SQL Languages

DQL

DML

DDL

DCL

DTL

D what the … L

SQL

INTRODUCTION - SQL - STRUCTURED QUERY LANGUAGE

- SQL is used for CRUD:
  - *Creating databases*
  - *Adding, modifying and deleting database structures*
  - *Inserting, deleting, and modifying records in databases*
  - *Querying databases (data retrieval)*
  - *Create/Update Database Users*

- SQL functions as a standard relational database language
  - It can be used (with minor dialectical variations) with the majority of relational DBMS software tools

INTRODUCTION – SQL Command Categories

- Data Definition Language (DDL)
  - Used to create Database Objects.

- Data Manipulation Language (DML) - Used to update data in a db.
  - insert, update, delete, merge

- Data Control Language (DCL): used to control access to data stored in a database (Authorization). Examples of DCL commands include:
  - **GRANT , REVOKE**

- Transaction Control Language (TCL)
  - These are used to manage the changes made by DML statements as Trasnactions.
  - It also allows statements to be grouped together into logical transactions.
    - **commit**; **rollback** to savepoint-name; **savepoint** savepoint-name;

- Data Query Language (DQL)
  - **SELECT**

INTRODUCTION – SQL Command Categories

- Data Control Language (DCL):
  - used to control access to data stored in a database (Authorization). Examples of DCL commands include:
    - **GRANT** to allow specified users to perform specified tasks.
    - **REVOKE** to cancel previously granted or denied permissions.
  - Examples:
    - **grant** create table to username;
    - **alter** user username quota unlimited on system;
    - **grant** create any table to username;
    - **grant** drop any table to username;

INTRODUCTION – SQL Command Categories

- Transaction Control Language (TCL)
  - used to manage transactions in a db
  - These are used to manage the changes made by DML statements.
  - It also allows statements to be grouped together into logical transactions.
    - **commit**;
    - **rollback** to savepoint-name;
    - **savepoint** savepoint-name;

INTRODUCTION - DDL

- Data Definition Language (DDL)
  - Used to create and modify the structure of the database
  - Example commands:

```
CREATE
ALTER
DROP
```

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints

- And also other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

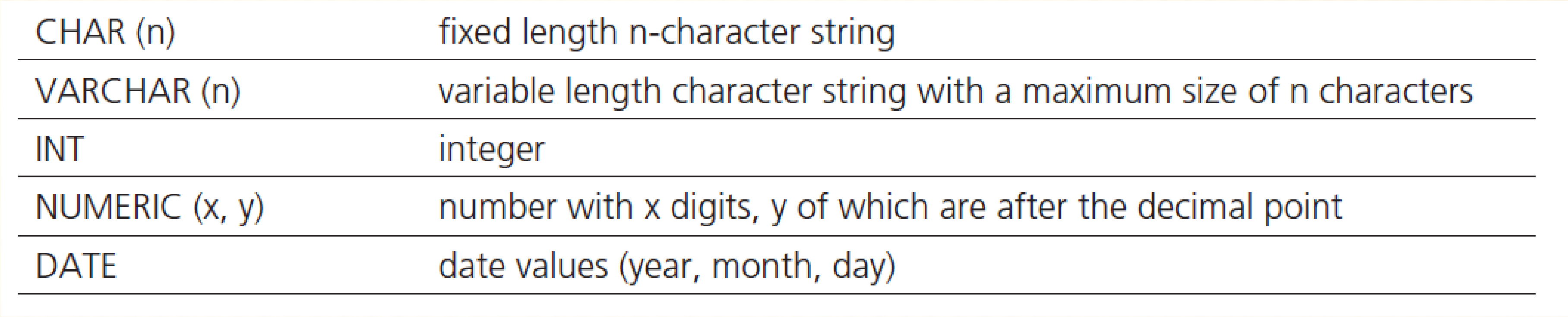
INTRODUCTION - DML

- Data Manipulation Language (DML)
  - Used to insert, modify, delete and retrieve data
  - Example commands:

```
INSERT INTO
UPDATE
DELETE
SELECT
```

INTRODUCTION

- **SQL data types**
  - Each column of each SQL created relation has a specified data type
  - Commonly used SQL data types:



CHAR (n)	fixed length n-character string
VARCHAR (n)	variable length character string with a maximum size of n characters
INT	integer
NUMERIC (x, y)	number with x digits, y of which are after the decimal point
DATE	date values (year, month, day)

INTRODUCTION - SQL

- Brief SQL syntax notes
  - **Semicolon** ";" following the end of an SQL statement, indicates the end of the SQL command
  - **SQL keywords**, as well as the table and column names used in the SQL commands, are not case sensitive
    - E.g. `SELECT` is the same as `select` or `SeLeCt`
  - An SQL statement can be written as one long sentence in one line of text
    - However, for legibility reasons SQL statements are usually broken down into multiple lines of text

DDL

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

CREATE TABLE

- CREATE TABLE
  - Used for creating and connecting relational tables

Example:

```
create table instructor (
ID              char(5),
name            varchar(20) not null,
dept_name       varchar(20),
salary          numeric(8,2))

insert into instructor  values (‘10211’, ’Smith’, ’Biology’, 66000);
insert into instructor  values (‘10211’, null, ’Biology’, 66000);
```

- **create table course (**

```
        course_id       varchar(8) primary key,
        title           varchar(50),
        dept_name        varchar(20),
        credits         numeric(2,0),
        foreign key (dept_name) references department) );
```

- Primary key declaration can be combined with attribute declaration as shown above

ALTER TABLE

- ALTER TABLE
  - Used to change the structure of the relation, once the relation is already created

Alter Statement 1:

```
ALTER TABLE vendor ADD
( vendorphonenumber CHAR(11));
```

Alter Statement 2:

```
ALTER TABLE vendor DROP
( vendorphonenumber);
```

DDL - DROP TABLE

- DROP TABLE
  - Used to remove a table from the database
  - DROP TABLE Students;

- TRUNCATE TABLE
  - **truncate table Student;**
  - Removes all tuples in Student table and reinitialize the table.
  - Different than DELETE command as delete just removes the tuples.
  - Truncate does note delete he scheme of the table.

INTEGRITY CONSTRAINTS IN CREATE TABLE

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

Example: Declare *dept_name* as the primary key for *department*
.

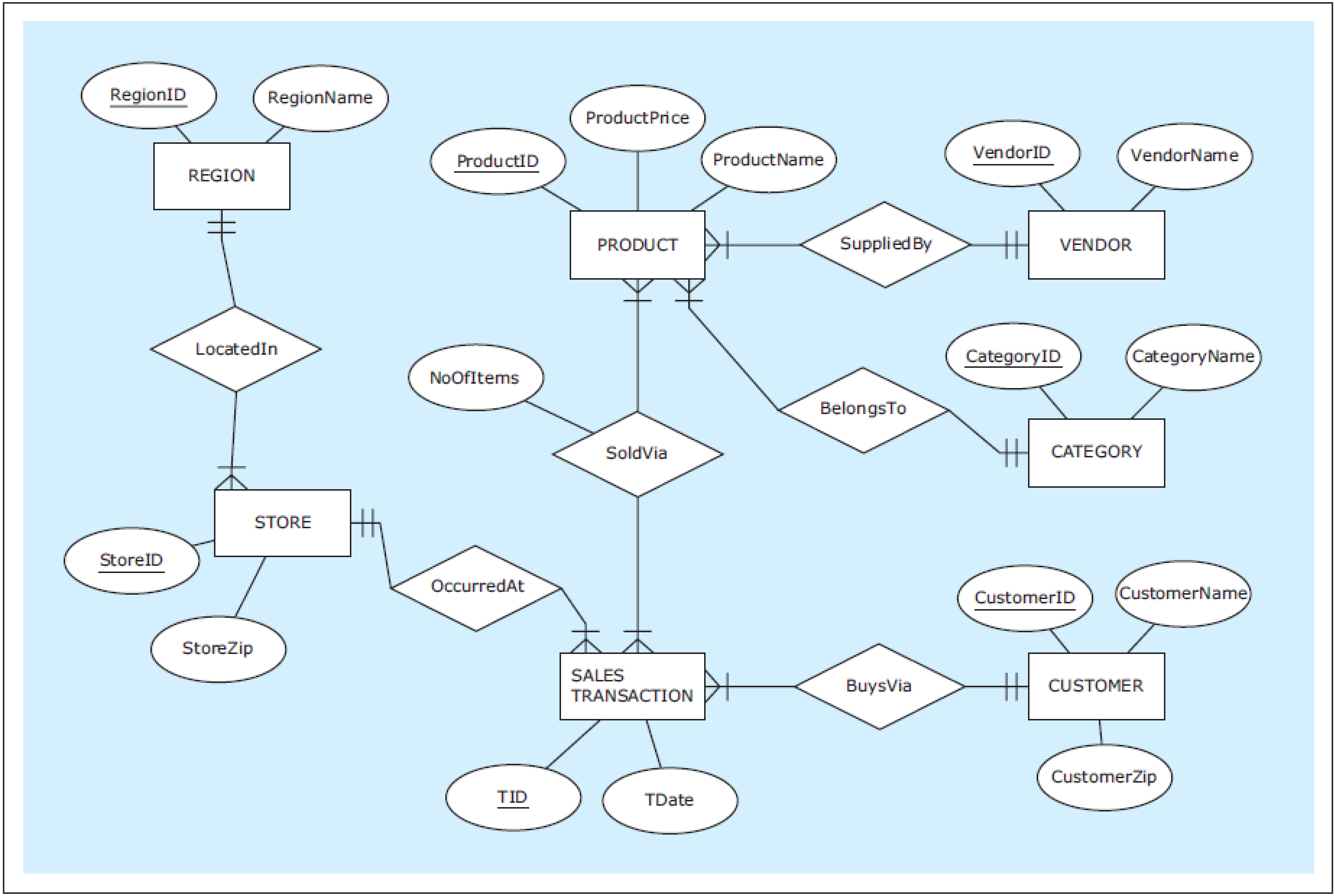
```
create table instructor (
          ID             char(5),
          name           varchar(20) not null,
          dept_name  varchar(20),
          salary          numeric(8,2),
          primary key (ID),
         foreign key (dept_name) references department)
```

INTEGRITY CONSTRAINTS IN CREATE TABLE

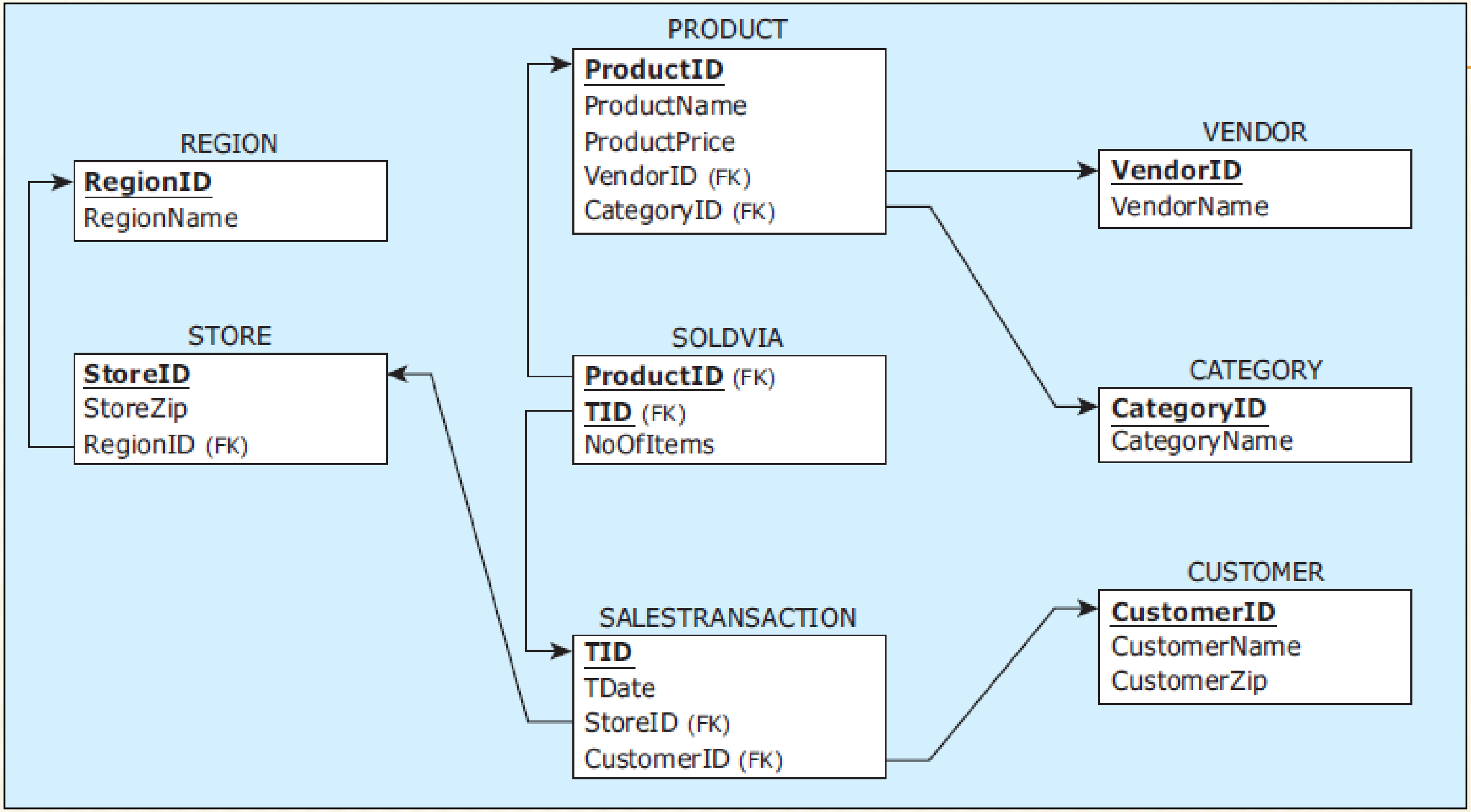
```
▪ create table student (
        ID                  varchar(5),
        name                varchar(20) not null,
        dept_name           varchar(20),
        tot_cred            numeric(3,0),
        primary key (ID),
        foreign key (dept_name) references department) );
```

```
▪ create table department (
        ID                  varchar(5),
        dept_name           varchar(20),
        building            varchar(8),
        budget              numeric(4,2),
        primary key (ID),
  );
```

ER diagram : ZAGI Retail Company Sales Department Database



Relational schema: ZAGI Retail Company Sales Department Database



CREATE TABLE statements for ZAGI Retail Company Sales Department Database

```
CREATE TABLE vendor
(        vendorid         CHAR(2)          NOT NULL,
         vendorname       VARCHAR(25)      NOT NULL,
         PRIMARY KEY (vendorid) );

CREATE TABLE category
(        categoryid       CHAR(2)          NOT NULL,
         categoryname     VARCHAR(25)      NOT NULL,
         PRIMARY KEY (categoryid) );

CREATE TABLE product
(        productid        CHAR(3)          NOT NULL,
         productname      VARCHAR(25)      NOT NULL,
         productprice     NUMERIC(7,2)     NOT NULL,
         vendorid         CHAR(2)          NOT NULL,
         categoryid       CHAR(2)          NOT NULL,
         PRIMARY KEY (productid),
         FOREIGN KEY (vendorid) REFERENCES vendor(vendorid),
         FOREIGN KEY (categoryid) REFERENCES category(categoryid) );

CREATE TABLE region
(        regionid         CHAR(1)          NOT NULL,
         regionname       VARCHAR(25)      NOT NULL,
         PRIMARY KEY (regionid) );
```

CREATE TABLE statements for ZAGI Retail Company Sales Department Database

```
CREATE TABLE store
(        storeid          VARCHAR(3)       NOT NULL,
         storezip         CHAR(5)          NOT NULL,
         regionid         CHAR(1)          NOT NULL,
         PRIMARY KEY (storeid),
         FOREIGN KEY (regionid) REFERENCES region(regionid) );

CREATE TABLE customer
(        customerid       CHAR(7)          NOT NULL,
         customername     VARCHAR(15)      NOT NULL,
         customerzip      CHAR(5)          NOT NULL,
         PRIMARY KEY (customerid) );

CREATE TABLE salestransaction
(        tid              VARCHAR(8)       NOT NULL,
         customerid       CHAR(7)          NOT NULL,
         storeid          VARCHAR(3)       NOT NULL,
         tdate            DATE             NOT NULL,
         PRIMARY KEY (tid),
         FOREIGN KEY (customerid) REFERENCES customer(customerid),
         FOREIGN KEY (storeid) REFERENCES store(storeid) );

CREATE TABLE soldvia
(        productid        CHAR(3)          NOT NULL,
         tid              VARCHAR(8)       NOT NULL,
         noofitems        INT              NOT NULL,
         PRIMARY KEY (productid, tid),
         FOREIGN KEY (productid) REFERENCES product(productid),
         FOREIGN KEY (tid) REFERENCES salestransaction(tid) );
```

DROP TABLE statements for ZAGI Retail Company Sales Department Database

INVALID SEQUENCE

```
DROP TABLE region;
DROP TABLE store;
DROP TABLE salestransaction;
DROP TABLE product;
DROP TABLE vendor;
DROP TABLE category;
DROP TABLE customer;
DROP TABLE soldvia;
```

VALID SEQUENCE

```
DROP TABLE soldvia;
DROP TABLE salestransaction;
DROP TABLE store;
DROP TABLE product;
DROP TABLE vendor;
DROP TABLE region;
DROP TABLE category;
DROP TABLE customer;
```

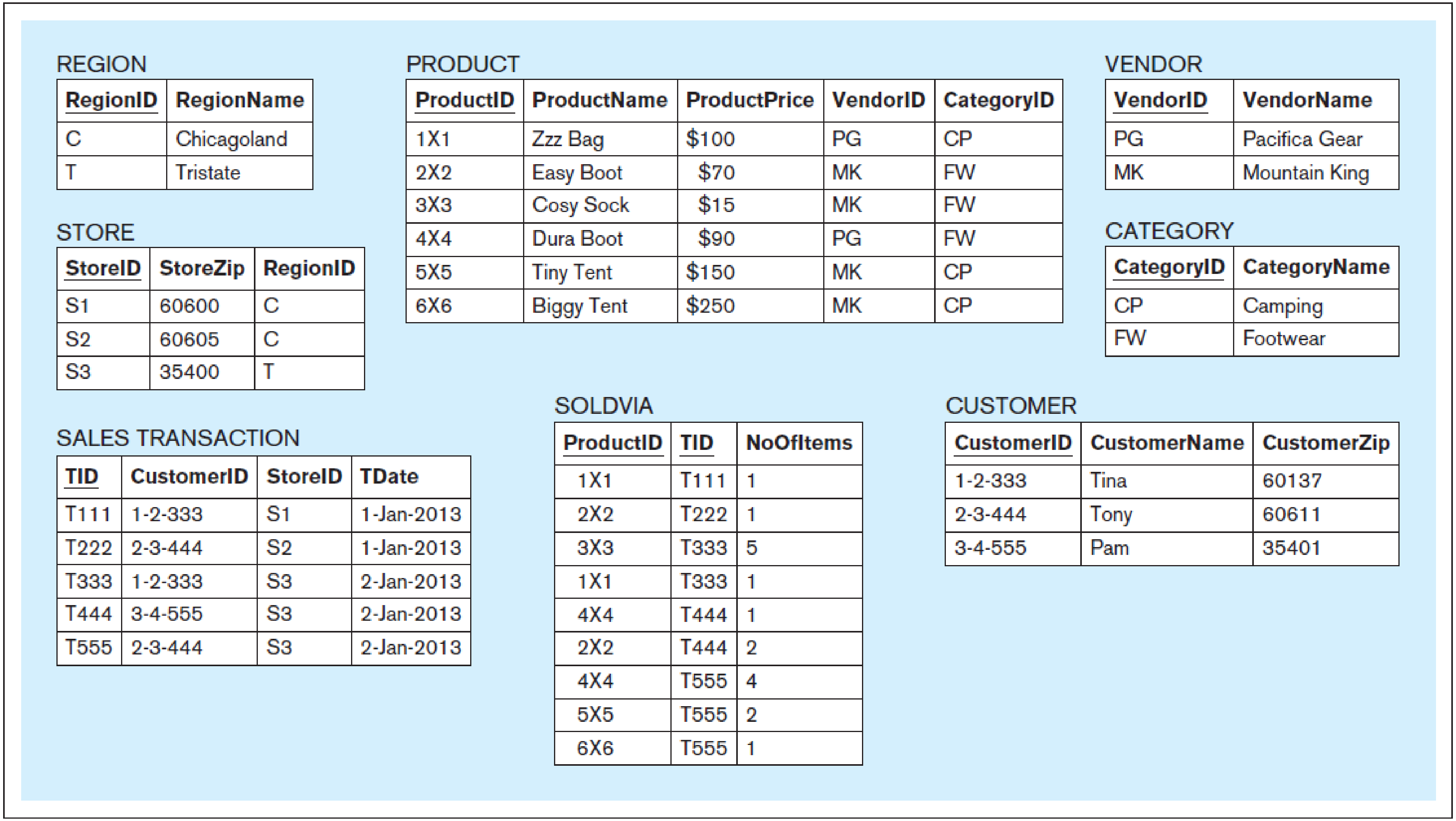
INSERT INTO

- INSERT INTO
  - Used to populate the created relations with data
  - OMAR ONLY - GOTO MYSQL AND SHOW DATABASE COMMANDS

