MySQL

2021.02, testworks

Day 1

1. Installing MySQL Server
2. SQL
   A. Database, DBMS
3. SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
4. JOIN 1
   A. INNER JOIN
   B. OUTER JOIN
   C. SELF JOIN
   D. CROSS JOIN
5. JOIN 2
   A. THREE WAYS JOIN
   B. Example

Day 2

6. Grouping Data
   A. GROUP BY, HAVING
7. Subquery

8. Set Operator
   UNION, UNION ALL, INTERSECT, MINUS
9. Data Manipulation Language
   A. INSERT, UPDATE, DELETE
   B. INSERT Multiple Rows, INSERT INTO SELECT, INSERT IGNORE
   C. UPDATE JOIN, ON DELETE CASCADE, DELETE JOIN
   D. REPLACE
10. Transaction
   A. ACID
   B. Commit, rollback
11. Data Type 1
   A. CHAR, VARCHAR, TEXT
   B. INT, DECIMAL, BIT, BOOLEAN
12. Data Type 2
   A. DATE, TIME, TIMESTAMP
   B. Other data type
13. Data Definition
14. ENDING

교육 전 설문

https://docs.google.com/forms/d/1xVR8AHz_CNUKaluorf-i3BZXmBGqCwEpsVdN9TsV6Os/edit#responses

참고 URL

MySQL : https://www.mysql.com/

MySQL Workbench Manual : https://dev.mysql.com/doc/

MySQL 8.0 Reference Manual : https://dev.mysql.com/doc/refman/8.0/en/

설치 & 툴 사용

```
[NS] No Sound,   [SV] Sound Video,   [LC] Live Coding
```

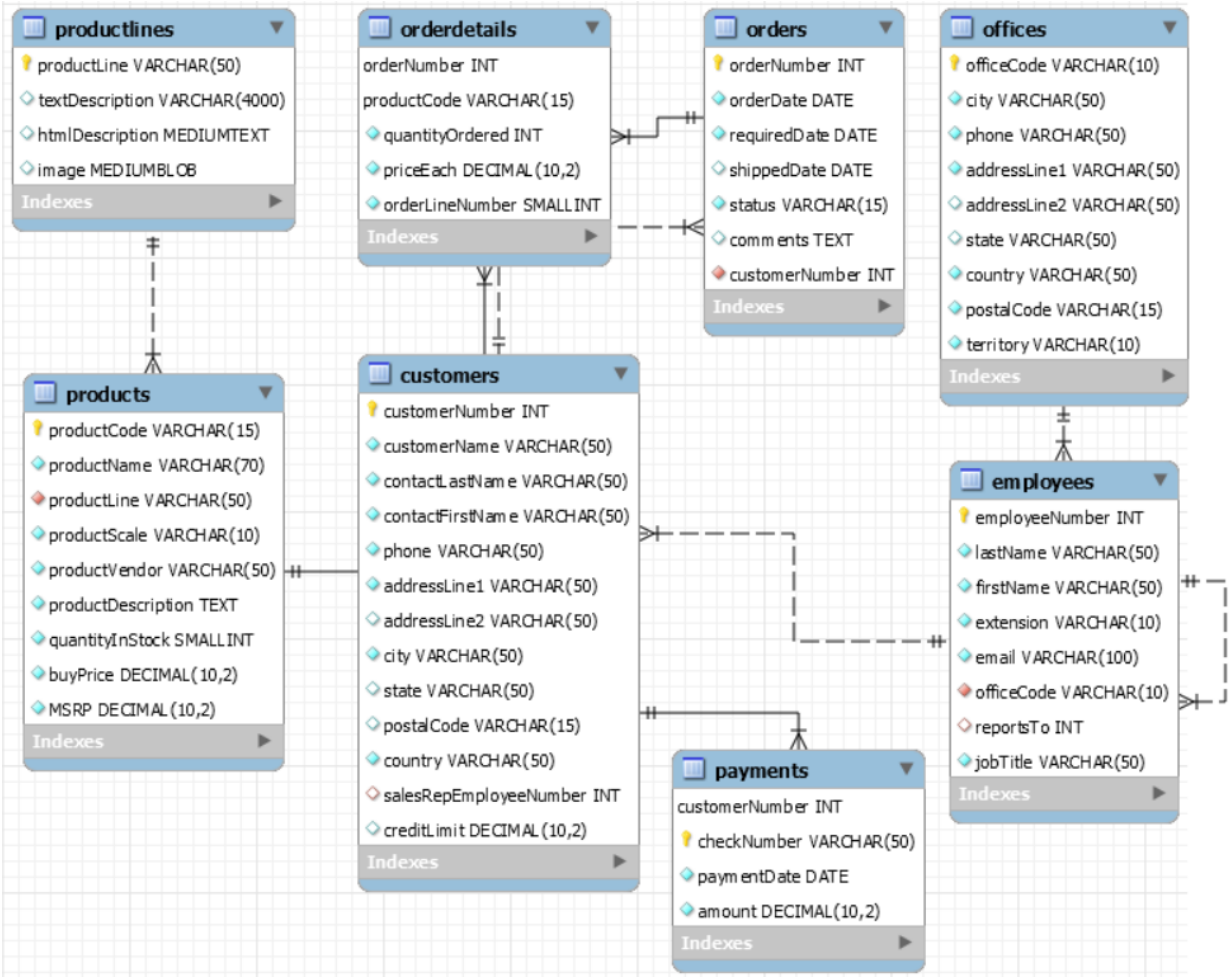
- download mysql, setup, create user
  - [NS] https://youtu.be/nT6tk4FdJMk (04:31)
- Create EMPLOYEE sample database
  - [NS] https://youtu.be/hXFLdzg0xN4 (05:22)

https://github.com/datacharmer/test_db 에서 download 후 압축 해제
path 에 C:\Program Files\MySQL\MySQL Server 8.0\bin 추가
OS> mysql -u root -p -t < employees.sql
Enter password: ****
또는
OS> mysql -u root -p
mysql> source employees.sql

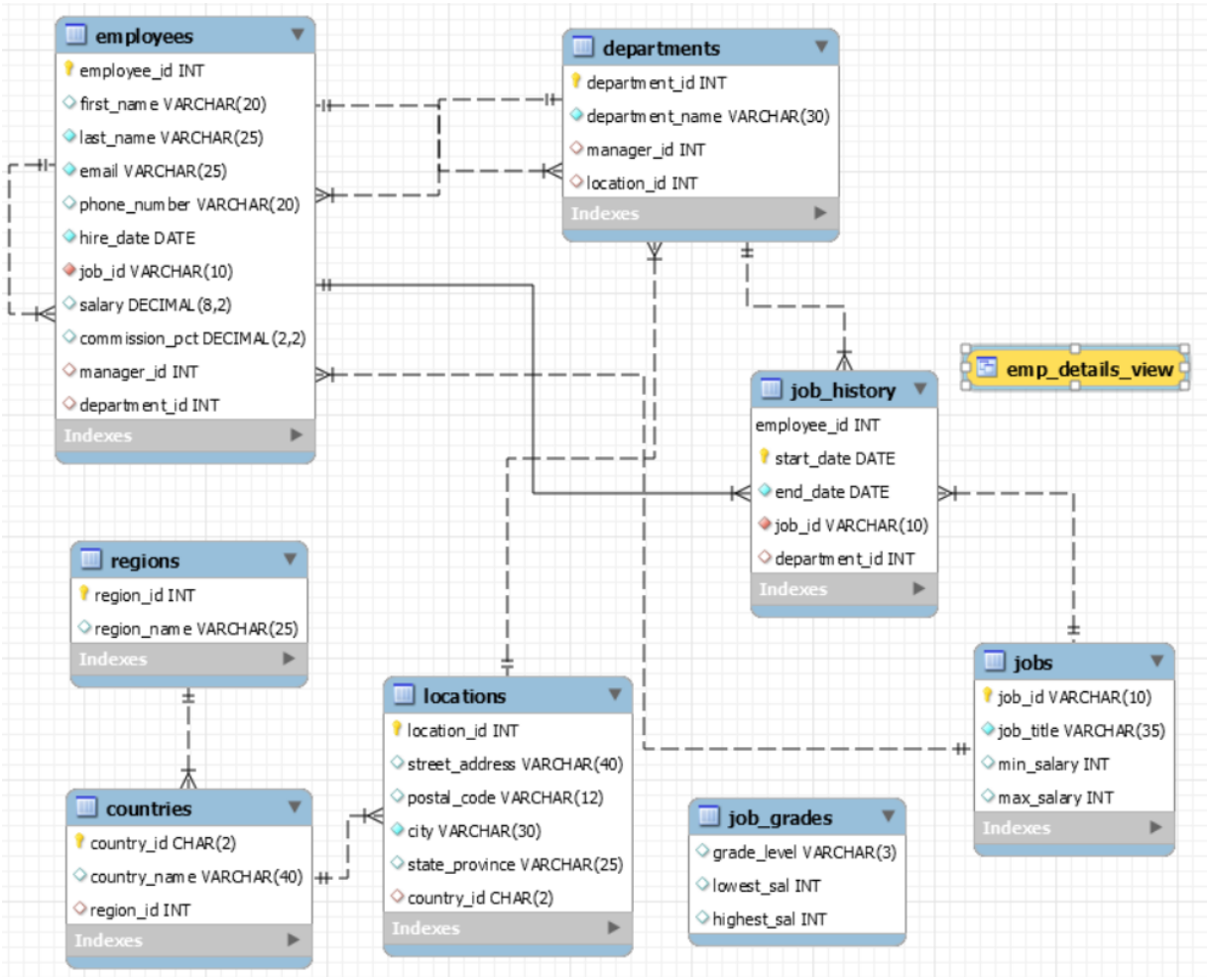
샘플 데이터베이스 생성

```
OS> dir *.sql
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: BE67-74D9
2016-05-24  오후 09:16           210,202 mysqlsampleclassicdb.sql
2021-02-16  오전 04:18            34,394 n1.sql
               2 개 파일             244,596 바이트
               0 개 디렉터리  673,037,352,960 바이트 남음

OS> mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 42
Server version: 8.0.23 MySQL Community Server - GPL

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> source mysqlsampleclassicdb.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)


mysql> source n1.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)


mysql> show databases;
+--------------------+
¦ Database           ¦
+--------------------+
¦ classicmodels      ¦
¦ information_schema ¦
¦ mysql              ¦
¦ n1                 ¦
¦ performance_schema ¦
¦ sys                ¦
¦ world              ¦
+--------------------+
```

sample schema, classicmodels



sample schema, n1



수업용 유저 생성(MySQL Workbench)

MySQL

MySQL은 Michael Monty Widenius가 David Axmark, Allan Larsson와 설립 한 스웨덴 회사 MySQL AB 가 만들었습니다. Widenius와 Axmark에 의한 MySQL의 최초 개발은 1994 년에 시작되었습니다. MySQL의 첫 번째 버전은 1995년 5월 23일에 출현하였습니다. 처음에는 mSQL을 이용하여 개인적으로 사용하기 위해 만들어졌으나 제작자들은 너무 느리다고 생각했습니다. 그들은 mSQL과 동일한 API를 유지하면서 새로운 SQL 인터페이스를 만들었습니다. API를 mSQL 시스템과 일관되게 유지함으로써 많은 개발자가 (독점적으로 라이선스가 부여 된) mSQL 대신 MySQL을 사용할 수 있었습니다.

MySQL New Features

2021년 2월 기준, MySQL은 최신 윈도우 installer 버전으로 mysql-installer-community-8.0.23.0을 제공합니다.

- Release history : https://en.wikipedia.org/wiki/MySQL#Release_history

- 5.1: http://download.nust.na/pub6/mysql/doc/refman/5.1/en/mysql-nutshell.html
- 5.4: http://download.nust.na/pub6/mysql/doc/refman/5.4/en/mysql-nutshell.html
- 5.5: https://dev.mysql.com/doc/refman/5.5/en/mysql-nutshell.html
- 5.6: https://dev.mysql.com/doc/refman/5.6/en/mysql-nutshell.html
- 5.7: https://dev.mysql.com/doc/refman/5.7/en/mysql-nutshell.html
- 8.0: https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html

cf. MySQL Server 8.0.0-DMR (Development Milestone Release)

- MySQL Reaches Milestone 8 Release : http://www.i-programmer.info/news/84-database/10081-mysql-reaches-milestone-8-release.html
- MySQL 5.7 vs MySQL 8.0 - What's new in MySQL 8.0? : https://www.eversql.com/mysql-5-7-vs-mysql-8-0-whats-new-in-mysql-8-0
- The MySQL 8.0.0 Milestone Release is available : http://mysqlserverteam.com/the-mysql-8-0-0-milestone-release-is-available
- MySQL Innovation: from 5.7 to 8.0 : https://www.slideshare.net/lefred.descamps/mysql-innovation-from-57-to-80

MySQL Workbench

MySQL 워크벤치(MySQL Workbench) 또한 mysql-installer-community-X.X.XX.X 에 같은 버전으로 통합되어 제공됩니다. MySQL 워크벤치는 MySQL SQL 개발과 관리, 데이터베이스 설계, 생성 그리고 유지를 위한 단일 개발 통합 환경을 제공하는 비주얼 데이터베이스 설계 도구입니다. fabFORCE.NET 의 DBDesigner4 의 후속판이며, 이전 소프트웨어 패키지인 MySQL GUI 툴즈 번들을 대체한 것입니다.

