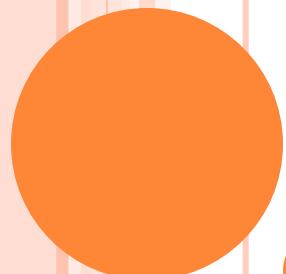


LAB 1. Creating and Managing Tables

(SQL : Data Definition Language)

week#1



**By Kanokwan Atchariyachanvanich
Faculty of Information Technology**

KMITL

Database System Concepts

2/2565

OUTLINE ก่อนสอบกลางภาค

| Date | SQL |
|--------------------|---|
| 9, 11 JAN 2023 | <ul style="list-style-type: none">• Lab Introduction• Introduction to DBLearn (SQL tool) |
| 16, 18 JAN 2023 | LAB 1 - Creating and Managing Tables (DDL) |
| 23, 25 JAN 2023 | LAB 2 - Including Constraints (DDL) |
| 30 JAN, 1 FEB 2023 | LAB 3 - Manipulating Data (DML) |
| 6, 8 FEB 2023 | LAB 4 - SQL SELECT Statements <ul style="list-style-type: none">• Writing Basic |
| 13, 15 FEB 2023 | LAB 5 - SQL SELECT Statements <ul style="list-style-type: none">• Restricting and Sorting Data |
| 20, 22 FEB 2023 | - ทวนก่อนสอบ Quiz 1 |
| 27 FEB, 1 MAR 2023 | Quiz 1: LAB 1 – LAB 5 |

OUTLINE หลังสอบกลางภาค

| Date | SQL |
|-----------------|---|
| 8 MAR 2023 | MIDTERM examination (9.30-12.30) |
| 20, 22 MAR 2023 | LAB 6 - Displaying Data from Multiple Tables (join) |
| 27, 29 MAR 2023 | LAB 7 - Displaying Data from Multiple Tables (outer join) |
| 3, 5 APR 2023 | LAB 8 - Group function |
| 10, 12 APR 2023 | LAB 9 - Subqueries |
| 17, 19 APR 2023 | นำเสนอคำสั่ง SQL ที่ใช้สร้างรายงานและตัวอย่างรายงานจากฐานข้อมูลในโปรเจค |
| 24, 26 APR 2023 | ทวนก่อนสอบ Quiz 2 |
| 1, 3 MAY 2023 | Quiz 2 : LAB 6 – LAB 9 |
| 10 MAY 2023 | FINAL examination (9.30-12.30) |

STRUCTURED QUERY LANGUAGE (SQL) STATEMENT

| Type | SQL Statement |
|--|---|
| Data Definition Language (DDL) อธิบายส่วนของ SQL ที่อนุญาตให้สร้าง, เปลี่ยน, และทำลายอ้อมเบกเก็ตฐานข้อมูล อ้อมเบกเก็ตฐานข้อมูลเหล่านี้รวมถึงแบบแผน, ตาราง, มุมมอง, ลำดับ, แคตalog, ดรอชนี, และ alias | CREATE ALTER DROP RENAME ANALYZE AUDIT COMMENT ASSOCIATE STATISTICS DISASSOCIATE STATISTICS |
| Data Manipulation Language (DML) ภาษาสำหรับจัดการข้อมูล คือส่วนของประโยชน์ SQL ที่อนุญาตให้คุณควบคุมหรือจัดการข้อมูล | SELECT INSERT UPDATE DELETE MERGE CALL EXPLAIN PLAN LOCK TABLE |
| Transaction Control จัดการ transaction จากการเปลี่ยนแปลงที่เกิดจาก DML | COMMIT ROLLBACK SAVEPOINT SET TRANSACTION |

OBJECTIVES

After completing this lesson, you should be able to do the following:

- Learn Data Definition Language (DDL)
- Create tables
- Describe the data types that can be used when specifying column definition
- Alter table definitions
- Drop, and rename tables

DATA DEFINITION LANGUAGE (DDL)

- SQL includes commands to create database objects as showed below

| Object | Description |
|----------|--|
| Table | Basic unit of storage; composed of rows and columns |
| View | Logically represents subsets of data from one or more tables |
| Sequence | Numeric value generator |
| Index | Improves the performance of some queries |
| Synonym | Gives alternative names to objects |

Note: More database objects are available but are not covered in this course.

EXAMPLE: TABLES IN REGISTRATION SYSTEM

| STUDENT | | | |
|--------------|----------------|-------|-------|
| Student_Name | Student_Number | Class | Major |
| Wirat | 13 | 1 | IT |
| Nannapas | 18 | 2 | IT |

| COURSE | | | | |
|-------------------------------------|---------------|--------|------------|--|
| Course_Name | Course_Number | Credit | Department | |
| Database System | 6100103 | 3 | IT | |
| Discrete Mathematics | 4100103 | 3 | MATH | |
| Economics of Information Technology | 6100110 | 3 | IT | |
| Management Information System | 6100106 | 3 | IT | |

| SECTION | | | | |
|----------------|---------------|----------|------|------------|
| Section_Number | Course_Number | Semester | Year | Instructor |
| 10 | 6100103 | 1 | 07 | Srinual |
| 11 | 6100103 | 1 | 07 | Srinual |
| 12 | 6100110 | 2 | 07 | Warunee |
| 20 | 4100103 | 2 | 07 | Prapan |

| GRADE_REPORT | | |
|----------------|----------------|-------|
| Student_Number | Section_Number | Grade |
| 13 | 10 | A |
| 13 | 12 | B |
| 18 | 11 | A |
| 18 | 20 | C |

| PREREQUISITE | |
|---------------|---------------------|
| Course_Number | Prerequisite_Number |
| 6100103 | 4100103 |
| 6100110 | 6100106 |

รูปที่ 1.1 แสดงฐานข้อมูลระบบลงทะเบียน



EXAMPLE: TABLES IN REGISTRATION SYSTEM

○ Meta-data

RELATIONS

| Relation_Name | No_of_Columns |
|---------------|---------------|
| STUDENT | 4 |
| COURSE | 4 |
| SECTION | 5 |
| GRADE_REPORT | 3 |
| PREREQUISITE | 2 |

COLUMNS

| Column_Name | Data_Type | Belongs_to_relation |
|---------------------|---------------|---------------------|
| Name | Character(30) | STUDENT |
| Student_Number | Character(4) | STUDENT |
| Class | Integer(1) | STUDENT |
| Major | Character(30) | STUDENT |
| Course_Name | Character(30) | COURSE |
| | | |
| | | |
| Prerequisite_Number | Character(8) | PREREQUISITE |

○ User data

STUDENT

| Student_Name | Student_Number | Class | Major |
|--------------|----------------|-------|-------|
| Wirat | 13 | 1 | IT |
| Nannapas | 18 | 2 | IT |

THE CREATE TABLE STATEMENT SYNTAX

- Creates a new table in the user's database schema

```
CREATE TABLE TABLE_NAME (column_name datatype,  
[column_name datatype, . . . ] );
```

- You must have:
 - CREATE TABLE privilege
 - A storage area
- You specify:
 - The table name can be specified as *db_name.tbl_name*
 - Table name
 - Column name, column data type, and column size

CREATING TABLES

- Example: Create the table named “dept”

```
CREATE TABLE dept  
(  
    deptno    INT (3),  
    dname     VARCHAR(14),  
    loc       VARCHAR(13)  
);
```

- Confirm table creation.

```
DESCRIBE dept ;
```

NAMING RULES FOR TABLES

Table names and column names:

- ควรตั้งชื่อให้มีความยาว ตั้งแต่ 1- 64 ตัวอักษร
- ควรจะกำหนดแบบ alphanumeric ประกอบด้วย A-Z, a-z, 0-9, _, \$,
สามารถขึ้นต้นด้วยตัวเลข แต่ไม่ใช่ตัวเลขทั้งหมด
- ไม่ควรใช้ spaces ใน การตั้งชื่อ
- ต้องไม่ตั้งชื่อตารางซ้ำกันภายในตัวเจ้าของตารางเดียวกัน
- ชื่อตารางไม่ควรมี ‘.’ และ ‘/’ หรือ ‘\'
- If an identifier contains special characters or is a **reserved word**,
you must quote- the backtick (‘’)
- Table Alias' name can contain 1- 256 characters
- Column name is not case sensitive on any platform
- Table name is not case sensitive. (server runs on Windows)

Note: For example, EMPLOYEES is treated as the same name as
eMPloyees or eMpLOYEES.

DATA TYPES (MySQL)

Text types:

| Data type | Description |
|------------------|---|
| CHAR(size) | Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters |
| VARCHAR(size) | Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT | Holds a string with a maximum length of 65,535 characters |
| BLOB | For BLOBS (Binary Large OBjects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBS (Binary Large OBjects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBS (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data |
| ENUM(x,y,z,etc.) | Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z') |
| SET | Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice |

DATA TYPES (MySQL)

Number types:

| Data type | Description |
|-----------------|---|
| TINYINT(size) | -128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis |
| SMALLINT(size) | -32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis |
| MEDIUMINT(size) | -8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis |
| INT(size) | -2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis |
| BIGINT(size) | -9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis |
| FLOAT(size,d) | A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |
| DOUBLE(size,d) | A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |
| DECIMAL(size,d) | A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |

DATA TYPES (MySQL)

Date types:

| Data type | Description |
|-------------|---|
| DATE() | A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME() | *A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59' |
| TIMESTAMP() | *A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC |
| TIME() | A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59' |
| YEAR() | A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069 |

EXERCISE 1: จะสร้างตารางชื่อ EMP ซึ่งมีโครงสร้างดังต่อไปนี้

| Column name | ID | LAST_NAME | FIRST_NAME | DEPT_ID |
|-------------|-----|-----------|------------|---------|
| Data type | INT | VARCHAR | VARCHAR | INT |
| Length | 7 | 25 | 25 | 7 |

```
CREATE TABLE EMP  
(  
    id          INT (7),  
    last_name   VARCHAR(25),  
    first_name  VARCHAR(25),  
    dept_id    INT(7)  
);
```

THE ALTER TABLE STATEMENT

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

THE ALTER TABLE STATEMENT SYNTAX

- Use the ALTER TABLE statement to add, modify, or drop columns.

```
ALTER TABLE table_name  
ADD column_name datatype [DEFAULT expr]  
[, column_name datatype]...;
```

```
ALTER TABLE table_name  
MODIFY column_name datatype [DEFAULT expr]  
[, column_name datatype]...;
```

```
ALTER TABLE table_name  
DROP column_name ;
```

ADDING A COLUMN

DEPT80

| EMPLOYEE_ID | LAST_NAME | ANNSAL | HIRE_DATE |
|-------------|-----------|--------|-----------|
| 145 | Russell | 168000 | 01-OCT-96 |
| 146 | Partners | 162000 | 05-JAN-97 |
| 147 | Errazuriz | 144000 | 10-MAR-97 |
| 148 | Cambrault | 132000 | 15-OCT-99 |
| 149 | Zlotkey | 126000 | 29-JAN-00 |
| 150 | Tucker | 120000 | 30-JAN-97 |

New Column

| JOB_ID |
|--------|
| |
| |
| |
| |

“Add a new column to the DEPT80 table.”

DEPT80

| EMPLOYEE_ID | LAST_NAME | ANNSAL | HIRE_DATE | JOB_ID |
|-------------|-----------|--------|-----------|--------|
| 145 | Russell | 168000 | 01-OCT-96 | |
| 146 | Partners | 162000 | 05-JAN-97 | |
| 147 | Errazuriz | 144000 | 10-MAR-97 | |
| 148 | Cambrault | 132000 | 15-OCT-99 | |
| 149 | Zlotkey | 126000 | 29-JAN-00 | |
| 150 | Tucker | 120000 | 30-JAN-97 | |

ADDING A COLUMN

- You use the ADD clause to add columns.

```
ALTER TABLE dept80  
ADD job_id VARCHAR (9) ;
```

- The new column becomes the last column.

| EMPLOYEE_ID | LAST_NAME | ANNSAL | HIRE_DATE | JOB_ID |
|-------------|-----------|--------|-----------|--------|
| 145 | Russell | 168000 | 01-OCT-96 | |
| 146 | Partners | 162000 | 05-JAN-97 | |
| 147 | Errazuriz | 144000 | 10-MAR-97 | |
| 148 | Cambrault | 132000 | 15-OCT-99 | |
| 149 | Zlotkey | 126000 | 29-JAN-00 | |
| 150 | Tucker | 120000 | 30-JAN-97 | |

EXERCISE 2: เพิ่มคอลัมน์ emp_id ในตาราง job_it และแสดงคำสั่งเพื่อดูโครงสร้างหลังจากการลับคอลัมน์

| Column | Data type & size |
|--------|------------------|
| emp_id | int(11) |

```
ALTER TABLE job_it  
ADD emp_id int(11) ;
```

คำสั่งเพื่อดูโครงสร้างหลังจากการแก้ไข

```
DESCRIBE job_it;
```

MODIFYING A COLUMN

- You can change a **column's size**, and default values (old varchar(25) => new **VARCHAR (30)**)

```
ALTER TABLE dept80  
MODIFY last_name VARCHAR (30) ;
```

Modify
column's size

- A change to the default value affects only subsequent insertions to the table.

MODIFYING A COLUMN

- You can change a column's data type & size
- Old varchar(25) => New CHAR (30)

```
ALTER TABLE dept80  
MODIFY last_name CHAR (30) ;
```

Modify data type

EXERCISE 3: แก้ไขคอลัมน์ job_title ในตาราง job_it ให้มีขนาดเป็น 50 และแสดงคำสั่งเพื่อตู้โครงสร้างหลังจากการแก้ไข

```
ALTER TABLE job_it  
MODIFY job_title VARCHAR(50) ;
```

คำสั่งเพื่อตู้โครงสร้างหลังจากการแก้ไข

```
DESCRIBE job_it;
```

ALTER TABLE แก้ไขชื่อคอลัมน์ ชนิดข้อมูลและขนาดข้อมูล

```
ALTER TABLE ชื่อตาราง  
CHANGE      ชื่อคอลัมน์เก่า  ชื่อคอลัมน์ใหม่ ชนิดข้อมูล(ขนาดข้อมูล);
```

```
ALTER TABLE sales_reps  
CHANGE      reps_id          id      varchar(30);
```

ALTER TABLE แก้ไขชื่อคอลัมน์

หมายเหตุ : ใช้กับ MySQL v.8 ขึ้นไป ดังนั้นใช้กับ DBLearn ไม่ได้ เนื่องจาก
ระบบ DBLearn ใช้ MySQL v.5.7.4

```
ALTER TABLE ชื่อตาราง  
RENAME COLUMN ชื่อคอลัมน์เก่า TO ชื่อคอลัมน์ใหม่ ;
```

```
ALTER TABLE sales_reps  
RENAME COLUMN name TO firstname ;
```

DROPPING A COLUMN

- Use the DROP COLUMN clause to drop columns you no longer need from the table.

```
ALTER TABLE dept80  
DROP job_id ;
```

- **Guidelines**

- คอลัมน์นั้นอาจจะมีหรือไม่มีข้อมูลก็ได้
- ตารางต้องมีอย่างน้อย 1 คอลัมน์เหลืออยู่หลังจากใช้ ALTER TABLE statement
- เมื่อคอลัมน์ถูกลบแล้ว จะไม่สามารถกู้คืนคอลัมน์นั้นได้อีก

EXERCISE 4: ลบคอลัมน์ MAX_SALARY จากตาราง job_it และแสดงคำสั่งเพื่อตู้โครงสร้างหลังจากการลบคอลัมน์

```
ALTER TABLE job_it
```

```
DROP max_salary ;
```

คำสั่งเพื่อตู้โครงสร้างหลังจากการแก้ไข

```
DESCRIBE job_it;
```

ADD, MODIFY, DROP มากกว่า 1 คอลัมน์

- ตัวอย่าง

```
ALTER TABLE sales_reps  
ADD address varchar(30),  
ADD tel varchar(15);
```

```
ALTER TABLE sales_reps  
MODIFY name varchar(30),  
MODIFY salary int(6);
```

```
ALTER TABLE sales_reps  
DROP salary,  
DROP commission_pct;
```

CHANGING THE NAME OF AN OBJECT

- To change the name of a table, view, sequence, or synonym, you execute the RENAME statement.
- You must be the owner of the object.

```
RENAME TABLE old_table_name TO new_table_name ;  
ALTER TABLE      old_table_name RENAME new_table_name;
```

```
RENAME TABLE      dept80      TO      detail_dept ;  
ALTER TABLE      dept80      RENAME detail_dept;
```

หมายเหตุ RENAME TABLE สามารถแก้ไขชื่อตารางตั้งแต่ 2 ตารางขึ้นไปพร้อมกันได้

DROPPING A TABLE

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- You cannot roll back the DROP TABLE statement.

```
DROP TABLE      table_name ;
```

```
DROP TABLE      detail_dept ;
```

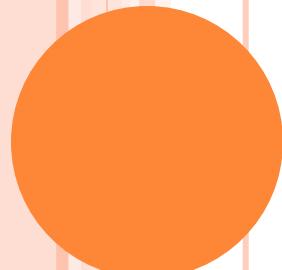
SUMMARY

- In this lesson, you should have learned how to use DDL statements to create, alter, drop, and rename tables.

| Statement | Description |
|--------------|--------------------------------------|
| CREATE TABLE | Creates a table |
| ALTER TABLE | Modifies table structures |
| DROP TABLE | Removes the rows and table structure |
| RENAME TABLE | Changes the name of a table |

LAB 2

DATA DEFINITION LANGUAGE (DDL) #2



By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts

2/2565



OUTLINE ก่อนสอบกลางภาค

| Date | SQL |
|--------------------|---|
| 9, 11 JAN 2023 | <ul style="list-style-type: none">• Lab Introduction• Introduction to DBLearn (SQL tool) |
| 16, 18 JAN 2023 | LAB 1 - Creating and Managing Tables (DDL) |
| 23, 25 JAN 2023 | LAB 2 - Including Constraints (DDL) |
| 30 JAN, 1 FEB 2023 | LAB 3 - Manipulating Data (DML) |
| 6, 8 FEB 2023 | LAB 4 - SQL SELECT Statements <ul style="list-style-type: none">• Writing Basic |
| 13, 15 FEB 2023 | LAB 5 - SQL SELECT Statements <ul style="list-style-type: none">• Restricting and Sorting Data |
| 20, 22 FEB 2023 | - ทวนก่อนสอบ Quiz 1 |
| 27 FEB, 1 MAR 2023 | Quiz 1: LAB 1 – LAB 5 |

OBJECTIVES

After completing this lesson, you should be able to do the following:

- **Describe constraints**
- **Create and maintain constraints**

WHAT ARE CONSTRAINTS?

- **Constraints enforce rules at the table level.**
- **Constraints prevent the deletion of a table if there are dependencies.**
- **The following constraint types are valid:**
 - **NOT NULL**
 - **UNIQUE**
 - **PRIMARY KEY**
 - **FOREIGN KEY**

DATA INTEGRITY CONSTRAINTS

| Constraint | Description |
|-------------|---|
| NOT NULL | กำหนดต้องมีค่าในคอลัมน์ที่กำหนด คือไม่สามารถมีค่า NULL ในคอลัมน์นี้ได้ |
| UNIQUE | กำหนด 1 คอลัมน์หรือรวมกันมากกว่า 1 คอลัมน์โดยค่าในคอลัมน์เหล่านี้ต้องแตกต่างกันจากทุกແລวในตารางเดียวกัน |
| PRIMARY KEY | ระบุว่าแต่ละແລวของตารางมีความแตกต่างกัน |
| FOREIGN KEY | กำหนดและบังคับใช้ความสัมพันธ์ foreign key relationship ระหว่างคอลัมน์ในตารางใดๆ และคอลัมน์ของตารางที่ถูกอ้างอิง |

CONSTRAINT GUIDELINES

- สามารถกำหนด Constraint ด้วย 2 วิธีคือ
 1. ในขณะที่สร้างตาราง
 2. หลังจากสร้างตารางแล้ว
- สามารถนิยาม constraint ได้ที่ระดับคอลัมน์หรือระดับตาราง
- สามารถดูการกำหนด constraint ได้จาก

```
SELECT    TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME  
FROM      INFORMATION_SCHEMA.KEY_COLUMN_USAGE ;
```

ชนิด CONSTRAINTS

1. Column Constraint level ระดับคอลัมน์ :

- References a single column and is defined within a specification for the owning column;
- can define any type of integrity constraint.

2. Table Constraint level ระดับตาราง :

- References one or more column and is defined separated from the definitions of the column in the table;
- can define any constraints except NOT NULL.

DEFINING CONSTRAINTS

```
CREATE TABLE table_name (  
    column datatype [column_constraint] ,  
    column datatype [column_constraint] ,  
    ... ,  
    [table_constraint] [, . . .]  
);
```

- **Table_name** is the name of the table
- **Column** is the name of the column
- **Column_constraint** is an integrity constraint as part of the column definition
- **Table_constraint** is an integrity constraint as part of the table definition

1. COLUMN CONSTRAINT

- can define any type of integrity constraint.

Column datatype **constraint_type**,

```
CREATE TABLE employ (
    employee_id INT(6),
    first_name   VARCHAR(20),
    ...
    hire_date    DATE NOT NULL,
    PRIMARY KEY (employee_id)
);
```

Column
constraint

2. TABLE CONSTRAINT

- can define any constraints except NOT NULL.

Last Column, . . .

`constraint_type `constraint_name` (column),`

```
CREATE TABLE employ (
    employee_id INT(6),
    first_name   VARCHAR(20),
    ...
    hire_date    DATE      NOT NULL,
    PRIMARY KEY `employ_employee_id` (employee_id )
);
```

Table
constraint

Column_name ที่เป็น
primary key

THE NOT NULL CONSTRAINT

- Ensures that null values are not permitted for the column:

| EMPLOYEE_ID | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | DEPARTMENT_ID |
|-------------|-----------|----------|--------------------|-----------|---------|--------|---------------|
| 100 | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | 90 |
| 101 | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP | 17000 | 90 |
| 102 | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-93 | AD_VP | 17000 | 90 |
| 103 | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-90 | IT_PROG | 9000 | 60 |
| 104 | Ernst | BERNST | 590.423.4568 | 21-MAY-91 | IT_PROG | 6000 | 60 |
| 178 | Grant | KGRANT | 011.44.1644.429263 | 24-MAY-99 | SA_REP | 7000 | |
| 200 | Whalen | JWHALEN | 515.123.4444 | 17-SEP-87 | AD_ASST | 4400 | 10 |



NOT NULL constraint
(ไม่มีແລວໃຫ້ທີ່ມີຄ່າເປັນ Null
ສໍາຮັບຄອລິມນ໌ last_name)

NOT NULL constraint
(ໄມ່ມີແລວໃຫ້ທີ່ມີຄ່າເປັນ Null ສໍາຮັບ
ຄອລິມນ໌ hire_date)

**ABSENCE OF NOT
NULL constraint**
(ມີບາງແລວທີ່ມີຄ່າເປັນ Null ສໍາຮັບ
ຄອລິມນ໌ department_id)

THE NOT NULL CONSTRAINT

- Is defined only at the column level:

```
CREATE TABLE employees(  
    employee_id      INT,  
    last_name        VARCHAR(25) NOT NULL ,  
    salary           INT ,  
    commission_pct   decimal (4, 2) ,  
    hire_date        DATE NOT NULL  
)
```

THE **UNIQUE** CONSTRAINT

- A UNIQUE key integrity constraint requires that every value in a column or set of column (key) be unique— that is, no two rows of a table can have duplicate values in a specified column or set of column.
- UNIQUE constraints allow the input of nulls unless you also define NOT NULL constraints for the same columns.
- UNIQUE constraints can be defined at the column or table level.
- A composite unique key is created by using the table level definition.

THE UNIQUE CONSTRAINT

UNIQUE
constraint

| EMPLOYEE_ID | LAST_NAME | EMAIL |
|-------------|-----------|----------|
| 100 | King | SKING |
| 101 | Kochhar | NKOCHHAR |
| 102 | De Haan | LDEHAAN |
| 103 | Hunold | AHUNOLD |
| 104 | Ernst | BERNST |

| EMPLOYEE_ID | LAST_NAME | EMAIL |
|-------------|-----------|--------|
| 208 | Smith | JSMITH |
| 209 | Smith | JSMITH |

Allowed
Not allowed;
Already exists

EXAMPLE: THE **UNIQUE** CONSTRAINT

- Defined at either the table level or the column level:

```
CREATE TABLE employ_uu(  
    employee_id      INT,  
    last_name        VARCHAR(25) NOT NULL ,  
    email            VARCHAR(25) ,  
    salary           INT ,  
    commission_pct   decimal (4,2) ,  
    hire_date        DATE NOT NULL,  
    UNIQUE employ_uu_email_uk (email) );
```

THE PRIMARY KEY CONSTRAINT

- แต่ละตารางสามารถมีได้เพียง 1 primary key
- Primary key constraint คือคอลัมน์หรือกลุ่มของคอลัมน์ที่ระบุว่าแต่ละแถวในตารางจะไม่ซ้ำกัน
- Primary key constraint บังคับใช้เอกสารลักษณ์ของคอลัมน์หรือการรวมกันของคอลัมน์
- Primary key constraint ทำให้แน่ใจว่า ไม่มี คอลัมน์ที่เป็นส่วนหนึ่งของคีย์หลักสามารถมีค่า Null ได้

THE PRIMARY KEY CONSTRAINT

DEPARTMENT

PRIMARY KEY

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |
| 60 | IT | 103 | 1400 |
| 80 | Sales | 149 | 2500 |

Not allowed
(NULL value)

INSERT INTO

| | | | |
|----|-------------------|-----|------|
| | Public Accounting | | 1400 |
| 50 | Finance | 124 | 1500 |

Not allowed
(50 already exists)

EXAMPLE: THE PRIMARY KEY CONSTRAINT

- Defined at either the table level or the column level:

```
CREATE TABLE depart(  
    department_id INT(4) PRIMARY KEY,  
    department_name VARCHAR(30) NOT NULL,  
    manager_id      INT(6),  
    location_id     INT(4)  
);
```

Column-level
constraint

NOTE: A table can have only one PRIMARY KEY constraint but can have several UNIQUE constraints.

EXAMPLE: THE PRIMARY KEY CONSTRAINT

- Defined at either the table level or the column level:

```
CREATE TABLE deps(  
    department_id      INT(4),  
    department_name    VARCHAR(30) NOT NULL,  
    manager_id         INT(6),  
    location_id        INT(4),  
    PRIMARY KEY `dept_id_pk` (department_id) );
```

Table-level
constraint

NOTE: A table can have only one PRIMARY KEY constraint but can have several UNIQUE constraints.

THE FOREIGN KEY CONSTRAINT (REFERENTIAL INTEGRITY CONSTRAINT)

DEPARTMENT (parent table)

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 50 | Shipping | 124 | 1500 |
| 60 | IT | 103 | 1400 |
| 80 | Sales | 149 | 2600 |

PRIMARY
KEY

EMPLOYEES (child table)

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100 | King | 90 |
| 101 | Kochhar | 90 |
| 102 | De Haan | 90 |
| 103 | Hunold | 60 |
| 104 | Ernst | 60 |
| 107 | Lorentz | 60 |

FOREIGN KEY

| | | |
|-----|------|----|
| 200 | Ford | 9 |
| 201 | Ford | 60 |

INSERT INTO

Not allowed
(9 does not exist)
Allowed

FOREIGN KEY CONSTRAINT KEYWORDS

- **FOREIGN KEY:** นิยามคอลัมน์ในตารางลูกที่ระดับ table constraint
- **REFERENCES:** กำหนดตารางและคอลัมน์ในตารางแม่
- **ON DELETE CASCADE:** ลบ dependent rows ในตารางลูก เมื่อถาวรในตารางแม่ทุกฉบับ
- **ON DELETE SET NULL:** แปลงค่าที่อยู่ใน foreign key เป็นค่า null
- ถ้าไม่มีคีเวิร์ด ON DELETE CASCADE หรือ ON DELETE SET NULL จะไม่สามารถลบถาวรใน parent table ได้ถ้ามันถูกอ้างอิงในตารางลูก

EXAMPLE: THE FOREIGN KEY CONSTRAINT

- Defined at the table level :

```
CREATE TABLE Persons  
(  
    PersonID int Primary Key,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address  varchar(255),  
    City     varchar(255)  
);
```

```
CREATE TABLE Orders  
(  
    OrderID      int NOT NULL,  
    OrderNumber  int NOT NULL,  
    PersonID     int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES  
    Persons(PersonID)  
    ON DELETE CASCADE  
);
```

คำสั่งเรียกดู CONSTRAINTS

```
SELECT constraint_name,table_name,constraint_type  
FROM information_schema.table_constraints  
WHERE table_name='ชื่อตารางที่ต้องการแสดง constraint'
```

Example:

```
SELECT constraint_name, table_name, constraint_type  
FROM information_schema.table_constraints  
WHERE table_name='countries'
```

SUMMARY

- In this lesson, you should have learned how to create constraints when creating table.
- Types of constraints:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
- You can query the **INFORMATION_SCHEMA.KEY_COLUMN_USAGE** table to view all constraint definitions and names.