SQL-InsertInto.sql

SQL-CreateTable.sql

Data records: ZAGI Retail Company Sales Department Database



REGION

RegionID	RegionName
C	Chicagoland
T	Tristate

STORE

StoreID	StoreZip	RegionID
S1	60600	C
S2	60605	C
S3	35400	T

SALES TRANSACTION

TID	CustomerID	StoreID	TDate
T111	1-2-333	S1	1-Jan-2013
T222	2-3-444	S2	1-Jan-2013
T333	1-2-333	S3	2-Jan-2013
T444	3-4-555	S3	2-Jan-2013
T555	2-3-444	S3	2-Jan-2013

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	$100	PG	CP
2X2	Easy Boot	$70	MK	FW
3X3	Cosy Sock	$15	MK	FW
4X4	Dura Boot	$90	PG	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

SOLDVIA

ProductID	TID	NoOfItems
1X1	T111	1
2X2	T222	1
3X3	T333	5
1X1	T333	1
4X4	T444	1
2X2	T444	2
4X4	T555	4
5X5	T555	2
6X6	T555	1

VENDOR

VendorID	VendorName
PG	Pacifica Gear
MK	Mountain King

CATEGORY

CategoryID	CategoryName
CP	Camping
FW	Footwear

CUSTOMER

CustomerID	CustomerName	CustomerZip
1-2-333	Tina	60137
2-3-444	Tony	60611
3-4-555	Pam	35401

INSERT INTO statements for ZAGI Retail Company Sales Department Database

```
INSERT INTO vendor VALUES ('PG','Pacifica Gear');
INSERT INTO vendor VALUES ('MK','Mountain King');

INSERT INTO category VALUES ('CP','Camping');
INSERT INTO category VALUES ('FW','Footwear');

INSERT INTO product VALUES ('1X1','Zzz Bag',100,'PG','CP');
INSERT INTO product VALUES ('2X2','Easy Boot',70,'MK','FW');
INSERT INTO product VALUES ('3X3','Cosy Sock',15,'MK','FW');
INSERT INTO product VALUES ('4X4','Dura Boot',90,'PG','FW');
INSERT INTO product VALUES ('5X5','Tiny Tent',150,'MK','CP');
INSERT INTO product VALUES ('6X6','Biggy Tent',250,'MK','CP');

INSERT INTO region VALUES ('C','Chicagoland');
INSERT INTO region VALUES ('T','Tristate');

INSERT INTO store VALUES ('S1','60600','C');
INSERT INTO store VALUES ('S2','60605','C');
INSERT INTO store VALUES ('S3','35400','T');

INSERT INTO customer VALUES ('1-2-333','Tina','60137');
INSERT INTO customer VALUES ('2-3-444','Tony','60611');
INSERT INTO customer VALUES ('3-4-555','Pam','35401');
```

INSERT INTO statements for ZAGI Retail Company Sales Department Database

```
INSERT INTO salestransaction VALUES ('T111','1-2-333','S1','01/Jan/2013');
INSERT INTO salestransaction VALUES ('T222','2-3-444','S2','01/Jan/2013');
INSERT INTO salestransaction VALUES ('T333','1-2-333','S3','02/Jan/2013');
INSERT INTO salestransaction VALUES ('T444','3-4-555','S3','02/Jan/2013');
INSERT INTO salestransaction VALUES ('T555','2-3-444','S3','02/Jan/2013');

INSERT INTO soldvia VALUES ('1X1','T111',1);
INSERT INTO soldvia VALUES ('2X2','T222',1);
INSERT INTO soldvia VALUES ('3X3','T333',5);
INSERT INTO soldvia VALUES ('1X1','T333',1);
INSERT INTO soldvia VALUES ('4X4','T444',1);
INSERT INTO soldvia VALUES ('2X2','T444',2);
INSERT INTO soldvia VALUES ('4X4','T555',4);
INSERT INTO soldvia VALUES ('5X5','T555',2);
INSERT INTO soldvia VALUES ('6X6','T555',1);
```

DQL

SELECT

SELECT

- **SELECT**
  - Used for the retrieval of data from the database relations
  - Most commonly issued SQL statement
  - Basic form:

```
SELECT    <columns>
FROM      <table>
```

A typical SQL query has the form:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

A_i represents an attribute
R_i represents a relation
P is a predicate.

The result of an SQL query is a relation.

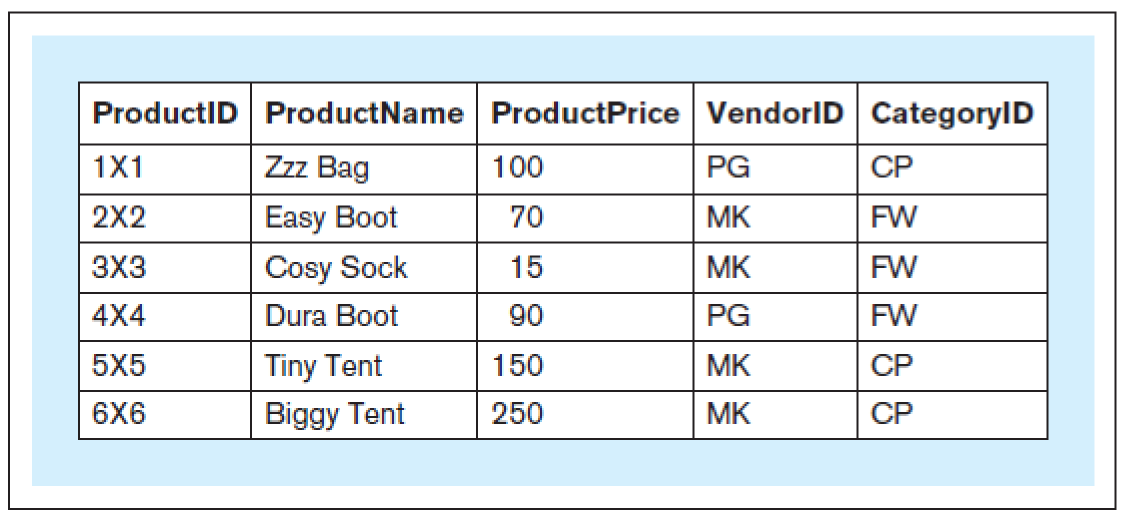
SELECT

Query 1 text: *Retrieve the entire contents of the relation PRODUCT*

Query 1:

```
SELECT productid, productname, productprice,
       vendorid, categoryid
FROM   product;
```

Query 1 result:



ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	100	PG	CP
2X2	Easy Boot	70	MK	FW
3X3	Cosy Sock	15	MK	FW
4X4	Dura Boot	90	PG	FW
5X5	Tiny Tent	150	MK	CP
6X6	Biggy Tent	250	MK	CP

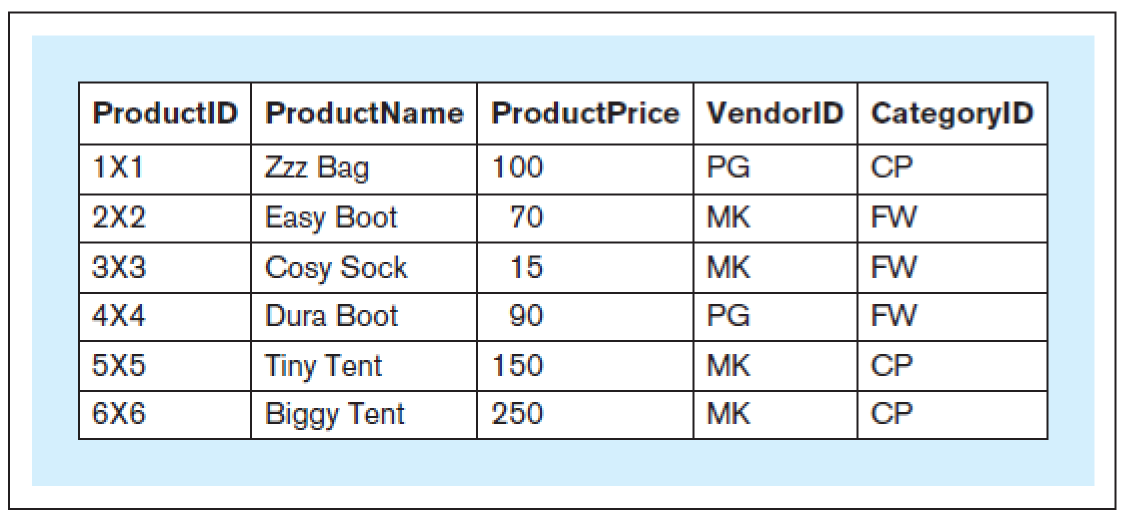
SELECT

Query 1 text: *Retrieve the entire contents of the relation PRODUCT*

Query 1a:

```
SELECT      *
FROM        product;
```

Query 1a result:



ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	100	PG	CP
2X2	Easy Boot	70	MK	FW
3X3	Cosy Sock	15	MK	FW
4X4	Dura Boot	90	PG	FW
5X5	Tiny Tent	150	MK	CP
6X6	Biggy Tent	250	MK	CP

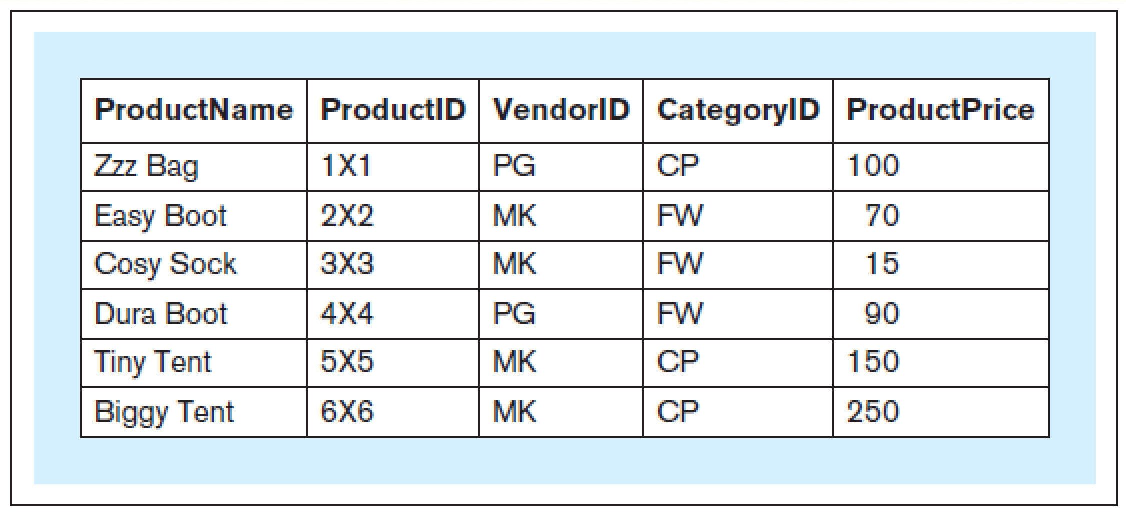
SELECT

Query 2 text: *Retrieve the entire contents of the relation PRODUCT and show the columns in the following order: ProductName, ProductID, VendorID, CategoryID, ProductPrice*

Query 2:

```
SELECT      productname, productid, vendorid,
            categoryid, productprice
FROM        product;
```

Query 2 result:



ProductName	ProductID	VendorID	CategoryID	ProductPrice
Zzz Bag	1X1	PG	CP	100
Easy Boot	2X2	MK	FW	70
Cosy Sock	3X3	MK	FW	15
Dura Boot	4X4	PG	FW	90
Tiny Tent	5X5	MK	CP	150
Biggy Tent	6X6	MK	CP	250

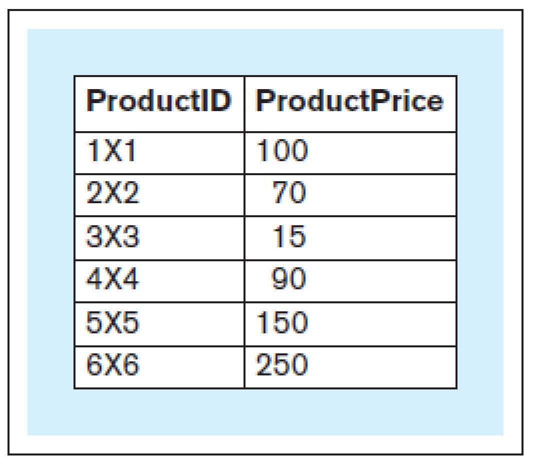
SELECT

Query 3 text: *For the relation PRODUCT, show the columns ProductID and ProductPrice*

Query 3:

```
SELECT        productid, productprice
FROM          product;
```

Query 3 result:



ProductID	ProductPrice
1X1	100
2X2	70
3X3	15
4X4	90
5X5	150
6X6	250

SELECT

- SELECT
  - In addition to displaying columns, the SELECT clause can be used to display derived attributes (calculated columns) represented as expressions
  - SELECT statement can be structured as follows:

```
SELECT     <columns, expressions>
FROM       <table>
```

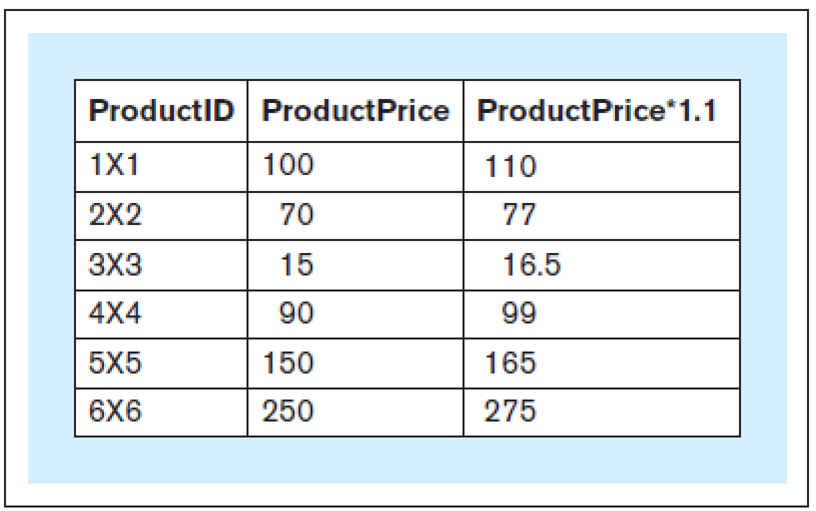
SELECT

Query 3a text: *For the relation PRODUCT, show the columns ProductID and ProductPrice and a column showing ProductPrice increased by 10%*

Query 3a:

```
SELECT   productid, productprice, productprice * 1.1
FROM     product;
```

Query 3a result:



ProductID	ProductPrice	ProductPrice*1.1
1X1	100	110
2X2	70	77
3X3	15	16.5
4X4	90	99
5X5	150	165
6X6	250	275

SELECT

- SELECT
  - The SELECT FROM statement can contain other optional keywords, such as WHERE, GROUP BY, HAVING, and ORDER BY, appearing in this order: :

```
SELECT <columns, expressions>
FROM <tables>
WHERE <row selection condition>
GROUP BY <grouping columns>
HAVING <group selection condition>
ORDER BY <sorting columns, expressions>
```

WHERE

- WHERE
  - WHERE condition determines which rows should be retrieved and consequently which rows should not be retrieved
  - The logical condition determining which records to retrieve can use one of the following logical comparison operators:

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
<>	Not equal to (alternative notation)

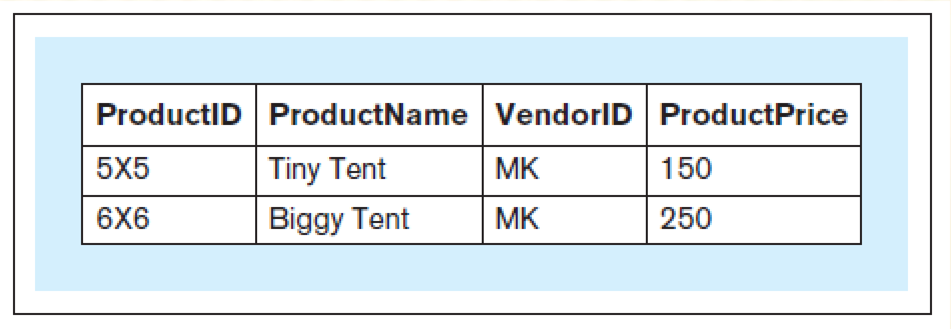
WHERE

Query 4 text: *Retrieve the product ID, product name, vendor ID, and product price for each product whose price is above $100*

Query 4:

```
SELECT        productid, productname, vendorid,
              productprice
FROM          product
WHERE         productprice > 100;
```

Query 4 result:



ProductID	ProductName	VendorID	ProductPrice
5X5	Tiny Tent	MK	150
6X6	Biggy Tent	MK	250

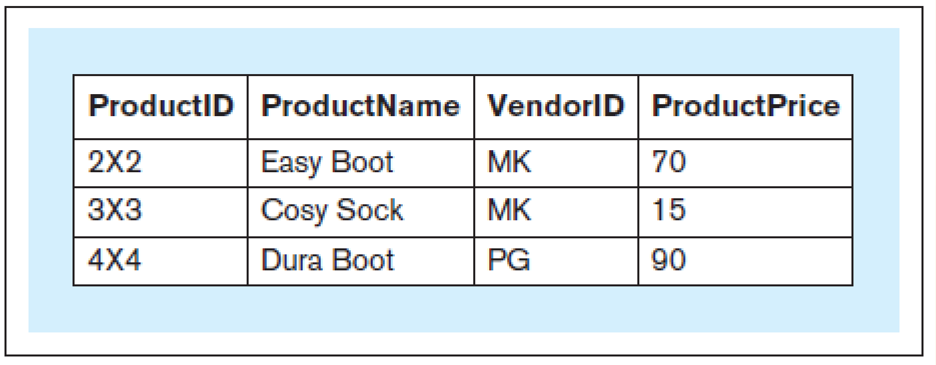
WHERE

Query 5 text: *Retrieve the product ID, product name, vendor ID, and product price for each product in the FW category whose price is equal to or below $110*

Query 5:

```
SELECT        productid, productname, vendorid,
              productprice
FROM          product
WHERE         productprice <= 110 AND
              categoryid = 'FW';
```

Query 5 result:



ProductID	ProductName	VendorID	ProductPrice
2X2	Easy Boot	MK	70
3X3	Cosy Sock	MK	15
4X4	Dura Boot	PG	90

DISTINCT

- DISTINCT
  - Can be used in conjunction with the SELECT statement
  - Eliminates duplicate values from a query result