현재 MySQL 워크벤치도 DBDesigner4 의 기능들을 추가 확장시켜 나가며 유저와 스키마 및 간단한 백업 복원 관리, 데이터베이스의 리버스 엔지니어링과 ERD 모델 생성관리, 데이터베이스 동기화 등 편리한 종합적인 기능을 제공합니다.

Monty Widenius

http://monty-says.blogspot.com/

몬티 와이드니어스(Ulf Michael Monty Widenius, 1962년 3월 3일 출생)는 핀란드 헬싱키 출신의 프로그래머이며, MySQL 데이터베이스, MaxDB 및 MariaDB의 창시자입니다. 흔히 "몬티"(Monty)라는 애칭으로 불립니다. 마이클 와이드니어스 라고도 합니다.

MySQL AB 회사의 창립 멤버이자 MariaDB Corporation AB의 CTO이며 오픈 소스 MySQL 데이터베이스의 최초 개발자입니다.

2008년 1월에 10억 달러(약 1조 원)를 받고 썬 마이크로시스템즈에 회사를 매각하였습니다. 이 매각으로 핀란드 10대 부자 중 한 명이 되었습니다. 이후 썬 마이크로시스템즈가 오라클에 다시 매각되고 오라클이 자바 라이선스를 두고 구글 등의 기업과 소송을 벌입니다. 몬티는 이를 크게 우려하면서 MySQL과 호환되는 MariaDB를 개발하고 Monty Program AB라는 회사를 설립하여 GPL 라이선스로 배포하고 있습니다.

Widenius는 두 번째 부인 Anna와 막내 딸 Maria와 함께 헬싱키에 살고 있습니다. Widenius에게는 My, Max 및 Maria 세 자녀가 있습니다.

Oracle MySQL Cloud Service

- https://cloud.oracle.com/ko_KR/mysql
- https://www.slideshare.net/MarkSwarbrick/oracle-mysql-cloud-service-69401405
  introducing_oracle_mysql_cloud_service.pdf 참조

cf.유튜브 영상

Oracle MySQL Cloud Service : https://www.youtube.com/watch?v=xXUlHvAYtiE (비공개)
Getting Started with MySQL Cloud Service : https://www.youtube.com/watch?v=yPlmr_K3uoU
Connecting to MySQL Cloud Service : https://www.youtube.com/watch?v=XlAoIuvOcJk (비공개)

Azure Database for MySQL
- https://azure.microsoft.com/ko-kr/services/mysql

Amazon Relational Database Service(RDS)
- https://aws.amazon.com/ko/rds

Bluemix 의 새 데이터베이스 서비스: Compose for MySQL
- https://developer.ibm.com/kr/cloud/bluemix/2017/01/07/compose-for-mysql-and-compose-for-scylladb-the-new-compose-databases-on-bluemix/

용어

TABLE

- 세로줄(row, record)과 가로줄(column, field, feature)의 모델을 이용하여 정렬된 표 형태 데이터 집합.

사원명	급여	부서명
유재하	8000000	운영부
신해철	5000000	영업부
서태지	5000000	개발부
박진영	4000000	운영부

- 정규화(Normalization)란 관계형 데이터베이스의 설계에서 중복의 최소화를 위해 데이터를 구조화하는 프로세스. 위 표의 부서 명은 아래와 같이 정규화하여 중복을 제거.

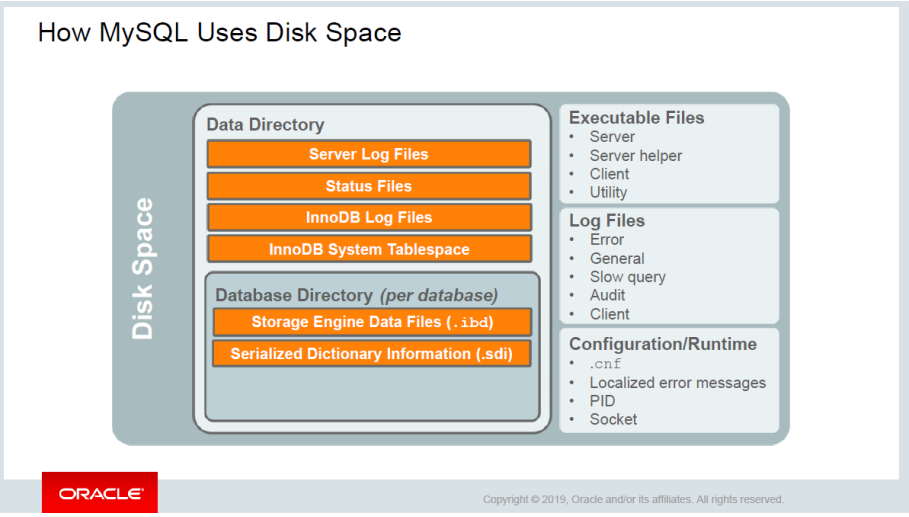
사원명	급여	부서번호
유재하	8000000	10
신해철	5000000	20
서태지	5000000	30
박진영	4000000	10

부서번호	부서명
10	운영부
20	영업부
30	개발부

Database, DBMS, RDBMS

관계형 데이터베이스(RDB, Relational Database)는 표(Table, 테이블)의 형태로 데이터를 저장한다. 그리고 테이블 형식의 데이터를 조작할 수 있는 관계 연산자(SQL, Structured Query Language)를 제공한다. 관리 시스템(MS, Management System)은 가볍게 소프트웨어라고 생각할 수 있다. 즉, "RDBMS 는 표의 형태로 데이터를 저장하고 관리하는 소프트웨어이다."

MySQL Server = Database + Instance



Database = datafile (C:\ProgramData\MySQL\MySQL Server 8.0\Data)
*.ibd, *.sdi 로 구성

MySQL Server process = Instance = mysqld

스키마(database schema)
데이터베이스의 구조.
데이터베이스에서 자료의 구조, 자료의 표현 방법, 자료 간의 관계를 형식 언어로 정의한 구조.
DBMS 가 주어진 설정에 따라 데이터베이스 스키마를 생성하며,
데이터베이스 사용자가 자료를 저장, 조회, 삭제, 변경할 때 DBMS 는 자신이 생성한 데이터베이스 스키마를 참조하여 명령을 수행한다.

스키마는 3 층 구조로 되어있다.

내부 스키마(Internal Schema) :
전체 데이터베이스의 물리적 저장 형태를 기술하는 것
외부 스키마(External Schema) :
프로그래머나 사용자의 입장에서 데이터베이스의 모습으로
조직의 일부분을 정의
개념 스키마(Conceptual Schema) :
모든 응용 시스템과 사용자들이 필요로 하는 데이터를 통합한
조직 전체의 데이터베이스 구조를 논리적으로 정의

MySQL Server 에서는 database == schema 라고 생각해도 좋다.
New Query Tab >

```
create database testdb1;
create schema   testdb2;
drop   database testdb2;

drop   schema   testdb1;
```

MySQL 서버 기본적으로 제공 데이터베이스(스키마)

```
mysql> show databases;

+--------------------+
| Database           |
+--------------------+
| mysql              |
| information_schema |
| performance_schema |
| sys                |
+--------------------+
```

mysql :
https://dev.mysql.com/doc/refman/8.0/en/sys-schema.html
information_schema :
https://dev.mysql.com/doc/refman/5.7/en/information-schema.html
performance_schema :
https://dev.mysql.com/doc/refman/5.7/en/performance-schema.html
sys :
https://dev.mysql.com/doc/refman/5.7/en/sys-schema.html
https://github.com/mysql/mysql-sys

```
SELECT table_name, table_type, engine
       FROM information_schema.tables
       WHERE table_schema = '스키마명'
       ORDER BY table_name;
```

MySQL Sample Database
- Employees Sample Database :
  https://dev.mysql.com/doc/employee/en 설치 & 툴 참고
- world Database :
  https://dev.mysql.com/doc/world-setup/en
- Sakila Sample Database :
  https://dev.mysql.com/doc/sakila/en
- MySQL Sample Database :
  http://www.mysqltutorial.org/mysql-sample-database.aspx

MySQL Programs

  Server programs
  - mysqld
  - Server Helper programs : mysql.server, mysql_safe, mysqld_multi

  Installation programs
  - mysql_secure_installation
  - mysql_tzinfo_to_sql
  - mysql_upgrade

  Utility programs
  - mysql_config_editor
  - mysqlbinlog
  - mysqldumpslow
  - mysql_ssl_rsa_setup

  Client programs
  - mysql
  - mysqladmin
  - mysqldump
  - mysqlimport
  - mysqlslap
  - mysqlshow
  - mysqlcheck

mysql 관련 프로그램 옵션 확인

    C:\Users\student> mysql --verbose --help
    C:\Users\student> mysqld --verbose --help

    아래와 같이 도움말을 상세하게 확인
    OS> mysql --verbose --help > mysql_help.txt
    OS> notepad mysql_help.txt

  MySQL 의 Socket 을 알아보자
  - http://jwprogramming.tistory.com/54
  - http://blog.naver.com/lyh1620/220817088671

SQL

Structured Query Language, 구조화 질의 어.

Relational DataBase System(RDBMS)의 데이터를 관리하기 위해 설계된 프로그래밍 언어

- Structured 는 '구조화된', '잘 짜인'이란 의미
- Query 는 '질의하는', '요청하는'이란 의미
- Language 는 언어 즉, 명령
- 즉, 요청을 잘 짜인 형태로 표현할 수 있는 언어

SQL 의 분류

- DML (Data Manipulation Language, 데이터 조작어)
  - select, insert, update, delete
  - select 는 원본 데이터를 변경하지 않는 데이터 조작어
- DCL (Data Control Language, 데이터 제어어)
  - grant, revoke...
- DDL (Data Definition Language, 데이터 정의어)
  - create, alter, drop, rename, truncate…
- TCL (Transaction Control Language, 트랜잭션 제어어)
  - commit, rollback, savepoint

NoSQL

Not Only SQL. 반드시 구조화/정형화된 RDBMS의 표 형태 데이터 만이 아닌 그 외 여러 다른 형태의 비정형 데이터들을 저장하기 위한 기술을 의미합니다. 대체로 Key-Value 형태의 데이터 구조를 가졌으나, 그 외 다른 여러 데이터 모델 구조도 사용됩니다. 최근 일반적인 RDBMS 제품들도 NoSQL 기술을 추가하고 있습니다. NoSQL DBMS 제품으로 MongoDB, HBase, Cassandra 와 같은 제품이 있습니다.

SELECT

SELECT 란?

검색/조회(X)

원하는 데이터 집합을 MySQL 서버에게 요청/묘사하는 언어← 우리의 정의

SELECT 의 기본 문형

```
SELECT * ¦ {[DISTINCT] column¦expression [alias],...}
FROM    table;
```

SELECT 구문의 7 가지 대표 절

SELECT 컬럼, …

FROM 테이블, 뷰, 인라인 뷰(파생 테이블)…

WHERE 조건식 …

GROUP BY …

HAVING …

ORDER BY 컬럼 ASC/DESC

LIMIT [offset,] 행 개수;

```
select last_name
from   employees;
```

```
select *
from   employees;
```

```
select last_name, salary
from   employees;
```

```
SELECT DATE_ADD(NOW(), INTERVAL 100 DAY);
```

연봉은 salary * 12 + salary *12 * commission_pct 일 경우라 할 때 모든 사원의 last_name 과 연봉 출력

NULL

Programming ?

Database ?

MySQL Beginner, ORACLE

Setting Data Types to NULL

NULL is a SQL keyword used to define data types that allow missing values. It means one of two things:

- Unknown: There is a value, but the precise value is not known at this time.
- Not applicable: If a value is specified, it would not be accurately representative.

Use **NULL**:

- In place of a real value in a number of situations
  - For example: “no value,” “unknown value,” “missing value,” “out of range,” “not applicable,” and so on
- To represent an empty query result

Plan for null values during database design.

ORACLE

```
SELECT CONCAT('I LOVE ' , ' YOU ' );

SELECT CONCAT('I LOVE ' ,null, ' YOU ' );
```

```
# Column Alias
# MySQL 은 backtic or single quotation 모두 사용 가능
# double quotation
select last_name as 이름
     , salary    급여
     , salary + 500 "500 달러인상"
     , salary * 1.1 as "10% 인상"
     , salary - 500 "10% 인상 salary * 12 + salary *12 * commission_pct"
    , salary /2
from   employees;

# double quotation
select last_name as 이름
     , salary    급여
     , salary + 500 '500 달러인상'
     , salary * 1.1 as '10% 인상'
     , salary - 500 '10% 인상 salary * 12 + salary *12 * commission_pct'
     , salary /2
from   employees;

# backtic
select last_name as 이름
     , salary    급여
     , salary + 500 `500 달러인상`
     , salary * 1.1 as `10% 인상`
     , salary - 500 `10% 인상 salary * 12 + salary *12 * commission_pct`
     , salary /2
from   employees;
```

DUAL table

Oracle DBMS 의 root 가 public 타입으로 가지고 있는 한 건의 레코드만 있는 테이블.

특정테이블의 값을 SELECT 하지 않는 경우 한 건의 데이터만 출력하고 싶을 때 SELECT~FROM 이라는 표준 SQL 구문 형식을 유지하기 위해 사용. 현재는 실제 쿼리에 dual 테이블을 드라이빙 하지는 않음.

