order.

Basic SQL query

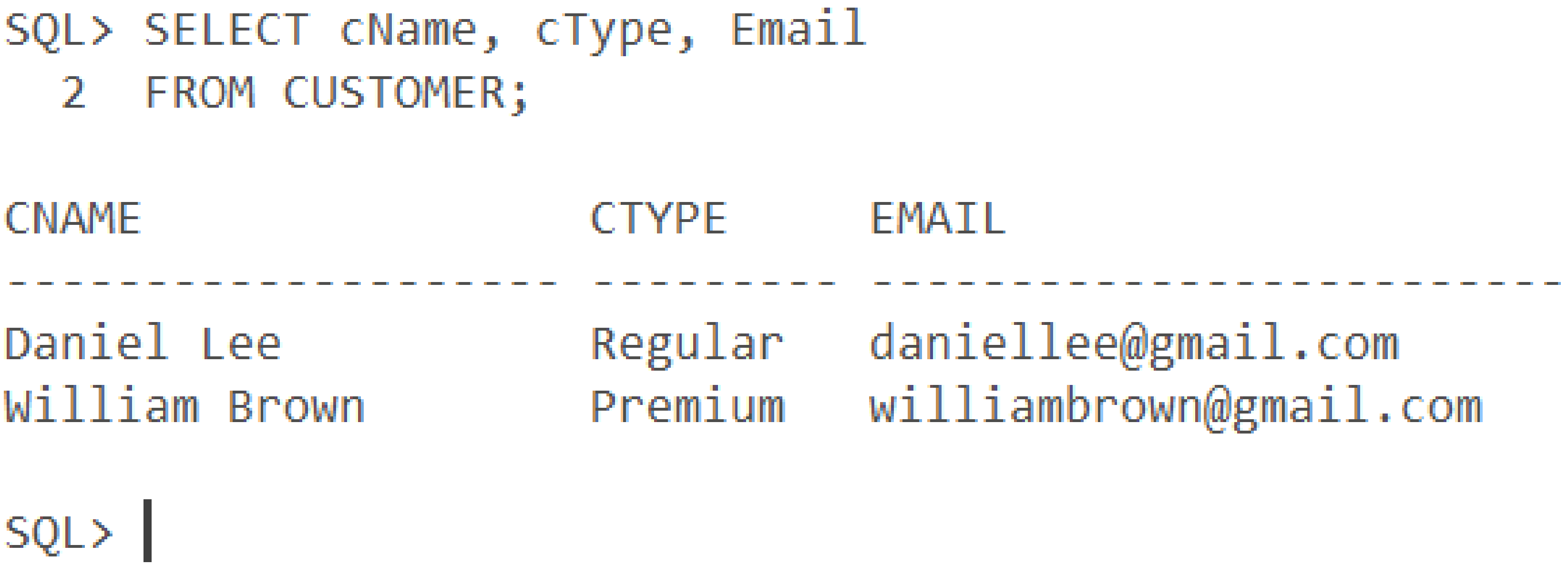
- For example, a user wants to produce the full details of orders made by all customers.



```
SQL> SELECT *
  2  FROM orders;

OCNUM                     OPNUM                     ODATE      OQUANTITY
------------------------- ------------------------- --------- ----------
C003                      P001                      18-MAR-19          1
C001                      P001                      11-MAR-19          2

SQL> |
```

Basic SQL query

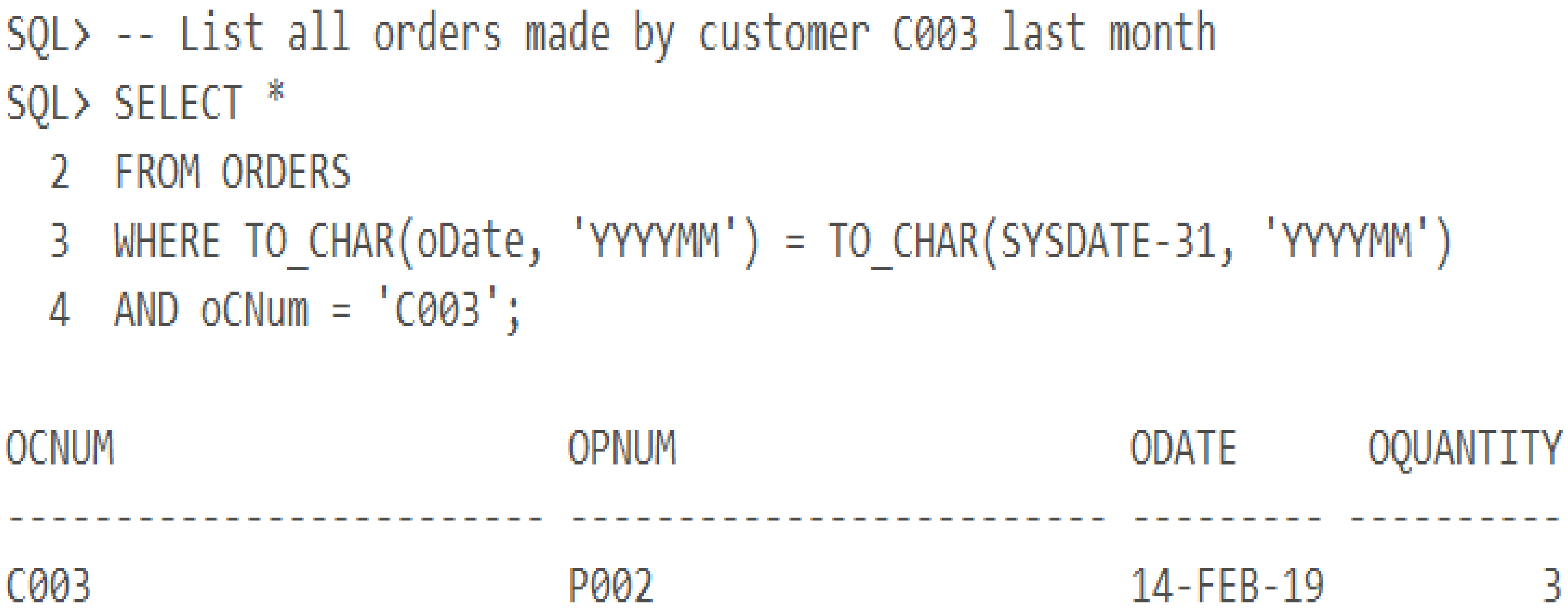
- For example, a user wants to list the customer name, email, and customer type.



```
SQL> SELECT cName, cType, Email
  2  FROM CUSTOMER;

CNAME                CTYPE     EMAIL
-------------------- --------- -------------------------
Daniel Lee           Regular   daniellee@gmail.com
William Brown        Premium   williambrown@gmail.com

SQL>
```

Query with predicates (condition)

- For example, a user wants to see all the orders made last month by the customer ‘C003’.



```
SQL> -- List all orders made by customer C003 last month
SQL> SELECT *
  2  FROM ORDERS
  3  WHERE TO_CHAR(oDate, 'YYYYMM') = TO_CHAR(SYSDATE-31, 'YYYYMM')
  4  AND oCNum = 'C003';

OCNUM                     OPNUM                     ODATE      OQUANTITY
------------------------- ------------------------- --------- ----------
C003                      P002                      14-FEB-19          3
```

Query with predicates (condition)

- When evaluating the conditions specified in the predicate, Oracle provides the following comparison operators:

Operator	Meaning
=	Equals
<	Is less than
>	Is greater than
<=	Is less than or equal to
>=	Is greater than or equal to
<>	Is not equal to
!=	Is not equal to (allowed in some dialects)

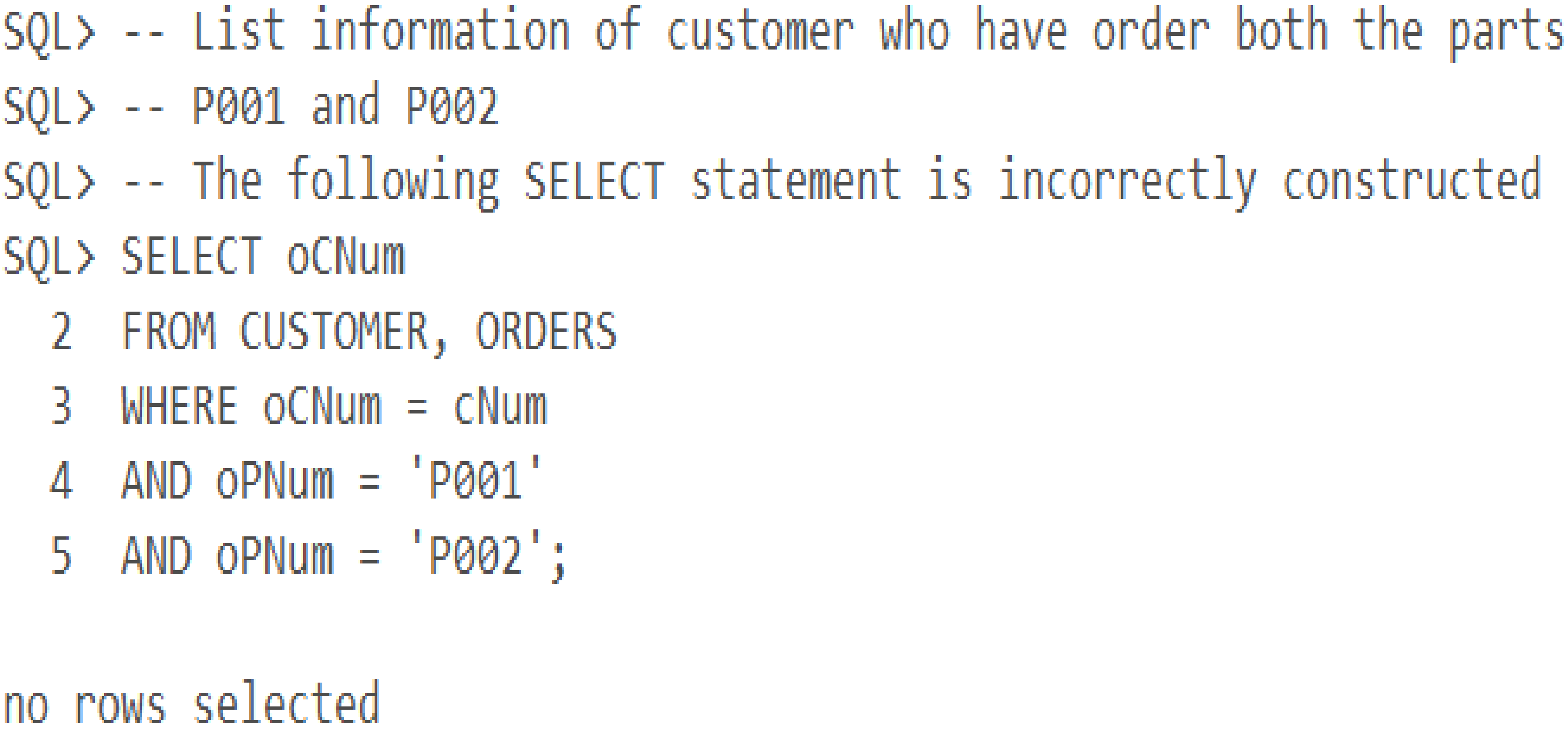
Query with predicates (condition)

- More complex search conditions can be generated using the logical operators AND, OR and NOT, with parentheses to show the order of evaluation. The rules for evaluating a conditional expression are:
  - An expression is evaluated **left to right**.
  - Subexpressions in brackets are evaluated first.
  - **NOT**s are evaluated before **AND**s and **OR**s.
  - **AND**s are evaluated before **OR**s.

