2 Database Language

Chapter Objectives

Database languages are necessary to use databases. SQL was developed for the use of relational databases, and has been standardized by ISO and JIS, and is currently in wide use.
In this chapter, we learn the method of using SQL to define tables and databases and to manipulate databases.

① Understanding the outline of database languages such as NDL and SQL.
② Understanding SQL structure, definitions of 'database,' 'schema,' 'table,' and 'view,' as well as database creation procedures including data control and entry.
③ Understanding data manipulation using SQL to be able to express the required processing using SQL.
④ Understanding the process of embedding SQL statements in application programs and cursor manipulation.

2.1 What are Database Languages?

A database language is used to define database schemata and refer to the actual data. SQL (Structured Query Language) and NDL are representative database languages.

- SQL : A database language for relational databases. Its standard specifications were established by ISO (International Organization for Standardization). SQL was also standardized as JIS X 3005 in Japan.
- NDL : A database language for CODASYL (network) databases. It was introduced by CODASYL, and standardized as JIS X 3004 in Japan.

Database languages are classified into the following three groups according to the users' standpoint and the purposes:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- End User Language (EUL)

2.1.1 Data Definition Language

The Data Definition Language, as its name signifies, is a language that defines databases. "Database definition" means the definition of the schema. Data Definition Language is broadly classified into two languages: the schema definition language used by a database administrator (DBA) to define the whole picture of the database (conceptual schema), and the subschema definition language that defines external schemata by the user.

2.1.2 Data Manipulation Language

The Data Manipulation Language is used to actually operate databases. This language is used on the creation side of the database system (programmers, etc.).

2.1.3 End User Language

The End User Language is a simple query language designed for general database users (end users). This language is generally used based on the interactive processing by using tables and simple commands.

2.2 SQL

2.2.1 SQL: Database Language

SQL (Structured Query Language) is a language to manipulate databases based on the relational data model.
SQL is designed to process relational databases (RDB) in which data are expressed in the table format, and can create, manipulate, update, and delete data in tables. Because SQL is a non-procedural language which does not require a description of every procedure in the programs, its statements are simple and easy to understand.
In addition to concrete statements on access to the tables, SQL can grant access authority to a specific person to define and manipulate the table.
The prototype of SQL was called SEQUEL (Structured English Query Language) originating as a language to access database "System R." It was developed as the relational database in 1979 at the San Jose Research Laboratory of IBM. After ISO established standard specifications for SQL in 1987, SQL was standardized by JIS as "JIS X3005-1995" in Japan.

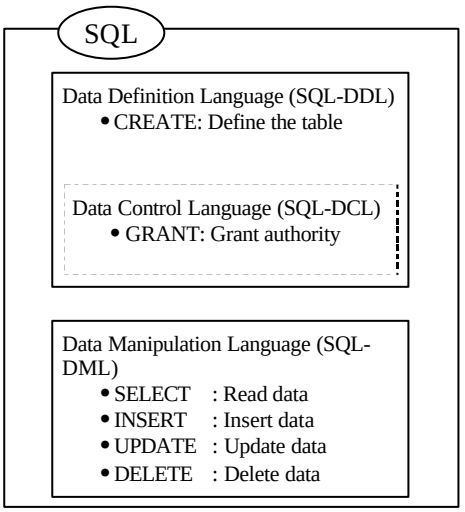
2.2.2 Structure of SQL

SQL is a complete database language to process relational databases, and can create, manipulate, update, and delete tables. It consists of the following languages (Figure 2-2-1):

- Data Definition Language (SQL-DDL)
- Data Control Language (SQL-DCL)
- Data Manipulation Language (SQL-DML)

The Data Control Language (DCL), a language to grant access authority to tables, is sometimes included in the category of the Data Definition Language.

Figure 2-2-1
What is SQL?



SQL can be used in a host language system (embedded SQL) and also as a self-contained system (interactive SQL).

- Host language system
  The host language system is a system to manipulate databases by programming languages. It performs processing by embedding SQL statements in programming languages such as COBOL and FORTRAN. → Embedded SQL
- Self-contained system
  The self-contained system is a system to manipulate databases only by the database manipulation language, independent of programming languages. Users perform interactive processing with terminals, using SQL. → Conversational SQL
  In the DBMS for personal computers, the instructions issued by users are converted into SQL statements (SQL - DML) and executed inside the DBMS by the query function (QBE: Query By Example).

2.3 Database Definition, Data Access Control and Loading

2.3.1 Definition of Database

To use a database, the database must be defined based on the database design. Specifically, the database can be defined by defining various schemata.
The following is an explanation of a database definition, taking Figure 2-3-1 as an example:

Figure 2-3-1 Normalized Data Tables

customer_table	customer_number	customer_name	customer_address
	C005	Tokyo Shoji	Kanda, Chiyoda-ku
	D010	Osaka Shokai	Doyama-cho, Kita-ku, Osaka-City
	G001	Chugoku Shoten	Moto-machi, Naka-ku, Hiroshima-City
	(4-digit character string) CHAR (4)	(10-digit kanji string) NCHAR (10)	(20-digit kanji string) NCHAR (20)

order_table	customer_number	order_slip_number	order_receiving_date
	C005	2001	08/07/1999
	C005	2002	09/01/1999
	D010	2101	07/28/1999
	G001	2201	09/10/1999
	(4-digit character string) CHAR (4)	(4-digit numeric value) INT	(Year/Month/Date (Christian era)) DATE

order_detail_table	customer_number	order_slip_number	raw_number	merchandise_number	quantity
	C005	2001	01	PR1	20
	C005	2001	02	PX0	15
	C005	2002	01	Q91	10
	C005	2002	02	S00	5
	D010	2101	01	PX0	30
	D010	2101	02	S00	6
	(4-digit character string) CHAR (4)	(4-digit numeric value) INT	(2-digit numeric value) SMALLINT	(3-digit character string) CHAR (3)	(3-digit numeric value) DEC (3)

merchandise_table	merchandise_number	merchandise_name	unit_price
	PR1	Printer 1-type	300
	PX0	Printer X-type	550
	Q91	Disk 1-type	910
	S00	System 0-type	4500
	(3-digit character string) CHAR (3)	(10-digit kanji string) NCHAR (10)	(5-digit numeric value) DEC (5)

2.3.2 Definition of Schema

(1) What is a schema?

Database definition information is called a schema. A schema is specified by the schema definition statement of the data definition language (SQL-DDL). The definition of the schema consists of the definitions of the table, view, and authorization.
The definition information related to the schemata is automatically registered in DD/D (Database Dictionary/Directory) by the DBMS.

(2) Authorization identifier

When defining a schema, it is necessary to know the person who defines the schema, so that the person can be identified. The schema authorization identifier is used for that purpose. The user who has the authorization identifier is granted authorization to process the tables and views created in the schema. As a user who does not have the authorization identifier cannot gain access to the database, the authorization identifier also serves as a protection of the database. In interactive processing in network systems, in many cases, the authorization identifier also serves as a user ID.
The schema authorization identifier is specified by the CREATE SCHEMA statement of SQL-DDL.

Definition of the schema (authorization identifier)

```
CREATE  SCHEMA
        AUTHORIZATION authorization_identifier
```

When the authorization identifier is specified as DRY, for example, the definition is as follows:

```
CREATE    SCHEMA
          AUTHORIZATION    DRY
```

2.3.3 Definition of Table

(1) Table_name

The actual data are stored in a table. A table has a two-dimensional structure consisting of rows and columns. In contrast to a view (virtual table) described later, a table is also called an "actual table." Although multiple tables can exist, the same table_name must be avoided because each table is identified by the table_name.
The definition of the table is specified by the CREATE TABLE statement of SQL-DDL.

Definition of the table

```
CREATE  TABLE    table_name
```

(2) Data type

A table consists of rows (tuples) and columns (attributes). To define the table, attributes (data type) must be defined.

Definition of the data type

```
column_name    data_type
```

Figure 2-3-2 shows the data types that can be defined by SQL. Note the extended functionalities of the SQL language provided by each vendor.

Figure 2-3-2
Data type

Data type	Definition	Contents
Character string type	CHARACTER	Also described as CHAR. A fixed-length character string with a specified length. Up to 255 characters.
Numeric value type	INTEGER	Also described as INT. An integer with a specified number of digits. 4-byte binary numeric value
	SMALLINT	A short integer with a specified number of digits. The precision contains fewer digits than INT. 2-byte binary numeric value
	NUMERIC	A numeric value with the decimal part and the integer part with a specified number of digits.
	DECIMAL	Also described as DEC. A numeric value with the decimal part and the integer part with a specified number of digits. A decimal number with up to 15-digit precision.
	FLOAT	A numeric value expressed by a binary number with a specified number of digits or smaller. Floating-point binary number
	REAL	Single-precision floating-point number
	DOUBLE PRECISION	Double-precision floating-point number
Kanji string type	NATIONAL CHARACTER	Also described as NCHAR. A kanji string with a specified length. Up to 128 characters.
Date type	DATE	Described in the format of Year/Month/Day (Christian Era)

In the definition of the data type of a database, "null values" can be set. A null value means "no value" or "the undecided value." When defining the data type, decide whether the use of null values is allowed or not. If the use of null values is not allowed for fields that contain data such as key items, specify "NOT NULL." As described later, the null value can be used as a query condition.

(3) PRIMARY KEY

In a table, the attribute to be a record key item is specified as a primary key. The primary key is defined by PRIMARY KEY clause in the SQL language.
When the record key is a concatenated key, column names are successively combined.

Definition of the primary key

```
PRIMARY KEY column_name
```

(4) FOREIGN KEY

The foreign key is a data item not used as a record key in a table, but used as a record key (primary key) in other tables. In the SQL language, the foreign key is defined by FOREIGN KEY clause and the tables in which the foreign key is used as a record key (primary key) are specified.

Definition of the foreign key

```
FOREIGN KEY    column_name
        REFERENCES    table_name
```

The definitions of the four tables in Figure 2-3-1 are as follows:

- Customer_table

```
CREATE TABLE customer_table
        (customer_number      CHAR (4)      NOT NULL,
        customer_name         NCHAR (10)    NOT NULL,
        customer_address      NCHAR (20)    NOT NULL,
        PRIMARY KEY (customer_number))
```

- Order_table

```
CREATE TABLE order_table
        (customer_number      CHAR (4)      NOT NULL,
        order_slip_number     INT           NOT NULL,
        order_receiving_date  DATE          NOT NULL,
        PRIMARY KEY (customer_number, order_slip_number),
        FOREIGN KEY (customer_number) REFERENCES customer_table)
```

- Order_detail_table

```
CREATE TABLE order_detail_table
        (customer_number      CHAR (4)      NOT NULL,
        order_slip_number     INT           NOT NULL,
        row_number            SMALLINT      NOT NULL,
        merchandise_number    CHAR (3)      NOT NULL,
        quantity              DEC (3),
        PRIMARY KEY (customer_number, order_slip_number, row_number),
        FOREIGN KEY (customer_number, order_slip_number) REFERENCES order_table,
        FOREIGN KEY (merchandise_number) REFERENCES merchandise_table)
```

- Merchandise_table

```
CREATE TABLE merchandise_table
        (merchandise_number   CHAR (3)      NOT NULL,
        merchandise_name      NCHAR (10)    NOT NULL,
        unit_price            DEC (5)       NOT NULL,
        PRIMARY KEY (merchandise_number))
```

2.3.4 Characteristics and Definition of View

(1) Characteristics of a view

A view is a look at part of an actual table or a virtual table, which combines necessary data items from multiple tables. One of the advantages of the relational data model over other data models is that it uses views. As views can be freely created depending on the situation, they are adaptable to routine operations as well as ad hoc operations.