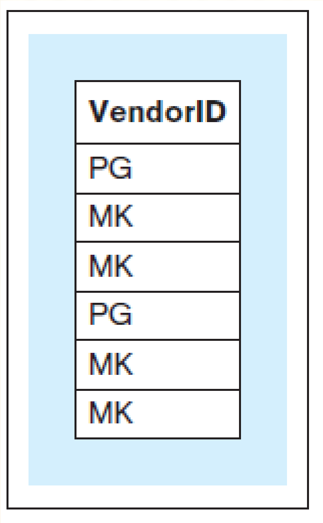
DISTINCT

Query 6 text: *Retrieve the VendorID value for each record in the relation PRODUCT*

Query 6:

```
SELECT       vendorid
FROM         product;
```

Query 6 result:



VendorID
PG
MK
MK
PG
MK
MK

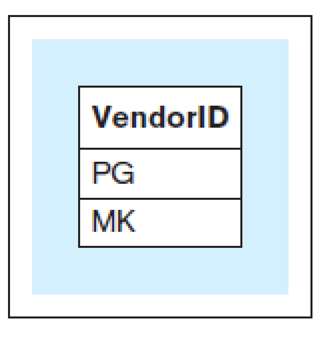
DISTINCT

Query 7 text: *Show one instance of all the different VendorID values in the relation PRODUCT*

Query 7:

```
SELECT      DISTINCT vendorid
FROM        product;
```

Query 7 result:



VendorID
PG
MK

ORDER BY

- ORDER BY
  - Used to sort the results of the query by one or more columns (or expressions)

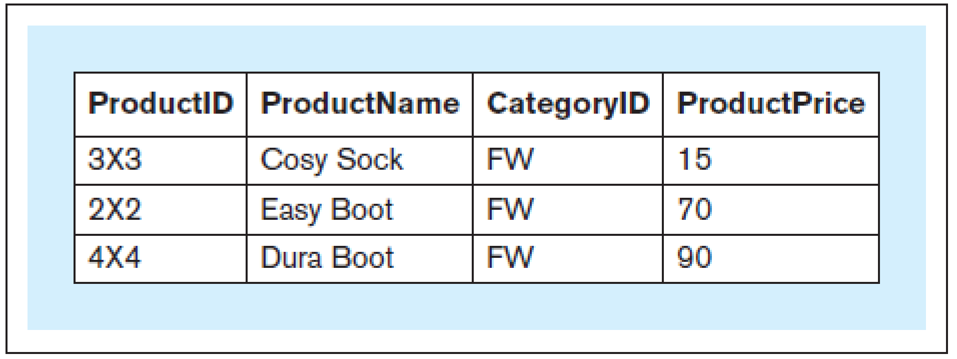
ORDER BY

Query 8 text: *Retrieve the product ID, product name, category ID, and product price for each product in the FW product category, sorted by product price*

Query 8:

```
SELECT        productid, productname, categoryid,
              productprice
FROM          product
WHERE         categoryid = 'FW'
ORDER BY      productprice;
```

Query 8 result:



ProductID	ProductName	CategoryID	ProductPrice
3X3	Cosy Sock	FW	15
2X2	Easy Boot	FW	70
4X4	Dura Boot	FW	90

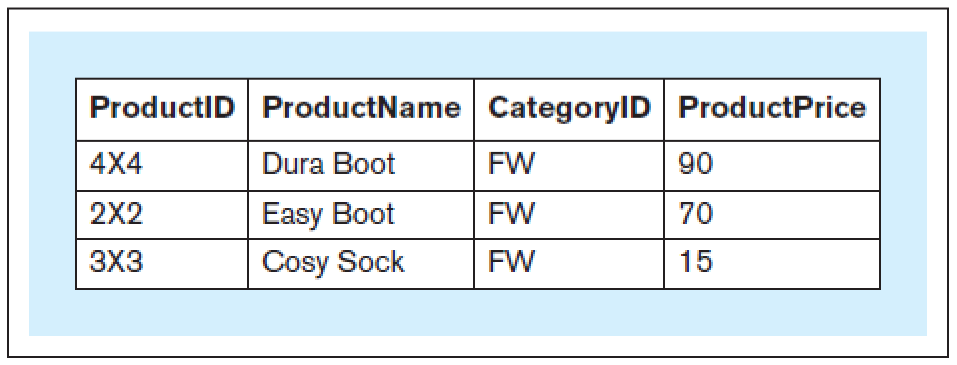
ORDER BY

Query 9 text: *Retrieve the product ID, product name, category ID, and product price for each product in the FW product category, sorted by product price in descending order*

Query 9:

```
SELECT       productid, productname, categoryid,
             productprice
FROM         product
WHERE        categoryid = 'FW'
ORDER BY     productprice DESC;
```

Query 9 result:



ProductID	ProductName	CategoryID	ProductPrice
4X4	Dura Boot	FW	90
2X2	Easy Boot	FW	70
3X3	Cosy Sock	FW	15

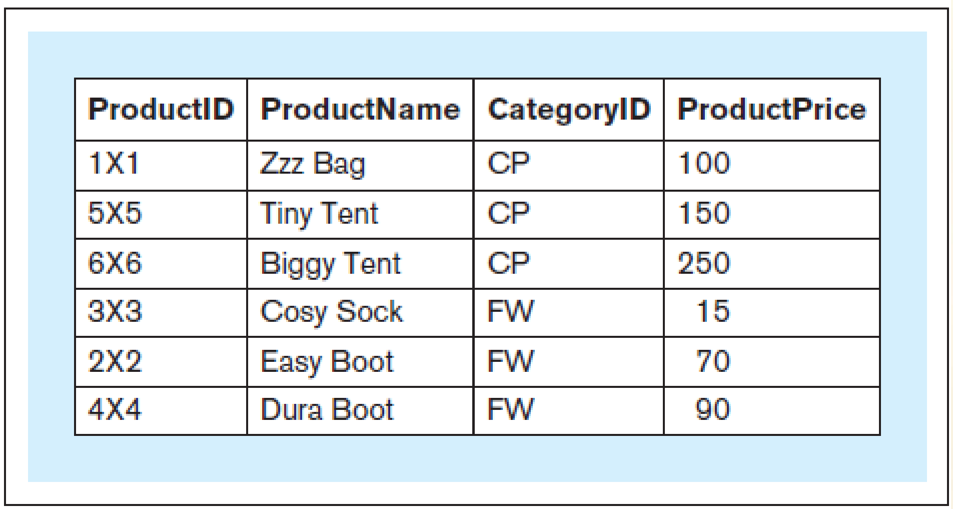
ORDER BY

Query 10 text: *Retrieve the product ID, product name, category ID, and product price for each product, sorted by category ID and, within the same category ID, by product price*

Query 10 :

```
SELECT          productid, productname, categoryid,
                productprice
FROM            product
ORDER BY        categoryid, productprice;
```

Query 10 result:



ProductID	ProductName	CategoryID	ProductPrice
1X1	Zzz Bag	CP	100
5X5	Tiny Tent	CP	150
6X6	Biggy Tent	CP	250
3X3	Cosy Sock	FW	15
2X2	Easy Boot	FW	70
4X4	Dura Boot	FW	90

LIKE

- LIKE
  - Used for retrieval of records whose values partially match a certain criteria

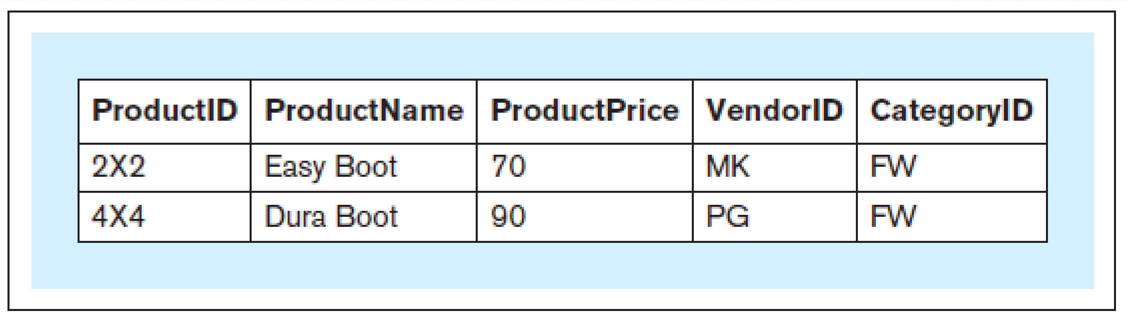
LIKE

Query 11 text: *Retrieve the record for each product whose product name contains the phrase 'Boot'*

Query 11 :

```
SELECT     *
FROM       product
WHERE      productname LIKE '%Boot%';
```

Query 11 result:



ProductID	ProductName	ProductPrice	VendorID	CategoryID
2X2	Easy Boot	70	MK	FW
4X4	Dura Boot	90	PG	FW

AGGREGATE FUNCTIONS

AGGREGATE FUNCTIONS

- **Aggregate functions**
  - For calculating and summarizing values in queries, SQL provides the following aggregate functions:
    - COUNT
    - SUM
    - AVG
    - MIN
    - MAX

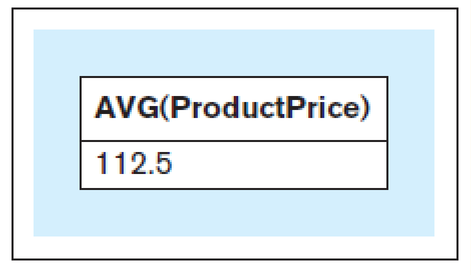
AGGREGATE FUNCTIONS

Query 12 text: *Retrieve the average price of all products*

Query 12 :

```
SELECT AVG(productprice)
FROM product;
```

Query 12 result:



AVG(ProductPrice)
112.5

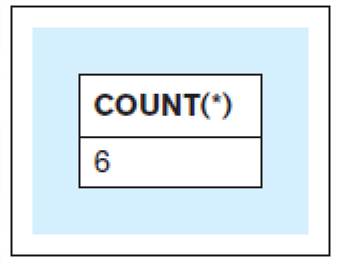
AGGREGATE FUNCTIONS

Query 13 text: *Show how many products we offer for sale*

Query 13 :

```
SELECT COUNT(*)
FROM product;
```

Query 13 result:



COUNT(*)
6

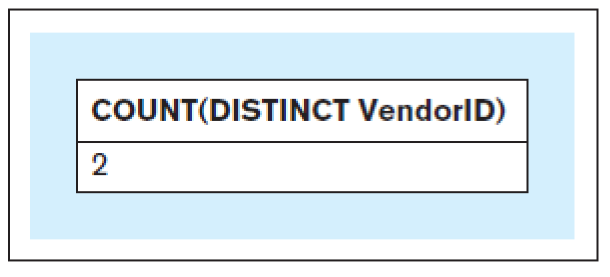
AGGREGATE FUNCTIONS

Query 14 text: *Retrieve the number of vendors that supply our products*

Query 14 :

```
SELECT COUNT(DISTINCT vendorid)
FROM product;
```

Query 14 result:



COUNT(DISTINCT VendorID)
2

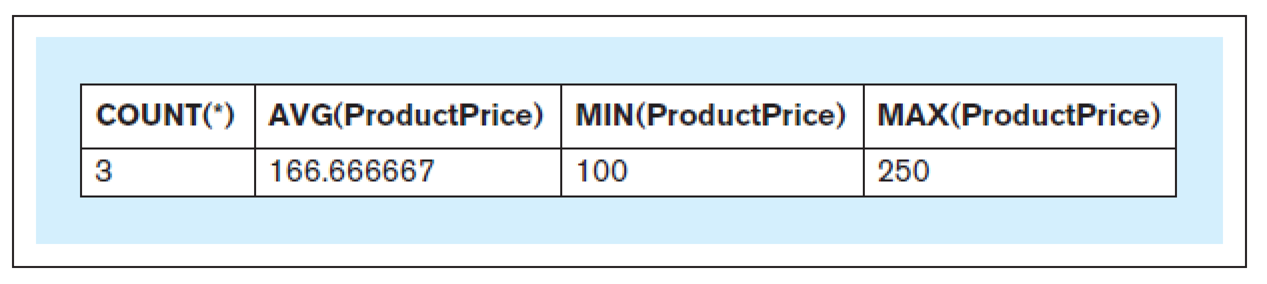
AGGREGATE FUNCTIONS

Query 15 text: *Retrieve the number of products, average product price, lowest product price, and highest product price in the CP product category*

Query 15 :

```
SELECT       COUNT(*), AVG(productprice),
             MIN(productprice), MAX(productprice)
FROM         product
WHERE        categoryid = 'CP';
```

Query 15 result:



COUNT(*)	AVG(ProductPrice)	MIN(ProductPrice)	MAX(ProductPrice)
3	166.666667	100	250

GROUP BY

- GROUP BY
  - Enables summarizations across the groups of related data within tables

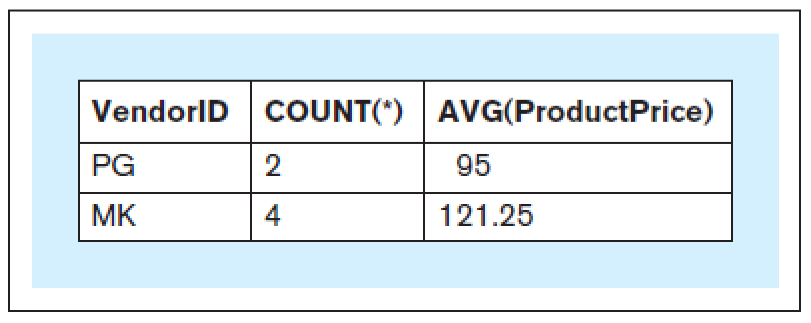
GROUP BY

Query 16 text: *For each vendor, retrieve the vendor ID, number of products supplied by the vendor, and average price of the products supplied by the vendor*

Query 16 :

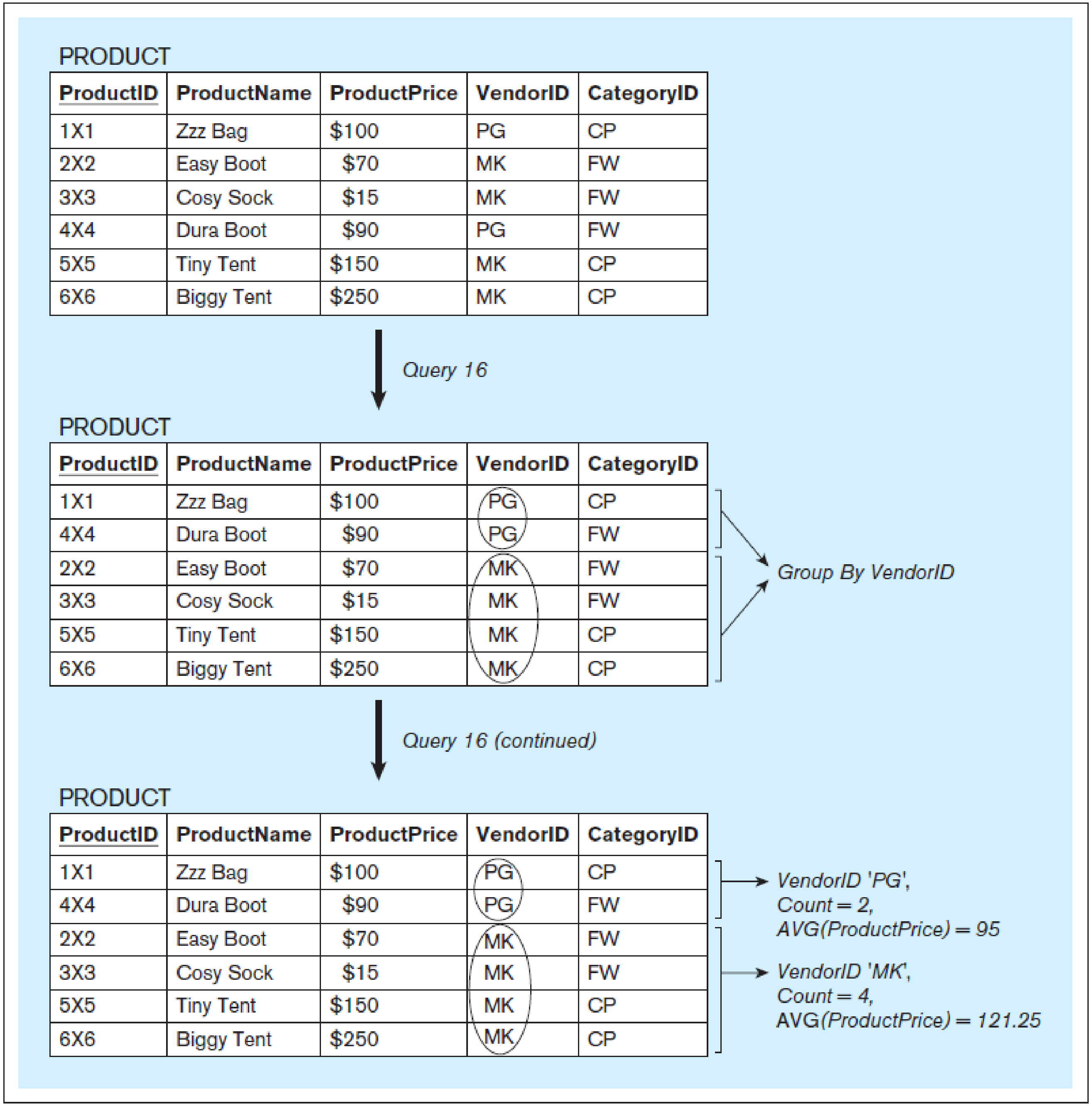
```
SELECT       vendorid, COUNT(*), AVG(productprice)
FROM         product
GROUP BY     vendorid;
```

Query 16 result:



VendorID	COUNT(*)	AVG(ProductPrice)
PG	2	95
MK	4	121.25

Query 16 illustration



PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	$100	PG	CP
2X2	Easy Boot	$70	MK	FW
3X3	Cosy Sock	$15	MK	FW
4X4	Dura Boot	$90	PG	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

Query 16

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	$100	PG	CP
4X4	Dura Boot	$90	PG	FW
2X2	Easy Boot	$70	MK	FW
3X3	Cosy Sock	$15	MK	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

Group By VendorID

Query 16 (continued)

PRODUCT

ProductID	ProductName	ProductPrice	VendorID	CategoryID
1X1	Zzz Bag	$100	PG	CP
4X4	Dura Boot	$90	PG	FW
2X2	Easy Boot	$70	MK	FW
3X3	Cosy Sock	$15	MK	FW
5X5	Tiny Tent	$150	MK	CP
6X6	Biggy Tent	$250	MK	CP

VendorID 'PG',
Count = 2,
AVG(ProductPrice) = 95

VendorID 'MK',
Count = 4,
AVG(ProductPrice) = 121.25

GROUP BY

Query 16 text: *For each vendor, retrieve the vendor ID, number of products supplied by the vendor, and average price of the products supplied by the vendor*

Query 16 :
INVALID

```
SELECT        vendorid, COUNT(*), AVG(productprice)
FROM          product;   ERROR MESSAGE RETURNED
```

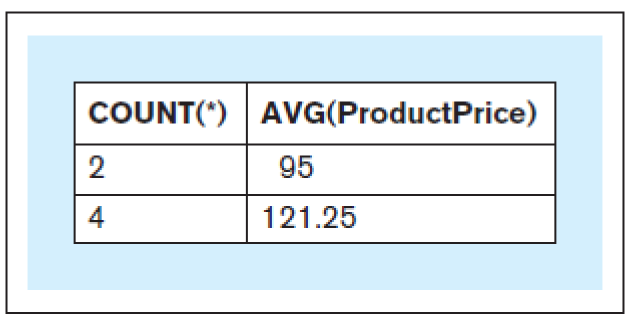
GROUP BY

Query 17 text: *For each vendor, retrieve the number of products supplied by the vendor and the average price of the products supplied by the vendor*

Query 17 :