EXERCISE 1: จงสร้างตารางชื่อ MEMBER ซึ่งมีโครงสร้างดังต่อไปนี้ (COLUMN-LEVEL CONSTRAINTS)

| Column name | MEMBER_ID | LAST_NAME | FIRST_NAME | ADDRESS | CITY |
|-----------------|-------------|-----------|------------|---------|---------|
| Key Type | Primary key | | | | |
| NULL/ UNIQUE | NN, U | NN | | | |
| Data type | INT | VARCHAR | VARCHAR | VARCHAR | VARCHAR |
| Length | 10 | 25 | 25 | 100 | 30 |

EXERCISE 2: จงสร้างตารางชื่อ MEMBER2 ซึ่งมีโครงสร้างดังต่อไปนี้ (TABLE-LEVEL CONSTRAINT)

| Column name | MEMBER_ID | LAST_NAME | FIRST_NAME | ADDRESS | CITY |
|--------------|-------------|-----------|------------|---------|---------|
| Key Type | Primary key | | | | |
| NULL/ UNIQUE | NN, U | NN | | | |
| Data type | INT | VARCHAR | VARCHAR | VARCHAR | VARCHAR |
| Length | 10 | 25 | 25 | 100 | 30 |

CONSTRAINT GUIDELINES

- สามารถกำหนด Constraint ด้วย 2 วิธีคือ

1. ในขณะที่สร้างตาราง
2. หลังจากสร้างตารางแล้ว

- สามารถดูการกำหนด constraint ได้จาก

```
SELECT      TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME  
FROM        INFORMATION_SCHEMA.KEY_COLUMN_USAGE ;
```

CONSTRAINTS FOR EXISTING TABLES

Use the **ALTER TABLE** statement to:

- Add or drop a constraint, but not modify its structure
- Add a **NOT NULL** constraint by using the **MODIFY** clause

ADDING A CONSTRAINT SYNTAX

- คำสั่ง เพิ่ม Constraint
- Note: สามารถกำหนด NOT NULL ให้กับคอลัมน์ได้ ก็ต่อเมื่อคอลัมน์นั้นว่างไม่มีข้อมูล หรือ ทุกแถวในคอลัมน์นั้นมีค่า

```
ALTER TABLE table_name  
ADD [CONSTRAINT constraint_name]  
constraint_type (column) ;
```

Exercise 3 : Adding Primary Key Constraint

- จงเพิ่ม PRIMARY KEY constraint ให้กับตาราง lab_emp

Exercise 4 : Adding Primary Key Constraint

- จงเพิ่ม PRIMARY KEY constraint ให้กับตาราง lab_location

ADDING A FOREIGN KEY CONSTRAINT

- ตัวอย่าง : เพิ่ม FOREIGN KEY constraint ชื่อ constraint_name ให้กับตาราง table_name ซึ่งมี Parent_name เป็น primary key ใน parent_table
- หมายเหตุ ต้องกำหนด parent_column เป็น primary ก่อนถึงจะสามารถเพิ่ม foreign key เชื่อมไปหาได้

```
ALTER TABLE      table_name  
ADD CONSTRAINT   constraint_name  
    FOREIGN KEY  (column_name)  
    REFERENCES   parent_table(parent_column);
```

Exercise 5 : Adding Foreign Key Constraint

- จงเพิ่ม FOREIGN KEY constraint ชื่อ lab_emp_fk ให้กับตาราง lab_emp โดยมี Parent table คือ lab_location

ADDING A UNIQUE CONSTRAINT

- ตัวอย่าง : เพิ่ม UNIQUE constraint ชื่อ constraint_name ให้กับ column_name ใน ตาราง table_name

```
ALTER TABLE      table_name  
ADD     CONSTRAINT constraint_name  
        UNIQUE    (column_name) ;
```

Exercise 6 : Adding UNIQUE Constraint

- จงเพิ่ม UNIQUE constraint ชื่อ loc_name_un ให้กับ location_name ในตาราง lab_location

ADDING A NOT NULL CONSTRAINT

- คำสั่งเพิ่ม NOT NULL Constraint ให้กับคอลัมน์ได้ ก็ต่อเมื่อคอลัมน์นั้นมีค่า

1. ตรวจสอบค่าปัจจุบันที่อยู่ในคอลัมน์ โดยใช้คำสั่ง

```
SELECT * FROM table_name WHERE column_name IS NULL ;
```

2. อัพเดตค่าที่เป็น NULL ให้เป็นค่าที่ไม่ใช่ NOT NULL

```
UPDATE table_name SET column_name = "new_value"
```

```
WHERE column_name IS NULL ;
```

3. เพิ่ม NOT NULL constraint

```
ALTER TABLE table_name  
MODIFY column_name datatype NOT NULL ;
```

Example 7: A NOT NULL CONSTRAINT

- จงเพิ่ม NOT NULL constraint ให้กับ `last_name` และ `first_name` ในตาราง `lab_emp`

DROPPING A CONSTRAINT SYNTAX

○ คำสั่งลบ Constraint

```
ALTER TABLE      table_name  
DROP PRIMARY KEY ;
```

```
ALTER TABLE      table_name  
DROP FOREIGN KEY constraint_name ;
```

DROPPING A PK/FK CONSTRAINT

- ตัวอย่าง : ลบ PRIMARY KEY constraint จากตาราง lab_emp และลบ FOREIGN KEY constraint ที่เกี่ยวข้อง (location_id) ในตาราง

```
ALTER TABLE      lab_emp  
DROP PRIMARY KEY ;
```

- ตัวอย่าง : ลบ FOREIGN KEY Constraint จากตาราง lab_emp

```
ALTER TABLE      lab_emp  
DROP FOREIGN KEY lab_emp_fk;
```

ตรวจสอบชื่อ constraint ด้วยคำสั่ง

```
SELECT      TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME  
FROM        INFORMATION_SCHEMA.KEY_COLUMN_USAGE ;
```

Exercise 8 : Drop Primary Key constraint

- จงลบ primary key constraint ของตาราง regions



Exercise 9 : Drop Foreign Key constraint

- จงลบ foreign key constraint ของตาราง locations

DROPPING A UNIQUE CONSTRAINT

- ตัวอย่าง : ลบ UNIQUE constraint จากตาราง lab_location จะลบด้วย ชื่อ Constraint ที่ตั้งไว้

```
ALTER TABLE      table_name  
DROP INDEX      constraint_name;
```

```
ALTER TABLE      lab_location  
DROP INDEX      loc_name_un;
```

ไม่มีคำสั่ง **DROP NOT NULL CONSTRAINT**

- แต่จะใช้ คำสั่งแก้ไข เป็น NULL

```
ALTER TABLE      table_name  
MODIFY          column_name datatype NULL;
```

- จงแก้ไข constraint ของ first_name ในตาราง lab_emp จาก NOT NULL เป็น NULL

```
ALTER TABLE      lab_emp  
MODIFY          first_name varchar(25) NULL;
```

VIEWING CONSTRAINTS

- Query the `key_column_usage` table to view all constraint definitions and names.

```
Select  table_name, column_name, constraint_name,  
        referenced_column_name, referenced_table_name  
From    information_schema.key_column_usage  
Where   table_name = 'lab_emp'
```

VIEWING THE COLUMNS ASSOCIATED WITH CONSTRAINTS

- View the columns associated with the constraint names in the information_schema.key_column_usage.

```
SELECT constraint_name, column_name  
FROM information_schema.key_column_usage  
WHERE table_name = 'lab_location' ;
```

DROP TABLE with CONSTRAINT ลบตารางที่กำหนด FK CONSTRAINT ไว้

- ไม่ลบ foreign key constraint

```
SET FOREIGN_KEY_CHECKS = 0;
```

```
DROP TABLE table_name;
```

```
SET FOREIGN_KEY_CHECKS = 1;
```

- ลบ foreign key constraint (ต้องลบทั้งหมดที่เชื่อมโยงไปยังตารางอื่น)

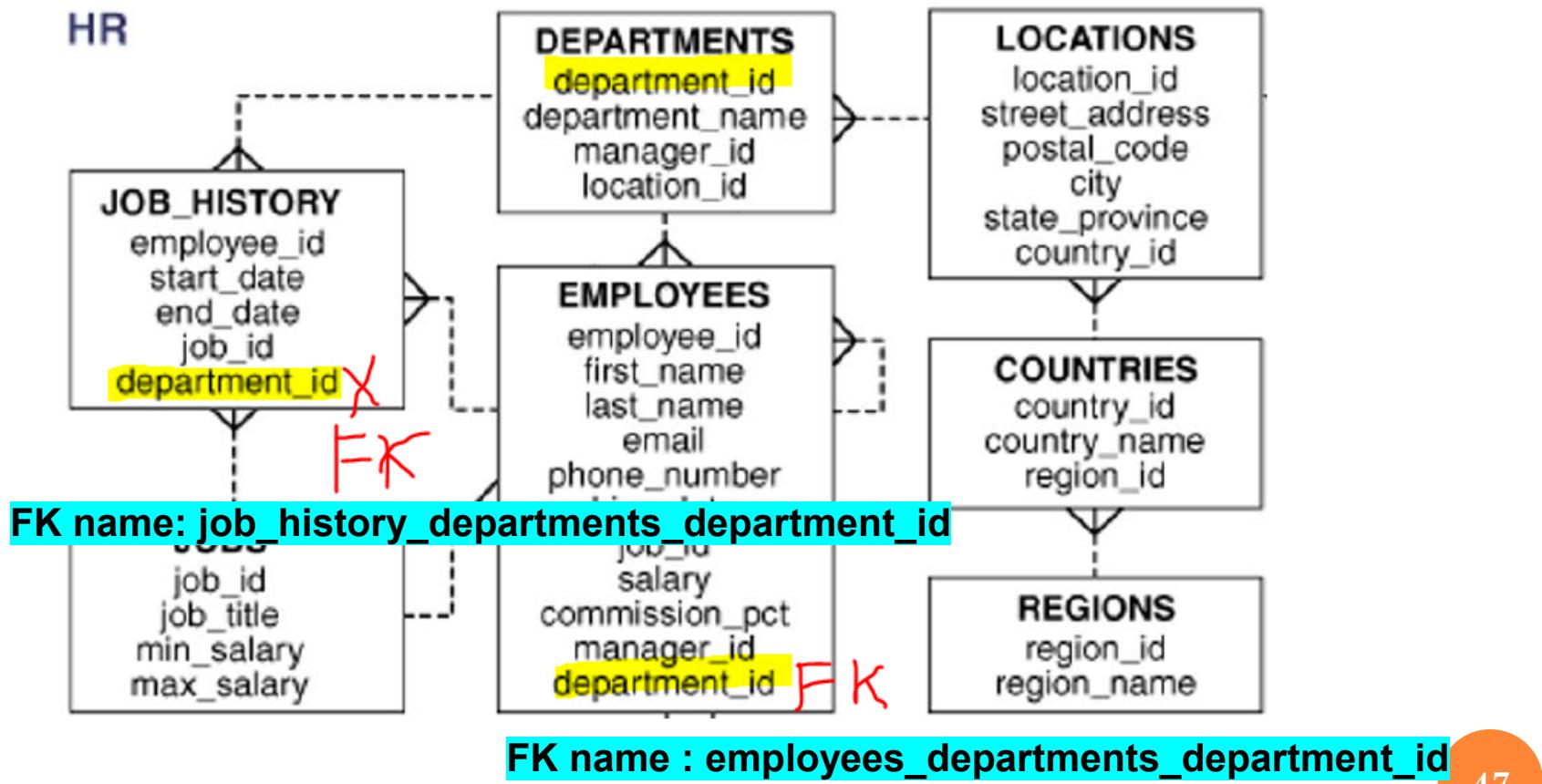
```
ALTER TABLE table_name
```

```
DROP FOREIGN KEY constraint_name;
```

```
DROP TABLE table_name;
```

DROP TABLE with CONSTRAINT

- ลบตาราง departments ด้วยวิธีการที่ต้องลบ foreign key constraint



Exercise 10 : Drop Table with constraint

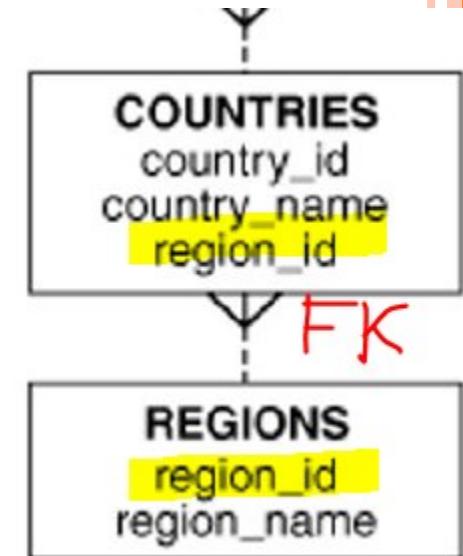
- จงลบตาราง departments ด้วยการลบ foreign key constraint

Exercise 11 : Drop Table with constraint

- จงลบตาราง regions ด้วยการ ไม่ลบ foreign key constraint

Exercise 12 : Drop Table with constraint

- จงลบตาราง regions ด้วยการลบ foreign key constraint



DROPPING A COLUMN with Constraint

1. ตรวจสอบ constraint name ด้วยคำสั่ง

```
SELECT    TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME  
FROM      INFORMATION_SCHEMA.KEY_COLUMN_USAGE ;
```

2. ลบ constraint นั้นออกจากตารางที่มีคอลัมน์ที่ต้องการลบ เช่น คอลัมน์นั้นเป็น foreign key จำเป็นต้องลบ constraint ของคอลัมน์นี้ ด้วยคำสั่ง

```
ALTER TABLE    table_name  
DROP foreign key    constraint_name ;
```

3. ลบคอลัมน์นั้นออกจากตาราง ด้วยคำสั่ง

```
ALTER TABLE    table_name  
DROP column_name ;
```

VIEWING CONSTRAINTS

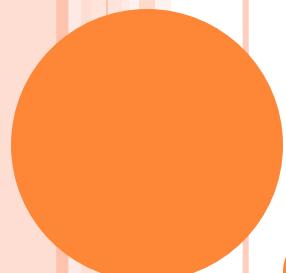
```
SELECT    TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME  
FROM      INFORMATION_SCHEMA.KEY_COLUMN_USAGE ;
```

| TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME |
|-------------|---------------|---------------------------------------|
| countries | country_id | PRIMARY |
| countries | region_id | countries_regions_region_id |
| departments | department_id | PRIMARY |
| departments | manager_id | departments_ibfk_1 |
| departments | location_id | departments_locations_location_id |
| employees | employee_id | PRIMARY |
| employees | department_id | employees_departments_department_id |
| employees | manager_id | employees_employees_employee_id |
| employees | job_id | employees_jobs_job_id |
| job_history | employee_id | employee_id |
| job_history | start_date | employee_id |
| job_history | department_id | job_history_departments_department_id |
| job_history | employee_id | job_history_employees_employee_id |
| job_history | job_id | job_history_jobs_job_id |
| jobs | job_id | PRIMARY |
| locations | location_id | PRIMARY |
| locations | country_id | locations_countries_country_id |
| regions | region_id | PRIMARY |

EXERCISE 13 : DROPPING A COLUMN with Constraint

จงลบคอลัมน์ job_id ในตาราง employees และแสดงคำสั่งเพื่อตู
โครงสร้างหลังจากการลบคอลัมน์

SQL: DATA MANIPULATION LANGUAGE (DML)



By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts



OUTLINE ก่อนสอบกลางภาค

| Date | SQL |
|--------------------|---|
| 9, 11 JAN 2023 | <ul style="list-style-type: none">• Lab Introduction• Introduction to DBLearn (SQL tool) |
| 16, 18 JAN 2023 | LAB 1 - Creating and Managing Tables (DDL) |
| 23, 25 JAN 2023 | LAB 2 - Including Constraints (DDL) |
| 30 JAN, 1 FEB 2023 | LAB 3 - Manipulating Data (DML) |
| 6, 8 FEB 2023 | LAB 4 - SQL SELECT Statements <ul style="list-style-type: none">• Writing Basic |
| 13, 15 FEB 2023 | LAB 5 - SQL SELECT Statements <ul style="list-style-type: none">• Restricting and Sorting Data |
| 20, 22 FEB 2023 | - ทวนก่อนสอบ Quiz 1 |
| 27 FEB, 1 MAR 2023 | Quiz 1: LAB 1 – LAB 5 |

SQL STATEMENT

| Type | SQL Statement |
|--|--|
| Data Manipulation Language (DML) ภาษาสำหรับจัดการข้อมูล คือส่วนของภาษา SQL ที่อนุญาตให้คุณ ควบคุมหรือจัดการข้อมูล | SELECT INSERT UPDATE DELETE MERGE CALL EXPLAIN PLAN LOCK TABLE |
| Data Definition Language (DDL) อธิบายส่วนของ SQL ที่อนุญาตให้สร้าง, เปลี่ยน, และทำลายอ็อบเจกต์ ฐานข้อมูล อ็อบเจกต์ฐานข้อมูลเหล่านี้รวมถึงแบบแผน, ตาราง, มุมมอง, ลำดับ, แคตาล็อก, ตัช尼, และ alias | CREATE ALTER DROP RENAME ANALYZE AUDIT COMMENT ASSOCIATE STATISTICS DISASSOCIATE STATISTICS |
| Transaction Control จัดการ transaction จากการเปลี่ยนแปลงที่เกิดจาก DML | COMMIT ROLLBACK SAVEPOINT SET TRANSACTION |

OBJECTIVE

- After completing this lesson, you should be able to do the following:
 - Describe each data manipulation language (DML) statement
 - Insert rows into a table
 - Update rows in a table
 - Delete rows from a table

DATA MANIPULATION LANGUAGE

- A DML statement is executed when you:
 - Add new rows to a table
 - Modify existing rows in a table
 - Remove existing rows from a table
- A transaction consists of a collection of DML statements that form a logical unit of work.

ADDING A NEW ROW TO A TABLE

DEPARTMENTS

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|-----------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 30 | Purchasing | 114 | 1700 |
| 40 | Human Resources | 203 | 2400 |
| 50 | Shipping | 121 | 1500 |
| 60 | IT | 103 | 1400 |

| New row | 280 | IT Planning | 101 | 1700 |
|---------|-----|-------------|-----|------|
| | | | | |

Insert new row
into the
DEPARTMENTS table

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|----------------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 30 | Purchasing | 114 | 1700 |
| 40 | Human Resources | 203 | 2400 |
| 50 | Shipping | 121 | 1500 |
| 60 | IT | 103 | 1400 |
| 70 | Public Relations | 204 | 2700 |
| 80 | Sales | 145 | 2500 |
| 90 | Executive | 100 | 1700 |
| 100 | Finance | 108 | 1700 |
| 110 | Accounting | 205 | 1700 |
| 120 | Treasury | | 1700 |
| 130 | Corporate Tax | | 1700 |
| 140 | Control And Credit | | 1700 |
| 150 | Shareholder Services | | 1700 |
| 160 | Benefits | | 1700 |
| 170 | Manufacturing | | 1700 |
| 180 | Construction | | 1700 |
| 190 | Contracting | | 1700 |
| 200 | Operations | | 1700 |
| 210 | IT Support | | 1700 |
| 220 | NOC | | 1700 |
| 230 | IT Helpdesk | | 1700 |
| 240 | Government Sales | | 1700 |
| 250 | Retail Sales | | 1700 |
| 260 | Recruiting | | 1700 |
| 270 | Payroll | | 1700 |
| 280 | IT Planning | 101 | 1700 |

INSERT STATEMENT SYNTAX

- Add new rows to a table by using the INSERT statement:

```
INSERT INTO table_name (column1, column2, .... ) ]  
VALUES          (value1, value2, .....);
```

- *table* is the name of the table
- *column* is the name of the column in the table to populate
- *value* is the corresponding value for the column

1. INSERTING A NEW ROW (ไม่ระบุคอลัมน์)

- ไม่แสดงคอลัมน์ในบรรทัด INSERT clause
- ใส่ค่า ทุกตัว ให้ตรงตามลำดับคอลัมน์ที่กำหนดในตาราง
- คอลัมน์ที่อนุญาตให้ใส่ Null value ได้ ไม่จำเป็นต้องระบุค่าในແກ່
- ล้อมตัวอักษรและวันที่ด้วย single quotation marks.

```
INSERT INTO regions  
VALUES      (5, 'South Africa') ;
```

| Field | Type | Null | Key |
|-------------|------------------|------|-----|
| region_id | int(11) unsigned | NO | PRI |
| region_name | varchar(25) | YES | |

1. INSERTING MULTIPLE ROWS (ไม่ระบุคอลัมน์)

- ไม่แสดงคอลัมน์ในบรรทัด INSERT clause
- ใส่ค่า ทุกตัว ให้ตรงตามลำดับคอลัมน์ที่กำหนดในตาราง
- คอลัมน์ที่อนุญาตให้ใส่ Null value ได้ ไม่จำเป็นต้องระบุค่าในແລວ
- ล้อมตัวอักษรและวันที่ด้วย single quotation marks.

```
INSERT INTO regions  
VALUES      (5, 'South Africa'),  
             (6, 'Australia and Oceania') ;
```

2. INSERTING A NEW ROW (ระบุคอลัมน์)

- แสดงคอลัมน์ในบรรทัด INSERT clause ที่ต้องการเพิ่มค่าในคอลัมน์นั้น
- ใส่ແລວໃໝ່ທີ່ມີຄ່າໃນແຕ່ລະຄອລັມນໍຕາມລຳດັບຄອລັມນໍທີ່ກໍາທັນດີໃນຕາຮາງ
- ລຶ້ອມຕົວອັກສະນະແລະວັນທີດ້ວຍ single quotation marks.

```
INSERT INTO locations(location_id, city, country_id)
```

```
VALUES (3300, 'Bangkok', 'TH') ;
```

```
INSERT INTO locations(location_id, city, country_id)
```

```
VALUES (3400, 'Kyoto', 'JP') ;
```

| Field | Type | Null | Key |
|----------------|------------------|------|-----|
| location_id | int(11) unsigned | NO | PRI |
| street_address | varchar(40) | YES | |
| postal_code | varchar(12) | YES | |
| city | varchar(30) | NO | |
| state_province | varchar(25) | YES | |
| country_id | char(2) | NO | MUL |

2. INSERTING A NEW ROW (ระบุค่าล้มน้ำ)

```
INSERT INTO countries(country_id, country_name, region_id)  
VALUES ('TH', 'Thailand', 3);
```

ก่อนที่จะเพิ่มข้อมูล Bangkok ในตาราง locations จะต้องเพิ่ม TH ใน countries ก่อน

```
INSERT INTO locations(location_id, city, country_id)  
VALUES (3300, 'Bangkok', 'TH');
```

Countries (Parent table)

| Field | Type | Null | Key |
|--------------|------------------|------|-----|
| country_id | char(2) | NO | PRI |
| country_name | varchar(40) | YES | |
| region_id | int(11) unsigned | NO | MUL |

Locations (Child table)

| Field | Type | Null | Key |
|----------------|------------------|------|-----|
| location_id | int(11) unsigned | NO | PRI |
| street_address | varchar(40) | YES | |
| postal_code | varchar(12) | YES | |
| city | varchar(30) | NO | |
| state_province | varchar(25) | YES | |
| country_id | char(2) | NO | MUL |

Exercise 1: INSERT INTO

- จะเขียน SQL เพิ่มข้อมูลให้กับตาราง locations และตารางอื่นที่เกี่ยวข้องตามข้อมูลนี้

| LOCATION_ID | STREET_ADDRESS | POSTAL_CODE | CITY | STATE_PROVINCE | COUNTRY_ID |
|-------------|----------------|-------------|-------|----------------|------------|
| 3500 | Sukhumvit | 21000 | Muang | Rayong | TH |

```
INSERT INTO countries
```

```
VALUES ('TH', 'Thailand', 3) ;
```

```
INSERT INTO locations
```

```
VALUES (3500, 'Sukhumvit', 21000, 'Muang', 'Rayong', 'TH') ;
```

Exercise 2: INSERT INTO

- จะเขียน SQL เพิ่มข้อมูลให้กับตาราง locations และตารางอื่นที่เกี่ยวข้องตามข้อมูลนี้

| LOCATION_ID | STREET_ADDRESS | POSTAL_CODE | CITY | STATE_PROVINCE | COUNTRY_ID |
|-------------|----------------|-------------|--------|----------------|------------|
| 3600 | | | Taipei | | TW |

```
INSERT INTO countries (country_id,country_name,region_id)
VALUES ('TW', 'Taiwan', 3) ;
```

```
INSERT INTO locations(location_id, country_id,city)
VALUES (3600, 'TW', 'Taipei') ;
```

Exercise 3: INSERT INTO

- จะเขียน SQL เพิ่มข้อมูลให้กับตาราง locations และตารางอื่นที่เกี่ยวข้องตามข้อมูลนี้

| LOCATION_ID | STREET_ADDRESS | POSTAL_CODE | CITY | STATE_PROVINCE | COUNTRY_ID |
|-------------|----------------|-------------|----------|----------------|------------|
| 3700 | | | Shanghai | | CN |

```
INSERT INTO locations(location_id, city, country_id)
VALUES      (3700, 'Shanghai', 'CN') ;
```

3. INSERTING ROWS WITH NULL VALUES

- ไม่กำหนดคอลัมน์ แต่ใส่ค่าในคอลัมน์ต่างๆ และ **NULL** ในคอลัมน์ที่ต้องการไม่ใส่ค่า

```
INSERT INTO locations  
VALUES      (3700, NULL, NULL, 'Xian', NULL,'CN' );
```

| Field | Type | Null | Key |
|----------------|------------------|------|-----|
| location_id | int(11) unsigned | NO | PRI |
| street_address | varchar(40) | YES | |
| postal_code | varchar(12) | YES | |
| city | varchar(30) | NO | |
| state_province | varchar(25) | YES | |
| country_id | char(2) | NO | MUL |

Exercise 4: INSERT INTO

- จงเพิ่มข้อมูลให้กับตาราง locations ตามข้อมูลนี้ แบบไม่ระบุชื่อคอลัมน์

| LOCATION_ID | STREET_ADDRESS | POSTAL_CODE | CITY | STATE_PROVINCE | COUNTRY_ID |
|-------------|----------------|-------------|--------|----------------|------------|
| 3900 | | | Harbin | | CN |

```
INSERT INTO locations
```

```
VALUES      (3900, NULL,NULL, 'Harbin', NULL, 'CN' ) ;
```

COMMON ERRORS ต้องระวัง

Common errors that can occur during user input:

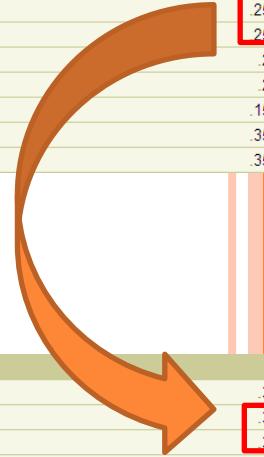
1. ใน NOT NULL column เกิด error เนื่องจากไม่ใส่ค่าใดๆ ในคำสั่ง INSERT ซึ่ง columnn นั้นห้ามเป็นค่า null
2. ใน column ที่กำหนด uniqueness constraint ใส่ค่าที่ซ้ำกับค่าที่มีอยู่เดิม
3. ละเมิด Foreign key constraint
4. ชนิดของข้อมูลที่ใส่ ไม่ตรงกับชนิดข้อมูลของ column นั้น
5. ค่าที่ใส่มีขนาดใหญ่เกินกว่าที่ columnn กำหนดไว้

CHANGING DATA IN A TABLE

SALES_REPS

| REPS_ID | NAME | SALARY | COMMISSION_PCT |
|---------|-----------|--------|----------------|
| 150 | Tucker | 10000 | .3 |
| 151 | Bernstein | 9500 | .25 |
| 152 | Hall | 9000 | .25 |
| 153 | Olsen | 8000 | .2 |
| 154 | Cambrault | 7500 | .2 |
| 155 | Tuvault | 7000 | .15 |
| 156 | King | 10000 | .35 |
| 157 | Sully | 9500 | .35 |

Update rows in the SALES_REPS tables



| REPS_ID | NAME | SALARY | COMMISSION_PCT |
|---------|-----------|--------|----------------|
| 150 | Tucker | 10000 | .3 |
| 151 | Bernstein | 9500 | .3 |
| 152 | Hall | 9000 | .3 |
| 153 | Olsen | 8000 | .2 |
| 154 | Cambrault | 7500 | .2 |
| 155 | Tuvault | 7000 | .15 |
| 156 | King | 10000 | .35 |
| 157 | Sully | 9500 | .35 |

UPDATE STATEMENT SYNTAX

- แก้ไขข้อมูลในตารางด้วย UPDATE statement:

```
UPDATE    table  
        SET      column = value [, column = value, ...]  
        [WHERE   condition ] ;
```

- Update more than one row at a time (if required).
 - *table* is the name of the table
 - *column* is the name of the column in the table to populate
 - *value* is the corresponding value or subquery for the column
 - *condition* identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators

COMPARISON CONDITIONS

| Operator | Meaning |
|----------|--------------------------|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| != | Not qual to |

....WHERE hire_date= '1998-05-04'

....WHERE salary>=6000

....WHERE last_name='Smite'

UPDATING ROWS IN A TABLE

- แบบที่ 1: แก้ไข 1 แถวหรือมากกว่า 1 แถว ด้วย WHERE clause

```
UPDATE employees  
SET salary = 10000  
WHERE employee_id = 101 ;
```

- แบบที่ 2: แก้ไขทุกแถวในตาราง ไม่จำเป็นต้องใช้ WHERE clause

```
UPDATE employees  
SET commission_pct = .3 ;
```

Exercise 5: UPDATE

- จงแก้ไขข้อมูลในตาราง locations เป็นตามข้อมูลใหม่นี้

| Column | Old | New |
|-------------|-------|-------|
| location_id | 1000 | 1000 |
| postal_code | 00989 | 10100 |

```
UPDATE locations  
SET postal_code = 10100  
WHERE location_id = 1000 ;
```

Exercise 6: UPDATE

- จงแก้ไขข้อมูลในตาราง jobs เป็นตามข้อมูลใหม่นี้

| Column | Old | New |
|------------|-----------|------|
| min_Salary | 4000-5000 | 6000 |

```
UPDATE    jobs
```

```
SET        min_salary = 6000
```

```
WHERE      min_salary >= 4000 and min_salary <=5000 ;
```

REMOVING A ROW FROM A TABLE

SALES_REPS

| REPS_ID | NAME | SALARY | COMMISSION_PCT |
|---------|------------|--------|----------------|
| 173 | Kumar | 6100 | .1 |
| 174 | Abel | 11000 | .3 |
| 175 | Hutton | 8800 | .25 |
| 176 | Taylor | 8600 | .2 |
| 177 | Livingston | 8400 | .2 |
| 179 | Johnson | 6200 | .1 |
| 202 | Fay | 6000 | |
| 203 | Mavis | 6500 | |
| 204 | Baer | 10000 | |

Delete a Row from a SALES_REPS Table

| REPS_ID | NAME | SALARY | COMMISSION_PCT |
|---------|------------|--------|----------------|
| 173 | Kumar | 6100 | .1 |
| 174 | Abel | 11000 | .3 |
| 175 | Hutton | 8800 | .25 |
| 176 | Taylor | 8600 | .2 |
| 177 | Livingston | 8400 | .2 |
| 179 | Johnson | 6200 | .1 |
| 202 | Fay | 6000 | |
| 203 | Mavis | 6500 | |

DELETE STATEMENT SYNTAX

- ลบแถวที่อยู่ในตารางด้วย DELETE statement:

```
DELETE FROM table  
[WHERE condition] ;
```

- *table* is the table name
- *condition* identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators

DELETING ROWS FROM A TABLE

- แบบที่ 1: ลบเฉพาะแถวที่ต้องการ ด้วย WHERE clause
- ตัวอย่าง : ลบแถวที่มี reps_id 204 ออกจากตาราง sales_reps

```
DELETE FROM sales_reps  
WHERE reps_id = 204 ;
```

- ตัวอย่าง : ลบแถวที่มี salary ตั้งแต่ 7000 ถึง 7500 ออกจากตาราง

```
DELETE FROM sales_reps  
WHERE salary >=7000 and salary <= 7500;
```

DELETING ROWS FROM A TABLE (CONT.)

- แบบที่ 2: ลบทุกแถวในตาราง โดยการห้ามใช้ WHERE clause

```
DELETE FROM sales_reps ;
```

- Note: ถึงแม้ไม่มีค่าในคอลัมน์ใดๆ เลย แต่ยังมี Table structure เหลืออยู่

DELETING ROWS: INTEGRITY CONSTRAINT ERROR

```
DELETE FROM regions ;
```

```
Cannot delete or update a parent row: a foreign key constraint fails  
(`g39336`.`countries`, CONSTRAINT `countries_regions_region_id`  
FOREIGN KEY (`region_id`) REFERENCES `regions` (`region_id`))
```

NOTE: คุณไม่สามารถลบแถวที่มี primary key ซึ่งใช้เป็น foreign key
ในตารางอื่นได้

Exercise 7: DELETE

- ลบข้อมูลในตาราง sales_reps โดยลบเฉพาะแถวที่มีพนักงานเงินเดือนมากกว่า 5000

```
DELETE FROM sales_reps  
WHERE salary > 5000;
```

Exercise 8: INSERT

- จะเขียน SQL statement เพิ่มข้อมูลให้กับตาราง lab_location ตามข้อมูลนี้ (แบบระบุคอลัมน์)

| LOCATION_ID | LOCATION_NAME |
|-------------|---------------|
| 002 | Rayong |
| 003 | Ranong |

```
INSERT INTO lab_location(location_id,location_name)  
VALUES (002,'Rayong'),(003,'Ranong');
```

Exercise 9: INSERT

- จงเขียน SQL statement เพื่อเพิ่มข้อมูลให้กับตาราง lab_emp ตามข้อมูลนี้
แบบไม่ระบุชื่อคอลัมน์

| ID | LAST_NAME | FIRST_NAME | SALARY | LOCATION_ID |
|-----|-----------|------------|--------|-------------|
| 004 | Woo | Woody | 15000 | 001 |
| 005 | Sun | Peng | | |

```
INSERT INTO lab_emp
```

```
VALUES
```

```
(004,'Woo','Woody',15000,001),(005,'Sun','Peng',NULL,NULL);
```

FOREIGN KEY CONSTRAINT (REFERENTIAL INTEGRITY CONSTRAINT)

- **FOREIGN KEY:** นิยามคอลัมน์ในตารางลูก
 - Insert Constraint: Value cannot be inserted in CHILD Table if the value is not lying in MASTER Table
 - Delete Constraint: Value cannot be deleted from MASTER Table if the value is lying in CHILD Table
- **ON DELETE CASCADE:** ลบ dependent rows ในตารางลูกเมื่อແກວໃນตารางแม่
ງູກລບ
- **ถ้ามี ON DELETE CASCADE**
 - เมื่อສ້າງລບແກວຂໍ້ອມຸລໃນตารางແມ່ ດ້ວຍຄໍາສັ່ງ “delete from ชື່ອຕາຮາງ;” ຈະລບແກວໃນ^{ທີ່}ตารางลູກທີ່ມາດ
 - เมื่อສ້າງລບແກວຂໍ້ອມຸລໃນตารางແມ່ ດ້ວຍຄໍາສັ່ງ “delete from ชື່ອຕາຮາງ where ;” ຈະລບ
ແກວໃນตารางລູກນັ້ນທີ່ຕຽບຕາມເຈື່ອນໄຂທ່ານນັ້ນ
- **ถ้าไม่มี ON DELETE CASCADE** เมื่อສ້າງລບແກວຂໍ້ອມຸລດ້ວຍຄໍາສັ່ງ “delete from ชື່ອຕາຮາງ;”
ມັນຈະພ້ອງເຕືອນ FK ດັ່ງນັ້ນຕ້ອງ
 - ທ່າງການລບ FK ໃນตารางແມ່ ແລະໃຊ້ຄໍາສັ່ງ “delete from ชື່ອຕາຮາງ” ເພື່ອລບແກວໃນตาราง
ແມ່ ແຕ່ຈະໄມ່ກະທບຂໍ້ອມຸລໃນตารางລູກ **ແຕ່ໄມ່ສົມເຫດສົມຜລ**

ถ้ามี ON DELETE CASCADE

```
CREATE TABLE Student (
    sno INT PRIMARY KEY,
    sname VARCHAR(20),
    age INT
);
```

```
CREATE TABLE Course (
    cno INT PRIMARY KEY,
    cname VARCHAR(20)
);
```

```
CREATE TABLE Enroll (
    sno INT,
    cno INT,
    jdate date,
    FOREIGN KEY(sno) REFERENCES Student(sno) ON DELETE CASCADE,
    FOREIGN KEY(cno) REFERENCES Course(cno) ON DELETE CASCADE
);
```

```
INSERT INTO Student(sno, sname, age)
VALUES(1, 'Ankit', 17),
```

```
(2, 'Ramya', 18),
(3, 'Ram', 16),
(4, 'Mai', 18);
```

```
INSERT INTO Course(cno, cname)
VALUES(101, 'c'),
(102, 'c++'),
(103, 'DBMS');
```

```
INSERT INTO Enroll(sno, cno, jdate)
VALUES(1, 101, '2021-06-05'),
(1, 102, '2021-06-07'),
(2, 103, '2021-06-05');
```

ถ้ามี ON DELETE CASCADE (ต่อ)

เมื่อใช้คำสั่ง Delete from student

```
where sno=1;
```

จะได้ผลลัพธ์คือ แถวข้อมูล sno=1 ในตาราง student จะลบ

| sno | sname | age |
|-----|-------|-----|
| 2 | Ramya | 18 |
| 3 | Ram | 16 |
| 4 | Mai | 18 |

และ แถวข้อมูลในตาราง enroll ที่มีค่าใน sno=1 จะลบไปด้วย

| sno | cno | jdate |
|-----|-----|------------|
| 2 | 103 | 2021-06-05 |

ถ้าไม่มี ON DELETE CASCADE

เมื่อใช้คำสั่ง Delete from student

```
where sno=1;
```

จะได้ผลลัพธ์คือ

Cannot delete or update a parent row: a foreign key constraint fails ('g36632`.`enroll`, CONSTRAINT `enroll_ibfk_1` FOREIGN KEY (`sno`) REFERENCES `student` (`sno`))

ดังนั้นต้อง ทำการลบ FK ในตารางแม่ และใช้คำสั่ง “delete from ชื่อตาราง” เพื่อลบแถวใน
ตารางแม่ แต่จะไม่กระทบข้อมูลในตารางลูก **แต่ไม่สมเหตุสมผล**

| student | sno | sname | age |
|---------|-----|-------|-----|
| | 2 | Ramya | 18 |
| | 3 | Ram | 16 |
| | 4 | Mai | 18 |

| enroll | sno | cno | jdate |
|--------|-----|-----|------------|
| | 1 | 101 | 2021-06-05 |
| | 1 | 102 | 2021-06-07 |
| | 2 | 103 | 2021-06-05 |