Under certain restrictions, you can perform various data operations such as query and update of data with a view like with a table. Update of data, however, cannot be performed for a view created from multiple tables. When there is any change in the data of the original table, the change results can be immediately reflected in the view.

Use of a view enables the following:

- Increase in usability
  By creating a new table (view) by extracting necessary columns from a table, the readability of the data in the table is improved. You can create a new table by combining multiple tables. SQL statements for these views become simpler than the ones for the original table.
- Security enhancement by limiting the data utilization range
  By creating a view from the specified rows or columns and granting access privileges to the view, the data utilization range is limited and security can be enhanced.
- Increased independence from data
  Even if the definition of the original table is changed (for example, addition of rows or division of a table), instructions to operate the view need not be changed.

(2) Definition of a view

When defining a view, a view name which is distinct from table_names and other view names in the same schema must be given to the view.
In the SQL language, a view is specified by the CREATE VIEW statement.

Definition of a view

```
CREATE VIEW view_name
       AS SELECT column_name FROM table_name
```

For example, the statement "define a view named 'customer_name table' consisting only of customer_numbers and customer_names from the customer_table" is given as follows:

```
CREATE VIEW customer_name_table
       AS SELECT customer_number, customer_name FROM customer_table
```

2.3.5 Data Access Control

Data access control means limiting persons who can manipulate the database (table) by granting access privileges.
When a table is used frequently in a database, the data may be destroyed intentionally or by accident. To prevent such destruction, users of the table should be limited by granting access privileges.
There are five types of access privileges:

- SELECT privilege to read data
- INSERT privilege to insert data
- DELETE privilege to delete data
- UPDATE privilege to update data
- REFERENCE privilege to redefine the table

These five privileges are automatically granted to the creator of the table. Specifying ALL PRIVILEGES means granting all privileges. The REVOKE statement on the other hand, is used to cancel the granted privileges.
When granting privileges to specified persons, the GRANT statement is used in SQL.

Granting privileges

```
GRANT privilege ON table_name TO authorization_identifier
```

For example, the statement "grant the ability (privilege) to read a customer_table to the person who has the authorization identifier WET" is given as follows:

```
GRANT SELECT ON customer_table TO WET
```

2.3.6 Data Loading

After defining the database, data must be loaded into the table actually defined.
There are three data loading methods:

(1) Interactive system

In the interactive system, data are loaded line by line using the INSERT statement of SQL in the self-contained system. Details are described later.
Because the data are loaded line by line, this system is not suitable for loading of large amounts of data.

(2) Host language system

In this system, data prepared separately are loaded using embedded SQL. In this case, it is necessary to prepare a data loading program by embedding an SQL statement (INSERT) beforehand (the method to embed an SQL statement is described later).
The host language system is suitable for loading data while processing separately prepared data or selecting data under certain conditions.

(3) Utility program system

In the utility program system, data prepared separately are loaded using a utility program (load utility). This method is suitable for simply loading large amounts of data without manipulating the prepared data.

2.4 Database Manipulation

2.4.1 Query Processing

Users who have been granted privileges by the GRANT statement can gain access to the table within the permitted range. Query means reading the data in tables.

(1) Basic syntax

Reading the data in tables is the most frequently performed data manipulation in the relational database processing, and it is performed by using the SELECT statement.

Data retrieval

```
SELECT    column_name        : Specify the column to retrieve
  FROM    table_name         : Specify the table to read
```

For example, the statement "retrieve customer_numbers and customer_names from the customer_table" is expressed as follows:

```
SELECT    customer_number,    customer_name
  FROM    customer_table
```

<Display result>

customer_number	customer_name
C005	Tokyo Shoji
D010	Osaka Shokai
G001	Chugoku Shoten

The column_names in the SELECT statement must be separated by a comma, and specified in the preferred order of display.
Multiple table_names can be specified in the FROM clause. Details are described later.
If the SELECT statement is specified as follows, all the columns to be read are displayed in the order of columns specified in the table definition.

```
SELECT    *      FROM    customer_table
```

<Display result>

customer_number	customer_name	customer_address
C005	Tokyo Shoji	Kanda, Chiyoda-ku
D010	Osaka Shokai	Doyama-cho, Kita-ku, Osaka City
G001	Chugoku Shoten	Moto-machi, Naka-ku, Hiroshima City

"Retrieve customer_numbers from the order_table" is expressed as follows:

```
SELECT    customer_number    FROM    order_table
```

<Display result>

customer_number
C005
C005
D010
G001

The above display result does not include mistakes. However, if you want to avoid displaying the records of the same contents (C005), use DISTINCT to eliminate the duplicate data.

```
SELECT DISTINCT customer_number FROM order_table
```

<Display result>

customer_number
C005
D010
G001

Exercise 1. Write an SQL statement to extract the following display result from the merchandise_table.

<Display result>

merchandise_name	unit_price
Printer 1-type	300
Printer X-type	550
Disk 1-type	910
System 0-type	4500

(Answer 1)

```
SELECT merchandise_name, unit_price FROM merchandise_table
```

Exercise 2. What is the display result when the following SQL statement is executed?

```
SELECT DISTINCT customer_number, order_slip_number FROM order_detail_table
```

(Answer 2)

<Display result>

customer_number	order_slip_number
C005	2001
C005	2002
D010	2101

---

(2) Query using conditional expression

The conditional query is an inquiry retrieving the specified rows under certain conditions. The conditions used to retrieve the rows are defined using the WHERE clause.

Conditional query

```
SELECT row_name
  FROM table_name
 WHERE query_conditions (the conditional to specify the rows to be selected)
```

Query_conditions are described in the form of expression using operators. The following are the representative operators used in the conditional expression.

- Comparison_operator (relational operator)
- Logical operator
- Character string comparison operator
- Null value operator

① Comparison_operator (relational operator)

The comparison operator, also called the "relational operator", is used to compare numeric type and character type data. The following operators are used in SQL.

- Equal (=)
- Larger than (>)
- Smaller than (<)

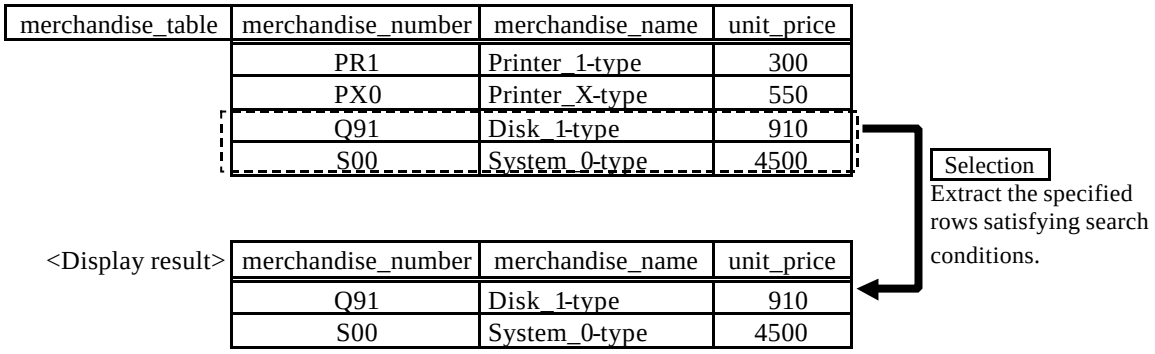
- Equal to or larger than (> =)
- Equal to or smaller than (< =)

In the SQL syntax, the form of "row_name comparison_operator value" is used in the WHERE clause. Selection and projection of relational algebra using comparison operators are written in the SQL as follows:

a. Selection

Selection is a manipulation to extract the rows satisfying query_conditions from the table.
For example, the statement "retrieve from merchandise_table the records whose unit_price is ¥800 or higher" is written as follows:

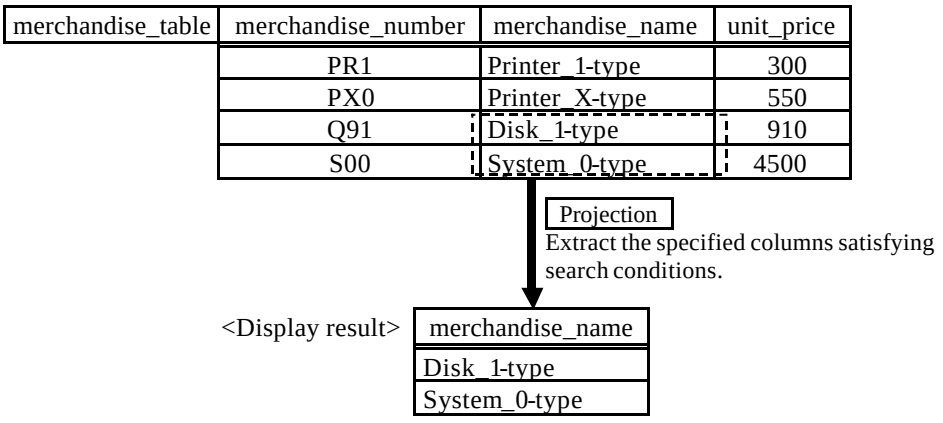
```
SELECT  *  FROM  merchandise_table
  WHERE  unit_price  > =  800
```



b. Projection

Projection is a manipulation to extract the columns satisfying query_conditions from the table.
For example, the statement "retrieve from merchandise_table the merchandise_names in the records whose unit_price is ¥800 or higher" is expressed as follows:

```
SELECT  merchandise_name  FROM  merchandise_table
  WHERE  unit_price  > =  800
```



Values in the conditional expression must agree with the data type of the column. Numeric type data are described only by numeric values, and character type data are surrounded by quotation marks (').
Kanji type data are surrounded by quotation marks, adding N (meaning national character) before the string.

[Character type (CHAR)]
For example, the statement "retrieve from the merchandise_table the merchandise_name and its price in the record whose merchandise_number is PR1" is expressed as follows:

```
SELECT  merchandise_name, unit_price  FROM  merchandise_table
  WHERE  merchandise_number  =  'PR 1'
```

[Kanji type (NCHAR)]
For example, the statement "retrieve from the merchandise_table the records whose merchandise_number is printer_1-type" is expressed as follows:

```
SELECT * FROM merchandise_table
  WHERE merchandise_number = Printer_1-type'
```

Exercise 3. Write an SQL statement meaning "retrieve from the order_detail_table the customer_numbers and the merchandise_numbers in the records whose quantity is less than 20."

<Display result>

customer_number	merchandise_number
C005	PX0
C005	Q91
C005	S00
D010	S00

(Answer 3)

```
SELECT customer_number, merchandise_number FROM order_detail_table
  WHERE quantity < 20
```

Exercise 4. As a result of the execution of an SQL statement, the following result was displayed. Write the executed SQL statement.