1 부터 6 사이의 주사위 난수 발생

[ORACLE] select trunc(dbms_random.value(1,7)) from dual;

[MySQL]

```
select floor(rand()*6)+1 from dual;

select floor(rand()*6)+1;
```

ORDER BY

select 문 을 사용하여 테이블에서 데이터를 쿼리 할 때 결과 집합은 임의의 순서입니다.
결과 집합을 정렬하려면 문에 ORDER BY 절을 추가합니다.

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column1 [ASC¦DESC],
   column2 [ASC¦DESC],
   ...;
```

- ORDER BY 절 뒤에 정렬 할 하나 이상의 열을 지정합니다 .
- 오름차순은 ASC, DESC 는 내림차순을 의미합니다.
- ORDER BY 절에 ASC 혹은 DESC 가 없는 경우 오름차순 정렬합니다.

A) contactLastName 열의 값 을 기준으로 오름차순으로 정렬합니다 .

```
SELECT
	contactLastname,
	contactFirstname
FROM
	customers
ORDER BY
	contactLastname;
```

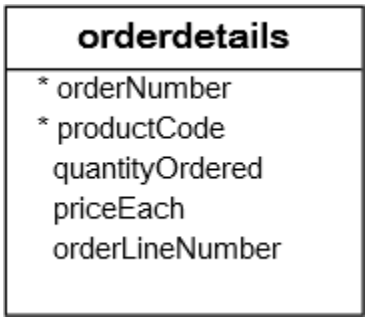
	contactLastname	contactFirstname
▶	Accorti	Paolo
	Altagar,G M	Raanan
	Andersen	Mel
	Anton	Carmen
	Ashworth	Rachel
	Barajas	Miguel
	Benitez	Violeta
	Bennett	Helen
	Berglund	Christina

B) 고객을 성을 기준으로 내림차순으로 정렬 한 다음 이름을 기준으로 오름차순으로 정렬

```
SELECT
	contactLastname,
	contactFirstname
FROM
	customers
ORDER BY
	contactLastname DESC,
	contactFirstname ASC;
```

	contactLastname	contactFirstname
▶	Young	Dorothy
	Young	Jeff
	Young	Julie
	Young	Mary
	Yoshido	Juri
	Walker	Brydey
	Victorino	Wendy
	Urs	Braun
	Tseng	Jerry

C)



orderdetails 테이블에서 orderNumber, orderlinenumber, quantityOrdered * priceEach 열을 출력하고 quantityOrdered * priceEach 열 기준 내림차순으로 결과 집합을 정렬

```
SELECT
    orderNumber,
    orderlinenumber,
    quantityOrdered * priceEach
FROM
    orderdetails
ORDER BY
   quantityOrdered * priceEach DESC;
```

orderNumber	orderlinenumber	quantityOrdered * priceEach
10403	9	11503.14
10405	5	11170.52
10407	2	10723.60
10404	3	10460.16
10312	3	10286.40
10424	6	10072.00
10348	8	9974.40
10405	3	9712.04
10196	5	9571.08

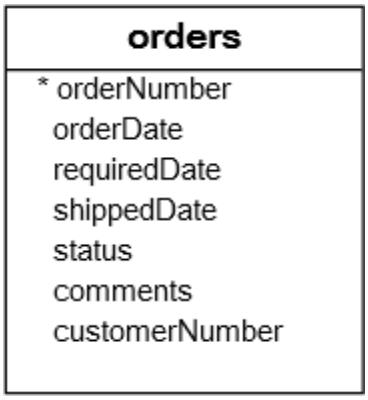
컬럼 ALIAS 를 사용

```
SELECT
    orderNumber,
    orderLineNumber,
    quantityOrdered * priceEach AS subtotal
FROM
    orderdetails
ORDER BY subtotal DESC;
```

orderNumber	orderLineNumber	subtotal
10403	9	11503.14
10405	5	11170.52
10407	2	10723.60
10404	3	10460.16
10312	3	10286.40
10424	6	10072.00
10348	8	9974.40
10405	3	9712.04
10196	5	9571.08
10206	6	9568.73
10304	6	9467.68

MySQL ORDER BY 을 사용하여 사용자 지정 목록을 사용하여 데이터 정렬

```
mysql> describe classicmodels.orders;
```



FIELD()함수를 사용하여 각 주문 상태를 숫자에 매핑 하고 함수 결과를 기준으로 결과를 정렬

```
SELECT
    orderNumber,
    status
FROM
    orders
ORDER BY
    FIELD(status,
        'In Process',
        'On Hold',
        'Cancelled',
        'Resolved',
        'Disputed',
        'Shipped');
```

orderNumber	status
10420	In Process
10421	In Process
10422	In Process
10423	In Process
10424	In Process
10425	In Process
10334	On Hold
10401	On Hold
10407	On Hold

FIELD(status, 'In Process', 'On Hold', 'Cancelled', 'Resolved', 'Disputed', 'Shipped');

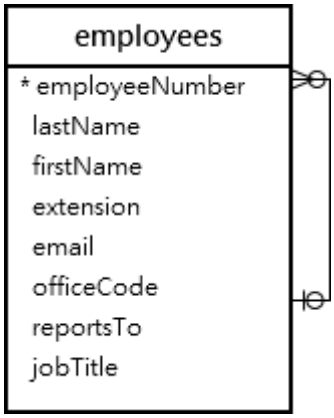
status 목록에서 의 인덱스를 반환합니다.
'In Process', 'On Hold', 'Cancelled', 'Resolved', 'Disputed', 'Shipped'.
예를 들어, status 가 In Process 인 경우 함수는 1 을 반환. On Hold 인 경우 함수는 2 를 반환.

WHERE 절

쿼리에서 반환 된 행에 대한 검색 조건을 지정

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

MySQL WHERE 절 예
mysql> desc classicmodels.employees;



1) WHERE 에 '=' 연산자 사용
직책이 Sales Rep 인 모든 직원을 쿼리.

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep';
```

lastname	firstname	jobtitle
Jennings	Leslie	Sales Rep
Thompson	Leslie	Sales Rep
Firrelli	Julie	Sales Rep
Patterson	Steve	Sales Rep
Tseng	Foon Yue	Sales Rep
Vanauf	George	Sales Rep
Bondur	Loui	Sales Rep

2) WHERE 절에 AND 연산자 사용

```
SELECT
    lastname,
    firstname,
    jobtitle,
    officeCode
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' AND
    officeCode = 1;
```

lastname	firstname	jobtitle	officeCode
Jennings	Leslie	Sales Rep	1
Thompson	Leslie	Sales Rep	1

3) OR 연산

```
SELECT
    lastName,
    firstName,
    jobTitle,
    officeCode
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' OR
    officeCode = 1
ORDER BY
    officeCode ,
    jobTitle;
```

lastName	firstName	jobTitle	officeCode
Murphy	Diane	President	1
Bow	Anthony	Sales Manager (NA)	1
Jennings	Leslie	Sales Rep	1
Thompson	Leslie	Sales Rep	1
Firrelli	Jeff	VP Marketing	1
Hill	Mary	VP Sales	1
Firrelli	Julie	Sales Rep	2
Patterson	Steve	Sales Rep	2
Tseng	Foon Yue	Sales Rep	3
Vanauf	George	Sales Rep	3
Bondur	Loui	Sales Rep	4
Hernandez	Gerard	Sales Rep	4
Castillo	Pamela	Sales Rep	4
Gerard	Martin	Sales Rep	4
Nishi	Mami	Sales Rep	5
Kato	Yoshimi	Sales Rep	5
Fixter	Andy	Sales Rep	6
Marsh	Peter	Sales Rep	6
King	Tom	Sales Rep	6
Bott	Larry	Sales Rep	7
Jones	Barry	Sales Rep	7

4) BETWEEN AND 연산
officeCode 가 1 이상 3 이하인 사무실에서 근무하는 직원을 쿼리.

```
SELECT
    firstName,
    lastName,
    officeCode
FROM
    employees
WHERE
    officeCode BETWEEN 1 AND 3
ORDER BY officeCode;
```

firstName	lastName	officeCode
Diane	Murphy	1
Mary	Hill	1
Jeff	Firrelli	1
Anthony	Bow	1
Leslie	Jennings	1
Leslie	Thompson	1
Julie	Firrelli	2
Steve	Patterson	2
Foon Yue	Tseng	3
George	Vanauf	3

5) LIKE 연산
LIKE 패턴을 형성하려면 %및 _와일드 카드 를 사용
% : 0 개 이상 문자의 문자열과 일치

_ : 모든 단일 문자와 일치

lastName 이 문자열 ‘ son ’ 으로 끝나는 직원을 쿼리.

```
SELECT
    firstName,
    lastName
FROM
    employees
WHERE
    lastName LIKE '%son'
ORDER BY firstName;
```

	firstName	lastName
▶	Leslie	Thompson
	Steve	Patterson
	William	Patterson

6) IN 연산

officeCode 가 1, 2, 3 인 사원을 쿼리.

```
SELECT
    firstName,
    lastName,
    officeCode
FROM
    employees
WHERE
    officeCode IN (1, 2, 3)
ORDER BY
    officeCode;
```

	firstName	lastName	officeCode
▶	Diane	Murphy	1
	Mary	Hill	1
	Jeff	Firrelli	1
	Anthony	Bow	1
	Leslie	Jennings	1
	Leslie	Thompson	1
	Julie	Firrelli	2
	Steve	Patterson	2
	Foon Yue	Tseng	3
	George	Vanauf	3

7) IS NULL

```
SELECT
    lastName,
    firstName,
```

```
    reportsTo
FROM
    employees
WHERE
    reportsTo IS NULL;
```

	lastName	firstName	reportsTo
▶	Murphy	Diane	NULL

8) IS NOT NULL

9) WHERE 관계 연산자
다음 표는 WHERE 절 에서 표현식을 구성하는 데 사용할 수있는 6 가지 관계 연산자를 보여줍니다.

연산자	설명
=	같다. 거의 모든 데이터 유형과 함께 사용할 수 있다.
<> 또는 !=	다르다
<	작다. 일반적으로 숫자 및 날짜 / 시간 데이터 유형과 함께 사용.
>	크다
<=	작거나 같디
>=	크거나 같다

같지 않음(<>) 연산자를 사용하여 Sales Rep 이 아닌 모든 직원을 쿼리.

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees
WHERE
    jobtitle <> 'Sales Rep';
```

	lastname	firstname	jobtitle
▶	Murphy	Diane	President
	Patterson	Mary	VP Sales
	Firrelli	Jeff	VP Marketing
	Patterson	William	Sales Manager (APAC)
	Bondur	Gerard	Sale Manager (EMEA)
	Bow	Anthony	Sales Manager (NA)

officecode 가 5 보다 큰 직원을 쿼리.

```
SELECT
    lastname,
    firstname,
    officeCode
FROM
    employees
WHERE
    officecode > 5;
```

	lastname	firstname	officeCode
▶	Patterson	William	6
	Bott	Larry	7
	Jones	Barry	7
	Fixter	Andy	6
	Marsh	Peter	6
	King	Tom	6

다음 쿼리는 officecode 가 4 (<= 4) 이하인 직원을 쿼리.

```
SELECT
    lastname,
    firstname,
    officeCode
FROM
    employees
WHERE
    officecode <= 4;
```

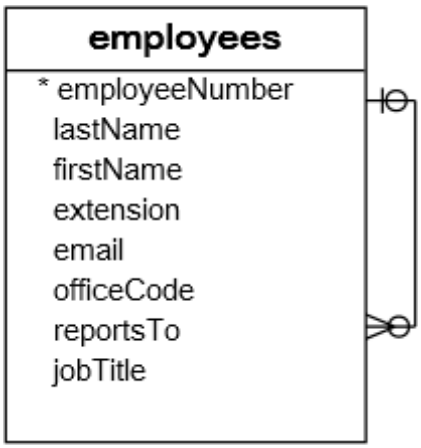
	lastname	firstname	officeCode
▶	Murphy	Diane	1
	Patterson	Mary	1
	Firrelli	Jeff	1
	Bondur	Gerard	4

DISTINCT

데이터를 쿼리 할 때 중복 행을 제거시 DISTINCT 절을 사용.

```
SELECT DISTINCT
    select_list
FROM
    table_name;
```

```
mysql> desc classicmodels.employees;
```




```
SELECT
    lastname
FROM
    employees
ORDER BY
    lastname;
```

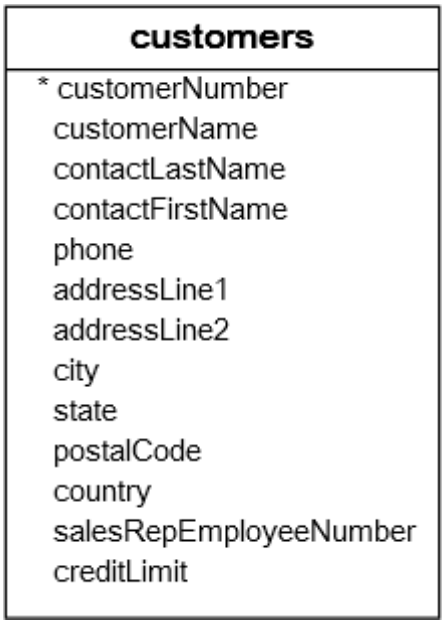
lastname
Bondur
Bondur
Bott
Bow
Castillo
Firrelli
Firrelli
Fixter
Gerard
Hernandez
Jennings

출력에서 볼 수 있듯이 일부 직원은 성이 같습니다 (예 : Bondur, Firrelli.)
이 문은 DISTINCT 절을 사용 하여 중복을 제거할 수 있습니다.