Oracle allows set operators such as **INTERSECT**, **UNION** and **MINUS** to be used in SQL.

For example, a user wants to list the customer information who had ordered **both** the product ‘P001’ and ‘P002’.

Queries with INTERSECT, UNION, and MINUS operators

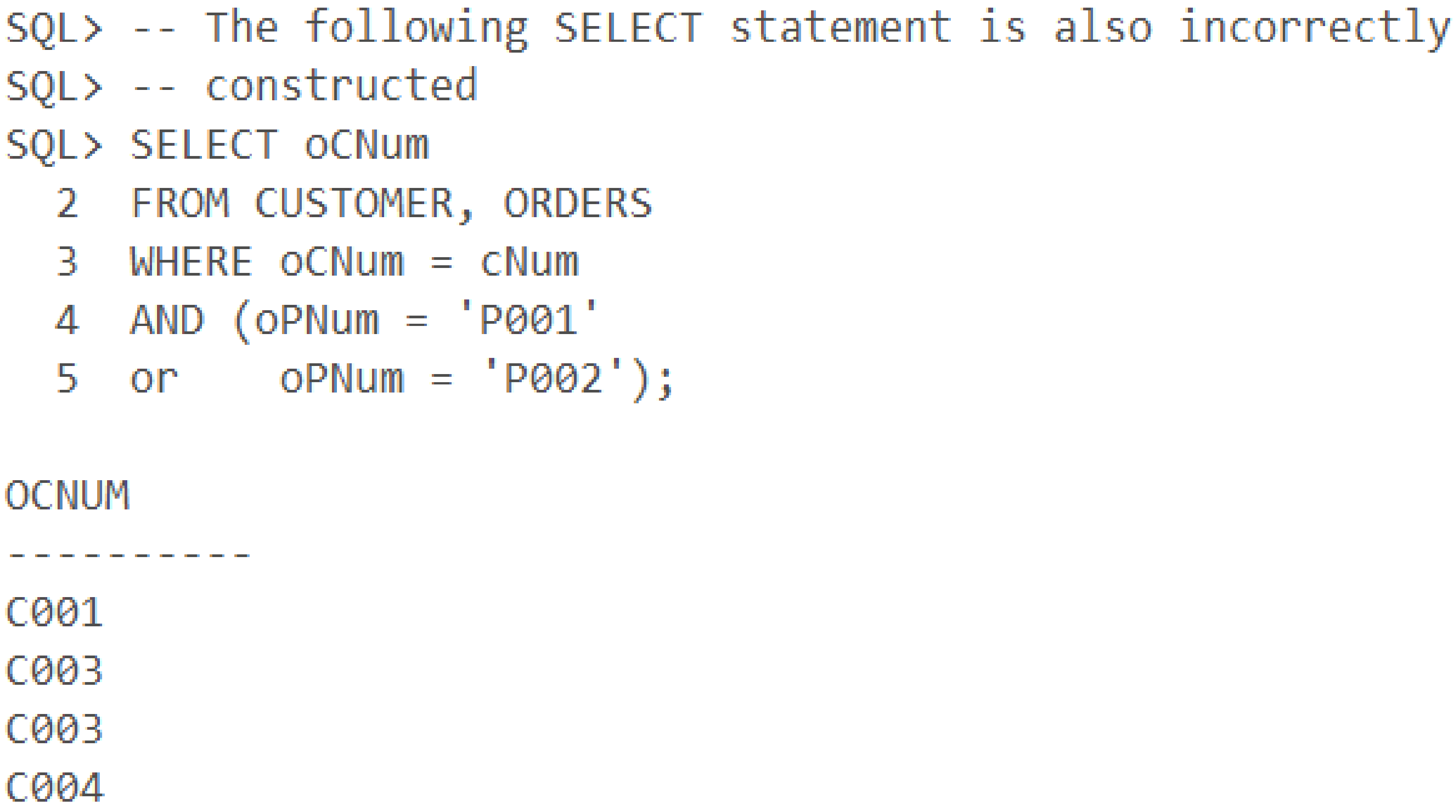


```
SQL> -- List information of customer who have order both the parts
SQL> -- P001 and P002
SQL> -- The following SELECT statement is incorrectly constructed
SQL> SELECT oCNum
  2  FROM CUSTOMER, ORDERS
  3  WHERE oCNum = cNum
  4  AND oPNum = 'P001'
  5  AND oPNum = 'P002';

no rows selected
```

Queries with INTERSECT, UNION, and MINUS operators

- The query shown above is incorrect because in the query, the attribute oPNum is tested twice using ‘AND’ logical operator;
    - the first time it is tested for part ‘P001’ and the second time it is tested for part ‘P002’.
    - In relational table, attribute value is atomic, that is, single value.
    - For a row if the value is ‘P001’, the same attribute will never be having another value ‘P002’. Hence, the condition oPNum = ‘P001’ AND oPNum = ‘P002’ will not happen.
    - That is why the query returns ‘no rows selected’ as its result.

Queries with INTERSECT, UNION, and MINUS operators

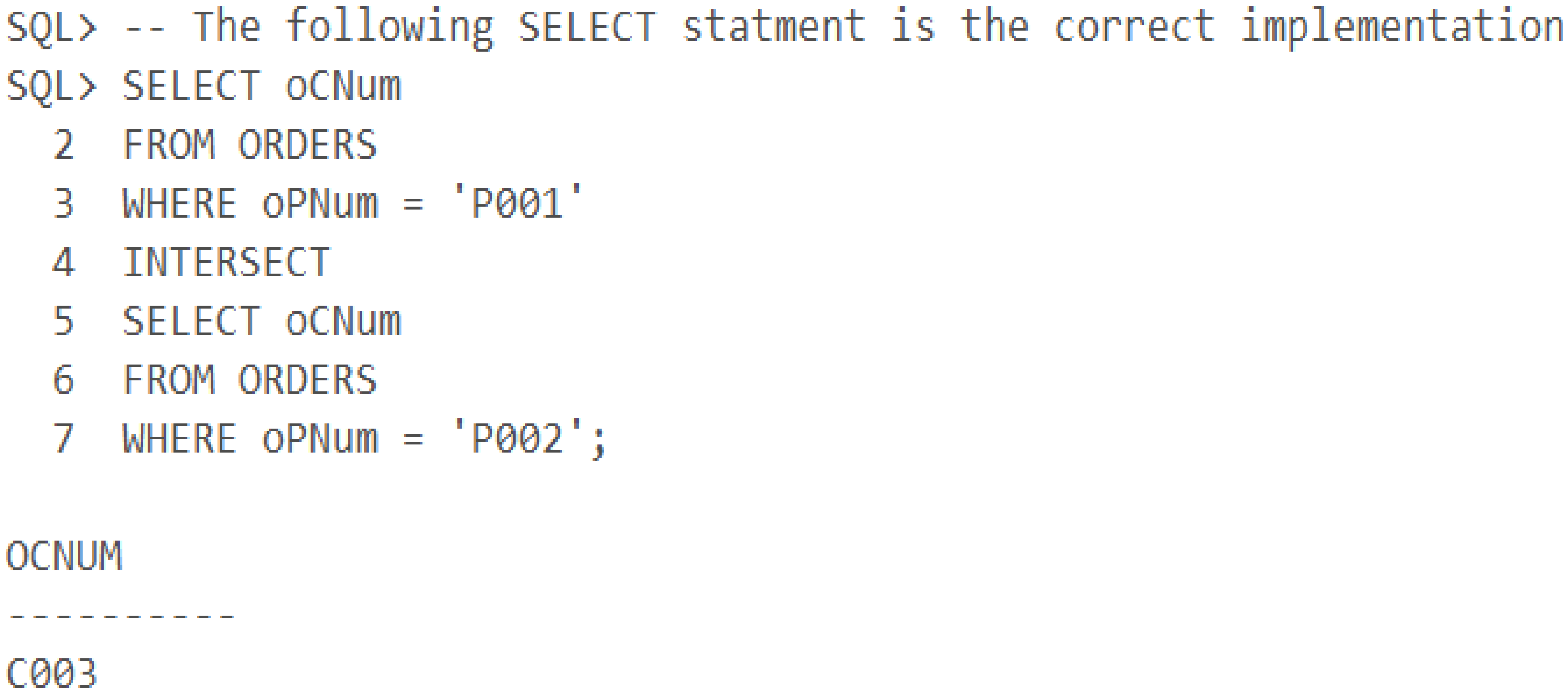


```
SQL> -- The following SELECT statement is also incorrectly
SQL> -- constructed
SQL> SELECT oCNum
  2  FROM CUSTOMER, ORDERS
  3  WHERE oCNum = cNum
  4  AND (oPNum = 'P001'
  5  or    oPNum = 'P002');

OCNUM
----------
C001
C003
C003
C004
```

Queries with INTERSECT, UNION, and MINUS operators

- In this query, the attribute oPNum is also tested two times, but it is tested with logical ‘OR’.
- We get some results now. They are customer numbers C001, C003 (two times) and C004.
- At a glance, it seems like these are the results, but with a careful analysis of the data, it becomes clear the result is incorrect.
- The keyword of the specification is ‘both’, which indicates that a customer must order both the parts ‘P001’ and ‘P002’, not just either one.
- If we carefully look at the records contained in ORDERS table, it is noted that customer C001 only ordered part P001 on 11 March 2019 and customer C004 ordered part P002, also on 11 March 2019. Only customer C003 had order both parts P001 and P002; P001 was ordered on 18 March 2019 and P002 was ordered on 14 February 2019.
- Hence, the correct result should consist of customer C003.