<Display result>

customer_number	order_slip_number
C005	2002
G001	2201

(Answer 4)
The tables including both the "customer_number" and "order_slip_number" are the "order_table" and the "order_detail_table." Of these two tables, only the "order_table" includes the customer_number 'G001'. Therefore, the SELECT statement is executed for the "order_table."
The condition common to the selected two records is that the order_receiving_date is 'after January 1999'. Therefore, the SQL statement is as follows:

```
SELECT customer_number, order_slip_number FROM order_table
  WHERE order_receiving_date > = '99/01/01'
```

---

② Logical operator

The logical operator, also called the "Boolean operators," is used to combine conditional expressions consisting of the above-mentioned comparison operators. The following operators are used in SQL.

- AND
- OR
- NOT

For example, the statement 'retrieve from the merchandise_table the merchandise_names and prices in the records whose unit_price is ¥500 to ¥1,000" is expressed as follows:

```
SELECT merchandise_name, unit_price FROM merchandise_table
WHERE unit_price >= 500 AND unit_price <= 1000
```

<Display result>

merchandise_name	unit_price
Printer_X-type	550
Disk_1-type	910

In the SQL, the SELECT statement shown above can also be expressed using the BETWEEN predicate.

column_name BETWEEN - AND - (equal to or larger than - and equal to or smaller than -)

Thus, a statement to "display the merchandise_names and prices in the records whose unit_price is ¥500 to ¥1,000" mentioned above can also be expressed as follows:

```
SELECT merchandise_name, unit_price FROM merchandise_table
  WHERE unit_price BETWEEN 500 AND 1000
```

Exercise 5. Write SQL statements for ① to ③ below, and display their results.

① "Retrieve from the customer_table the customer_names in the records whose customer_number is C005 or G001."

② "Retrieve from the order_detail_table the order_slip_numbers and the merchandise numbers in the records whose customer_number is C005 and whose quantity is 10 or larger."

③ " Retrieve from the order_table the customer_numbers in the records whose order_slip_number is 2100 to 2199."

(Answer 5)

①
```
SELECT customer_name FROM customer_table
  WHERE customer_number = 'C005' OR customer_number = 'G001'
```

<Display result>

customer_name
Tokyo Shoji
Chugoku Shoten

② SELECT order_slip_number merchandise number FROM order_detail_table
WHERE customer_number = 'C005' AND quantity >= 10

<Display result>

order_slip_number	merchandise_number
2001	PR1
2001	PX0
2002	Q91

③ SELECT customer_number FROM order_table
WHERE order_slip_number BETWEEN 2100 AND 2199

<Display result>

customer_number
D010

Exercise 6. Show the retrieved results when SQL statements ① to ⑥ are executed. If no result is obtained, answer "none."

①
```
SELECT * FROM order_detail_table
  WHERE customer_number = 'C005' AND row_number = 02 AND quantity > 10
```

② SELECT * FROM order_detail_table
WHERE customer_number = 'C005' OR row_number = 02 OR quantity > 10

③ SELECT * FROM order_detail_table
WHERE customer_number = 'C005' AND row_number = 02 OR quantity > 10

④ SELECT * FROM order_detail_table
WHERE customer_number = 'C005' AND (row_number = 02 OR quantity > 10)

⑤ SELECT * FROM order_detail_table
WHERE customer_number = 'C005' OR row_number = 02 AND quantity > 10

⑥ SELECT * FROM order_detail_table
WHERE (customer_number = 'C005' OR row_number = 02) AND quantity > 10

(Answer 6)

① <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	02	PX0	15

② <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15
C005	2002	01	Q91	10
C005	2002	02	S00	5
D010	2101	01	PX0	30
D010	2101	02	S00	6

③ <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15
C005	2002	02	S00	5
D010	2101	01	PX0	30

④ <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15
C005	2002	02	S00	5

⑤ <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15
C005	2002	01	Q91	10
C005	2002	02	S00	5

⑥ <Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15

③ Character string comparison operator

In SQL, the LIKE predicate is used to compare character strings such as "begin with …," "end with …," and "include … in the middle." For actual specifications, % (percent sign wildcard) or _ (underscore wildcard) are used. % matches any sequence of zero or more characters, and _ matches any single character.

For example, to express a character string code beginning with A, the following two specification methods can be used. However, you should note that these two methods have different meanings.

- A_ _ : A 3-character code beginning with A
- A% : A code beginning with A (any number of characters is acceptable)

The LIKE predicate can be used only for the character type (double-byte kanji, etc.).

For example, the statement "Retrieve from the customer_table the records whose customer_address is Nagoya City" is created as follows:

```
SELECT customer_number, customer_name, customer_address FROM customer_table
  WHERE customer_address LIKE 'Nagoya City %'
```

<Display result>

customer_number	customer_name	customer_address

* In this case, no record is displayed because the customer_table includes no customers whose address is Nagoya City.

For example, the statement "Retrieve from the merchandise_table the records whose merchandise_number begins with P" is written as follows:

```
SELECT * FROM merchandise_table
  WHERE merchandise_number LIKE 'P_ _'
```

<Display result>

Merchandise_number	merchandise_name	unit_price
PR1	Printer_1-type	300
PX0	Printer_X-type	550

Exercise 7. Write SQL statements for ① and ② below, and show the results.

① "Retrieve the merchandise_numbers and quantities in the records the second digit of whose merchandise_number is 0."

② "Retrieve the merchandise_numbers and unit_prices in the records whose merchandise_name includes '1'."

(Answer 7)

①
```
SELECT merchandise_number, quantity FROM order_detail_table
  WHERE merchandise number LIKE '_0_'
```

<Display result>

merchandise_number	quantity
S00	5
S00	6

②
```
SELECT merchandise_number, unit_price FROM merchandise_table
  WHERE merchandise_name LIKE N'%1%'
```

<Display result>

merchandise_number	unit_price
PR1	300
Q91	910

④ Null value operator

If a null value (NULL) is allowed in the table, the null value can be used as a query condition. In that case, the IS NULL statement is used in SQL.
For example, the statement "Retrieve from the order_detail_table the order_slip_numbers and the row_numbers in the records whose quantity is null" is created as follows:

```
SELECT  order_slip_number,  row_number  FROM  order_detail_table
  WHERE  quantity  IS  NULL
```

<Display result>

order_slip_number	row_number

When NULL is used as a query condition, it must be IS NULL instead of = NULL.
This is because it is impossible to compare a NULL value, and = NULL becomes an error.

(3) Aggregation and sorting of data

① Grouping and the aggregate functions (column functions)

The aggregate functions, also called "column functions," is used to process grouped column data. There are the following aggregate functions:

- SUM (column_name) : Return the sum in the numeric column
- AVG (column_name) : Return the average in the numeric column
- MIN (column_name) : Return the minimum value in the numeric column
- MAX (column_name) : Return the maximum value in the numeric column
- COUNT (*) : Count the number of rows satisfying the condition.
- COUNT (DISTINCT column_name) : Count the number of rows satisfying the condition, excluding duplication.

All these aggregate functions perform calculations for the specified group in the specified column. In SQL, an aggregate function and a GROUP BY clause for grouping are combined.
For example, the statement "calculate the sum of order quantities by merchandise number from the order_detail_table, and display" is expressed as follows:

```
SELECT  merchandise_number,  SUM (quantity)  FROM  order_detail_table
  GROUP  BY  merchandise_number
```

Figure 2-4-1 Grouping

order_detail_table

customer_number	order_slip_number	row_number	merchandise_number	quantity
C005	2001	01	PR1	20
C005	2001	02	PX0	15
C005	2002	01	Q91	10
C005	2002	02	S00	5
D010	2101	01	PX0	30
D010	2101	02	S00	6

Grouping
GROUP BY merchandise_number

merchandise_number	quantity
PR1	20
PX0	15
PX0	30
Q91	10
S00	5
S00	6

SUM (quantity)

<Display result>

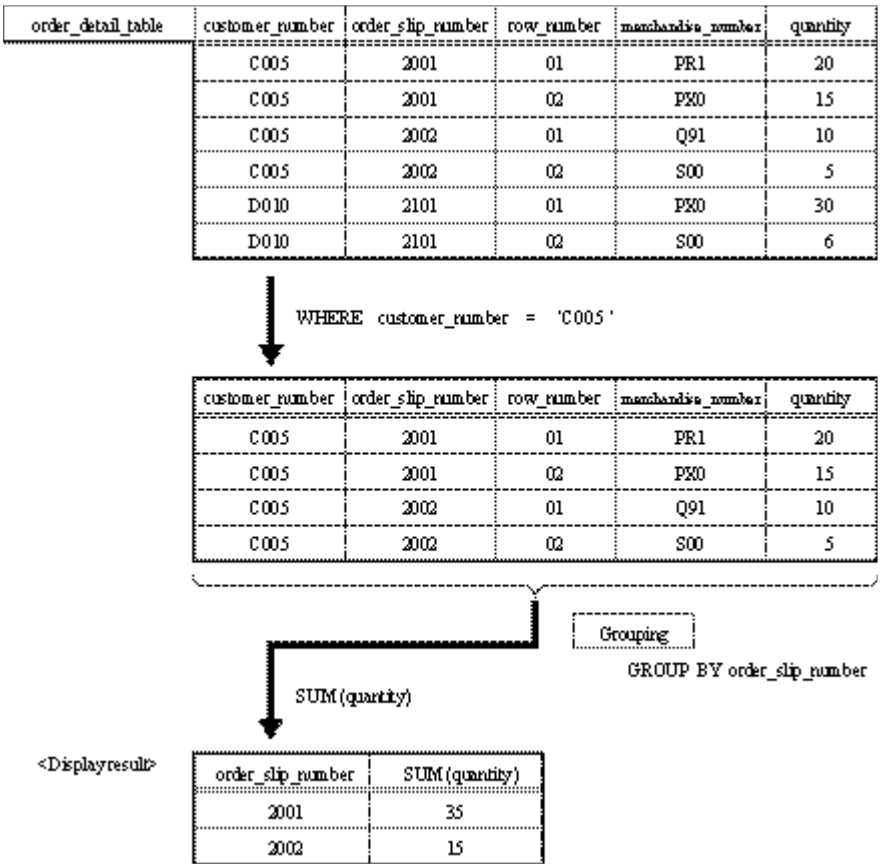
merchandise_number	sum (quantity)
PR1	20
PX0	45
Q91	10
S00	11

When the GROUP BY clause and the WHERE clause are written at the same time, the WHERE clause is

executed first, and then the GROUP BY clause is executed based on the execution result of WHERE clause.
For example, the statement "calculate the sum of order_quantities of customer_number C005 by order_slip_number from order_detail_table, and display" is expressed as follows:

```
SELECT   order_slip_number,   SUM (quantity)
  FROM   order_detail_table
  WHERE   customer_number   =   'C005'
  GROUP   BY   order_slip_number
```