FOREIGN KEY CONSTRAINT (REFERENTIAL INTEGRITY CONSTRAINT)

- **FOREIGN KEY:** นิยามคอลัมน์ในตารางลูก
- **ON DELETE SET NULL:** แปลงค่าที่อยู่ใน foreign key เป็นค่า null เมื่อลบในตารางแม่ทุกกลบ
- ถ้ามี ON DELETE SET NULL
 - เมื่อสั่งลบแถวข้อมูลในตารางแม่ ด้วยคำสั่ง “delete from ชื่อตาราง;” จะเปลี่ยนค่าของ foreign key ในตารางลูกทั้งหมดเป็น ค่า NULL
 - เมื่อสั่งลบแถวข้อมูลในตารางแม่ ด้วยคำสั่ง “delete from ชื่อตาราง where ;” จะเปลี่ยนค่าของ foreign key ในตารางลูกที่ตรงตามเงื่อนไขเท่านั้นให้เป็น ค่า NULL

ถ้ามี ON DELETE SET NULL

```
CREATE TABLE Student (
    sno INT PRIMARY KEY,
    sname VARCHAR(20),
    age INT
);
```

```
CREATE TABLE Course (
    cno INT PRIMARY KEY,
    cname VARCHAR(20)
);
```

```
CREATE TABLE Enroll (
    sno INT,
    cno INT,
    jdate date,
    FOREIGN KEY(sno) REFERENCES Student(sno) ON DELETE SET NULL,
    FOREIGN KEY(cno) REFERENCES Course(cno) ON DELETE SET NULL
);
```

```
INSERT INTO Student(sno, sname, age)
VALUES(1, 'Ankit', 17),
```

```
(2, 'Ramya', 18),
(3, 'Ram', 16),
(4, 'Mai', 18);
```

```
INSERT INTO Course(cno, cname)
VALUES(101, 'c'),
(102, 'c++'),
(103, 'DBMS');
```

```
INSERT INTO Enroll(sno, cno, jdate)
VALUES(1, 101, '2021-06-05'),
(1, 102, '2021-06-07'),
(2, 103, '2021-06-05');
```

ถ้ามี ON DELETE SET NULL (ต่อ)

เมื่อใช้คำสั่ง Delete from student

```
where sno=1;
```

จะได้ผลลัพธ์คือ แถวข้อมูล sno=1 ในตาราง student ถูกลบ

| sno | sname | age |
|-----|-------|-----|
| 2 | Ramya | 18 |
| 3 | Ram | 16 |
| 4 | Mai | 18 |

และ แถวข้อมูลในตาราง enroll ที่มีค่าใน sno=1 เปลี่ยนเป็น NULL

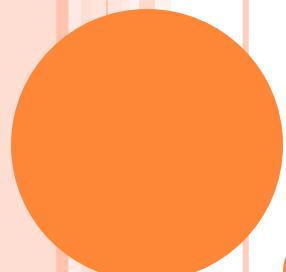
| sno | cno | jdate |
|------|-----|------------|
| NULL | 101 | 2021-06-05 |
| NULL | 102 | 2021-06-07 |
| 2 | 103 | 2021-06-05 |



INTRODUCTION TO STRUCTURED QUERY LANGUAGE

LAB 4

- **WRITING BASIC SQL SELECT STATEMENTS**



By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts

2/2565

OUTLINE ก่อนสอบกลางภาค

| Date | SQL |
|--------------------|---|
| 9, 11 JAN 2023 | <ul style="list-style-type: none">• Lab Introduction• Introduction to DBLearn (SQL tool) |
| 16, 18 JAN 2023 | LAB 1 - Creating and Managing Tables (DDL) |
| 23, 25 JAN 2023 | LAB 2 - Including Constraints (DDL) |
| 30 JAN, 1 FEB 2023 | LAB 3 - Manipulating Data (DML) |
| 6, 8 FEB 2023 | LAB 4 - SQL SELECT Statements <ul style="list-style-type: none">• Writing Basic |
| 13, 15 FEB 2023 | LAB 5 - SQL SELECT Statements <ul style="list-style-type: none">• Restricting and Sorting Data |
| 20, 22 FEB 2023 | - ทวนก่อนสอบ Quiz 1 |
| 27 FEB, 1 MAR 2023 | Quiz 1: LAB 1 – LAB 5 |

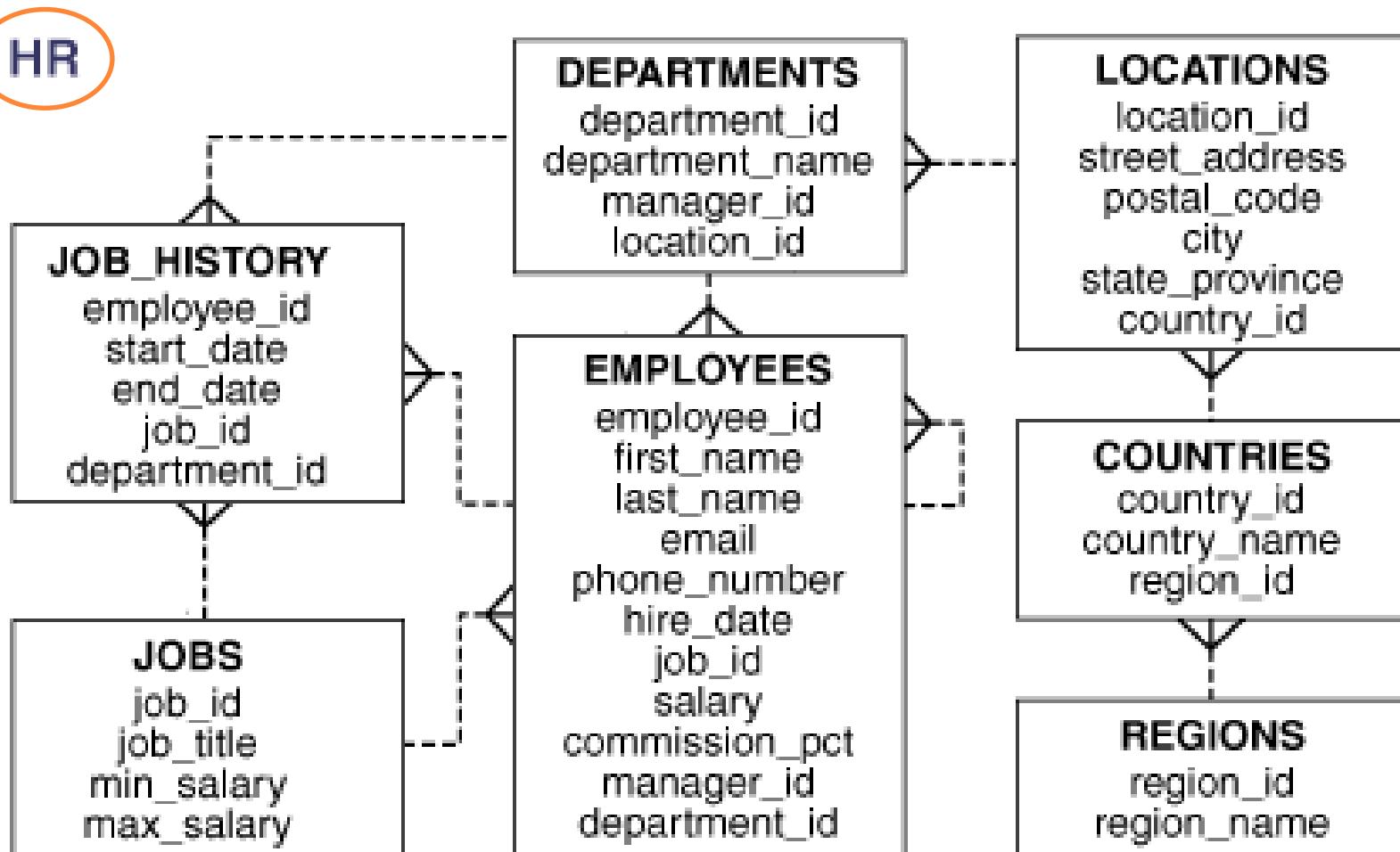
SQL STATEMENT

| Type | SQL Statement |
|---|---|
| Data Manipulation Language (DML) ภาษาสำหรับจัดการข้อมูล คือส่วนของภาษา SQL ที่อนุญาตให้คุณ ควบคุมหรือจัดการข้อมูล | SELECT INSERT UPDATE DELETE MERGE CALL EXPLAIN PLAN LOCK TABLE |
| Data Definition Language (DDL) อธิบายส่วนของ SQL ที่อนุญาตให้สร้าง, เปลี่ยน, และทำลายอ็อบเจกต์ ฐานข้อมูล อ็อบเจกต์ฐานข้อมูลเหล่านี้รวมถึงแบบแผน, ตาราง, มุมมอง , ลำดับ, แคดล็อก, ดัชนี, และ alias | CREATE ALTER DROP RENAME ANALYZE AUDIT COMMENT ASSOCIATE STATISTICS DISASSOCIATE STATISTICS |
| Transaction Control จัดการ transaction จากการเปลี่ยนแปลงที่เกิดจาก DML | COMMIT ROLLBACK SAVEPOINT SET TRANSACTION |

OBJECTIVE

- Introduction to Structured query language
- Writing Basic SQL SELECT Statements
- Sorting Data

TABLES USED IN THE COURSE



EXAMPLE: EMPLOYEE TABLE DESCRIPTION

| Column Name | Null? | Type |
|----------------|----------|----------------|
| EMPLOYEE_ID | NOT NULL | NUMBER (6) |
| FIRST_NAME | | VARCHAR2 (20) |
| LAST_NAME | NOT NULL | VARCHAR2 (25) |
| EMAIL | NOT NULL | VARCHAR2 (20) |
| PHONE_NUMBER | | VARCHAR2 (20) |
| HIRE_DATE | NOT NULL | DATE |
| JOB_ID | NOT NULL | VARCHAR2 (10) |
| SALARY | | NUMBER (8 , 2) |
| COMMISSION_PCT | | NUMBER (2 , 2) |
| MANAGER_ID | | NUMBER (6) |
| DEPARTMENT_ID | | NUMBER (4) |

HOW TO KNOW THE COLUMN LIST

```
DESCRIBE departments;
```

| Name | Null? | Type |
|-----------------|----------|--------------|
| DEPARTMENT_ID | NOT NULL | NUMBER(4) |
| DEPARTMENT_NAME | NOT NULL | VARCHAR2(30) |
| MANAGER_ID | | NUMBER(6) |
| LOCATION_ID | | NUMBER(4) |

WRITING SQL STATEMENTS

- SQL statements are **not case sensitive**.
- SQL statements can be **on one or more lines**.
- Keywords **cannot be abbreviated or split across lines**.
- Keywords typically are entered in **uppercase**.
- Clauses are usually placed on **separate lines**.
- Indents are used to enhance **readability**.

BASIC **SELECT** STATEMENT

```
SELECT *|[DISTINCT] column_name | expression [alias],...  
FROM table_name;
```

SELECT **is a list of one or more columns**

* **selects all column**

DISTINCT **suppresses duplicates**

Column_name|expression **selects the named column or the
expression**

alias **gives selected column different headings**

FROM *table_name* **specifies the table containing the columns**

SELECTING ALL COLUMNS

```
SELECT *  
FROM departments;
```

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|----------------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 30 | Purchasing | 114 | 1700 |
| 40 | Human Resources | 203 | 2400 |
| 50 | Shipping | 121 | 1500 |
| 60 | IT | 103 | 1400 |
| 70 | Public Relations | 204 | 2700 |
| 80 | Sales | 145 | 2500 |
| 90 | Executive | 100 | 1700 |
| 100 | Finance | 108 | 1700 |
| 110 | Accounting | 205 | 1700 |
| 120 | Treasury | | 1700 |
| 130 | Corporate Tax | | 1700 |
| 140 | Control And Credit | | 1700 |
| 150 | Shareholder Services | | 1700 |
| 160 | Benefits | | 1700 |
| 170 | Manufacturing | | 1700 |
| 180 | Construction | | 1700 |
| 190 | Contracting | | 1700 |
| 200 | Operations | | 1700 |

....

27 rows selected.

SELECTING **SPECIFIC** COLUMNS

ชื่อคอลัมน์มาจากการคอลัมน์ในตาราง
หลังคำ FROM

```
SELECT department_id, location_id  
FROM departments;
```

| DEPARTMENT_ID | LOCATION_ID |
|---------------|-------------|
| 10 | 1700 |
| 20 | 1800 |
| 30 | 1700 |
| 40 | 2400 |
| 50 | 1500 |
| 60 | 1400 |
| 70 | 2700 |
| 80 | 2500 |
| 90 | 1700 |
| 100 | 1700 |
| 110 | 1700 |
| 120 | 1700 |
| 130 | 1700 |
| 140 | 1700 |
| 150 | 1700 |
| 160 | 1700 |
| 170 | 1700 |
| 180 | 1700 |
| 190 | 1700 |
| 200 | 1700 |
| | |

27 rows selected.

ARITHMETIC EXPRESSIONS

- Create expressions by using arithmetic operators.
- Arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

| Operator | Description |
|----------|-------------|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |

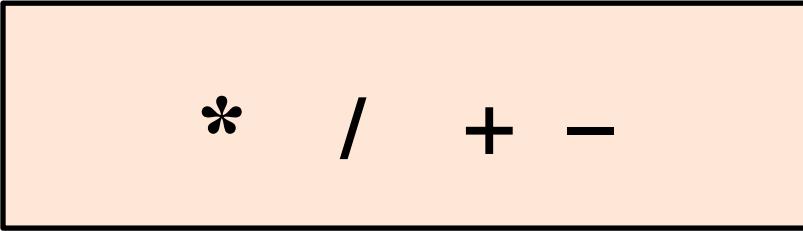
USING ARITHMETIC OPERATORS

```
SELECT last_name, salary, salary + 300  
FROM employees;
```

| LAST_NAME | SALARY | SALARY+300 |
|-----------|--------|------------|
| King | 24000 | 24300 |
| Kochhar | 17000 | 17300 |
| De Haan | 17000 | 17300 |
| Hunold | 9000 | 9300 |
| Ernst | 6000 | 6300 |
| Austin | 4800 | 5100 |
| Pataballa | 4800 | 5100 |
| Lorentz | 4200 | 4500 |
| Greenberg | 12000 | 12300 |
| Faviet | 9000 | 9300 |
| Chen | 8200 | 8500 |
| Sciarra | 7700 | 8000 |
| ... | | |
| Hartstein | 13000 | 13300 |
| Fay | 6000 | 6300 |
| Mavris | 6500 | 6800 |
| Baer | 10000 | 10300 |
| Higgins | 12000 | 12300 |
| Gietz | 8300 | 8600 |

107 rows selected.

OPERATOR PRECEDENCE



* / + -

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

OPERATOR PRECEDENCE (CONT.)

```
SELECT last_name, salary, 12*salary+100  
FROM employees;
```

| LAST_NAME | SALARY | 12*SALARY+100 |
|-----------|--------|---------------|
| King | 24000 | 288100 |
| Kochhar | 17000 | 204100 |
| De Haan | 17000 | 204100 |
| Hunold | 9000 | 108100 |
| Ernst | 6000 | 72100 |
| Austin | 4800 | 57700 |
| Pataballa | 4800 | 57700 |
| Lorentz | 4200 | 50500 |
| Greenberg | 12000 | 144100 |
| Faviet | 9000 | 108100 |
| ... | | |
| Mavris | 6500 | 78100 |
| Baer | 10000 | 120100 |
| Higgins | 12000 | 144100 |
| Gietz | 8300 | 99700 |

107 rows selected.

USING PARENTHESES

```
SELECT last_name, salary, 12*(salary+100)  
FROM employees;
```

| LAST_NAME | SALARY | 12*(SALARY+100) |
|-----------|--------|-----------------|
| King | 24000 | 289200 |
| Kochhar | 17000 | 205200 |
| De Haan | 17000 | 205200 |
| Hunold | 9000 | 109200 |
| Ernst | 6000 | 73200 |
| Austin | 4800 | 58800 |
| Pataballa | 4800 | 58800 |
| Lorentz | 4200 | 51600 |
| Greenberg | 12000 | 145200 |
| Faviet | 9000 | 109200 |
| Chen | 8200 | 99600 |
| Sciarra | 7700 | 93600 |
| Urman | 7800 | 94800 |
| Popp | 6900 | 84000 |
| ... | | |
| Fay | 6000 | 73200 |
| Mavris | 6500 | 79200 |
| Baer | 10000 | 121200 |
| Higgins | 12000 | 145200 |
| Gietz | 8300 | 100800 |

107 rows selected.

DEFINING A NULL VALUE

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as zero or a blank space.

```
SELECT last_name, job_id, salary, commission_pct  
FROM employees;
```

| LAST_NAME | JOB_ID | SALARY | COMMISSION_PCT |
|-----------|------------|--------|----------------|
| King | AD_PRES | 24000 | |
| Kochhar | AD_VP | 17000 | |
| ... | | | |
| Zlotkey | SA_MAN | 10500 | .2 |
| Abel | SA_REP | 11000 | .3 |
| Taylor | SA_REP | 8600 | .2 |
| ... | | | |
| Gietz | AC_ACCOUNT | 8300 | |

107 rows selected.

Remark: NOT NULL and PRIMARY KEY prevents nulls in column

Null

NULL VALUES IN ARITHMETIC EXPRESSIONS

- Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name, 12* salary*commission_pct  
FROM employees;
```

| LAST_NAME | 12*SALARY*COMMISSION_PCT |
|-----------|--------------------------|
| King | |
| Kochhar | |
| ... | |
| Zlotkey | 25200 |
| Abel | 39600 |
| Taylor | 20640 |
| ... | |
| Gietz | |

107 rows selected.

Null

Null

DEFINING A COLUMN ALIAS (นามแฝง)

- Oracle
 - การใช้ Alias แทนชื่อคอลัมน์เดิมในตาราง
 - Default แสดงตัวพิมพ์ใหญ่
 - ชื่อattribute **AS** นามแฝง
 - ชื่อattribute เว้นวรรค นามแฝง
 - นามแฝงต้องมี double quotation (" ")
ถ้านามแฝงนั้นมีช่องว่าง ตัวอักษรพิเศษ หรือ เป็น case sensitive
- MySQL
 - การใช้ Alias แทนชื่อคอลัมน์เดิมในตาราง
 - รูปแบบการเขียนคือ
 - ชื่อattribute **AS** นามแฝง
 - ชื่อattribute เว้นวรรค นามแฝง
 - ถ้านามแฝงนั้นมีช่องว่าง ตัวอักษรพิเศษ นามแฝงต้องมี back-tick ` ` หรือ double quote " " หรือ single quote ''
 - ต้องมี back tick(`_) สำหรับอ้างอิงไปยังคอลัมน์ที่ตั้งนามแฝงไว้

USING COLUMN ALIASES (MySQL)

```
SELECT last_name AS name, commission_pct comm  
FROM employees;
```

| | name | comm |
|---------|------|------|
| King | | |
| Kochhar | | |
| De Haan | | |
| ... | | |

107 rows selected.

```
SELECT last_name Name, salary*12 `Annual Salary`  
FROM employees;
```

| | Name | Annual Salary |
|---------|------|---------------|
| King | | 288000 |
| Kochhar | | 204000 |
| De Haan | | 204000 |
| ... | | |

107 rows selected.

CONCATENATION OPERATOR

- MySql uses Concat function
 - concat('string1', 'string2', 'string3',....)
 - concat(ชื่อattribute1, 'string2', ชื่อattribute2,...)
 - Example: concat(firstname, ' _ ', lastname)
- Oracle A concatenation operator:
 - Concatenates columns or character strings to other columns
เชื่อมคอลัมน์หรือตัวอักษรกับคอลัมน์อื่น
 - Is represented by two vertical bars (||)
 - Creates a resultant column that is a character expression

USING THE CONCATENATION OPERATOR (MySQL)

```
SELECT concat(last_name, job_id) Employees  
FROM employees;
```

| Employees |
|----------------|
| KingAD_PRES |
| KochharAD_VP |
| De HaanAD_VP |
| HunoldIT_PROG |
| ErnstIT_PROG |
| LorentzIT_PROG |
| MourgosST_MAN |
| RajsST_CLERK |
| ... |

107 rows selected.

LITERAL CHARACTER STRINGS

- กำหนด ตัวอักษร หรือ วันที่ ลงในบรรทัด SELECT
- A literal is a character, or a date included in the SELECT list.
- Date and character literal values must be enclosed within single quotation ('_) marks or double quotation (" _ ") marks.
- Each character string is output once for each row returned.
- In DBLearning (MySQL), Date format -> **YYYY-MM-DD**

USING LITERAL CHARACTER STRINGS (MySQL)

```
SELECT concat(last_name, ' is a ', job_id) `Employee Details`  
FROM employees;
```

| Employee Details |
|----------------------|
| King is a AD_PRES |
| Kochhar is a AD_VP |
| De Haan is a AD_VP |
| Hunold is a IT_PROG |
| Ernst is a IT_PROG |
| Lorentz is a IT_PROG |
| Mourgos is a ST_MAN |
| Rajs is a ST_CLERK |
| ... |

107 rows selected.

USING LITERAL CHARACTER STRINGS (ORACLE)

```
SELECT last_name || ' is a ' || job_id AS "Employee Details"  
FROM employees;
```

| Employee Details |
|----------------------|
| King is a AD_PRES |
| Kochhar is a AD_VP |
| De Haan is a AD_VP |
| Hunold is a IT_PROG |
| Ernst is a IT_PROG |
| Lorentz is a IT_PROG |
| Mourgos is a ST_MAN |
| Rajs is a ST_CLERK |
| ... |

107 rows selected.

DUPLICATE ROWS

```
SELECT department_id  
FROM employees;
```

| DEPARTMENT_ID | |
|---------------|----|
| | 90 |
| | 90 |
| | 90 |
| | 60 |
| | 60 |
| | 60 |
| | 50 |
| | 50 |
| | 50 |
| | |

107 rows selected.

ELIMINATING DUPLICATE Rows

- Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

```
SELECT DISTINCT department_id  
FROM employees;
```

| DEPARTMENT_ID |
|---------------|
| 10 |
| 20 |
| 30 |
| 40 |
| 50 |
| 60 |
| 70 |
| 80 |
| 90 |
| 100 |
| 110 |

12 rows selected.

ORDER BY CLAUSE

- Sort rows with the ORDER BY clause
- The **ORDER BY** clause comes last in the SELECT statement.

```
SELECT      last_name, job_id, department_id, hire_date  
FROM        employees  
ORDER BY    hire_date;
```

| LAST_NAME | JOB_ID | DEPARTMENT_ID | HIRE_DATE |
|-----------|---------|---------------|-----------|
| King | AD_PRES | 90 | 17-JUN-87 |
| Whalen | AD_ASST | 10 | 17-SEP-87 |
| Kochhar | AD_VP | 90 | 21-SEP-89 |
| Hunold | IT_PROG | 60 | 03-JAN-90 |
| Ernst | IT_PROG | 60 | 21-MAY-91 |
| ... | | | |

107 rows selected.

SUMMARY

- In this lesson, you should have learned how to:
- Retrieve data from 1 table by specifying column name or expression

```
SELECT    *|[DISTINCT]  column_name | expression [alias],...
FROM      table_name
```

SORTING DATA

- เรียงลำดับแล้วข้อมูลของผลลัพธ์จาก Query ด้วย ORDER BY
- 2 รูปแบบ
 - ASC (Ascending Order) หรือไม่ระบุ (by default) เรียงจาก น้อยไปมาก, A-Z, จำกวันที่อุดิตมาวันที่ปัจจุบัน
 - DESC (Descending Order) เรียงจาก มากไปน้อย, Z-A, จำกวันที่ปัจจุบัน
 - ย้อนไปวันที่อุดิต

```
SELECT    *[{[DISTINCT] column_name | expression [alias],...}]  
FROM      table_name  
[ ORDER BY {column_name, expr} [ASC | DESC] ] ;
```

SORTING IN ASCENDING ORDER

```
SELECT      last_name, job_id, department_id, hire_date  
FROM        employees  
ORDER BY    hire_date ASC ;
```

| last_name | job_id | department_id | hire_date |
|-----------|------------|---------------|------------|
| King | AD_PRES | 90 | 1987-06-17 |
| Whalen | AD_ASST | 10 | 1987-09-17 |
| Kochhar | AD_VP | 90 | 1989-09-21 |
| Hunold | IT_PROG | 60 | 1990-01-03 |
| Ernst | IT_PROG | 60 | 1991-05-21 |
| De Haan | AD_VP | 90 | 1993-01-13 |
| Mavris | HR_REP | 40 | 1994-06-07 |
| Baer | PR_REP | 70 | 1994-06-07 |
| Higgins | AC_MGR | 110 | 1994-06-07 |
| Gietz | AC_ACCOUNT | 110 | 1994-06-07 |

SORTING IN DESCENDING ORDER

```
SELECT      last_name, job_id, department_id, hire_date
FROM        employees
ORDER BY    hire_date DESC ;
```

| last_name | job_id | department_id | hire_date |
|------------|----------|---------------|------------|
| Banda | SA_REP | 80 | 2000-04-21 |
| Kumar | SA_REP | 80 | 2000-04-21 |
| Ande | SA_REP | 80 | 2000-03-24 |
| Markle | ST_CLERK | 50 | 2000-03-08 |
| Lee | SA_REP | 80 | 2000-02-23 |
| Philtanker | ST_CLERK | 50 | 2000-02-06 |
| Geoni | SH_CLERK | 50 | 2000-02-03 |
| Zlotkey | SA_MAN | 80 | 2000-01-29 |
| Marvins | SA_REP | 80 | 2000-01-24 |
| Grant | SH_CLERK | 50 | 2000-01-13 |
| | | | |

107 rows selected.

SORTING BY COLUMN ALIAS

```
SELECT      employee_id, last_name, salary*12 annsal  
FROM        employees  
ORDER BY    annsal ;
```

| EMPLOYEE_ID | LAST_NAME | ANNSAL |
|-------------|-------------|--------|
| 132 | Olson | 25200 |
| 128 | Markle | 26400 |
| 136 | Phil tanker | 26400 |
| 127 | Landry | 28800 |
| 135 | Gee | 28800 |
| 119 | Colmenares | 30000 |
| 140 | Patel | 30000 |
| 144 | Vargas | 30000 |
| 191 | Perkins | 30000 |
| 182 | Sullivan | 30000 |
| 131 | Marlow | 30000 |
| 118 | Himuro | 31200 |
| 143 | Matos | 31200 |
| | | |

107 rows selected.

SORTING BY COLUMN ALIAS WITH SPACE (MySQL)

```
SELECT      employee_id, last_name, salary*12 `annual sal`  
FROM        employees  
ORDER BY    `annual sal` ;
```

| EMPLOYEE_ID | LAST_NAME | ANNSAL |
|-------------|-------------|--------|
| 132 | Olson | 25200 |
| 128 | Markle | 26400 |
| 136 | Phil tanker | 26400 |
| 127 | Landry | 28800 |
| 135 | Gee | 28800 |
| 119 | Colmenares | 30000 |
| 140 | Patel | 30000 |
| 144 | Vargas | 30000 |
| 191 | Perkins | 30000 |
| 182 | Sullivan | 30000 |
| 131 | Marlow | 30000 |
| 118 | Himuro | 31200 |
| 143 | Matos | 31200 |
| | | |

107 rows selected.

SORTING BY MULTIPLE COLUMNS

- The order of ORDER BY list is the order of sort.
- Example: Display the last name and salaries of all employees. Order the results by department number, and then in descending order by salary

```
SELECT      last_name, department_id, salary  
FROM        employees  
ORDER BY    department_id, salary DESC ;
```

| LAST_NAME | DEPARTMENT_ID | SALARY |
|------------|---------------|--------|
| Whalen | 10 | 4400 |
| Hartstein | 20 | 13000 |
| Fay | 20 | 6000 |
| Raphaely | 30 | 11000 |
| Khoo | 30 | 3100 |
| Baida | 30 | 2900 |
| Tobias | 30 | 2800 |
| Himuro | 30 | 2600 |
| Colmenares | 30 | 2500 |
| Mavris | 40 | 6500 |
| Fripp | 50 | 8200 |
| Weiss | 50 | 8000 |
| Kaufling | 50 | 7900 |
| ... | | |

107 rows selected.

Remark: You can sort by a column that is not in the SELECT list.

ORDER BY CLAUSE

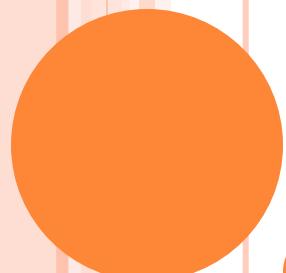
- Sort rows with the ORDER BY clause
 - ASC: ascending order, default
 - DESC: descending order
- The **ORDER BY** clause comes last in the SELECT statement.

```
SELECT    *|[DISTINCT]  column_name | expression [alias],...  
FROM      table_name  
[ WHERE    condition(s) ]  
[ ORDER BY {column_name, expr} [ASC | DESC] ] ;
```

INTRODUCTION TO STRUCTURED QUERY LANGUAGE

LAB 5

- RESTRICTING DATA



By Kanokwan Atchariyachanvanich
Faculty of Information Technology

KMITL

Database System Concepts

2/2565

OUTLINE ก่อนสอบกลางภาค

| Date | SQL |
|--------------------|---|
| 9, 11 JAN 2023 | <ul style="list-style-type: none">• Lab Introduction• Introduction to DBLearn (SQL tool) |
| 16, 18 JAN 2023 | LAB 1 - Creating and Managing Tables (DDL) |
| 23, 25 JAN 2023 | LAB 2 - Including Constraints (DDL) |
| 30 JAN, 1 FEB 2023 | LAB 3 - Manipulating Data (DML) |
| 6, 8 FEB 2023 | LAB 4 - SQL SELECT Statements <ul style="list-style-type: none">• Writing Basic |
| 13, 15 FEB 2023 | LAB 5 - SQL SELECT Statements <ul style="list-style-type: none">• Restricting Data |
| 20, 22 FEB 2023 | - ทวนก่อนสอบ Quiz 1 |
| 27 FEB, 1 MAR 2023 | Quiz 1: LAB 1 – LAB 5 |

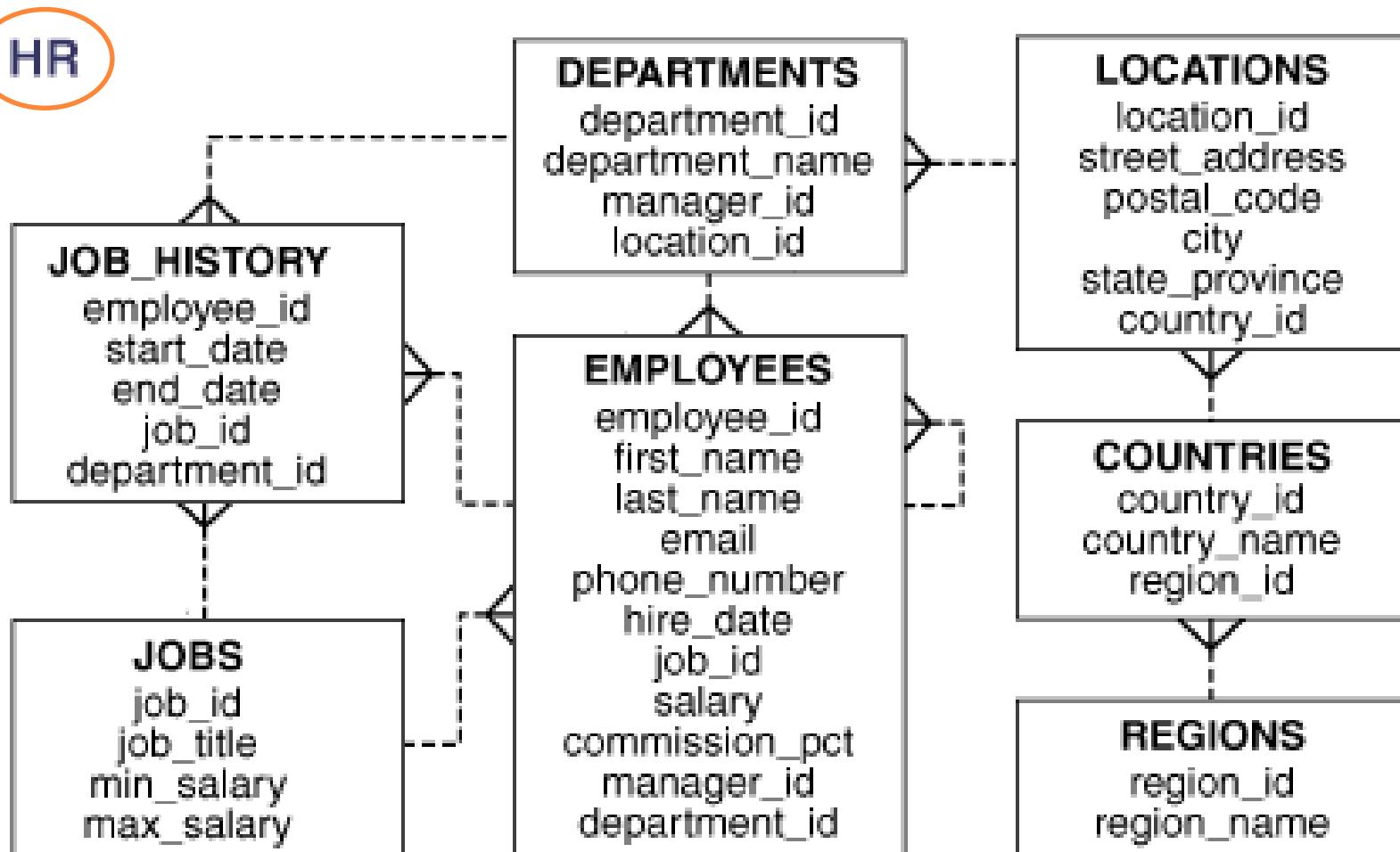
SQL STATEMENT

| Type | SQL Statement |
|---|---|
| Data Manipulation Language (DML) ภาษาสำหรับจัดการข้อมูล คือส่วนของภาษา SQL ที่อนุญาตให้คุณ ควบคุมหรือจัดการข้อมูล | SELECT INSERT UPDATE DELETE MERGE CALL EXPLAIN PLAN LOCK TABLE |
| Data Definition Language (DDL) อธิบายส่วนของ SQL ที่อนุญาตให้สร้าง, เปลี่ยน, และทำลายอ็อบเจกต์ ฐานข้อมูล อ็อบเจกต์ฐานข้อมูลเหล่านี้รวมถึงแบบแผน, ตาราง, มุมมอง , ลำดับ, แคดล็อก, ดัชนี, และ alias | CREATE ALTER DROP RENAME ANALYZE AUDIT COMMENT ASSOCIATE STATISTICS DISASSOCIATE STATISTICS |
| Transaction Control จัดการ transaction จากการเปลี่ยนแปลงที่เกิดจาก DML | COMMIT ROLLBACK SAVEPOINT SET TRANSACTION |

OBJECTIVES

- After completing this lesson, you should be able to do the following:
 - Limit the rows retrieved by a query

TABLES USED IN THE COURSE



EXAMPLE: EMPLOYEE TABLE DESCRIPTION

| Column Name | Null? | Type |
|----------------|----------|----------------|
| EMPLOYEE_ID | NOT NULL | NUMBER (6) |
| FIRST_NAME | | VARCHAR2 (20) |
| LAST_NAME | NOT NULL | VARCHAR2 (25) |
| EMAIL | NOT NULL | VARCHAR2 (20) |
| PHONE_NUMBER | | VARCHAR2 (20) |
| HIRE_DATE | NOT NULL | DATE |
| JOB_ID | NOT NULL | VARCHAR2 (10) |
| SALARY | | NUMBER (8 , 2) |
| COMMISSION_PCT | | NUMBER (2 , 2) |
| MANAGER_ID | | NUMBER (6) |
| DEPARTMENT_ID | | NUMBER (4) |

LIMITING ROWS USING A SELECTION

| EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|-------------|-----------|---------|---------------|
| 100 | King | AD_PRES | 90 |
| 101 | Kochhar | AD_VP | 90 |
| 102 | De Haan | AD_VP | 90 |
| 103 | Hunold | IT_PROG | 60 |
| 104 | Ernst | IT_PROG | 60 |
| 107 | Lorentz | IT_PROG | 60 |
| 124 | Mourgos | ST_MAN | 50 |

....

20 rows selected.

“retrieve all employees in department 90”

| EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|-------------|-----------|---------|---------------|
| 100 | King | AD_PRES | 90 |
| 101 | Kochhar | AD_VP | 90 |
| 102 | De Haan | AD_VP | 90 |

LIMITING THE Rows SELECTED

- Restrict the rows returned by using the **WHERE** clause.