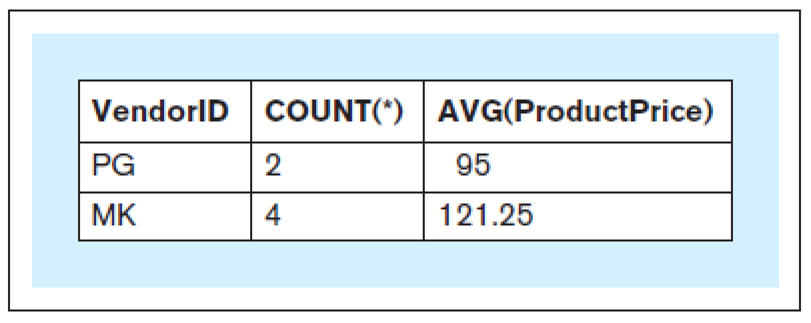
```
SELECT        COUNT(*), AVG(productprice)
FROM          product
GROUP BY      vendorid;
```

Query 17 result (vs. Query 16):



COUNT(*)	AVG(ProductPrice)
2	95
4	121.25

Query 17 result



VendorID	COUNT(*)	AVG(ProductPrice)
PG	2	95
MK	4	121.25

Query 16 result

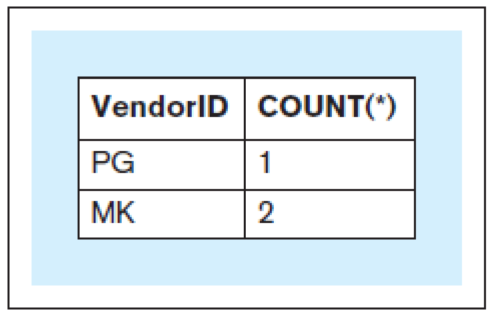
GROUP BY

Query 18 text: *For each vendor, retrieve the vendor ID and the number of products with a product price of $100 or higher supplied by the vendor*

Query 18 :

```
SELECT vendorid, COUNT(*)
FROM   product
WHERE  productprice >= 100
GROUP  BY vendorid;
```

Query 18 result:



VendorID	COUNT(*)
PG	1
MK	2

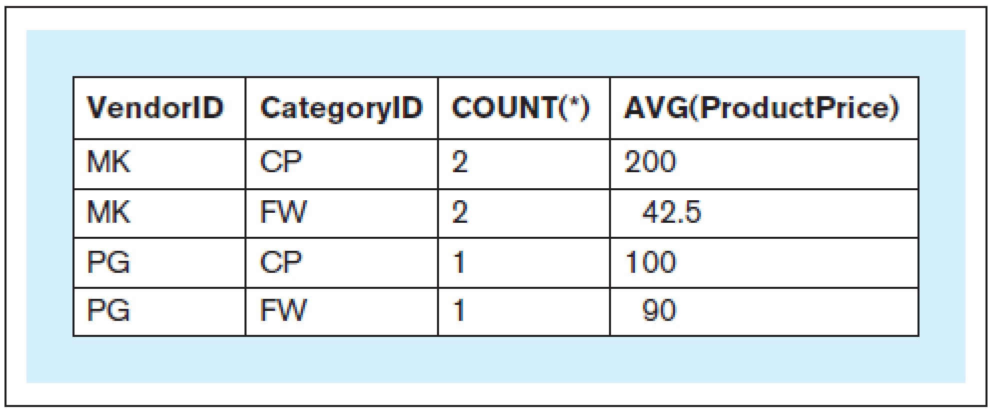
GROUP BY

Query 19 text: *Consider the groups of products where each group contains the products that are from the same category supplied by the same vendor. For each such group, retrieve the vendor ID, product category ID, number of products in the group, and average price of the products in the group.*

Query 19 :

```
SELECT      vendorid, categoryid, COUNT(*),
            AVG(productprice)
FROM        product
GROUP BY    vendorid, categoryid;
```

Query 19 result:



VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
MK	CP	2	200
MK	FW	2	42.5
PG	CP	1	100
PG	FW	1	90

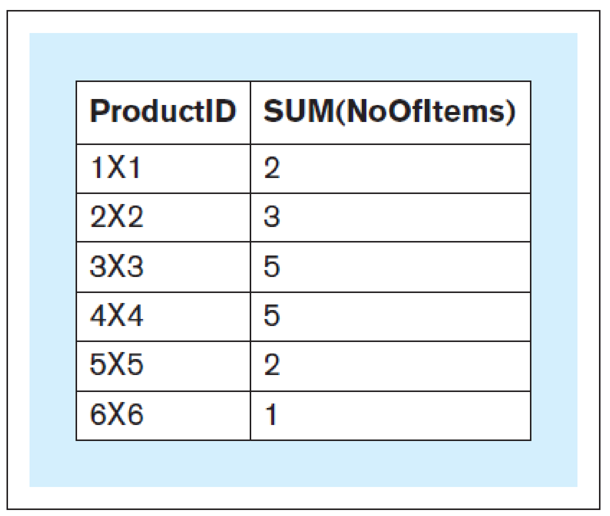
GROUP BY

Query 20 text: *For each product, retrieve the ProductID value and the total number of product items sold within all sales transactions.*

Query 20 :

```
SELECT productid, SUM(noofitems)
FROM   soldvia
GROUP BY productid;
```

Query 20 result:



ProductID	SUM(NoOfItems)
1X1	2
2X2	3
3X3	5
4X4	5
5X5	2
6X6	1

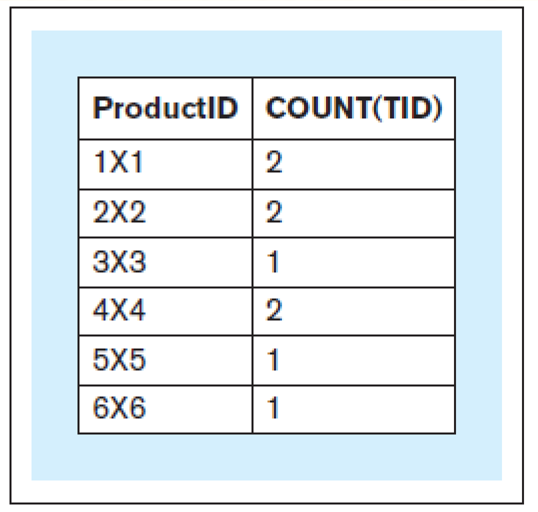
GROUP BY

Query 21 text: *For each product, retrieve the ProductID value and the number of sales transactions in which the product was sold*

Query 21:

```
SELECT productid, COUNT(*)
FROM   soldvia
GROUP BY productid;
```

Query 21 result:



ProductID	COUNT(TID)
1X1	2
2X2	2
3X3	1
4X4	2
5X5	1
6X6	1

HAVING

- HAVING
  - Enables summarizations across the groups of related data within tables
  - Determines which groups will be displayed in the result of a query and, consequently, which groups will not be displayed in the result of the query
  - A query that contains a HAVING clause must also contain a GROUP BY clause

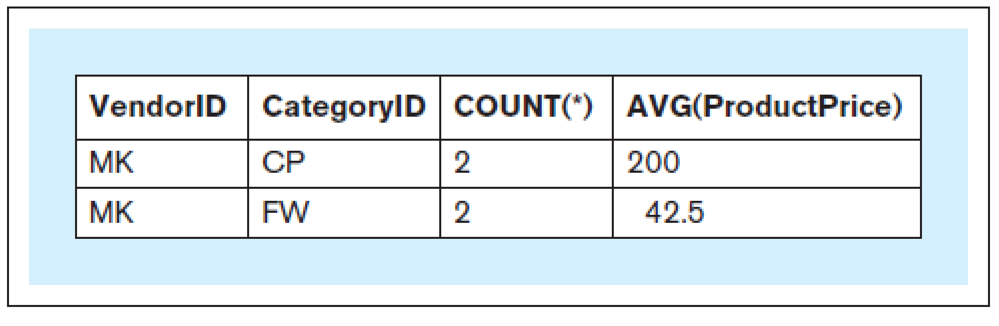
HAVING

Query 22 text: *Consider the groups of products where each group contains the products that are from the same category and supplied by the same vendor. For each such group that has more than one product, retrieve the vendor ID, product category ID, number of products in the group, and average price of the products in the group.*

Query 22:

```
SELECT        vendorid, categoryid, COUNT(*),
              AVG(productprice)
FROM          product
GROUP BY      vendorid, categoryid
HAVING        COUNT(*) > 1;
```

Query 22 result:



VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
MK	CP	2	200
MK	FW	2	42.5

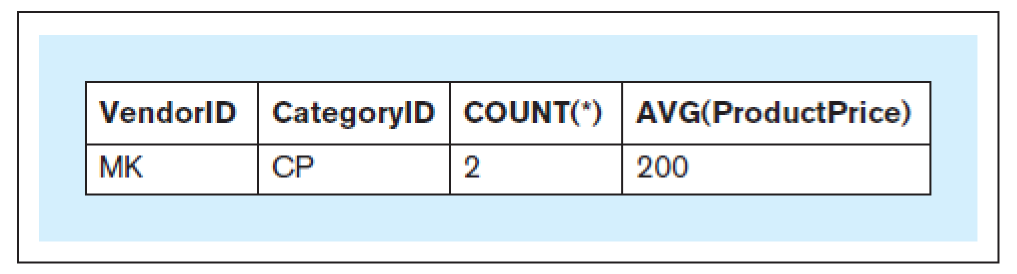
HAVING

Query 23 text: *Consider the groups of products where each group contains the products that are from the same category, supplied by the same vendor, and whose product price is $50 or higher. For each such group that has more than one product, retrieve the vendor ID, product category ID, number of products in the group, and average price of the products.*

Query 23:

```
SELECT       vendorid, categoryid, COUNT(*),
             AVG(productprice)
FROM         product
WHERE        productprice >= 50
GROUP BY     vendorid, categoryid
HAVING       COUNT(*) > 1;
```

Query 23 result:



VendorID	CategoryID	COUNT(*)	AVG(ProductPrice)
MK	CP	2	200

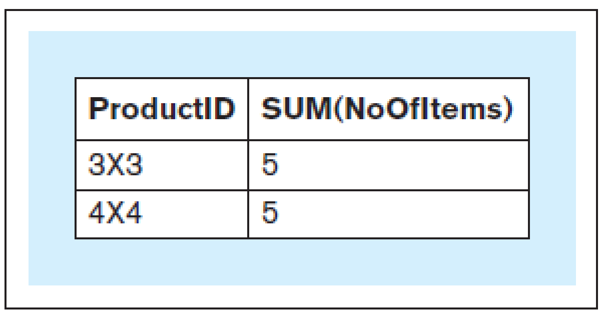
HAVING

Query 24 text: *For each product that has more than three items sold within all sales transactions, retrieve the ProductID value and the total number of product items sold within all sales transactions*

Query 24:

```
SELECT       productid, SUM(noofitems)
FROM         soldvia
GROUP BY     productid
HAVING       SUM(noofitems) > 3;
```

Query 24 result:



ProductID	SUM(NoOfItems)
3X3	5
4X4	5

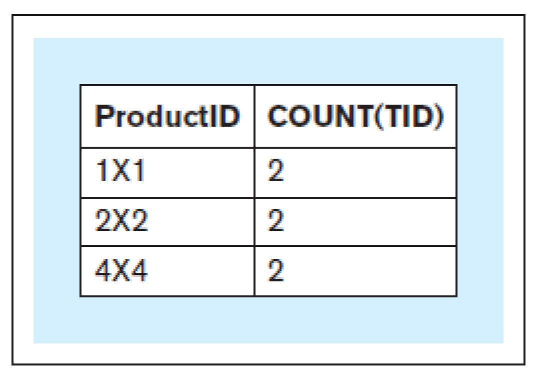
HAVING

Query 25 text: *For each product that was sold in more than one sales transaction, retrieve the ProductID value and the number of sales transactions in which the product was sold*

Query 25:

```
SELECT        productid, COUNT(*)
FROM          soldvia
GROUP BY      productid
HAVING        COUNT(*) > 1;
```

Query 25 result:



ProductID	COUNT(TID)
1X1	2
2X2	2
4X4	2

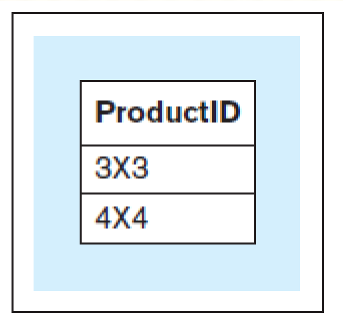
HAVING

Query 26 text: *For each product that has more than three items sold within all sales transactions, retrieve the ProductID value*

Query 26:

```
SELECT productid
FROM soldvia
GROUP BY productid
HAVING SUM(noofitems) > 3;
```

Query 26 result:



ProductID
3X3
4X4

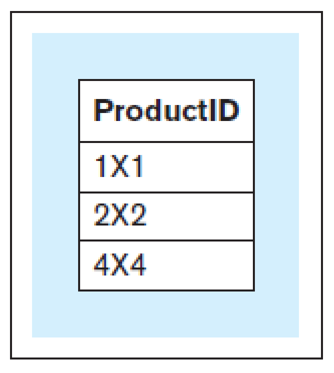
HAVING

Query 27 text: *For each product that was sold in more than one sales transaction, retrieve the ProductID value*

Query 27:

```
SELECT          productid
FROM            soldvia
GROUP BY        productid
HAVING          COUNT(*) > 1;
```

Query 27 result:



ProductID
1X1
2X2
4X4

NESTED QUERIES

- Nested Query
  - A query that is used within another query
    - A nested query is also referred to as an **inner query**,
    - The query that uses the nested query is referred to as an **outer query**

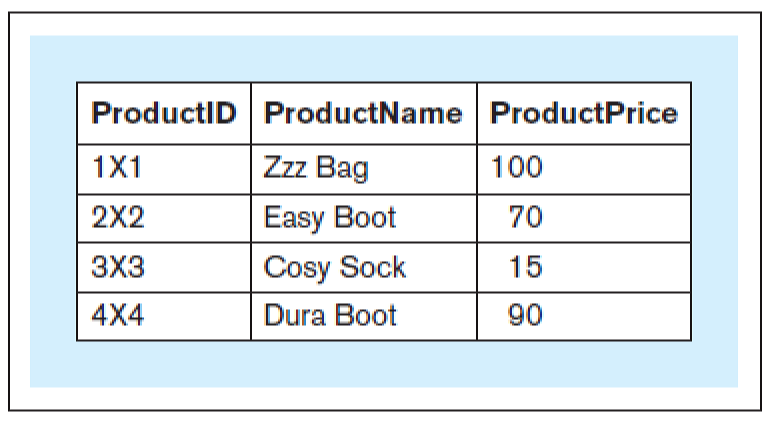
NESTED QUERIES

Query 28 text: *For each product whose product price is below the average price of all products, retrieve the product ID, product name, and product price*

Query 28:

```
SELECT productid, productname, productprice
FROM   product
WHERE  productprice < ( SELECT AVG(productprice)
                        FROM product);
```

Query 28 result:



ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
2X2	Easy Boot	70
3X3	Cosy Sock	15
4X4	Dura Boot	90

NESTED QUERIES

Query 28 text: *For each product whose product price is below the average price of all products, retrieve the product ID, product name, and product price*

Query 28:
INVALID

```
SELECT productid, productname, productprice
FROM   product
WHERE  productprice < AVG(productprice);
```

IN

- IN
  - Used for comparison of a value with a set of values

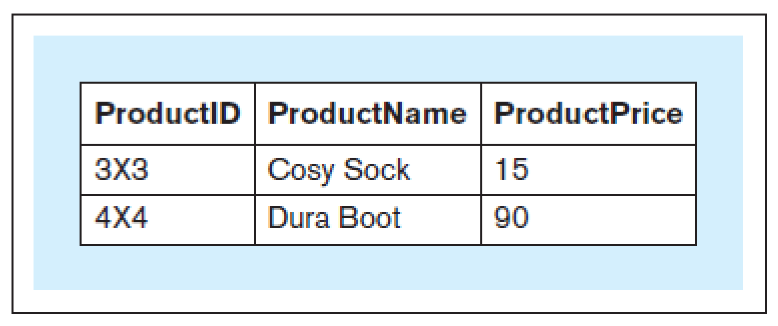
IN

Query 29 text: *For each product that has more than three items sold within all sales transactions, retrieve the product ID, product name, and product price*

Query 29:

```
SELECT       productid, productname, productprice
FROM         product
WHERE        productid IN
             (SELECT productid
              FROM soldvia
              GROUP BY productid
              HAVING SUM(noofitems) > 3);
```

Query 29 result:



ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Dura Boot	90

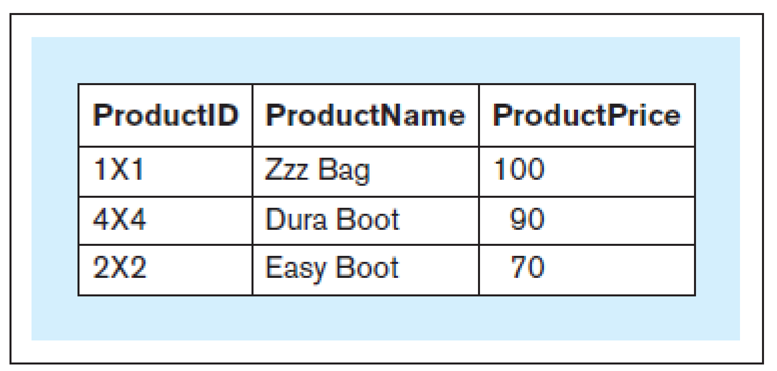
IN

Query 30 text: *For each product whose items were sold in more than one sales transaction, retrieve the product id, product name and product price*

Query 30 :

```
SELECT        productid, productname, productprice
FROM          product
WHERE         productid IN
              (SELECT productid
               FROM soldvia
               GROUP BY productid
               HAVING COUNT(*) > 1);
```

Query 30 result:



ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
4X4	Dura Boot	90
2X2	Easy Boot	70

JOIN Quick Intro

JOIN

- Facilitates the querying of multiple tables

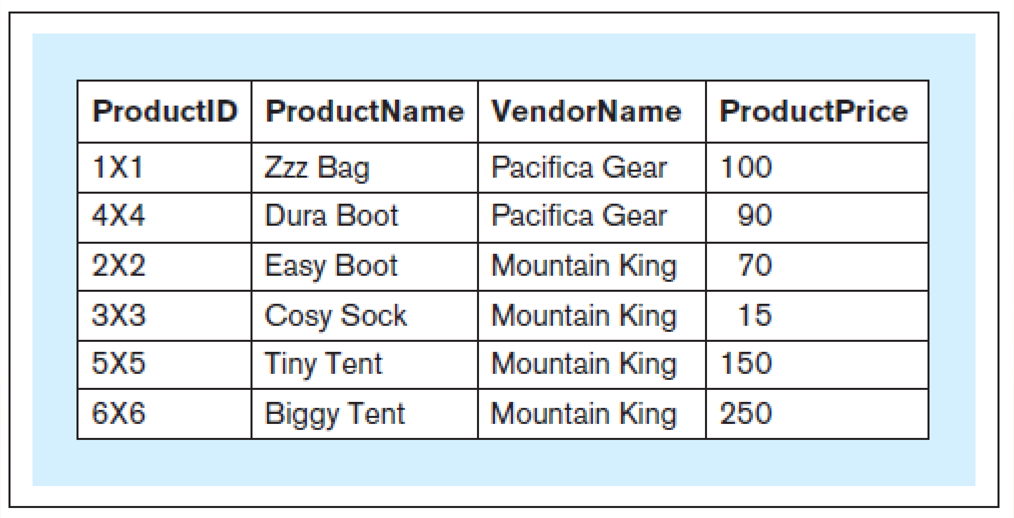
JOIN

Query 31 text: *For each product, retrieve the product ID, name of the product, name of the vendor of the product, and price of the product*

Query 31:

```
SELECT      productid, productname, vendorname,
            productprice
FROM        product, vendor
WHERE       product.vendorid = vendor.vendorid;
```

Query 31 result:



ProductID	ProductName	VendorName	ProductPrice
1X1	Zzz Bag	Pacifica Gear	100
4X4	Dura Boot	Pacifica Gear	90
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