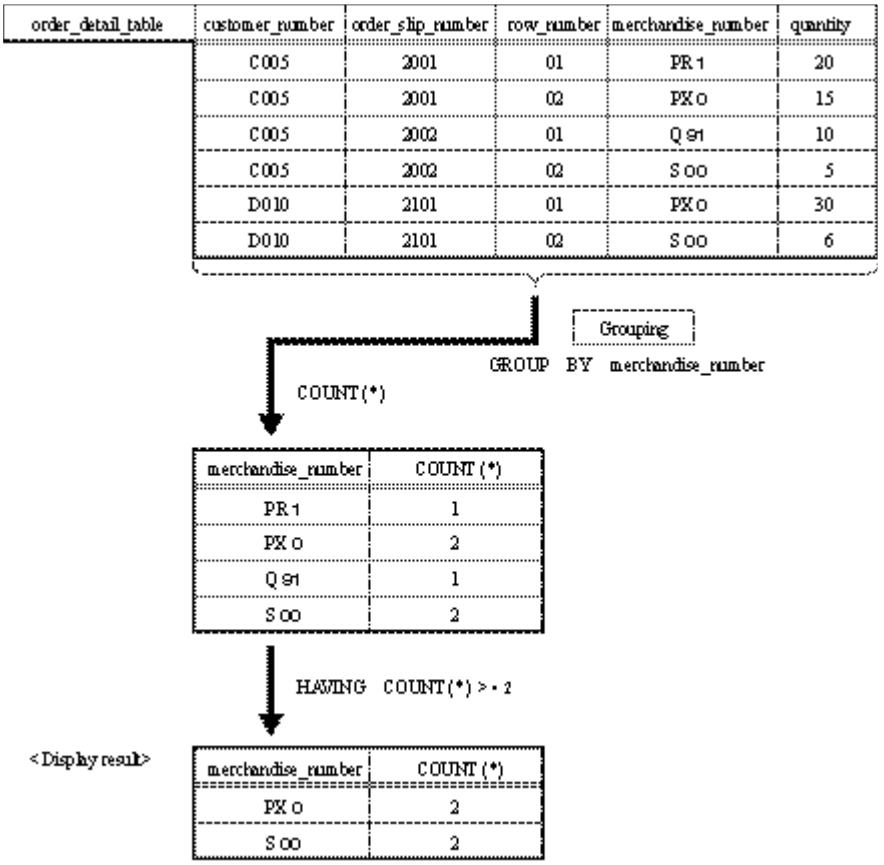
Figure 2-4-2



To use the result extracted by the GROUP BY clause and the aggregate function as a condition, the HAVING clause is used.
For example, the statement "retrieve the merchandise numbers recorded twice or more, and display them with their number of records" is expressed as follows:

```
SELECT  merchandise_number,  COUNT (*)  FROM  order_detail_table
  GROUP  BY  merchandise_number
  HAVING  COUNT (*)  > =  2
```

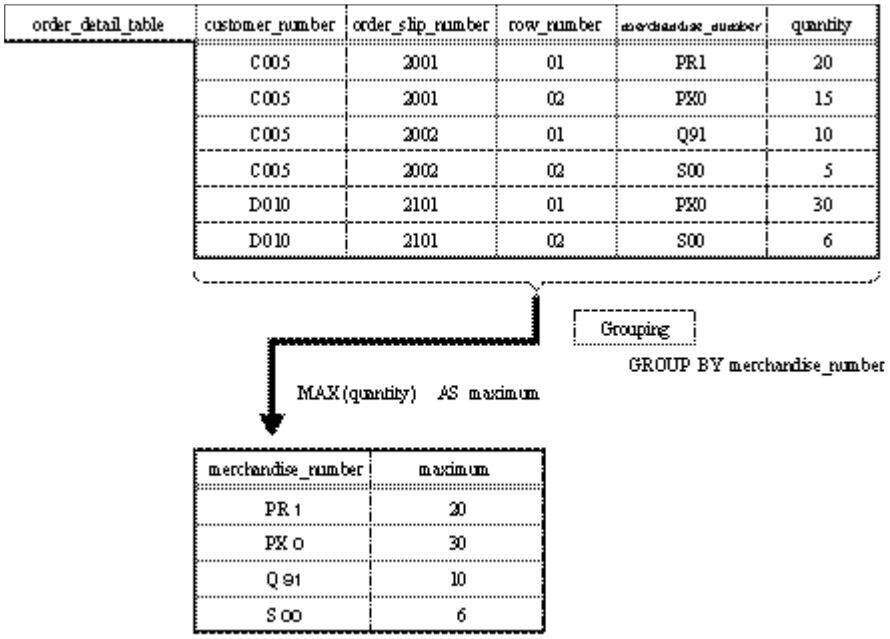
Figure 2-4-3



To give a new column_name to the column extracted by the aggregate function, the AS clause is used.
For example, the statement "retrieve the maximum order quantity by merchandise_number from the order_detail_table, and display the extracted order_quantities with the column_name <maximum>" is expressed as follows:

```
SELECT  merchandise_number,  MAX (quantity)  AS  maximum  FROM  order_detail_table
  GROUP  BY  merchandise_number
```

Figure 2-4-4



Exercise 8. Write SQL statements for ① to ③ below, and display the results.

① "Calculate the average order quantity by customer_number from the order_detail_table, and display the quantities with the customer_numbers, with the column_name <average>."

② "Calculate the number of records whose merchandise_number begins with 'P' by merchandise from the order_detail_table, and display the number_of_records with the merchandise_numbers, with the column_name <number_of_records>."

③ "Calculate the sum of quantities by order_slip_number from the order_detail_table, and display the order_slip_numbers whose total_quantity is 20 or larger with their total_quantity, with the column_name <total_quantity>."

(Answer 8)

① SELECT customer_number, AVG (quantity) AS average FROM order_detail_table
GROUP BY customer_number

<Display result>

customer_number	average
C005	13
D010	18

← 13 is displayed by rounding 12.5

② SELECT merchandise_number, COUN'T (*) AS number_of_records FROM order_detail_table
WHERE merchandise_number LIKE 'P%'
GROUP BY merchandise_number

<Display result>

merchandise_number	number_of_records
PR1	2
PX0	1

③ SELECT order_slip_number, SUM (quantity) AS total_quantity FROM order_detail_table
GROUP BY order_slip_number
HAVING SUM (quantity) > = 20

<Display result>

order_slip_number	total_quantity
2001	35
2101	36

---

② Sorting of data

Rows extracted from a table are not always sorted in the specified order. Therefore, rows are displayed after being rearranged in the order of values in a certain column to improve readability.
In SQL, the sorting is specified by the ORDER BY clause.

- When sorted in the ascending order : ASC (ascending)
- When sorted in the descending order : DESC (descending)

When there is no specification, ASC is used as the default. The numeric type data and the character type data are sorted in ascending/descending order by the size of the numeric values and character code values, respectively.
For example, the statement "display the order_slip_numbers and order_receiving_date from the order_table in the ascending order" is expressed as follows:

SELECT order_slip_number, order-receiving_date FROM order_table
ORDER BY order_receiving_date ASC …… ASC can be omitted.

<Display result>

order_slip_number	order_receiving_date
2101	07/28/1999
2001	08/07/1999
2002	09/01/1999
2201	09/10/1999

By specifying multiple columns, data can be sorted into major classifications, intermediate classifications, and minor classifications.
For example, the statement "display all data from the order_detail_table in the ascending order of the row_numbers and in the descending order of quantity" is written as follows:

SELECT * FROM order_detail_table
ORDER BY row_number ASC, quantity DESC

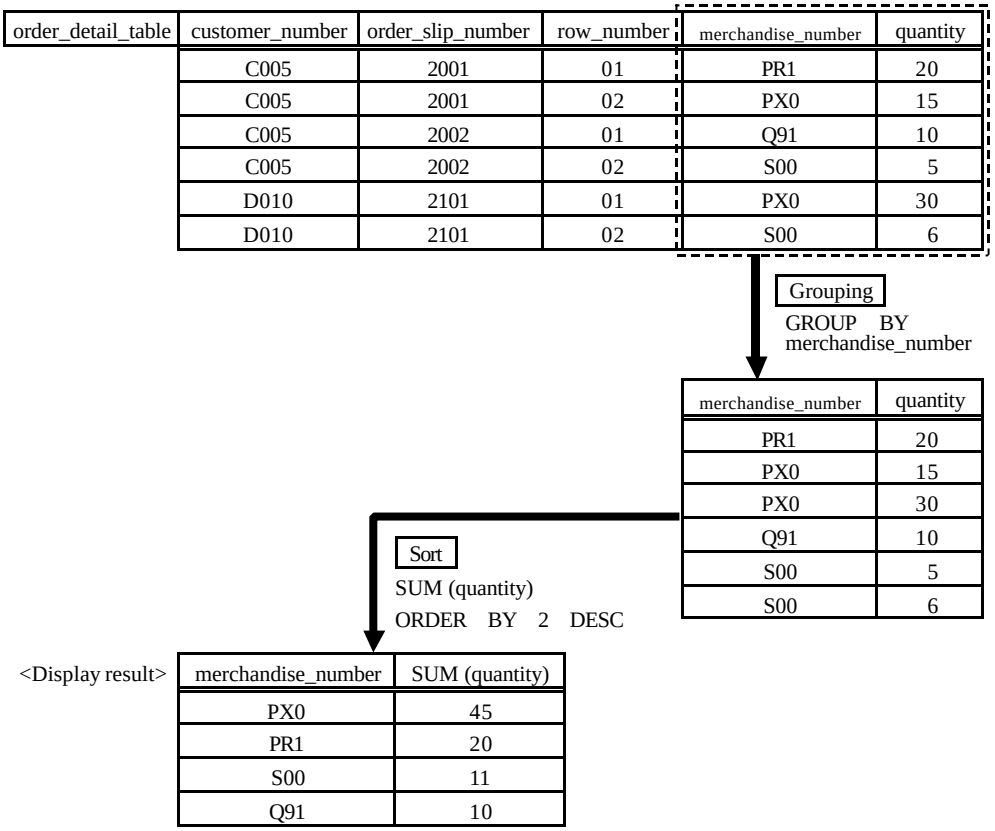
<Display result>

customer_number	order_slip_number	row_number	merchandise_number	quantity
D010	2101	01	PX0	30
C005	2001	01	PR1	20
C005	2002	01	Q91	10
C005	2001	02	PX0	15
D010	2101	02	S00	6
C005	2002	02	S00	5

The result gained by the aggregate function can be used as a sort key.
For example, the statement "calculate the sum of order quantities by the merchandise_number from the order_detail_table, and display the merchandise_numbers in the descending order of the total order quantities" is expressed as follows:

SELECT merchandise_number, SUM (quantity) FROM order_detail_table
GROUP BY merchandise_number
ORDER BY 2 DESC

Figure 2-4-5
Sort



In this example, a "2" written after the ORDER BY clause shows the position of the corresponding column in the SELECT statement. In this case, as the data are sorted (in the descending order) based on "SUM (Quantity)" located in the second position in the SELECT statement, "2" is specified.
Depending on the DBMS type, "ORDER BY SUM (quantity) DESC" is acceptable. However, it is important to note that some types of DBMS accept only the column of the table or the position in the SELECT statement in the ORDER BY clause.

Exercise 9. Write SQL statements for ① to ③ below, and display the results.

① "Display merchandise_names and their unit_prices from the merchandise_table in the ascending order of merchandise_names."

② "Display merchandise_numbers and quantities from the order_detail_table in the ascending order of merchandise_numbers and in the descending order of the quantities."