```
SELECT
    DISTINCT lastname
FROM
    employees
ORDER BY
    lastname;
```

	lastname
	Bondur
	Bott
	Bow
	Castillo
	Firrelli
	Fixter
	Gerard
	Hernandez
	Jennings
	Jones

DISTINCT 를 컬럼에 사용하는 경우 MySQL 은 모든 값을 동일한 값으로 취급 NULL 하므로 하나의 값만 유지합니다.




```
SELECT DISTINCT state
FROM customers;
```

	state
	NULL
	NV
	Victoria
	CA
	NY
	PA
	CT
	MA
	Osaka
	BC
	Québec
	Isle of Wight
	NSW
	NJ

DISTINCT 여러 열이 있는 MySQL
DISTINCT 둘 이상의 열과 함께 절을 사용할 수 있습니다. 이 경우 MySQL 은 이러한 열의 값 조합을 사용하여 결과 집합에 있는 행의 고유성을 확인합니다.
예를 들어, customers 테이블 에서 도시와 주의 고유 한 조합을 가져 오려면 다음 쿼리를 사용합니다.

```
SELECT DISTINCT
    state, city
FROM
    customers
WHERE
    state IS NOT NULL
ORDER BY
    state,
    city;
```

	state	city
	BC	Tsawassen
	BC	Vancouver
	CA	Brisbane
	CA	Burbank
	CA	Burlingame
	CA	Glendale
	CA	Los Angeles
	CA	Pasadena
	CA	San Diego
	CA	San Francisco
	CA	San Jose
	CA	San Rafael
	Co. Cork	Cork
	CT	Bridgewater
	CT	Glendale

DISTINCT가 없을 경우 다음과 같이 중복 조합을 얻게 됩니다.

```
SELECT
    state, city
FROM
    customers
WHERE
    state IS NOT NULL
ORDER BY
    state ,
    city;
```

state	city
BC	Tsawassen
BC	Vancouver
CA	Brisbane
CA	Burbank
CA	Burlingame
CA	Glendale
CA	Los Angeles
CA	Pasadena
CA	San Diego
CA	San Francisco
CA	San Francisco
CA	San Jose
CA	San Rafael
Co. Cork	Cork
CT	Bridgewater
CT	Glendale

```
SELECT
    state
FROM
    customers
GROUP BY state;

SELECT DISTINCT
    state
FROM
    customers;
```

state
NULL
NV
Victoria
CA
NY
PA

MySQL 8.0 은 GROUP BY 절에 대한 묵시적 정렬을 제거했습니다. 8.0 미만 버전은 GROUP BY 절은 정렬 될 수 있습니다.
정렬이 꼭 필요할 경우 명시적으로 정렬해야 합니다.

```
SELECT DISTINCT
    state
FROM
```

```
    customers
ORDER BY
    state;
```

DISTINCT with LIMIT 절
MySQL 은 LIMIT 절에 지정된 고유 행 수를 찾으면 즉시 검색을 중지합니다 .
다음 쿼리는 customers 테이블 에서 Null 이 아닌 처음 5 개의 고유 상태를 선택 합니다.

```
SELECT DISTINCT
    state
FROM
    customers
WHERE
    state IS NOT NULL
LIMIT 5;
```

	state
▶	NV
	Victoria
	CA
	NY
	PA

OR 연산자

OR 연산자는 두 개의 부울 표현식을 결합하고 두 조건 중 하나가 참이면 참을 반환.

```
SELECT FALSE OR FALSE;
SELECT FALSE OR TRUE;
SELECT NULL OR TRUE;
SELECT NULL OR NULL;
```

OR 연산 진리표

OR	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

```
SELECT 1 = 1 OR 1 / 0;
```

표현식 1 = 1 은 항상 true 를 반환하기 때문에 MySQL 은 1 / 0 을 평가하지 않습니다.

연산자 우선 순위

```
SELECT true OR false AND false;

true OR false AND false
-----------------------
1
```

평가 순서를 변경하려면 다음과 같이 괄호를 사용.

```
SELECT (true OR false) AND false;

(true OR false) AND false
-------------------------
0

mysql> desc classicmodels.customers;
```

미국 또는 프랑스에 거주하는 고객을 쿼리

```
SELECT
        customername,
        country
FROM
        customers
WHERE country = 'USA' OR
      country = 'France';
```

	customername	country
▶	Atelier graphique	France
	Signal Gift Stores	USA
	La Rochelle Gifts	France
	Mini Gifts Distributors Ltd.	USA
	Mini Wheels Co.	USA
	Land of Toys Inc.	USA
	Saveley & Henriot, Co.	France
	Muscle Machine Inc	USA
	Diecast Classics Inc.	USA
	Technics Stores Inc.	USA
	American Souvenirs Inc	USA
	Daedalus Designs Imports	France

미국 또는 프랑스에 거주하고 신용 한도가 100,000 을 초과하는 고객을 쿼리.

```
SELECT
        customername,
        country,
```

```
        creditLimit
FROM
        customers
WHERE (country = 'USA'
                OR country = 'France')
          AND creditlimit > 100000;
```

	customername	country	creditLimit
	La Rochelle Gifts	France	118200
	Mini Gifts Distributors Ltd.	USA	210500
	Land of Toys Inc.	USA	114900
	Saveley & Henriot, Co.	France	123900
	Muscle Machine Inc	USA	138500
	Diecast Classics Inc.	USA	100600
	Collectable Mini Designs Co.	USA	105000
	Marta's Replicas Co.	USA	123700

괄호를 사용하지 않는 경우 쿼리는 신용 한도가 10,000 보다 큰 미국에 있는 고객 또는 프랑스에 있는 고객을 반환.

```
SELECT
        customername,
        country,
        creditLimit
FROM
        customers
WHERE country = 'USA'
                OR country = 'France'
                AND creditlimit > 10000;
```

	customername	country	creditLimit
	Signal Gift Stores	USA	71800
	La Rochelle Gifts	France	118200
	Mini Gifts Distributors Ltd.	USA	210500
	Mini Wheels Co.	USA	64600
	Land of Toys Inc.	USA	114900
	Saveley & Henriot, Co.	France	123900
	Muscle Machine Inc	USA	138500
	Diecast Classics Inc.	USA	100600
	Technics Stores Inc.	USA	84600

AND 연산자

```
SELECT FALSE AND FALSE;
SELECT FALSE AND TRUE;
SELECT NULL AND TRUE;
SELECT NULL AND NULL;
```

AND 연산 진리표

AND	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

```
SELECT 1/0;
```

MySQL 에서 0 은 거짓, 0 이 아닌 것은 참 입니다.

```
SELECT 1 = 0 AND 1 / 0 ;
```

MySQL 1 = 0 은 표현식 의 첫 부분 만 평가합니다. 표현식 1 = 0 이 거짓을 반환 하기 때문에 MySQL 은 전체 표현식의 결과를 거짓으로 결론을 내릴 수 있습니다. MySQL 은 표현식의 나머지 부분 인 1/0 을 평가하지 않습니다.

```
mysql> desc classicmodels.customers
```

다음 명령문은 AND 연산자를 사용하여 미국 캘리포니아 (CA)에 거주하는 고객을 찾습니다.

```
SELECT
    customername,
    country,
    state
FROM
    customers
WHERE
    country = 'USA' AND state = 'CA';
```

customername	country	state
Mini Gifts Distributors Ltd.	USA	CA
Mini Wheels Co.	USA	CA
Technics Stores Inc.	USA	CA
Toys4GrownUps.com	USA	CA
Boards & Toys Co.	USA	CA
Collectable Mini Designs Co.	USA	CA
Corporate Gift Ideas Co.	USA	CA
Men 'R' US Retailers, Ltd.	USA	CA
The Sharp Gifts Warehouse	USA	CA
West Coast Collectables Co.	USA	CA
Signal Collectibles Ltd.	USA	CA

AND 연산자를 사용하여 두 개 이상의 부울 식을 결합 할 수 있습니다.
다음 쿼리는 미국 캘리포니아에 거주하고 신용 한도가 10 만 보다 큰 고객을 반환합니다.

```
SELECT
       customername,
       country,
       state,
       creditlimit
FROM
       customers
WHERE country = 'USA'
       AND state = 'CA'
       AND creditlimit > 100000;
```

customername	country	state	creditlimit
Mini Gifts Distributors Ltd.	USA	CA	210500
Collectable Mini Designs Co.	USA	CA	105000
Corporate Gift Ideas Co.	USA	CA	105000

IN 연산자

IN 연산자 의 구문

```
SELECT
    column1,column2,...
FROM
    table_name
WHERE
       (expr¦column_1) IN ('value1','value2',...);
```

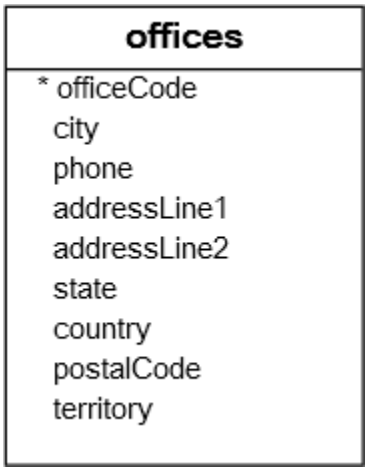
- WHERE 절에서 column 또는 표현식과 IN 다음에 원하는 값을 쉼표 (,)로 구분.
  IN 의 값이 경우 작업자는 1 을 리턴 column_1 하거나 결과 expr 식은 리스트의 값과 동일하고, 그렇지 않으면 0 을 반환한다.

목록의 값이 모두 상수이면 MySQL 은 다음 단계를 수행합니다.

- 1. 컬럼 또는 표현식의 값을 평가
- 2. 값을 정렬
- 3. 이진 검색 알고리즘을 사용하여 값을 검색, 값이 존재하지 않으면 NULL 반환
-

```
select now() from dual where null in (null, 1);
select now() from dual where 1 in (null, 1);
```

```
mysql> desc classicmodels.offices
```



미국과 프랑스에있는 사무실을 찾으려면 IN 연산자를 다음 쿼리로 사용할 수 있습니다 .

```
SELECT
    officeCode,
    city,
    phone,
    country
```

```
FROM
    offices
WHERE
    country IN ('USA' , 'France');
```

	officeCode	city	phone	country
	1	San Francisco	+1 650 219 4782	USA
	2	Boston	+1 215 837 0825	USA
	3	NYC	+1 212 555 3000	USA
	4	Paris	+33 14 723 4404	France

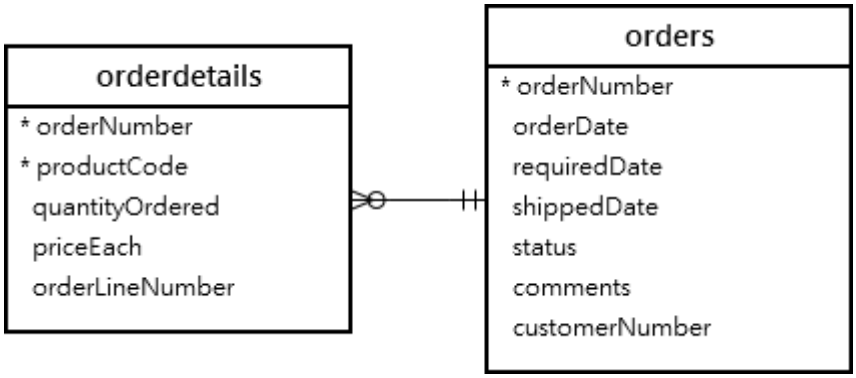
```
SELECT
    officeCode,
    city,
    phone
FROM
    offices
WHERE
    country = 'USA' OR country = 'France';
```

목록에 많은 값이 있는 경우 IN 연산자를 사용하면 쿼리를 줄이고 더 읽기 쉽게 만들 수 있습니다.

미국과 프랑스에 위치하지 않는 사무실을 쿼리 할 경우.

```
SELECT
    officeCode,
    city,
    phone
FROM
    offices
WHERE
    country NOT IN ('USA' , 'France');
```

	officeCode	city	phone	country
	5	Tokyo	+81 33 224 5000	Japan
	6	Sydney	+61 2 9264 2451	Australia
	7	London	+44 20 7877 2041	UK



60000 달러 이상 주문배송이 완료된 주문번호와 구매 고객, 배송완료일 확인

```
SELECT
	orderNumber,
	customerNumber,
	status,
	shippedDate
FROM
	orders
WHERE orderNumber IN
(
	 SELECT
		orderNumber
	 FROM
		orderDetails
	 GROUP BY
		orderNumber
	 HAVING SUM(quantityOrdered * priceEach) > 60000
);
```

	orderNumber	customerNumber	status	shippedDate
	10165	148	Shipped	2003-12-26
	10287	298	Shipped	2004-09-01
	10310	259	Shipped	2004-10-18

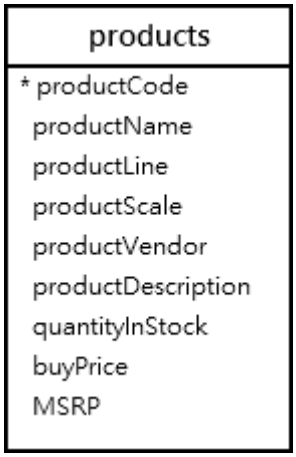
BETWEEN 연산자

BETWEEN 연산자는 범위를 지정할 수 있는 논리 연산자입니다. SELECT, UPDATE, DELETE 의 WHERE 절에 사용됩니다.

BETWEEN 연산자의 구문
expr [NOT] BETWEEN begin_expr AND end_expr;

expr, begin_expr, end_expr 는 데이터 타입이 일치해야 합니다 .
expr 은 begin_expr 이상 end_expr 이하 값이어야 합니다. 그렇지 않으면 0 을 반환.
begin_expr 보다 end_expr 의 값이 커야 합니다. 그렇지 않으면 0 을 반환.
표현식이 NULL 이면 BETWEEN 연산자는 NULL 을 반환.

mysql> desc classicmodels.products;



다음 쿼리는 BETWEEN 연산자를 사용하여 구매 가격이 90~100 사이인 제품을 찾습니다.

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice BETWEEN 90 AND 100;
```

	productCode	productName	buyPrice
	S10_1949	1952 Alpine Renault 1300	98.58
	S10_4698	2003 Harley-Davidson Eagle Drag Bike	91.02
	S12_1099	1968 Ford Mustang	95.34
	S12_1108	2001 Ferrari Enzo	95.59
	S18_1984	1995 Honda Civic	93.89
	S18_4027	1970 Triumph Spitfire	91.92
	S24_3856	1956 Porsche 356A Coupe	98.3

이 쿼리는 연산자 대신 크거나 같음 (>=) 및 작거나 같음 (<=) 연산자를 사용 BETWEEN 과 동일한 결과를 얻습니다.