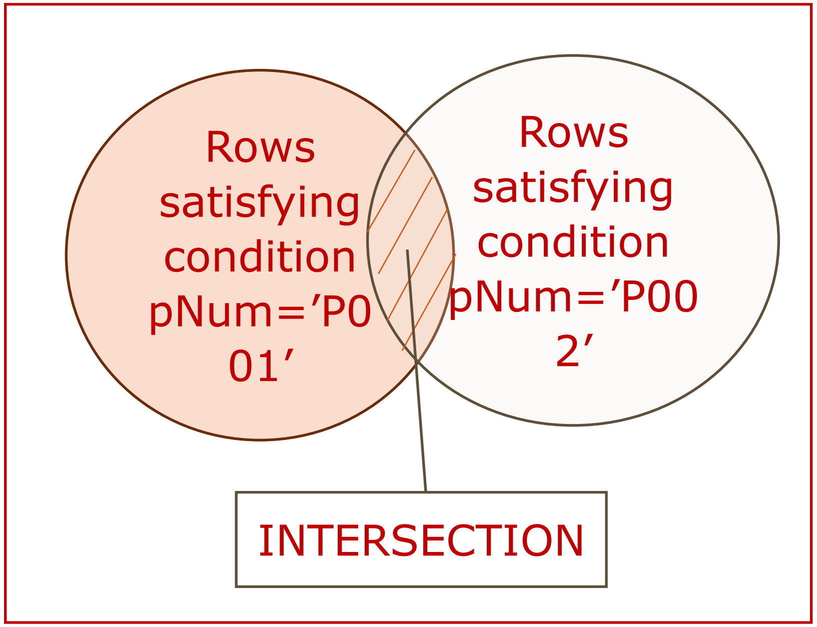
Queries with INTERSECT, UNION, and MINUS operators



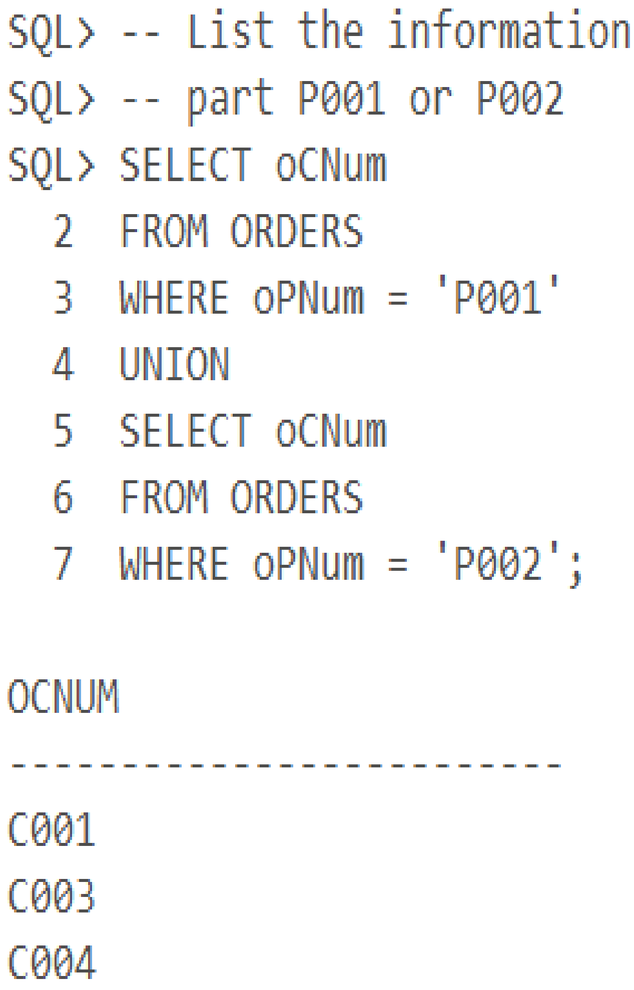
```
SQL> -- The following SELECT statment is the correct implementation
SQL> SELECT oCNum
     FROM ORDERS
     WHERE oPNum = 'P001'
     INTERSECT
     SELECT oCNum
     FROM ORDERS
     WHERE oPNum = 'P002';

OCNUM
----------
C003
```

Queries with INTERSECT, UNION, and MINUS operators

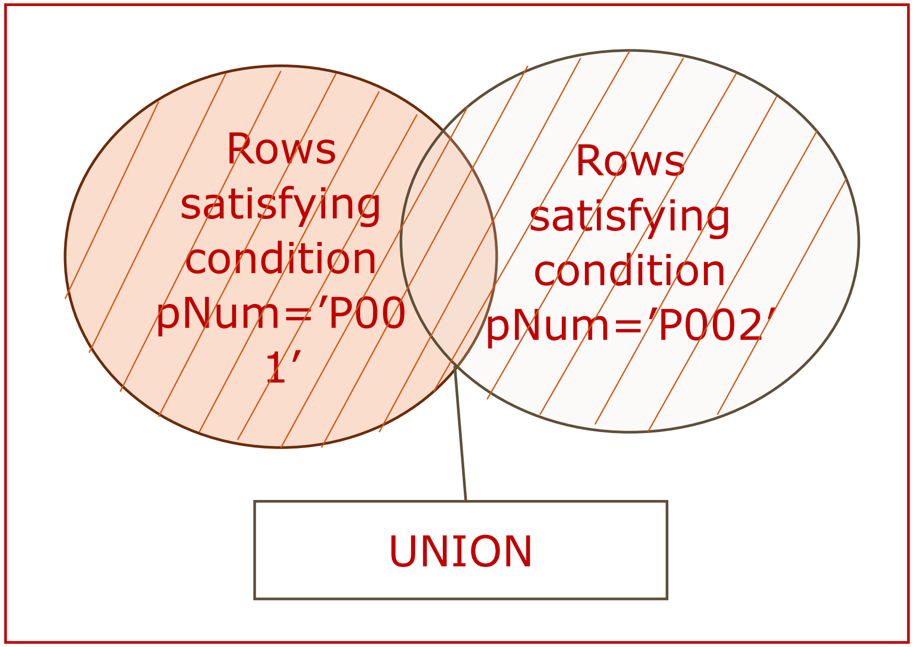
- In this implementation, two queries are used.
  - The first query is used to extract rows that satisfy the condition part number is ‘P001’ and
  - the second query is used to extract rows that satisfy the condition part number is ‘P002’.
  - These two sets of results are then intersected to obtain the correct result.


Rows satisfying condition pNum=’P001’
Rows satisfying condition pNum=’P002’
INTERSECTION

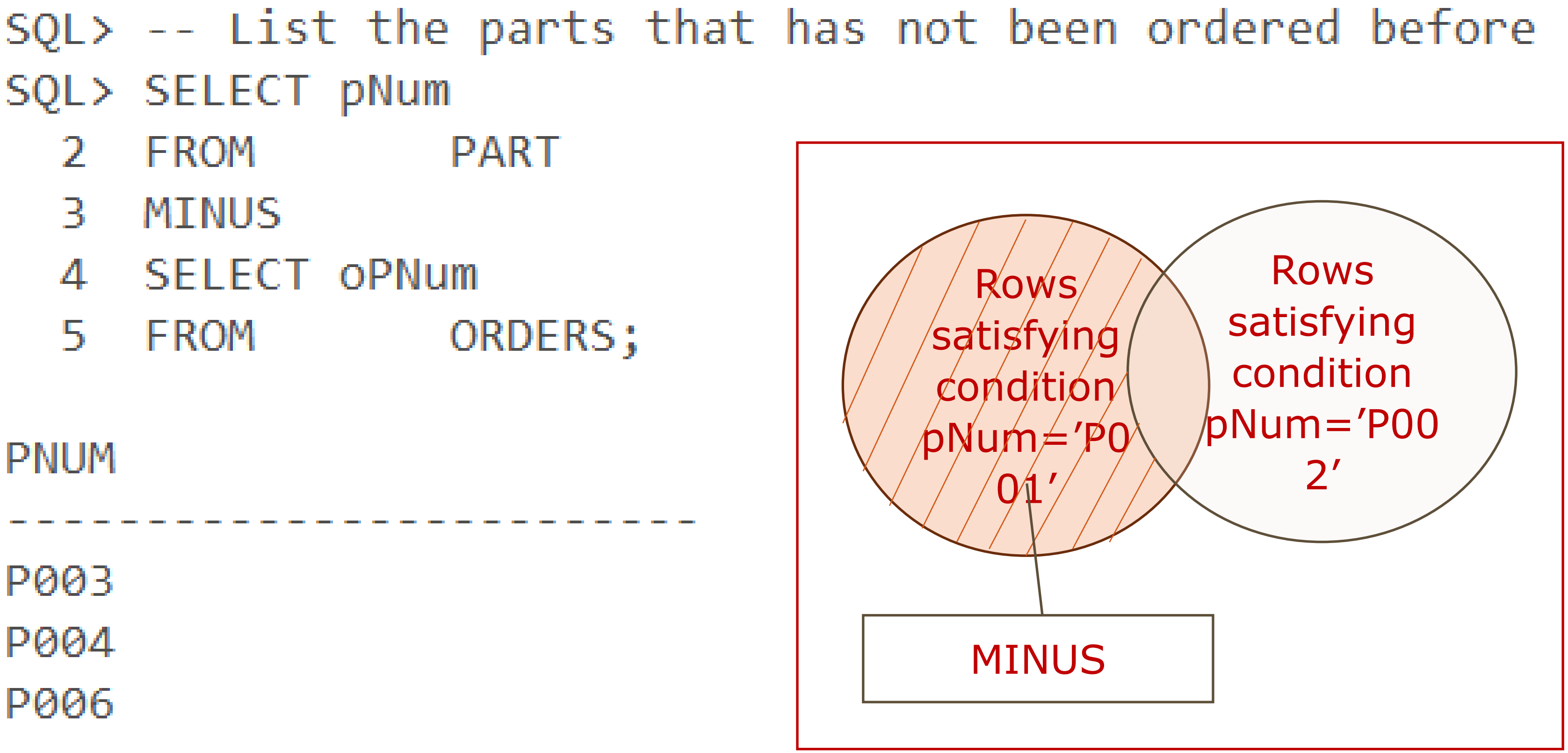
Queries with INTERSECT, UNION, and MINUS operators