③ "Calculate the sum of order_quantities by order_slip_number from the order_detail_table, and display order_slip_numbers in the descending order of the total_order_quantities."

(Answer 9)

① SELECT merchandise_name, unit_price FROM merchandise_table
ORDER BY merchandise_name ASC

<Display result>

merchandise_name	unit_price
System_0-type	4500
Disk_1-type	910
Printer_1-type	300
Printer_X-type	550

② SELECT merchandise_number, quantity FROM order_detail_table
ORDER BY merchandise_number ASC, quantity DESC

<Display result>

merchandise_number	quantity
PR1	20
PX0	30
PX0	15
Q91	10
S00	6
S00	5

③ SELECT order_slip_number, SUM (quantity) AS total_quantity FROM order_detail_table
GROUP BY order_slip_number
ORDER BY 2 DESC

<Display result>

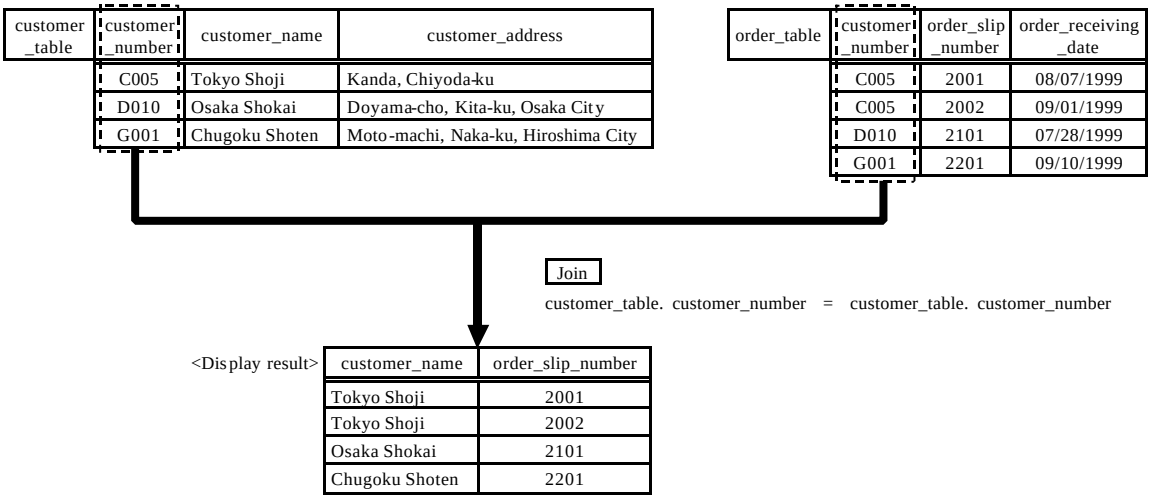
order_slip_number	total_quantity
2101	36
2001	35
2002	15

2.4.2 Join Processing

Join processing combines values in the specified rows in multiple tables. To perform this process, columns of the same data attribute must exist. Multiple tables are usually combined using the primary key and the external key.
For example, the statement "Combine the customer_table and the order_table, and retrieve customer_names and order_slip_numbers" is written as follows. In this case, to combine the customer_table and the order_table, customer_numbers are used as the (relational) key.

SELECT customer_name, order_slip_number FROM customer_table, order_table
WHERE customer_table. customer_number = order_table. customer_number

Figure 2-4-6 Join processing



Thus, in the SELECT statement to combine, two table_names are specified in the FROM clause, and columns to combine are connected by the equal sign in the WHERE clause. In most cases, the two column_names are the same. Therefore, the table_name and the column_name are connected by a period to distinguish between the two column_names.

The above SQL statement can also be written as follows:

```
SELECT customer_name, order_slip_number FROM customer_table X, order_table Y
  WHERE X.customer_number = Y.customer_number
```

In this SQL statement, the columns of the same name are distinguished by naming the customer_table X and the order_table Y, and specifying like "X.customer_number = Y.customer_number." X and Y, in this case, are called the "correlation name."

Exercise 10. Write SQL statements for ① to ④ below, and display the results.

① "Combine the customer_table and the order_detail_table, and display customer_names, merchandise_numbers, and quantities."

② "Combine the customer_table and the order_table, and display the names of the customers who placed orders in September 1999."

③ "Combine the order_detail_table and the merchandise_table, and calculate the sum of quantities by merchandise, and display the total_quantities with the merchandise_names, naming the column <total_quantity>."

④ "Combine the customer_table, the order_detail_table, and the merchandise_table, and calculate the sum of the amount by customer, and display the total amount with the customer_names, naming the column <total_amount>."
- The amount by merchandise is calculated by "quantity × unit_price."
- "total_amount" is the total by customer.

(Answer 10)

①
```
SELECT customer_name, merchandise_number, quantity FROM customer_table, order_detail_table
  WHERE customer_table. customer_number = order_detail_table. customer_number
```

<Display result>

customer_name	merchandise_number	quantity
Tokyo Shoji	PR1	20
Tokyo Shoji	PX0	15
Tokyo Shoji	Q91	10
Tokyo Shoji	S00	5
Osaka Shokai	PX0	30
Osaka Shokai	S00	6

②
```
SELECT customer_name FROM customer_table X, order_table Y
  WHERE X.customer_number = Y. customer_number
    AND order_receiving_date LIKE '99/09/_ _'
```

<Display result>

customer_name
Tokyo Shoji
Chugoku Shoten

③
```
SELECT merchandise_name, SUM (quantity) AS total_quantity
    FROM order_detail_table X, merchandise_table Y
  WHERE X. merchandise_number = Y. merchandise_number
  GROUP BY merchandise_name
```

<Display result>

merchandise_name	total_quantity
Printer_1-type	20
Printer_X-type	45
Disk_1-type	10
System_0-type	11

④ SELECT customer_name, SUM (quantity*unit_price) AS total_amount
FROM customer_table X, order_detail_table Y, order_table Z
WHERE X. customer_number = Y. customer_number
AND Y. merchandise_number = Z. merchandise_number
GROUP BY customer_name

<Display result>

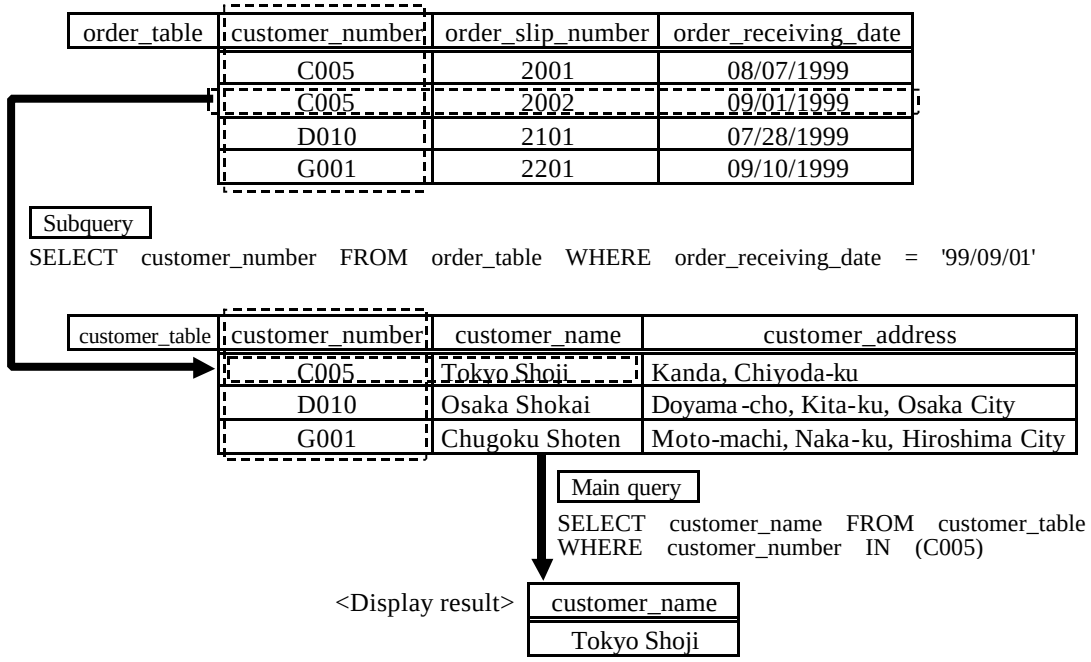
customer_name	total_amount
Tokyo Shoji	45850
Osaka Shokai	43500

2.4.3 Using Subqueries

A subquery is a query made for different tables or the same table, using a query result as a retrieval condition. In other words, subquery means making the next query (main query) based on the first query. To perform this process, specify the SELECT statement of the subquery by using the IN predicate in the SELECT statement.
For example, the statement "Extract the customer_names who placed orders in September 1st, 1999." is expressed as follows:

```
SELECT customer_name
    FROM customer_table WHERE customer_number
        IN (SELECT customer_number FROM order_table
                WHERE order_receiving_date = '99/09/01')
```

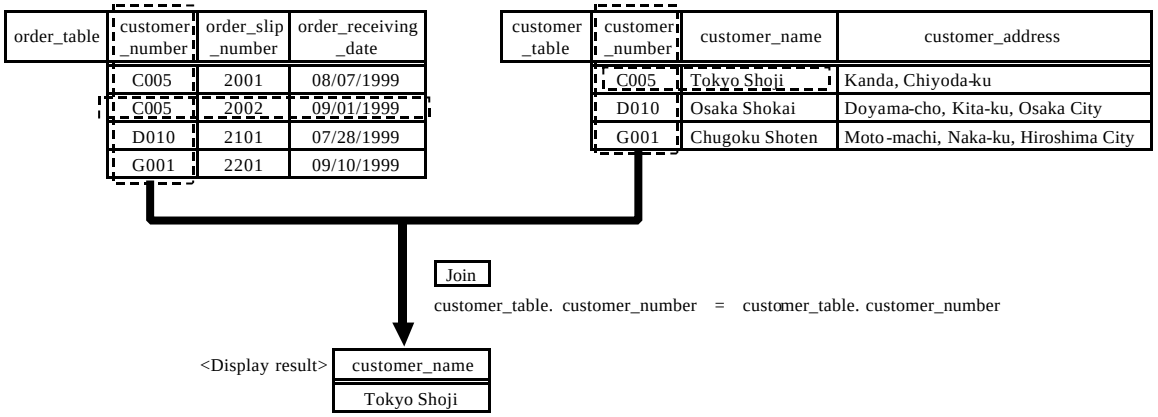
Figure 2-4-7 Subquery Processing Using the IN Predicate



The SQL statement using a subquery can be rewritten as the SQL statement of join processing as follows:

```
SELECT  customer_name  FROM  order_table,  customer_table
  WHERE  order_receiving_date  =  '99/09/01'
              AND  order_table. customer_number  =  order_table. customer_number
```