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice >= 90 AND buyPrice <= 100;
```

구매 가격이 $20 에서 $100 사이가 아닌 제품을 찾으려면 다음과 같이 NOT BETWEEN 연산자를 사용합니다.

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice NOT BETWEEN 20 AND 100;
```

	productCode	productName	buyPrice
	S10_4962	1962 LanciaA Delta 16V	103.42
	S18_2238	1998 Chrysler Plymouth Prowler	101.51
	S24_2840	1958 Chevy Corvette Limited Edition	15.91
	S24_2972	1982 Lamborghini Diablo	16.24

위의 쿼리를 다시 작성할 수 있습니다 .

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
```

```
WHERE
    buyPrice < 20 OR buyPrice > 100;
```

2) BETWEEN 날짜와 함께 MySQL 사용 예제
날짜 타입에 BETWEEN 연산자를 사용하는 경우, 최상의 결과를 얻으려면 cast function 을 사용하여 명시적으로 열 또는 표현식 유형을 날짜 타입으로 변환합니다.

다음 예는 requireddate 가 2003 년 1 월 1 일부터 2003 년 1 월 31 일 사이에 속한 주문을 쿼리합니다.

```
SELECT
   orderNumber,
   requiredDate,
   status
FROM
   orders
WHERE
   requireddate BETWEEN
     CAST('2003-01-01' AS DATE) AND
     CAST('2003-01-31' AS DATE);
```

	orderNumber	requiredDate	status
	10100	2003-01-13	Shipped
	10101	2003-01-18	Shipped
	10102	2003-01-18	Shipped

```
SELECT
   orderNumber,
   requiredDate,
   status
FROM
   orders
WHERE
   requireddate BETWEEN
     '2003-01-01' AND '2003-01-31';
```

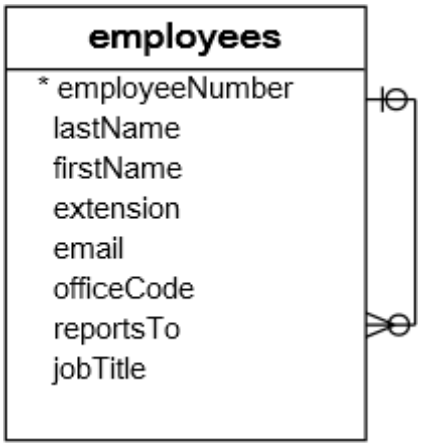
LIKE 연산자
LIKE 연산자는 검사 문자열에 특정 패턴을 포함하는 논리 연산자이다.
LIKE 연산자 의 구문은 다음과 같습니다.

```
expression LIKE pattern ESCAPE escape_character
```

MySQL 은 패턴을 구성하기 위해 %와 _의 두 가지 와일드 카드 문자를 제공합니다.

- (%) 와일드 카드는 0 개 이상의 문자로 구성된 문자열과 일치합니다.
- (_) 와일드 카드는 단일 문자와 일치합니다.

```
mysql> desc classicmodels.employees;
```




```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    firstName LIKE 'a%';
```

	employeeNumber	lastName	firstName
	1143	Bow	Anthony
	1611	Fixter	Andy

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastName LIKE '%on';
```

	employeeNumber	lastName	firstName
	1056	Patterson	Mary
	1088	Patterson	William
	1166	Thompson	Leslie
	1216	Patterson	Steve

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
```

```
    employees
WHERE
    lastname LIKE '%on%';
```

employeeNumber	lastName	firstName
1056	Patterson	Mary
1088	Patterson	William
1102	Bondur	Gerard
1166	Thompson	Leslie
1216	Patterson	Steve
1337	Bondur	Loui
1504	Jones	Barry

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    firstname LIKE 'T_m';
```

employeeNumber	lastName	firstName
1619	King	Tom

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastName NOT LIKE 'B%';
```

employeeNumber	lastName	firstName
1002	Murphy	Diane
1056	Patterson	Mary
1076	Firrelli	Jeff
1088	Patterson	William
1165	Jennings	Leslie
1166	Thompson	Leslie
1188	Firrelli	Julie
1216	Patterson	Steve
1286	Tseng	Foon Yue
1323	Vanauf	George

패턴은 대소 문자를 구분하지 않으므로 b%또는 B%패턴은 동일한 결과를 반환합니다.
때로는 일치시키려는 패턴에 와일드 카드 문자 (예: 10%, _20)가 포함됩니다.
이 경우 이스케이프 문자를 지정할 수 있습니다. 이스케이프 문자를 명시적으로
지정하지 않으면 백 슬래시 문자 \가 기본 이스케이프 문자입니다.

```
SELECT
    productCode,
    productName
FROM
    products
WHERE
    productCode LIKE '%\_20%';
```

또는 다음 절 $을 사용하여 명시적으로 이스케이프 문자를 지정할 수 있습니다

```
SELECT
    productCode,
    productName
FROM
    products
WHERE
    productCode LIKE '%$_20%' ESCAPE '$';
```

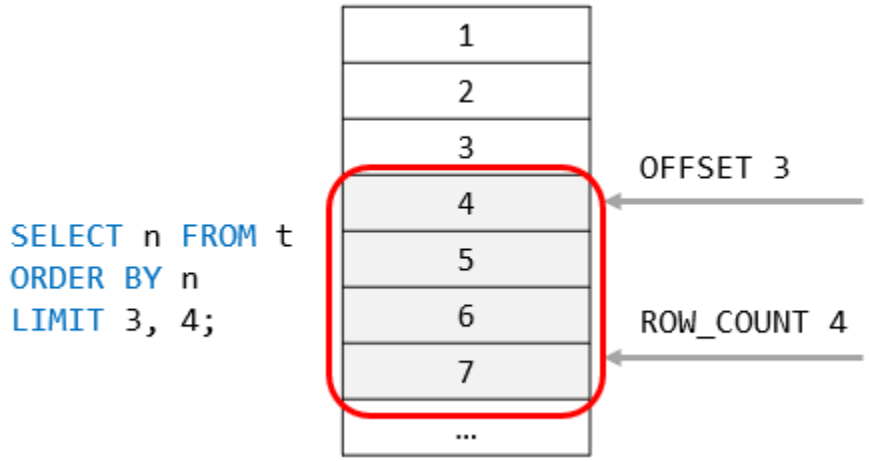
	productCode	productName
	S10_2016	1996 Moto Guzzi 1100i
	S24_2000	1960 BSA Gold Star DBD34
	S24_2011	18th century schooner
	S24_2022	1938 Cadillac V-16 Presidential Limousine
	S700_2047	HMS Bounty

이 경우 \문자를 ESCAPE 로 명시적인 지정을 하면 에러입니다.

LIMIT 절

SELECT 문장에서 LIMIT 절은 반환 할 행 수를 제한하기 위해 사용됩니다.
LIMIT 절은 하나 또는 두 개의 인수를 허용합니다. 두 인수의 값은 0 또는 양의 정수이어야 합니다 .
다음은 LIMIT 두 개의 인수가 있는 절 구문을 보여줍니다 .

```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
```



출력되는 데이터는 행은 0 부터 이므로 LIMIT 3, 4 는 4 번째 행부터 4 개의 행을 가져옵니다.

다음 두 절은 동일합니다.

```
LIMIT row_count;
LIMIT 0 , row_count;
```

MySQL 은 PostgreSQL LIMIT 과의 호환성을 위해 OFFSET 대체 절을 제공합니다.
(오라클은 12C 부터 offset 과 fetch 사용)

```
LIMIT row_count OFFSET offset
```

SELECT 문에 ORDER BY 절이 없다면 지정되지 않은 순서로 행을 반환합니다.
항상 LIMIT 절과 함께 ORDER BY 절을 사용하여 1. 결과 행을 원하는 순서로 정렬 후 2. 원하는 데이터 집합을 가져오도록 합니다.

```
SELECT select_list
FROM table_name
ORDER BY order_expression
LIMIT offset, row_count;
```

SELECT 명령문 기술 순서가 아닌 MySQL 서버에서 명령문 처리 순서.


```
mysql> desc classicmodels.customers;
```

신용도 상위 5 명의 고객을 출력.

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY creditLimit DESC
LIMIT 5;
```

	customernumber	customername	creditlimit
▶	141	Euro+ Shopping Channel	227600
	124	Mini Gifts Distributors Ltd.	210500
	298	Vida Sport, Ltd	141300
	151	Muscle Machine Inc	138500
	187	AV Stores, Co.	136800

신용도 하위 5 명의 고객을 출력.

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
```

```
    customers
ORDER BY creditLimit
LIMIT 5;
```

	customernumber	customername	creditlimit
▶	223	Natürlich Autos	0
	168	American Souvenirs Inc	0
	169	Porto Imports Co.	0
	206	Asian Shopping Network, Co	0
	125	Havel & Zbyszek Co	0

신용도 0 인 고객이 5 명 이상이므로 위 쿼리의 결과가 일치하지 않을 수 있습니다.

LIMIT 페이지 매김을 위해 MySQL 사용
애플리케이션에 데이터를 표시 할 때 행을 페이지로 나누고 싶을 때가 있습니다. 각 페이지에는 5, 10, 20 과 같은 특정 수의 행이 포함됩니다.
페이지 수를 계산하려면 총 행을 페이지 당 행 수로 나눈 값을 얻습니다. 특정 페이지의 행을 가져 오기 위해 LIMIT 절을 사용할 수 있습니다.
이 쿼리는 COUNT(*) 를 사용 하여 customers 테이블 에서 총 레코드 수를 가져옵니다.

```
SELECT COUNT(*) FROM customers;
+----------+
¦ COUNT(*) ¦
+----------+
¦      122 ¦
+----------+
1 row in set (0.00 sec)
```

각 페이지에 10 개의 행이 있다고 가정하여 122 명의 고객을 표시하기 위해 13 개의 페이지가 있습니다.
마지막 13 번째 페이지에는 두 개의 행만 있습니다.
이 쿼리는 LIMIT 절을 사용하여 고객 이름 순으로 정렬된 처음 10 명의 고객을 포함하는 1 페이지를 가져옵니다.

```
SELECT
    customerNumber,
    customerName
FROM
    customers
ORDER BY customerName
LIMIT 10;
```

customerNumber	customerName
242	Alpha Cognac
168	American Souvenirs Inc
249	Amica Models & Co.
237	ANG Resellers
276	Anna's Decorations, Ltd
465	Anton Designs, Ltd.
206	Asian Shopping Network, Co
348	Asian Treasures, Inc.
103	Atelier graphique
471	Australian Collectables, Ltd

이 쿼리는이 LIMIT 절을 사용하여 11-20 행을 포함하는 두 번째 페이지의 행을 가져옵니다.

```
SELECT
    customerNumber,
    customerName
FROM
    customers
ORDER BY customerName
LIMIT 10, 10;
```

customerNumber	customerName
114	Australian Collectors, Co.
333	Australian Gift Network, Co
256	Auto Associés & Cie.
406	Auto Canal+ Petit
198	Auto-Moto Classics Inc.
187	AV Stores, Co.
121	Baane Mini Imports
415	Bavarian Collectables Imports, Co.
293	BG&E Collectables
128	Blauer See Auto, Co.

MySQL LIMIT 을 사용하여 n 번째 최고 또는 최저 값 얻기
n 번째로 높거나 낮은 값을 얻으려면 다음 LIMIT 절을 사용합니다.

```
SELECT select_list
FROM table_name
ORDER BY sort_expression
LIMIT n-1, 1;
```

예를 들어 다음은 두 번째로 높은 신용을 가진 고객을 찾습니다.

```
SELECT
    customerName,
    creditLimit
FROM
```

```
    customers
ORDER BY
    creditLimit DESC
LIMIT 1,1;
```

	customerName	creditLimit
▶	Mini Gifts Distributors Ltd.	210500.00

IS NULL 연산자

NULL 값 여부를 테스트하려면 연산자 IS NULL 을 사용합니다.
IS NULL 연산자의 기본 구문은 다음과 같습니다.

```
value IS NULL
```

값이 NULL 이면 식은 true 를 반환합니다. 그렇지 않으면 false 를 반환합니다.

MySQL 은 true 는 1, false 는 0 을 의미합니다.
IS NULL 는 비교 연산자 이기 때문에 연산자를 사용할 수있는 모든 곳에서 사용할 수 있습니다.