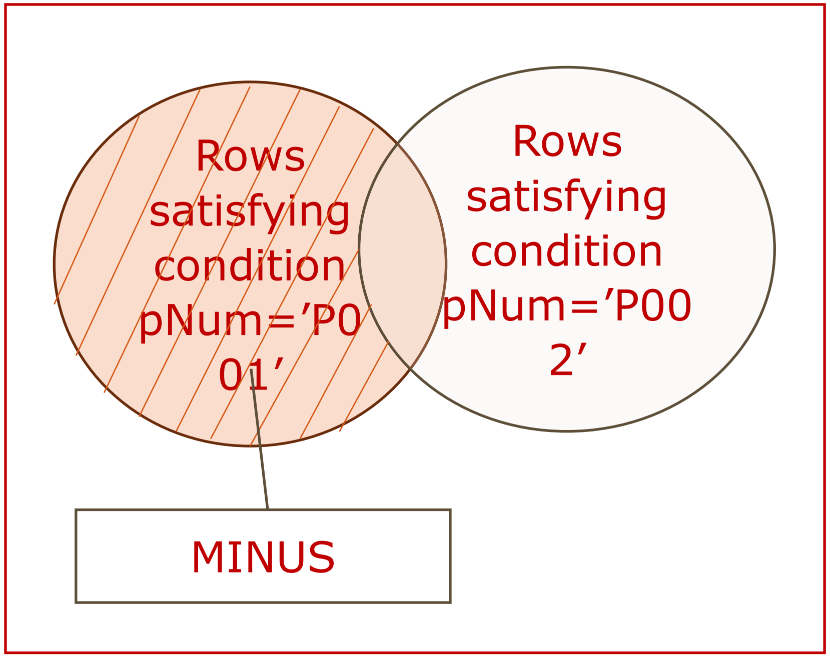
```
SQL> -- List the information of customer who have order either the
SQL> -- part P001 or P002
SQL> SELECT oCNum
  2  FROM ORDERS
  3  WHERE oPNum = 'P001'
  4  UNION
  5  SELECT oCNum
  6  FROM ORDERS
  7  WHERE oPNum = 'P002';

OCNUM
--------------------------
C001
C003
C004
```


Rows satisfying condition pNum='P001'
Rows satisfying condition pNum='P002'
UNION

Queries with INTERSECT, UNION, and MINUS operators



```
SQL> -- List the parts that has not been ordered before
SQL> SELECT pNum
  2  FROM       PART
  3  MINUS
  4  SELECT oPNum
  5  FROM       ORDERS;

PNUM
-------------------------
P003
P004
P006
```

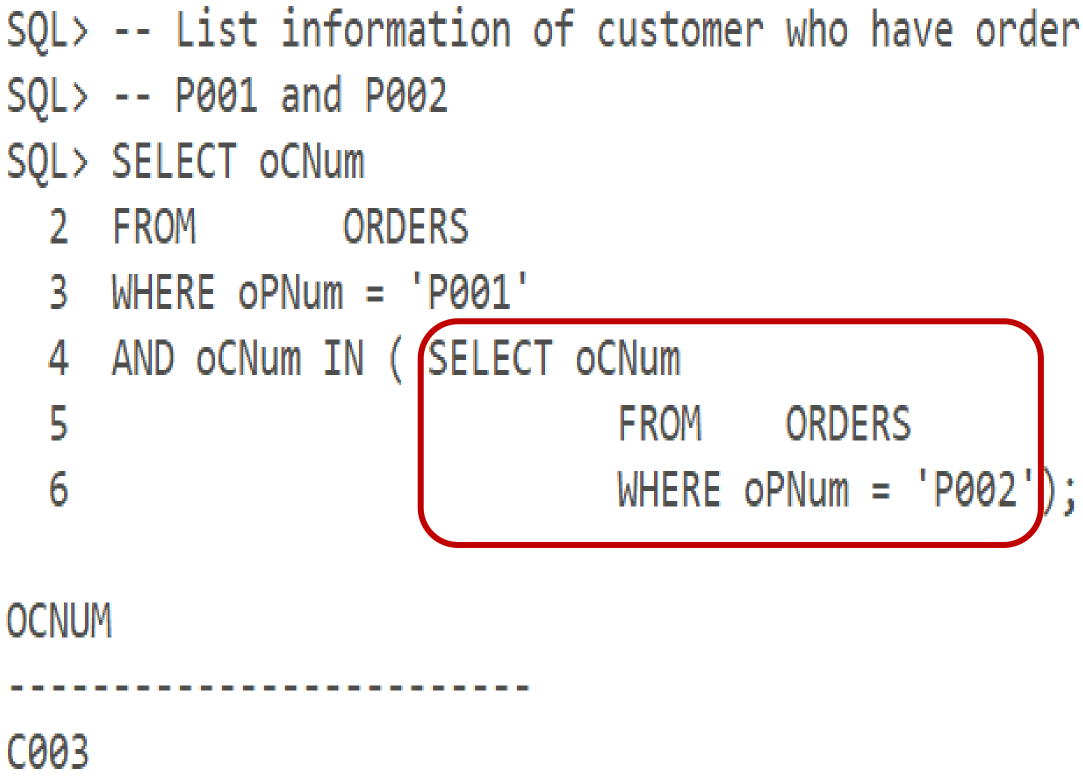

Rows satisfying condition pNum='P001'
Rows satisfying condition pNum='P002'
MINUS

Nested Queries / subqueries

- Oracle allows queries to be nested.
- Nested subquery is a query within another query.
- A subquery can be nested at the ‘WHERE’ clause, ‘FROM’ clause, as well as ‘SELECT’ clause.
- Using of subquery allows set of rows to be filtered in each query and slowly match the condition to produce the required result.
- Using of subqueries is more efficient in term of query processing as compared to implementation using JOIN operator, which produces cartesian products, to obtain the same result.

Nested Queries / subqueries

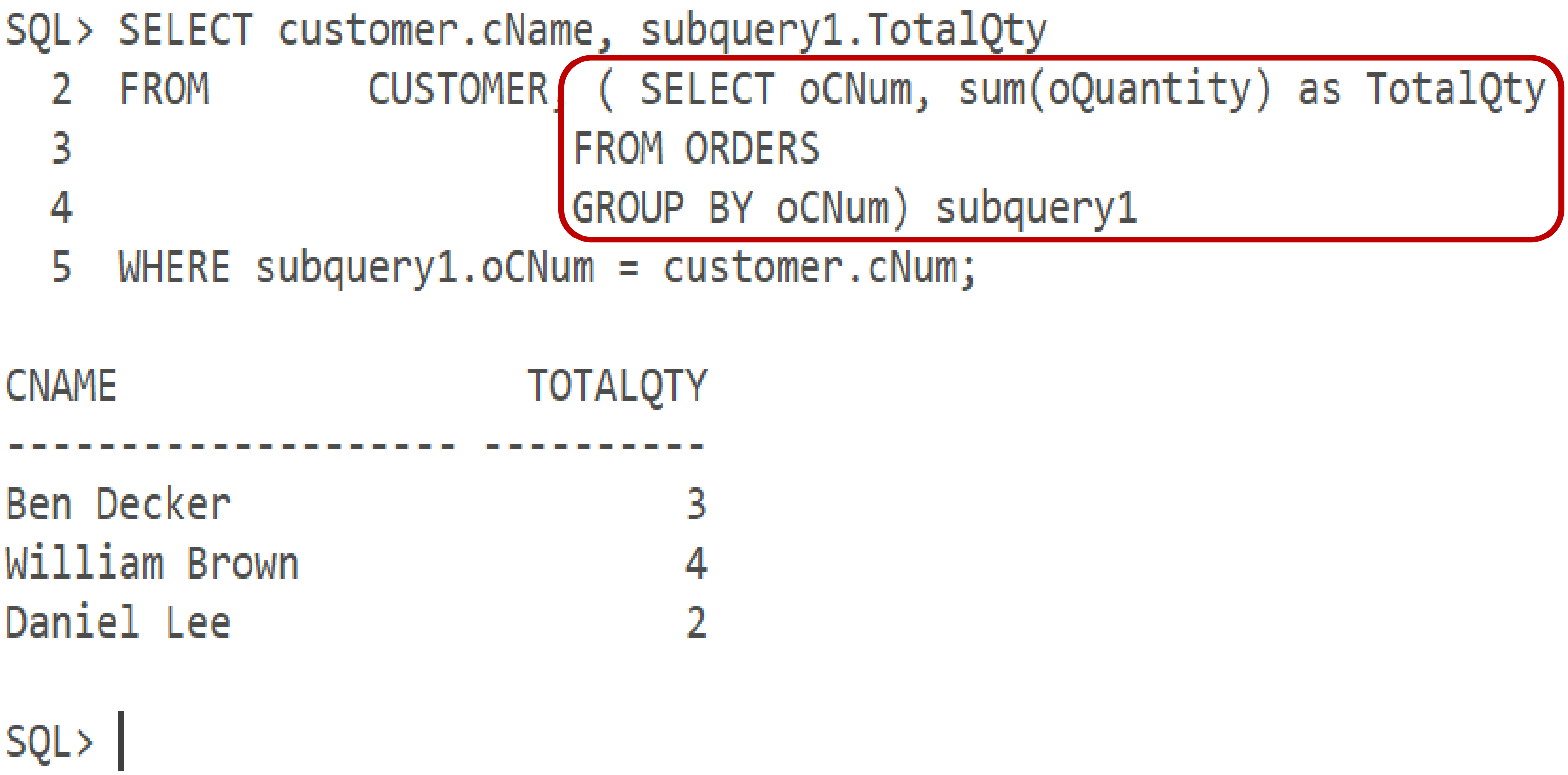
An example of subquery specified in ‘WHERE’ clause.



```
SQL> -- List information of customer who have order both the parts
SQL> -- P001 and P002
SQL> SELECT oCNum
  2  FROM       ORDERS
  3  WHERE oPNum = 'P001'
  4  AND oCNum IN ( SELECT oCNum
  5                          FROM    ORDERS
  6                          WHERE oPNum = 'P002');

OCNUM
-------------------------
C003
```

