- Basic SELECT Queries
- SELECT Statement Options
- SQL Data Manipulation Language (DML)
- Basic SELECT Syntax
- SELECT Clause
- Use Wildcard in Expression
- Column Definition of PRODUCT and VENDOR
- Table: CH07_SALECO PRODUCT
- Select an Entire PRODUCT Table
- Select with a Column List
- Using Column Aliases
- Using Computed Columns
- Numeric Calculation
- Date Arithmetic
- Date Arithmetic
- Listing Unique Values
- FROM Clause Options
- FROM Clause Options
- ORDER BY Clause Options
- WHERE Clause Options
- Using Comparison Operator on Numeric Attribute
- Using Comparison Operator on Character Attribute
- Using Comparison Operator on Date Attribute
- Logical Operators: AND, OR and NOT
- Special Operators in WHERE Clause
- Illustrations of Special Operators
- MySQL Comparison Operators
- MySQL Booleans or Conditions
- JOIN Operations
- JOIN Illustration
- Three Ways to Do Inner Join (Join)
- Example of JOIN USING
- Example of JOIN ON
- Example of Old-Style JOIN
- Illustrate Why Old-Style Join is Not Preferred
- Outer Joins
- Left Outer Join
- Right Outer Join

- Full Outer Join (Not Support in MySQL)
- Cross Join
- JOINs in MySQL
- Joining Tables with an Alias
- Recursive Joins
- Aggregate Processing
- Count
- MIN and MAX
- SUM and AVG
- Grouping Data (1)
- Grouping Data (2)
- HAVING Clause
- Subqueries
- WHERE Subqueries
- IN Subqueries
- HAVING Subqueries
- Multirow Subquery Operators: ALL and any
- FROM Subqueries
- Attribute List Subqueries (1)
- Attribute List Subqueries (2)
- Correlated Subqueries (Definition)
- Correlated Subqueries (Example)
- Correlated Subqueries (SQL)
- Correlated Subqueries (Exists)
- Correlated Subqueries (Example of Exists)
- Built-in SQL Functions
- MySQL String Functions
- MySQL Date/Time Functions
- MySQL Numeric Functions
- MySQL Conversion Functions
- Relational Set Operators (UNION)
- Relational Set Operators (UNION ALL)
- Relational Set Operators (INTERSECT)
- Relational Set Operators (MINUS / EXCEPT)
- Crafting SELECT Queries
- Review Questions
- Homework #C

marp: true theme: default class: invert size: 16:9 paginate: true footer: 國立陽明交通大學電子與光子學士學位學程 headingDivider: 1 style: | section::after { content: attr(data-marpit-pagination) '/' attr(data-marpit-pagination-total); } .columns { display: grid; grid-template-columns: repeat(2, minmax(0, 1fr)); gap: 1rem; } .middle-grid { display: grid; grid-template-columns: repeat(2, minmax(0, 1fr)); gap: 1rem; } .middle-grid img { width: 75%; } .grid { display: grid; grid-template-columns: 1fr 1fr; gap: 10px; } .grid img { width: 100%; } .red-text { color: red; }

.blue-text { color: lightskyblue;
}

.brown-text { color: brown;
}

.small-text { font-size: 0.70rem; }

Introduction to Structured Query Language (SQL)

- SQL is composed of commands that enable users
  - create database and table structures
  - perform various types of data manipulation
  - execute data administration
  - query the database to extract useful information.
- All RDBMS supports SQL, and many software vendors have developed extensions to the basic SQL command set.

Introduction to MySQL

- MySQL is a relational database management system (RDBMS)
- MySQL is open-source and free
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, scalable, and easy to use
- MySQL is cross-platform
- MySQL is compliant with the ANSI SQL standard
- MySQL was first released in 1995

- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter, My

SQL Statement in MySQL

- SQL keywords are NOT case sensitive: select is the same as SELECT
- Use semicolon(;) at the end of each SQL statement to separate each SQL statement
- Some of The Most Important SQL Commands

> INSERT INTO - [C]eate new data into a database SELECT - [R]ead data from a database UPDATE - [U]pdates data in a database DELETE - [D]eletes data from a database CREATE DATABASE - creates a new database; ALTER DATABASE - modifies a database CREATE TABLE - creates a new table; ALTER TABLE - modifies a table; DROP TABLE - deletes a table CREATE INDEX - creates an index (search key); DROP INDEX - deletes an index

SQL Basics

- Described in ANSI/ISO SQL
    - The American National Standards Institute (ANSI) prescribes a standard SQL.
    - International Organization for Standardization (ISO) also accept.
- SQL functions fit into several broad categories:
    - Data manipulation language (DML): INSERT, SELECT, UPDATE, DELETE
    - Data definition language (DDL): CREATE TABLE
    - Transaction control language (TCL): COMMIT, ROLLBACK
    - Data control language (DCL): GRANT, REVOKE

類別	功能	例子
DML	操作資料(CRUD)	`SELECT`,`INSERT`,`UPDATE`,`DELETE`
DDL	定義資料表結構、修改資料表	`CREATE`,`DROP`,`ALTER`
TCL	控制交易一致性	`COMMIT`,`ROLLBACK`,`SAVEPOINT`
DCL	控制權限	`GRANT`,`REVOKE`

- SQL is a nonprocedural language, including many set operators

SQL Data Manipulation Commands

TABLE 7.1

SQL DATA MANIPULATION COMMANDS

COMMAND, OPTION, OR OPERATOR	DESCRIPTION	COVERED
SELECT	**Selects attributes from rows in one or more tables or views**	**Chapter 7**
FROM	Specifies the tables from which data should be retrieved	Chapter 7
WHERE	Restricts the selection of rows based on a conditional expression	Chapter 7
GROUP BY	Groups the selected rows based on one or more attributes	Chapter 7
HAVING	Restricts the selection of grouped rows based on a condition	Chapter 7
ORDER BY	Orders the selected rows based on one or more attributes	Chapter 7
INSERT	**Inserts row(s) into a table**	**Chapter 8**
UPDATE	**Modifies an attribute's values in one or more table's rows**	**Chapter 8**
DELETE	**Deletes one or more rows from a table**	**Chapter 8**
Comparison operators		**Chapter 7**
=, <, >, <=, >=, <>, !=	Used in conditional expressions	Chapter 7
Logical operators		**Chapter 7**
AND/OR/NOT	Used in conditional expressions	Chapter 7
Special operators	**Used in conditional expressions**	**Chapter 7**
BETWEEN	Checks whether an attribute value is within a range	Chapter 7
IN	Checks whether an attribute value matches any value within a value list	Chapter 7
LIKE	Checks whether an attribute value matches a given string pattern	Chapter 7
IS NULL	Checks whether an attribute value is null	Chapter 7
EXISTS	Checks whether a subquery returns any rows	Chapter 7
DISTINCT	Limits values to unique values	Chapter 7
Aggregate functions	**Used with SELECT to return mathematical summaries on columns**	**Chapter 7**
COUNT	Returns the number of rows with non-null values for a given column	Chapter 7
MIN	Returns the minimum attribute value found in a given column	Chapter 7
MAX	Returns the maximum attribute value found in a given column	Chapter 7
SUM	Returns the sum of all values for a given column	Chapter 7
AVG	Returns the average of all values for a given column	Chapter 7

SQL Data Definition Commands

TABLE 7.2

SQL DATA DEFINITION COMMANDS

COMMAND OR OPTION	DESCRIPTION	COVERED
CREATE SCHEMA AUTHORIZATION	**Creates a database schema**	**Chapter 8**
CREATE TABLE	**Creates a new table in the user's database schema**	**Chapter 8**
NOT NULL	Ensures that a column will not have null values	Chapter 8
UNIQUE	Ensures that a column will not have duplicate values	Chapter 8
PRIMARY KEY	Defines a primary key for a table	Chapter 8
FOREIGN KEY	Defines a foreign key for a table	Chapter 8
DEFAULT	Defines a default value for a column (when no value is given)	Chapter 8
CHECK	Validates data in an attribute	Chapter 8
CREATE INDEX	**Creates an index for a table**	**Chapter 8**
CREATE VIEW	**Creates a dynamic subset of rows and columns from one or more tables**	**Chapter 8**
ALTER TABLE	**Modifies a table's definition (adds, modifies, or deletes attributes or constraints)**	**Chapter 8**
CREATE TABLE AS	**Creates a new table based on a query in the user's database schema**	**Chapter 8**
DROP TABLE	**Permanently deletes a table (and its data)**	**Chapter 8**
DROP INDEX	**Permanently deletes an index**	**Chapter 8**
DROP VIEW	**Permanently deletes a view**	**Chapter 8**

Basic Data Types

- Numeric
- Character
- Date

TABLE 8.1

SOME COMMON SQL DATA TYPES

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or –134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as *Smith* and *Katzenjammer* are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