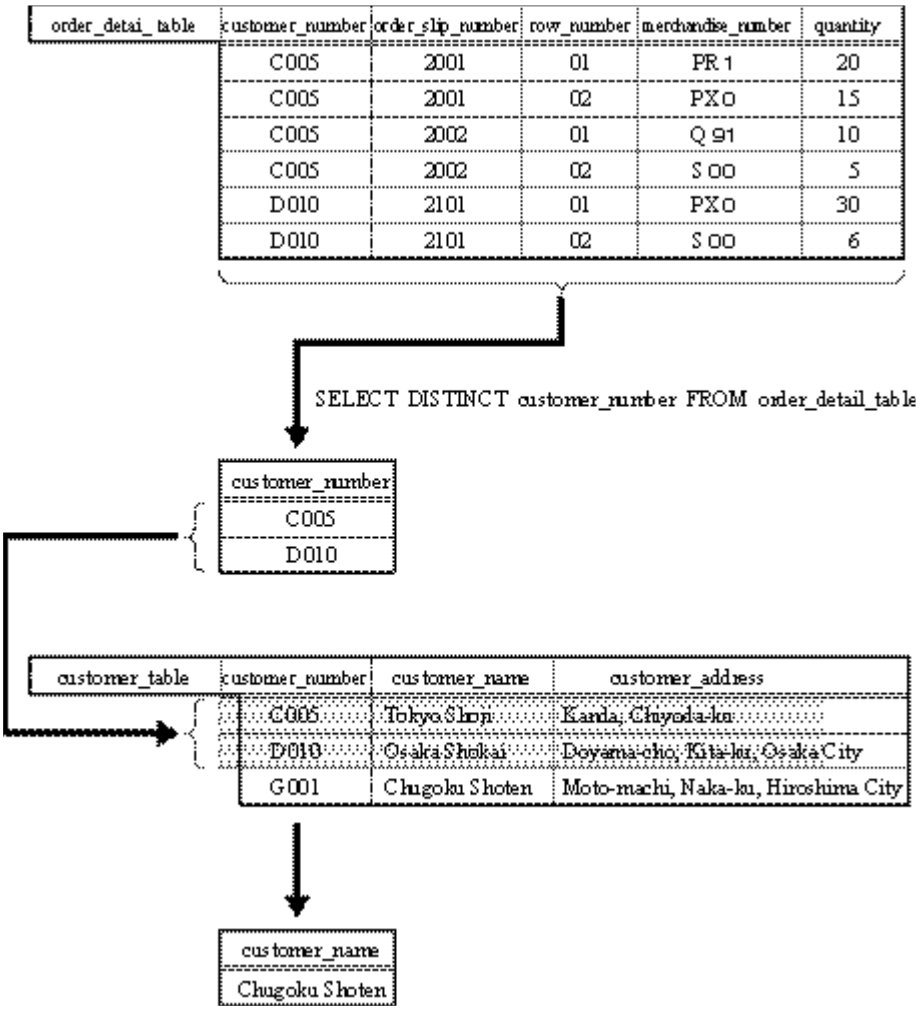
Figure 2-4-8 Subquery Processing Using Join Processing



Use the NOT IN predicate if you want to use a result other than the subquery result as a condition of the main query.
For example, the statement "display the customer_names not recorded in the order_detail_table" is expressed as follows:

```
SELECT customer_name FROM customer_table WHERE customer_number
  NOT IN (SELECT DISTINCT customer_number FROM order_detail_table)
```

Figure 2-4-9 Subquery Processing Using the NOT IN Predicate



Exercise 11. Write SQL statements for ① to ③ below, and display the results.

① "Display the names and addresses of customers who ordered the merchandise number 'PX 0'."

② "Display the merchandise numbers and quantities of merchandise ordered on other than September 1999."

③ "Display the names of customers who placed at least one order amounting to 10,000 or more per merchandise."
- The amount per merchandise is calculated by "quantity × unit_price."

(Answer 11)

① SELECT customer_name, customer_address FROM customer_table
WHERE customer_number
IN (SELECT customer_number FROM order_detail_table
WHERE merchandise number = 'PX0')

<Display result>

customer_name	customer_address
Tokyo Shoji	Kanda, Chiyoda-ku
Osaka Shokai	Doyama-cho, Kita-ku, Osaka City

② SELECT merchandise_number, quantity FROM order_detail_table
WHERE order_slip_number
NOT IN (SELECT order_slip_number FROM order_table
WHERE order_receiving_date = '99/09/_ _')

<Display result>

merchandise_number	quantity
PR1	20
PX0	15
PX0	30
S00	6

③ SELECT customer_name FROM customer_table
WHERE customer_number
IN (SELECT DISTINCT customer_number
FROM order_detail_table X, merchandise_table Y
WHERE X. merchandise_number = Y. merchandise_number
AND quantity * unit_price > = 10000)

<Display result>

customer_name
Tokyo Shoji
Osaka Shokai

2.4.4 Use of View

As already stated, a view is defined by the data definition language (SQL-DDL). A view can be defined by extracting part of an actual table and by combining multiple tables. In this section, creating a view by combining multiple tables, is explained.
For example, the statement "combine the customer_table and the order_table, and extract customer_names and order_slip_numbers" used in join process can also be defined as "create a view consisting of customer_names and order_slip_numbers."

```
CREATE VIEW customer_order_slip_table
  AS SELECT customer_name, order_slip_number FROM customer_table X, order_table
Y
     WHERE X. customer_number = Y. customer_number
```

<Display result>

customer_order_slip_table	customer_name	order_slip_number
	Tokyo Shoji	2001
	Tokyo Shoji	2002
	Osaka Shokai	2101
	Chugoku Shoten	2201

As a result, a "customer_order_slip_table," created by joining the customer_table and order_table is defined as a view.
This is called a "query" in the DBMS used on personal computers. In the data manipulation by the DBMS on personal computers, only data satisfying certain conditions can be extracted from the database (actual table) by defining a query (view). A query can be defined by specifying the query name, target table/query name, field (column) name, and query conditions.
As explained in 2.3.4, once a view is defined, data in the view become accessible. This improves the usability of the view.
For example, the statement "display the customer_name whose order_slip_number is 2101" is defined by the SQL statement of join processing as follows:

```
SELECT customer_name FROM customer_table, order_table
  WHERE customer_table. customer_number = order_table. customer_number
        AND order_slip_number = 2101
```

Using the previously defined view "customer_order_slip_table," the above example can be defined by the SQL statement as follows:

```
SELECT customer_name FROM customer_order_slip_table
  WHERE order_slip_number = 2101
```

When the above two SQL statements are compared, the one using the view is simpler. If the view has been defined, the data in the view "customer_order_slip" are automatically updated when order records increase and actual tables, "customer_table" and "order_table," are updated.
Thus, when extracting required data from multiple tables, the method to create a view including the required data beforehand and extract the data form the view is more efficient.

2.4.5 Change Processing

In this section, as data change processing insert, update, and deletion of data are explained.

(1) Data insertion

Data insertion is performed for an actual table (data cannot be inserted into a view), and it is manipulated by "INSERT statement" in SQL.

Data insertion

```
INSERT INTO the name of the table in which the data are inserted (column names to be inserted)
       VALUES values to be inserted
```

For example, the statement "add new customer information (A001, Yokohama Shokai, Nishi-shiba, Kanazawa-ku, Yokohama City) to the customer table" is written as follows:

```
INSERT INTO customer_table (customer_number, customer_name, customer_address)
    VALUES ('A001', N'Yokohama Shokai', N'Nishi-shiba, Kanazawa-ku, Yokohama City')
```

customer_table

customer_number	customer_name	customer_address
C005	Tokyo Shoji	Kanda, Chiyoda-ku
D010	Osaka Shokai	Doyama-cho, Kita-ku, Osaka City
G001	Chugoku Shoten	Moto-machi, Naka-ku, Hiroshima City
A001	Yokohama Shokai	Nishi-shiba, Kanazawa-ku, Yokohama City

← Insert

Data values after the VALUES clause correspond to the column_names after the table_name. When inserting data, if the column_names and their order correspond to those of the table in which the data are inserted, column_names following the table_name after INSERT INTO need not be specified.

(2) Data update

Data update means updating values in the specified rows in the actual table, and it is manipulated by "UPDATE statement" in SQL.

Data update

```
UPDATE table_name
    SET column_name = expression WHERE query_condition
```

For example, the statement "raise the price of printers in the merchandise_table by 10%" is expressed as follows:

```
UPDATE merchandise_table
    SET unit_price = unit_price * 1.1
 WHERE merchandise_name LIKE N' printer %'
```

merchandise_table

merchandise_number	merchandise_name	unit_price
PR1	Printer_1-type	300
PX0	Printer_X-type	550
Q91	Disk_1-type	910
S00	System_0-type	4500

Update

330
605

In the above definition, the specified rows are selected by the WHERE clause and the specified columns are updated by the SET clause.

(3) Data deletion

Data deletion means deleting the specified rows in the actual table, and it is controlled by "DELETE statement" in SQL.

Data deletion

```
DELETE FROM table_name WHERE query_condition
```

For example, the statement "delete the data of Chugoku Shoten from the customer_table" is expressed as follows:

```
DELETE FROM customer_table
    WHERE customer_name = 'Chugoku Shoten'
```

customer_table	customer_number	customer_name	customer_address	
	C005	Tokyo Shoji	Kanda, Chiyoda-ku	
	D010	Osaka Shokai	Doyama-cho, Kita-ku, Osaka City	
	G001	Chugoku Shoten	Moto-machi, Naka-ku, Hiroshima City	→ Delete

In the above definition, the specific rows selected by the WHERE clause are deleted. If the WHERE clause is omitted, the whole rows of the table is deleted.

2.4.6 Summary of SQL

In this section, the contents in the preceding sections are confirmed by creating SQL statements for Q1 to Q20 to execute a series of processes from the definition to the manipulation of tables.

Q1. Define the table ① to ③ below by SQL. These tables and data are also used in Q2 and later.

① <student table> primary key: student number

student number	name	gender	address
1201	Shizuka Yamamoto	Female	Yokohama City
1221	Yuka Motoyama	Female	Kawasaki City
1231	Jiro Yamada	Male	Kawasaki City
1232	Shiro Yamamoto	Male	Yokohama City
1233	Karin Kida	Female	Yokosuka City
1235	Shinji Kimoto	Male	Yokohama City
4-character text	10-character kanji text	1-character kanji	5-character kanji text

② <score table> primary key: student_number + subject_code, foreign key: subject_code

student_number	subject_code	score	examination_date
1201	A01	60	10/10/1999
1201	B01	85	10/11/1999
1221	A01	70	10/10/1999
1221	B02	60	10/11/1999
1231	A02	90	10/10/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999
4-character text	3- character text	3- character numeric value	Date type

③ <subject_table> primary key: subject_code

subject_code	subject_name
A01	Mathematics I
A02	Mathematics II
B01	English I
B02	English II
3- character text	5- character kanji text

Q2. As the data of "student number" and "name" are frequently used, it is necessary to create a name table as shown below by extracting these two items from the student table. Write the SQL statement to set the new table.

<name table>

student_number	name
1201	Shizuka Yamamoto
1221	Yuka Motoyama
1231	Jiro Yamada
1232	Shiro Yamamoto
1233	Karin Kida
1235	Shinji Kimoto

Q3. The authority concerning the student table is defined as ① to ③ below. Write SQL statements for ① to ③. ( ) shows the authorization identifier (department or person given the authority).

① (The administrative department) has full authority.
② (The instruction department) has the authority to refer to and update the student table.
③ (Teachers) have the authority to refer to the student table.