```
SELECT 1 IS NULL,  -- 0
       0 IS NULL,  -- 0
       NULL IS NULL; -- 1
```

값이 NULL 이 아닌지 확인하려면 연산자 IS NOT NULL 을 사용합니다.
value IS NOT NULL

이 표현식은 NULL 값이 아닌 경우 true (1)를 반환합니다. 그렇지 않으면 false (0)를 반환합니다.

```
SELECT 1 IS NOT NULL, -- 1
       0 IS NOT NULL, -- 1
       NULL IS NOT NULL; -- 0
```

mysql> desc classicmodels.customers;

영업 담당자가 없는 고객을 찾습니다.

```
SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NULL
ORDER BY
    customerName;
```

customerName	country	salesrepemployeenumber
ANG Resellers	Spain	NULL
Anton Designs, Ltd.	Spain	NULL
Asian Shopping Network, Co	Singapore	NULL
Asian Treasures, Inc.	Ireland	NULL
BG&E Collectables	Switzerland	NULL
Cramer Spezialit?ten, Ltd	Germany	NULL
Der Hund Imports	Germany	NULL
Feuer Online Stores, Inc	Germany	NULL
Franken Gifts, Co	Germany	NULL

IS NOT NULL 연산자를 사용하여 영업 담당자가있는 고객을 가져옵니다.

```
SELECT
    customerName,
    country,
    salesrepemployeenumber
FROM
    customers
WHERE
    salesrepemployeenumber IS NOT NULL
ORDER BY
   customerName;
```

customerName	country	salesrepemployeenumber
Alpha Cognac	France	1370
American Souvenirs Inc	USA	1286
Amica Models & Co.	Italy	1401
Anna's Decorations, Ltd	Australia	1611
Atelier graphique	France	1370
Australian Collectables, Ltd	Australia	1611
Australian Collectors, Co.	Australia	1611
Australian Gift Network, Co	Australia	1611
Auto Associ?s & Cie.	France	1370

MySQL 은 IS NULL 연산자의 일부 특수 기능을 지원합니다.
날짜 '0000-00-00'처리

projects 테이블을 만듭니다.

```
CREATE TABLE IF NOT EXISTS projects (
    id INT AUTO_INCREMENT,
    title VARCHAR(255),
    begin_date DATE NOT NULL,
    complete_date DATE NOT NULL,
    PRIMARY KEY(id)
);
```

프로젝트 테이블에 일부 행을 삽입합니다.
0000-00-00 00 월 00 일 형태의 날짜는 존재할 수 없으며 알 수 없는 값으로 처리됩니다.

```
INSERT INTO projects(title,begin_date, complete_date)
VALUES('New CRM','2020-01-01','0000-00-00'),
      ('ERP Future','2020-01-01','0000-00-00'),
      ('VR','2020-01-01','2030-01-01');
```

complete_date 열이 null 인 값을 검색합니다.

```
SELECT *
FROM projects
WHERE complete_date IS NULL;
```

id	title	begin_date	complete_date
1	New CRM	2020-01-01	0000-00-00
2	ERP Future	2020-01-01	0000-00-00

JOIN

JOIN CASE	KEYWORD
equi join	join ~ on • left, right • join 은 outer 생략 가능 • outer join 이 아닌 경우 명시적으로 inner 를 붙인다. natural join ← 사용하지 말자!!!! join ~ using cross join
non equi join	
outer join left outer join right outer join full outer join (X)	
self join	
threeways join	

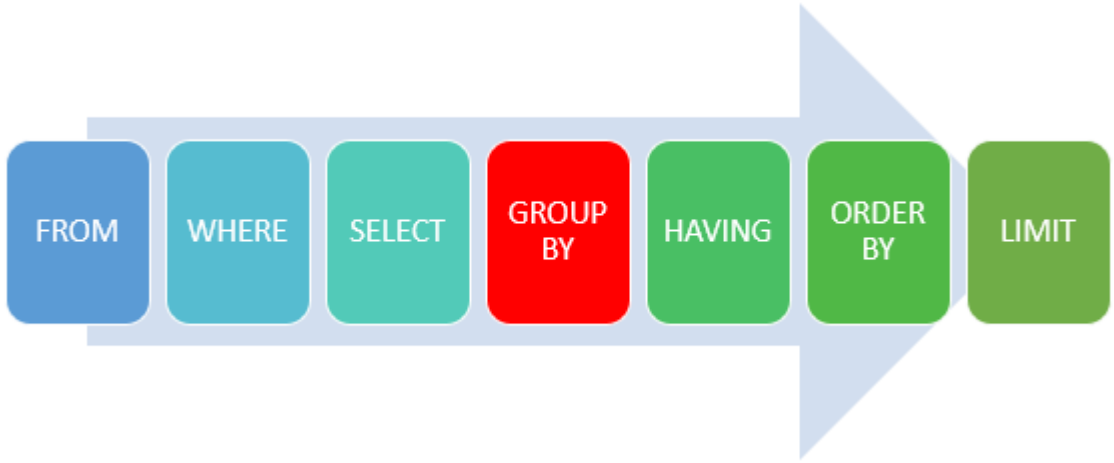
아래 내용들을 출력하는 threeways join 을 만드세요.

```
n1 계정
모든 사원의
사원명(employees.last_name),
업무 (jobs.job_title),
급여 (employees.salary),
사원등급 (job_grades.grade_level),
부서명(departments.department_name)
부서가 위치한 도시(locations.city)
도시가 위치한 나라를 출력(countries.country_name)
나라가 위치한 지역(regions.region_name)

사원명 ¦업무 ¦급여¦ 등급 ¦사수명 ¦부서명 ¦ 도시 ¦ 나라 ¦ 지역
```

GROUP BY 절

GROUP BY 절은 열을 기준으로 행 집합에 집계함수를 적용하여 그룹화합니다. GROUP BY 절은 각 그룹에 대해 하나의 행을 반환합니다. 즉, 결과 집합의 행 수를 줄입니다. 구문은 아래와 같습니다.

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```




```
mysql> desc classicmodels.orders
```

orders
* orderNumber
orderDate
requiredDate
shippedDate
status
comments
customerNumber

```
SELECT
    status
FROM
    orders
GROUP BY status;
```

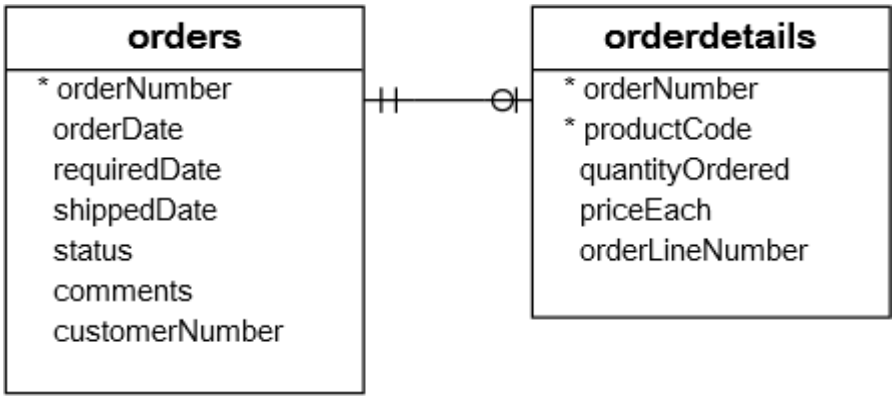
	status
▶	Cancelled
	Disputed
	In Process
	On Hold
	Resolved
	Shipped

```
SELECT DISTINCT
    status
FROM
    orders;
```

GROUP BY 집계 함수와 함께 사용
집계 함수는 행 집합의 계산을 수행하고 단일 값을 반환 할 수 있습니다. 이 GROUP BY 절은 대체로 집계 함수와 함께 사용되어 계산을 수행하고 각 하위 그룹에 대해 단일 값을 반환합니다.
예를 들어 각 상태의 주문 수를 알고 싶다면 GROUP BY 절과 COUNT() 함수를 함께 사용할 수 있습니다.

```
SELECT
    status, COUNT(*)
FROM
    orders
GROUP BY status;
```

	status	COUNT(*)
▶	Cancelled	6
	Disputed	3
	In Process	6
	On Hold	4
	Resolved	4
	Shipped	303



다음은 status 별 주문의 총 금액을 계산하는 쿼리입니다.

```
SELECT
    status,
    SUM(quantityOrdered * priceEach) AS amount
FROM
    orders
INNER JOIN orderdetails
    USING (orderNumber)
GROUP BY
    status;
```

	status	amount
▶	Cancelled	238854.18
	Disputed	61158.78
	In Process	135271.52
	On Hold	169575.61
	Resolved	134235.88
	Shipped	8865094.64

다음 쿼리는 주문 번호별 합계를 출력합니다.

```
SELECT
    orderNumber,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orderdetails
GROUP BY
    orderNumber;
```

	orderNumber	total
▶	10100	10223.83
	10101	10549.01
	10102	5494.78
	10103	50218.95
	10104	40206.20
	10105	53959.21
	10106	52151.81
	10107	22292.62

GROUP BY 과 표현 식
열 외에도 표현 식으로 행을 그룹화 할 수 있습니다. 다음 쿼리는 매년 총 매출을 가져옵니다.

```
SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
INNER JOIN orderdetails
    USING (orderNumber)
WHERE
    status = 'Shipped'
GROUP BY
    YEAR(orderDate);
```

	year	total
▶	2003	3223095.80
	2004	4300602.99
	2005	1341395.85

GROUP BY 절에서 반환된 결과를 제한하기 위해 HAVING 절을 사용합니다.
다음 쿼리는 HAVING 절을 사용하여 2003 년 이후 연도의 총 매출을 선택합니다.

```
SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
INNER JOIN orderdetails
    USING (orderNumber)
WHERE
    status = 'Shipped'
```

```
GROUP BY
    year
HAVING
    year > 2003;
```

	year	total
▶	2004	4300602.99
	2005	1341395.85

GROUP BY 절: ANSI SQL 과의 차이
표준 SQL 에서는 GROUP BY 절에 별칭을 사용할 수 없지만 MySQL 은 이를 지원합니다. 예를 들어 다음 쿼리는 주문 날짜에서 연도를 추출합니다. 먼저 year 표현식 YEAR(orderDate)를 year 별칭을 붙인 다음 GROUP BY 절 에서 별칭을 사용합니다. 이 쿼리는 ANSI SQL 에서 유효하지 않습니다.

```
SELECT
    YEAR(orderDate) AS year,
    COUNT(orderNumber)
FROM
    orders
GROUP BY
    year;
```

	year	COUNT(orderNumber)
▶	2003	111
	2004	151
	2005	64

MySQL 을 사용하면 표준 SQL 은 그렇지 않은 반면 오름차순 또는 내림차순으로 그룹을 정렬 할 수 있습니다. 기본 순서는 오름차순 입니다. 예를 들어, 상태 별 주문 수를 가져오고 내림차순으로 상태를 정렬하려면 GROUP BY 절에 DESC 를 사용할 수 있습니다.

```
SELECT
    status,
    COUNT(*)
FROM
    orders
GROUP BY
    status DESC;
```

	status	COUNT(*)
▶	Shipped	303
	Resolved	4
	On Hold	4
	In Process	6
	Disputed	3
	Cancelled	6

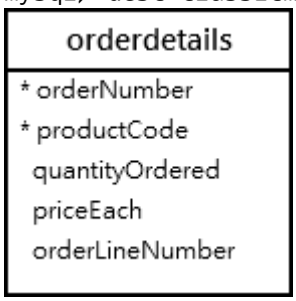
HAVING 절

HAVING 은 SELECT 명령문에서 행 또는 집계 그룹에 대한 제한 조건을 지정합니다.

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
GROUP BY
    group_by_expression
HAVING
    group_condition;
```

이 구문에서는 HAVING 절에 조건을 지정합니다. group by 절에 의해 생성 된 행으로 인해 group_condition 이 참으로 평가되면 쿼리는 결과 집합에 해당 행을 포함합니다.

```
mysql> desc classicmodels.orderdetails;
```



다음은 GROUP BY 절을 사용해 주문 번호, 주문 당 판매 된 품목 수 및 orderdetails 테이블에서 각각에 대한 총 판매량을 가져옵니다.

```
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber;
```

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

아래와 같이 HAVING 을 사용하여 총 매출이 1000 을 초과하는 주문을 찾을 수 있습니다.

```
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY
   ordernumber
HAVING
   total > 1000;
```

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

다음 예는 HAVING 절을 사용하여 총 금액이 1000 보다 많고 주문 수량이 600 을 초과하는 주문을 찾습니다.

```
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM
    orderdetails
GROUP BY ordernumber
HAVING
    total > 1000 AND
    itemsCount > 600;
```

ordernumber	itemsCount	total
10106	675	52151.81
10126	617	57131.92
10135	607	55601.84
10165	670	67392.85
10168	642	50743.65
10204	619	58793.53
10207	615	59265.14
10212	612	59830.55
10222	717	56822.65

배송 상태에 있고 총 금액이 1500 보다 큰 모든 주문을 찾으려면 테이블을 조인하고 열 및 집계에 조건을 적용 할 수 있습니다.

```
SELECT
    a.ordernumber,
    status,
    SUM(priceeach*quantityOrdered) total
FROM
    orderdetails a
INNER JOIN orders b
    ON b.ordernumber = a.ordernumber
GROUP BY
    ordernumber,
    status
HAVING
    status = 'Shipped' AND
    total > 1500;
```

ordernumber	status	total
10100	Shipped	10223.83
10101	Shipped	10549.01
10102	Shipped	5494.78
10103	Shipped	50218.95
10104	Shipped	40206.20
10105	Shipped	53959.21
10106	Shipped	52151.81