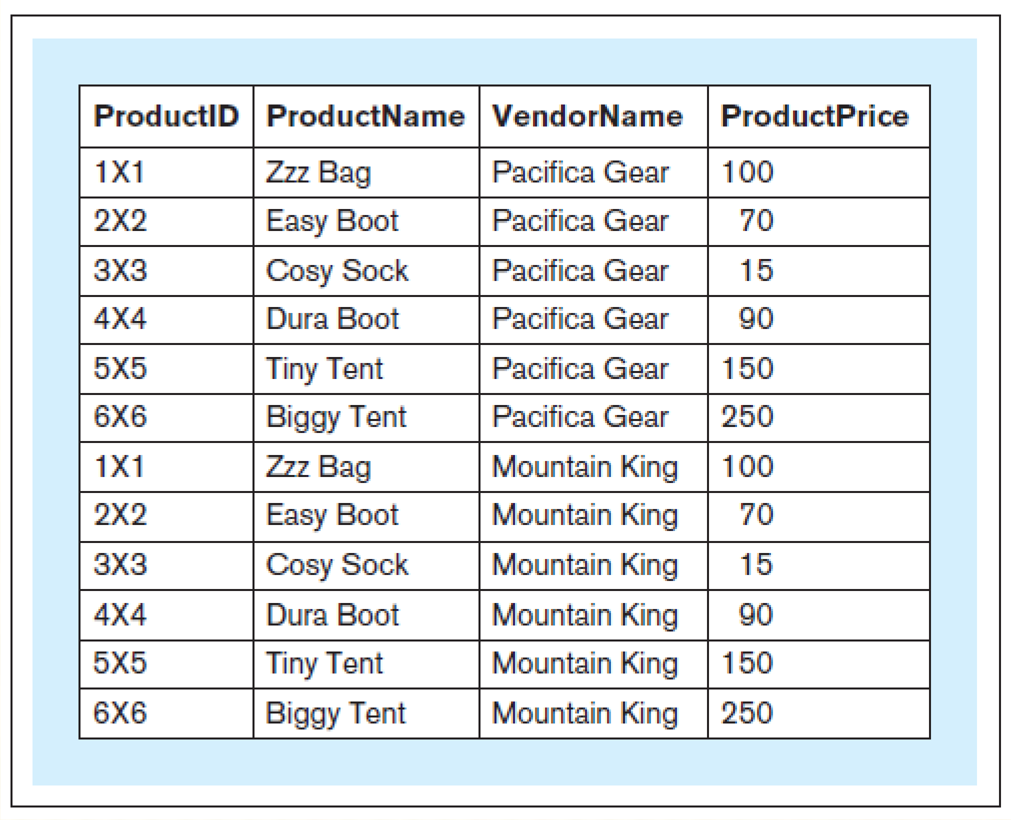
JOIN

Query 32:

```
SELECT      productid, productname, vendorname,
            productprice
FROM        product, vendor;
```

JOIN

Query 32 result:



ProductID	ProductName	VendorName	ProductPrice
1X1	Zzz Bag	Pacifica Gear	100
2X2	Easy Boot	Pacifica Gear	70
3X3	Cosy Sock	Pacifica Gear	15
4X4	Dura Boot	Pacifica Gear	90
5X5	Tiny Tent	Pacifica Gear	150
6X6	Biggy Tent	Pacifica Gear	250
1X1	Zzz Bag	Mountain King	100
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
4X4	Dura Boot	Mountain King	90
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

JOIN

Query 33:

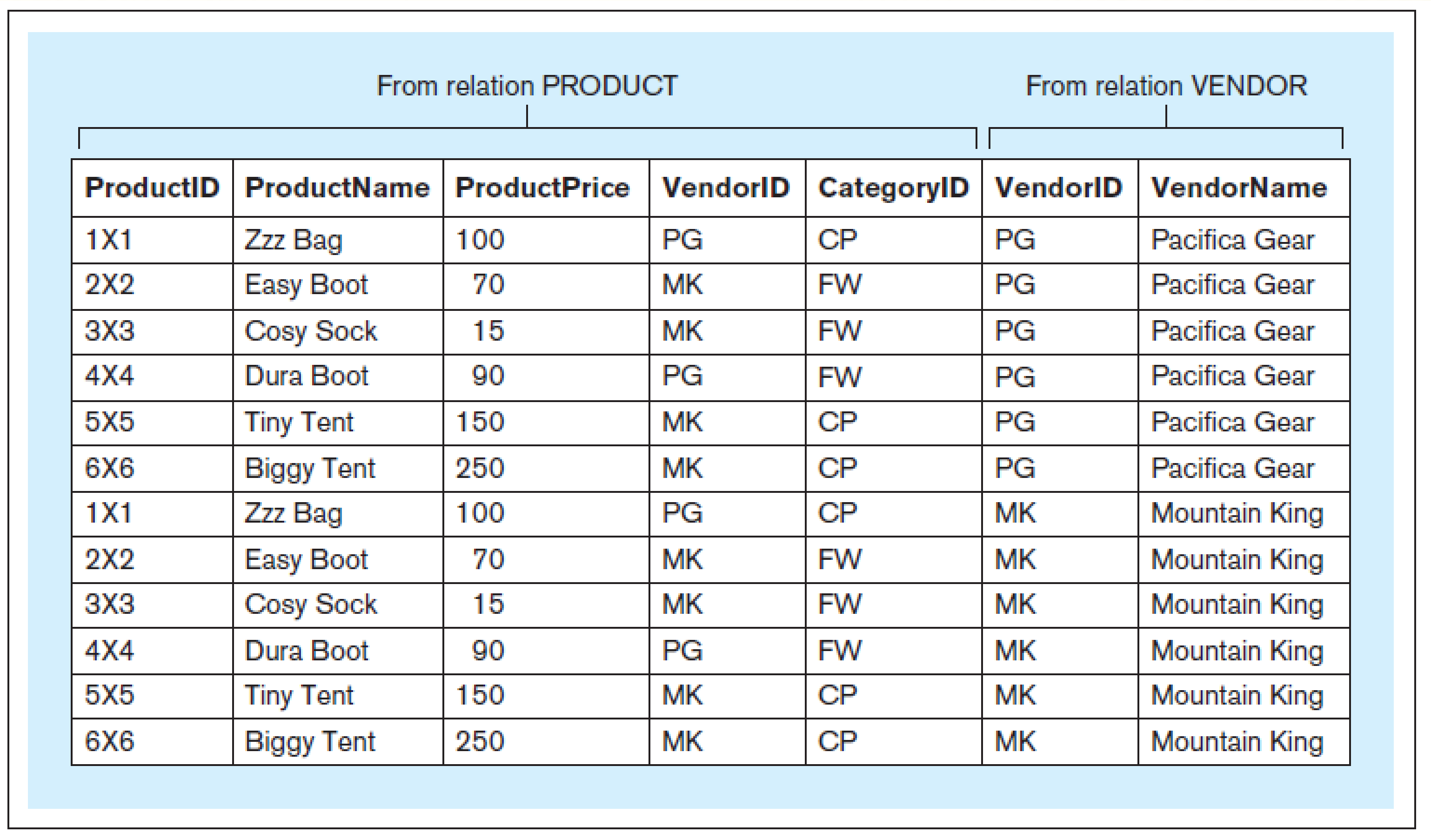
```
SELECT      *
FROM        product, vendor;
```

Query 34:

```
SELECT      *
FROM        product, vendor
WHERE       product.vendorid = vendor.vendorid;
```

JOIN

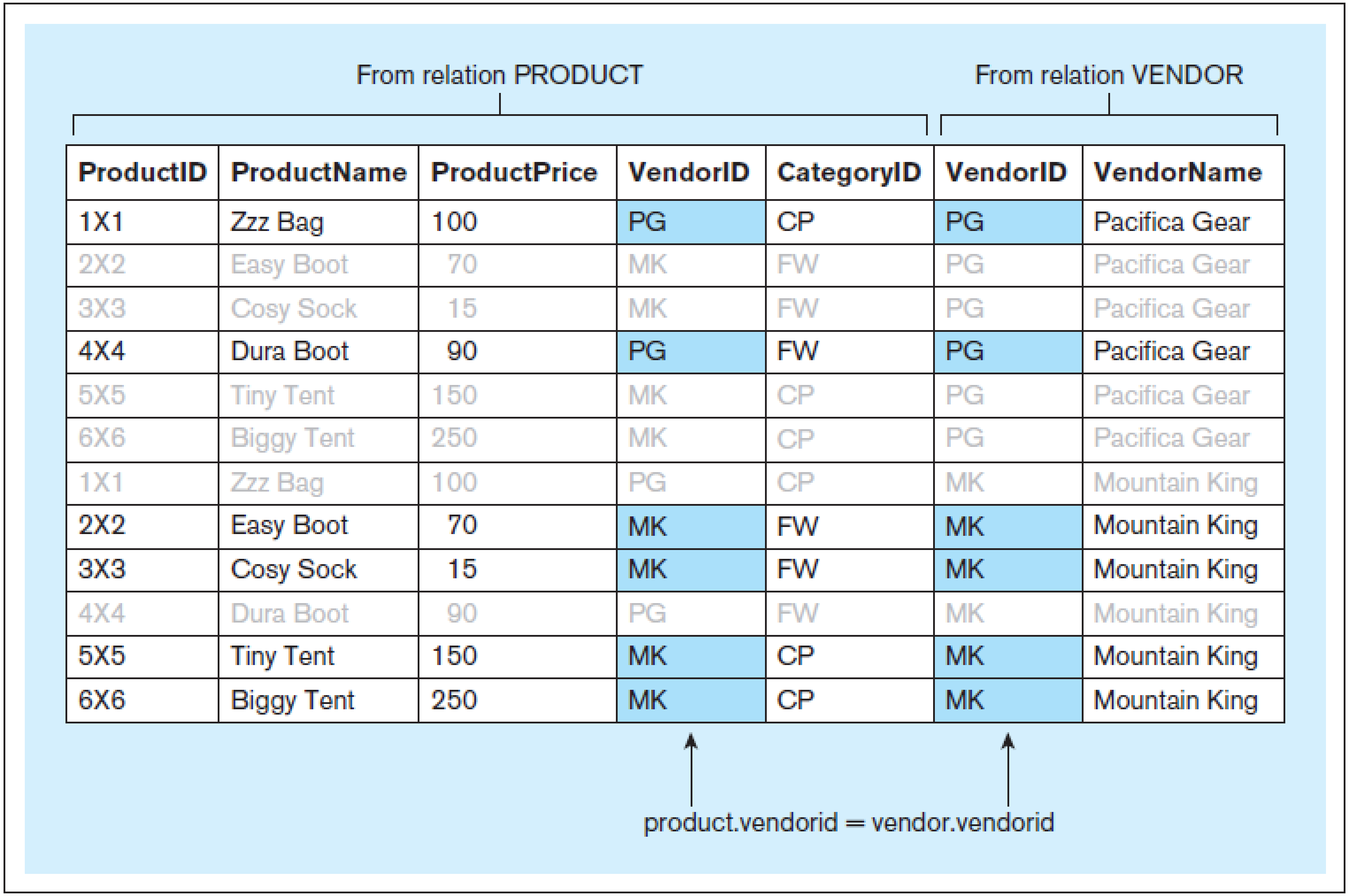
Query 33 result:



From relation PRODUCT					From relation VENDOR	
ProductID	**ProductName**	**ProductPrice**	**VendorID**	**CategoryID**	**VendorID**	**VendorName**
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	PG	Pacifica Gear
3X3	Cosy Sock	15	MK	FW	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
5X5	Tiny Tent	150	MK	CP	PG	Pacifica Gear
6X6	Biggy Tent	250	MK	CP	PG	Pacifica Gear
1X1	Zzz Bag	100	PG	CP	MK	Mountain King
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
4X4	Dura Boot	90	PG	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

JOIN

Formation of the result of Query 34:



From relation PRODUCT

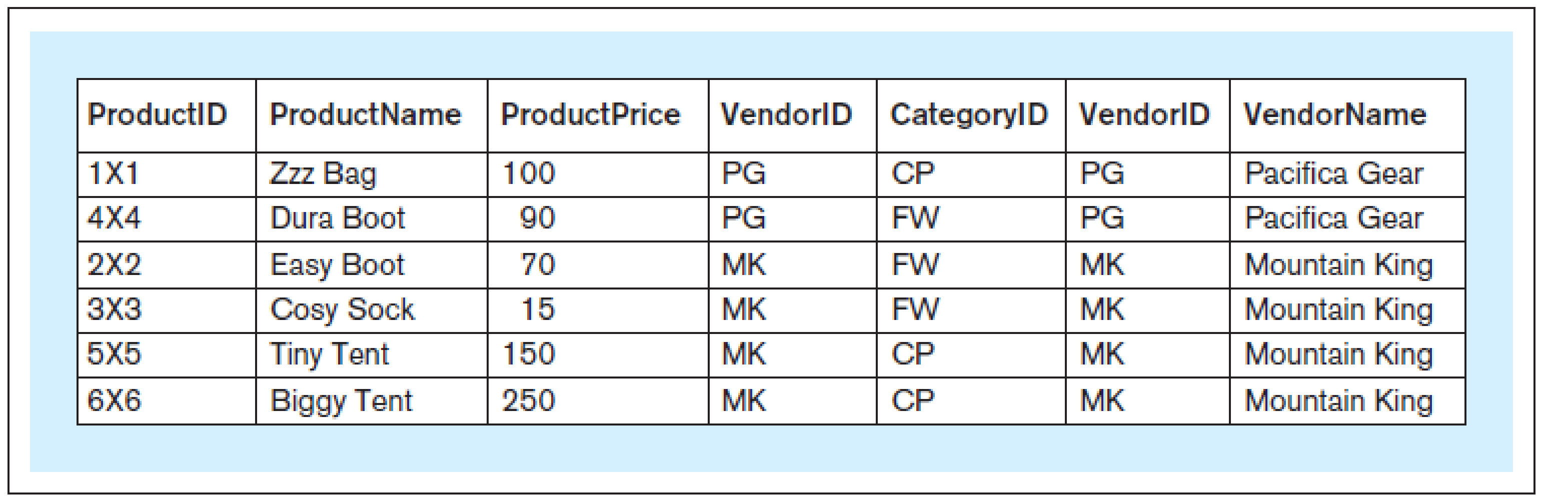
From relation VENDOR

ProductID	ProductName	ProductPrice	VendorID	CategoryID	VendorID	VendorName
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	PG	Pacifica Gear
3X3	Cosy Sock	15	MK	FW	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
5X5	Tiny Tent	150	MK	CP	PG	Pacifica Gear
6X6	Biggy Tent	250	MK	CP	PG	Pacifica Gear
1X1	Zzz Bag	100	PG	CP	MK	Mountain King
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
4X4	Dura Boot	90	PG	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

product.vendorid = vendor.vendorid

JOIN

Query 34 result:



ProductID	ProductName	ProductPrice	VendorID	CategoryID	VendorID	VendorName
1X1	Zzz Bag	100	PG	CP	PG	Pacifica Gear
4X4	Dura Boot	90	PG	FW	PG	Pacifica Gear
2X2	Easy Boot	70	MK	FW	MK	Mountain King
3X3	Cosy Sock	15	MK	FW	MK	Mountain King
5X5	Tiny Tent	150	MK	CP	MK	Mountain King
6X6	Biggy Tent	250	MK	CP	MK	Mountain King

ALIAS

- Alias
    - An alternative and usually shorter name that can be used anywhere within a query instead of the full relation name

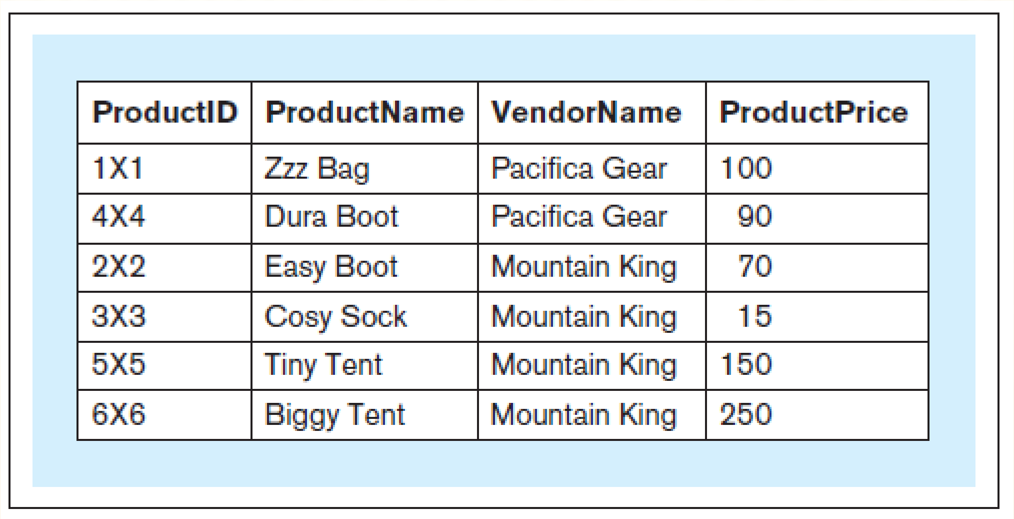
ALIAS

Query 31 text: *For each product, retrieve the product ID, name of the product, name of the vendor of the product, and price of the product*

Query 31:

```
SELECT        productid, productname, vendorname,
              productprice
FROM          product, vendor
WHERE         product.vendorid = vendor.vendorid;
```

Query 31 result:



ProductID	ProductName	VendorName	ProductPrice
1X1	Zzz Bag	Pacifica Gear	100
4X4	Dura Boot	Pacifica Gear	90
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

ALIAS

Query 31a text: *For each product, retrieve the product ID, name of the product, name of the vendor of the product, and price of the product*

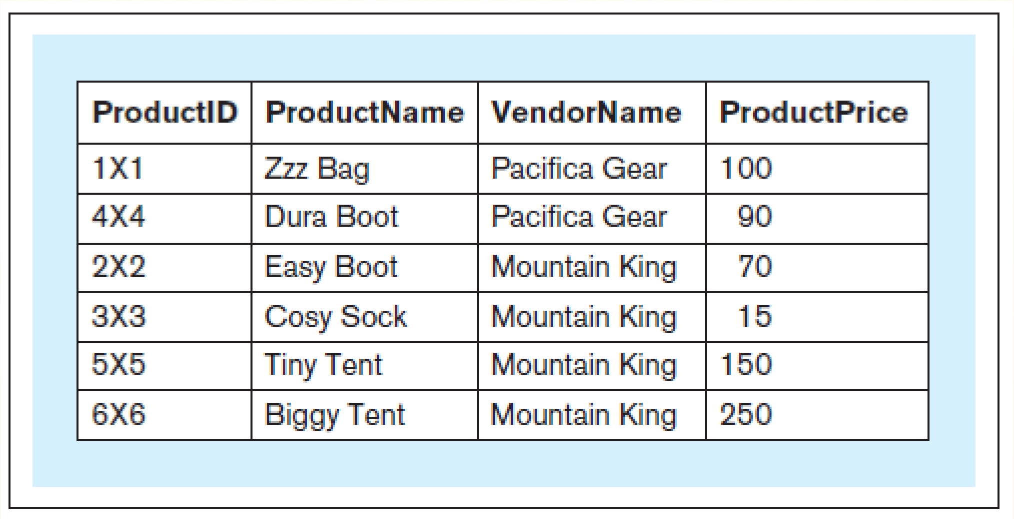
(same query)

Query 31a:

```
SELECT       p.productid, p.productname,
             v.vendorname, p.productprice
FROM         product p, vendor v
WHERE        p.vendorid = v.vendorid;
```

Query 31a result:

(same result)



ProductID	ProductName	VendorName	ProductPrice
1X1	Zzz Bag	Pacifica Gear	100
4X4	Dura Boot	Pacifica Gear	90
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

ALIAS

Query 31b text: *For each product, retrieve the product id, name of the product, name of the vendor of the product, and price of the product*