Q4. Write the SQL statement to extract (project) names and addresses from the student table and display the results.

<Display result>

name	address
Shizuka Yamamoto	Yokohama City
Yuka Motoyama	Kawasaki City
Jiro Yamada	Kawasaki City
Shiro Yamamoto	Yokohama City
Karin Kida	Yokosuka City
Shinji Kimoto	Yokohama City

Q5. Write the SQL statement to extract (select) the students whose (gender is 'female') from the student table and display the results.

<Display result>

student_number	name	gender	address
1201	Shizuka Yamamoto	Female	Yokohama City
1221	Yuka Motoyama	Female	Kawasaki City
1233	Karin Kida	Female	Yokosuka City

Q6. Write the SQL statement to extract the records whose "student_number is not '1221'" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1201	A01	60	10/10/1999
1201	B01	85	10/11/1999
1231	A02	90	10/10/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999

Q7. Write the SQL statement to extract the records whose "examination date is '10/10/1999'" and "score is 80 or higher" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1231	A02	90	10/10/1999

Q8. Write the SQL statement to extract the records whose "examination date is '10/10/1999'" or "score is 80 or higher" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1201	A01	60	10/10/1999
1201	B01	85	10/11/1999
1221	A01	70	10/10/1999
1231	A02	90	10/10/1999
1231	B01	80	10/11/1999

Q9. Write the SQL statement to extract the records whose "score is 70 to 80" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1221	A01	70	10/10/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999

Q10. Write the SQL statement to extract the records whose "subject code begins with 'A'" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1201	A01	60	10/10/1999
1221	A01	70	10/10/1999
1231	A02	90	10/10/1999

Q11. Write the SQL statement to extract the records whose "student number's third position of characters is '2'" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1221	A01	70	10/10/1999
1221	B02	60	10/11/1999

Q12. Write the SQL statement to extract the records whose "score is 70 or higher," and "examination date is '10/11/1999"" or "subject code's last character is '1'" from the score table and display the results.

<Display result>

student_number	subject_code	score	examination_date
1201	B01	85	10/11/1999
1221	A01	70	10/10/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999

Q13. Write the SQL statement to calculate the total score of each student from the score table and display the results. Calculate the total score by grouping scores by student number.

<Display result>

student_number	SUM (score)
1201	145
1221	130
1231	245

Q14. Write the SQL statement to calculate the average score of each subject from the score table and display the results. Calculate the average score by grouping scores by subject code.

<Display result>

subject_code	average_score
A01	65
A02	90
B01	83
B02	68

Q15. Write the SQL statement to calculate the total number of examinees by examination date from the score table and display the results. Calculate the total number of examinees by grouping examinees by examination date.

[Duplication is counted]

<Display result>

examination_date	total_number_of_examinees
10/10/1999	3
10/11/1999	4

[Duplication is not counted (examinees of the same student number are counted as one examinee)]

<Display result>

examination_date	total_number_of_examinees
10/10/1999	3
10/11/1999	3

Q16. Write the SQL statement to sort scores in the score table in the descending order and display the results.

<Display result>

student_number	subject_code	score	examination_date
1231	A02	90	10/10/1999
1201	B01	85	10/11/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999
1221	A01	70	10/10/1999
1201	A01	60	10/10/1999
1221	B02	60	10/11/1999

Q17. Write the SQL statement to sort scores in the score table by subject code in descending order and display the results.

<Display result>

student_number	subject_code	score	examination_date
1221	A01	70	10/10/1999
1201	A01	60	10/10/1999
1231	A02	90	10/10/1999
1201	B01	85	10/11/1999
1231	B01	80	10/11/1999
1231	B02	75	10/11/1999
1221	B02	60	10/11/1999

Q18. Write the SQL statement to calculate the total score of each student from the score_table and sort them in descending order, and display the results.

<Display result>

student_number	SUM (score)
1231	245
1201	145
1221	130

Q19. Write the SQL statement to extract the student numbers, the subject names of the examinations, and the scores from the score table and the subject table, and display the results.

<Display result>

student_number	subject_name	score
1201	Mathematics I	60
1201	English I	85
1221	Mathematics I	70
1221	English II	60
1231	Mathematics II	90
1231	English I	80
1231	English II	75

Q20. Write the SQL statement to extract the name of the students whose score is 60 or lower from the student table and the score table, and display the results.

<Display result>

name
Shizuka Yamamoto
Yuka Motoyama

Answer 1.

```
① CREATE TABLE student_table
    (student_number   CHAR (4),
     name             NCHAR (10),
     gender           NCHAR (1),
     address          NCHAR (5),
     PRIMARY          KEY  student_number)
② CREATE TABLE score_table
    (student_number   CHAR (4),
     subject_code     CHAR (3),
     score            INT (3),
     examination_date DATE,
     PRIMARY          KEY  (student_number, subject_code),
     FOREIGN          KEY subject_code  REFERENCES  subject_table)
③ CREATE TABLE subject_table
    (subject_code     CHAR (3),
     subject_name     NCHAR (5),
     PRIMARY          KEY  subject_code)
```

Answer 2.

```
SELECT  VIEW  name_table
      AS  SELECT  student_number, name
            FROM  student_table
```

Answer 3.

```
① GRANT  ALL  PRIVILEGES  ON  student_table  TO  administration_department
② GRANT  SELECT  UPDATE   ON  student_table  TO  instruction_department
③ GRANT  SELECT           ON  student_table  TO  teacher
```

Answer 4.

```
SELECT  name,  address  FROM  student_table
```

Answer 5.

```
SELECT  *  FROM  student_table
  WHERE  gender  =  'female'
```

Answer 6.

```
SELECT  *  FROM  score_table
  WHERE  student_number  NOT  =  '1221'
```

Answer 7.

```
SELECT  *  FROM  score_table
  WHERE  examination_date  =  '10/10/1999"  AND  score  >=  80
```

Answer 8.

```
SELECT  *  FROM  score_table
  WHERE  examination_date  =  '10/10/1999"  OR  score  >=  80
```

Answer 9.

```
SELECT  *  FROM  score_table
  WHERE  score  BETWEEN  70  AND  80
```

Answer 10.

```
SELECT  *  FROM  score_table
  WHERE  subject_code  LIKE  'A%'
```

Answer 11.
```
SELECT * FROM score_table
  WHERE student_number LIKE '_ _2_ '
```

Answer 12.
```
SELECT * FROM score_table
  WHERE score > = 70
    AND (examination_date = '10/11/1999' OR subject_code LIKE '_ _1')
```

Answer 13.
```
SELECT student_number, SUM (score) FROM score_table
  GROUP BY student_number
```

Answer 14.
```
SELECT subject_code, AVG (score) AS average_score FROM score_table
  GROUP BY subject_code
```

Answer 15.
[Duplication is counted]
```
SELECT examination_date, COUNT (*) AS total_number_of_examinees FROM
   score_table
GROUP BY examination_date
```

[Duplication is not counted (examinees of the same student_number are counted as one examinee)]
```
SELECT examination_date, COUNT (DISTINCT student_number) AS total_
    number_of_examinees FROM score_table
  GROUP BY examination_date
```

Answer 16.
```
SELECT * FROM score_table
  GROUP BY score DESC
```

Answer 17.
```
SELECT * FROM score_table
ORDER BY subject_code, score DESC
```

Answer 18.
```
SELECT student_number, SUM (score) FROM score_table
  GROUP BY student_number
  ORDER BY 2 DESC
```

Answer 19.
```
SELECT student_number, subject_name, score FROM score_table, subject_table
  WHERE score_table.subject_code = subject_table. subject_code
or
SELECT student_number, subject_name, score FROM score_table X,
    subject_table Y
  WHERE X. subject_code = Y. subject_code
```

Answer 20.
```
SELECT name FROM student_table
  WHERE student_number IN
     (SELECT student_number FROM score_table
       WHERE score <= 60)
or
SELECT name FROM student_table X, score_table Y
  WHERE X .student_number = Y. student_number
    AND score < = 60
```

2.5 Extended Use of SQL

Generally, SQL is used as a supplementary language (data sub language) to use databases, rather than used independently.
As a data sub-language, SQL is used in the following three ways:

- Embedded SQL
  Use SQL by embedding it in application programs written in high-level languages.
- Module language
  Use a module language developed to abstract the interface combining a high-level language and SQL.
- API (Application Programming Interface)
  Use API, the interface of functions, commands, etc., prepared for programmers to develop applications.

In this section, the use of the embedded SQL is described in detail.

2.5.1 Embedded SQL

By embedding SQL statements in application programs, routine operational processing, large amounts of data processing, and the processing of relational databases while processing files become more efficient.
In the embedded SQL, the cursor is used for operation. However, the operation of reading a row from a relational database can be performed without using the cursor (non-cursor operation).

2.5.2 Cursor Operation

When reading multiple rows from a table (relational database), the cursor is used. After instructing the reading of tables with the SELECT statement, rows are received one by one according by another instruction. The cursor is used to read one row at a time.
The following explain the cursor operations classified into the "program definition part" and the "program processing part."

(1) Program definition part

① Input/output work area

Processing of a relational database is instructed by the embedded SQL statement, and the process result is returned to the work area (variable) of the program definition part. The variable as a work area is called the "host variable."
The host variable as an input/output work area can be defined in the following format:

Definition of the host variable

```
EXEC SQL BEGIN DECLARE SECTION
    [host variable]
EXEC SQL END DECLARE SECTION
```

*One host variable is defined by one line.
*In SQL provided by vendors, a host variable is defined in the format defined by the normal programming.

It is important to define the host variable as an input/output work area to have the same attribute as the definition of the data type of the column in the table. If the defined data type is different, the value in the column may be truncated.

② SQLCODE

Definition of SQLCODE (SQLCOD in FORTRAN) is mandatory as a host variable. SQLCODE sets the return code showing whether every SQL statement is normally executed or not.
The contents of SQLCODE are mainly classified into the following three types:

- SQLCODE = 0 … Normal status
- SQLCODE = 100 … End status (end of the table and no corresponding row)
- SQLCODE < 0 … Error status

SQLCODE must be defined as having the same attributes as INTEGER (4-byte integer type), the data type of the column. Examples of the description in each language are shown below:

Definition of SQLCODE

```
<COBOL>
   01  SQLCODE  PIC  S9 (9)  COMP.
<PL/I>
   DCL  SQLCODE  BIN  FIXED  (31) ;
<FORTRAN>
   INTEGER  *  4  SQLCOD
<C>
   long  sqlcode;
```

③ Cursor

The cursor is defined in the program definition part using the SELECT statement. In the definition, the GROUP BY clause, the ORDER BY clause, and column functions can be included. Therefore, instructions of grouping and classification are not required in the program.
Avoid using duplicate cursor names in a program.

Cursor definition

```
EXEC  SQL  DECLARE  [cursor name]  CURSOR  FOR
    SELECT clause
      FROM [table_name]
    WHERE  [table_name. column_name]  =  [table_name. column_name]
```

(2) Program processing part

Cursor processing in the program processing part is performed in the order of the OPEN statement, the FETCH statement, and the CLOSE statement as shown below:

1. After the execution of the OPEN statement, the SELECT statement defined by the cursor is executed, and the cursor points to the first row of the corresponding table.
2. The FETCH statement fetches the row specified by the cursor, and returns the row to the host variable of the INTO clause. After fetching one row, the cursor points to the next row. And FETCH statement is repeated until no row is left in the table. That is, the termination condition of the FETCH statement is SQLCODE=100.
3. The CLOSE statement is used when there is no more row to be read in the table, and the cursor is closed.