MySQL Data Types

- String: char, varchar, text
- Numeric: int, tinyint, boolean, decimal, float
- String: char, text, binary, blob
- Numeric: integer, fixed-point, floating point, boolean
- Date: date, time, datetime

MySQL String Data Types

Data Type	Description	Max Size	Use Case Example
CHAR(n)	Fixed-length string (right-padded with spaces)	255 chars	country codes
VARCHAR(n)	Variable-length string	64K bytes	names, emails, titles
TEXT	Large text data	by subtype	articles, comments, blog
BINARY(n)	Fixed-length binary data	255 bytes	Binary tokens, hashes
VARBINARY(n)	Variable-length binary data	64K bytes	Compressed data
BLOB	Large binary data	by subtype	Images, files, multimedia
ENUM	A string object with a predefined set of possible values	64K values	Status like ('pending', 'shipped')
SET	A string object that can store multiple predefined values (comma-separated)	64 members	Tags like ("sports", "music","tech")

List of MySQL Data Types

MySQL Numeric Data Types - Integer

Data Type	Storage	Range	Example Use Case
TINYINT	1 byte	-128 to 127	Status flags (0 = off, 1 = on)
SMALLINT	2 bytes	-32,768 to 32,767	Age field
MEDIUMINT	3 bytes	-8,388,608 to 8,388,607	Moderate row IDs or counts
INT/INTEGER	4 bytes	-2.1B to 2.1B	User IDs, product IDs
BIGINT	8 bytes	-9.2 quintillion to -9.2 quintillion	Order numbers, financial records

MySQL Numeric Data Types - Decimal Type

- Exact, stored as string-like binary, no precision loss
- Slower for math operations
- DECIMAL(10, 2): 12345678.90
- Financial data, money, tax, rates

Data Type	Description	Example
DECIMAL(5,2)	5 digits total, 2 after decimal precise	-999.99 ~ 999.99
NUMERIC(5,2)	Alias of DECIMAL	

MySQL Numeric Data Types - Floating-Point Type

- Approximate, stored as binary float, can lose precision
- Faster and uses less storage

- DOUBLE -> 3.14159265358979
- Scientific data, measurements

Data Type	Storage	Example Use Case	Precision
FLOAT	4 bytes	Weight: 12.34	~7 digits
DOUBLE	8 bytes	GPS coordinates: 25.036793, 121.564558	~15 to 16 digits

MySQL Numeric Data Types - Boolean Type

Data Type	Example Use Case
BOOLEAN	TRUE or FALSE
BOOL	same as BOOLEAN

MySQL Date and Time Data Types

Data Type	Format	Example Value	Use Case
DATE	YYYY-MM-DD	'2025-04-22'	birthdays
DATETIME	YYYY-MM-DD HH:MM:SS	'2025-04-22 13:45:00'	Exact date & time of an event
TIMESTAMP	YYYY-MM-DD HH:MM:SS	'2025-04-22 05:00:00'	Auto-tracking changes, auditing
TIME	HH:MM:SS	'14:30:00'	Duration, business hours
YEAR	YYYY	'2025'	product release year

Steps to Develop Database

1. Design ER model (Fig 7.1 or Fig 8.1)
2. Create database
3. Create database **schema** (a logical group of database objects, like tables and indexes)
4. Insert data

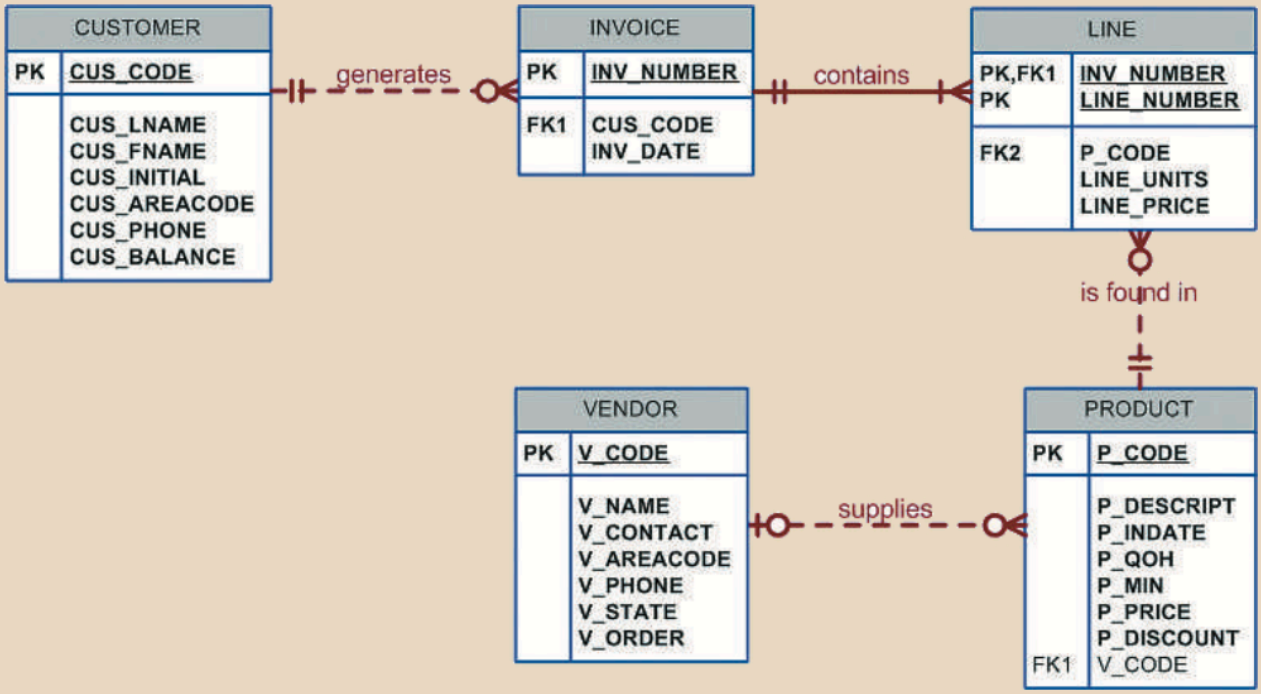
Step1: Analyze Biz Rules to Design ER Model

Step1A: Analyze Biz Rules to Design ER Model

- A customer may generate many invoices. Each invoice is generated by one customer.
- An invoice contains one or more invoice lines. Each invoice line is associated with one invoice.
- Each invoice line references one product. A product may be found in many invoice lines.
- A vendor may supply many products. Some vendors do not yet supply products.
- If a product is vendor-supplied, it is supplied by only a single vendor.
- Some products are not supplied by a vendor.

Step1: Deliver ER Diagram

FIGURE 7.1 THE DATABASE MODEL

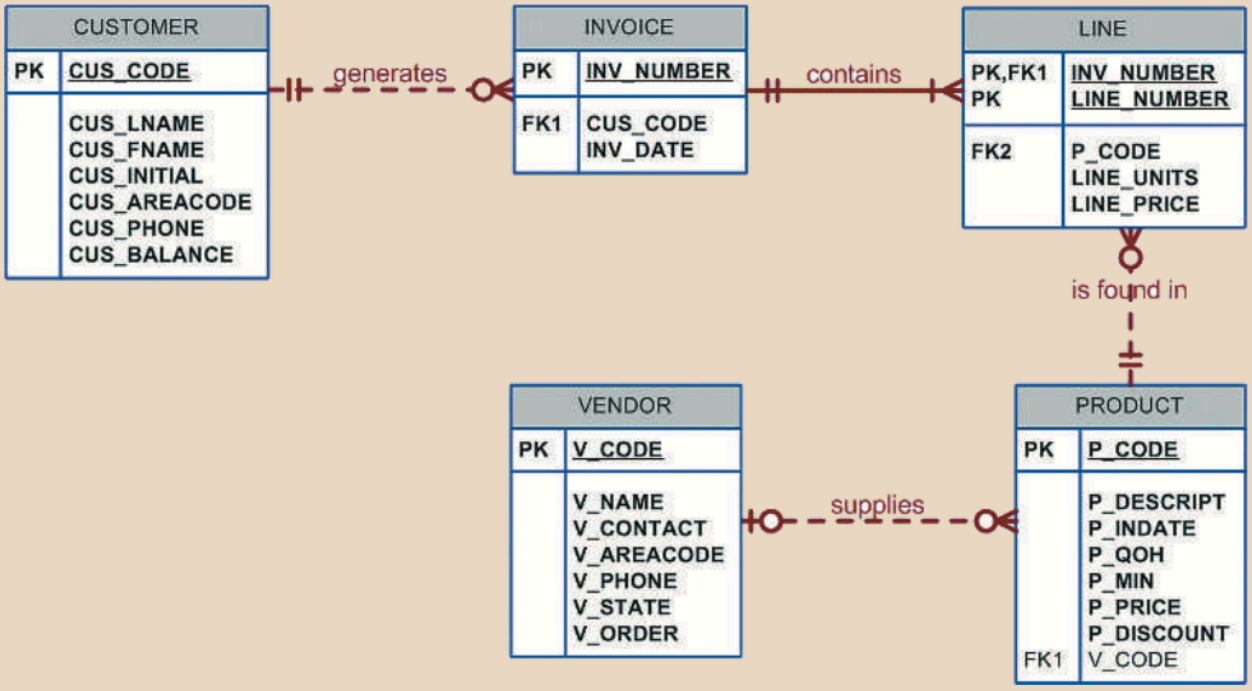


Step2: Create Database (CH07_SALECO)