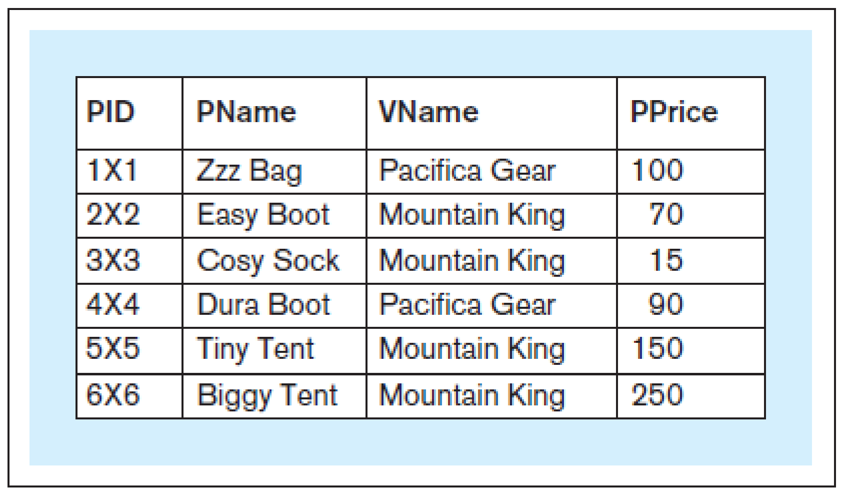
(same query)

Query 31b:

```
SELECT       p.productid pid, p.productname pname,
             v.vendorname vname, p.productprice pprice
FROM         product p, vendor v
WHERE        p.vendorid = v.vendorid;
```

Query 31b result:

(same result, different column names in the result)



PID	PName	VName	PPrice
1X1	Zzz Bag	Pacifica Gear	100
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
4X4	Dura Boot	Pacifica Gear	90
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

ALIAS

Query 31c text: *For each product, retrieve the product id, name of the product, name of the vendor of the product, and price of the product*

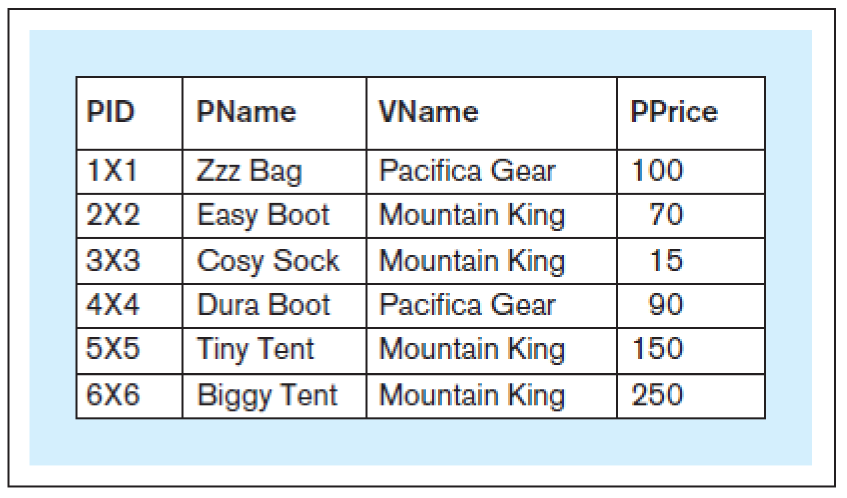
(same query)

Query 31c:

```
SELECT p.productid AS pid, p.productname AS pname,
       v.vendorname AS vname, p.productprice AS pprice
FROM   product p, vendor v
WHERE  p.vendorid = v.vendorid;
```

Query 31c result:

(same result, as Query 31b)



PID	PName	VName	PPrice
1X1	Zzz Bag	Pacifica Gear	100
2X2	Easy Boot	Mountain King	70
3X3	Cosy Sock	Mountain King	15
4X4	Dura Boot	Pacifica Gear	90
5X5	Tiny Tent	Mountain King	150
6X6	Biggy Tent	Mountain King	250

JOINING MULTIPLE RELATIONS

- Joining multiple relations
  - A query can contain multiple JOIN conditions, joining multiple relations

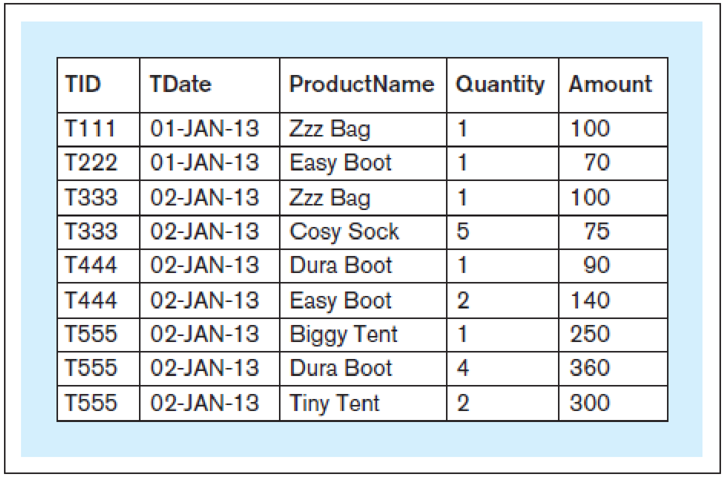
JOINING MULTIPLE RELATIONS

Query 35 text: *For each line item of a sales transaction, retrieve the transaction identifier, date of the transaction, name of the product that was sold, quantity sold, and amount charged*

Query 35:

```
SELECT t.tid, t.tdate, p.productname,
       sv.noofitems AS quantity,
       (sv.noofitems * p.productprice) AS amount
FROM   product p, salestransaction t, soldvia sv
WHERE  sv.productid = p.productid AND
       sv.tid = t.tid
ORDER BY t.tid;
```

Query 35 result:



TID	TDate	ProductName	Quantity	Amount
T111	01-JAN-13	Zzz Bag	1	100
T222	01-JAN-13	Easy Boot	1	70
T333	02-JAN-13	Zzz Bag	1	100
T333	02-JAN-13	Cosy Sock	5	75
T444	02-JAN-13	Dura Boot	1	90
T444	02-JAN-13	Easy Boot	2	140
T555	02-JAN-13	Biggy Tent	1	250
T555	02-JAN-13	Dura Boot	4	360
T555	02-JAN-13	Tiny Tent	2	300

UPDATE

UPDATE - USED TO MODIFY THE DATA STORED IN DATABASE RELATIONS

Insert Statement 1:

```
INSERT INTO product VALUES ('7×7','Airy Sock',1000,'MK','CP');
```

Update Statement 1:

```
UPDATE          product
SET             productprice = 10
WHERE           productid = '7×7';
```

Alter Statement 3:

```
ALTER TABLE product ADD
(discount NUMERIC(3,2) );
```

Update Statement 2:

```
UPDATE product
SET discount = 0.2;
```

Update Statement 3:

```
UPDATE product
SET discount = 0.3
WHERE vendorid = 'MK';
```

Alter Statement 4:

```
ALTER TABLE product DROP (discount);
```

DELETE

- DELETE
  - Used to delete the data stored in database relations

  - *Delete Statement 1:*

  -
    ```
    DELETE FROM  product
     WHERE            productid = '7×7';
    ```

DELETE

VIEW

- VIEW
  - Mechanism in SQL that allows the structure of a query to be saved in the RDBMS
  - Also known as a **virtual table**
    - View is not an actual table and does not have any data physically saved
  - Every time a view is invoked, it executes a query that retrieves the data from the actual tables
  - A view can be used in SELECT statements just like any other table from a database

VIEW

Create View Statement 1:

```
CREATE VIEW     products_more_than_3_sold AS
SELECT          productid, productname, productprice
FROM                        product
WHERE                       productid IN
    (
            SELECT productid
            FROM            soldvia
            GROUP BY        productid
            HAVING SUM(noofitems) > 3
    );
```

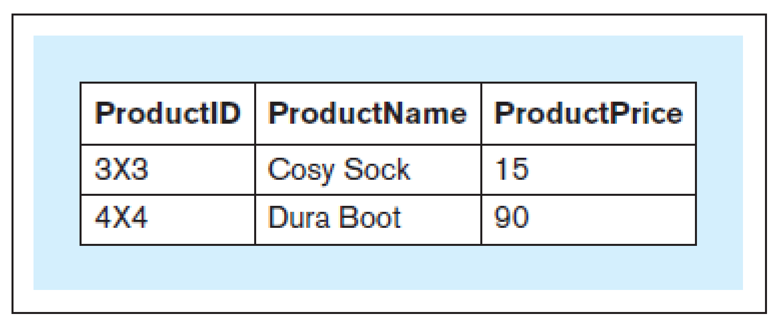
VIEW

Query 29 text: *For each product that has more than three items sold within all sales transactions, retrieve the product ID, product name, and product price*

Query 29:

```
SELECT        productid, productname, productprice
FROM          product
WHERE         productid IN
              (SELECT productid
               FROM soldvia
               GROUP BY productid
               HAVING SUM(noofitems) > 3);
```

Query 29 result:



ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Dura Boot	90

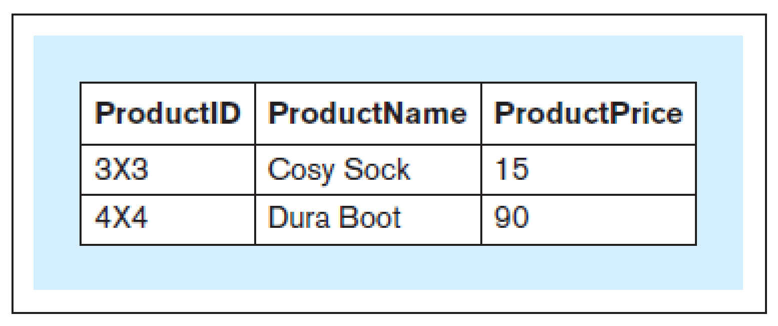
VIEW

Query 29a text: *For each product that has more than three items sold*
(same query) *within all sales transactions, retrieve the product ID, product name, and product price*

Query 29a:

```
SELECT       *
FROM products_more_than_3_sold;
```

Query 29a result:
(same result)



ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Dura Boot	90

VIEW

Create View Statement 2:

```
CREATE VIEW  products_in_multiple_trnsc AS
SELECT       productid, productname, productprice
FROM         product
WHERE        productid IN
             (SELECT productid
              FROM soldvia
              GROUP BY productid
              HAVING COUNT(*) > 1);
```

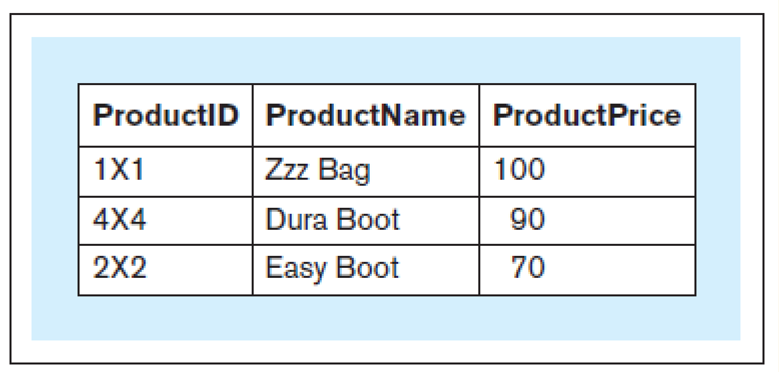
VIEW

Query 30 text: *For each product whose items were sold in more than one sales transaction, retrieve the product name and product price*

Query 30 :

```
SELECT          productid, productname, productprice
FROM            product
WHERE           productid IN
                (SELECT productid
                 FROM soldvia
                 GROUP BY productid
                 HAVING COUNT(*) > 1);
```

Query 30 result:



ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
4X4	Dura Boot	90
2X2	Easy Boot	70

VIEW

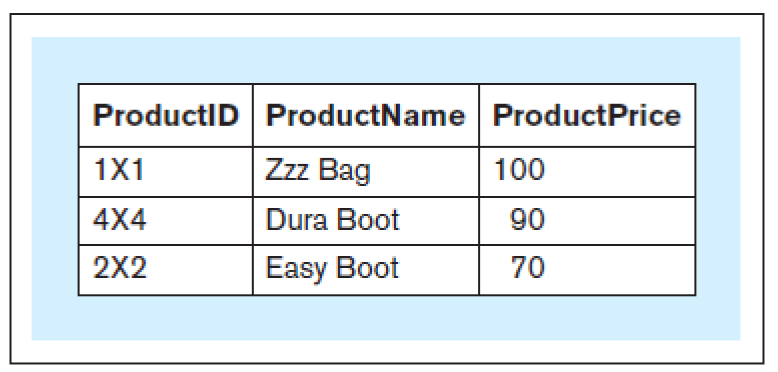
Query 30a text: *For each product whose items were sold in more than one sales transaction, retrieve the product name and product price*

Query 30a :

```
SELECT        *
FROM          products_in_multiple_trnsc;
```

Query 30a result:

(same result)



ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
4X4	Dura Boot	90
2X2	Easy Boot	70

SET OPERATORS

- **Set operators**
  - Standard set operators: **union**, **intersection**, and **difference**
  - Used to combine the results of two or more SELECT statements that are *union compatible*
  - Two sets of columns are **union compatible** if they contain the same number of columns, and if the data types of the columns in one set match the data types of the columns in the other set
    - The first column in one set has a compatible data type with the data type of the first column in the other set, the second column in one set has a compatible data type with the data type of the second column in the other set, and so on.
  - The set operators can combine results from SELECT statements querying relations, views, or other SELECT queries.

SET OPERATORS

- UNION
  - Used to combine the union compatible results of two SELECT statements by listing all rows from the result of the first SELECT statement and all rows from the result of the other SELECT statement
    - If two or more rows are identical only one of them is shown (duplicates are eliminated from the result)

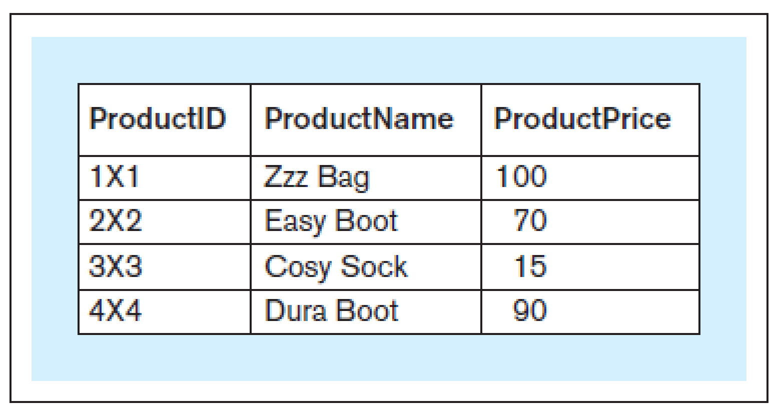
SET OPERATORS

Query 36 text: *Retrieve the product ID, product name, and product price for each product that has more than three items sold within all sales transactions or whose items were sold in more than one sales transaction*

Query 36:

```
SELECT        *
FROM          products_more_than_3_sold
UNION
SELECT        *
FROM          products_in_multiple_trnsc;
```

Query 36 result:



ProductID	ProductName	ProductPrice
1X1	Zzz Bag	100
2X2	Easy Boot	70
3X3	Cosy Sock	15
4X4	Dura Boot	90

SET OPERATORS

- INTERSECT
  - Used to combine the results of two SELECT statements that are union compatible by listing every row that appears in the result of both of the SELECT statements

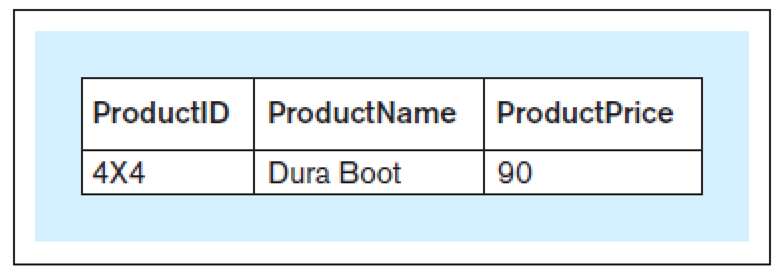
SET OPERATORS

Query 37 text: *Retrieve the product ID, product name, and product price for each product that has more than three items sold within all sales transactions and whose items were sold in more than one sales transaction*

Query 37:

```
SELECT       *
FROM         products_more_than_3_sold
INTERSECT
SELECT       *
FROM         products_in_multiple_trnsc;
```

Query 37 result:



ProductID	ProductName	ProductPrice
4X4	Dura Boot	90

SET OPERATORS

- MINUS (EXCEPT)
  - Used to combine the results of two SELECT statements that are union compatible by listing every row from the result of the first SELECT statement that does not appear in the result of the other SELECT statement

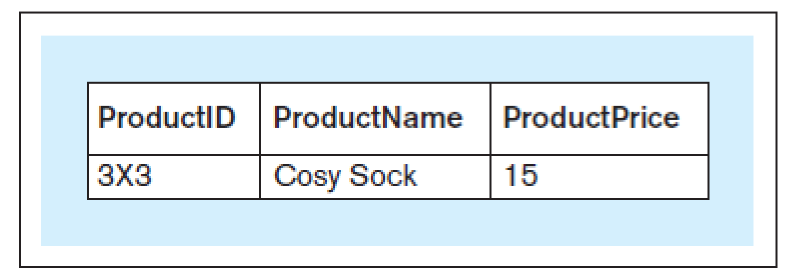
SET OPERATORS

Query 38 text: *Retrieve the product ID, product name, and product price for each product that has more than three items sold within all sales transactions but whose items were not sold in more than one sales transaction*

Query 38:

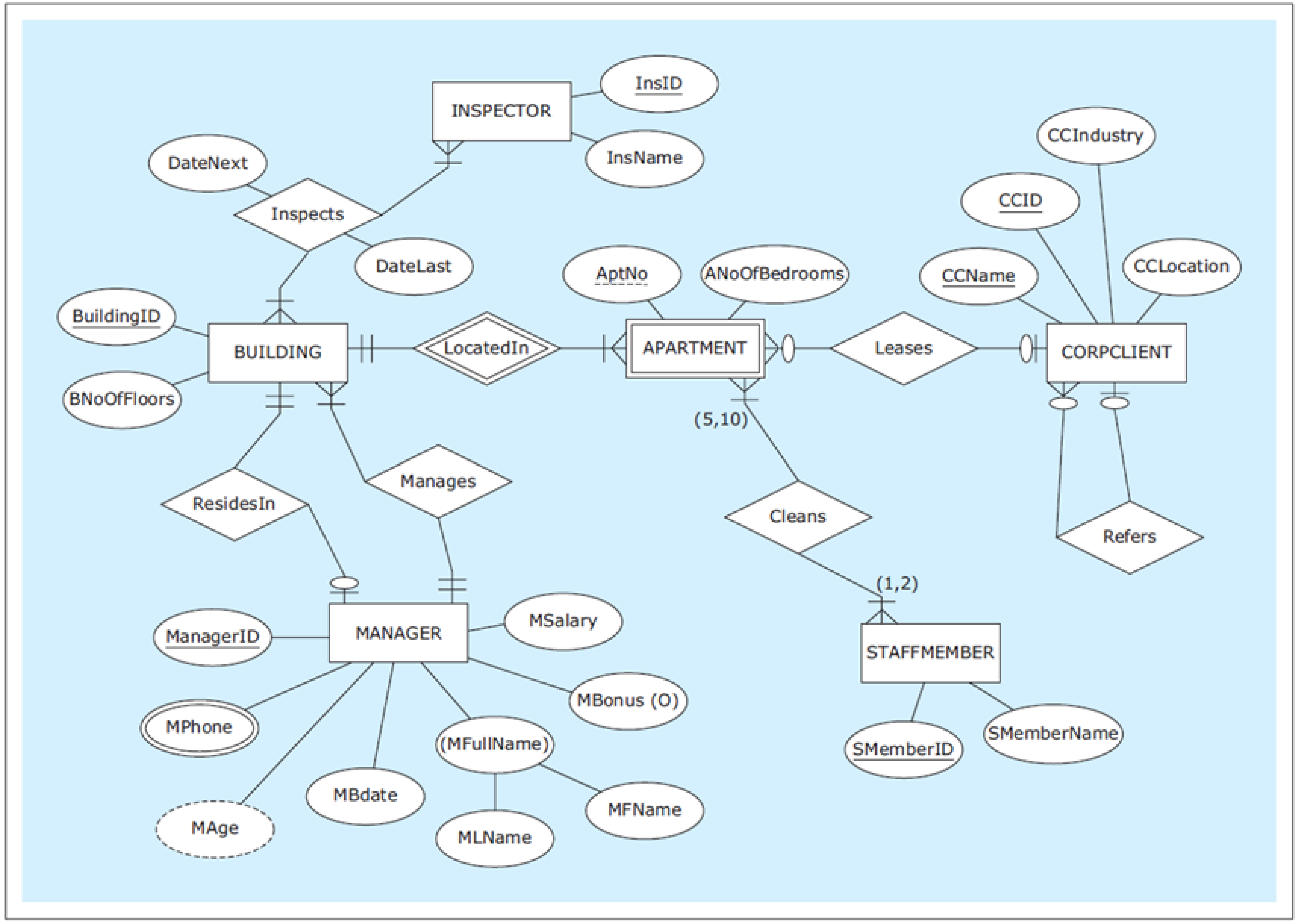
```
SELECT       *
FROM         products_more_than_3_sold
MINUS
SELECT       *
FROM         products_in_multiple_trnsc;
```

Query 38 result:

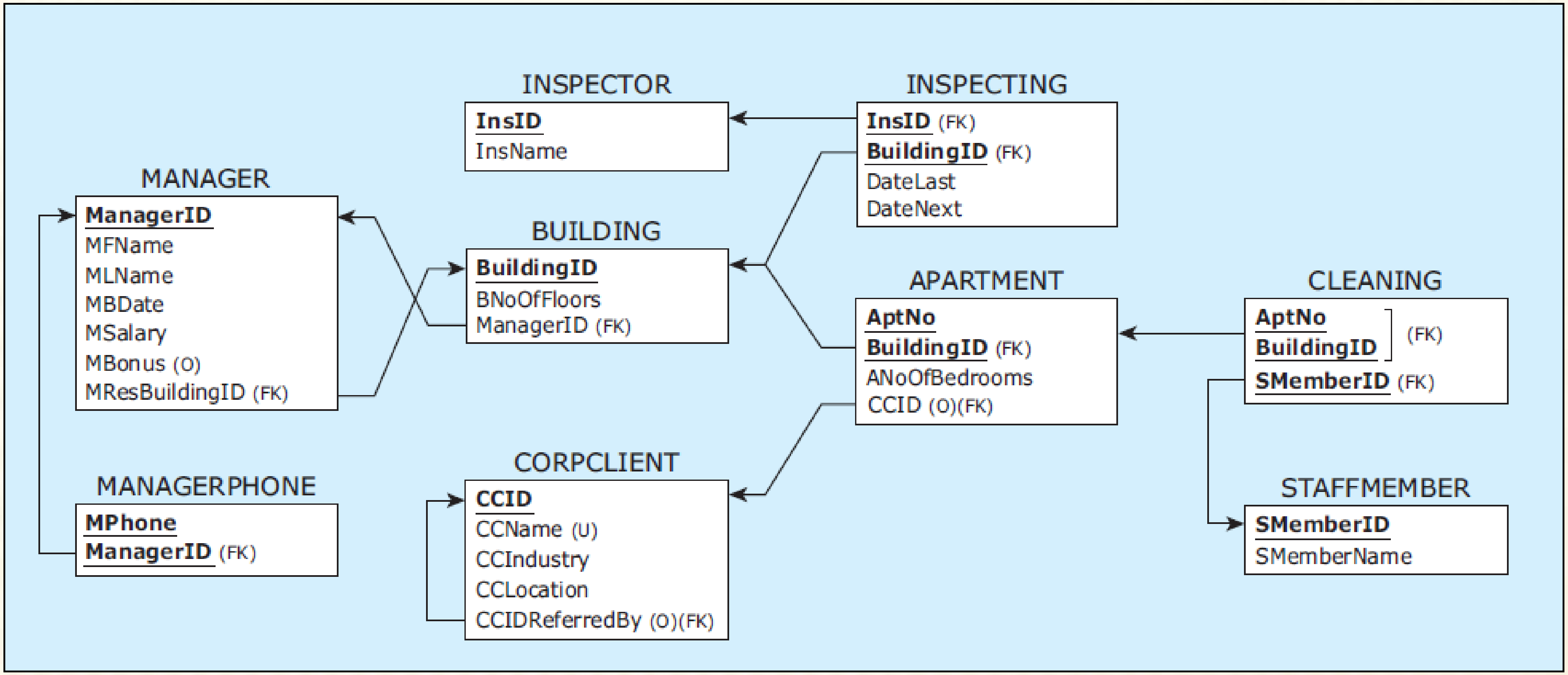


ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15

ER diagram: HAFH Realty Company Property Management Database



Relational schema: HAFH Realty Company Property Management Database



INSPECTOR
InsID
InsName

INSPECTING
InsID (FK)
BuildingID (FK)
DateLast
DateNext

MANAGER
ManagerID
MFName
MLName
MBDate
MSalary
MBonus (O)
MResBuildingID (FK)

BUILDING
BuildingID
BNoOfFloors
ManagerID (FK)

APARTMENT
AptNo
BuildingID (FK)
ANoOfBedrooms
CCID (O)(FK)

CLEANING
AptNo
BuildingID
(FK)
SMemberID (FK)

MANAGERPHONE
MPhone
ManagerID (FK)

CORPCLIENT
CCID
CCName (U)
CCIndustry
CCLocation
CCIDReferredBy (O)(FK)

STAFFMEMBER
SMemberID
SMemberName

CREATE TABLE statements for HAFH Realty Company Property Management Database

```
CREATE TABLE manager
(        managerid        CHAR(4)          NOT NULL,
         mfname           VARCHAR(15)      NOT NULL,
         mlname           VARCHAR(15)      NOT NULL,
         mbdate           DATE             NOT NULL,
         msalary          NUMERIC(9,2)     NOT NULL,
         mbonus           NUMERIC(9,2),
         mresbuildingid   CHAR(3),
PRIMARY KEY (managerid) );


CREATE TABLE managerphone
(        managerid        CHAR(4)          NOT NULL,
         mphone           CHAR(11)         NOT NULL,
PRIMARY KEY (managerid, mphone),
FOREIGN KEY (managerid) REFERENCES manager(managerid) );


CREATE TABLE building
(        buildingid       CHAR(3)          NOT NULL,
         bnooffloors      INT              NOT NULL,
         bmanagerid       CHAR(4)          NOT NULL,
PRIMARY KEY (buildingid),
FOREIGN KEY (bmanagerid) REFERENCES manager(managerid) );
```

CREATE TABLE statements for HAFH Realty Company Property Management Database

```
CREATE TABLE inspector
(       insid               CHAR(3)           NOT NULL,
        insname             VARCHAR(15)       NOT NULL,
PRIMARY KEY (insid) );


CREATE TABLE inspecting
(       insid               CHAR(3)           NOT NULL,
        buildingid          CHAR(3)           NOT NULL,
        datelast            DATE              NOT NULL,
        datenext            DATE              NOT NULL,
PRIMARY KEY (insid, buildingid),
FOREIGN KEY (insid) REFERENCES inspector(insid),
FOREIGN KEY (buildingid) REFERENCES building(buildingid) );


CREATE TABLE corpclient
(       ccid                CHAR(4)           NOT NULL,
        ccname              VARCHAR(25)       NOT NULL,
        ccindustry          VARCHAR(25)       NOT NULL,
        cclocation          VARCHAR(25)       NOT NULL,
        ccidreferredby      CHAR(4),
PRIMARY KEY (ccid),
UNIQUE (ccname),
FOREIGN KEY (ccidreferredby) REFERENCES corpclient(ccid) );
```

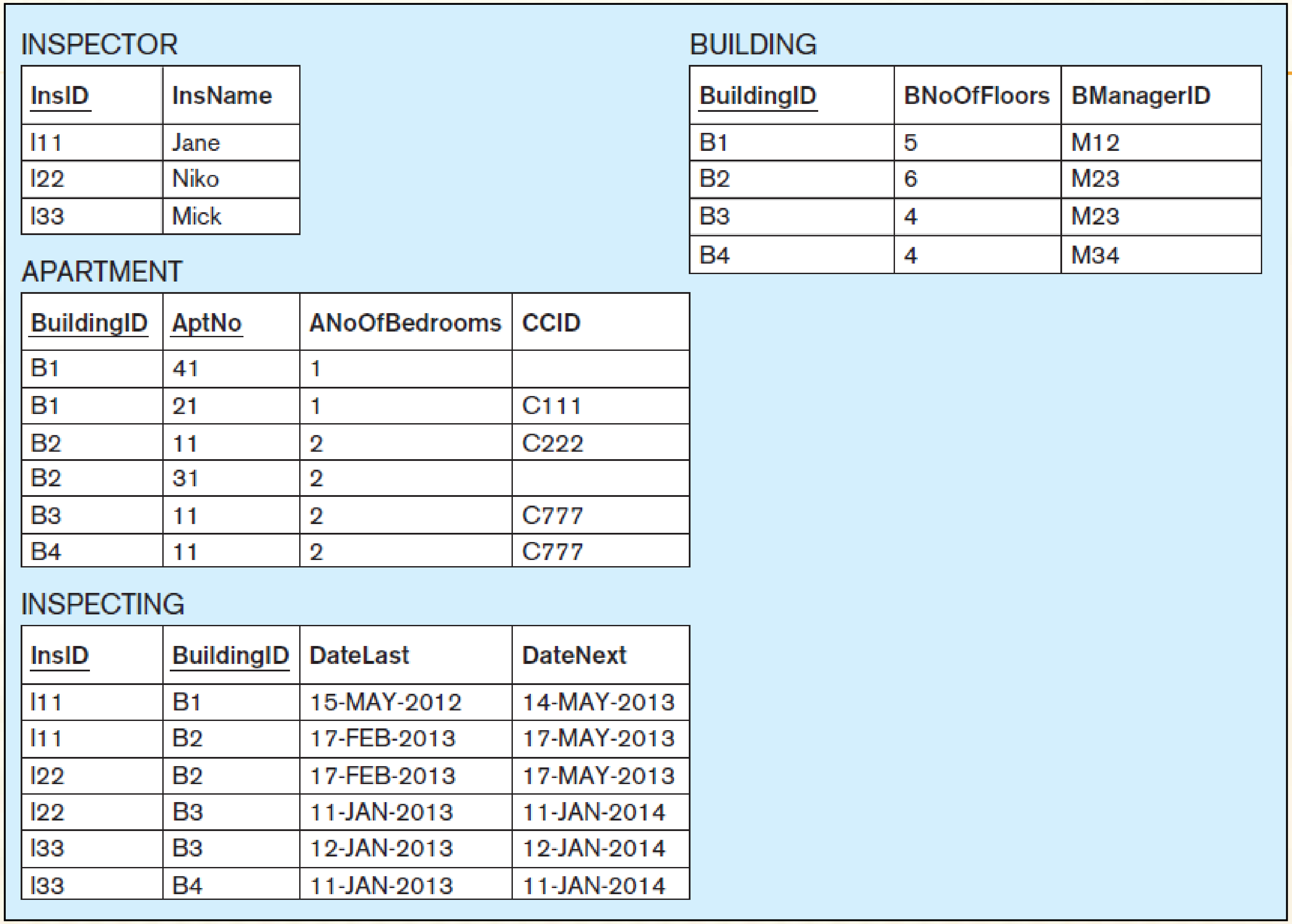
CREATE TABLE statements for HAFH Realty Company Property Management Database

```
CREATE TABLE apartment
(        buildingid       CHAR(3)          NOT NULL,
         aptno            CHAR(5)          NOT NULL,
         anoofbedrooms    INT              NOT NULL,
         ccid             CHAR(4),
PRIMARY KEY (buildingid, aptno),
FOREIGN KEY (buildingid) REFERENCES building(buildingid),
FOREIGN KEY (ccid) REFERENCES corpclient(ccid) );


CREATE TABLE staffmember
(        smemberid        CHAR(4)          NOT NULL,
         smembername      VARCHAR(15)      NOT NULL,
PRIMARY KEY (smemberid) );


CREATE TABLE cleaning
(        buildingid       CHAR(3)          NOT NULL,
         aptno            CHAR(5)          NOT NULL,
         smemberid        CHAR(4)          NOT NULL,
CONSTRAINT cleaningpk PRIMARY KEY (buildingid, aptno, smemberid),
CONSTRAINT cleaningfk1 FOREIGN KEY (buildingid, aptno)
                      REFERENCES apartment(buildingid, aptno),
CONSTRAINT cleaningfk2 FOREIGN KEY (smemberid)
                      REFERENCES staffmember(smemberid) );
```

Data records: HAFH Realty Company Property Management Database (part 1)



INSPECTOR

InsID	InsName
I11	Jane
I22	Niko
I33	Mick

BUILDING

BuildingID	BNoOfFloors	BManagerID
B1	5	M12
B2	6	M23
B3	4	M23
B4	4	M34

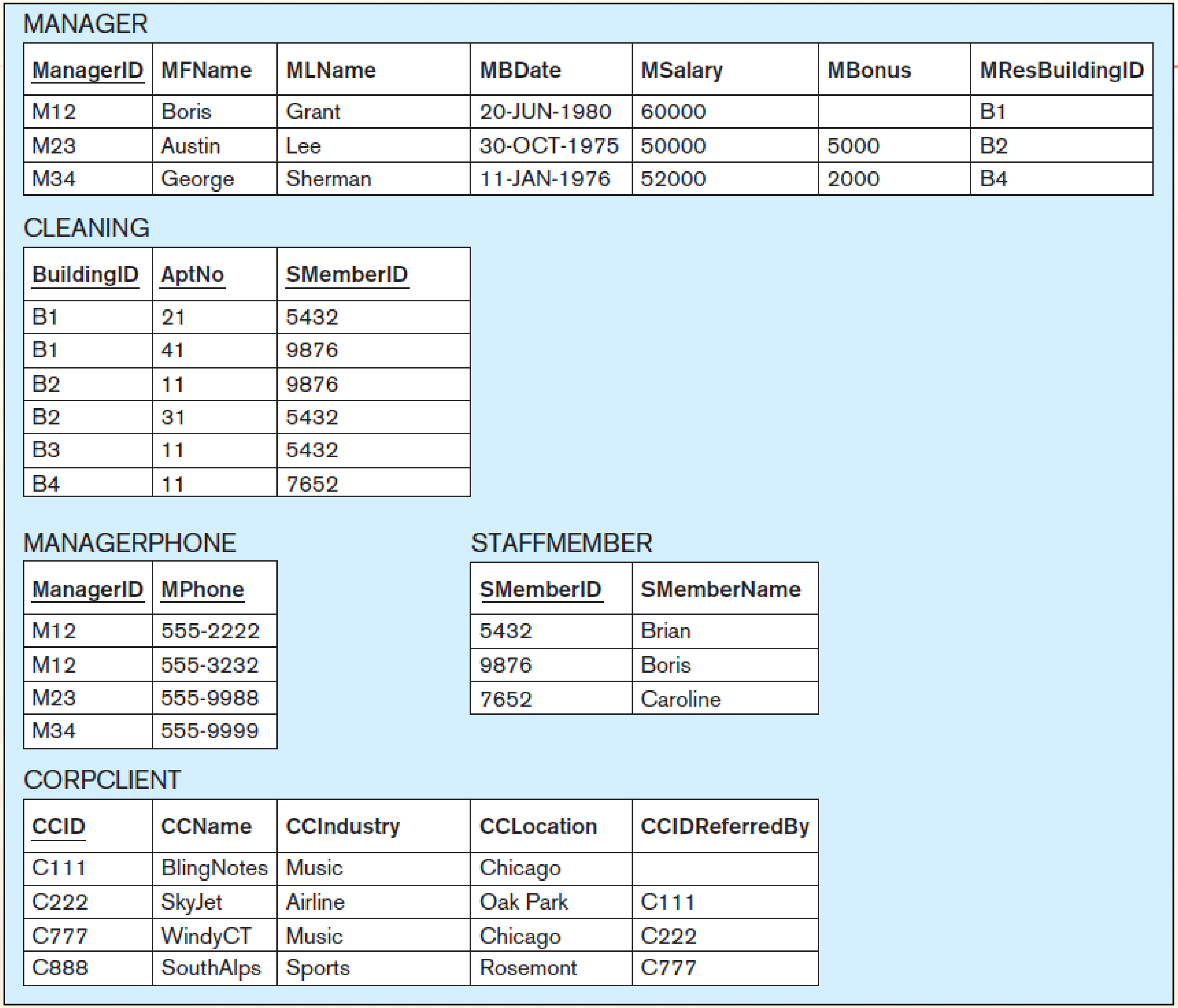
APARTMENT

BuildingID	AptNo	ANoOfBedrooms	CCID
B1	41	1	
B1	21	1	C111
B2	11	2	C222
B2	31	2	
B3	11	2	C777
B4	11	2	C777

INSPECTING

InsID	BuildingID	DateLast	DateNext
I11	B1	15-MAY-2012	14-MAY-2013
I11	B2	17-FEB-2013	17-MAY-2013
I22	B2	17-FEB-2013	17-MAY-2013
I22	B3	11-JAN-2013	11-JAN-2014
I33	B3	12-JAN-2013	12-JAN-2014
I33	B4	11-JAN-2013	11-JAN-2014

Data records: HAFH Realty Company Property Management Database (part 2)



MANAGER

ManagerID	MFName	MLName	MBDate	MSalary	MBonus	MResBuildingID
M12	Boris	Grant	20-JUN-1980	60000		B1
M23	Austin	Lee	30-OCT-1975	50000	5000	B2
M34	George	Sherman	11-JAN-1976	52000	2000	B4

CLEANING

BuildingID	AptNo	SMemberID
B1	21	5432
B1	41	9876
B2	11	9876
B2	31	5432
B3	11	5432
B4	11	7652

MANAGERPHONE

ManagerID	MPhone
M12	555-2222
M12	555-3232
M23	555-9988
M34	555-9999

STAFFMEMBER

SMemberID	SMemberName
5432	Brian
9876	Boris
7652	Caroline

CORPCLIENT

CCID	CCName	CCIndustry	CCLocation	CCIDReferredBy
C111	BlingNotes	Music	Chicago	
C222	SkyJet	Airline	Oak Park	C111
C777	WindyCT	Music	Chicago	C222
C888	SouthAlps	Sports	Rosemont	C777

INSERT INTO statements for HAFH Realty Company Property Management Database

```
INSERT INTO manager VALUES ('M12', 'Boris', 'Grant', '20/Jun/1980', 60000, null, null);
INSERT INTO manager VALUES ('M23', 'Austin', 'Lee', '30/Oct/1975', 50000, 5000, null);
INSERT INTO manager VALUES ('M34', 'George', 'Sherman', '11/Jan/1976', 52000, 2000, null);

INSERT INTO managerphone VALUES ('M12','555-2222');
INSERT INTO managerphone VALUES ('M12','555-3232');
INSERT INTO managerphone VALUES ('M23','555-9988');
INSERT INTO managerphone VALUES ('M34','555-9999');

INSERT INTO building VALUES ('B1', '5', 'M12');
INSERT INTO building VALUES ('B2', '6', 'M23');
INSERT INTO building VALUES ('B3', '4', 'M23');
INSERT INTO building VALUES ('B4', '4', 'M34');

INSERT INTO inspector VALUES ('I11', 'Jane');
INSERT INTO inspector VALUES ('I22', 'Niko');
INSERT INTO inspector VALUES ('I33', 'Mick');