```
SELECT *|[DISTINCT] column_name | expression [alias],...
FROM    table_name
WHERE   condition(s);
```

- Where restricts the query to rows that meet a condition
condition is composed of column names, expressions, constants, literal values, and a comparison operator

COMPARISON CONDITIONS

| Operator | Meaning |
|----------|--------------------------|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| != | Not qual to |

....WHERE hire_date='1998-05-04'

....WHERE salary>=6000

....WHERE last_name='Smite'

USING THE WHERE CLAUSE

```
SELECT employee_id, last_name, job_id, department_id  
FROM employees  
WHERE department_id = 90 ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|-------------|-----------|---------|---------------|
| 100 | King | AD_PRES | 90 |
| 101 | Kochhar | AD_VP | 90 |
| 102 | De Haan | AD_VP | 90 |

EXERCISE # 1

- จงแสดงรหัสพนักงาน ชื่อจริง และนามสกุล ของพนักงานที่มีรหัส 100

```
SELECT employee_id, first_name, last_name  
FROM employees  
WHERE employee_id = 100;
```

CHARACTER STRINGS AND DATES

- Character strings and date values are enclosed in single quotation marks ('').
- Character values are not case sensitive.
- The default date format is YYYY-MM-DD.

```
SELECT last_name, job_id, department_id  
FROM employees  
WHERE last_name = 'wHalen';
```

| LAST_NAME | JOB_ID | DEPARTMENT_ID |
|-----------|---------|---------------|
| Whalen | AD_ASST | 10 |

EXERCISE # 2

- จงแสดงรหัสสถานที่ตั้งสำนักงาน ชื่อเมือง และรหัสไปรษณีย์ ของสถานที่ตั้งสำนักงานที่ไม่ได้ตั้งอยู่ในประเทศสหรัฐ US

```
SELECT location_id, city, postal_code  
FROM locations  
WHERE country_id != "US";
```

USING COMPARISON CONDITIONS

```
SELECT last_name, salary  
FROM employees  
WHERE salary <= 3000;
```

| LAST_NAME | SALARY |
|-----------|--------|
| Matos | 2600 |
| Vargas | 2500 |
| | |

| LAST_NAME | SALARY |
|-----------|--------|
| OConnell | 2600 |
| Grant | 2600 |

26 rows selected.

EXERCISE # 3

- จงแสดงชื่อจริง นามสกุล และวันที่เริ่มจ้างงาน ของพนักงานที่ถูกจ้างงานตั้งแต่วันที่ 1 มกราคม 2000 เป็นต้นไป โดยเรียงลำดับข้อมูลเงินเดือนจากมากไปน้อย

```
SELECT      first_name, last_name, hire_date  
FROM        employees  
WHERE       hire_date >= "2000-01-01"  
ORDER BY    salary DESC;
```

OTHER COMPARISON CONDITIONS

| Operator | Meaning |
|----------------------------|---------------------------------|
| BETWEEN . . . AND . . . | Between two values (inclusive), |
| IN (set) | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

USING THE BETWEEN CONDITION

- Use the **BETWEEN** condition to display rows based on a range of values.

```
SELECT last_name, salary  
FROM employees  
WHERE salary BETWEEN 2500 AND 3500 ;
```



Lower limit

Upper limit

| LAST_NAME | SALARY |
|-------------|--------|
| Khoo | 3100 |
| Baida | 2900 |
| Tobias | 2800 |
| Himuro | 2600 |
| Colmenares | 2500 |
| Nayer | 3200 |
| Mikkilineni | 2700 |
| ... | |

EXERCISE # 4

- จงแสดงรหัสพนักงาน อีเมล และเบอร์โทรศัพท์ ของพนักงานที่มีรหัสตั้งแต่ 110 ถึง 120 (ใช้ BETWEEN)

```
SELECT employee_id, email, phone_number  
FROM employees  
WHERE employee_id BETWEEN 110 AND 120;
```

EXERCISE # 5

- จงแสดงรหัสตำแหน่งงาน ชื่อตำแหน่งงาน และเงินเดือนตำแหน่งงานที่มีเงินเดือนตำแหน่งที่สุดที่ไม่ได้อยู่ในช่วง 3000 ถึง 5000 (ใช้ BETWEEN)

```
SELECT    job_id, job_title, min_salary  
FROM      jobs  
WHERE    min_salary NOT BETWEEN 3000 AND 5000;
```

USING THE IN CONDITION

- Use the **IN** membership condition to test for values in a list.

```
SELECT employee_id, last_name, salary, manager_id  
FROM employees  
WHERE manager_id IN (100, 101, 201);
```

| EMPLOYEE_ID | LAST_NAME | SALARY | MANAGER_ID |
|-------------|-----------|--------|------------|
| 101 | Kochhar | 17000 | 100 |
| 102 | De Haan | 17000 | 100 |
| 114 | Raphaely | 11000 | 100 |
| 120 | Weiss | 8000 | 100 |
| 121 | Fripp | 8200 | 100 |
| 122 | Kaufling | 7900 | 100 |
| 123 | Vollman | 6500 | 100 |
| 124 | Mourgos | 5800 | 100 |
| 145 | Russell | 14000 | 100 |
| 146 | Partners | 13500 | 100 |
| 147 | Errazuriz | 12000 | 100 |
| 148 | Cambrault | 11000 | 100 |
| 149 | Zlotkey | 10500 | 100 |
| 201 | Hartstein | 13000 | 100 |
| 108 | Greenberg | 12000 | 101 |
| 200 | Whalen | 4400 | 101 |
| 203 | Mavris | 6500 | 101 |
| 204 | Baer | 10000 | 101 |
| 205 | Higgins | 12000 | 101 |
| 202 | Fay | 6000 | 201 |

Remark: if characters or dates are used in the list,
they must be enclosed in single quotation marks

EXERCISE # 6

- จงแสดงนามสกุล รหัสงาน และรหัสแผนก ของพนักงานที่ทำงานอยู่ในแผนกรหัส 10 20 30 หรือ 40 (ใช้ IN)

```
SELECT    last_name, job_id, department_id  
FROM      employees  
WHERE     department_id IN (10,20,30,40);
```

EXERCISE # 7

- จงแสดงรหัสสถานที่ตั้งสำนักงาน ชื่อเมือง และรหัสไปรษณีย์ ของสถานที่ตั้งสำนักงานที่ไม่ได้ตั้งอยู่ในเมือง Venice, Tokyo, Toronto และ Bern (ใช้ IN)

```
SELECT location_id, city, postal_code  
FROM locations  
WHERE city NOT IN ("Venice", "Tokyo", "Toronto", "Bern");
```

USING THE LIKE CONDITION

- Use the **LIKE** condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
 - % denotes zero or many characters.
 - _ denotes one character.

```
SELECT first_name  
FROM employees  
WHERE first_name LIKE 'S%';
```

13 rows selected.

USING THE LIKE CONDITION

- You can combine pattern-matching characters.
- Example: displays the names of all employees whose last names have an o as the second character

```
SELECT last_name  
FROM employees  
WHERE last_name LIKE '_o%' ;
```

| LAST_NAME |
|------------|
| Colmenares |
| Doran |
| Fox |
| Johnson |
| Jones |
| Kochhar |
| Lorentz |
| Mourgos |
| Popp |
| Rogers |
| Tobias |
| Vollman |

12 rows selected.

EXERCISE # 8

- จงแสดง รหัสพนักงาน และรหัสงาน ของพนักงานที่เริ่มทำงานในเดือนมกราคม ของปีเดียวกัน กำหนดให้ใช้ LIKE

```
SELECT employee_id, job_id  
FROM employees  
WHERE hire_date LIKE '%-01-%';
```

EXERCISE # 9

- จงแสดง ชื่อเต็ม (คั่นชื่อจริงและนามสกุลด้วย whitespace เช่น David Austin ตั้งชื่อว่า name) และเงินเดือน โดยแสดงเฉพาะพนักงานที่ชื่อจริงมี 5 ตัวอักษร และอักษรตัวที่สองคือตัว a

```
SELECT    CONCAT(first_name, ' ', last_name) name, salary  
FROM      employees  
WHERE     first_name LIKE '_a__';
```

USING THE LIKE CONDITION (MySQL)

- **ESCAPE identifier default : **
- **ESCAPE identifier to search for the *actual % and _ symbols*, or have special “*and & or |*”**
- Example: search for job_id that contains 'SA_'

```
SELECT employee_id, last_name, job_id  
FROM employees  
WHERE job_id LIKE "%SA\_%";
```

| employee_id | last_name | job_id |
|-------------|-----------|--------|
| 145 | Russell | SA_MAN |
| 146 | Partners | SA_MAN |
| 147 | Errazuriz | SA_MAN |
| 148 | Cambrault | SA_MAN |
| 149 | Zlotkey | SA_MAN |

USING THE LIKE CONDITION (MySQL)

- **ESCAPE identifier default : **
- **ESCAPE identifier ex: \$ to search for the *actual % and _ symbols*, or have special “*and & or |*”**
- Example: search for job_id that contains 'SA_'

```
SELECT employee_id, last_name, job_id  
FROM employees  
WHERE job_id LIKE '%SA$_%' ESCAPE '$';
```

| employee_id | last_name | job_id |
|-------------|-----------|--------|
| 145 | Russell | SA_MAN |
| 146 | Partners | SA_MAN |
| 147 | Errazuriz | SA_MAN |
| 148 | Cambrault | SA_MAN |
| 149 | Zlotkey | SA_MAN |

35 rows selected.

USING THE NULL CONDITION

- Test for nulls with the IS NULL operator.
- Example: retrieve the last names and managers of all employee who do not have a manager

```
SELECT last_name, manager_id  
FROM employees  
WHERE manager_id IS NULL ;
```

| LAST_NAME | MANAGER_ID |
|-----------|------------|
| King | |

LOGICAL CONDITIONS

| Operator | Meaning |
|----------|---|
| AND | Returns TRUE if <i>both</i> component conditions are true |
| OR | Returns TRUE if <i>either</i> component condition is true |
| NOT | Returns TRUE if the following condition is false |

USING THE AND OPERATOR

- **AND requires both conditions to be true.**
- Example: only employees who have a job title that contains the string MAN *and* earn \$10,000 or more are selected

```
SELECT      employee_id, last_name, job_id, salary  
FROM        employees  
WHERE       salary >=10000  
AND         job_id LIKE '%MAN%' ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|-------------|-----------|--------|--------|
| 114 | Raphaely | PU_MAN | 11000 |
| 145 | Russell | SA_MAN | 14000 |
| 146 | Partners | SA_MAN | 13500 |
| 147 | Errazuriz | SA_MAN | 12000 |
| 148 | Cambrault | SA_MAN | 11000 |
| 149 | Zlotkey | SA_MAN | 10500 |
| 201 | Hartstein | MK_MAN | 13000 |

USING THE OR OPERATOR

- **OR** requires either conditions to be true.
- Example: any employee who has a job ID containing MAN or earn \$10,000 or more are selected

```
SELECT      employee_id, last_name, job_id, salary  
FROM        employees  
WHERE       salary >=10000  
OR          job_id LIKE '%MAN%' ;
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|-------------|-----------|---------|--------|
| 100 | King | AD_PRES | 24000 |
| 101 | Kochhar | AD_VP | 17000 |
| 102 | De Haan | AD_VP | 17000 |
| 108 | Greenberg | FL_MGR | 12000 |
| 114 | Raphaely | PU_MAN | 11000 |
| 120 | Weiss | ST_MAN | 8000 |
| 121 | Fripp | ST_MAN | 8200 |
| 122 | Kaufling | ST_MAN | 7900 |
| 123 | Vollman | ST_MAN | 6500 |
| 124 | Mourgos | ST_MAN | 5800 |
| 145 | Russell | SA_MAN | 14000 |
| 146 | Partners | SA_MAN | 13500 |
| 147 | Errazuriz | SA_MAN | 12000 |
| 148 | Cambrault | SA_MAN | 11000 |
| 149 | Zlotkey | SA_MAN | 10500 |
| 150 | Tucker | SA REP | 10000 |
| 156 | King | SA REP | 10000 |
| 162 | Vishney | SA REP | 10500 |
| 168 | Ozer | SA REP | 11500 |
| 169 | Bloom | SA REP | 10000 |
| 174 | Abel | SA REP | 11000 |
| 201 | Hartstein | MK_MAN | 13000 |
| 204 | Baer | PR REP | 10000 |
| 205 | Higgins | AC_MGR | 12000 |

EXERCISE # 10

- จงแสดงชื่อแผนก รหัสผู้จัดการ และรหัสสถานที่ตั้งสำนักงาน ของแผนกที่มีชื่อ
ขึ้นต้นด้วย IT หรือมีผู้จัดการดูแลอยู่

```
SELECT department_name, manager_id, location_id  
FROM departments  
WHERE department_name LIKE "IT%"  
OR manager_id IS NOT NULL;
```

EXERCISE # 11

- จงแสดงชื่อจริง นามสกุล และเงินเดือน ของพนักงานที่อยู่ในแผนกรหัส 80 และ มีเงินเดือนสูงกว่า 10000

```
SELECT      first_name, last_name, salary  
FROM        employees  
WHERE       department_id = 80  
AND         salary > 10000;
```

EXERCISE # 12

- จงแสดงชื่อจริง และอีเมลของพนักงานที่มีเงินเดือนอยู่ระหว่าง \$2000 และ \$3000 และรหัสงานขึ้นต้นด้วย ST_ (ต้องใช้ BETWEEN, LIKE)

| first_name | email |
|------------|----------|
| Irene | IMIKKILI |
| James | JLANDRY |

```
SELECT first_name, email  
FROM employees  
WHERE salary between 2000 and 3000  
AND job_id LIKE 'ST\_%';
```

USING THE NOT OPERATOR

- Example: displays the last names and job ID of all employees whose job ID *is not* IT_PROG, ST_CLERK, or SA_REP

```
SELECT      last_name, job_id  
FROM        employees  
WHERE       job_id  
           NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

| LAST_NAME | JOB_ID |
|-----------|---------|
| King | AD_PRES |
| Kochhar | AD_VP |
| De Haan | AD_VP |
| Mourgos | ST_MAN |
| Zlotkey | SA_MAN |
| Whalen | AD_ASST |

52 rows selected.

EXERCISE # 13

- จงแสดงชื่อถนนและที่อยู่ และ รัฐ/จังหวัด ของสถานที่ตั้งสำนักงาน (คั่นด้วย comma ตามด้วย whitespace เช่น 2017 Shinjuku-ku, Tokyo Prefecture ตั้งชื่อใหม่ว่า Address) ที่มีรหัสที่ตั้ง ที่ไม่ได้อยู่ระหว่าง 1500 และ 1800 และข้อมูล รัฐ/จังหวัด ต้องไม่เป็นค่าว่าง

Address

2017 Shinjuku-ku, Tokyo Prefecture

2014 Jabberwocky Rd, Texas

```
SELECT concat(street_address, ', ', state_province) Address  
FROM locations  
WHERE location_id NOT BETWEEN 1500 AND 1800  
AND state_province IS NOT NULL ;
```

RULES OF PRECEDENCE

| Order Evaluated | Operator |
|-----------------------|-------------------------------|
| 1 * / + - | Arithmetic operators |
| 2 | Concatenation operator |
| 3 =, >, >=, <, <=, <> | Comparison conditions |
| 4 | IS [NOT] NULL, LIKE, [NOT] IN |
| 5 | [NOT] BETWEEN |
| 6 | NOT logical condition |
| 7 | AND logical condition |
| 8 | OR logical condition |

Override rules of precedence by using parentheses.

RULES OF PRECEDENCE

```
SELECT      last_name, job_id, salary  
FROM        employees  
WHERE       job_id= 'SA_REP'  
OR          job_id= 'AD_PRES'  
AND         salary > 15000;
```

The diagram illustrates the precedence of operators in the SQL query. Braces group the conditions into two main parts: one for the OR condition (labeled 1) and another for the AND condition (labeled 2).

- Select the row if an employee is a president *and* earns more than \$15,000, *or* if the employee is a sales representative

| LAST_NAME | JOB_ID | SALARY |
|-----------|---------|--------|
| King | AD_PRES | 24000 |
| Abel | SA_REP | 11000 |
| Taylor | SA_REP | 8600 |
| Grant | SA_REP | 7000 |

31 rows selected.

RULES OF PRECEDENCE

- Use parentheses to force priority

```
SELECT      last_name, job_id, salary  
FROM        employees  
WHERE       ( job_id= 'SA_REP'  
OR          job_id= 'AD_PRES')  
AND         salary > 15000;
```

The diagram illustrates the precedence of operators in the WHERE clause. Braces group the conditions into two sets: one for SA_REP and another for AD_PRES. The AND operator has brace 2, indicating it applies to the entire WHERE clause.

- Select the row if an employee is a president *or* a sales representative, *and* if the employee earns more than \$15,000

| LAST_NAME | JOB_ID | SALARY | 40 |
|-----------|---------|--------|----|
| King | AD_PRES | 24000 | |

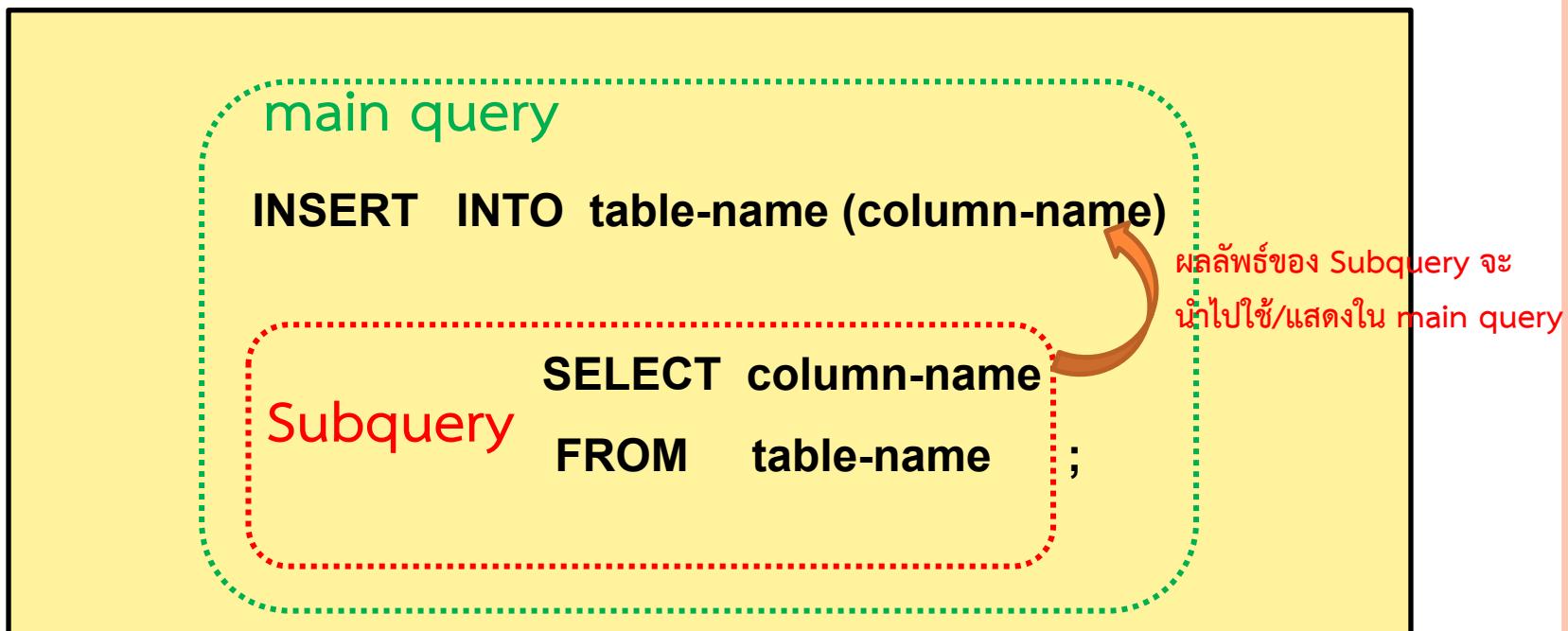
SUMMARY

- In this lesson, you should have learned how to:
- Use the WHERE clause to restrict rows of output
 - Use the comparison conditions
 - Use the BETWEEN, IN, LIKE, and NULL conditions
 - Apply the logical AND, OR, and NOT operators
- Use the ORDER BY clause to sort rows of output

```
SELECT    *|[DISTINCT] column_name | expression [alias],...  
FROM      table_name  
[ WHERE    condition(s) ]  
[ ORDER BY {column_name, expr, alias} [ASC | DESC] ] ;
```

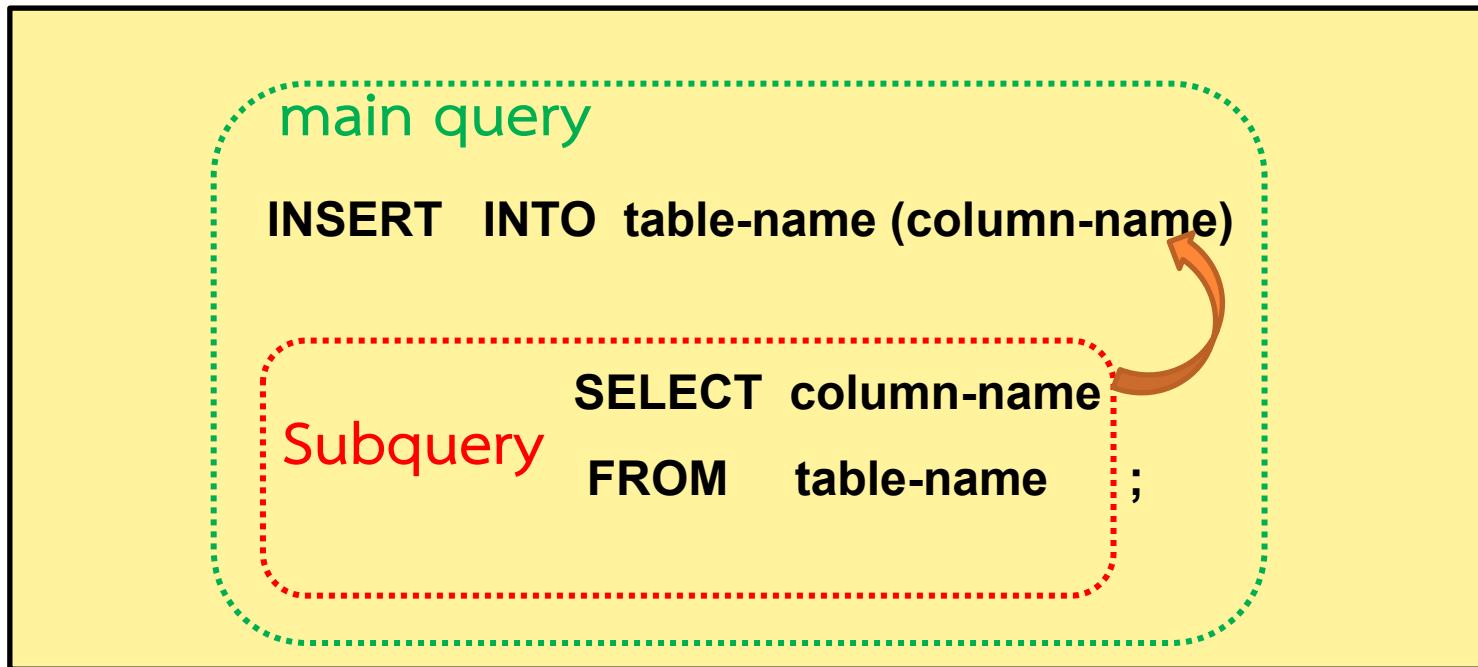
SELECT & DATA MANIPULATION LANGUAGE

○ INSERT



INSERTING MULTIPLE NEW ROWS

- COPY ROWS FROM ANOTHER TABLE ต้องการนำข้อมูลจากตารางอื่นมาใส่ในตารางใหม่ ดังนั้นข้อมูลที่เราไม่รู้จำเป็นต้องใช้ subquery ดึงข้อมูลนั้นมา
- เขียน INSERT statement ด้วย subquery ห้ามใช้บรรทัด VALUES clause.
- จับคู่คอลัมน์ใน INSERT clause กับคอลัมน์จาก subquery.



INSERTING MULTIPLE NEW ROWS

- COPYING ROWS FROM ANOTHER TABLE
- เขียน INSERT statement ด้วย subquery
- ห้ามใช้บรรทัด VALUES clause.
- จับคู่คอลัมน์ใน INSERT clause กับคอลัมน์จาก subquery.

```
INSERT INTO sales_reps (reps_id, name, salary, commission_pct)
    SELECT employee_id, last_name, salary, commission_pct
        FROM employees ;
```

Subquery
returns
rows into
the table

INSERTING MULTIPLE NEW ROWS

- COPYING ROWS FROM ANOTHER TABLE
- เขียน INSERT statement ด้วย subquery
- ห้ามใช้บรรทัด VALUES clause.
- จับคู่คอลัมน์ใน INSERT clause กับคอลัมน์จาก subquery.

```
INSERT INTO sales_reps (reps_id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
    FROM employees
   WHERE job_id > 'R' ;
```

Subquery
returns
rows into
the table

INSERT INTO + Subquery

Exercise # 14

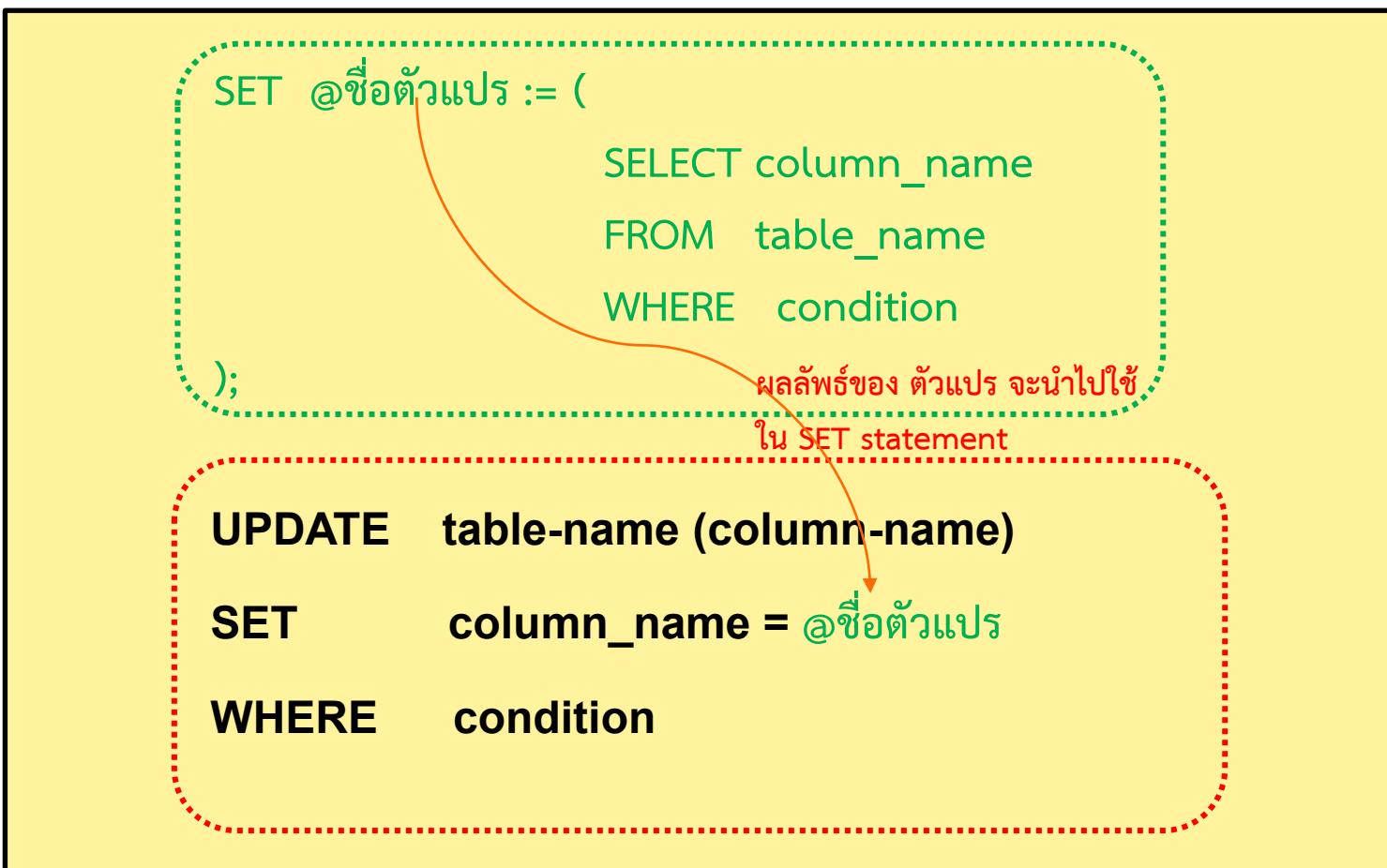
- จงเพิ่มข้อมูลให้กับตาราง sales_reps โดยนำข้อมูลจากตาราง employees ซึ่ง เอาเฉพาะพนักงานที่มีเงินเดือนมากกว่า 5000 (ระบุคอลัมน์)

| Table | Column1 | Column2 | Column3 | Column4 |
|------------|-------------|-----------|---------|----------------|
| sales_reps | reps_id | name | salary | commission_pct |
| employees | employee_id | last_name | salary | commission_pct |

```
INSERT INTO sales_reps (reps_id, name, salary, commission_pct)
    SELECT employee_id, last_name, salary, commission_pct
        FROM employees
    WHERE salary > 5000 ;
```

SELECT & DATA MANIPULATION LANGUAGE

- UPDATE แก้ไขค่าในคอลัมน์ ด้วยการใช้ defined value (ตัวแปร) ใน UPDATE statements เพื่อแก้ไขค่าต่างๆ ในคอลัมน์ตามค่าที่ได้จาก SELECT statement ซึ่งเป็นตารางเดียวกับที่ต้องการแก้ไข



UPDATING ONE COLUMN WITH A SUBQUERY (MySQL)

- แก้ไข 1 คอลัมน์ในตารางเดียวกัน ด้วยการใช้ user defined variable
- ต้องกำหนด user defined variable เพื่อเก็บ result set ที่ได้จาก subquery
- ตัวอย่าง : แก้ไขเงินเดือนของพนักงานรหัส 150 ให้เหมือนกับเงินเดือนของพนักงานรหัส 153.