- Database stores database schema, a group of database objects

Step1B: Deliver ER Diagram

FIGURE 7.1 THE DATABASE MODEL



Step1C: Data Dict

TABLE 8.2

DATA DICTIONARY FOR THE CH08_SALECO DATABASE

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
PRODUCT	P_CODE	Product code	VARCHAR(10)	XXXXXXXXXX	NA	Y	PK	
	P_DESCRIPT	Product description	VARCHAR(35)	Xxxxxxxxxxx	NA	Y		
	P_INDATE	Stocking date	DATE	DD-MON-YYYY	NA	Y		
	P_QOH	Units available	SMALLINT	####	0–9999	Y		
	P_MIN	Minimum units	SMALLINT	####	0–9999	Y		
	P_PRICE	Product price	NUMBER(8,2)	####.##	0.00–9999.00	Y		
	P_DISCOUNT	Discount rate	NUMBER(5,2)	0.##	0.00–0.20	Y		
	V_CODE	Vendor code	INTEGER	###	100–999		FK	VENDOR
VENDOR	V_CODE	Vendor code	INTEGER	#####	1000–9999	Y	PK	
	V_NAME	Vendor name	VARCHAR(35)	Xxxxxxxxxxxxx	NA	Y		
	V_CONTACT	Contact person	VARCHAR(25)	Xxxxxxxxxxxxx	NA	Y		
	V_AREACODE	Area code	CHAR(3)	999	NA	Y		
	V_PHONE	Phone number	CHAR(8)	999–9999	NA	Y		
	V_STATE	State	CHAR(2)	XX	NA	Y		
	V_ORDER	Previous order	CHAR(1)	X	Y or N	Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed-length character data, 1 to 255 characters
VARCHAR = Variable-length character data, 1 to 2,000 characters. VARCHAR is automatically converted to VARCHAR2 in Oracle.
NUMBER = Numeric data. NUMBER(9,2) is used to specify numbers that have two decimal places and are up to nine digits long, including the decimal places. Some RDBMSs permit the use of a MONEY or a CURRENCY data type.
NUMERIC = Numeric data. DBMSs that do not support the NUMBER data type typically use NUMERIC instead.
INT = Integer values only. INT is automatically converted to NUMBER in Oracle.
SMALLINT = Small integer values only. SMALLINT is automatically converted to NUMBER in Oracle.
DATE formats vary. Commonly accepted formats are DD-MON-YYYY, DD-MON-YY, MM/DD/YYYY, and MM/DD/YY.

*Not all the ranges shown here will be illustrated in this chapter. However, you can use these constraints to practice writing your own.

Step2: Create Database (MySQL syntax)

```
CREATE DATABASE [IF NOT EXISTS] database_name;
```

Database (schema) name: IIM_SALECO or EPPS_SALECO

```
CREATE DATABASE EPPS_SALECO;
CREATE DATABASE IF NOT EXISTS EPPS_SALECO;
USE EPPS_SALECO;
```

Step3: Create Database Tables (MySQL syntax)

```
CREATE TABLE [IF NOT EXISTS] table_name (
  column_name1 data_type [column_constraints],
  column_name2 data_type [column_constraints],
  ...
  [table_constraints]
);
```

Step3: Create Database Tables

VENDOR table

Create VENDOR Table

```
CREATE TABLE IF NOT EXISTS VENDOR (
  V_CODE INT,
  V_NAME VARCHAR(35) NOT NULL,
  V_CONTACT VARCHAR(15) NOT NULL,
  V_AREACODE CHAR(3) NOT NULL,
  V_PHONE CHAR(8) NOT NULL,
  V_STATE CHAR(2) NOT NULL,
  V_ORDER CHAR(1) NOT NULL,
```

```
  PRIMARY KEY (V_CODE)
);
```

PRODUCT table

Create PRODUCT Table

```
CREATE TABLE IF NOT EXISTS PRODUCT (
  P_CODE VARCHAR(10),
  P_DESCRIPT VARCHAR(35) NOT NULL,
  P_INDATE DATETIME NOT NULL,
  P_QOH INTEGER NOT NULL,
  P_MIN INTEGER NOT NULL,
  P_PRICE NUMERIC(8,2) NOT NULL,
  P_DISCOUNT NUMERIC(4,2) NOT NULL,
  V_CODE INTEGER,
  PRIMARY KEY (P_CODE),
  FOREIGN KEY (V_CODE) REFERENCES VENDOR (V_CODE)
);
```

STEP4: Insert Data

VENDOR table

STEP4: Insert Data (MySQL Syntax)

```
/* basic syntax */
INSERT INTO table_name (column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);

/* insert multiple rows */
INSERT INTO table_name (column1, column2)
VALUES
  (value1a, value2a),
  (value1b, value2b),
  (value1c, value2c);

/* insert without specifying columns (must match column order) */
```

```
INSERT INTO table_name
VALUES (value1, value2, ..., valueN);
```

Insert Into VENDOR Table

```
INSERT INTO VENDOR VALUES(21225,'Bryson, Inc.'    ,'Smithson','615','223-
3234','TN','Y');
INSERT INTO VENDOR VALUES(21226,'SuperLoo, Inc.'  ,'Flushing','904','215-
8995','FL','N');
INSERT INTO VENDOR VALUES(21231,'D&E Supply'      ,'Singh'   ,'615','228-
3245','TN','Y');
INSERT INTO VENDOR VALUES(21344,'Gomez Bros.'     ,'Ortega'  ,'615','889-
2546','KY','N');
INSERT INTO VENDOR VALUES(22567,'Dome Supply'     ,'Smith'   ,'901','678-
1419','GA','N');
INSERT INTO VENDOR VALUES(23119,'Randsets Ltd.'   ,'Anderson','901','678-
3998','GA','Y');
INSERT INTO VENDOR VALUES(24004,'Brackman Bros.'  ,'Browning','615','228-
1410','TN','N');
INSERT INTO VENDOR VALUES(24288,'ORDVA, Inc.'     ,'Hakford' ,'615','898-
1234','TN','Y');
INSERT INTO VENDOR VALUES(25443,'B&K, Inc.'       ,'Smith'   ,'904','227-
0093','FL','N');
INSERT INTO VENDOR VALUES(25501,'Damal Supplies'  ,'Smythe'  ,'615','890-
3529','TN','N');
INSERT INTO VENDOR VALUES(25595,'Rubicon Systems' ,'Orton'   ,'904','456-
0092','FL','Y');
```