Nested Queries / subqueries

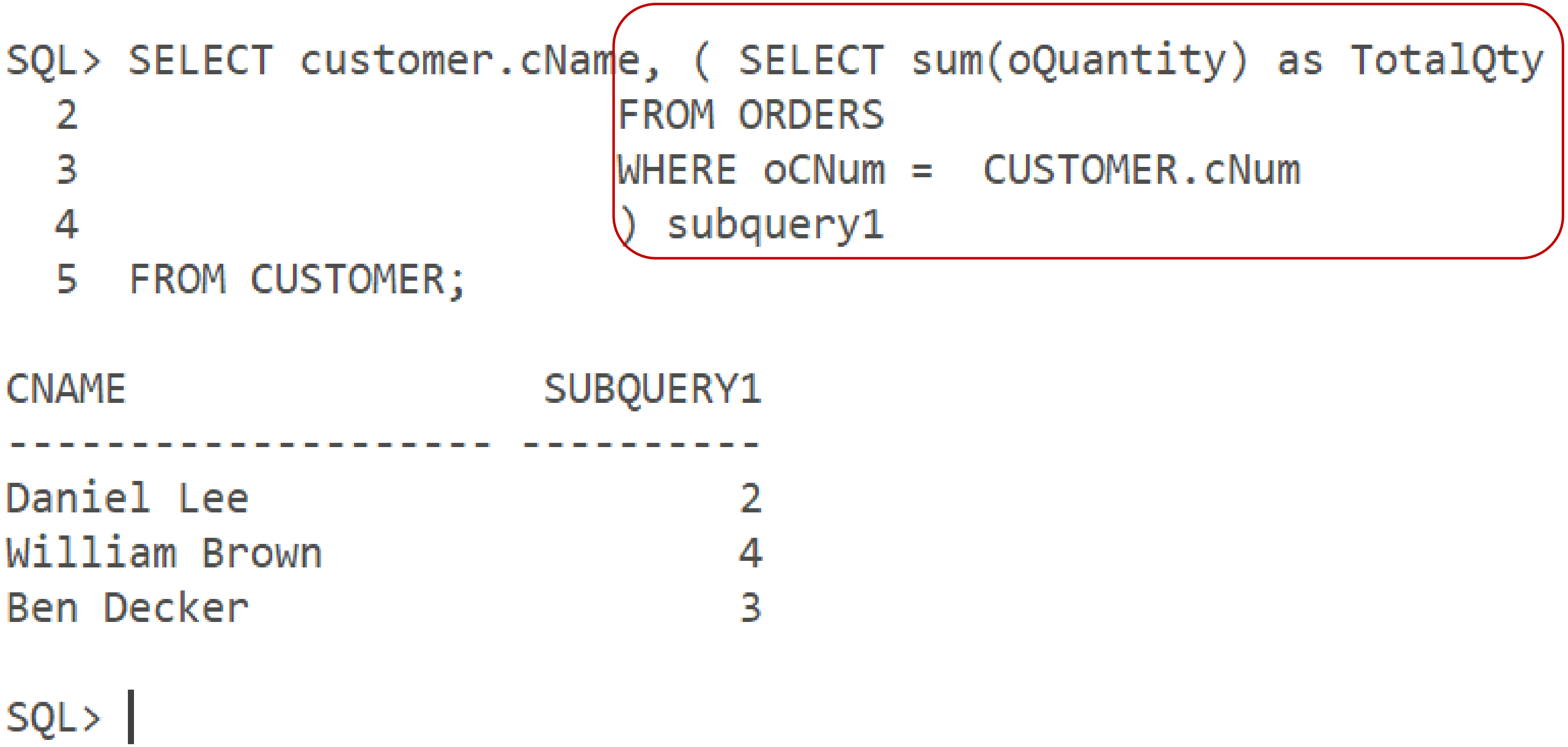
An example of subquery specified in ‘FROM’ clause.



```
SQL> SELECT customer.cName, subquery1.TotalQty
  2  FROM       CUSTOMER, ( SELECT oCNum, sum(oQuantity) as TotalQty
  3                        FROM ORDERS
  4                        GROUP BY oCNum) subquery1
  5  WHERE subquery1.oCNum = customer.cNum;

CNAME                  TOTALQTY
-------------------- ----------
Ben Decker                    3
William Brown                 4
Daniel Lee                    2

SQL> |
```

Nested Queries / subqueries

An example of subquery specified inn ‘SELECT’ clause.



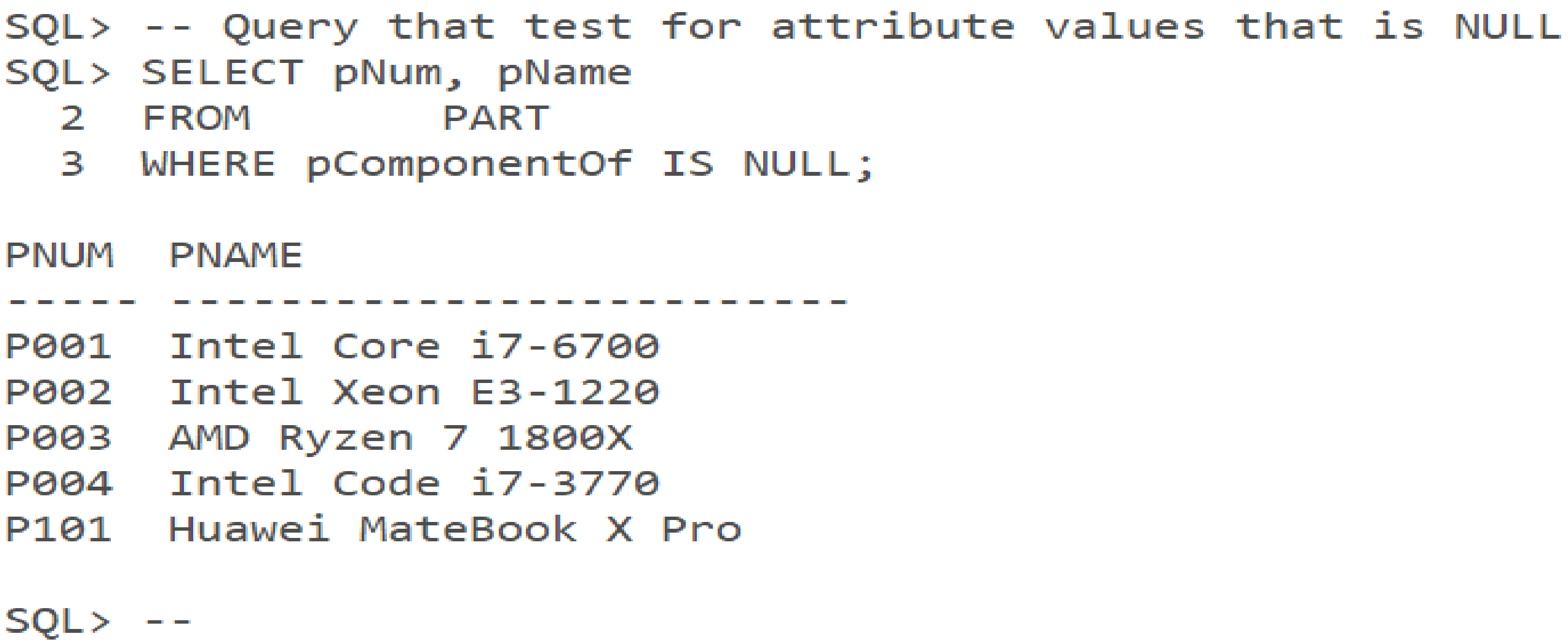
```
SQL> SELECT customer.cName, ( SELECT sum(oQuantity) as TotalQty
                              FROM ORDERS
                              WHERE oCNum =  CUSTOMER.cNum
                              ) subquery1
     FROM CUSTOMER;

CNAME                 SUBQUERY1
-------------------- ----------
Daniel Lee                    2
William Brown                 4
Ben Decker                    3

SQL>
```

Query with NULL search condition

- NULL is a special type of value in Oracle.
    - It is used for varchar2 type attribute if the attribute has no value;
    - it can be used for DATE type attribute if the attribute has no value;
    - it can also be used for numeric type attribute if the attributes has no value.
- If we have a query that needs to check or test for NULL value in the attribute, we use IS NULL or IS NOT NULL to test for the condition.
- We cannot test using = NULL or != NULL because NULL can be equated to any datatype attributes.
- The = and != operators are used to test for equal or not equal matching.

Nested Queries / subqueries

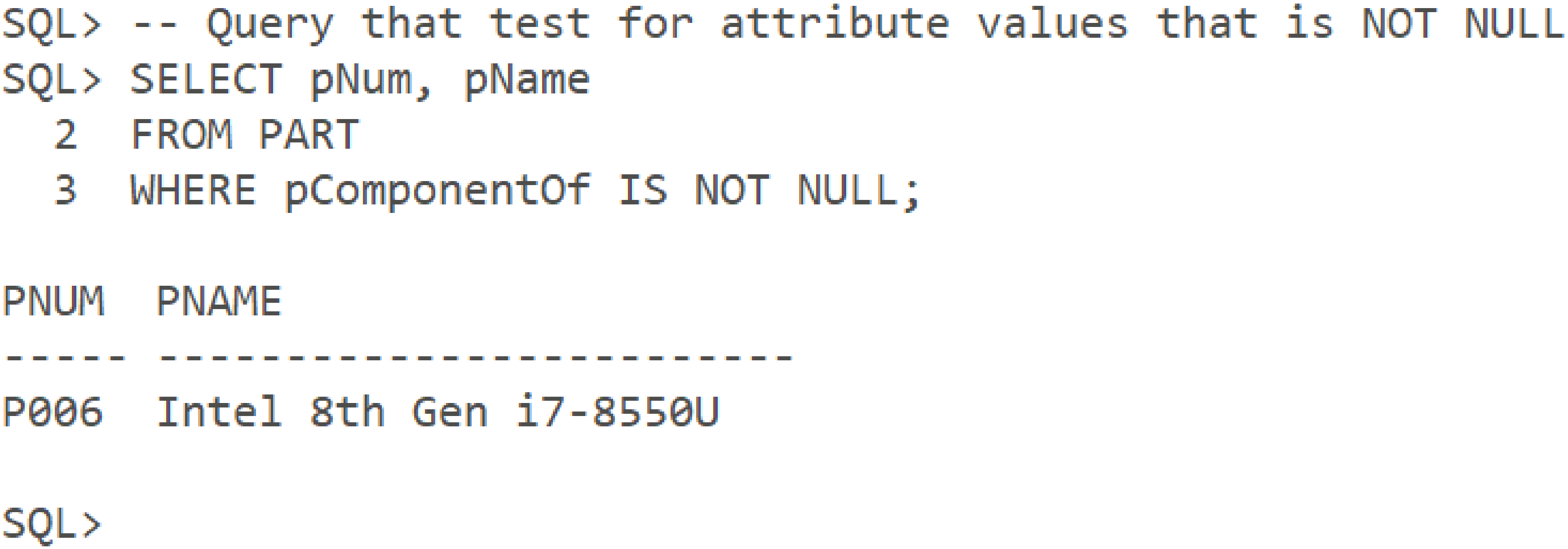
An example of query involving testing for NULL or NOT NULL attribute values.



```
SQL> -- Query that test for attribute values that is NULL
SQL> SELECT pNum, pName
  2  FROM       PART
  3  WHERE pComponentOf IS NULL;

PNUM  PNAME
----- -------------------------
P001  Intel Core i7-6700
P002  Intel Xeon E3-1220
P003  AMD Ryzen 7 1800X
P004  Intel Code i7-3770
P101  Huawei MateBook X Pro

SQL> --
```