```
SET      @value_id := (  
                        SELECT salary  
                        FROM    employees  
                        WHERE   employee_id = 153  
);  
  
UPDATE  employees  
        SET      salary      = @value_id  
        WHERE   employee_id = 150 ;
```

UPDATE + Defined value

Exercise # 15

- จงเขียน SQL statement เพื่อแก้ไขอัตราคอมมิชชัน (commission_pct) ของ พนักงานรหัส 180 ให้เหมือนกับอัตราคอมมิชชันของพนักงานรหัส 170 โดยตั้ง ชื่อตัวแปรคือ comm

```
SET      @comm := (
                      SELECT  commission_pct
                      FROM    employees
                      WHERE   employee_id = 170
);
UPDATE  employees
SET      commission_pct      = @comm
WHERE   employee_id = 180 ;
```

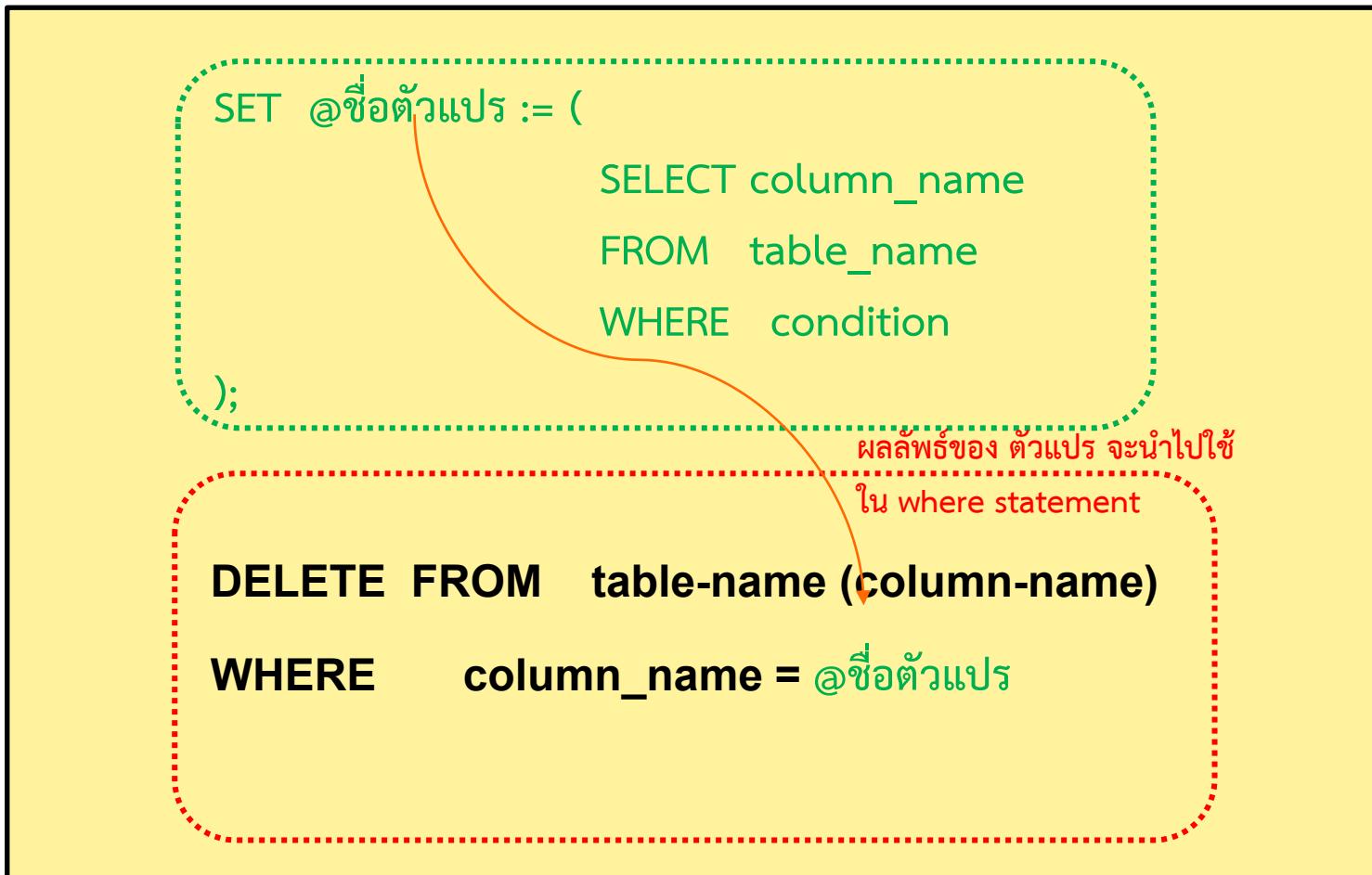
UPDATING ROWS BASED ON ANOTHER TABLE

- แก้ไขด้วยการใช้ Subquery ใน UPDATE statements เพื่ออัปเดตผล
ต่างๆ ในตารางตามค่าที่อยู่ในอีกตารางหนึ่ง

```
UPDATE sales_reps
SET salary = (SELECT salary
               FROM employees
               WHERE employee_id = 151) ,
    commission_pct = (SELECT commission_pct
                       FROM employees
                       WHERE employee_id = 151)
WHERE reps_id = 151 ;
```

SELECT & DATA MANIPULATION LANGUAGE

- DELETE: ลบແລວຂໍ້ມູນໃນຕາງໆ ຈຶ່ງມີເງື່ອນໄຂຕາມຄ່າທີ່ໄດ້ຈາກການໃໝ່ SELECT statement ໂດຍທີ່ຕາງໆທີ່ຖຸກລົບແລວຂໍ້ມູນກັບຕາງໆທີ່ໄດ້ຄ່າເງື່ອນໄຂ ອີຕາງເດືອກນິກົນ



DELETE + Defined value

Exercise # 16

- ลบແກວໃນຕາറາງ sales_reps ທີ່ມີເງິນເດືອນນ້ອຍກວ່າ ເງິນເດືອນຂອງ sales_reps ຮຫສ 105 ໂດຍຕັ້ງຊື່ອຕົວແປຣຄື່ອ sal

```
SET      @sal := (  
                      select  salary  
                      from    sales_reps  
                     where reps_id = 105);
```

```
DELETE  FROM    sales_reps  
 WHERE   salary < @sal ;
```

DELETING ROWS BASED ON ANOTHER TABLE

- ลบคอลัมน์ ด้วยการใช้ Subquery ใน DELETE FROM statements เพื่อ
ลบແດວຕ່າງໆ ในตารางตามค่าที่อยู่ในอีกตารางหนึ่ง
- ตัวอย่าง : ลบແດວໃນตาราง sales_reps ທີ່ມີ reps_id ແນ້ອນກັບ ຮັດ
ພນັກງານທີ່ມີນາມສກຸລຄື່ອ Banda ຈາກตาราง employees

```
DELETE FROM sales_reps
WHERE reps_id =
      (SELECT employee_id
       FROM employees
       WHERE last_name='Banda') ;
```

- Note: Don't forget to verify your deletion.*

DELETE + Subquery

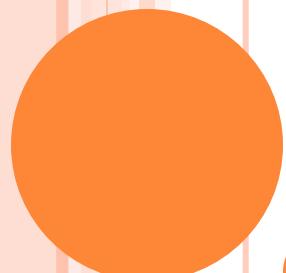
Exercise # 17

- ลบแถวในตาราง sales_reps ที่มีเงินเดือนน้อยกว่า เงินเดือนของ employee รหัส 105 ในตาราง employees

```
DELETE FROM sales_reps  
WHERE salary < (select salary  
from employees  
where employee_id = 105);
```

INTRODUCTION TO SQL

DISPLAYING DATA FROM MULTIPLE TABLES (JOIN)



By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts



OBJECTIVE

- After completing this lesson, you should be able to do the following:
- Write SELECT statements to access data from more than one table using equijoins and nonequijoins
- Join a table to itself by using a self-join

TYPES OF JOINS

- Joins that are compliant with the SQL:1999 standard include the following:
 1. Cross joins (ผลลัพธ์เมื่อjoin กับ Cartesian product)
 2. Equijoin (Old-style join)
 3. Natural joins
 4. Join with ON clause
 5. Join with USING clause
 6. Self-Joins Using the ON Clause
 7. Full (or two-sided) outer joins

INNER VERSUS OUTER JOINS

- In SQL:1999, the join of two tables returning only matched rows is called an **inner join**. Inner Join consists of :
 - Natural join
 - Join with On clause
 - Join with Using clause
 - Equijoin (Old-style join)
- A join between two tables that returns the results of an inner join as well as the results of unmatched rows is **outer join**.
 - Left outer join
 - Right outer join
 - Full outer join

1. CREATING CROSS JOINS

- The **CROSS JOIN** clause produces the cross-product of two tables.
- This is also called a **Cartesian product** between the two tables.

```
SELECT *
FROM      employees
CROSS JOIN departments ;
```

Cartesian product

```
SELECT *
FROM      employees, departments ;
```

RECALL: CARTESIAN PRODUCT

*** Attributes of both relations have distinct names

R

| | P_CODE | P_DESCRPT | PRICE |
|---|--------|------------|---------|
| ▶ | 123456 | Flashlight | \$5.26 |
| | 123457 | Lamp | \$25.15 |
| | 123458 | Box Fan | \$10.99 |
| | 213345 | 9v battery | \$1.92 |
| | 254467 | 100W bulb | \$1.47 |
| | 311452 | Powerdrill | \$34.99 |

6 rows

PRODUCT

S

| | STORE | AISLE | SHELF |
|---|-------|-------|-------|
| ▶ | 23 | V | 5 |
| | 24 | K | 9 |
| | 25 | Z | 6 |

3 rows

yields

C

| | P_CODE | P_DESCRPT | PRICE | STORE | AISLE | SHELF |
|---|--------|------------|---------|-------|-------|-------|
| ▶ | 123456 | Flashlight | \$5.26 | 23 | V | 5 |
| | 123456 | Flashlight | \$5.26 | 24 | K | 9 |
| | 123456 | Flashlight | \$5.26 | 25 | Z | 6 |
| | 123457 | Lamp | \$25.15 | 23 | V | 5 |
| | 123457 | Lamp | \$25.15 | 24 | K | 9 |
| | 123457 | Lamp | \$25.15 | 25 | Z | 6 |
| | 123458 | Box Fan | \$10.99 | 23 | V | 5 |
| | 123458 | Box Fan | \$10.99 | 24 | K | 9 |
| | 123458 | Box Fan | \$10.99 | 25 | Z | 6 |
| | 213345 | 9v battery | \$1.92 | 23 | V | 5 |
| | 213345 | 9v battery | \$1.92 | 24 | K | 9 |
| | 213345 | 9v battery | \$1.92 | 25 | Z | 6 |
| | 311452 | Powerdrill | \$34.99 | 23 | V | 5 |
| | 311452 | Powerdrill | \$34.99 | 24 | K | 9 |
| | 311452 | Powerdrill | \$34.99 | 25 | Z | 6 |
| | 254467 | 100W bulb | \$1.47 | 23 | V | 5 |
| | 254467 | 100W bulb | \$1.47 | 24 | K | 9 |
| | 254467 | 100W bulb | \$1.47 | 25 | Z | 6 |

SELECT *
FROM R, S ;

18 rows

GENERATING A CARTESIAN PRODUCT

EMPLOYEES (107 rows)

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-----------|------------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | | | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP | 17000 | | 100 | 90 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-93 | AD_VP | 17000 | | 100 | 90 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-90 | IT_PROG | 9000 | | 102 | 60 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-91 | IT_PROG | 6000 | | 103 | 60 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 25-JUN-97 | IT_PROG | 4800 | | 103 | 60 |
| 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-98 | IT_PROG | 4800 | | 103 | 60 |
| 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 07-FEB-99 | IT_PROG | 4200 | | 103 | 60 |
| 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-94 | FI_MGR | 12000 | | 101 | 100 |
| 109 | Daniel | Faviet | DFAVIET | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000 | | 108 | 100 |
| 110 | John | Chen | JCHEN | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200 | | 108 | 100 |

DEPARTMENTS (27 rows)

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|----------------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 30 | Purchasing | 114 | 1700 |
| 40 | Human Resources | 203 | 2400 |
| 50 | Shipping | 121 | 1500 |
| 60 | IT | 103 | 1400 |
| 70 | Public Relations | 204 | 2700 |
| 80 | Sales | 145 | 2500 |
| 90 | Executive | 100 | 1700 |
| 100 | Finance | 108 | 1700 |
| 110 | Accounting | 205 | 1700 |
| 120 | Treasury | | 1700 |
| 130 | Corporate Tax | | 1700 |
| 140 | Control And Credit | | 1700 |
| 150 | Shareholder Services | | 1700 |
| 160 | Benefits | | 1700 |
| 170 | Manufacturing | | 1700 |
| 180 | Construction | | 1700 |
| 190 | Contracting | | 1700 |
| 200 | Operations | | 1700 |

```
SELECT *
FROM employees, departments;
```

Cartesian Product:
107 x 27 = 2889 rows



| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-----------|------------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | | | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP | 17000 | | 100 | 90 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-93 | AD_VP | 17000 | | 100 | 90 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-90 | IT_PROG | 9000 | | 102 | 60 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-91 | IT_PROG | 6000 | | 103 | 60 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 25-JUN-97 | IT_PROG | 4800 | | 103 | 60 |
| 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-98 | IT_PROG | 4800 | | 103 | 60 |
| 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 07-FEB-99 | IT_PROG | 4200 | | 103 | 60 |
| 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-94 | FI_MGR | 12000 | | 101 | 100 |
| 109 | Daniel | Faviet | DFAVIET | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000 | | 108 | 100 |
| 110 | John | Chen | JCHEN | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200 | | 108 | 100 |
| 111 | Ismael | Sciarra | ISCIARRA | 515.124.4369 | 30-SEP-97 | FI_ACCOUNT | 7700 | | 108 | 100 |

2889 rows selected.

GENERATING A CROSS JOIN

EMPLOYEES (107 rows)

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-----------|------------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | | | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP | 17000 | | 100 | 90 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-93 | AD_VP | 17000 | | 100 | 90 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-90 | IT_PROG | 9000 | | 102 | 60 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-91 | IT_PROG | 6000 | | 103 | 60 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 25-JUN-97 | IT_PROG | 4800 | | 103 | 60 |
| 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-98 | IT_PROG | 4800 | | 103 | 60 |
| 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 07-FEB-99 | IT_PROG | 4200 | | 103 | 60 |
| 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-94 | FI_MGR | 12000 | | 101 | 100 |
| 109 | Daniel | Faviet | DFAVIET | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000 | | 108 | 100 |
| 110 | John | Chen | JCHEN | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200 | | 108 | 100 |

```
SELECT *  
FROM employees  
CROSS JOIN departments;
```

Cartesian Product:
 $107 \times 27 = 2889$ rows



DEPARTMENTS (27 rows)

| DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---------------|----------------------|------------|-------------|
| 10 | Administration | 200 | 1700 |
| 20 | Marketing | 201 | 1800 |
| 30 | Purchasing | 114 | 1700 |
| 40 | Human Resources | 203 | 2400 |
| 50 | Shipping | 121 | 1500 |
| 60 | IT | 103 | 1400 |
| 70 | Public Relations | 204 | 2700 |
| 80 | Sales | 145 | 2500 |
| 90 | Executive | 100 | 1700 |
| 100 | Finance | 108 | 1700 |
| 110 | Accounting | 205 | 1700 |
| 120 | Treasury | | 1700 |
| 130 | Corporate Tax | | 1700 |
| 140 | Control And Credit | | 1700 |
| 150 | Shareholder Services | | 1700 |
| 160 | Benefits | | 1700 |
| 170 | Manufacturing | | 1700 |
| 180 | Construction | | 1700 |
| 190 | Contracting | | 1700 |
| 200 | Operations | | 1700 |

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|-----------|----------|--------------|-----------|------------|--------|----------------|------------|---------------|
| 100 | Steven | King | SKING | 515.123.4567 | 17-JUN-87 | AD_PRES | 24000 | | | 90 |
| 101 | Neena | Kochhar | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP | 17000 | | 100 | 90 |
| 102 | Lex | De Haan | LDEHAAN | 515.123.4569 | 13-JAN-93 | AD_VP | 17000 | | 100 | 90 |
| 103 | Alexander | Hunold | AHUNOLD | 590.423.4567 | 03-JAN-90 | IT_PROG | 9000 | | 102 | 60 |
| 104 | Bruce | Ernst | BERNST | 590.423.4568 | 21-MAY-91 | IT_PROG | 6000 | | 103 | 60 |
| 105 | David | Austin | DAUSTIN | 590.423.4569 | 25-JUN-97 | IT_PROG | 4800 | | 103 | 60 |
| 106 | Valli | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-98 | IT_PROG | 4800 | | 103 | 60 |
| 107 | Diana | Lorentz | DLORENTZ | 590.423.5567 | 07-FEB-99 | IT_PROG | 4200 | | 103 | 60 |
| 108 | Nancy | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-94 | FI_MGR | 12000 | | 101 | 100 |
| 109 | Daniel | Faviet | DFAVIET | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000 | | 108 | 100 |
| 110 | John | Chen | JCHEN | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200 | | 108 | 100 |
| 111 | Ismael | Sciarras | ISCIARRA | 515.124.4369 | 30-SEP-97 | FI_ACCOUNT | 7700 | | 108 | 100 |

2889 rows selected.

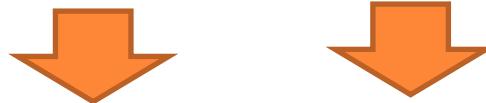
OBTAINING DATA FROM MULTIPLE TABLES

EMPLOYEES

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100 | King | 90 |
| 101 | Kochhar | 90 |
| ... | | |
| 202 | Fay | 20 |
| 205 | Higgins | 110 |
| 206 | Gietz | 110 |

DEPARTMENTS

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---------------|-----------------|-------------|
| 10 | Administration | 1700 |
| 20 | Marketing | 1800 |
| 50 | Shipping | 1500 |
| 60 | IT | 1400 |
| 80 | Sales | 2500 |
| 90 | Executive | 1700 |
| 110 | Accounting | 1700 |
| 190 | Contracting | 1700 |



| EMPLOYEE_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|-------------|---------------|-----------------|
| 200 | 10 | Administration |
| 201 | 20 | Marketing |
| 202 | 20 | Marketing |
| ... | | |
| 102 | 90 | Executive |
| 205 | 110 | Accounting |
| 206 | 110 | Accounting |

2. EQUIJOIN (OLD-STYLE JOIN)

- used to combine rows from two or more tables, based on a related column between them.
- Cartesian product with condition WHERE

```
SELECT table1_name.column_name, table2_name.column_name
FROM   table1_name, table2_name
WHERE  table1_name.column_name = table2_name.column_name;
```

The diagram illustrates the structure of an EQUIJOIN query. It shows the `SELECT` and `FROM` clauses grouped together by a blue brace on the right, and the `WHERE` clause grouped by a blue brace below it, both pointing to the text "Cartesian product".

EXAMPLE: EQUIJOIN

Query: Find all orders with their product names and price.

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

```
SELECT *
FROM ORDER, PRODUCT
WHERE
ORDER.P_CODE = PRODUCT.P_CODE ;
```

EXAMPLE : EQUIJOIN

ORDER

PRODUCT

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE |
|----------|-------------|--------------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

| PRODUCT.P_CODE | P_NAME | PRICE |
|----------------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

Automatically
rename

Step 1: Cartesian Product

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 222220 | คอมพิวเตอร์ | 30000 |
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 1 | C001 | 111110 | 333330 | ปากกา | 500 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |
| 2 | C002 | 222220 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 333330 | ปากกา | 500 |

EXAMPLE : EQUIJOIN

Step 2 : Select Join attribute P_CODE values are equal

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 222220 | คอมพิวเตอร์ | 30000 |
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 1 | C001 | 111110 | 333330 | ปากกา | 500 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |
| 2 | C002 | 222220 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 333330 | ปากกา | 500 |



| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |

EXAMPLE : EQUIJOIN

Step 3 : use a Projection to eliminate the duplicate attributes

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |



```
SELECT *
FROM   ORDER, PRODUCT
WHERE
ORDER.P_CODE = PRODUCT.P_CODE ;
```

| ORDER_NO | CUSTOMER_NO | P_CODE | P_NAME | PRICE |
|----------|-------------|--------|-------------|-------|
| 1 | C001 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | คอมพิวเตอร์ | 30000 |

2. EQUIJOIN

Department Table

| DEPARTMENT_ID | DEPARTMENT_NAME |
|---------------|------------------|
| 10 | Administration |
| 20 | Marketing |
| 30 | Purchasing |
| 40 | Human Resources |
| 50 | Shipping |
| 60 | T |
| 70 | Public Relations |
| 80 | Sales |
| 90 | Executive |
| 100 | Finance |
| 110 | Accounting |



Primary Key

Employees Table

| EMPLOYEE_ID | DEPARTMENT_ID |
|-------------|---------------|
| 100 | 90 |
| 101 | 90 |
| 102 | 90 |
| 103 | 60 |
| 104 | 60 |
| 105 | 60 |
| 106 | 60 |
| 107 | 60 |
| 108 | 100 |
| 109 | 100 |
| 110 | 100 |



Foreign Key



2. RETRIEVING RECORDS WITH EQUIJOIN

จงเขียน SQL Query แสดงรหัสพนักงาน นามสกุล รหัสแผนก และรหัสสถานที่ของแผนกนั้นๆ

```
SELECT employee_id, last_name, employees.department_id, location_id  
FROM employees, departments  
WHERE employees.department_id = departments.department_id ;
```

| employee_id | last_name | department_id | location_id |
|-------------|-----------|---------------|-------------|
| 103 | Hunold | 60 | 1400 |
| 104 | Ernst | 60 | 1400 |
| 105 | Austin | 60 | 1400 |
| 106 | Pataballa | 60 | 1400 |
| 107 | Lorentz | 60 | 1400 |

ADDITIONAL SEARCH CONDITIONS USING THE AND OPERATOR

To display employee Matos' employee_id, department number and department name.

```
SELECT      employee_id, employees.department_id, department_name  
FROM        employees, departments  
WHERE       employees.department_id = departments.department_id  
AND         last_name = 'Matos' ;
```

| EMPLOYEE_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|-------------|---------------|-----------------|
| 143 | 50 | Shipping |



USING TABLE ALIASES

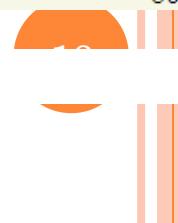
- Use table aliases to simplify queries.
- Use table aliases to improve performance.

Table Aliases

```
SELECT employee_id, last_name, location_id, d.department_id  
FROM employees e, departments d  
WHERE e.department_id = d.department_id;
```

| EMPLOYEE_ID | LAST_NAME | LOCATION_ID | DEPARTMENT_ID |
|-------------|-----------|-------------|---------------|
| 200 | Whalen | 1700 | 10 |
| 201 | Hartstein | 1800 | 20 |
| 202 | Fay | 1800 | 20 |
| 114 | Raphaely | 1700 | 30 |
| 115 | Khoo | 1700 | 30 |
| 116 | Baida | 1700 | 30 |
| ... | | | |

106 rows selected.



GUIDELINES FOR TABLE ALIASES (MySQL)

- Table aliases can be up to 256 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid for only the current SELECT statement.
- Help keep SQL code smaller therefore using less memory

JOINING MORE THAN TWO TABLES

EMPLOYEES

| LAST_NAME | DEPARTMENT_ID |
|-----------|---------------|
| Whalen | 10 |
| Hartstein | 20 |
| Fay | 20 |
| Raphaely | 30 |
| Khoo | 30 |
| Baida | 30 |

| DEPARTMENT_ID |
|---------------|
| 10 |
| 20 |
| 20 |
| 30 |
| 30 |
| 30 |

| DEPARTMENTS |
|-----------------|
| department_id |
| department_name |
| manager_id |
| location_id |

| EMPLOYEES |
|----------------|
| employee_id |
| first_name |
| last_name |
| email |
| phone_number |
| hire_date |
| job_id |
| salary |
| commission_pct |
| manager_id |
| department_id |

| LOCATIONS |
|----------------|
| location_id |
| street_address |
| postal_code |
| city |
| state_province |
| country_id |

NOTE: To join n tables, you need a minimum of n-1 join conditions

DEPARTMENTS

| LOCATION_ID |
|-------------|
| 1700 |
| 1800 |
| 1800 |
| 1700 |
| 1700 |
| 1700 |

LOCATIONS

| LOCATION_ID | CITY | STATE_PROVINCE |
|-------------|---------------------|------------------|
| 1000 | Roma | |
| 1100 | Venice | |
| 1200 | Tokyo | Tokyo Prefecture |
| 1300 | Hiroshima | |
| 1400 | Southlake | Texas |
| 1500 | South San Francisco | California |
| 1600 | South Brunswick | New Jersey |
| 1700 | Seattle | Washington |
| 1800 | Toronto | Ontario |

EXAMPLE: EQUIJOIN MORE THAN TWO TABLES

```
SELECT last_name, department_name, city  
FROM employees e , departments d, locations l  
WHERE e.department_id = d.department_id  
AND d.location_id = l.location_id;
```

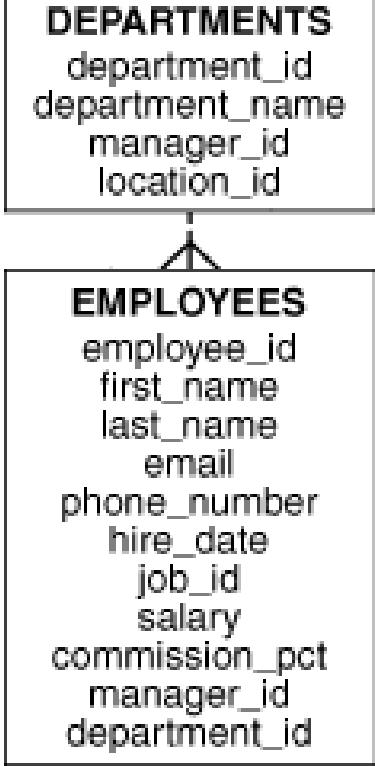
| LAST_NAME | DEPARTMENT_NAME | CITY |
|-----------|------------------|-----------|
| King | Executive | Seattle |
| Kochhar | Executive | Seattle |
| De Haan | Executive | Seattle |
| Hunold | IT | Southlake |
| Ernst | IT | Southlake |
| Austin | IT | Southlake |
| Pataballa | IT | Southlake |
| Lorentz | IT | Southlake |
| Greenberg | Finance | Seattle |
| ... | | |
| Baer | Public Relations | Munich |
| Higgins | Accounting | Seattle |
| Gietz | Accounting | Seattle |

106 rows selected.

EXERCISE # 1- EQUI-JOIN (OLD-JOIN)

จงเขียน SQL Query ที่แสดงชื่อ นามสกุล รหัสแผนก
(department_id) และ ชื่อแผนก (department_name) ของ
พนักงานที่ทำงานในแผนก Shipping (ใช้ Equijoin)
ตั้ง Table alias สำหรับ employees คือ e , departments คือ d

```
SELECT      first_name,           last_name,  
            e.department_id,       department_name  
  
FROM        Employees e,      departments d  
  
WHERE       e.department_id =  d.department_id  
  
AND         department_name = 'Shipping';
```



INNER VERSUS OUTER JOINS

- In SQL:1999, the join of two tables returning only matched rows is called an **inner join**. Inner Join consists of :
 - Natural join
 - Join with On clause
 - Join with Using clause
 - Equijoin (Old-style join)
- A join between two tables that returns the results of an inner join as well as the results of unmatched rows is **outer join**.
 - Left outer join
 - Right outer join
 - Full outer join

3. CREATING NATURAL JOINS

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- Natural join will perform the following tasks:
 - ตรวจสอบ common attribute โดยมองหา attribute ที่ชื่อเหมือนกันและมีชนิดข้อมูลเดียวกัน (ปกติคือ foreign key)
 - เลือกเฉพาะจากทั้งสองตารางที่มีค่าเหมือนกันในทุกคอลัมน์ที่เป็น common attribute
 - เชื่อมตารางโดยเลือกเฉพาะที่มีค่าเหมือนกันของattributeที่ชื่อเดียวกัน (common attribute)
 - เงื่อนไขการเชื่อมตารางสำหรับ Natural join คือการทำ equijoin ของทุกคอลัมน์ด้วยชื่อคอลัมน์ที่เหมือนกัน
 - If there are no common attributes, return the relational product of the two tables.
 - If the columns having the same names have different data types, an error is returned.

3. CREATING NATURAL JOINS

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- NATURAL JOIN is structured in such a way that, columns with the same name of associate tables will appear once only.
- เนื่องจากการเชื่อมตารางสำหรับ Natural join คือการทำ equijoin ของทุกคอลัมน์ด้วย ชื่อคอลัมน์ที่เหมือนกัน

```
SELECT      *|[column_name]  
FROM        table1_name  
NATURAL JOIN table2_name ;
```

EXAMPLE : NATURAL JOIN

Query: Find all customers with their agent code and agent phone.

```
SELECT *  
FROM CUSTOMER  
NATURAL JOIN AGENT ;
```

Table name: CUSTOMER

| | CUS_CODE | CUS_LNAME | CUS_ZIP | AGENT_CODE |
|---|----------|-----------|---------|------------|
| ▶ | 1132445 | Walker | 32145 | 231 |
| | 1217782 | Adares | 32145 | 125 |
| | 1312243 | Rakowski | 34129 | 167 |
| | 1321242 | Rodriguez | 37134 | 125 |
| | 1542311 | Smithson | 37134 | 421 |
| | 1657399 | Vanloo | 32145 | 231 |

Table name: AGENT

| | AGENT_CODE | AGENT_PHONE |
|---|------------|-------------|
| ▶ | 125 | 6152439887 |
| | 167 | 6153426778 |
| | 231 | 6152431124 |
| | 333 | 9041234445 |

EXAMPLE : NATURAL JOIN

Table name: CUSTOMER

| | CUS_CODE | CUS_LNAME | CUS_ZIP | AGENT_CODE |
|---|----------|-----------|---------|------------|
| ► | 1132445 | Walker | 32145 | 231 |
| | 1217782 | Adares | 32145 | 125 |
| | 1312243 | Rakowski | 34129 | 167 |
| | 1321242 | Rodriguez | 37134 | 125 |
| | 1542311 | Smithson | 37134 | 421 |
| | 1657399 | Vanloo | 32145 | 231 |

Table name: AGENT

| | AGENT_CODE | AGENT_PHONE |
|---|------------|-------------|
| ► | 125 | 6152439887 |
| | 167 | 6153426778 |
| | 231 | 6152431124 |
| | 333 | 9041234445 |

Step 1: Cartesian Product

| | CUS_CODE | CUS_LNAME | CUS_ZIP | CUSTOMER.AGENT_CODE | AGENT.AGENT_CODE | AGENT_PHONE |
|---|----------|-----------|---------|---------------------|------------------|-------------|
| ► | 1132445 | Walker | 32145 | 231 | 125 | 6152439887 |
| | 1132445 | Walker | 32145 | 231 | 167 | 6153426778 |
| | 1132445 | Walker | 32145 | 231 | 231 | 6152431124 |
| | 1132445 | Walker | 32145 | 231 | 333 | 9041234445 |
| | 1217782 | Adares | 32145 | 125 | 125 | 6152439887 |
| | 1217782 | Adares | 32145 | 125 | 167 | 6153426778 |
| | 1217782 | Adares | 32145 | 125 | 231 | 6152431124 |
| | 1217782 | Adares | 32145 | 125 | 333 | 9041234445 |
| | 1312243 | Rakowski | 34129 | 167 | 125 | 6152439887 |
| | 1312243 | Rakowski | 34129 | 167 | 167 | 6153426778 |
| | 1312243 | Rakowski | 34129 | 167 | 231 | 6152431124 |
| | 1312243 | Rakowski | 34129 | 167 | 333 | 9041234445 |
| | 1321242 | Rodriguez | 37134 | 125 | 125 | 6152439887 |
| | 1321242 | Rodriguez | 37134 | 125 | 167 | 6153426778 |
| | 1321242 | Rodriguez | 37134 | 125 | 231 | 6152431124 |
| | 1321242 | Rodriguez | 37134 | 125 | 333 | 9041234445 |
| | 1542311 | Smithson | 37134 | 421 | 125 | 6152439887 |
| | 1542311 | Smithson | 37134 | 421 | 167 | 6153426778 |
| | 1542311 | Smithson | 37134 | 421 | 231 | 6152431124 |
| | 1542311 | Smithson | 37134 | 421 | 333 | 9041234445 |
| | 1657399 | Vanloo | 32145 | 231 | 125 | 6152439887 |
| | 1657399 | Vanloo | 32145 | 231 | 167 | 6153426778 |
| | 1657399 | Vanloo | 32145 | 231 | 231 | 6152431124 |
| | 1657399 | Vanloo | 32145 | 231 | 333 | 9041234445 |

EXAMPLE : NATURAL JOIN

Step 2 : Select Join attribute AGENT_CODE values are equal

| | CUS_CODE | CUS_LNAME | CUS_ZIP | CUSTOMER.AGENT_CODE | AGENT.AGENT_CODE | AGENT_PHONE |
|---|----------|-----------|---------|---------------------|------------------|-------------|
| ▶ | 1132445 | Walker | 32145 | 231 | 125 | 6152439887 |
| | 1132445 | Walker | 32145 | 231 | 167 | 6153426778 |
| | 1132445 | Walker | 32145 | 231 | 231 | 6152431124 |
| | 1132445 | Walker | 32145 | 231 | 333 | 9041234445 |
| | 1217782 | Adares | 32145 | 125 | 125 | 6152439887 |
| | 1217782 | Adares | 32145 | 125 | 167 | 6153426778 |
| | 1217782 | Adares | 32145 | 125 | 231 | 6152431124 |
| | 1217782 | Adares | 32145 | 125 | 333 | 9041234445 |
| | 1312243 | Rakowski | 34129 | 167 | 125 | 6152439887 |
| | 1312243 | Rakowski | 34129 | 167 | 167 | 6153426778 |
| | 1312243 | Rakowski | 34129 | 167 | 231 | 6152431124 |
| | 1312243 | Rakowski | 34129 | 167 | 333 | 9041234445 |
| | 1321242 | Rodriguez | 37134 | 125 | 125 | 6152439887 |
| | 1321242 | Rodriguez | 37134 | 125 | 167 | 6153426778 |
| | 1321242 | Rodriguez | 37134 | 125 | 231 | 6152431124 |
| | 1321242 | Rodriguez | 37134 | 125 | 333 | 9041234445 |
| | 1542311 | Smithson | 37134 | 421 | 125 | 6152439887 |
| | 1542311 | Smithson | 37134 | 421 | 167 | 6153426778 |
| | 1542311 | Smithson | 37134 | 421 | 231 | 6152431124 |
| | 1542311 | Smithson | 37134 | 421 | 333 | 9041234445 |
| | 1657399 | Vanloo | 32145 | 231 | 125 | 6152439887 |
| | 1657399 | Vanloo | 32145 | 231 | 167 | 6153426778 |
| | 1657399 | Vanloo | 32145 | 231 | 231 | 6152431124 |
| | 1657399 | Vanloo | 32145 | 231 | 333 | 9041234445 |

| | CUS_CODE | CUS_LNAME | CUS_ZIP | CUSTOMER.AGENT_CODE | AGENT.AGENT_CODE | AGENT_PHONE |
|---|----------|-----------|---------|---------------------|------------------|-------------|
| ▶ | 1217782 | Adares | 32145 | 125 | 125 | 6152439887 |
| | 1321242 | Rodriguez | 37134 | 125 | 125 | 6152439887 |
| | 1312243 | Rakowski | 34129 | 167 | 167 | 6153426778 |
| | 1132445 | Walker | 32145 | 231 | 231 | 6152431124 |
| | 1657399 | Vanloo | 32145 | 231 | 231 | 6152431124 |

Example : Natural Join

Step 3 : use a Projection to eliminate the duplicate attributes

| | CUS_CODE | CUS_LNAME | CUS_ZIP | CUSTOMER.AGENT_CODE | AGENT.AGENT_CODE | AGENT_PHONE |
|---|----------|-----------|---------|---------------------|------------------|-------------|
| ▶ | 1217782 | Adares | 32145 | 125 | 125 | 6152439887 |
| | 1321242 | Rodriguez | 37134 | 125 | 125 | 6152439887 |
| | 1312243 | Rakowski | 34129 | 167 | 167 | 6153426778 |
| | 1132445 | Walker | 32145 | 231 | 231 | 6152431124 |
| | 1657399 | Vanloo | 32145 | 231 | 231 | 6152431124 |

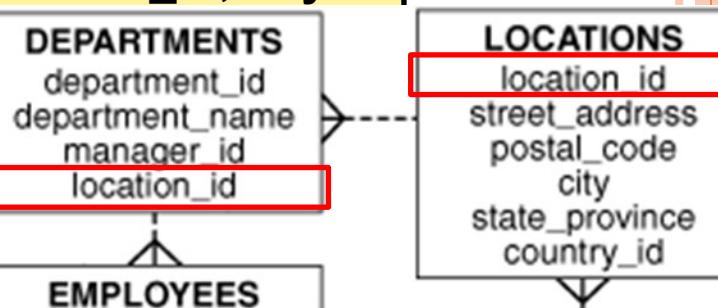


| | CUS_CODE | CUS_LNAME | CUS_ZIP | AGENT_CODE | AGENT_PHONE |
|---|----------|-----------|---------|------------|-------------|
| ▶ | 1217782 | Adares | 32145 | 125 | 6152439887 |
| | 1321242 | Rodriguez | 37134 | 125 | 6152439887 |
| | 1312243 | Rakowski | 34129 | 167 | 6153426778 |
| | 1132445 | Walker | 32145 | 231 | 6152431124 |
| | 1657399 | Vanloo | 32145 | 231 | 6152431124 |

3. RETRIEVING RECORDS WITH NATURAL JOINS (2 TABLES)