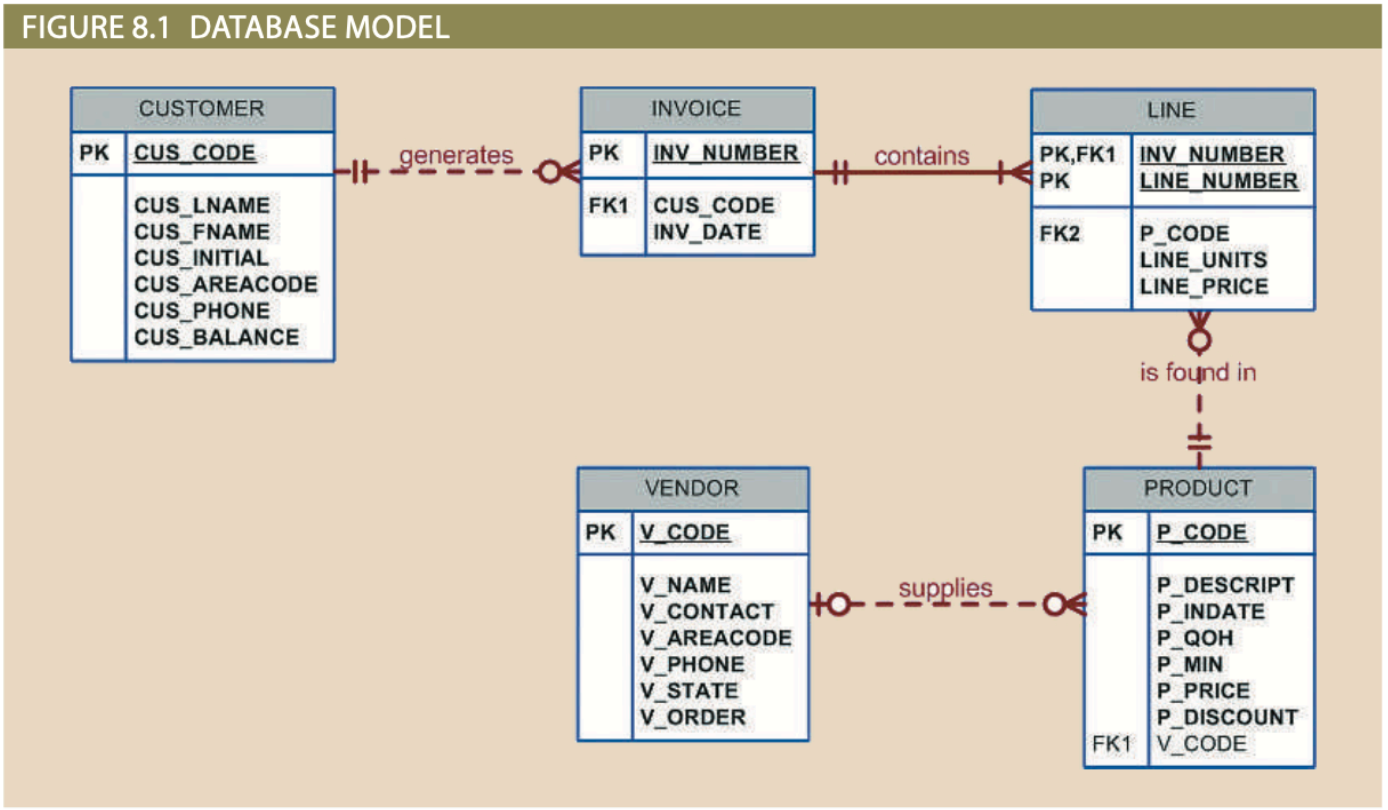
PRODUCT table

```
# Insert Into PRODUCT table
```sql
INSERT INTO PRODUCT VALUES('11QER/31','Power painter, 15 psi., 3-nozzle'
,'2021-11-03',  8,  5,109.99,0.00,25595);
INSERT INTO PRODUCT VALUES('13-Q2/P2','7.25-in. pwr. saw blade'
,'2021-12-13', 32, 15, 14.99,0.05,21344);
INSERT INTO PRODUCT VALUES('14-Q1/L3','9.00-in. pwr. saw blade'
,'2021-11-13', 18, 12, 17.49,0.00,21344);
INSERT INTO PRODUCT VALUES('1546-QQ2','Hrd. cloth, 1/4-in., 2x50'
,'2022-01-15', 15,  8, 39.95,0.00,23119);
INSERT INTO PRODUCT VALUES('1558-QW1','Hrd. cloth, 1/2-in., 3x50'
,'2022-01-15', 23,  5, 43.99,0.00,23119);
INSERT INTO PRODUCT VALUES('2232/QTY','B&D jigsaw, 12-in. blade'
,'2021-12-30',  8,  5,109.92,0.05,24288);
INSERT INTO PRODUCT VALUES('2232/QWE','B&D jigsaw, 8-in. blade'
```
```

```
,'2021-12-24',  6,  5, 99.87,0.05,24288);
INSERT INTO PRODUCT VALUES('2238/QPD','B&D cordless drill, 1/2-in.'
,'2022-01-20', 12,  5, 38.95,0.05,25595);
INSERT INTO PRODUCT VALUES('23109-HB','Claw hammer'
,'2022-01-20', 23, 10,  9.95,0.10,21225);
INSERT INTO PRODUCT VALUES('23114-AA','Sledge hammer, 12 lb.'
,'2022-01-02',  8,  5, 14.40,0.05,NULL);
INSERT INTO PRODUCT VALUES('54778-2T','Rat-tail file, 1/8-in. fine'
,'2021-12-15', 43, 20,  4.99,0.00,21344);
INSERT INTO PRODUCT VALUES('89-WRE-Q','Hicut chain saw, 16 in.'
,'2022-02-07', 11,  5,256.99,0.05,24288);
INSERT INTO PRODUCT VALUES('PVC23DRT','PVC pipe, 3.5-in., 8-ft'
,'2022-02-20',188, 75,  5.87,0.00,NULL);
INSERT INTO PRODUCT VALUES('SM-18277','1.25-in. metal screw, 25'
,'2022-03-01',172, 75,  6.99,0.00,21225);
INSERT INTO PRODUCT VALUES('SW-23116','2.5-in. wd. screw, 50'
,'2022-02-24',237,100,  8.45,0.00,21231);
INSERT INTO PRODUCT VALUES('WR3/TT3' ,'Steel matting, 4''x8''x1/6", .5"
mesh','2022-01-17', 18,  5,119.95,0.10,25595);
```

Sample Database Model

FIGURE 8.1 DATABASE MODEL



Data in Database

FIGURE 8.2 VENDOR AND PRODUCT TABLES

Table name: VENDOR **Database name: Ch08_SaleCo**

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-17	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-17	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-17	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-18	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-18	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-17	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-17	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-18	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-18	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-18	8	5	14.4	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-17	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-18	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-18	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-18	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-18	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-18	18	5	119.95	0.10	25595

SQL Queries

- Many SQL queries are used to perform actions such as adding or deleting rows or changing attribute values within tables
- Data retrieval is done in SQL using a SELECT query
- A SQL set-oriented command works over a set of rows
- A SELECT query specifies which data should be retrieved and how it should be filtered, aggregated, and displayed

Basic SELECT Queries

Each clause in a SELECT query performs the following functions:

- SELECT – specifies the attributes to be returned by the query
- FROM – specifies the table(s) from which the data will be retrieved
- WHERE – filters the rows of data based on provided criteria
- GROUP BY – groups the rows of data into collections based on sharing the same values in one or more attributes

- HAVING – filters the groups formed in the GROUP BY clause based on provided criteria
- ORDER BY – sorts the final query result rows in ascending or descending order based on the values of one or more attributes

SELECT Statement Options

```
SELECT        columnlist
FROM          tablelist;
```

- A wildcard character is a symbol that can be used as a general substitute for other characters or commands

SQL Data Manipulation Language (DML)

- Many SQL DML are used to perform actions such as adding or deleting rows or changing attribute values within tables
- Data retrieval is done using SELECT which specifies what data should be retrieved and how it should be filtered, aggregated, and displayed

Basic SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[GROUP BY column]
[HAVING condition]
[ORDER BY column [ASC|DESC]]
[LIMIT number OFFSET offset];
```

SELECT Clause

- SELECT – specifies the attributes to be returned (column name or *)
- FROM – specifies the table(s)
- WHERE – filters the rows of data

- GROUP BY – groups the rows of data into collections based on columns
- HAVING – filters the groups formed by GROUP BY clause
- ORDER BY – sorts the final query result rows in ascending or descending order by columns

Use Wildcard in Expression

A wildcard character is a symbol that can be used as a general substitute for other characters or commands

- * : all columns
- % : matches zero or more characters
- _ : matches exactly one character