Nested Queries / subqueries

An example of query involving testing for NULL or NOT NULL attribute values.



```
SQL> -- Query that test for attribute values that is NOT NULL
SQL> SELECT pNum, pName
  2  FROM PART
  3  WHERE pComponentOf IS NOT NULL;

PNUM  PNAME
----- -------------------------
P006  Intel 8th Gen i7-8550U

SQL>
```

Query with aggregate functions

- Aggregate functions are used to compute an aggregated value over a group of rows. The following are the five common aggregate functions used in Oracle:

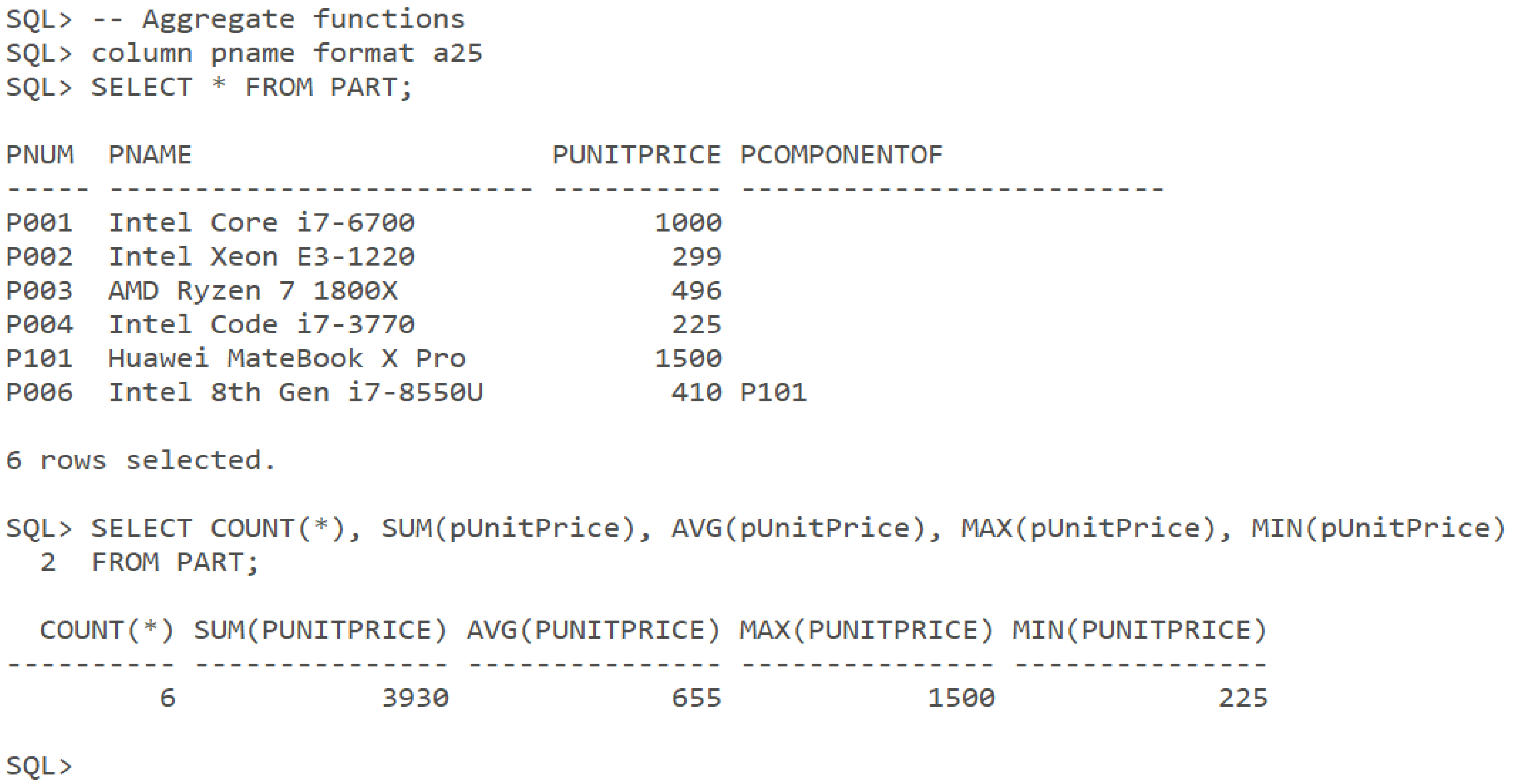
Function Name	Function
COUNT()	Return the number of occurrences in a specified column over a group of rows.
SUM()	Return the sum of the values in a specified column over a group of rows.
AVG()	Return the average of the values in a specified column over a group of rows.
MIN()	Return the smallest value in a specified column over a group of rows.
MAX()	Return the largest value in a specified column over a group of rows.

Query with aggregate functions

- Aggregate functions are often used together with the GROUP BY clause.
  - The GROUP BY clause divides the rows into groups.
  - The aggregate function calculates and returns an aggregated value for each group.
  - If the GROUP BY clause is not specified, then the aggregate functions treat the entire rows of the table or view as a group and calculate the aggregated value.
- Aggregate functions are also used in the HAVING clause to filter groups from the output based on the results of the aggregate functions.

Query with aggregate functions

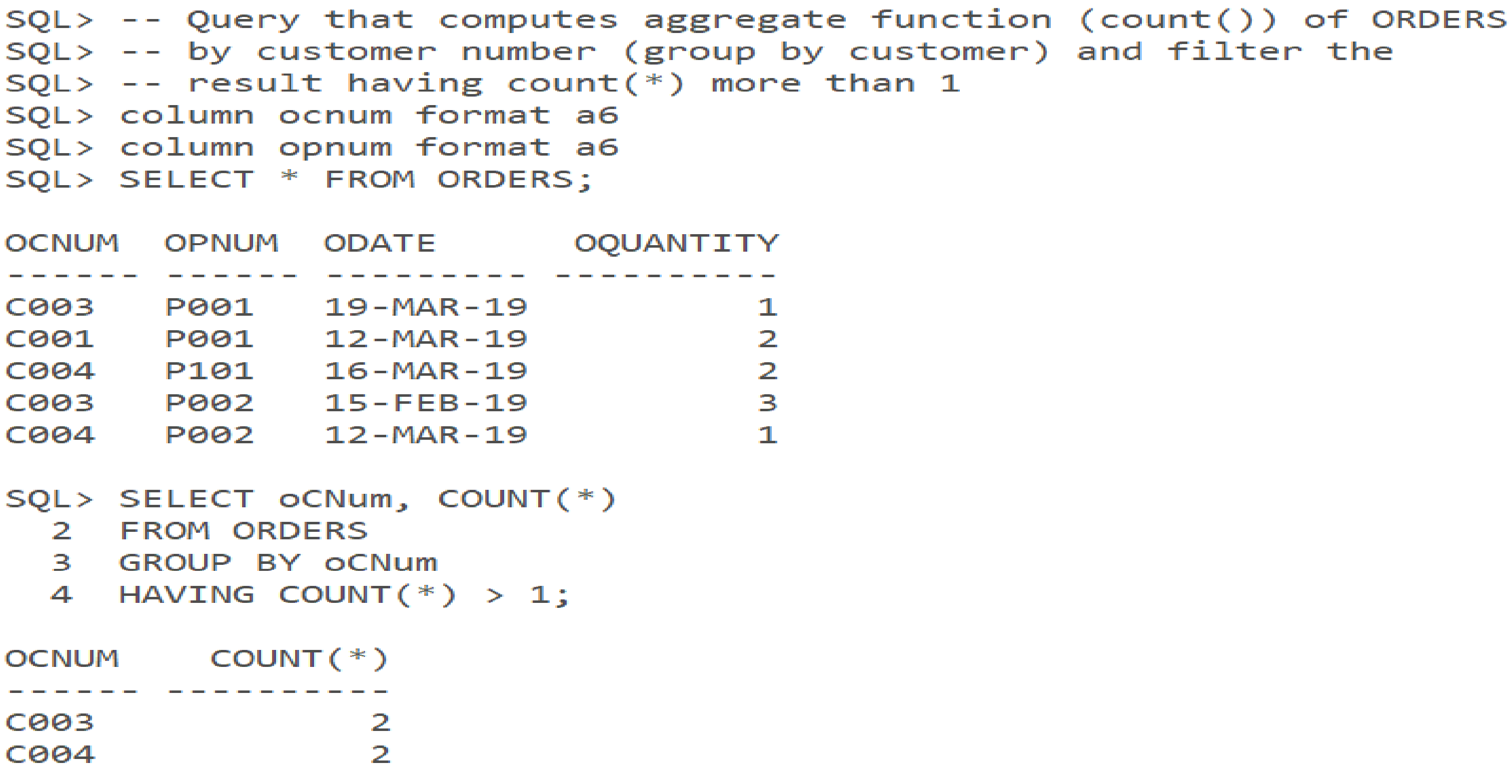
An example of various aggregate functions.



```
SQL> -- Aggregate functions
SQL> column pname format a25
SQL> SELECT * FROM PART;

PNUM  PNAME                     PUNITPRICE PCOMPONENTOF
----- ------------------------- ---------- -------------------------
P001  Intel Core i7-6700              1000
P002  Intel Xeon E3-1220               299
P003  AMD Ryzen 7 1800X                496
P004  Intel Code i7-3770               225
P101  Huawei MateBook X Pro           1500
P006  Intel 8th Gen i7-8550U           410 P101

6 rows selected.

SQL> SELECT COUNT(*), SUM(pUnitPrice), AVG(pUnitPrice), MAX(pUnitPrice), MIN(pUnitPrice)
  2  FROM PART;

  COUNT(*) SUM(PUNITPRICE) AVG(PUNITPRICE) MAX(PUNITPRICE) MIN(PUNITPRICE)
---------- --------------- --------------- --------------- ---------------
         6            3930             655            1500             225

SQL>
```