```
SELECT department_id, department_name, location_id, city  
FROM departments  
NATURAL JOIN locations ;
```

departments
locations ;



| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY |
|---------------|----------------------|-------------|---------------------|
| 60 | IT | 1400 | Southlake |
| 50 | Shipping | 1500 | South San Francisco |
| 10 | Administration | 1700 | Seattle |
| 30 | Purchasing | 1700 | Seattle |
| 90 | Executive | 1700 | Seattle |
| 100 | Finance | 1700 | Seattle |
| 110 | Accounting | 1700 | Seattle |
| 120 | Treasury | 1700 | Seattle |
| 130 | Corporate Tax | 1700 | Seattle |
| 140 | Control And Credit | 1700 | Seattle |
| 150 | Shareholder Services | 1700 | Seattle |
| 160 | Benefits | 1700 | Seattle |
| 170 | Manufacturing | 1700 | Seattle |
| 180 | Construction | 1700 | Seattle |
| 190 | Contracting | 1700 | Seattle |
| 200 | Operations | 1700 | Seattle |
| 210 | IT Support | 1700 | Seattle |
| 220 | NOC | 1700 | Seattle |
| 230 | IT Helpdesk | 1700 | Seattle |
| 240 | Government Sales | 1700 | Seattle |
| 250 | Retail Sales | 1700 | Seattle |
| 260 | Recruiting | 1700 | Seattle |
| 270 | Payroll | 1700 | Seattle |
| 20 | Marketing | 1800 | Toronto |
| 40 | Human Resources | 2400 | London |
| 80 | Sales | 2500 | Oxford |
| 70 | Public Relations | 2700 | Munich |

27 rows selected.

Note: number of rows equal to number of rows from the common attribute (location_id) between DEPARTMENTS and LOCATIONS.

3. NATURAL JOINS (2 TABLES) WITH 2 COMMON ATTRIBUTES

DEPARTMENTS
department_id
department_name
manager_id
location_id



EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id

**SELECT employee_id, last_name, department_id,
manager_id, location_id**

FROM

NATURAL JOIN

employees

departments ;

| employee_id | last_name | department_id | manager_id | location_id |
|-------------|------------|---------------|------------|-------------|
| 202 | Fay | 20 | 201 | 1800 |
| 115 | Khoo | 30 | 114 | 1700 |
| 116 | Baida | 30 | 114 | 1700 |
| 117 | Tobias | 30 | 114 | 1700 |
| 118 | Himuro | 30 | 114 | 1700 |
| 119 | Colmenares | 30 | 114 | 1700 |
| 129 | Bissot | 50 | 121 | 1500 |
| 130 | Atkinson | 50 | 121 | 1500 |
| 131 | Marlow | 50 | 121 | 1500 |
| 132 | Olson | 50 | 121 | 1500 |

SELECT employee_id, last_name, employees.department_id, employees.manager_id, location_id

FROM employees,departments

WHERE employees.manager_id=departments.manager_id

AND employees.department_id=departments.department_id

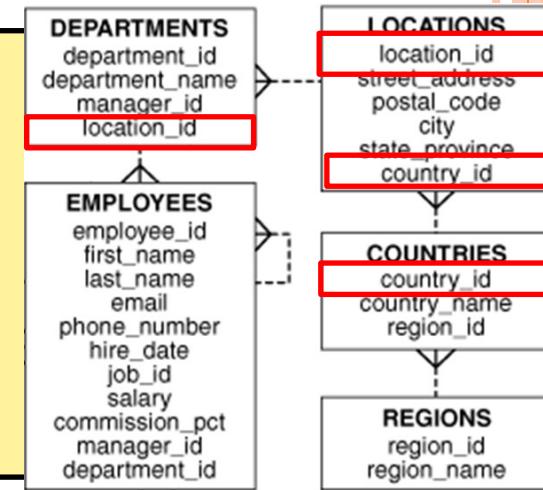
Note: number of rows equal to number of rows from EMPLOYEES and DEPARTMENTS that have equal values in all matched columns (department_id, manager_id).

3. RETRIEVING RECORDS WITH NATURAL JOINS (3 TABLES)

**SELECT department_id, department_name, location_id,
city, country_name**

FROM
NATURAL JOIN
NATURAL JOIN

departments
locations
countries ;



| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY | COUNTRY_NAME |
|---------------|----------------------|-------------|---------------------|--------------------------|
| 60 | IT | 1400 | Southlake | United States of America |
| 50 | Shipping | 1500 | South San Francisco | United States of America |
| 10 | Administration | 1700 | Seattle | United States of America |
| 30 | Purchasing | 1700 | Seattle | United States of America |
| 90 | Executive | 1700 | Seattle | United States of America |
| 100 | Finance | 1700 | Seattle | United States of America |
| 110 | Accounting | 1700 | Seattle | United States of America |
| 120 | Treasury | 1700 | Seattle | United States of America |
| 130 | Corporate Tax | 1700 | Seattle | United States of America |
| 140 | Control And Credit | 1700 | Seattle | United States of America |
| 150 | Shareholder Services | 1700 | Seattle | United States of America |
| 160 | Benefits | 1700 | Seattle | United States of America |
| 170 | Manufacturing | 1700 | Seattle | United States of America |
| 180 | Construction | 1700 | Seattle | United States of America |
| 190 | Contracting | 1700 | Seattle | United States of America |
| 200 | Operations | 1700 | Seattle | United States of America |
| 210 | IT Support | 1700 | Seattle | United States of America |
| 220 | NOC | 1700 | Seattle | United States of America |
| 230 | IT Helpdesk | 1700 | Seattle | United States of America |
| 240 | Government Sales | 1700 | Seattle | United States of America |
| 250 | Retail Sales | 1700 | Seattle | United States of America |
| 260 | Recruiting | 1700 | Seattle | United States of America |
| 270 | Payroll | 1700 | Seattle | United States of America |
| 20 | Marketing | 1800 | Toronto | Canada |
| 40 | Human Resources | 2400 | London | United Kingdom |
| 80 | Sales | 2500 | Oxford | United Kingdom |
| 70 | Public Relations | 2700 | Munich | Germany |

27 rows selected.

Note: number of rows equal to number of rows from the common attribute (location_id) between DEPARTMENTS and LOCATIONS.

3. NATURAL JOINS WITH A WHERE CLAUSE

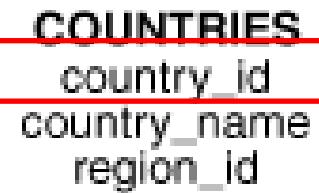
```
SELECT department_id, department_name, location_id, city  
FROM departments  
NATURAL JOIN locations  
WHERE department_id IN (20, 50) ;
```

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY |
|---------------|-----------------|-------------|---------------------|
| 20 | Marketing | 1800 | Toronto |
| 50 | Shipping | 1500 | South San Francisco |

EXERCISE # 2

1. จงเขียน Query เพื่อแสดง รหัสที่ตั้ง, ชื่อถนนและที่อยู่, เมือง, รัฐ/จังหวัด และ ชื่อประเทศ จากตาราง locations และ countries (ใช้ NATURAL JOIN)

```
SELECT      location_id,    street_address,  
            city,        state_province,   country_name  
  
FROM        countries  
  
NATURAL JOIN locations ;
```



4. CREATING JOINS WITH THE ON CLAUSE

- ถ้าใช้ join โดยไม่มี on คือการทำ Cartesian product หรือ cross join
- แต่ใช้ ON clause เพื่อระบุเงื่อนไขอื่นหรือระบุคอลัมน์ที่มีชื่อต่างกันเพื่อเชื่อมตาราง
- ON clause จำเป็นต้องมี ชื่อตารางหรือนามแฝงของตาราง สำหรับ common attribute

```
SELECT *
FROM   table1_name นามแฝง1
JOIN   table2_name นามแฝง2
ON     (นามแฝง1.column_name = นามแฝง2.column_name) ;
```

= SELECT *
FROM table1_name t1, table2_name t2;

4. RETRIEVING RECORDS WITH THE ON CLAUSE

```
SELECT e.employee_id, e.last_name, e.department_id, d.location_id  
FROM employees e  
JOIN departments d  
ON (e.department_id = d.department_id);
```

| employee_id | last_name | department_id | location_id |
|-------------|-----------|---------------|-------------|
| 103 | Hunold | 60 | 1400 |
| 104 | Ernst | 60 | 1400 |
| 105 | Austin | 60 | 1400 |
| 106 | Pataballa | 60 | 1400 |
| 107 | Lorentz | 60 | 1400 |
| 120 | Weiss | 50 | 1500 |
| 121 | Fripp | 50 | 1500 |
| 122 | Kaufling | 50 | 1500 |
| 123 | Vollman | 50 | 1500 |
| 124 | Mourgos | 50 | 1500 |

4. RETRIEVING RECORDS WITH THE ON CLAUSE

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.manager_id  
FROM   employees e  
JOIN   departments d  
ON     (e.employee_id < d.manager_id) ;
```

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | MANAGER_ID |
|-------------|-----------|---------------|------------|
| 100 | King | 90 | 103 |
| 100 | King | 90 | 108 |
| 100 | King | 90 | 114 |
| 100 | King | 90 | 121 |
| 100 | King | 90 | 145 |
| 100 | King | 90 | 200 |
| 100 | King | 90 | 201 |
| 100 | King | 90 | 203 |
| 100 | King | 90 | 204 |
| 100 | King | 90 | 205 |
| 101 | Kochhar | 90 | 103 |
| 101 | Kochhar | 90 | 108 |
| 101 | Kochhar | 90 | 114 |
| 101 | Kochhar | 90 | 121 |
| 101 | Kochhar | 90 | 145 |
| 101 | Kochhar | 90 | 200 |
| 101 | Kochhar | 90 | 201 |
| 101 | Kochhar | 90 | 203 |

APPLYING ADDITIONAL CONDITIONS TO A JOIN

- displays only employees who have a manager ID of 149.

```
SELECT      e.employee_id, e.last_name, e.department_id, d.location_id  
FROM        employees e  
JOIN        departments d  
ON          (e.department_id = d.department_id)  
AND         e.manager_id = 149;
```

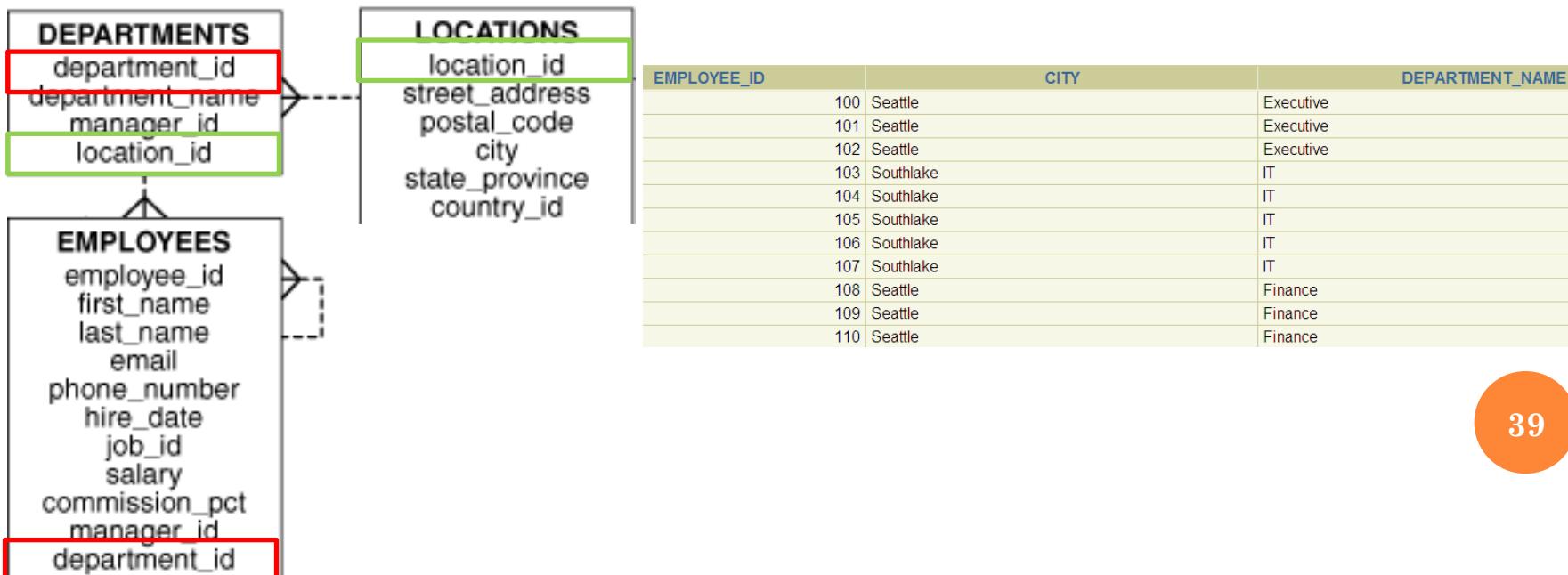
| employee_id | last_name | department_id | location_id |
|-------------|------------|---------------|-------------|
| 174 | Abel | 80 | 2500 |
| 175 | Hutton | 80 | 2500 |
| 176 | Taylor | 80 | 2500 |
| 177 | Livingston | 80 | 2500 |
| 179 | Johnson | 80 | 2500 |

Remark: You can use a WHERE clause instead of AND clause

Showing 1 to 5 of 5 entries

CREATING THREE-WAY JOINS WITH THE ON CLAUSE

```
SELECT      employee_id, city, department_name  
FROM        employees e  
JOIN        departments d  
ON          (e.department_id = d.department_id)  
JOIN        locations l  
ON          (d.location_id = l.location_id) ;
```



EXERCISE # 3 –JOIN ON CLAUSE

จงเขียน Query เพื่อแสดง รหัสที่ตั้ง, ชื่อถนนและที่อยู่, เมือง, รัฐ/จังหวัด และ ชื่อประเทศ โดยแสดงเฉพาะผลลัพธ์ที่ตั้งอยู่ที่ประเทศนีลำดับมา ก่อน India และเรียงลำดับตามชื่อประเทศจาก A-Z

```
SELECT      location_id, street_address, city,  
            state_province, country_name  
  
FROM        locations l  
  
JOIN        countries c  
  
ON          (l.country_id = c.country_id)  
  
WHERE       c.country_name < 'India'  
  
ORDER BY    country_name ASC;
```

LOCATIONS

location_id
street_address
postal_code
city
state_province
country_id

COUNTRIES

country_id
country_name
region_id

5. CREATING JOINS WITH THE USING CLAUSE

- สำหรับ Natural Join, เชื่อมตารางโดยเลือก เฉพาะแ眷ที่มีค่าเหมือนกันของ attribute ที่ซื่อเดียวกัน (common attributes)
- ถ้าการเชื่อมตาราง มี common attributes มากกว่า 1 ตัว สามารถใช้ join โดยเพิ่ม USING clause เพื่อบรุคคลัมณ์ที่ต้องการเชื่อมกันเท่านั้น
- สรุป ใช้ USING clause เพื่อจับคู่เพียง 1 คอลัมน์เมื่อมี common attribute มากกว่า 1 ห้ามใช้ นามแฝงของตาราง

```
SELECT *
  FROM table1_name
 JOIN table2_name
 USING (common attribute) ;
```

5. RETRIEVING RECORDS WITH THE USING CLAUSE

```
SELECT employee_id, last_name,  
       location_id, department_id  
  FROM employees  
 JOIN departments  
 USING (department_id) ;
```



| EMPLOYEE_ID | LAST_NAME | LOCATION_ID | DEPARTMENT_ID |
|-------------|------------|-------------|---------------|
| 200 | Whalen | 1700 | 10 |
| 201 | Hartstein | 1800 | 20 |
| 202 | Fay | 1800 | 20 |
| 114 | Raphaely | 1700 | 30 |
| 115 | Khoo | 1700 | 30 |
| 116 | Baida | 1700 | 30 |
| 119 | Colmenares | 1700 | 30 |
| 118 | Himuro | 1700 | 30 |
| 117 | Tobias | 1700 | 30 |
| 203 | Mavris | 2400 | 40 |
| 120 | Weiss | 1500 | 50 |
| 121 | Fripp | 1500 | 50 |
| 123 | Vollman | 1500 | 50 |
| 132 | Olson | 1500 | 50 |
| 131 | Marlow | 1500 | 50 |

106 row selected

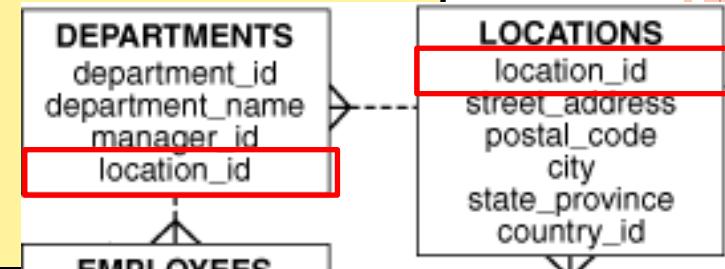
5. RETRIEVING RECORDS WITH THE USING CLAUSE

```
SELECT department_id, department_name, location_id, city
```

```
FROM locations
```

```
JOIN departments
```

```
USING (location_id) ;
```



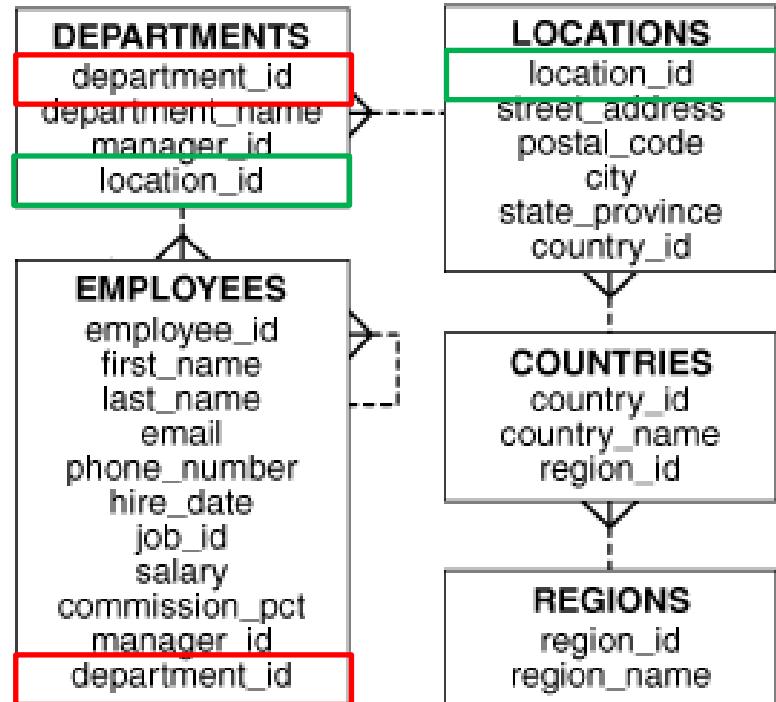
| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY |
|---------------|----------------------|-------------|---------------------|
| 60 | IT | 1400 | Southlake |
| 50 | Shipping | 1500 | South San Francisco |
| 10 | Administration | 1700 | Seattle |
| 30 | Purchasing | 1700 | Seattle |
| 90 | Executive | 1700 | Seattle |
| 100 | Finance | 1700 | Seattle |
| 110 | Accounting | 1700 | Seattle |
| 120 | Treasury | 1700 | Seattle |
| 130 | Corporate Tax | 1700 | Seattle |
| 140 | Control And Credit | 1700 | Seattle |
| 150 | Shareholder Services | 1700 | Seattle |
| 160 | Benefits | 1700 | Seattle |
| 170 | Manufacturing | 1700 | Seattle |
| 180 | Construction | 1700 | Seattle |
| 190 | Contracting | 1700 | Seattle |
| 200 | Operations | 1700 | Seattle |
| 210 | IT Support | 1700 | Seattle |
| 220 | NOC | 1700 | Seattle |
| 230 | IT Helpdesk | 1700 | Seattle |
| 240 | Government Sales | 1700 | Seattle |
| 250 | Retail Sales | 1700 | Seattle |
| 260 | Recruiting | 1700 | Seattle |
| 270 | Payroll | 1700 | Seattle |
| 20 | Marketing | 1800 | Toronto |
| 40 | Human Resources | 2400 | London |
| 80 | Sales | 2500 | Oxford |
| 70 | Public Relations | 2700 | Munich |

27 row selected

EXERCISE # 4 JOIN- USING CLAUSE

จงเขียน SQL Query ที่แสดงชื่อ นามสกุล รหัสงาน
รหัสแผนก และชื่อแผนกของพนักงานทุกคนที่
ทำงานในเมือง多伦多 (Toronto) (ใช้ USING
clause)

```
SELECT      first_name, last_name, job_id,  
            department_id, department_name  
FROM        employees  
JOIN        departments  
USING      (department_id)  
JOIN        locations  
USING      (location_id)  
WHERE       city = 'Toronto' ;
```



6. SELF-JOINS USING THE ON CLAUSE

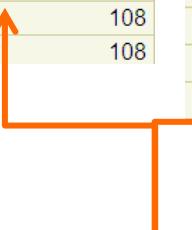
- ต้องการทราบชื่อผู้จัดการของพนักงานแต่ละคน

EMPLOYEES

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | MANAGER_ID |
|-------------|------------|-----------|------------|
| 100 | Steven | King | |
| 101 | Neena | Kochhar | 100 |
| 102 | Lex | De Haan | 100 |
| 103 | Alexander | Hunold | 102 |
| 104 | Bruce | Ernst | 103 |
| 105 | David | Austin | 103 |
| 106 | Valli | Pataballa | 103 |
| 107 | Diana | Lorentz | 103 |
| 108 | Nancy | Greenberg | 101 |
| 109 | Daniel | Faviet | 108 |
| 110 | John | Chen | 108 |

EMPLOYEES (MANAGER)

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME |
|-------------|------------|-----------|
| 100 | Steven | King |
| 101 | Neena | Kochhar |
| 102 | Lex | De Haan |
| 103 | Alexander | Hunold |
| 104 | Bruce | Ernst |
| 105 | David | Austin |
| 106 | Valli | Pataballa |
| 107 | Diana | Lorentz |
| 108 | Nancy | Greenberg |
| 109 | Daniel | Faviet |
| 110 | John | Chen |



```
SELECT attribute1, attribute2  
FROM employees e  
JOIN employees m  
ON (e.manager_id = m.employee_id) ;
```

MANAGER_ID in the EMPLOYEES table is equal to EMPLOYEES_ID in the MANAGER table.

EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id

6. SELF-JOINS USING THE ON CLAUSE

- เพื่อหาชื่อผู้จัดการของพนักงานแต่ละคน จะเป็นต้องทำการเชื่อมตาราง EMPLOYEE กับตัวมันเอง เรียกว่า SELF JOIN

```
SELECT e.last_name emp, m.last_name mgr
FROM employees e
JOIN employees m
ON (e.manager_id = m.employee_id) ;
```

MANAGER_ID in the EMPLOYEES table is equal to
EMPLOYEES _ID in the MANAGER table.

| EMP | MGR |
|-----------|---------|
| Hartstein | King |
| Zlotkey | King |
| Cambrault | King |
| Errazuriz | King |
| Partners | King |
| Russell | King |
| Mourgos | King |
| Vollman | King |
| Kaufling | King |
| Fripp | King |
| Weiss | King |
| Raphaely | King |
| De Haan | King |
| Kochhar | King |
| Higgins | Kochhar |

| EMPLOYEES |
|----------------|
| employee_id |
| first_name |
| last_name |
| email |
| phone_number |
| hire_date |
| job_id |
| salary |
| commission_pct |
| manager_id |
| department_id |

EXERCISE # 5 SELF-JOIN

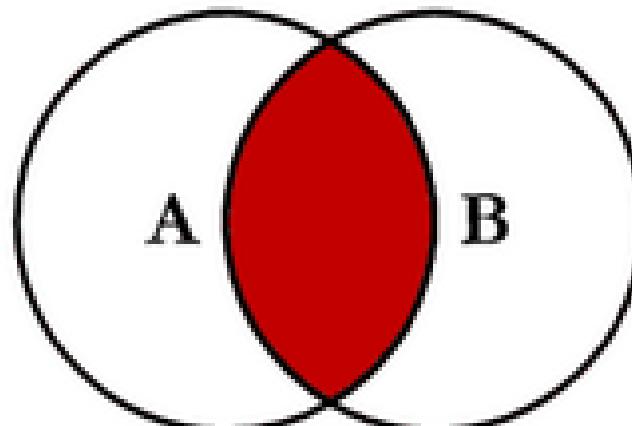
จงแสดงชื่อ นามสกุล วันที่เริ่มทำงานของพนักงาน พร้อมกับ ชื่อ นามสกุล วันที่เริ่มทำงานของผู้จัดการของพนักงานคนนั้นๆ (ตั้งชื่อ คอลัมน์ของผู้จัดการคือ Mgr First Name, Mgr Last Name, Mgr Hired)

แสดงเฉพาะผลลัพธ์ที่มีวันเริ่มทำงานของพนักงาน เริ่มก่อนวันเริ่ม ทำงานของผู้จัดการของตนเอง

```

SELECT e.first_name , e.last_name, e.hire_date,
       m.first_name 'Mgr First Name',
       m.last_name 'Mgr Last Name', m.hire_date 'Mgr Hired'
FROM employees e
JOIN employees m
ON (e.manager_id = m.employee_id)
Where e.hire_date < m.hire_date;
    
```

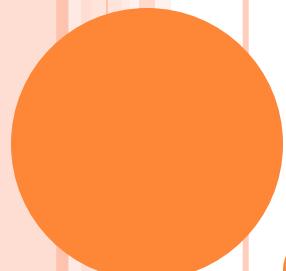
SQL JOINS



```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

INTRODUCTION TO SQL

DISPLAYING DATA FROM MULTIPLE TABLES (OUTER JOIN)

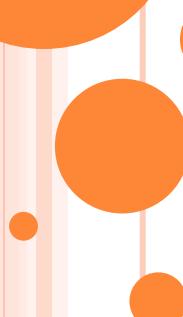


By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts



EXAMPLE: JOIN WITH ON CLAUSE

Query: Find all orders with their product names and price.

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

```
SELECT *
FROM ORDER
Join PRODUCT
On (Order.P_CODE = Product.P_CODE);
```

RECALL : JOIN ORDER

```
Select      *
From       PRODUCT
JOIN        ORDER
USING      (P_CODE) ;
```

PRODUCT

| PRODUCT.P_CODE | P_NAME | PRICE |
|----------------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

Step 1: Cartesian Product

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 222220 | คอมพิวเตอร์ | 30000 |
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 1 | C001 | 111110 | 333330 | ปากกา | 500 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |
| 2 | C002 | 222220 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 333330 | ปากกา | 500 |

Automatically
rename

RECALL : JOIN

Step 2 : Select Join attribute P_CODE values are equal

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 222220 | คอมพิวเตอร์ | 30000 |
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 1 | C001 | 111110 | 333330 | ปากกา | 500 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |
| 2 | C002 | 222220 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 333330 | ปากกา | 500 |



| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |

RECALL : JOIN

Step 3 : use a Projection to eliminate the duplicate attributes

| ORDER_NO | CUSTOMER_NO | ORDER.P_CODE | PRODUCT.P_CODE | P_NAME | PRICE |
|----------|-------------|--------------|----------------|-------------|-------|
| 1 | C001 | 111110 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | 222220 | คอมพิวเตอร์ | 30000 |

```
Select      *
From       PRODUCT
JOIN        ORDER
USING      (P_CODE);
```

| ORDER_NO | CUSTOMER_NO | P_CODE | P_NAME | PRICE |
|----------|-------------|--------|-------------|-------|
| 1 | C001 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | คอมพิวเตอร์ | 30000 |

7. OUTER JOINS

- สามารถแสดงข้อมูลແລວที่ไม่สามารถจับคู่เงื่อนไขเท่ากันกับอีกตาราง ซึ่งปกติจะไม่แสดงเวลาใช้ join

EMPLOYEES

| LAST_NAME | LAST_NAME | FIRST_NAME | DEPARTMENT_ID |
|------------|------------|------------|---------------|
| King | King | Steven | 90 |
| Kochhar | Kochhar | Neena | 90 |
| De Haan | De Haan | Lex | 90 |
| Hunold | Hunold | Alexander | 60 |
| Ernst | Ernst | Bruce | 60 |
| Taylor | Taylor | Jonathon | 80 |
| Livingston | Livingston | Jack | 80 |
| Grant | Grant | Kimberely | |
| Johnson | Johnson | Charles | 80 |
| Taylor | Taylor | Winston | 50 |
| Fleaur | Fleaur | Jean | 50 |
| Sullivan | Sullivan | Martha | 50 |
| Geoni | Geoni | Girard | 50 |

107 row selected

DEPARTMENTS

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|------------|---------------|-----------------|
| Whalen | 10 | Administration |
| Hartstein | 20 | Marketing |
| Fay | 20 | Marketing |
| Raphaely | 30 | Purchasing |
| Khoo | 30 | Purchasing |
| Baida | 30 | Purchasing |
| Colmenares | 30 | Purchasing |

106 row selected

UNMATCHED ROWS VERSUS MATCHED ROWS

- To return the unmatched rows, you can use an outer join
 - There are three types of outer joins:
 - FULL OUTER
 - LEFT OUTER
 - RIGHT OUTER
- Reminder: Joining tables with the NATURAL JOIN, USING or ON clauses results in matched rows displayed in the output.

7. OUTER JOINS

- A join between two tables that returns the results of the inner join as well as
 - Full outer join: เป็นการ join ที่ขยายมาจากการ join-using คือ สามารถแสดงข้อมูลเฉพาะที่ สามารถและไม่สามารถจับคู่เงื่อนไขเท่ากันกับอีกตาราง ซึ่งปกติจะไม่แสดงเวลาใช้ join-using
 - Left outer join: แสดงเฉพาะในตารางที่อยู่ทางซ้ายมืออกมาหั้งหมด ถึงแม้จะมีเฉพาะในตารางซ้ายมีค่าของฟิลด์ที่ไม่ตรงกับฟิลด์เขื่อมของตารางทางขวา มือส่วนตารางทางขวา มือ ก็จะ return ค่าเป็น NULL สำหรับเฉพาะที่เงื่อนไขไม่ match กับตารางหลัก
 - Right outer join: แสดงเฉพาะในตารางที่อยู่ทางขวา มืออกมาหั้งหมด ถึงแม้จะมีเฉพาะในตารางขวา มีค่าของฟิลด์ที่ไม่ตรงกับฟิลด์เขื่อมของตารางทางซ้าย มือส่วนตารางทางซ้าย มือ ก็จะ return ค่าเป็น NULL สำหรับเฉพาะที่เงื่อนไขไม่ match กับตารางหลัก

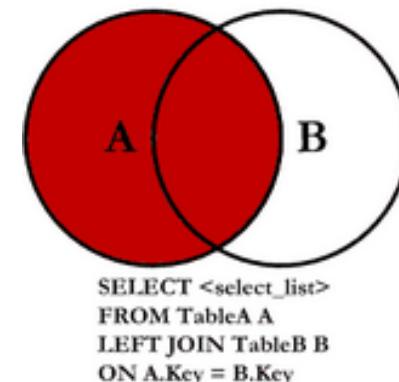
LEFT OUTER JOIN