```
SELECT * FROM PRODUCT WHERE P_CODE LIKE '15%';
SELECT * FROM PRODUCT WHERE P_CODE LIKE '2232/Q__';
```

Column Definition of PRODUCT and VENDOR

TABLE 8.2

DATA DICTIONARY FOR THE CH08_SALECO DATABASE

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
PRODUCT	P_CODE	Product code	VARCHAR(10)	XXXXXXXXXX	NA	Y	PK	
	P_DESCRIPT	Product description	VARCHAR(35)	Xxxxxxxxxxx	NA	Y		
	P_INDATE	Stocking date	DATE	DD-MON-YYYY	NA	Y		
	P_QOH	Units available	SMALLINT	####	0–9999	Y		
	P_MIN	Minimum units	SMALLINT	####	0–9999	Y		
	P_PRICE	Product price	NUMBER(8,2)	####.##	0.00–9999.00	Y		
	P_DISCOUNT	Discount rate	NUMBER(5,2)	0.##	0.00–0.20	Y		
	V_CODE	Vendor code	INTEGER	###	100–999		FK	VENDOR
VENDOR	V_CODE	Vendor code	INTEGER	#####	1000–9999	Y	PK	
	V_NAME	Vendor name	VARCHAR(35)	Xxxxxxxxxxxxx	NA	Y		
	V_CONTACT	Contact person	VARCHAR(25)	Xxxxxxxxxxxxx	NA	Y		
	V_AREACODE	Area code	CHAR(3)	999	NA	Y		
	V_PHONE	Phone number	CHAR(8)	999–9999	NA	Y		
	V_STATE	State	CHAR(2)	XX	NA	Y		
	V_ORDER	Previous order	CHAR(1)	X	Y or N	Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed-length character data, 1 to 255 characters
VARCHAR = Variable-length character data, 1 to 2,000 characters. VARCHAR is automatically converted to VARCHAR2 in Oracle.
NUMBER = Numeric data. NUMBER(9,2) is used to specify numbers that have two decimal places and are up to nine digits long, including the decimal places. Some RDBMSs permit the use of a MONEY or a CURRENCY data type.
NUMERIC = Numeric data. DBMSs that do not support the NUMBER data type typically use NUMERIC instead.
INT = Integer values only. INT is automatically converted to NUMBER in Oracle.
SMALLINT = Small integer values only. SMALLINT is automatically converted to NUMBER in Oracle.
DATE formats vary. Commonly accepted formats are DD-MON-YYYY, DD-MON-YY, MM/DD/YYYY, and MM/DD/YY.

*Not all the ranges shown here will be illustrated in this chapter. However, you can use these constraints to practice writing your own.

Table: CH07_SALECO PRODUCT

Database (Schema): CH07_SALECO Table: PRODUCT

FIGURE 7.2 SELECT AN ENTIRE TABLE

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-17	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-17	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-17	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-18	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-18	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-17	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-17	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-18	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-18	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-18	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-17	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-18	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-18	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-18	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-18	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-18	18	5	119.95	0.10	25595

Select an Entire PRODUCT Table

```
SELECT *
FROM EPPS_SALECO.PRODUCT;

USE EPPS_SALECO;
SELECT *
FROM PRODUCT;
```

Select with a Column List

```
SELECT P_CODE, P_DESCRIPT, P_PRICE, P_QOH
FROM EPPS_SALECO.PRODUCT;
```

Using Column Aliases

```
SELECT P_CODE, P_DESCRIPT AS DESCRIPTION, P_PRICE AS "UNIT PRICE", P_QOH AS
QTY
FROM PRODUCT;
```

Using Computed Columns

```
SELECT P_DESCRIPT AS DESCRIPTION, P_PRICE AS "UNIT PRICE", P_QOH AS QTY,
P_QOH * P_PRICE AS "TOTAL VALUE"
FROM PRODUCT;
```

Numeric Calculation

```
SELECT
P_PRICE as ORG_PRICE,
```

```
P_DISCOUNT as DISCOUNT,
P_PRICE * (1 - P_DISCOUNT) as PROD_PRICE
FROM PRODUCT;
```

Date Arithmetic

- Follow the rules of precedence
- +, -, *, /, div, %, mod

Date Arithmetic

```
SELECT NOW() + INTERVAL 7 DAY;
SELECT CURDATE() - INTERVAL 1 MONTH;
SELECT '2025-04-01' + INTERVAL 1 DAY;
```

Listing Unique Values

SQL's DISTINCT clause produces a list of only those values that are different from one another

```
SELECT DISTINCT V_CODE
FROM PRODUCT;
```

FROM Clause Options

- The FROM clause of the query specifies the table or tables from which the data is to be retrieved
- Only columns in the table specified in the FROM clause are available throughout the rest of the query

FROM Clause Options

- The FROM clause specifies table(s) which is involved
- Only columns in tables in FROM clause are available throughout the rest of the query
- Multiple tables must be combined using a type of JOIN operation

ORDER BY Clause Options

```
SELECT  columnlist
FROM            tablelist
[ORDER BY       columnlist [ASC|DESC] ];
```

```
SELECT P_CODE, P_DESCRIPT, P_QOH, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE;
```

```
SELECT P_CODE, P_DESCRIPT, P_QOH, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE DESC;
```

```
SELECT EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_PHONE
FROM EMPLOYEE
ORDER BY EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

WHERE Clause Options

- Comparison operator: =, <, <=, >, >=, <> or !=

```
SELECT columnlist
FROM tablelist
[WHERE conditionlist ];
```

Using Comparison Operator on Numeric Attribute

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 21344;

SELECT P_DESCRIPT, P_QOH, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_PRICE <= 10;
```

Using Comparison Operator on Character Attribute

```
SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_CODE < '1558-QW1';
```

Using Comparison Operator on Date Attribute

```
SELECT P_DESCRIPT, P_QOH, P_MIN, P_PRICE, P_INDATE
FROM PRODUCT
WHERE P_INDATE >= '2021-11-05';
```

Logical Operators: AND, OR and NOT

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE P_PRICE < 50 AND P_INDATE > '2021-01-01';

/* use parentheses and compare below two select statements */
SELECT P_DESCRIPT, P_PRICE, V_CODE
FROM PRODUCT
WHERE (V_CODE = 25595 OR V_CODE = 24288) AND P_PRICE > 100;
```

```
SELECT P_DESCRIPT, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 25595 OR V_CODE = 24288 AND P_PRICE > 100;
-- AND before OR --

SELECT *
FROM PRODUCT
WHERE NOT (V_CODE = 21344);
```

Special Operators in WHERE Clause

- BETWEEN – Used to check whether an attribute value is within a range
- IN – Used to check whether an attribute value matches any value within a value list
- LIKE – Used to check whether an attribute value matches a given string pattern
- IS NULL – Used to check whether an attribute value is null
- NOT – Used to negate a condition

Illustrations of Special Operators

```
SELECT *
FROM PRODUCT
WHERE P_PRICE BETWEEN 50.00 AND 100.00;
SELECT *
FROM PRODUCT
WHERE V_CODE IN (21344, 24288);
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE V_CONTACT LIKE 'Smith%';
-- wildcard % for zero or more chars, _ for any one char
SELECT P_CODE, P_DESCRIPT, V_CODE
FROM PRODUCT
WHERE V_CODE IS NULL;
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE UPPER(V_CONTACT) NOT LIKE 'SMITH%';
```

MySQL Comparison Operators

Symbol or keyword(s)	Description
=, !=, <>	Equal, Not equal

Symbol or keyword(s)	Description
>, >=, <, <=	Great / Less than or equal to
is null, is not null	check null or not
between and, not between and	within a range
in, not in	match a value in a list
like, not like	match a pattern

MySQL Booleans or Conditions

Conditions: not, and, or Booleans

```
create table bachelor (name     varchar(100), employed_flag bool);

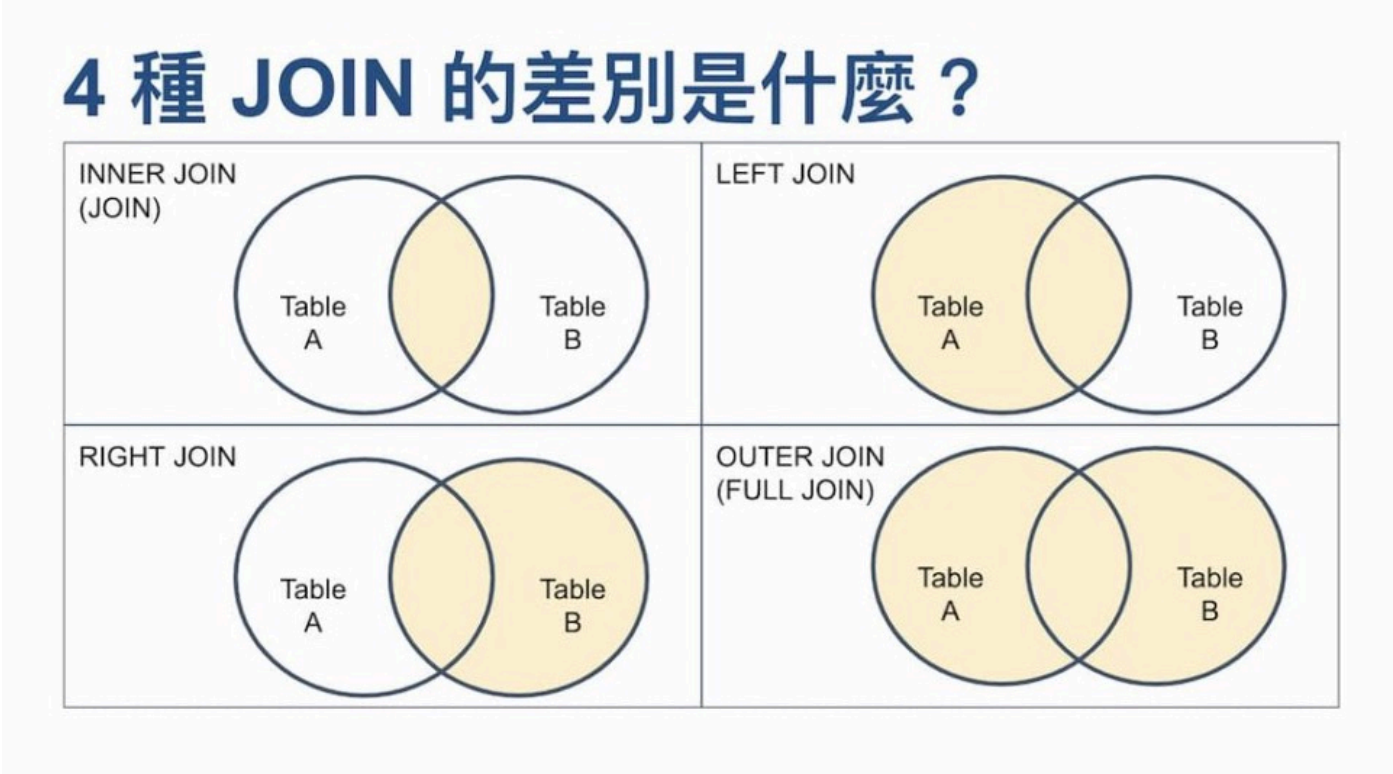
insert into bachelor(name, employed_flag)
values ('Hector Handsome', true),('Frank Freeloader', false);
select * from bachelor where employed_flag is true;
select * from bachelor where employed_flag;
select * from bachelor where employed_flag = true;
select * from bachelor where employed_flag != false;
select * from bachelor where employed_flag = 1;
select * from bachelor where employed_flag != 0;
```