INSERT INTO inspecting VALUES ('I11','B1','15/May/2012','14/May/2013');
INSERT INTO inspecting VALUES ('I11','B2','17/Feb/2013','17/May/2013');
INSERT INTO inspecting VALUES ('I22','B2','17/Feb/2013','17/May/2013');
INSERT INTO inspecting VALUES ('I22','B3','11/Jan/2013','11/Jan/2014');
INSERT INTO inspecting VALUES ('I33','B3','12/Jan/2013','12/Jan/2014');
INSERT INTO inspecting VALUES ('I33','B4','11/Jan/2013','11/Jan/2014');

INSERT INTO corpclient VALUES ('C111', 'BlingNotes', 'Music', 'Chicago', null);
INSERT INTO corpclient VALUES ('C222', 'SkyJet', 'Airline', 'Oak Park', 'C111');
INSERT INTO corpclient VALUES ('C777', 'WindyCT', 'Music', 'Chicago', 'C222');
INSERT INTO corpclient VALUES ('C888', 'SouthAlps', 'Sports', 'Rosemont', 'C777');
```

INSERT INTO statements for HAFH Realty Company Property Management Database

```
INSERT INTO apartment VALUES ('B1', '21', 1, 'C111');
INSERT INTO apartment VALUES ('B1', '41', 1, null);
INSERT INTO apartment VALUES ('B2', '11', 2, 'C222');
INSERT INTO apartment VALUES ('B2', '31', 2, null);
INSERT INTO apartment VALUES ('B3', '11', 2, 'C777');
INSERT INTO apartment VALUES ('B4', '11', 2, 'C777');

INSERT INTO staffmember VALUES ('5432', 'Brian');
INSERT INTO staffmember VALUES ('9876', 'Boris');
INSERT INTO staffmember VALUES ('7652', 'Caroline');

INSERT INTO cleaning VALUES ('B1', '21', '5432');
INSERT INTO cleaning VALUES ('B1', '41', '9876');
INSERT INTO cleaning VALUES ('B2', '11', '9876');
INSERT INTO cleaning VALUES ('B2', '31', '5432');
INSERT INTO cleaning VALUES ('B3', '11', '5432');
INSERT INTO cleaning VALUES ('B4', '11', '7652');
```

CONSTRAINT MANAGEMENT

Alter Statement 5:

```
ALTER TABLE         manager
ADD CONSTRAINT      fkresidesin
FOREIGN KEY (mresbuildingid)
       REFERENCES building (buildingid);
```

Update Statement 4:

```
UPDATE       manager
SET          mresbuildingid = 'B1'
WHERE        managerid = 'M12';
```

Update Statement 5:

```
UPDATE       manager
SET          mresbuildingid = 'B2'
WHERE        managerid = 'M23';
```

Update Statement 6:

```
UPDATE       manager
SET          mresbuildingid = 'B4'
WHERE        managerid = 'M34';
```

Alter Statement 6:

```
ALTER TABLE  manager
MODIFY       (mresbuildingid NOT NULL);
```

CONSTRAINT MANAGEMENT

DROP TABLE sequence HAFH database—First seven tables:

```
DROP TABLE cleaning;
DROP TABLE staffmember;
DROP TABLE apartment;
DROP TABLE corpclient;
DROP TABLE inspecting;
DROP TABLE inspector;
DROP TABLE managerphone;
```

Alter Statement 7:

```
ALTER TABLE manager
DROP CONSTRAINT fkresidesin;
```

DROP TABLE sequence HAFH database—Last two tables:

```
DROP TABLE building;
DROP TABLE manager;
```

SELF-JOIN

- Self-JOIN
  - A join statement that includes a relation that contains a foreign key referring to itself, and joins a relation with itself in a query

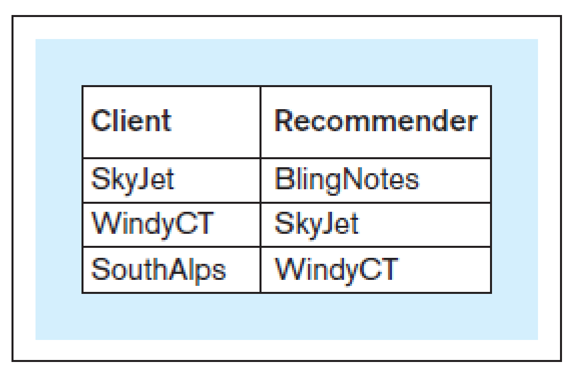
SELF-JOIN

Query 39 text: *For all corporate clients that were referred by other corporate clients, retrieve the name of the corporate client and the name of the corporate client that referred it*

Query 39:

```
SELECT c.ccname AS client, r.ccname AS recommender
FROM corpclient c, corpclient r
WHERE r.ccid = c.ccidreferredby;
```

Query 39 result:



Client	Recommender
SkyJet	BlingNotes
WindyCT	SkyJet
SouthAlps	WindyCT

OUTER JOIN

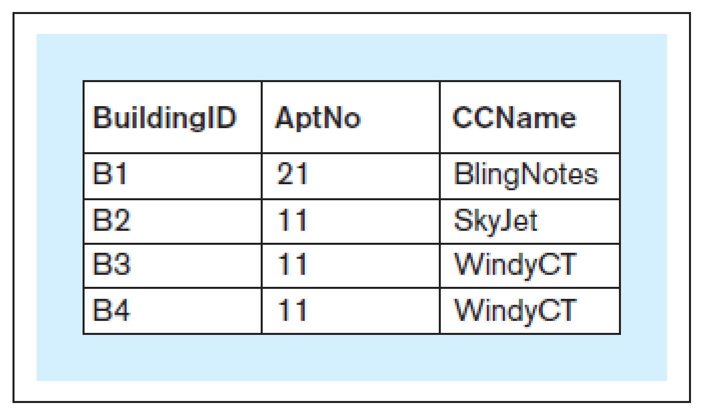
- OUTER JOIN
  - Variation of the JOIN operation that supplements the results with the records from one relation that have no match in the other relation
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN

INNER JOIN

Query 40:

```
SELECT         a.buildingid, a.aptno, c.ccname
FROM           apartment a, corpclient c
WHERE          a.ccid = c.ccid;
```

Query 40 result:



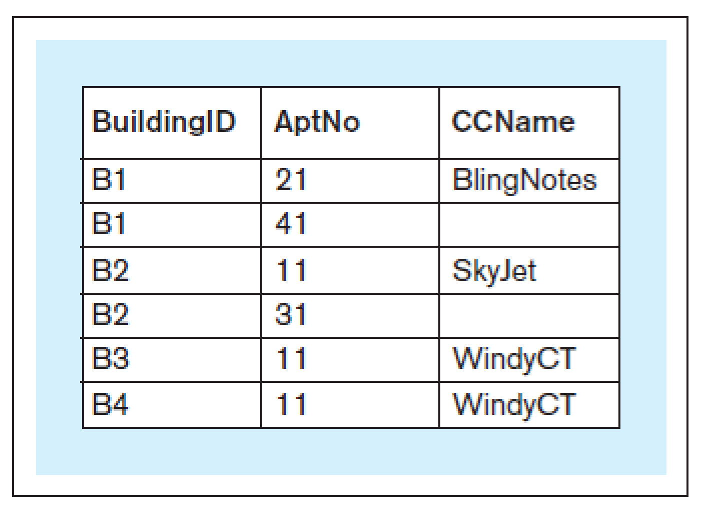
BuildingID	AptNo	CCName
B1	21	BlingNotes
B2	11	SkyJet
B3	11	WindyCT
B4	11	WindyCT

OUTER JOIN

Query 41:

```
SELECT a.buildingid, a.aptno, c.ccname
FROM   apartment a LEFT OUTER JOIN corpclient c
ON     a.ccid = c.ccid;
```

Query 41 result:



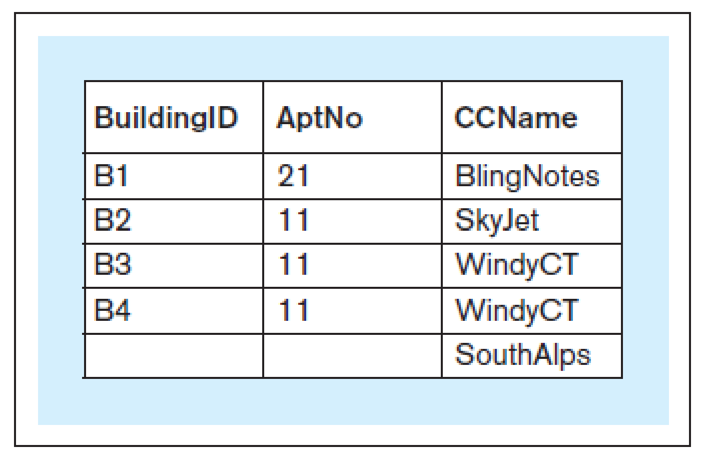
BuildingID	AptNo	CCName
B1	21	BlingNotes
B1	41	
B2	11	SkyJet
B2	31	
B3	11	WindyCT
B4	11	WindyCT

OUTER JOIN

Query 42:

```
SELECT a.buildingid, a.aptno, c.ccname
FROM   apartment a RIGHT OUTER JOIN corpclient c
ON     a.ccid = c.ccid;
```

Query 42 result:



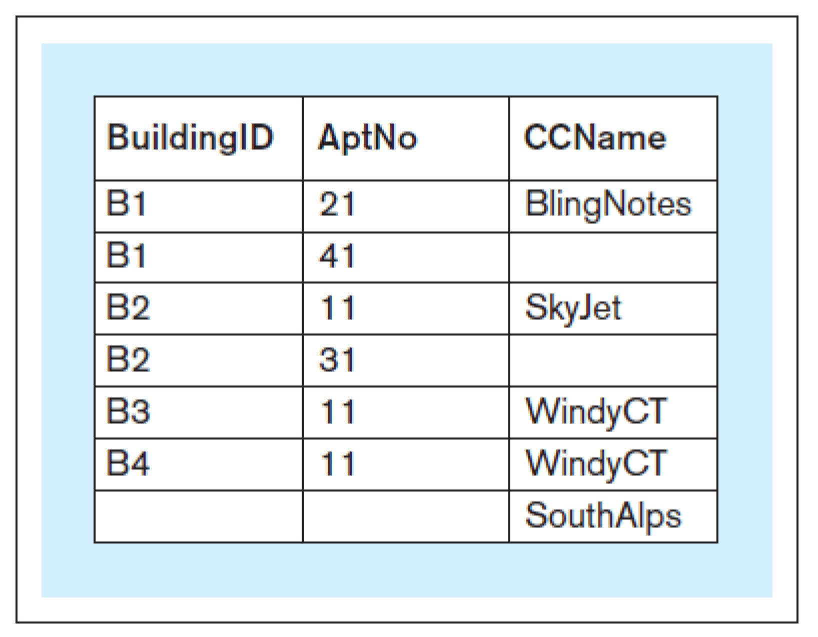
BuildingID	AptNo	CCName
B1	21	BlingNotes
B2	11	SkyJet
B3	11	WindyCT
B4	11	WindyCT
		SouthAlps

OUTER JOIN

Query 43:

```
SELECT a.buildingid, a.aptno, c.ccname
FROM   apartment a FULL OUTER JOIN corpclient c
ON     a.ccid = c.ccid;
```

Query 43 result:



BuildingID	AptNo	CCName
B1	21	BlingNotes
B1	41	
B2	11	SkyJet
B2	31	
B3	11	WindyCT
B4	11	WindyCT
		SouthAlps

JOIN WITHOUT USING A PRIMARY KEY/ FOREIGN KEY COMBINATION

- **Join without using a primary key/foreign key combination**
  - It is possible to join two tables without joining a foreign key column in one table with a primary key column in another table.
  - A JOIN condition can connect a column from one table with a column from the other table as long as those columns contain the same values.

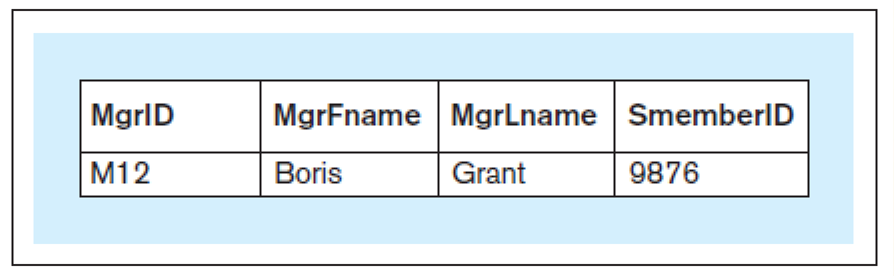
JOIN WITHOUT USING A PRIMARY KEY/FOREIGN KEY COMBINATION

Query 44 text: *For each manager who has a staff member with the same name as the manager's first name, show the manager's ID, first name, and last name and the ID of the staff members who have the same name as the manager's first name*

Query 44:

```
SELECT m.managerid, m.mfname, m.mlname, s.smemberid
FROM   manager m, staffmember s
WHERE  m.mfname = s.smembername;
```

Query 44 result:



MgrID	MgrFname	MgrLname	SmemberID
M12	Boris	Grant	9876

IS NULL

- IS NULL
  - Used in queries that contain comparisons with an empty value in a column of a record

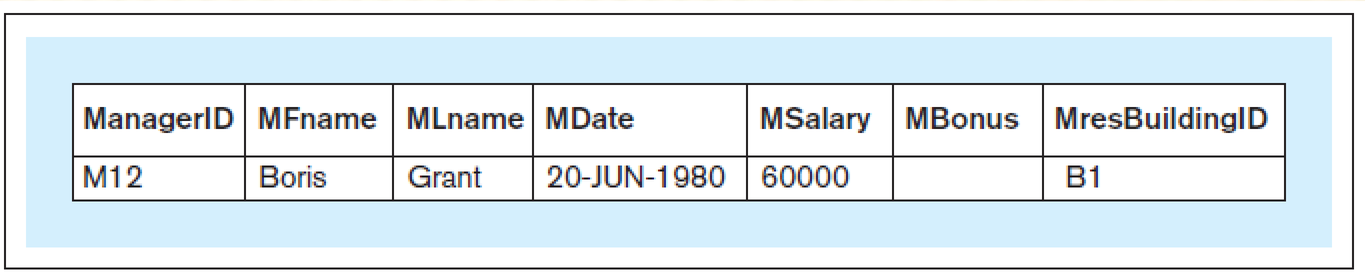
IS NULL

Query 45 text: *Retrieve records for all managers who do not have a bonus*

Query 45:

```
SELECT      *
FROM        manager
WHERE       mbonus IS NULL;
```

Query 45 result:



ManagerID	MFname	MLname	MDate	MSalary	MBonus	MresBuildingID
M12	Boris	Grant	20-JUN-1980	60000		B1

EXISTS

- EXISTS
  - In queries where the inner query (nested query) uses columns from the relations listed in the SELECT part of the outer query, the inner query is referred to as a **correlated subquery**
  - In such cases, the EXISTS operator can be used to check if the result of the inner correlated query is empty

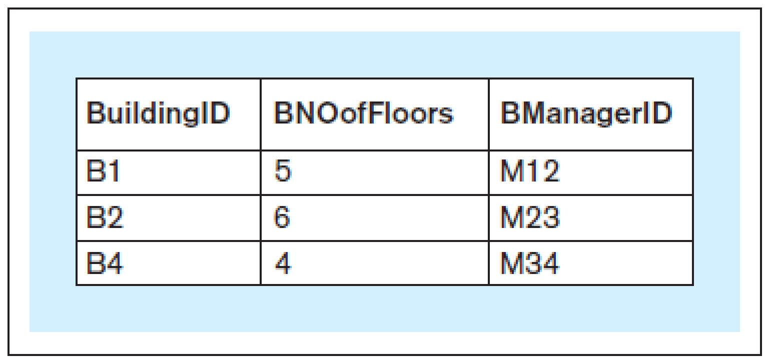
EXISTS

Query 46 text: *Retrieve records for all buildings that have managers living in them*

Query 46:

```
SELECT *
FROM   building b
WHERE  EXISTS
       (SELECT *
       FROM manager m
       WHERE b.buildingid = m.mresbuildingid);
```

Query 46 result:



BuildingID	BNOofFloors	BManagerID
B1	5	M12
B2	6	M23
B4	4	M34

NOT

- NOT
  - Can be used in conjunction with the condition comparison statements returning the Boolean values TRUE or FALSE

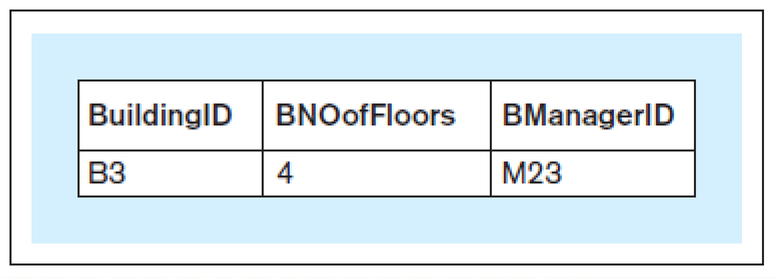
NOT

Query 47 text: *Retrieve records for all buildings that do not have managers living in them*

Query 47:

```
SELECT *
FROM   building b
WHERE  NOT EXISTS
       (SELECT *
       FROM manager m
       WHERE b.buildingid = m.mresbuildingid);
```

Query 47 result:



BuildingID	BNOofFloors	BManagerID
B3	4	M23

INSERTING FROM A QUERY

- Inserting from a query
  - A query retrieving the data from one relation can be used to populate another relation

INSERTING FROM A QUERY

Create Table Statement 1:

```
CREATE TABLE cleaningdenormalized
(      buildingid    CHAR(3)       NOT NULL,
       aptno         CHAR(5)       NOT NULL,
       smemberid     CHAR(4)       NOT NULL,
       smembername   VARCHAR(15)   NOT NULL,
       PRIMARY KEY (buildingid, aptno, smemberid));
```

Insert Statement 2:

```
INSERT INTO cleaningdenormalized
SELECT c.buildingid, c.aptno, s.smemberid, s.smembername
FROM   cleaning c, staffmember s
WHERE  c.smemberid = s.smemberid;
```

INAPPROPRIATE USE OF OBSERVED VALUES IN SQL

- Inappropriate use of Observed Values in SQL
  - A common beginner’s SQL mistake occurs when novice user creates a simplistic query that produces the correct result by inappropriately using observed values

INAPPROPRIATE USE OF OBSERVED VALUES IN SQL

Request A *For each product that has more than three items sold within all sales transactions, retrieve the product id, product name, and product price*

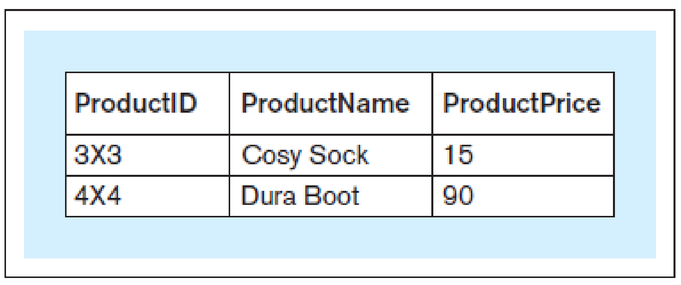
SQL Query A:

```
SELECT          productid, productname, productprice
FROM            product
WHERE           productid IN
                (SELECT productid
                FROM soldvia
                GROUP BY productid
                HAVING SUM(noofitems) > 3);
```

SQL Query B:

```
SELECT          productid, productname, productprice
FROM            product
WHERE           productid IN ('3X3','4X4');
```

Query A and B Result:



ProductID	ProductName	ProductPrice
3X3	Cosy Sock	15
4X4	Dura Boot	90

SQL STANDARD AND SQL SYNTAX DIFFERENCES

- SQL Standard
  - SQL became the standard language for querying data contained in a relational database

SQL STANDARD AND SQL SYNTAX DIFFERENCES

- **SQL standard and SQL syntax differences**
  - Minor SQL syntax differences exist in SQL implementations in various popular RDBMS packages, such as differences in:
    - DATE and TIME data types
    - FOREIGN KEY syntax
    - Usage of AS keyword with aliases
    - ALTER TABLE syntax
    - Set operators
    - FULL OUTER JOIN implementation
    - Constraint management
    - GROUP BY restrictions