```
SELECT      attribute  
FROM        A  
LEFT OUTER JOIN B  
USING      (attribute) ;
```

- LEFT OUTER JOIN ใช้สำหรับการ แสดงແຄວໃນຕາරາງທີ່ອຢູ່ທາງໜ້າຍມືອອກມາ ທັງໝົດ ຄື່ງແມ່ຈະມີແຄວໃນຕາරາງໜ້າຍມືອມີຄ່າຂອງຟິລດ໌ທີ່ໄມ່ຕຽງກັບຟິລດ໌ເຊື່ອມຂອງຕາරາງທາງໜ້າມືອ ສ່ວນຕາරາງທາງໜ້າມືອ ກີ່ຈະ return ດ້ວຍຄ່າເປັນ NULL ສໍາຮັບແຄວທີ່ເງື່ອນໄຂໄໝ່ match ກັບຕາරາງຫລັກ
- ຈະรวมຂໍ້ມູນທີ່ຕຽງກັນ ໂດຍຈະ ເນັ້ນຕາරາງຜົ່ງໜ້າຍເປັນຫລັກ
- Matched pairs are retained and any unmatched values in A table are retained too, but any unmatched values in B table are eliminated.

Result set

1. Matched values from result of Join-using
2. Unmatched values of attributes in A



EXAMPLE: LEFT OUTER JOIN

Query: Find all orders with their product names and price.

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

SELECT *

FROM ORDER

Left Outer Join PRODUCT

On (Order.P_CODE = Product.P_CODE) ;

EXAMPLE : LEFT OUTER JOIN

1. Yields **result from ORDER Join PRODUCT**
2. Plus **all rows in ORDER table, including those that do not have a matching value in the PRODUCT table**

```
SELECT ORDER_NO, CUSTOMER_NO,  
       P_CODE, P_NAME, PRICE  
FROM   ORDER  
LEFT OUTER JOIN PRODUCT  
On (Order.P_CODE = Product.P_CODE);
```

| ORDER_NO | CUSTOMER_NO | P_CODE | P_NAME | PRICE |
|----------|-------------|--------|-------------|-------|
| 1 | C001 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | คอมพิวเตอร์ | 30000 |

1. Matched values from result of ORDER \bowtie PRODUCT

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

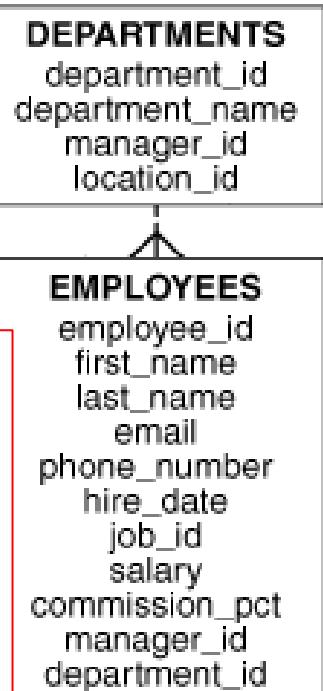
2. Unmatched values from ORDER (NOT FOUND)

EXAMPLE

1. จงเขียน SQL Query ที่แสดงข้อมูลพนักงานที่มีชื่อแผนกและที่ยังไม่มีแผนก

```
SELECT *  
FROM employees  
LEFT OUTER JOIN departments  
ON (employees.department_id= departments. department_id);
```

```
SELECT *  
FROM departments  
RIGHT OUTER JOIN employees  
ON (employees.department_id= departments. department_id);
```



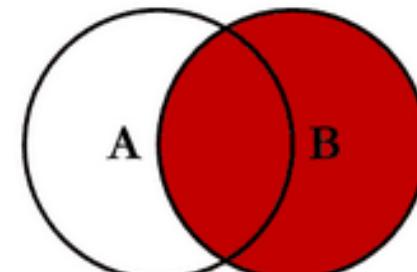
RIGHT OUTER JOIN

```
SELECT          attribute  
FROM           A  
RIGHT OUTER JOIN    B  
USING          (attribute) ;
```

- RIGHT OUTER JOIN ใช้สำหรับการแสดงผลในตารางที่อยู่ทางขวาไม้ออกมาหั้งหมด ถึงแม้จะมีเฉพาะในตารางขวา มีค่าของพิล์ด์ที่ไม่ตรงกับพิล์ด์ซึ่งมีอยู่ในตารางทางซ้ายมือ ส่วนตารางทางซ้ายมือ ก็จะ return ค่าเป็น NULL สำหรับเฉพาะที่เงื่อนไขไม่ match กับตารางหลัก
- จะรวมข้อมูลที่ตรงกัน โดยจะเน้นตารางฝั่งขวาเป็นหลัก
- Matched pairs are retained **and any unmatched values in B table** are retained too, but **any unmatched values in A table are eliminated.**

Result set

1. Matched values from result of Join-using
2. Unmatched values of attributes in B



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```

RIGHT OUTER JOIN

นำไปประยุกต์ตอบโจทย์ >> แสดงรหัส ชื่อสินค้าและราคาน้ำที่ไม่มีการสั่งซื้อจากลูกค้า

1. Yields **result from ORDER join PRODUCT**
2. Plus all rows in **PRODUCT** table, including those that do not have a matching value in the ORDER table

```
SELECT ORDER_NO, CUSTOMER_NO,  
P_CODE, P_NAME, PRICE  
FROM ORDER  
RIGHT OUTER JOIN PRODUCT  
On (Order.P_CODE = Product.P_CODE);
```

| ORDER_NO | CUSTOMER_NO | P_CODE | P_NAME | PRICE |
|----------|-------------|--------|-------------|-------|
| 1 | C001 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | คอมพิวเตอร์ | 30000 |
| NULL | NULL | 333330 | ปากกา | 500 |

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

1. Matched values from result of ORDER \bowtie PRODUCT

2. Unmatched values from PRODUCT

จะทราบสินค้าที่ไม่ถูกสั่งซื้อ

EXERCISE

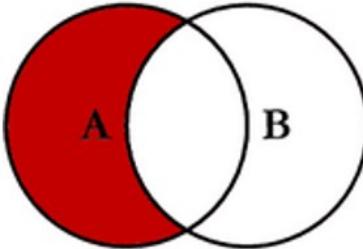
1. จงเขียน SQL Query ที่แสดงข้อมูลแผนกที่ยังไม่มีพนักงาน

```
SELECT *  
FROM departments  
LEFT OUTER JOIN employees  
ON (employees.department_id = departments.department_id)
```

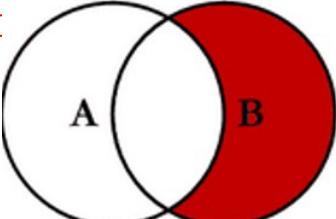
Where employees.department_id IS NULL;

```
SELECT *  
FROM employees  
RIGHT OUTER JOIN departments  
ON (employees.department_id = departments.department_id)
```

Where employees.department_id IS NULL;



```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```

FULL OUTER JOIN

```
SELECT          attribute  
FROM            A  
LEFT OUTER JOIN B  
USING          (attribute)  
UNION  
SELECT          attribute  
FROM            A  
RIGHT OUTER JOIN B  
USING          (attribute) ;
```

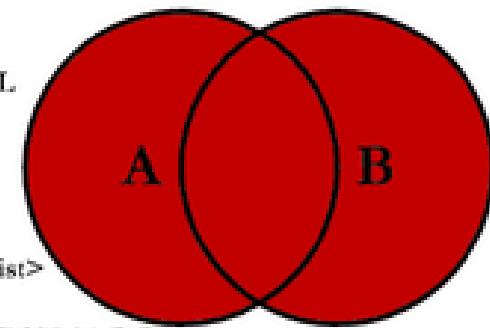
- เป็นการ join ที่ขยายมาจากการ join-using คือ สามารถแสดงข้อมูลแ雷ว์ที่ไม่สามารถจับคู่เงื่อนไขเท่ากันกับอีกตาราง ซึ่งปกติจะไม่แสดงเวลาใช้ join-using
- จะแสดงแ雷ว์ในตารางที่อยู่ทั้งทางซ้ายและขวา ของเงื่อนไขอุปกรณ์ทั้งหมด ตารางใดไม่ match กับเงื่อนไข ก็จะ return ค่าอุปกรณ์เป็น NULL
- Process of Outer Join is the same as that of Join-using, but the result of Outer Join is

Result-set

1. Matched values from result of Join-using
2. Unmatched values of attributes in A
3. Unmatched values of attributes in B

```
JOIN TableB B  
ON Key = B.Key  
WHERE B.Key IS NULL.
```

```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



EXAMPLE: FULL OUTER JOIN

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

Full Outer Join ใช้กับ Oracle
ถ้า MySQL ใช้ LEFT OUTER JOIN
UNION RIGHT OUTER JOIN

```
SELECT ORDER_NO, CUSTOMER_NO, P_CODE,  
P_NAME, PRICE  
FROM ORDER  
FULL OUTER JOIN PRODUCT  
On (Order.P_CODE = Product.P_CODE) ;
```

EXAMPLE: FULL OUTER JOIN

```

SELECT ORDER_NO, CUSTOMER_NO, P_CODE,
P_NAME, PRICE
FROM ORDER
FULL OUTER JOIN PRODUCT
On (Order.P_CODE = Product.P_CODE) ;

```

| ORDER_NO | CUSTOMER_NO | P_CODE | P_NAME | PRICE |
|----------|-------------|--------|-------------|-------|
| 1 | C001 | 111110 | สมุด | 120 |
| 2 | C002 | 222220 | คอมพิวเตอร์ | 30000 |
| NULL | NULL | 333330 | ปากกา | 500 |

1. Matched values from result of ORDER \bowtie PRODUCT

2. Unmatched values from PRODUCT

3. NOT FOUND
Unmatched values from ORDER

PRODUCT

| P_CODE | P_NAME | PRICE |
|--------|-------------|-------|
| 222220 | คอมพิวเตอร์ | 30000 |
| 111110 | สมุด | 120 |
| 333330 | ปากกา | 500 |

ORDER

| ORDER_NO | CUSTOMER_NO | P_CODE |
|----------|-------------|--------|
| 1 | C001 | 111110 |
| 2 | C002 | 222220 |

RECALL EXAMPLE: JOIN-USING

Note: EMPLOYEES and DEPARTMENTS tables have more than one column matches (department_id, manager_id). So, USING clause was used to match only one column

```
SELECT      last_name, department_id, department_name  
FROM        employees  
JOIN        departments  
ON          (employees.department_id = departments.department_id)
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|------------|---------------|-----------------|
| Whalen | 10 | Administration |
| Hartstein | 20 | Marketing |
| Fay | 20 | Marketing |
| Raphaely | 30 | Purchasing |
| Khoo | 30 | Purchasing |
| Baida | 30 | Purchasing |
| Colmenares | 30 | Purchasing |
| Himuro | 30 | Purchasing |
| Tobias | 30 | Purchasing |
| ... | | |
| Sciarra | 100 | Finance |
| Urman | 100 | Finance |
| Popp | 100 | Finance |
| Higgins | 110 | Accounting |
| Gietz | 110 | Accounting |

106 rows selected.

7.2 LEFT OUTER JOIN

```
SELECT employees.last_name, employees.department_id, department_name  
FROM employees  
LEFT OUTER JOIN departments  
ON (employees.department_id = departments.department_id)
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|------------|---------------|-----------------|
| Whalen | 10 | Administration |
| Fay | 20 | Marketing |
| Hartstein | 20 | Marketing |
| Colmenares | 30 | Purchasing |
| Himuro | 30 | Purchasing |
| ... | | |
| Faviet | 100 | Finance |
| Greenberg | 100 | Finance |
| Gietz | 110 | Accounting |
| Higgins | 110 | Accounting |
| Grant | | |

107 rows selected.

7.3 RIGHT OUTER JOIN

```
SELECT employees.last_name, employees.department_id, department_name  
FROM employees  
RIGHT OUTER JOIN departments  
ON (employees.department_id = departments.department_id)
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|----------------------|
| Whalen | 10 | Administration |
| ... | | |
| Gietz | 110 | Accounting |
| | | Treasury |
| | | Corporate Tax |
| | | Control And Credit |
| | | Shareholder Services |
| | | Benefits |
| | | Manufacturing |
| | | Construction |
| | | Contracting |
| | | Operations |
| | | IT Support |
| | | NOC |
| | | IT Helpdesk |
| | | Government Sales |
| | | Retail Sales |
| | | Recruiting |
| | | Payroll |

7.1 FULL OUTER JOIN (ORACLE)

```
SELECT          last_name, department_id, department_name  
FROM            employees  
FULL OUTER JOIN departments  
USING          (department_id);
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|--|
| Whalen | 10 | Administration |
| Higgins | 110 | Accounting |
| Grant | | Treasury Corporate Tax Control And Credit Shareholder Services Benefits Manufacturing Construction Contracting Operations IT Support NOC IT Helpdesk Government Sales Retail Sales Recruiting Payroll |
| | | 22 |

7.1 FULL OUTER JOIN

```
SELECT          last_name, department_id, department_name  
FROM            employees  
FULL OUTER JOIN departments  
USING          (department_id);
```

(ORACLE)

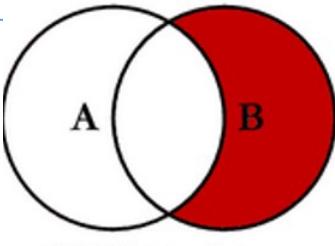
```
SELECT employees.last_name, employees.department_id, department_name  
FROM      employees  
LEFT OUTER JOIN departments  
ON        (employees.department_id = departments.department_id)  
UNION  
SELECT employees.last_name, departments.department_id, department_name  
FROM      employees  
RIGHT OUTER JOIN departments  
ON        (employees.department_id = departments.department_id);
```

(MySQL)

EXERCISE # 1

1. จงเขียน SQL Query ที่แสดงชื่อแผนกที่ยังไม่มีพนักงาน

```
SELECT department_name  
FROM employees  
RIGHT OUTER JOIN departments  
ON (employees.department_id = departments.department_id)  
WHERE employees.department_id IS NULL;
```

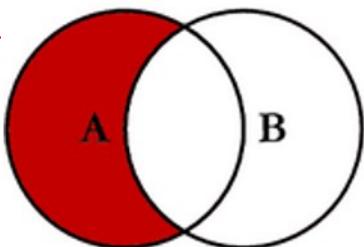


```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL.
```

| DEPARTMENTS |
|-----------------|
| department_id |
| department_name |
| manager_id |
| location_id |

| EMPLOYEES |
|----------------|
| employee_id |
| first_name |
| last_name |
| email |
| phone_number |
| hire_date |
| job_id |
| salary |
| commission_pct |
| manager_id |
| department_id |

```
SELECT department_name  
FROM departments  
LEFT OUTER JOIN employees  
ON (departments.department_id = employees.department_id)  
WHERE employees.department_id IS NULL;
```



```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL.
```

PRACTICE

- Assume we have two relations:

จงเขียน SQL query

- แสดงข้อมูลเมนูต่างๆที่ไม่ได้รับการสั่งจากลูกค้าคนใดเลย
- แสดงข้อมูลลูกค้าที่สั่งอาหารแต่ไม่มีอาหารนั้นในตารางเมนู
- แสดงข้อมูลลูกค้าพร้อมทั้งเมนูต่างๆที่ลูกค้าสั่งในเมนู
- แสดงข้อมูลลูกค้าพร้อมทั้งเมนูต่างๆที่ลูกค้าสั่งและข้อมูลอื่นๆ ที่ไม่สามารถจับคู่ตามเงื่อนไขได้

Menu

| Food | Day |
|-----------|-----------|
| Pizza | Monday |
| Hamburger | Tuesday |
| Chicken | Wednesday |
| Pasta | Thursday |
| Tacos | Friday |

Customers

| Name | Age | Food |
|-------|-----|-----------|
| Alice | 21 | Hamburger |
| Bill | 24 | Pizza |
| Carl | 23 | Beer |
| Dina | 19 | Shrimp |

*Reference: http://cir.dcs.uni-pannon.hu/cikkek/DB_relational_algebra_v2.pdf

PRACTICE 2

2. แสดงข้อมูลของเมนูทั้งหมดที่ไม่ได้รับการสั่งจากลูกค้าคนใดเลย

```
SELECT      Menu.food,    day
FROM        menu
LEFT OUTER JOIN customers
ON          (menu.food = customers.food)
WHERE       customers.food IS NULL ;
```

Menu

| Food | Day |
|-----------|-----------|
| Pizza | Monday |
| Hamburger | Tuesday |
| Chicken | Wednesday |
| Pasta | Thursday |
| Tacos | Friday |

```
SELECT      Menu.food,    day
FROM        customers
RIGHT OUTER JOIN menu
ON          (menu.food = customers.food)
WHERE       customers.food IS NULL ;
```

Customers

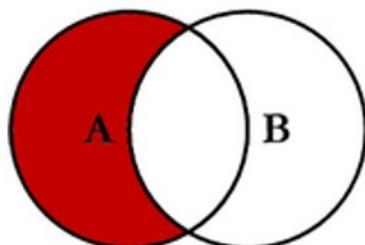
| Name | Age | Food |
|-------|-----|-----------|
| Alice | 21 | Hamburger |
| Bill | 24 | Pizza |
| Carl | 23 | Beer |
| Dina | 19 | Shrimp |



PRACTICE 3

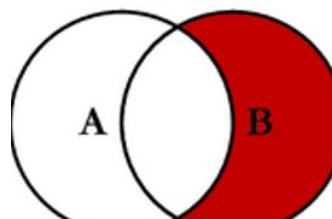
3. แสดงชื่อ และอายุของลูกค้า ที่สั่งอาหารแต่ไม่มีอาหารนั้นในตารางเมนู

```
SELECT          name, age  
FROM           customers  
LEFT OUTER JOIN menu  
ON            (menu.food = customers.food)  
WHERE          Menu.food IS NULL ;
```



```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```

```
SELECT          name, age  
FROM           menu  
RIGHT OUTER JOIN customers  
ON            (menu.food = customers.food)  
WHERE          Menu.food IS NULL ;
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



PRACTICE 4

4. แสดง ชื่อลูกค้า อายุ ชื่อเมนูและวันที่ลูกค้าสั่งในเมนู

```
SELECT          name,    age,    food,   day  
FROM           customers  
  
NATURAL JOIN   menu ;
```

```
SELECT          name,    age,    customers.food,day  
FROM           customers  
  
JOIN           menu  
ON             (menu.food = customers.food);
```



PRACTICE 5

5. แสดงชื่อลูกค้า อายุ ชื่อเมนู และวันที่ลูกค้าสั่งสิ่งเมนูต่างๆ และข้อมูล
อื่นๆ ที่ไม่สามารถจับคู่ตามเงื่อนไขได้

ORACLE

```
SELECT      name,  age, food, day
FROM        customers
FULL OUTER JOIN menu
ON          (menu.food =
customers.food);
```

MySQL

```
SELECT      name, age,customers.food, day
FROM        customers
LEFT OUTER JOIN menu
ON          (menu.food = customers.food)
UNION
SELECT      name, age, Menu.food, day
FROM        customers
RIGHT OUTER JOIN menu
ON          (menu.food = customers.food) ;
```

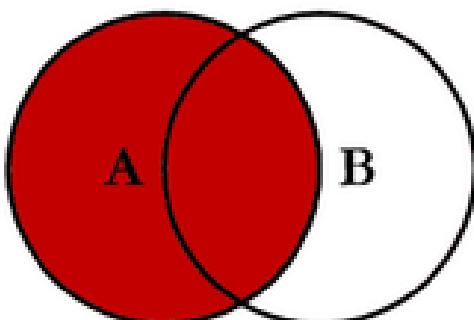


SUMMARY

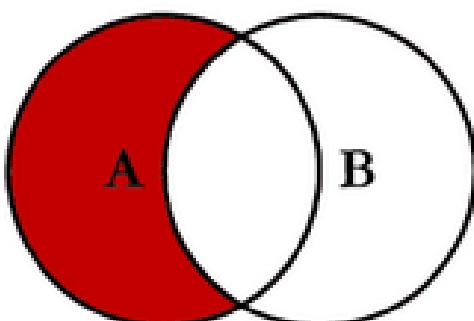
- **Cartesian Products**
 - A Cartesian product results in a display of all combinations of rows.
 - This is done by either omitting the WHERE clause or specifying the CROSS JOIN clause.
- **Table Aliases**
 - Table aliases speed up database access.
 - Table aliases can help to keep SQL code smaller by conserving memory.
- There are multiple ways to join tables. **Types of Joins**
 - Cross joins
 - Equijoins
 - Outer joins
 - Self-joins
 - Natural joins
 - Full (or two-sided) outer joins

| JOIN CLASSIFICATION | JOIN TYPE | SQL SYNTAX EXAMPLE | DESCRIPTION |
|---------------------|----------------|---|---|
| CROSS | CROSS JOIN | SELECT * FROM T1, T2 | Returns the Cartesian product of T1 and T2 (old style). |
| | | SELECT * FROM T1 CROSS JOIN T2 | Returns the Cartesian product of T1 and T2. |
| INNER | Old-Style JOIN | SELECT * FROM T1, T2 WHERE T1.C1=T2.C1 | Returns only the rows that meet the join condition in the WHERE clause (old style). Only rows with matching values are selected. |
| | NATURAL JOIN | SELECT * FROM T1 NATURAL JOIN T2 | Returns only the rows with matching values in the matching columns. The matching columns must have the same names and similar data types. |
| | JOIN USING | SELECT * FROM T1 JOIN T2 USING (C1) | Returns only the rows with matching values in the columns indicated in the USING clause. |
| | JOIN ON | SELECT * FROM T1 JOIN T2 ON T1.C1=T2.C1 | Returns only the rows that meet the join condition indicated in the ON clause. |
| OUTER | LEFT JOIN | SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1 | Returns rows with matching values and includes all rows from the left table (T1) with unmatched values. |
| | RIGHT JOIN | SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1 | Returns rows with matching values and includes all rows from the right table (T2) with unmatched values. |
| | FULL JOIN | SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1 | Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values. |

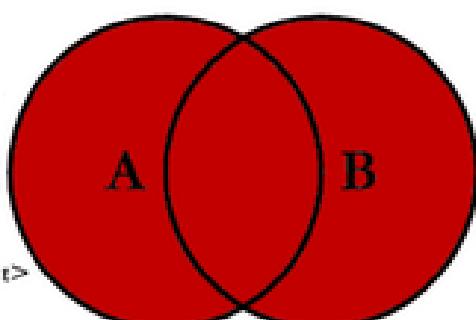
SQL JOINS



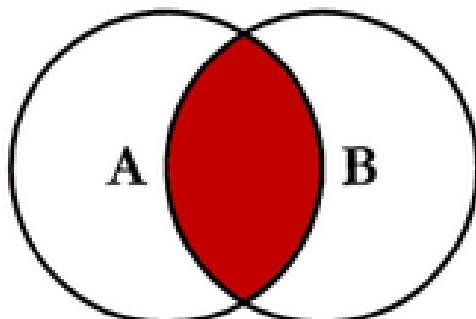
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



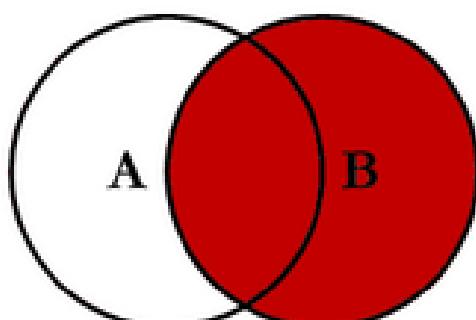
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL.
```



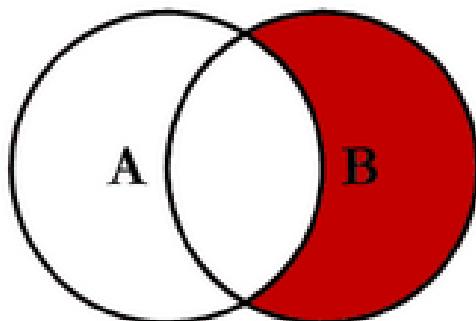
```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



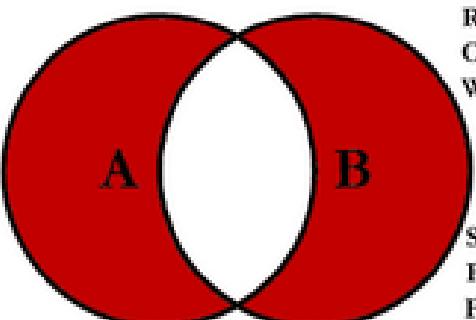
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL.
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL.  
OR B.Key IS NULL.
```

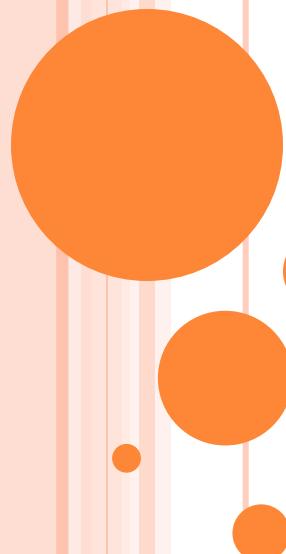
JOINING TABLES USING SQL:1999 SYNTAX

- Use a join to query data from more than one table:

```
SELECT table1.column, table2.column
      FROM    table1
      [CROSS JOIN table2] |
      [NATURAL JOIN table2] |
      [JOIN table2 USING (column_name) ] |
      [JOIN table2
           ON (table1.column_name = table2.column_name) ] |
      [LEFT | RIGHT | FULL OUTER JOIN table2
           ON (table1.column_name = table2.column_name) ] ;
```

INTRODUCTION TO SQL

GROUP FUNCTION



By Kanokwan Atchariyachanvanich
Faculty of Information Technology
KMITL
Database System Concepts

OBJECTIVES

- After completing this lesson, you should be able to do the following:
 - Describe the use of group by functions
 - Group data by using the GROUP BY clause
 - Include or exclude grouped rows by using the HAVING clause

WHAT ARE AGGREGATE (GROUP BY) FUNCTIONS?

- Group functions operate on sets of rows to give one result per group.

EMPLOYEES

| | DEPARTMENT_ID | SALARY |
|----|---------------|--------|
| 1 | 10 | 4400 |
| 2 | 20 | 13000 |
| 3 | 20 | 6000 |
| 4 | 110 | 12000 |
| 5 | 110 | 8300 |
| 6 | 90 | 24000 |
| 7 | 90 | 17000 |
| 8 | 90 | 17000 |
| 9 | 60 | 9000 |
| 10 | 60 | 6000 |
| 11 | 60 | 4200 |
| 12 | 50 | 5800 |
| 13 | 50 | 3500 |
| 14 | 50 | 3100 |
| 15 | 50 | 2600 |

Maximum salary in
EMPLOYEES table

| | MAX(SALARY) |
|---|-------------|
| 1 | 24000 |

TYPES OF AGGREGATE (GROUP BY) FUNCTIONS

| Function | Description |
|--|--|
| AVG([DISTINCT ALL] <i>n</i>) | Average value of <i>n</i> , ignoring null values |
| COUNT({* [DISTINCT ALL] <i>expr</i> }) | Number of rows, where <i>expr</i> evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls) |
| MAX([DISTINCT ALL] <i>expr</i>) | Maximum value of <i>expr</i> , ignoring null values |
| MIN([DISTINCT ALL] <i>expr</i>) | Minimum value of <i>expr</i> , ignoring null values |
| STDDEV([DISTINCT ALL] <i>x</i>) | Standard deviation of <i>n</i> , ignoring null values |
| SUM([DISTINCT ALL] <i>n</i>) | Sum values of <i>n</i> , ignoring null values |
| VARIANCE([DISTINCT ALL] <i>x</i>) | Variance of <i>n</i> , ignoring null values |

GROUP BY FUNCTIONS: SYNTAX

```
SELECT      [column,] aggregate_function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

Note: All aggregate functions ignore null values. To substitute a value for null values, use the NVL, NVL2, or COALESCE functions. (Oracle)

USING THE **AVG** AND **SUM** FUNCTIONS

- You can use AVG and SUM for numeric data.
- ตัวอย่าง: แสดงค่าเฉลี่ย ค่าสูงสุด ค่าต่ำสุด และผลรวม ของเงินเดือนของ พนักงานตัวแทนขายทั้งหมด (sales representatives).

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
  FROM employees
 WHERE job_id LIKE '%REP%';
```

| AVG(SALARY) | MAX(SALARY) | MIN(SALARY) | SUM(SALARY) |
|-------------|-------------|-------------|-------------|
| 8272.72727 | 11500 | 6000 | 273000 |

USING THE **MIN** AND **MAX** FUNCTIONS

- You can use **MIN** and **MAX** for numeric, character, and date data types.
- ตัวอย่าง: วันที่ที่พนักงานเข้าทำงานเริ่มแรกสุด และวันที่ที่พนักงานเข้าทำงานวันล่าสุด

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

| MIN(HIRE_DATE) | MAX(HIRE_DATE) |
|----------------|----------------|
| 17-JUN-87 | 21-APR-00 |

USING THE COUNT FUNCTION

1. COUNT(*) returns the number of rows in a table:

- ตัวอย่าง: แสดงจำนวนพนักงานทั้งหมด

```
SELECT      COUNT(*)  
FROM        employees
```

COUNT(*)

107

Showing 1 to 1 of 1 entries

NOTE:

1. COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns.
2. If a WHERE clause is included in the SELECT statement, COUNT(*) returns 8 the number of rows that satisfy the condition in the WHERE clause.

USING THE COUNT FUNCTION

1. COUNT(*) returns the number of rows in a table:

- ตัวอย่าง: แสดงจำนวนพนักงานทั้งหมดที่อยู่ในแผนกรหัสที่ 60

```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 60;
```

COUNT(*)

5

NOTE:

1. COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns.
2. If a WHERE clause is included in the SELECT statement, COUNT(*) returns 9 the number of rows that satisfy the condition in the WHERE clause.

USING THE COUNT FUNCTION

- COUNT(column_name) returns the number of rows in a specific column in a table:

- ตัวอย่าง: แสดงจำนวนพนักงานทั้งหมดที่กำหนดให้รับค่าคอมมิชชัน

```
SELECT COUNT(commission_pct)  
FROM employees
```

COUNT(commission_pct)

35

Showing 1 to 1 of 1 entries

NOTE:

- COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns.
- If a WHERE clause is included in the SELECT statement, COUNT(*) returns ¹⁰ the number of rows that satisfy the condition in the WHERE clause.

USING THE COUNT FUNCTION

2. COUNT(expr) returns the number of rows with non-null values for expr

- ตัวอย่าง: แสดงจำนวนพนักงานทั้งหมดที่อยู่ในแผนกรหัสที่ 80 ที่รับค่าคอมมิชชัน

```
SELECT COUNT(commission_pct)  
FROM employees  
WHERE department_id = 80;
```

NOTE: COUNT(expr) returns the number of non-null values that are in the column identified by expr.

```
SELECT COUNT(department_id)  
FROM employees ;
```

USING THE **DISTINCT** FUNCTION

3. COUNT(DISTINCT expr) returns the number of distinct non-null values of expr.
 - ตัวอย่าง: แสดงจำนวนแผนกที่ไม่ซ้ำกันในตาราง employees

```
SELECT COUNT(DISTINCT department_id)  
FROM employees;
```

COUNT(DISTINCTDEPARTMENT_ID)

11

PRACTICE 1: AGGREGATE FUNCTION

1. จงเขียน SQL statement แสดงจำนวนพนักงานที่เริ่มทำงานในปี 1999

```
SELECT      count(employee_id)  
FROM        employees  
where       hire_date between '1999-01-01' and '1999-12-31'
```

AGGREGATE FUNCTIONS AND **NULL** VALUES

1. Aggregate functions ignore null values in the column

- Example: The average is calculated as the total commission that is paid to all employees **divided by the number of employees receiving a commission** (มี 35 คน).

```
SELECT AVG(commission_pct)  
FROM employees;
```

| |
|---------------------|
| AVG(COMMISSION_PCT) |
| .222857143 |

The average is calculated based on *only* those rows in the table where a valid value is stored in the COMMISSION_PCT column.

AGGREGATE FUNCTIONS AND **NULL** VALUES

2. The IFNULL function : IFNULL(column_name, replace_with) forces group functions to include null values
- Example: The average is calculated as the total commission that is paid to all employees **divided by the total number of employees in the company** (มี 107 คน).

MySQL

```
SELECT      AVG(IFNULL(commission_pct, 0))  
FROM        employees;
```

ORACLE

```
SELECT      AVG(NVL(commission_pct, 0))  
FROM        employees;
```

AVG(NVL(COMMISSION_PCT,0))

.072897196

The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMMISSION_PCT column.