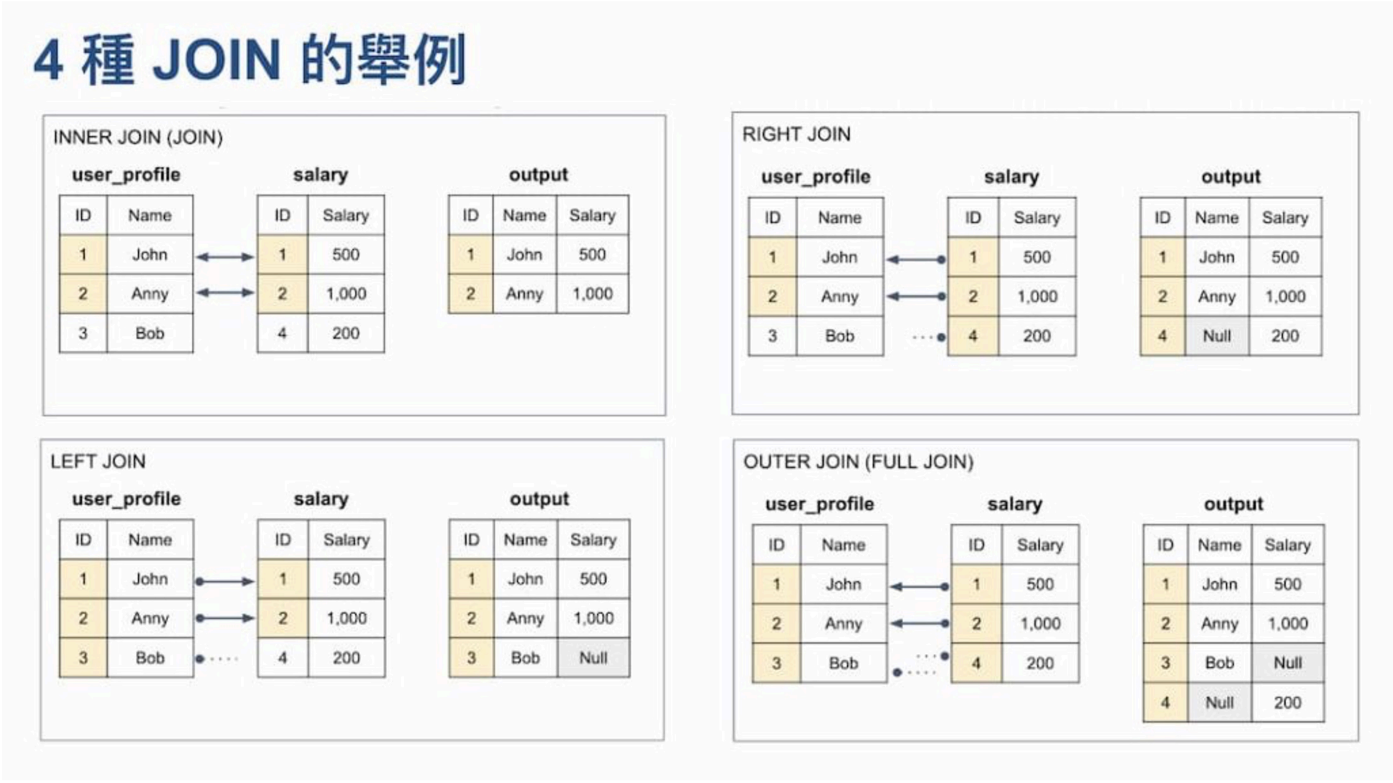
JOIN Operations

JOIN operators are used to combine data from multiple tables

- Inner joins return only rows from the tables that match on a common value
- Outer joins return the same matched rows as the inner join, plus unmatched rows from one table or the other
    - Left (outer) join
    - Right (outer) join
    - Full (outer) join

JOIN Illustration





Three Ways to Do Inner Join (Join)

```
-- JOIN USING
SELECT column-list FROM table1 JOIN table2 USING (common-column)

-- JOIN ON
SELECT column-list FROM table1 JOIN table2 ON join-condition

-- Old-style JOIN
SELECT column-list FROM table1, table2 WHERE table1.column = table2.column
```

- In practice, **JOIN ON** is typically considered as a preference.

Example of JOIN USING

```
SELECT P_CODE, P_DESCRIPT, V_CODE, V_NAME, V_AREACODE, V_PHONE
FROM PRODUCT JOIN VENDOR USING (V_CODE);
```

Example of JOIN ON

```
SELECT INVOICE.INV_NUMBER, PRODUCT.P_CODE, P_DESCRIPT, LINE_UNITS,
LINE_PRICE
FROM INVOICE
JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;

-- Compare to JOIN ON
SELECT INV_NUMBER, P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE
FROM INVOICE
JOIN LINE USING(INV_NUMBER)
JOIN PRODUCT USING(P_CODE);
```

Example of Old-Style JOIN

```
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_NAME
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;

-- Compare to JOIN USING
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_NAME
FROM PRODUCT JOIN VENDOR USING(V_CODE);

-- Compare to JOIN ON
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_NAME
FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE;
```

- The task of joining the tables is split across both the FROM and WHERE which makes complex queries more difficult to maintain
- They are susceptible to undetected errors

Illustrate Why Old-Style Join is Not Preferred

```
-- Get wrong result and easy to find no condition when join PRODUCT
SELECT CUS_FNAME, CUS_LNAME, V_NAME
FROM CUSTOMER
JOIN INVOICE ON CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
JOIN PRODUCT
JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
WHERE V_STATE = 'TN';

-- Get wrong result and hard to debug
SELECT CUS_FNAME, CUS_LNAME, V_NAME
FROM CUSTOMER, INVOICE, LINE, PRODUCT, VENDOR
WHERE V_STATE = 'TN'
AND CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND PRODUCT.V_CODE = VENDOR.V_CODE;
```

Outer Joins

Three types of outer join: Left (outer) join, Right (outer) join, Full (outer) join

Left Outer Join

```
SELECT column-list
FROM table1 LEFT[OUTER] JOIN table2 ON join-condition

SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR
LEFT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

Right Outer Join

```
SELECT column-list
FROM table1 RIGHT[OUTER] JOIN table2 ON join-condition

SELECT P_CODE, VENDOR.V_CODE, V_NAME
```

```
FROM VENDOR
RIGHT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;

SELECT VENDOR.V_CODE, V_NAME, P_CODE
FROM PRODUCT
RIGHT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
WHERE P_CODE IS NULL;
```

Full Outer Join (Not Support in MySQL)

```
SELECT column-list
FROM table1 FULL[OUTER] JOIN table2 ON join-condition

SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR
FULL JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