Query with aggregate functions

Query with aggregate function counting the total number of orders made by customers. The output is then filtered using HAVING clause



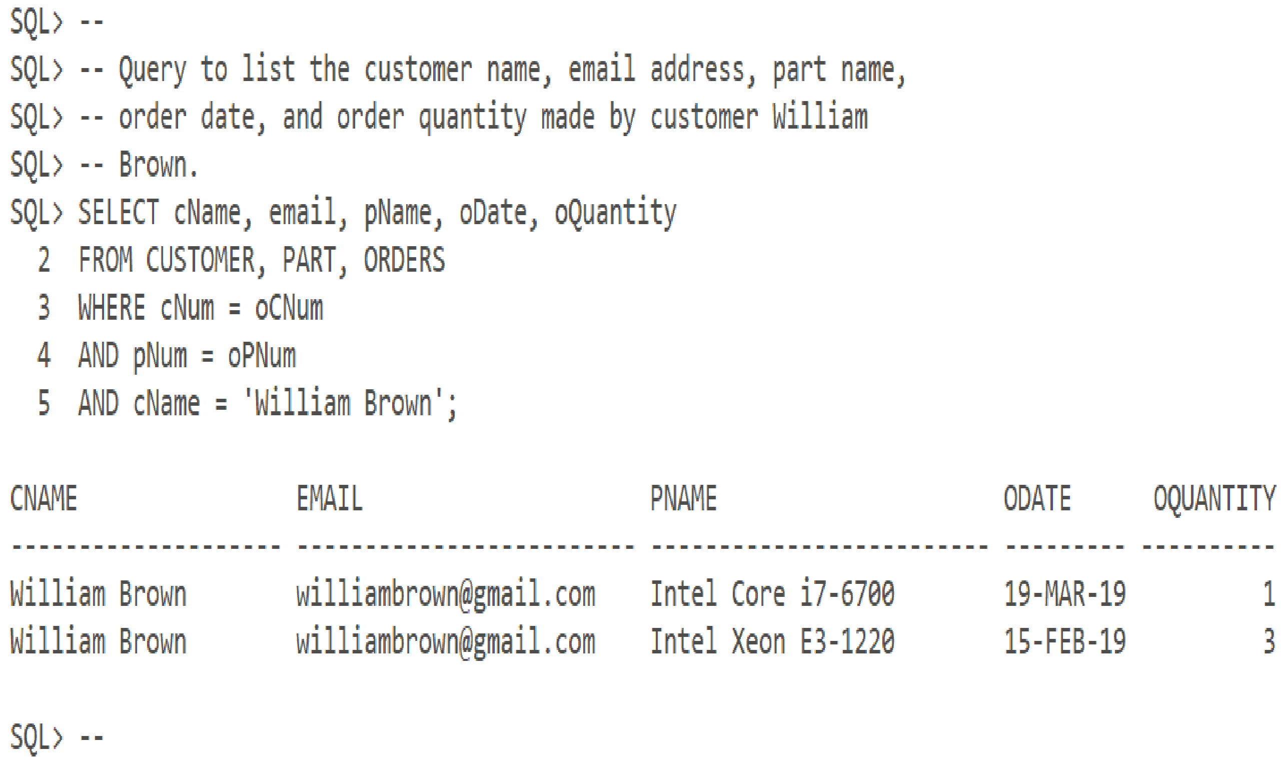
```
SQL> -- Query that computes aggregate function (count()) of ORDERS
SQL> -- by customer number (group by customer) and filter the
SQL> -- result having count(*) more than 1
SQL> column ocnum format a6
SQL> column opnum format a6
SQL> SELECT * FROM ORDERS;

OCNUM  OPNUM  ODATE      OQUANTITY
------ ------ --------- ----------
C003   P001   19-MAR-19          1
C001   P001   12-MAR-19          2
C004   P101   16-MAR-19          2
C003   P002   15-FEB-19          3
C004   P002   12-MAR-19          1

SQL> SELECT oCNum, COUNT(*)
  2  FROM ORDERS
  3  GROUP BY oCNum
  4  HAVING COUNT(*) > 1;

OCNUM    COUNT(*)
------ ----------
C003            2
C004            2
```

Query with simple JOIN operation

- When the information to be retrieve resides in different relational tables, we need to JOIN the relational tables to retrieve the required information.
- An Oracle JOIN is performed whenever two or more tables are joined in an SQL statement.
- Oracle supports both ANSI and non-ANSI JOIN syntax.

Query with simple JOIN operation (Non-ANSI)

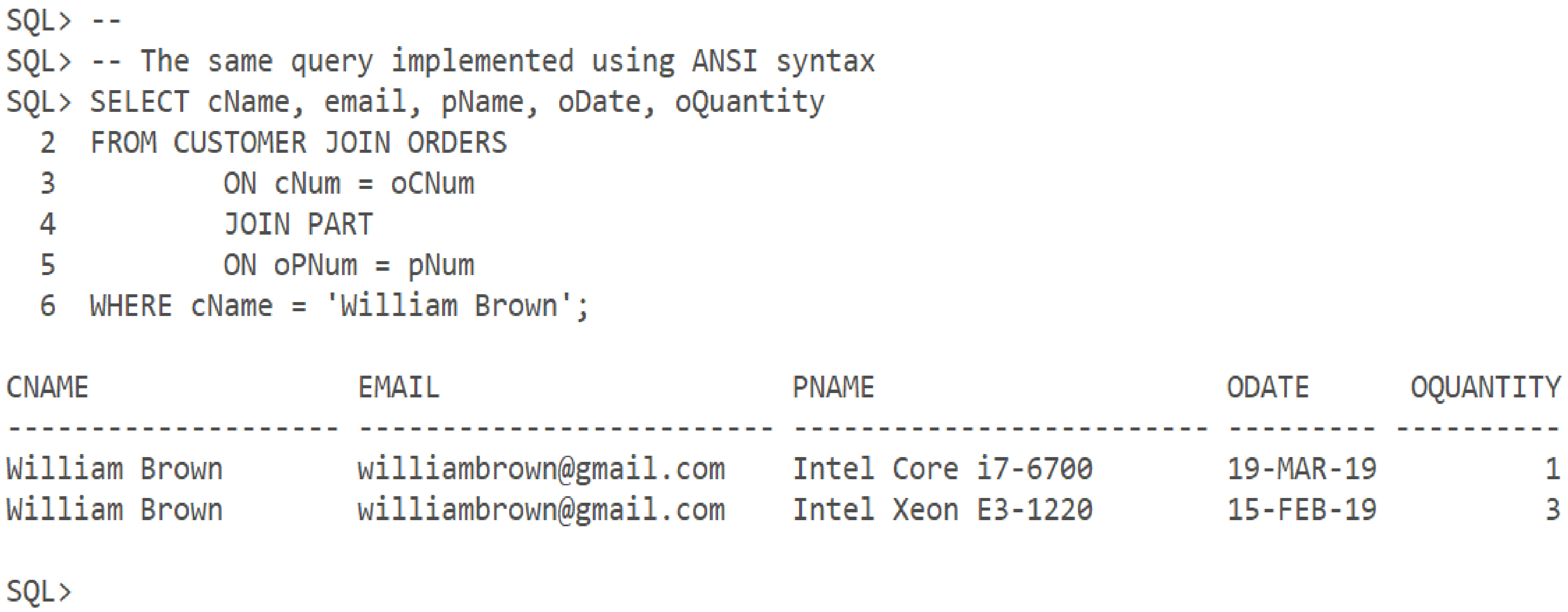
- For example, a user wants to list the customer’s name, email address, part name, quantity ordered, and order date made by the customer William Brown.



```
SQL> --
SQL> -- Query to list the customer name, email address, part name,
SQL> -- order date, and order quantity made by customer William
SQL> -- Brown.
SQL> SELECT cName, email, pName, oDate, oQuantity
  2  FROM CUSTOMER, PART, ORDERS
  3  WHERE cNum = oCNum
  4  AND pNum = oPNum
  5  AND cName = 'William Brown';

CNAME                EMAIL                     PNAME                     ODATE      OQUANTITY
-------------------- ------------------------- ------------------------- --------- ----------
William Brown        williambrown@gmail.com    Intel Core i7-6700        19-MAR-19          1
William Brown        williambrown@gmail.com    Intel Xeon E3-1220        15-FEB-19          3

SQL> --
```

Query with simple JOIN operation (Non-ANSI)

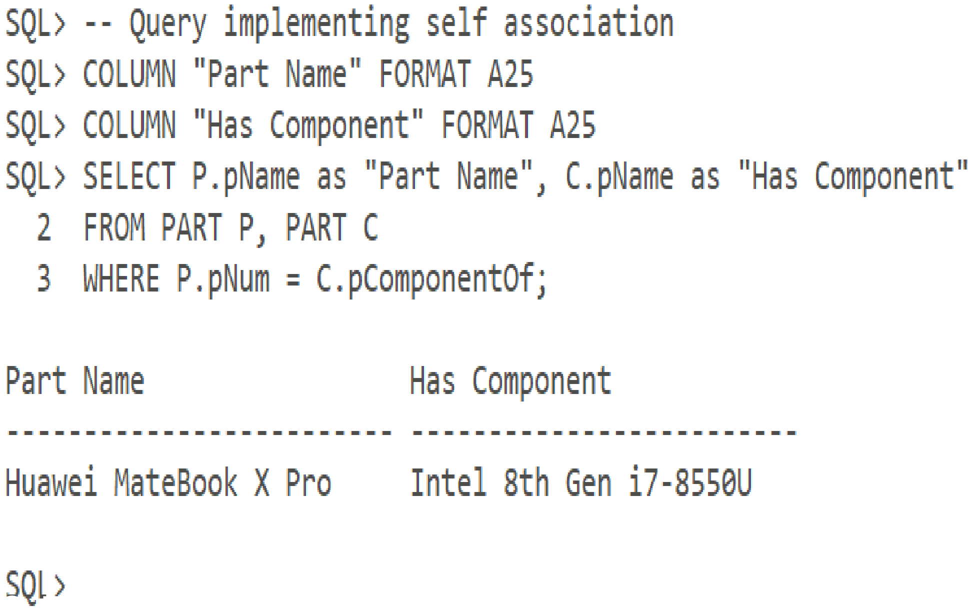
The same query implemented using ANSI syntax.



```
SQL> --
SQL> -- The same query implemented using ANSI syntax
SQL> SELECT cName, email, pName, oDate, oQuantity
  2  FROM CUSTOMER JOIN ORDERS
  3          ON cNum = oCNum
  4          JOIN PART
  5          ON oPNum = pNum
  6  WHERE cName = 'William Brown';

CNAME                EMAIL                     PNAME                     ODATE      OQUANTITY
-------------------- ------------------------- ------------------------- --------- ----------
William Brown        williambrown@gmail.com    Intel Core i7-6700        19-MAR-19          1
William Brown        williambrown@gmail.com    Intel Xeon E3-1220        15-FEB-19          3

SQL>
```