Definition of the cursor processing statement

<OPEN> … Open the cursor
EXEC SQL OPEN [cursor name] END-EXEC

<FETCH> … Fetch the cursor
EXEC SQL FETCH [cursor name] INTO [host variable]
END-EXEC

<CLOSE> … Close the cursor
EXEC SQL CLOSE [cursor name] END-EXEC

Basically, the concept of the cursor operation is the same as that of the file operation.
First, open the file (or the cursor) and continue the processing of records one by one until the processing of all the records has finished, and then close the file (or the cursor). To read one record, the READ statement is used in the case of the file, while the FETCH statement is used in the case of the cursor.
For example, "print customer_numbers and customer_names in the customer_number order from the customer_table" is described by the embedded type SQL using COBOL as the host language as follows:

Program definition part

```
DATA    DIVISION.
WORKING-STORAGE    SECTION.
  EXEC    SQL    BEGIN    DECLARE  SECTION    END-EXEC.
    01    CUSTNO    PIC    X  (4).
    01    CUSTNAME    PIC    N  (10).
    01    SQLCODE    PIC    S  9  (9)    COMP.
  EXEC    SQL    END    DECLARE  SECTION    EDN-EXEC.
  EXEC    SQL    DECLARE    CUSTOMER    CURSOR
                 FOR    SELECT    customer_number,    customer_name
                        FROM    customer_table
                   ORDER    BY    customer_number    END-EXEC.
```

Program processing part

```
PROCEDURE    DIVISION.
  EXEC    SQL    OPEN    CUST    END-EXEC.
  EXEC    SQL    FETCH    CUST
                 INTO    :CUSTNO,    :CUSTNAME    END-EXEC.
  PERFORM    UNTIL    SQLCODE    =    100
    IF    SQLCODE    <    0
          THEN    PERFORM    (Error processing)
          ELSE    PERFORM    (One-line print processing)
  END-PERFORM.

[Error processing]
        |
[One-line print processing]
        |
```

(3) Data changes

The FETCH statement is used to read data from the table. The methods to update and delete the read data are explained below.

① Update by cursor processing

When updating rows read by the FETCH statement under certain conditions in the program, an update instruction is given using the UPDATE statement after the FETCH statement.
In the UPDATE statement format, it is important to use

WHERE CURRENT OF [cursor_name]

instead of the WHERE clause.

Definition of cursor update processing

```
EXEC SQL UPDATE [table_name]
                SET [update_expression]
            WHERE CURRENT OF [cursor_name] END-EXEC.
```

For example, "update the customer number of Tokyo Shoji to C100" is described by the embedded SQL as follows:

[Program definition part]

```
EXEC SQL END DECLARE SECTION END-EXEC.
EXEC SQL DECLARE TOKYO CURSOR
        FOR SELECT customer_number, customer_name FROM customer_table
            WHERE customer_name = 'Tokyo Shoji' END-EXEC.
```

[Program processing part]

```
EXEC SQL OPEN TOKYO END-EXEC.
EXEC SQL FETCH TOKYO
        INTO :CUSTNO, :CUSTNAME END-EXEC.

PERFORM UNTIL SQLCODE = 100
    IF SQLCODE < 0
        THEN PERFORM (error processing)
        ELSE
            EXEC SQL UPDATE customer_table
                        SET customer_number = 'C100'
                    WHERE CURRENT OF TOKYO END-EXEC.
```

② Deletion by cursor processing

Deletion by cursor processing can also be performed by instructing the DELETE statement after the FETCH statement in the same way as the update. In the DELETE statement format, as in the UPDATE statement format,

```
WHERE CURRENT OF [cursor_name]
```

is used instead of the WHERE clause.

Definition of cursor deletion processing

```
EXEC SQL DELETE FROM [table_name]
WHERE CURRENT OF [cursor_name] END-EXEC.
```

For example, "delete the data of Tokyo Shoji" is described by the embedded SQL as follows:

[Program definition part]

```
EXEC SQL DECLARE TOKYO CURSOR
        FOR SELECT customer_number, customer_name FROM customer_table
            WHERE cursor_name = 'Tokyo Shoji' END-EXEC.
```

[Program processing part]

```
EXEC SQL OPEN TOKYO END-EXEC.
EXEC SQL FETCH TOKYO
        INTO :CUSTNO, :CUSTNAME END-EXEC.

PERFORM UNTIL SQLCODE = 100
    IF SQLCODE < 0
        THEN PERFORM (error processing)
        ELSE
            EXEC SQL DELETE FROM customer_table
                WHERE CURRENT OF TOKYO END-EXEC.
```

2.5.3 Non-Cursor Operation

The non-cursor operation is a method to embed SQL statements without making a cursor declaration. This operation, however, is available only when one data item is read from the table.
Although the method of specification of SQL statements is almost the same as conversational SQL, descriptions in the program definition part and program processing part are slightly different because no cursor declaration is made.
For example, "update the customer_number of Tokyo Shoji to C100" used in data update by cursor processing can be processed by the non-cursor operation as follows, because only one data item is read from the table (customer_table).

[Program definition part]

```
        |

    EXEC  SQL  END  DECLARE  SECTION  END-EXEC.
```

[Program processing part]

```
    EXEC  SQL  UPDATE  customer_table
                   SET  customer_number  =  'C100'
                 WHERE  customer_name  =  'Tokyo Shoji'  END-EXEC.
```

Exercises

Q1 Choose <u>two correct</u> answers from the following descriptions concerning characteristics of the CODASYL-type database.

a) The data structure is represented by a hierarchy.
b) The data structure is represented by a table format consisting of rows and columns.
c) The data structure is represented as a network.
d) NDL is used as its standard database language.
e) SQL is used as its standard database language.

Q2 Which of the following SQL statements defines a schema?

a) CREATE b) DELETE c) INSERT d) SELECT

Q3 Which of the following is <u>not the SQL statement</u>?

a) CREATE
b) DELETE
c) DIVIDE
d) INSERT
e) UPDATE

Q4 Which of the following SQL statements can extract employee_name s whose salary is ¥300,000 or higher from the table "human_resource?"

a)
```
SELECT salary FROM human_resource
    WHERE employee_name > = 300000
    GROUP BY salary
```
b)
```
SELECT employee_name COUNT (*) FROM human_resource
    WHERE salary > = 300000
    GROUP BY employee_name
```
c)
```
SELECT employee_name FROM human_resource
    WHERE salary > = 300000
```
d)
```
SELECT employee_name, salary FROM human_resource
    GROUP BY salary
    HAVING COUNT (*) > = 300000
```
e)
```
SELECT employee_name, salary FROM human_resource
    WHERE employee_name > = 300000
```

Q5 **In SQL, the SELECT statement is used to extract records from a two-dimensional table. If the following statement is executed for the leased apartments below, which data group is extracted?**

```
SELECT property FROM leased_apartment_table
  WHERE (district = 'Minami-cho' OR time_from_the_station
    < 15)
          AND floor space > 60
```

Leased Apartment Table

property	district	area	time apartment from the station
A	Kita-cho	66	10
B	Minami-cho	54	5
C	Minami-cho	98	15
D	Naka-cho	71	15
E	Kita-cho	63	20

a) A b) A, C c) A, C, D, E
d) B, D, E e) C

Q6 **Which of the following two descriptions on the operation of the customer_table is <u>wrong</u>?**

Customer_table

CUSTOMER_NO	CUSTOMER_NAME	ADDRESS
A0005	Tokyo Shoji	Toranomon, Minato-ku, Tokyo
D0010	Osaka Shokai	Kyo-cho, Tenmanbashi, Chuo-ku, Osaka-City
K0300	Chugoku Shokai	Teppo-cho, Naka-ku, Hiroshima-City
G0041	Kyushu Shoji	Hakataekimae, Hakata-ku, Fukuoka-City

Operation 1
```
SELECT CUSTOMER_NAME, ADDRESS FROM CUSTOMER
```

Operation 2
```
SELECT * FROM CUSTOMER
    WHERE CUSTOMER_NO = 'D0010'
```

a) The table extracted by operation 1 has four rows.
b) The table extracted by operation 1 has two columns.
c) Operation 1 is PROJECTION and operation 2 is SELECTION.
d) The table extracted by operation 2 has one row.
e) The table extracted by operation 2 has two columns.

Q7 **Which of the following SQL statements for the table "Shipment Record" produces the largest value as a result of its execution?**

shipment_record

merchandise_number	quantity	date
NP200	3	19991010
FP233	2	19991010
TP300	1	19991011
IP266	2	19991011

a) SELECT AVG (quantity) FROM shipment_record

b) SELECT COUNT (*) FROM shipment_record

c) SELECT MAX (quantity) FROM shipment_record

d) SELECT SUM (quantity) FROM shipment_record
WHERE date = '19991011'

Q8 In SQL, DISTINCT in the SELECT statement is used to "eliminate redundant duplicate rows" from the table gained by the SELECT statement. How many rows are included in the table gained as a result of execution of the following SELECT statement with DISTINCT?

[SELECT statement]

```
SELECT DISTINCT customer_name, merchandise_name, unit_price FROM
 order_table, merchandise_table
        WHERE order_table. Merchandise_number = merchandise_table.
        Merchandise_number
```

[order_table]

customer_name	merchandise_number
Oyama Shoten	TV28
Oyama Shoten	TV28W
Oyama Shoten	TV32
Ogawa Shokai	TV32
Ogawa Shokai	TV32W

[merchandise_table]

merchandise_number	merchandise_name	unit_price
TV28	28-inch television	250,000
TV28W	28-inch television	250,000
TV32	32-inch television	300,000
TV32W	32-inch television	300,000

a) 2 b) 3 c) 4 d) 5

Q9 Which of the following SQL statements can extract the average salary by department from tables A and B?

table_A

name	belonging_code	salary
Sachiko Ito	101	200,000
Eiichi Saito	201	300,000
Yuichi Suzuki	101	250,000
Kazuhiro Honda	102	350,000
Goro Yamada	102	300,000
Mari Wakayama	201	250,000

table_B

department_code	department_name
101	Sales department I
102	Sales department II
201	Administration department

a) SELECT department_code, department_name, AVG (salary) FROM table_A, table_B ORDER BY department_code

b) SELECT department_code, department_name, AVG (salary) FROM table_A, table_B WHERE table_A. belonging code = table_B. department_code

c) SELECT department_code, department_name, AVG (salary) FROM table_A, table_B WHERE table_A. belonging code = table_B. department_code GROUP BY department_code, department_name

d) SELECT department_code, department_name, AVG (salary) FROM table_A, table_B WHERE table_A. belonging_code = table_B. department_code ORDER BY department_code

Q10 In a relational database system, which of the following SQL statements is used to extract rows specified by the cursor after it has been defined?

a) DECLARE statement b) FETCH statement c) OPEN statement

d) READ statement e) SELECT statement