PRACTICE 2: AGGREGATE FUNCTIONS

| JOBS |
|------------|
| job_id |
| job_title |
| min_salary |
| max_salary |

2. จงเขียน SQL statement แสดงค่าเฉลี่ยของเงินเดือนขั้นต่ำ (average_salary), ค่าสูงสุดของเงินเดือนขั้นต่ำ (max_salary) และค่าต่ำสุดของเงินเดือนขั้นต่ำ (low_salary) จากตารางงานทั้งหมด พร้อมทั้งแสดงจำนวนรหัสงานที่มีอยู่ทั้งหมด (count_job)

| average_salary | max_salary | low_salary | count_job |
|----------------|------------|------------|-----------|
| 7240.0000 | 20000 | 2000 | 20 |

Showing 1 to 1 of 1 entries

Previous 1 Next

```
SELECT AVG(min_salary) average_salary ,  
       MAX(min_salary) max_salary,  
       MIN(min_salary) low_salary,  
       COUNT(job_id) count_job  
  
FROM    JOBS ;
```

CREATING GROUPS OF DATA

EMPLOYEES

| | DEPARTMENT_ID | SALARY |
|----|---------------|--------|
| 1 | 10 | 4400 |
| 2 | 20 | 13000 |
| 3 | 20 | 6000 |
| 4 | 50 | 2500 |
| 5 | 50 | 2600 |
| 6 | 50 | 3100 |
| 7 | 50 | 3500 |
| 8 | 50 | 5800 |
| 9 | 60 | 9000 |
| 10 | 60 | 6000 |
| 11 | 60 | 4200 |
| 12 | 80 | 11000 |
| 13 | 80 | 8600 |
| 14 | 80 | 10500 |
| 15 | 90 | 17000 |
| 16 | 90 | 24000 |
| 17 | 90 | 17000 |
| 18 | 110 | 8300 |
| 19 | 110 | 12000 |
| 20 | (null) | 7000 |

4400
9500
3500
6400
10033
19333
10150
7000

Average salary in the EMPLOYEES table for each department

| | DEPARTMENT_ID | Avg(Salary) |
|---|---------------|-----------------------|
| 1 | 10 | 4400 |
| 2 | 20 | 9500 |
| 3 | 50 | 3500 |
| 4 | 60 | 6400 |
| 5 | 80 | 10033.333333333333... |
| 6 | 90 | 19333.333333333333... |
| 7 | 110 | 10150 |
| 8 | (null) | 7000 |

you need to divide the table of information into smaller groups.
You can do this by using the **GROUP BY** clause

CREATING GROUPS OF DATA: **GROUP BY** CLAUSE SYNTAX

- สามารถแบ่งชุดแถวข้อมูลเป็นกลุ่มย่อยๆ โดยใช้ GROUP BY clause.

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

In the syntax:

group_by_expression specifies columns whose values determine the basis for grouping rows

Guidelines

- You must include the **columns** in the GROUP BY clause.
- You **cannot use a column alias** in the GROUP BY clause.

USING THE GROUP BY CLAUSE

- GROUP BY column ไม่จำเป็นต้องอยู่ในรายการ SELECT
- ตัวอย่าง: แสดงรหัสแผนก และค่าเฉลี่ยเงินเดือนของแต่ละแผนก

```
SELECT      department_id, AVG(salary)  
FROM        employees  
GROUP BY    department_id ;
```

| department_id | AVG(salary) |
|---------------|--------------|
| NULL | 7000.000000 |
| 10 | 4400.000000 |
| 20 | 9500.000000 |
| 30 | 4150.000000 |
| 40 | 6500.000000 |
| 50 | 3475.555556 |
| 60 | 5760.000000 |
| 70 | 10000.000000 |
| 80 | 8955.882353 |
| 90 | 19333.333333 |
| 100 | 8600.000000 |
| 110 | 10150.000000 |

Showimg 1 to 12 of 12 entries

USING THE ORDER BY CLAUSE

- ตัวอย่าง: แสดงรหัสแผนกและค่าเฉลี่ยเงินเดือนสำหรับแต่ละแผนก พร้อมทั้งจัดลำดับค่าเฉลี่ยเงินเดือนจากน้อยไปมาก

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY    department_id
ORDER BY    AVG(salary) ;
```

| DEPARTMENT_ID | AVG(SALARY) |
|---------------|-------------|
| 50 | 3475.55556 |
| 30 | 4150 |
| 10 | 4400 |
| 60 | 5760 |
| 40 | 6500 |
| | 7000 |
| 100 | 8600 |
| 80 | 8955.88235 |
| 20 | 9500 |
| 70 | 10000 |
| 110 | 10150 |
| 90 | 19333.3333 |

12 rows selected.

GROUPING BY MORE THAN ONE COLUMN

EMPLOYEES

| | DEPARTMENT_ID | JOB_ID | SALARY |
|----|---------------|------------|--------|
| 1 | | 10 AD_ASST | 4400 |
| 2 | | 20 MK_MAN | 13000 |
| 3 | | 20 MK_REP | 6000 |
| 4 | 50 | ST_CLERK | 2500 |
| 5 | 50 | ST_CLERK | 2600 |
| 6 | 50 | ST_CLERK | 3100 |
| 7 | 50 | ST_CLERK | 3500 |
| 8 | 50 | ST_MAN | 5800 |
| 9 | 60 | IT_PROG | 9000 |
| 10 | 60 | IT_PROG | 6000 |
| 11 | 60 | IT_PROG | 4200 |
| 12 | 80 | SA_REP | 11000 |
| 13 | 80 | SA_REP | 8600 |
| 14 | 80 | SA_MAN | 10500 |
| 15 | 90 | AD_VP | 17000 |
| 16 | 90 | AD_PRES | 24000 |
| 17 | 90 | AD_VP | 17000 |
| 18 | 110 | AC_ACCOUNT | 8300 |
| 19 | 110 | AC_MGR | 12000 |
| 20 | (null) | SA_REP | 7000 |

Add the salaries in the EMPLOYEES table for each job, grouped by department

| | DEPARTMENT_ID | JOB_ID | SUM(SALARY) |
|----|---------------|------------|-------------|
| 1 | | 10 AD_ASST | 4400 |
| 2 | | 20 MK_MAN | 13000 |
| 3 | | 20 MK_REP | 6000 |
| 4 | 50 | ST_CLERK | 11700 |
| 5 | 50 | ST_MAN | 5800 |
| 6 | 60 | IT_PROG | 19200 |
| 7 | 80 | SA_MAN | 10500 |
| 8 | 80 | SA_REP | 19600 |
| 9 | 90 | AD_PRES | 24000 |
| 10 | 90 | AD_VP | 34000 |
| 11 | 110 | AC_ACCOUNT | 8300 |
| 12 | 110 | AC_MGR | 12000 |
| 13 | (null) | SA_REP | 7000 |

Sometimes you need to see results for groups within groups. The slide shows a report that displays the total salary that is paid to each job title in each department.

USING THE GROUP BY CLAUSE ON MULTIPLE COLUMNS

- return summary results for groups and subgroups by listing more than one GROUP BY column

```
SELECT      department_id dept_id, job_id, SUM(salary)  
FROM        employees  
GROUP BY    department_id, job_id;
```

| DEPT_ID | JOB_ID | SUM(SALARY) |
|---------|------------|-------------|
| | SA_RFP | 7000 |
| 10 | AD_ASST | 4400 |
| 20 | MK_MAN | 13000 |
| 20 | MK_REP | 6000 |
| 30 | PU_MAN | 11000 |
| 30 | PU_CLERK | 13900 |
| 40 | HR REP | 6500 |
| 50 | ST_MAN | 36400 |
| 50 | SH_CLERK | 64300 |
| 50 | ST_CLERK | 55700 |
| 60 | IT_PROG | 28800 |
| 70 | PR_REP | 10000 |
| 80 | SA_MAN | 61000 |
| 80 | SA_REP | 243500 |
| 90 | AD_VP | 34000 |
| 90 | AD_PRES | 24000 |
| 100 | FI_MGR | 12000 |
| 100 | FI_ACCOUNT | 39600 |
| 110 | AC_MGR | 12000 |
| 110 | AC_ACCOUNT | 8300 |

PRACTICE 3: GROUPS OF DATA

3. แสดงรหัสแผนก รหัสงาน รวมเงินเดือน (total_salary) ในรหัสงานเดียวกันที่อยู่ในแผนกเดียวกัน พร้อมทั้งแสดงผลลัพธ์เรียงด้วยรหัสแผนก จากน้อยไปมาก โดยในแผนกเดียวกันให้เรียงรวมเงินเดือนจากมากไปน้อย

| department_id | job_id | total_salary |
|---------------|----------|--------------|
| NULL | SA_REP | 7000.00 |
| 10 | AD_ASST | 4400.00 |
| 20 | MK_MAN | 13000.00 |
| 20 | MK_REP | 6000.00 |
| 30 | PU_CLERK | 13900.00 |
| 30 | PU_MAN | 11000.00 |
| 40 | HR_REP | 6500.00 |
| 50 | SH_CLERK | 64300.00 |
| 50 | ST_CLERK | 53100.00 |
| 50 | ST_MAN | 36400.00 |

```
SELECT department_id, job_id, Sum(salary) total_salary  
FROM employees  
GROUP BY department_id, job_id  
ORDER BY department_id, Sum(salary) DESC;
```

PRACTICE 4: GROUPS OF DATA

4. จะเขียน SQL statement แสดงชื่อเมือง (city) เฉพาะที่มีแผนก
และจำนวนแผนกที่อยู่ในแต่ละเมือง เมือง ตั้งชื่อคอลัมน์คือ
number_of_dep

| city | number_of_dep |
|---------------------|---------------|
| London | 1 |
| Munich | 1 |
| Oxford | 1 |
| Seattle | 21 |
| South San Francisco | 1 |
| Southlake | 1 |
| Toronto | 1 |

Showing 1 to 7 of 7 entries

Previous 1 Next

SELECT City, count(department_id) number_of_dep
FROM departments
JOIN locations
USING (location_id)
GROUP BY City ;



RESTRICTING GROUP RESULTS

EMPLOYEES

| | DEPARTMENT_ID | SALARY |
|----|---------------|--------|
| 1 | 10 | 4400 |
| 2 | 20 | 13000 |
| 3 | 20 | 6000 |
| 4 | 110 | 12000 |
| 5 | 110 | 8300 |
| 6 | 90 | 24000 |
| 7 | 90 | 17000 |
| 8 | 90 | 17000 |
| 9 | 60 | 9000 |
| 10 | 60 | 6000 |
| 11 | 60 | 4200 |
| 12 | 50 | 5800 |
| 13 | 50 | 3500 |
| 14 | 50 | 3100 |
| 15 | 50 | 2600 |

The maximum salary per department when it is greater than \$10,000

| | DEPARTMENT_ID | MAX(SALARY) |
|---|---------------|-------------|
| 1 | 20 | 13000 |
| 2 | 80 | 11000 |
| 3 | 90 | 24000 |
| 4 | 110 | 12000 |

...

ใช้ HAVING clause เพื่อจำกัดกลุ่ม. To find the maximum salary in each of the departments that have a maximum salary greater than \$10,000, you need to do the following:

- หาค่าสูงสุดของเงินเดือนของแต่ละแผนกโดยการจัดกลุ่มด้วย department_id
- จำกัดเฉพาะกลุ่มของแผนกที่มีเงินเดือนสูงสุดมากกว่า \$10,000

RESTRICTING GROUP RESULTS WITH THE HAVING CLAUSE

- When you use the HAVING clause, the Oracle server restricts groups as follows:
 - จัดกลุ่มแล้ว Rows are grouped.
 - ใช้ group function
 - กลุ่มที่ตรงกับเงื่อนไขใน HAVING clause ถูกแสดงออกมา

SELECT *column, group_function*

FROM *table*

[**WHERE** *condition*]

[**GROUP BY** *group_by_expression*]

[**HAVING** *group_condition*]

[**ORDER BY** *column*];

USING THE **HAVING** CLAUSE

- จะแสดงรหัสแผนกและค่าสูงสุดของเงินเดือนของแต่ละแผนกโดยการจัดกลุ่ม และจำกัดเฉพาะกลุ่มของแผนกที่มีเงินเดือนสูงสุดมากกว่า \$10,000

```
SELECT      department_id, MAX(salary)  
FROM        employees  
GROUP BY    department_id  
HAVING      MAX(salary)>10000 ;
```

| DEPARTMENT_ID | MAX(SALARY) |
|---------------|-------------|
| 20 | 13000 |
| 30 | 11000 |
| 80 | 14000 |
| 90 | 24000 |
| 100 | 12000 |
| 110 | 12000 |

- ใช้ GROUP BY clause โดยไม่จำเป็นต้องใช้ group function ใน SELECT
- ถ้าจำกัดผลลัพธ์ของ Group function คุณต้องมี GROUP BY clause และ HAVING clause. ใน sql statement.

USING THE **HAVING** CLAUSE

- จะแสดงรหัสแผนกและค่าเฉลี่ยของเงินเดือนของแต่ละแผนกด้วยการจัดกลุ่ม และ จำกัดเฉพาะกลุ่มของแผนกที่มีเงินเดือนสูงสุดมากกว่า \$10,000

```
SELECT      department_id, AVG(salary)  
FROM        employees  
GROUP BY    department_id  
HAVING      MAX(salary)>10000;
```

| DEPARTMENT_ID | AVG(SALARY) |
|---------------|-------------|
| 20 | 9500 |
| 30 | 4150 |
| 80 | 8955.88235 |
| 90 | 19333.3333 |
| 100 | 8600 |
| 110 | 10150 |

USING THE **HAVING** CLAUSE

- แสดงรหัสงานและผลรวมเงินเดือนของแต่ละรหัสงานที่มีผลรวมเกิน \$13,000 โดยไม่รวมตัวแทนขาย และเรียงลำดับตามผลรวมเงินเดือนจากน้อยไปมาก

```
SELECT      job_id, SUM(salary) PAYROLL  
FROM        employees  
WHERE       job_id NOT LIKE '%REP%'  
GROUP BY    job_id  
HAVING      SUM(salary) > 13000  
ORDER BY    SUM(salary);
```

| JOB_ID | PAYROLL |
|------------|---------|
| PU_CLERK | 13900 |
| AD_PRES | 24000 |
| IT_PROG | 28800 |
| AD_VP | 34000 |
| ST_MAN | 36400 |
| FI_ACCOUNT | 39600 |
| ST_CLERK | 55700 |
| SA_MAN | 61000 |
| SH_CLERK | 64300 |

9 rows selected.

NESTING GROUP FUNCTIONS

- Group functions can be nested to a depth of two.
- To display the maximum average salary. แสดงเงินเดือนเฉลี่ยที่สูงสุดจากทุกแผนก โดยคำนวณจากค่าเฉลี่ยเงินเดือนของพนักงานในแผนกเดียวกัน

ORACLE

```
SELECT MAX(AVG(salary))  
      FROM employees  
  GROUP BY department_id;
```

MySQL

```
SELECT MAX(avsal.sal)  
FROM (select avg(e.salary) sal  
      from employees e  
     group by department_id) avsal ;
```

MAX(AVG(SALARY))

19333.3333

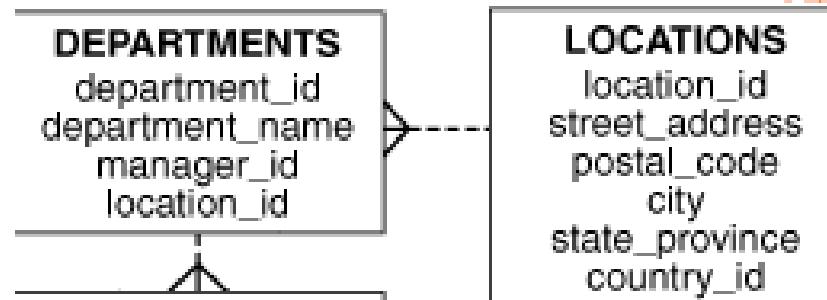
PRACTICE 5: HAVING CLAUSE

5. จงเขียน SQL statement แสดงชื่อเมือง (city) เฉพาะที่มีแผนก
และจำนวนแผนกที่มีมากกว่า 1 แผนกในแต่ละเมือง ตั้งชื่อคอลัมน์คือ
`number_of_dep`

| city | number_of_dep |
|---------|---------------|
| Seattle | 21 |

Showing 1 to 1 of 1 entries

```
SELECT      City, count(department_id) number_of_dep
FROM        departments
JOIN        locations
USING       (location_id)
GROUP BY    City
HAVING      Count(department_id) > 1;
หรือ HAVING  number_of_dep > 1 ;
```

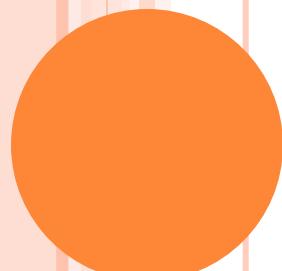


SUMMARY

- In this lesson, you should have learned how to:
 - Use the group functions COUNT, MAX, MIN, and AVG
 - Write queries that use the GROUP BY clause and HAVING clause
- You can create subgroups by using the GROUP BY clause.
- Groups can be restricted using the HAVING clause.
- Place the HAVING and GROUP BY clauses after the WHERE clause in a statement.
- The order of the HAVING and GROUP clauses following the WHERE clause is not important.
- Place the ORDER BY clause last.
- The Oracle server evaluates the clauses in the following order:
 1. If the statement contains a WHERE clause, the server establishes the candidate rows.
 2. The server identifies the groups that are specified in the GROUP BY clause.
 3. The HAVING clause further restricts result groups that do not meet the group criteria in the HAVING clause.

INTRODUCTION TO SQL

SUBQUERIES



By Kanokwan Atchariyachanvanich

Faculty of Information Technology

KMITL

Database System Concepts



OBJECTIVES

- After completing this lesson, you should be able to do the following:
- Describe the types of problem that subqueries can solve
- Define subqueries
- List the types of subqueries
- Write single-row and multiple-row subqueries

SUBQUERY

- Inner query หรือ Nested query
- ดึงข้อมูลใน table จาก ผลลัพธ์ของการทำ SQL Select query ก่อนหน้านี้
- สามารถใช้ subquery ในส่วน From เพื่อสร้างตารางเสริมอื่นจากผลลัพธ์ SQL select query
- สามารถใช้ subquery ใน Where Clause
 - ถูกใช้ในเงื่อนไขที่ main query ไม่สามารถดึงข้อมูลได้ตามเงื่อนไขปกติที่สามารถทำได้ หรือ
 - ติดข้อจำกัดภายใต้กฎของ SQL
- Subquery ถูกใช้ภายใต้
 - SQL Select, SQL Insert, SQL Update SQL Delete ที่เป็น statement และ SQL Operator ต่างๆ

USING A SUBQUERY TO SOLVE A PROBLEM

- Who has a salary greater than Abel's?

Main Query:



Which employees have salaries greater than Abel's salary?

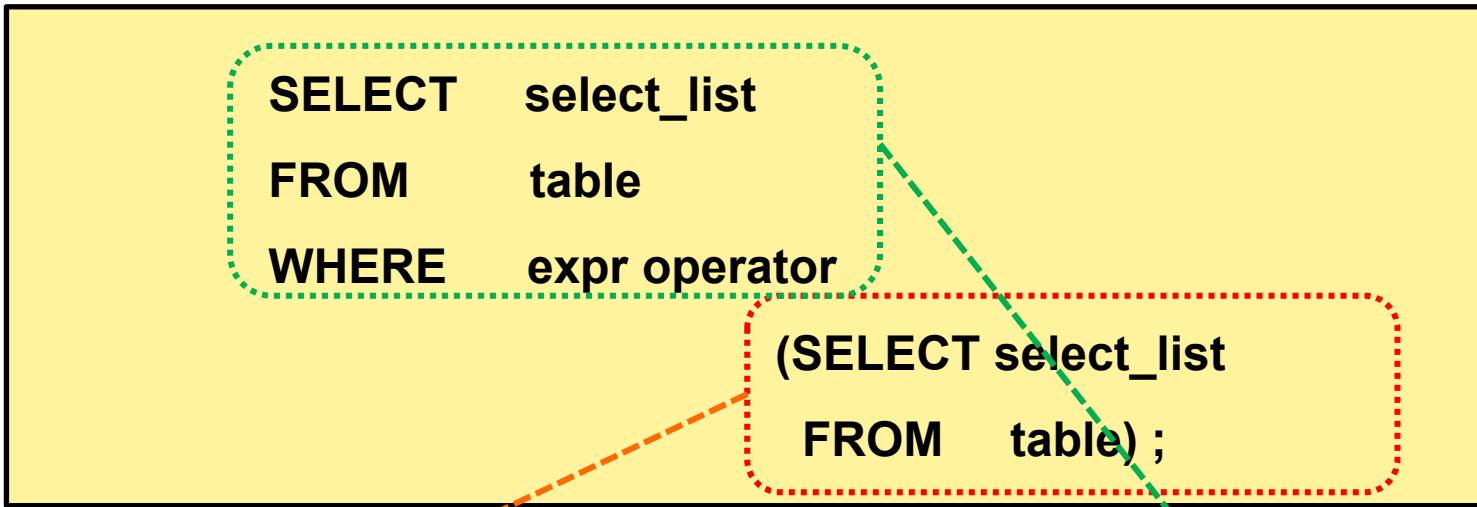


Subquery



What is Abel's salary?

SUBQUERY SYNTAX



- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by **the main query** (outer query).

GUIDELINES FOR USING SUBQUERIES

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- Operators Usage
 - single-row operators ($>$, $=$, \geq , $<$, \leq , \neq) with single-row subqueries
 - multiple-row operators (IN, ANY, ALL) with multiple-row subqueries.

USING A SUBQUERY

Who has a salary greater than Abel's?

แสดงทุกคน ที่มีเงินเดือนมากกว่าเงินเดือนของ Abel

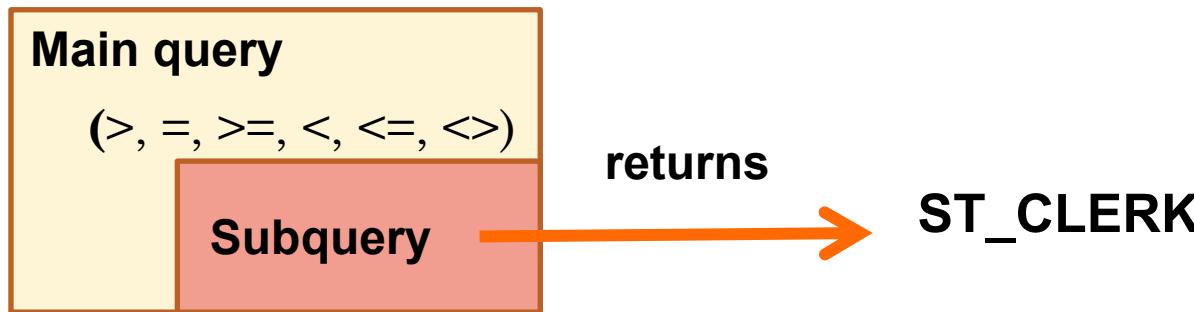
```
SELECT last_name  
FROM employees  
WHERE salary >  
      (SELECT salary  
       FROM employees  
       WHERE last_name = 'Abel');
```

| LAST_NAME |
|-----------|
| King |
| Kochhar |
| De Haan |
| Greenberg |
| Russell |
| Partners |
| Errazuriz |
| Ozer |
| Hartstein |
| Higgins |

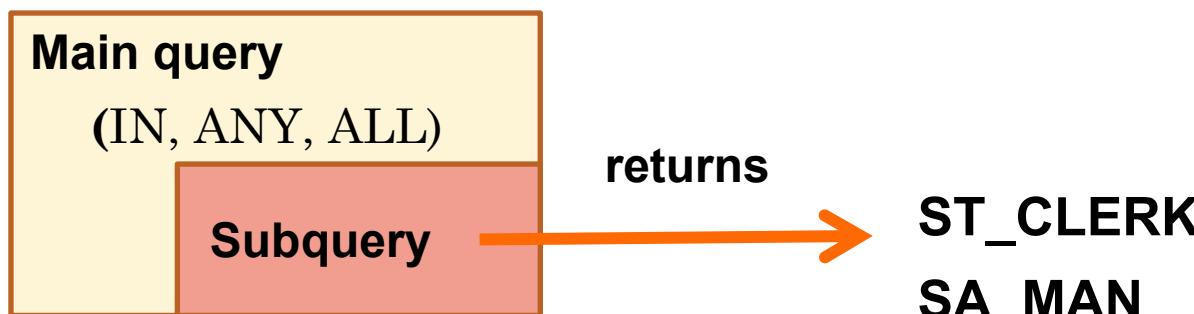
10 rows selected.

TYPES OF SUBQUERIES

- Single-row subquery



- Multiple-row subquery



SINGLE-ROW SUBQUERIES

- Return only one row
- Use single-row comparison operators

| Operator | Meaning |
|----------|--------------------------|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| <> | Not equal to |

EXECUTING SINGLE-ROW SUBQUERIES

- แสดงนามสกุล รหัสงาน และเงินเดือนของพนักงานทุกคนที่มีรหัสงาน เมื่อันกับพนักงานรหัส 141

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE job_id = ST_CLERK  
  
(SELECT job_id  
FROM employees  
WHERE employee_id = 141);
```

| LAST_NAME | JOB_ID | SALARY |
|-------------|----------|--------|
| Nayer | ST_CLERK | 3200 |
| Mikkilineni | ST_CLERK | 2700 |
| Bisot | ST_CLERK | 3300 |
| Atkinson | ST_CLERK | 2800 |

EXECUTING SINGLE-Row SUBQUERIES (CONT.)

แสดงนามสกุล รหัสงาน และเงินเดือนของพนักงานทุกคนที่มีรหัสงานเหมือนกับพนักงานรหัส 141 และมีเงินเดือนมากกว่าเงินเดือนของพนักงานรหัส 143

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE job_id = ST_CLERK  
      (SELECT job_id  
       FROM employees  
      WHERE employee_id = 141)  
  
AND salary > 2600  
      (SELECT salary  
       FROM employees  
      WHERE employee_id = 143);
```

USING GROUP FUNCTIONS IN A SUBQUERY

- แสดงนามสกุล รหัสงาน และเงินเดือนของพนักงานทุกคนที่มีเงินเดือนเท่ากับเงินเดือนต่ำสุด

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE salary = 2100  
(SELECT MIN(salary)  
FROM employees) ;
```

| LAST_NAME | JOB_ID | SALARY |
|-----------|----------|--------|
| Olson | ST_CLERK | 2100 |

PRACTICE# 1

- แสดงชื่อ นามสกุล รหัสงานและวันที่เริ่มทำงานของพนักงานที่เริ่มทำงานหลังรหัสพนักงาน 105 เรียงลำดับตามวันที่เริ่มทำงานอีตไปล่าสุด

```
SELECT first_name, last_name, job_id, hire_date  
FROM employees  
WHERE hire_date > 1998-02-05  
  
(SELECT hire_date  
FROM employees  
WHERE employee_id = 105 )  
  
order by hire_date
```

THE HAVING CLAUSE WITH SUBQUERIES

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.
- แสดงรหัสแผนก เงินเดือนต่ำสุดในแผนกเดียวกัน เช่น กลุ่มที่มีเงินเดือนต่ำสุดมากกว่าพนักงานในแผนกรหัส 50 ที่ได้เงินเดือนต่ำสุด

```
SELECT      department_id, MIN(salary)
FROM        employees
GROUP BY    department_id
HAVING      MIN(salary) > 2100
            (SELECT MIN(salary)
             FROM   employees
             WHERE  department_id = 50);
```

PRACTICE# 2

- แสดงรหัสแผนก เงินเดือนสูงสุดในแผนกเดียวกัน เช่นกลุ่มที่มีเงินเดือนสูงสุดมากกว่าพนักงานในแผนกรหัส 80 ที่ได้เงินเดือนสูงสุด

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING      MAX(salary) > 14000
            (SELECT MAX(salary)
             FROM   employees
             WHERE  department_id = 80) ;
```

WHAT IS WRONG WITH THIS STATEMENT?

```
SELECT employee_id, last_name  
FROM employees  
WHERE salary =  
      (SELECT MIN(salary)  
       FROM employees  
      GROUP BY department_id) ;
```

Single-row operator (=) with multiple-row subquery

ERROR at line 4:
ORA-01427: single-row subquery returns more than one row

WILL THIS STATEMENT RETURN Rows?

```
SELECT last_name, job_id  
FROM employees  
WHERE job_id = NULL  
  
(SELECT job_id  
FROM employees  
WHERE last_name = 'Hass') ;
```

PRACTICE#3

- แสดงนามสกุล รหัสงาน และวันที่รับเข้าทำงานของพนักงานที่อยู่ในแผนกเดียวกัน กับแผนกที่ Austin(last_name) ทำงานอยู่ และไม่ต้องแสดงข้อมูลของ Austin

```
SELECT last_name, job_id, hire_date  
FROM employees  
WHERE department_id = 60  
(SELECT department_id  
FROM employees  
WHERE last_name = 'Austin')  
AND last_name <> 'Austin' ;
```

MULTIPLE-Row SUBQUERIES

- Return more than one row
- Use multiple-row comparison operators
- place an =, <>, >, <, <= or >= operator before ANY, ALL in your query.

| Operator | Meaning |
|----------|---|
| IN | Equal to any member in the list |
| ANY | Compare value to each value returned by the subquery |
| ALL | Compare value to every value returned by the subquery |

เป็นจริงกับค่าใดค่าหนึ่ง

เป็นจริงกับทุกค่า

MULTIPLE-Row SUBQUERIES

- IN is an alias for = ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

MULTIPLE-Row SUBQUERIES

- NOT IN is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

USING THE IN OPERATOR IN MULTIPLE-Row SUBQUERIES

- แสดงพนักงานที่ได้รับเงินเดือนเท่ากับเงินเดือนต่ำสุดของแผนกต่างๆ

```
SELECT last_name, salary, department_id  
FROM employees  
WHERE salary IN  
      (SELECT MIN(salary)  
       FROM employees  
       GROUP BY department_id);  
  
4400, 6000, 2500, .....
```

PRACTICE# 4

- แสดงชื่อ นามสกุลพนักงาน เงินเดือน รหัสแผนก ที่ได้รับเงินเดือนเท่ากับเงินเดือนสูงสุดของแผนกใดแผนกหนึ่ง (อนุโลมให้รวมกลุ่มพนักงานที่ไม่มีแผนกด้วย)

```
SELECT first_name, last_name, salary, department_id  
FROM employees  
WHERE salary IN 7000, 4400, 13000, .....  
  
(SELECT MAX(salary)  
FROM employees  
GROUP BY department_id) ;
```

PRACTICE# 5

- แสดงนามสกุล รหัสแผนก และรหัสงานของพนักงานที่แผนกทำงานตั้งอยู่ใน location รหัส 1700

```
SELECT last_name, department_id, job_id  
FROM employees  
WHERE department_id IN 10,30,90,100, ...  
  
(SELECT department_id  
FROM departments  
WHERE location_id = 1700) ;
```

USING THE ANY OPERATOR IN MULTIPLE-Row SUBQUERIES

- แสดงรหัสพนักงาน นามสกุลพนักงาน รหัสงานและเงินเดือน ที่ได้รับเงินเดือน
น้อยกว่าเงินเดือนของพนักงานที่มีรหัสงานเป็น IT_PROG **บางคน** และไม่ต้อง~~แสดง~~พนักงานที่มีรหัสงาน IT_PROG

```
SELECT      employee_id, last_name, job_id, salary
FROM        employees
WHERE       salary < ANY
              (SELECT      salary
               FROM        employees
               WHERE      job_id = 'IT_PROG')
AND         job_id <> 'IT_PROG' ;
```

USING THE **ALL** OPERATOR IN MULTIPLE-Row SUBQUERIES

- แสดงพนักงานที่ได้รับเงินเดือนน้อยกว่าเงินเดือนของ IT Programmer

ทุกคน และที่ไม่ใช่ IT Programmer

```
SELECT      employee_id, last_name, job_id, salary
FROM        employees
WHERE       salary < ALL
              (SELECT      salary
               FROM        employees
               WHERE      job_id = 'IT_PROG')
AND        job_id <> 'IT_PROG' ;
```

9000, 6000, 4800, 4800, 4200

PRACTICE# 6

- แสดงชื่อ นามสกุลพนักงาน เงินเดือน รหัสแผนก ที่ได้รับเงินเดือนมากกว่า พนักงานทุกคนในแผนก 20

```
SELECT first_name, last_name, salary, department_id  
FROM employees  
WHERE salary > ALL  
(SELECT salary  
FROM employees  
WHERE department_id=20) ;
```

13000,6000



PRACTICE# 7

- แสดงชื่อ นามสกุลพนักงาน เงินเดือน รหัสแผนก ที่ได้รับเงินเดือนมากกว่า พนักงานบางคนในแผนก 20 และไม่ต้องแสดงข้อมูลพนักงานในแผนก 20

```
SELECT first_name, last_name, salary, department_id  
FROM employees  
WHERE salary > ANY  
      (SELECT salary  
       FROM employees  
       WHERE department_id=20)  
AND department_id <> 20 ;
```

NULL VALUES IN A SUBQUERY

- แสดงพนักงานทุกคนที่ไม่มีผู้ใต้บังคับบัญชา (ไม่มีลูกน้อง)/ที่ไม่ใช่ผู้จัดการ

```
SELECT emp.last_name, emp.employee_id, emp.manager_id  
FROM employees emp  
WHERE emp.employee_id NOT IN  
      (SELECT mgr.manager_id  
       FROM employees mgr);  
          NULL, 100, 100, 102,.....
```

No rows selected.

- All conditions that compare a null value result in a null.
- So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator.
- The NOT IN operator is equivalent to <> ALL.

USING THE **NOT IN** OPERATOR IN MULTIPLE-ROW SUBQUERIES

- แสดงพนักงานทุกคนที่ไม่มีผู้ใต้บังคับบัญชา (ลูกน้อง) / ที่ไม่ใช่ผู้จัดการ

```
SELECT emp.last_name, emp.employee_id, emp.manager_id  
FROM employees emp  
WHERE emp.employee_id NOT IN  
      (SELECT mgr.manager_id  
       FROM employees mgr  
       WHERE manager_id IS NOT NULL);  
          ↑  
          100, 100, 102,.....
```

NESTING GROUP FUNCTIONS USING SUBQUERIES

- Group functions can be nested to a depth of two.
- To display the maximum average salary. แสดงเงินเดือนเฉลี่ยที่สูงสุดจากทุกแผนก โดยคำนวณจากค่าเฉลี่ยเงินเดือนของพนักงานในแผนกเดียวกัน

ORACLE

```
SELECT MAX(AVG(salary))  
      FROM employees  
     GROUP BY department_id;
```

MAX(AVG(SALARY))

MySQL

```
SELECT MAX(avsal.sal)  
FROM (select avg(e.salary) sal  
      from employees e  
     group by department_id) avsal ;
```

Column alias

Table alias

19333.3333

SUMMARY

- In this lesson, you should have learned how to:
- Identify when a subquery can help solve a question
- Write subqueries when a query is based on unknown values

```
SELECT select_list  
FROM   table  
WHERE  expr operator
```

```
(SELECT select_list  
FROM   table) ;
```