ROLLUP

아래 CTAS 구문을 수행하면 sales 테이블이 생성됩니다.

```
CREATE TABLE sales
AS
SELECT
    productLine,
    YEAR(orderDate) orderYear,
    SUM(quantityOrdered * priceEach) orderValue
FROM
    orderDetails
        INNER JOIN
    orders USING (orderNumber)
        INNER JOIN
    products USING (productCode)
GROUP BY
    productLine ,
    YEAR(orderDate);

SELECT
    *
FROM
    sales;
```

productLine	orderYear	orderValue
Vintage Cars	2003	4080.00
Classic Cars	2003	5571.80
Trucks and Buses	2003	3284.28
Trains	2003	2770.95
Ships	2003	5072.71
Planes	2003	4825.44
Motorcycles	2003	2440.50
Classic Cars	2004	8124.98
Vintage Cars	2004	2819.28
Trains	2004	4646.88
Ships	2004	4301.15
Planes	2004	2857.35
Motorcycles	2004	2598.77
Trucks and Buses	2004	4615.64
Motorcycles	2005	4004.88
Classic Cars	2005	5971.35
Vintage Cars	2005	5346.50
Trucks and Buses	2005	6295.03
Trains	2005	1603.20
Ships	2005	3774.00
Planes	2005	4018.00

```
SELECT
    productline,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline;
```

	productline	totalOrderValue
▶	Vintage Cars	12245.78
	Classic Cars	19668.13
	Trucks and Buses	14194.95
	Trains	9021.03
	Ships	13147.86
	Planes	11700.79
	Motorcycles	9044.15

```
SELECT
    SUM(orderValue) totalOrderValue
FROM
    sales;
```

	totalOrderValue
▶	89022.69

하나의 쿼리에서 두 개 이상의 그룹화 집합을 함께 생성하려면 UNION ALL 다음과 같이 연산자를 사용할 수 있습니다.

```
SELECT
    productline,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline
UNION ALL
SELECT
    NULL,
    SUM(orderValue) totalOrderValue
FROM
    sales;
```

	productline	totalOrderValue
▶	Vintage Cars	12245.78
	Classic Cars	19668.13
	Trucks and Buses	14194.95
	Trains	9021.03
	Ships	13147.86
	Planes	11700.79
	Motorcycles	9044.15
	NULL	89022.69

위 쿼리는 제품 라인 별 총 주문 값과 총 합계 행을 생성 할 수 있습니다. 그러나 두 가지 문제가 있습니다.

1. 쿼리가 길다.
2. 데이터베이스 엔진이 내부적으로 두 개의 개별 쿼리를 실행하고 결과 집합을 하나로 결합해야 하므로 쿼리 성능이 좋지 않을 수 있습니다.

이러한 문제를 해결하기 위해 ROLLUP 절을 사용합니다.
이 ROLLUP 절은 GROUP BY 다음 구문 을 사용하는 절의 확장입니다.

```
SELECT
    productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline WITH ROLLUP;
```

	productLine	totalOrderValue
▶	Classic Cars	19668.13
	Motorcycles	9044.15
	Planes	11700.79
	Ships	13147.86
	Trains	9021.03
	Trucks and Buses	14194.95
	Vintage Cars	12245.78
	NULL	89022.69

출력에서 명확하게 볼 수 있듯이 ROLLUP 절은 부분합뿐만 아니라 주문 값의 총합계도 생성합니다.

```
SELECT
    productLine,
    orderYear,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline,
    orderYear
WITH ROLLUP;
```

productLine	orderYear	totalOrderValue
Classic Cars	2003	5571.80
Classic Cars	2004	8124.98
Classic Cars	2005	5971.35
Classic Cars	NULL	19668.13
Motorcycles	2003	2440.50
Motorcycles	2004	2598.77
Motorcycles	2005	4004.88
Motorcycles	NULL	9044.15
Planes	2003	4825.44
Planes	2004	2857.35
Planes	2005	4018.00
Planes	NULL	11700.79
Ships	2003	5072.71
Ships	2004	4301.15
Ships	2005	3774.00
Ships	NULL	13147.86
Trains	2003	2770.95
Trains	2004	4646.88
Trains	2005	1603.20
Trains	NULL	9021.03
Trucks and Buses	2003	3284.28
Trucks and Buses	2004	4615.64
Trucks and Buses	2005	6295.03
Trucks and Buses	NULL	14194.95
Vintage Cars	2003	4080.00
Vintage Cars	2004	2819.28
Vintage Cars	2005	5346.50
Vintage Cars	NULL	12245.78
NULL	NULL	89022.69

```
SELECT
    orderYear,
    productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    orderYear,
    productline
WITH ROLLUP;
```

orderYear	productLine	totalOrderValue
2003	Classic Cars	5571.80
2003	Motorcycles	2440.50
2003	Planes	4825.44
2003	Ships	5072.71
2003	Trains	2770.95
2003	Trucks and Buses	3284.28
2003	Vintage Cars	4080.00
2003	NULL	28045.68
2004	Classic Cars	8124.98
2004	Motorcycles	2598.77
2004	Planes	2857.35
2004	Ships	4301.15
2004	Trains	4646.88
2004	Trucks and Buses	4615.64
2004	Vintage Cars	2819.28
2004	NULL	29964.05
2005	Classic Cars	5971.35
2005	Motorcycles	4004.88
2005	Planes	4018.00
2005	Ships	3774.00
2005	Trains	1603.20
2005	Trucks and Buses	6295.03
2005	Vintage Cars	5346.50
2005	NULL	31012.96
NULL	NULL	89022.69

ROLLUP 소계마다 orderYear 의 변화와 productLine 결과 세트의 끝에 총계를 생성합니다.
이 예의 계층 구조는 다음과 같습니다.
orderYear > productLine

GROUPING() 함수

NULL 결과 집합이 부분합 또는 총합을 나타내는지 확인하려면 GROUPING() 함수를 사용합니다.

```
SELECT
    orderYear,
    productLine,
    SUM(orderValue) totalOrderValue,
    GROUPING(orderYear),    # 추가
    GROUPING(productLine)   # 추가
FROM
    sales
GROUP BY
    orderYear,
    productline
WITH ROLLUP;
```

orderYear	productLine	totalOrderValue	GROUPING(orderYear)	GROUPING(productLine)
2003	Classic Cars	5571.80	0	0
2003	Motorcycles	2440.50	0	0
2003	Planes	4825.44	0	0
2003	Ships	5072.71	0	0
2003	Trains	2770.95	0	0
2003	Trucks and Buses	3284.28	0	0
2003	Vintage Cars	4080.00	0	0
2003	NULL	28045.68	0	1
2004	Classic Cars	8124.98	0	0
2004	Motorcycles	2598.77	0	0
2004	Planes	2857.35	0	0
2004	Ships	4301.15	0	0
2004	Trains	4646.88	0	0
2004	Trucks and Buses	4615.64	0	0
2004	Vintage Cars	2819.28	0	0
2004	NULL	29964.05	0	1
2005	Classic Cars	5971.35	0	0
2005	Motorcycles	4004.88	0	0
2005	Planes	4018.00	0	0
2005	Ships	3774.00	0	0
2005	Trains	1603.20	0	0
2005	Trucks and Buses	6295.03	0	0
2005	Vintage Cars	5346.50	0	0
2005	NULL	31012.96	0	1
NULL	NULL	89022.69	1	1

아래와 같이 NULL 을 레이블로 대체할 수 있습니다.

```
SELECT
    IF(GROUPING(orderYear),
        '*All Years',
        orderYear) orderYear,
    IF(GROUPING(productLine),
        '*All Product Lines',
        productLine) productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    orderYear ,
    productline
WITH ROLLUP;
```

orderYear	productLine	totalOrderValue
2003	Classic Cars	5571.80
2003	Motorcycles	2440.50
2003	Planes	4825.44
2003	Ships	5072.71
2003	Trains	2770.95
2003	Trucks and Buses	3284.28
2003	Vintage Cars	4080.00
2003	All Product Lines	28045.68
2004	Classic Cars	8124.98
2004	Motorcycles	2598.77
2004	Planes	2857.35
2004	Ships	4301.15
2004	Trains	4646.88
2004	Trucks and Buses	4615.64
2004	Vintage Cars	2819.28
2004	All Product Lines	29964.05
2005	Classic Cars	5971.35
2005	Motorcycles	4004.88
2005	Planes	4018.00
2005	Ships	3774.00
2005	Trains	1603.20
2005	Trucks and Buses	6295.03
2005	Vintage Cars	5346.50
2005	All Product Lines	31012.96
All Years	All Product Lines	89022.69

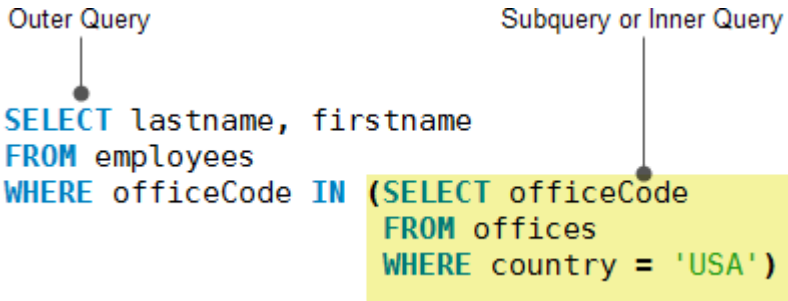
Subquery

서브 쿼리는 다른 쿼리 내에 중첩 된 쿼리입니다.
SELECT, INSERT, UPDATE, DELETE 또는 다른 서브 쿼리에 중첩 될 수 있습니다.
MySQL 서브 쿼리를 이너 쿼리, 서브 쿼리를 포함하는 쿼리를 메인 쿼리 혹은 아우터 쿼리라고 합니다.
서브 쿼리는 해당 표현식이 사용되는 모든 곳에서 사용할 수 있으며 괄호로 묶어야 합니다.

다음 쿼리는 미국에 있는 사무실에서 일하는 직원을 반환합니다.

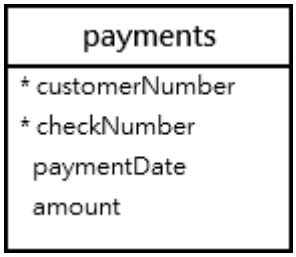
```
SELECT
    lastName, firstName
FROM
    employees
WHERE
    officeCode IN (SELECT
            officeCode
        FROM
            offices
        WHERE
            country = 'USA');
```

서브 쿼리는 미국에 있는 사무실의 모든 사무실 코드를 반환합니다 .
메인 쿼리는 서브 쿼리에서 반환 된 결과 집합의 사무실 코드가 있는 사무실에서 일하는 직원의 성과 이름을 출력합니다.



쿼리가 실행되면 서브 쿼리가 먼저 실행되고 결과 집합이 반환됩니다. 그런 다음이 결과 집합이 메인 쿼리에 대한 입력으로 사용됩니다.

```
mysql> desc classicmodels.payments;
```



비교 연산자가 있는 MySQL 서브 쿼리
비교 연산자 (예 : =,>, <)를 사용하여 서브 쿼리에서 반환된 단일 값을 WHERE 절의 식과 비교할 수 있습니다.

다음 쿼리는 최대 금액을 지불한 고객을 출력합니다.

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount = (SELECT MAX(amount) FROM payments);
```

	customerNumber	checkNumber	amount
▶	141	JE105477	120166.58

이퀄 연산자 외에 크다, 작다, 작거나 같다, 같다, 다르다와 같은 다른 관계 연산자를 사용할 수 있습니다.

예를 들어, 서브 쿼리를 사용하여 지불액이 평균 지불액보다 많은 고객을 찾을 수 있습니다.

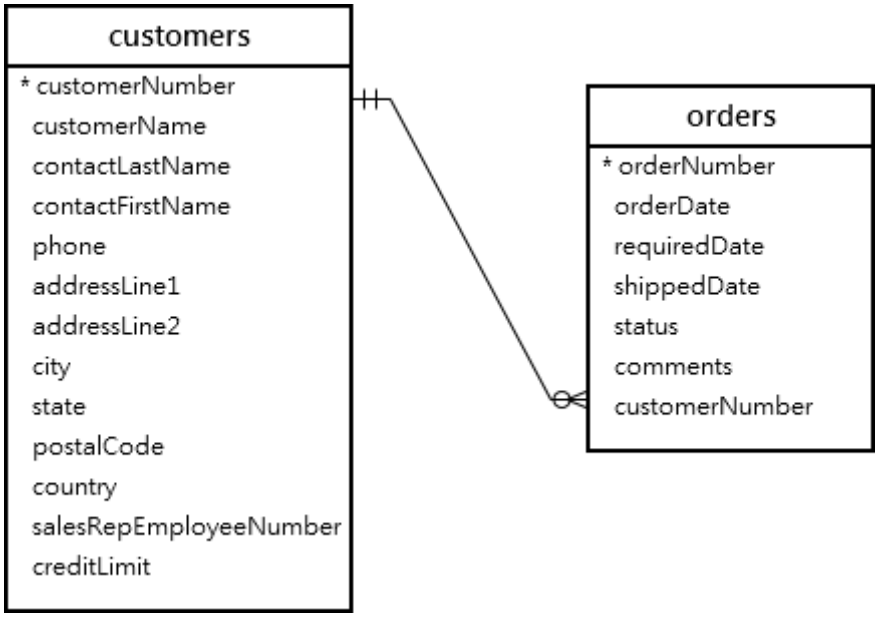
```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount > (SELECT
            AVG(amount)
        FROM
            payments);
```

customerNumber	checkNumber	amount
112	HQ55022	32641.98
112	ND748579	33347.88
114	GG31455	45864.03
114	MA765515	82261.22
114	NR27552	44894.74
119	LN373447	47924.19
119	NG94694	49523.67
121	DB889831	50218.95
121	MA302151	34638.14

먼저 서브 쿼리를 사용하여 AVG 집계 함수를 사용하여 평균 금액을 계산합니다.
다음으로 메인 쿼리는 서브 쿼리에서 반환 한 평균 금액보다 큰 금액을 출력합니다.

IN 및 NOT IN 연산자를 사용하는 MySQL 서브 쿼리
서브 쿼리가 둘 이상의 값을 반환하는 경우 WHERE 절 에서 IN or NOT IN 연산자와 같은 다른 연산자를 사용할 수 있습니다.