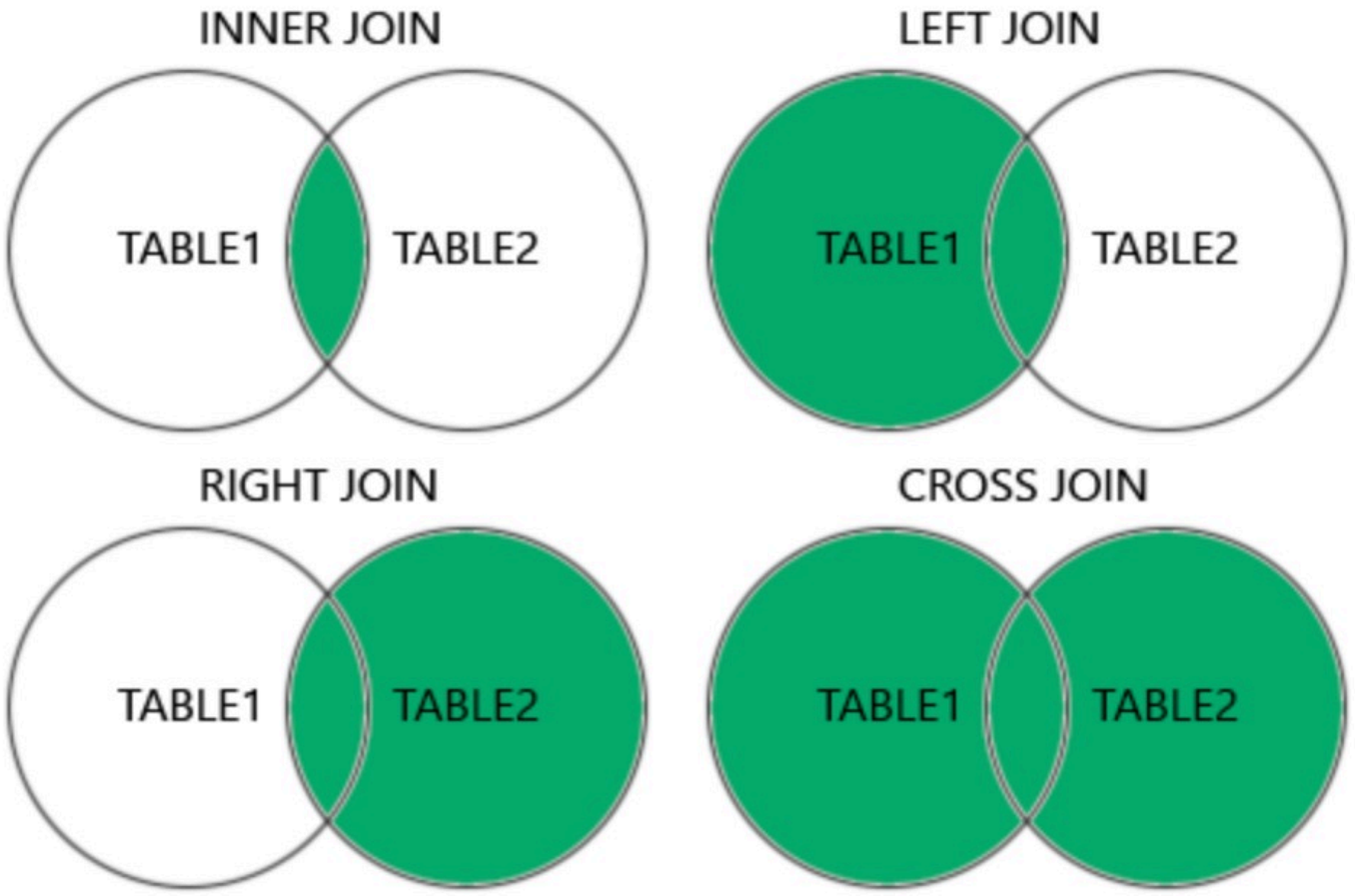
Cross Join

- A cross join performs a relational product (also known as the Cartesian product) of two tables.
- Despite the name, CROSS JOIN is not truly a join operation because it does not unite the rows of the tables based on a common attribute.

```
SELECT column-list FROM table1 CROSS JOIN table2

SELECT * FROM INVOICE CROSS JOIN LINE;
```

JOINs in MySQL



Joining Tables with an Alias

Using a table alias allows the database programmer to improve the maintainability

```
SELECT P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM
PRODUCT P
JOIN VENDOR V ON P.V_CODE = V.V_CODE;
```

Recursive Joins

A query that joins a table to itself

```
SELECT E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME
FROM EMP E
JOIN EMP M ON E.EMP_MGR = M.EMP_NUM;
```

Aggregate Processing

SQL provides useful aggregate functions that count, find minimum and maximum values, calculate averages, etc.

- Count
- MIN and MAX
- SUM and AVG

Count

```
SELECT COUNT(P_CODE)
FROM PRODUCT;

SELECT COUNT(P_PRICE)
FROM PRODUCT
WHERE P_PRICE < 10;

-- count how many V_CODE in PRODUCT which is not NULL
SELECT COUNT(V_CODE)
FROM PRODUCT;

-- count how many rows in the table
SELECT COUNT(*)
FROM PRODUCT;

SELECT COUNT(DISTINCT V_CODE) AS "COUNT DISTINCT"
FROM PRODUCT;
```

MIN and MAX

The MIN and MAX functions help you find answers to problems such as the highest and lowest (maximum and minimum) prices in the PRODUCT table.

```
SELECT MAX(P_PRICE) AS MAXPRICE, MIN(P_PRICE) as MINPRICE
FROM PRODUCT;
```

SUM and AVG

```
SELECT SUM(CUS_BALANCE) AS TOTAL_BALANCE
FROM CUSTOMER;
```

```
SELECT SUM(P_QOH * P_PRICE) as TOTAL_VALUE
FROM PRODUCT;

SELECT AVG(P_PRICE) AS AVG_PRICE
FROM PRODUCT;
```

Grouping Data (1)

```
SELECT columnlist
FROM tablelist
[WHERE conditionlist]
[GROUP BY columnlist]
[ORDER BY columnlist [ASC|DESC]];

SELECT V_CODE, AVG(P_PRICE) AS AVG_PRICE
FROM PRODUCT
GROUP BY V_CODE;

SELECT VENDOR.V_CODE, V_NAME, COUNT(P_CODE) AS NUMPRODS, AVG(P_PRICE) AS
AVGPRICE
FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
GROUP BY V_CODE
ORDER BY V_NAME;
```

Grouping Data (2)

```
-- Get execution error
SELECT VENDOR.V_CODE, V_NAME, P_QOH, COUNT(P_CODE) AS NUMPRODS, AVG(P_PRICE)
AS AVGPRICE
FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
GROUP BY V_CODE
ORDER BY V_NAME;

-- Fixed 1: sum of P_QOH
SELECT VENDOR.V_CODE, V_NAME, SUM(P_QOH), COUNT(P_CODE) AS NUMPRODS,
AVG(P_PRICE) AS AVGPRICE
FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
GROUP BY V_CODE
ORDER BY V_NAME;

-- Fixed 2: put P_QOH into group by
SELECT VENDOR.V_CODE, V_NAME, P_QOH, COUNT(P_CODE) AS NUMPRODS, AVG(P_PRICE)
AS AVGPRICE
FROM PRODUCT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
GROUP BY V_CODE, P_QOH
ORDER BY V_NAME;
```

HAVING Clause

```
SELECT columnlist FROM tablelist
[WHERE conditionlist]
[GROUP BY columnlist]
[HAVING conditionlist]
[ORDER BY columnlist [ASC|DESC]];

SELECT V_CODE, COUNT(P_CODE) AS NUMPRODS
FROM PRODUCT
GROUP BY V_CODE
HAVING AVG(P_PRICE) < 10
ORDER BY V_CODE;

SELECT P.V_CODE, V_NAME, SUM(P_QOH * P_PRICE) AS TOTCOST
FROM PRODUCT P JOIN VENDOR V ON P.V_CODE = V.V_CODE
WHERE P_DISCOUNT > 0
GROUP BY V_CODE, V_NAME
HAVING (SUM(P_QOH * P_PRICE) > 500)
ORDER BY SUM(P_QOH * P_PRICE) DESC;
```

Subqueries

We want to generate a list of vendors who do not provide products.

```
-- Right outer join
SELECT VENDOR.V_CODE, V_NAME
FROM PRODUCT
RIGHT JOIN VENDOR ON PRODUCT.V_CODE = VENDOR.V_CODE
WHERE P_CODE IS NULL;

-- Subquery
SELECT V_CODE, V_NAME
FROM VENDOR
WHERE V_CODE NOT IN (
    SELECT V_CODE FROM PRODUCT WHERE V_CODE IS NOT NULL);
```

WHERE Subqueries

```
-- List all customers who order a claw hammer
SELECT P_CODE, P_PRICE
FROM PRODUCT
WHERE P_PRICE >=
```

```
    (SELECT AVG(P_PRICE) FROM PRODUCT);

SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
FROM CUSTOMER
JOIN INVOICE USING (CUS_CODE)
JOIN LINE USING (INV_NUMBER)
JOIN PRODUCT USING (P_CODE)
WHERE P_CODE = (
    SELECT P_CODE
    FROM PRODUCT
    WHERE P_DESCRIPT = 'Claw hammer');

SELECT DISTINCT CUSTOMER.CUS_CODE, CUS_LNAME, CUS_FNAME
FROM CUSTOMER
JOIN INVOICE ON CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
JOIN PRODUCT ON PRODUCT.P_CODE = LINE.P_CODE
WHERE P_DESCRIPT = 'Claw hammer';
```