Self-join query

- Self-join involves joining a relational table to itself.
- Usually, self-join implies sort of hierarchical relationship.



```
SQL> -- Query implementing self association
SQL> COLUMN "Part Name" FORMAT A25
SQL> COLUMN "Has Component" FORMAT A25
SQL> SELECT P.pName as "Part Name", C.pName as "Has Component"
  2  FROM PART P, PART C
  3  WHERE P.pNum = C.pComponentOf;

Part Name                 Has Component
------------------------- -------------------------
Huawei MateBook X Pro     Intel 8th Gen i7-8550U

SQL>
```

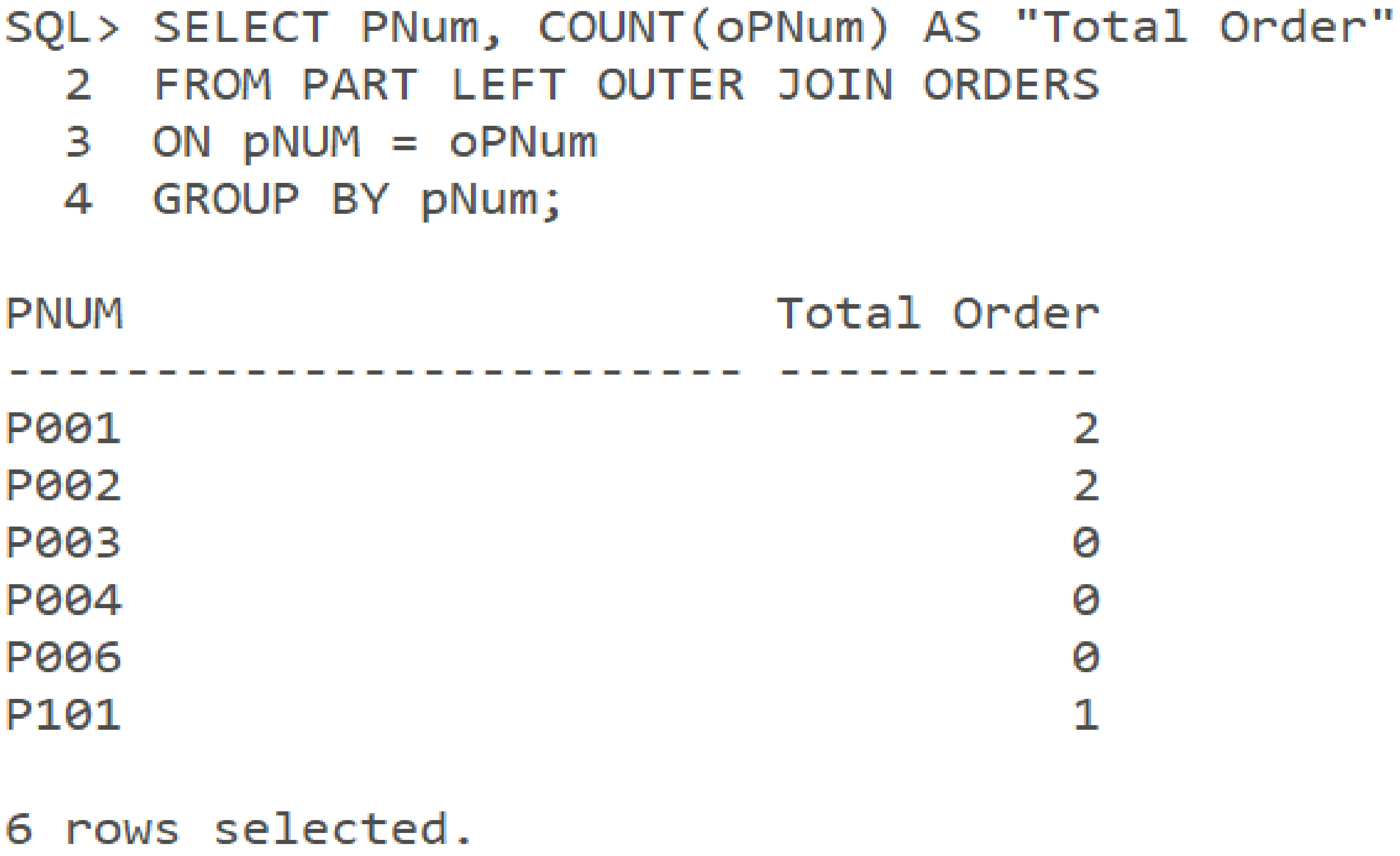
Query with OUTER JOIN operation

- So far, all the JOIN operations we have discussed return rows from one relational table that have matching condition with rows from the other relational table.
- There are occasions where we want to return all the rows from one relational table and only the rows from the other relational table that have matching condition with the rows in the first relational table.

Query with OUTER JOIN operation

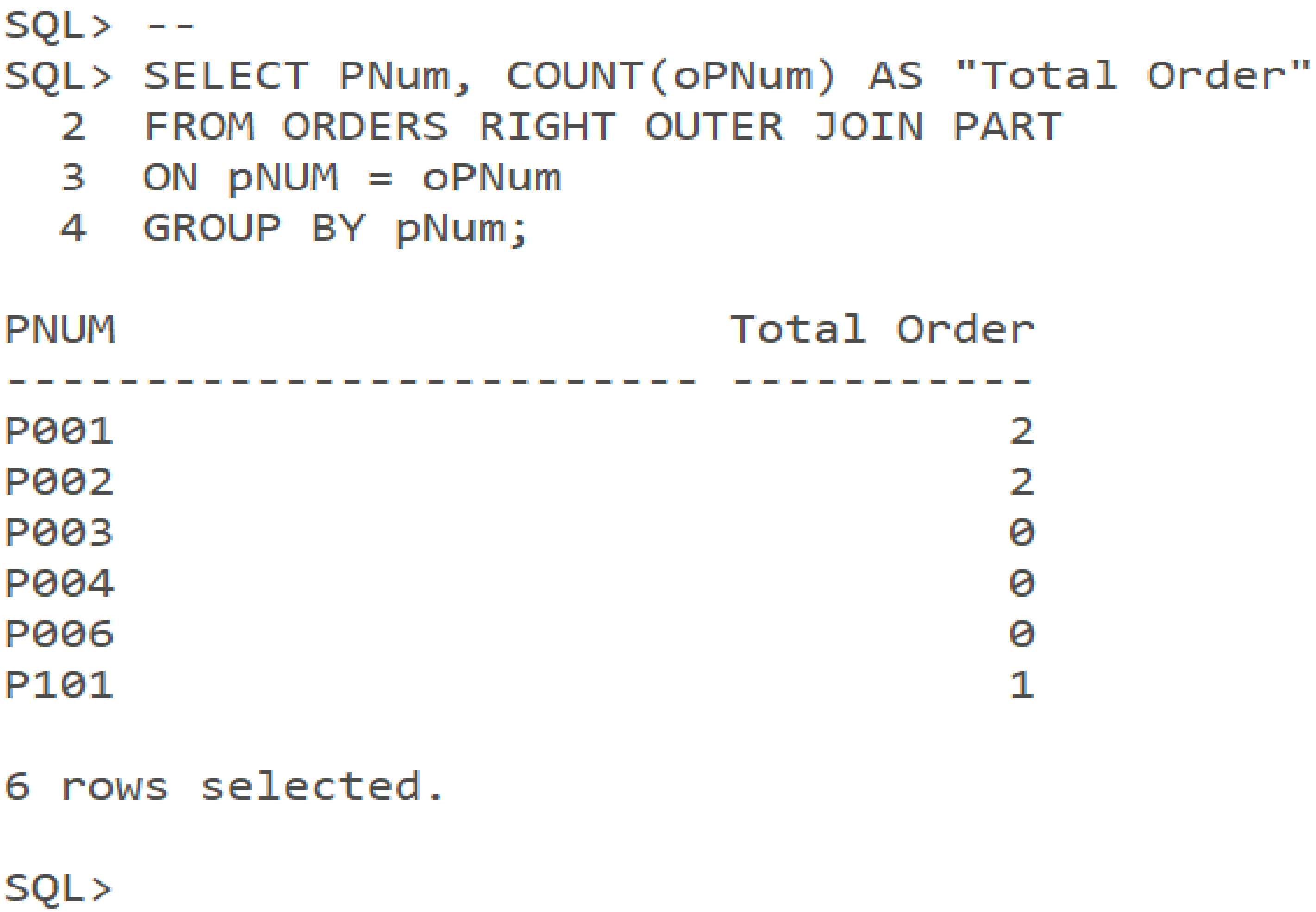
- For example, we want to see all the parts and also the number of times the parts have been ordered by customer.
- Of course, from the ORDERS table we can find the occurrences on how many times the parts have been ordered, but there is also possibility that some parts are not ordered before by any customer, and hence are not in ORDERS table.
- In such requirement, we need to list out all the parts (every row in the PART table), and the aggregate count of the parts in ORDERS table, if the parts are ordered before.

Query with OUTER JOIN operation



```
SQL> SELECT PNum, COUNT(oPNum) AS "Total Order"
  2  FROM PART LEFT OUTER JOIN ORDERS
  3  ON pNUM = oPNum
  4  GROUP BY pNum;

PNUM                      Total Order
------------------------- -----------
P001                                2
P002                                2
P003                                0
P004                                0
P006                                0
P101                                1

6 rows selected.
```

Query with OUTER JOIN operation



```
SQL> --
SQL> SELECT PNum, COUNT(oPNum) AS "Total Order"
  2  FROM ORDERS RIGHT OUTER JOIN PART
  3  ON pNUM = oPNum
  4  GROUP BY pNum;

PNUM                      Total Order
------------------------- -----------
P001                                2
P002                                2
P003                                0
P004                                0
P006                                0
P101                                1

6 rows selected.

SQL>
```