IN Subqueries

List all customers who have purchased hammers, saws, or saw blades.

```
SELECT DISTINCT CUSTOMER.CUS_CODE, CUS_LNAME, CUS_FNAME
FROM CUSTOMER
JOIN INVOICE ON CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE
WHERE PRODUCT.P_CODE IN
  (SELECT P_CODE
   FROM PRODUCT
   WHERE P_DESCRIPT LIKE '%hammer%' OR P_DESCRIPT LIKE '%saw%');
```

HAVING Subqueries

List all products with a total quantity sold greater than the average quantity sold

```
SELECT P_CODE, SUM(LINE_UNITS) AS TOTALUNITS
FROM LINE
GROUP BY P_CODE
HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS) FROM LINE);
```

Multirow Subquery Operators: ALL and any

Which products cost more than all individual products provided by vendors from Florida

```
SELECT P_CODE, P_QOH * P_PRICE AS TOTALVALUE
FROM PRODUCT
WHERE P_QOH * P_PRICE >
    ALL (SELECT P_QOH * P_PRICE
         FROM PRODUCT
         WHERE V_CODE IN
         (SELECT V_CODE
          FROM VENDOR
          WHERE V_STATE = 'FL'));
```

- Greater than ALL" is equivalent to "greater than the highest product cost of the list
- ANY operator to compare a single value to a list of values and select only the rows for which the inventory cost is greater than any value in the list
- Use the equal to ANY operator, which would be the equivalent of the IN operator.

FROM Subqueries

List all customers who purchased both products ('13-Q2/P2', '23109-HB'), not just one.

```
SELECT DISTINCT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME
FROM CUSTOMER
JOIN
    (SELECT INVOICE.CUS_CODE
     FROM INVOICE
     JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
     WHERE P_CODE = '13-Q2/P2') CP1
ON CUSTOMER.CUS_CODE = CP1.CUS_CODE
JOIN
    (SELECT INVOICE.CUS_CODE
     FROM INVOICE
     JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
     WHERE P_CODE = '23109-HB') CP2
ON CP1.CUS_CODE = CP2.CUS_CODE;
```

Attribute List Subqueries (1)

List the difference between each product's price and the average product price

```
SELECT
  P_CODE, P_PRICE,
  (SELECT AVG(P_PRICE) FROM PRODUCT) AS AVGPRICE,
  P_PRICE - (SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF
FROM PRODUCT;
```

Attribute List Subqueries (2)

List the product code, the total sales by product, and the contribution by employee of each product's sales.

```
SELECT
P_CODE,
SUM(LINE_UNITS * LINE_PRICE) AS SALES,
(SELECT COUNT(*) FROM EMPLOYEE) AS ECOUNT,
SUM(LINE_UNITS * LINE_PRICE)/(SELECT COUNT(*) FROM EMPLOYEE) AS CONTRIB
FROM LINE
GROUP BY P_CODE;

SELECT P_CODE, SALES, ECOUNT, SALES/ECOUNT AS CONTRIB
FROM (SELECT P_CODE,
             SUM(LINE_UNITS * LINE_PRICE) AS SALES,
             (SELECT COUNT(*) FROM EMPLOYEE) AS ECOUNT
      FROM LINE
      GROUP BY P_CODE) AS T;
```

Correlated Subqueries (Definition)

- Inner subquery
    - Inner subqueries execute independently.
    - The inner sub-query executes first; its **output** is used by the outer query, which then executes until the last outer query finishes (the first SQL statement in the code).
- Correalted subquery

- A subquery that executes once for each row in the outer query.
- The inner query is related to the outer query
- The inner query references a column of the outer subquery.

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

Correlated Subqueries (Example)

List all product sales in which the units sold value is greater than the average units sold value for that product (as opposed to the average for all products).

1. Compute the average units sold for a product.
2. Compare the average computed in Step 1 to the units sold in each sale row, and then select only the rows in which the number of units sold is greater.

Correlated Subqueries (SQL)

```
SELECT INV_NUMBER, P_CODE, LINE_UNITS
FROM LINE LS
WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
                       FROM LINE LA
                       WHERE LA.P_CODE = LS.P_CODE);

SELECT INV_NUMBER, P_CODE, LINE_UNITS, (SELECT AVG(LINE_UNITS)
                                        FROM LINE LX
                                        WHERE LX.P_CODE = LS.P_CODE) AS AVG
FROM LINE LS
WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
                       FROM LINE LA
                       WHERE LA.P_CODE = LS.P_CODE);
```

Correlated Subqueries (Exists)

```
-- list all vendors, but only if there are products to order.
SELECT *
FROM VENDOR
WHERE EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN * 2);

-- list the names of all customers who have placed an order lately.
```

```
SELECT CUS_CODE, CUS_LNAME, CUS_FNAME
FROM CUSTOMER
WHERE EXISTS (SELECT CUS_CODE
              FROM INVOICE
              WHERE INVOICE.CUS_CODE = CUSTOMER.CUS_CODE);
```

Correlated Subqueries (Example of Exists)

Suppose that you want to know what vendors you must contact to order products that are approaching the minimum quantity-on-hand value that is less than double the minimum quantity.

```
SELECT V_CODE, V_NAME
FROM VENDOR
WHERE EXISTS (SELECT *
              FROM PRODUCT
              WHERE P_QOH < P_MIN * 2 AND VENDOR.V_CODE = PRODUCT.V_CODE);
```

Built-in SQL Functions

- Basic Functions

```
SELECT pi();
SELECT UPPER("hello world");
SELECT ROUND(2.71828);
SELECT ROUND(2.71828, 2);
SELECT ROUND(PI());
SELECT NOW();
SELECT CURDATE();
SELECT CURTIME();
```

- Aggregate Functions: count(), max(), min(), sum(), avg()

MySQL String Functions

```
SELECT CONCAT(EMP_FNAME, " ", EMP_LNAME)
FROM EMP;
```

```
SELECT FORMAT(P_QOH * P_PRICE, 0) as Total_Value
FROM PRODUCT
-- LEFT and RIGHT
SELECT LEFT(EMP_LNAME, 3)
FROM EMP;
-- UPPER and LOWER
SELECT UPPER(LEFT(EMP_LNAME, 3))
FROM EMP;
-- Others: SUBSTRING, TRIM, LTRIM, RTRIM
```

MySQL Date/Time Functions

MySQL Numeric Functions

MySQL Conversion Functions

Relational Set Operators (UNION)

```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER
UNION
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER_2;
```

Relational Set Operators (UNION ALL)

```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER
UNION ALL
```

```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER_2;
```

Relational Set Operators (INTERSECT)

List the customer codes for all customers who are in area code 615 and who have made purchases. (If a customer has made a purchase, there must be an invoice record for that customer.)

```
-- MySQL does not support INTERSECT
SELECT CUS_CODE FROM CUSTOMER WHERE CUS_AREACODE = "615"
INTERSECT
SELECT DISTINCT CUS_CODE FROM INVOICE;

-- Use Join instead of
SELECT DISTINCT C.CUS_CODE
FROM CUSTOMER C
INNER JOIN INVOICE I ON C.CUS_CODE = I.CUS_CODE
WHERE C.CUS_AREACODE = '615';
```

Relational Set Operators (MINUS / EXCEPT)

```
-- MySQL does not support MINUS
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER
MINUS
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER_2;

-- Use Join instead of
SELECT C.CUS_LNAME, C.CUS_FNAME, C.CUS_INITIAL, C.CUS_AREACODE, C.CUS_PHONE
FROM CUSTOMER C
LEFT JOIN CUSTOMER_2 C2
ON C.CUS_LNAME = C2.CUS_LNAME
AND C.CUS_FNAME = C2.CUS_FNAME
AND C.CUS_INITIAL = C2.CUS_INITIAL
AND C.CUS_AREACODE = C2.CUS_AREACODE
AND C.CUS_PHONE = C2.CUS_PHONE
WHERE C2.CUS_LNAME IS NULL;
```

Crafting SELECT Queries

- Know Your Data: the importance of understanding the data model that you are working in cannot be overstated
- Know the Problem: understand the question you are attempting to answer
- Build clauses in the following order
    - FROM
    - WHERE
    - GROUP BY
    - HAVING
    - SELECT
    - ORDER BY

Review Questions

- Explain the difference between an ORDER BY clause and a GROUP BY clause.
- What three join types are included in the OUTER JOIN classification?
- What are the four categories of SQL functions

Homework #C

資料庫課程作業(C)