```
mysql> desc classicmodels.customers;
mysql> desc classicmodels.orders;
```



NOT IN 연산자 와 함께 서브 쿼리를 사용하여 다음과 같이 주문을 하지 않은 고객을 찾을 수 있습니다.

```
SELECT
    customerName
FROM
    customers
WHERE
    customerNumber NOT IN (SELECT DISTINCT
```

```
        customerNumber
    FROM
        orders);
```

	customername
▶	Havel & Zbyszek Co
	American Souvenirs Inc
	Porto Imports Co.
	Asian Shopping Network, Co
	Natürlich Autos
	ANG Resellers
	Messner Shopping Network
	Franken Gifts, Co
	BG&E Collectables

FROM 절의 서브 쿼리
FROM 절에서 서브 쿼리를 사용하면 서브 쿼리에서 반환 된 결과 집합이 임시 테이블로 사용됩니다.
이 테이블을 파생 된 테이블 또는 구체화 된 서브 쿼리라고 합니다.

다음 서브 쿼리는 판매 주문에서 최대, 최소 및 평균 항목 수를 출력합니다.

```
SELECT
    MAX(items),
    MIN(items),
    FLOOR(AVG(items))
FROM
    (SELECT
        orderNumber, COUNT(orderNumber) AS items
    FROM
        orderdetails
    GROUP BY orderNumber) AS lineitems;
```

	MAX(items)	MIN(items)	FLOOR(AVG(items))
▶	18	1	9

FLOOR() 함수는 소수점 이하 자릿수를 제거하는 데 사용됩니다.

상호 연관 서브 쿼리
이전 예에서 서브 쿼리가 독립적임을 알 수 있습니다.

```
SELECT
    orderNumber,
    COUNT(orderNumber) AS items
FROM
    orderdetails
```

```
GROUP BY orderNumber;
```

독립 실행 형 서브 쿼리와 달리 상호연관 서브 쿼리는 메인 쿼리의 데이터를 사용하는 서브 쿼리입니다. 즉, 상호연관 서브 쿼리는 메인 쿼리에 종속됩니다. 상관 서브 쿼리는 메인 쿼리의 각 행에 대해 한 번씩 평가됩니다.
다음 쿼리에서는 구매 가격이 각 제품 군에 있는 모든 제품의 평균 구매 가격보다 큰 제품을 선택합니다.

```
SELECT
    productname,
    buyprice
FROM
    products p1
WHERE
    buyprice > (SELECT
            AVG(buyprice)
        FROM
            products
        WHERE
            productline = p1.productline)
```

productname	buyprice
1952 Alpine Renault 1300	98.58
1996 Moto Guzzi 1100i	68.99
2003 Harley-Davidson Eagle Drag Bike	91.02
1972 Alfa Romeo GTA	85.68
1962 LanciaA Delta 16V	103.42
1968 Ford Mustang	95.34
2001 Ferrari Enzo	95.59
1958 Setra Bus	77.90

제품 라인이 모든 행에 대해 변경되기 때문에 내부 쿼리는 모든 제품 라인에 대해 실행됩니다. 따라서 평균 구매 가격도 변경됩니다. 메인 쿼리는 서브 쿼리에서 제품 라인 당 평균 구매 가격 보다 구매 가격이 높은 제품만 제한 합니다.

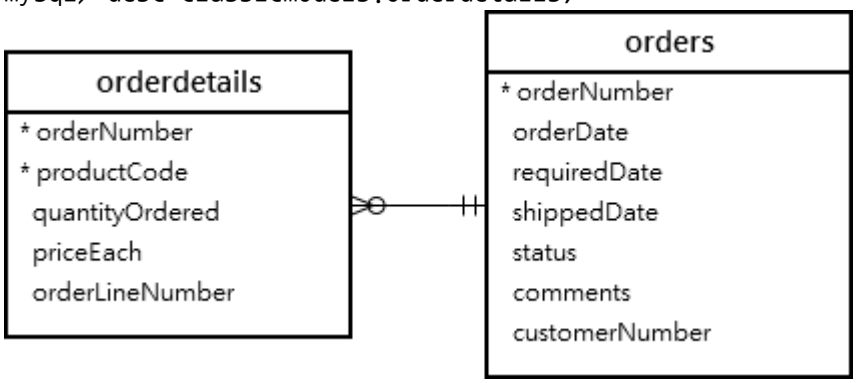
MySQL 의 서브 쿼리와 EXISTS 와 NOT EXISTS

서브 쿼리가 EXISTS 또는 NOT EXISTS 연산자 와 함께 사용되면 서브 쿼리는 부울 값 TRUE 또는 FALSE 를 반환합니다. 다음 쿼리는 EXISTS 연산자와 함께 사용되는 서브 쿼리를 보여줍니다.

```
SELECT
    *
```

```
FROM
    table_name
WHERE
    EXISTS( subquery );
```

```
mysql> desc classicmodels.orders;
mysql> desc classicmodels.orderdetails;
```



다음 쿼리는 총 값이 60K 보다 큰 판매 주문을 찾습니다.

```
SELECT
    orderNumber,
    SUM(priceEach * quantityOrdered) total
FROM
    orderdetails
        INNER JOIN
    orders USING (orderNumber)
GROUP BY orderNumber
HAVING SUM(priceEach * quantityOrdered) > 60000;
```

orderNumber	total
10165	67392.85
10287	61402.00
10310	61234.67

3 개의 행을 반환합니다. 즉, 총 값이 60K 보다 큰 3 개의 판매 주문이 있습니다. 위의 쿼리를 상호연관 서브 쿼리로 사용하여 EXISTS 연산자를 사용하여 총 금액이 60K 보다 큰 판매 주문한 고객을 찾을 수 있습니다.

```
SELECT
    customerNumber,
    customerName
FROM
    customers
WHERE
    EXISTS( SELECT
            orderNumber, SUM(priceEach * quantityOrdered)
        FROM
            orderdetails
                INNER JOIN
```

```
    orders USING (orderNumber)
WHERE
    customerNumber = customers.customerNumber
GROUP BY orderNumber
HAVING SUM(priceEach * quantityOrdered) > 60000);
```

customerNumber	customerName
148	Dragon Souveniers, Ltd.
259	Toms Spezialitäten, Ltd
298	Vida Sport, Ltd

인라인 뷰(Inline View)

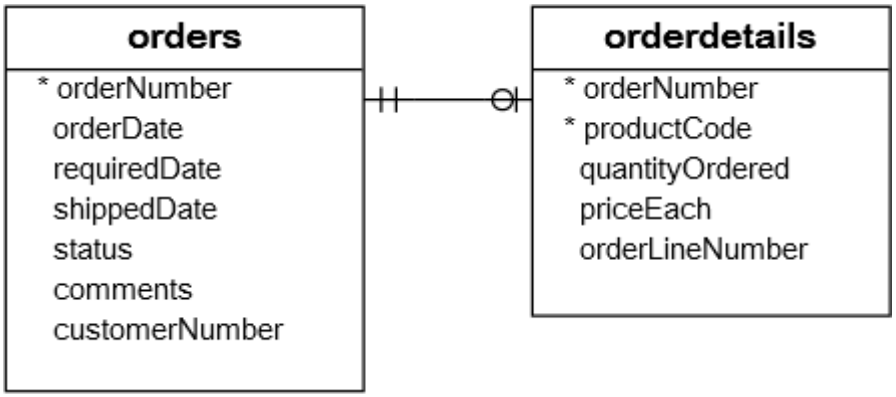
인라인 뷰, 혹은 파생 테이블이란 SELECT 명령문에서 FROM 절에 사용되는 독립 실행 형 서브 쿼리를 의미합니다.

```
SELECT
    column_list
FROM
    (SELECT
        column_list
    FROM
        table_1) derived_table_name;
WHERE derived_table_name.c1 > 0;
```

FROM 절의 독립 실행 형 서브 쿼리는 이를 포함하는 문과 독립적으로 실행할 수 있는 서브 쿼리입니다. MySQL에서 인라인 뷰는 서브 쿼리와 달리 나중에 쿼리에서 참조 할 수 있도록 별칭이 필수이며 AS를 사용할 수 있습니다. 파생 테이블에 별칭이 없는 경우 MySQL은 다음 오류를 발생시킵니다.

```
Every derived table must have its own alias.
```

다음 쿼리는 orders및 orderdetails 테이블에서 2003년 판매 수익 기준 상위 5개 제품을 가져옵니다.

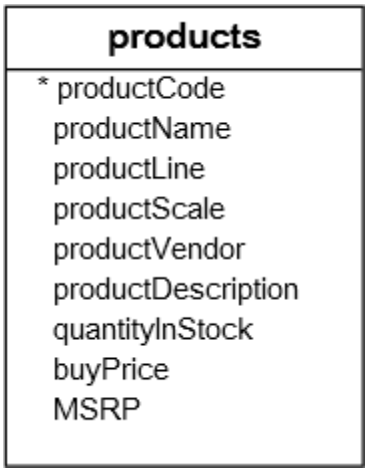



```
SELECT
    productCode,
    ROUND(SUM(quantityOrdered * priceEach)) sales
```

```
FROM
    orderdetails
        INNER JOIN
    orders USING (orderNumber)
WHERE
    YEAR(shippedDate) = 2003
GROUP BY productCode
ORDER BY sales DESC
LIMIT 5;
```

productCode	sales
S18_3232	103480
S10_1949	67985
S12_1108	59852
S12_3891	57403
S12_1099	56462

이 쿼리의 결과를 인라인 뷰로 사용하고 products다음과 같이 테이블과 조인 할 수 있습니다.




```
SELECT
    productName, sales
FROM
    (SELECT
```

```
        productCode,
        ROUND(SUM(quantityOrdered * priceEach)) sales
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2003
    GROUP BY productCode
    ORDER BY sales DESC
    LIMIT 5) top5products2003
INNER JOIN
    products USING (productCode);
```

productName	sales
1992 Ferrari 360 Spider red	103480
1952 Alpine Renault 1300	67985
2001 Ferrari Enzo	59852
1969 Ford Falcon	57403
1968 Ford Mustang	56462

2003년에 제품을 구입한 고객을 platinum, gold, silver로 분류 하려고 합니다.

주문량이 10만 이상은 플래티넘 고객

주문량이 1만에서 10만 사이는 골드 고객

주문량이 10,000 개 미만은 실버 고객

이 쿼리를 생성하려면 먼저 다음과 같이 CASE식과 GROUP BY절을 사용하여 각 고객을 해당 그룹에 넣어야 합니다.

```
SELECT
    customerNumber,
```

```
    ROUND(SUM(quantityOrdered * priceEach)) sales,
    (CASE
        WHEN SUM(quantityOrdered * priceEach) < 10000 THEN 'Silver'
        WHEN SUM(quantityOrdered * priceEach) BETWEEN 10000 AND 100000 THEN 'Gold'
        WHEN SUM(quantityOrdered * priceEach) > 100000 THEN 'Platinum'
    END) customerGroup
FROM
    orderdetails
        INNER JOIN
    orders USING (orderNumber)
WHERE
    YEAR(shippedDate) = 2003
GROUP BY customerNumber;
```

customerNumber	sales	customerGroup
103	14571	Gold
112	32642	Gold
114	53429	Gold
121	51710	Gold
124	167783	Platinum
128	34651	Gold
129	40462	Gold
131	22293	Gold
141	189840	Platinum

이제 이 쿼리를 인라인 뷰로 사용하고 그룹화를 수행 할 수 있습니다.

```
SELECT
    customerGroup,
    COUNT(cg.customerGroup) AS groupCount
FROM
    (SELECT
        customerNumber,
```

```
            ROUND(SUM(quantityOrdered * priceEach)) sales,
            (CASE
                WHEN SUM(quantityOrdered * priceEach) < 10000 THEN 'Silver'
                WHEN SUM(quantityOrdered * priceEach) BETWEEN 10000 AND 100000 THEN
'Gold'
                WHEN SUM(quantityOrdered * priceEach) > 100000 THEN 'Platinum'
            END) customerGroup
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2003
    GROUP BY customerNumber) cg
GROUP BY cg.customerGroup;
```

쿼리는 고객 그룹과 고객 수를 반환합니다.

customerGroup	groupCount
Gold	61
Silver	8
Platinum	4

EXISTS 연산자

다음은 EXISTS 연산자의 기본 구문입니다.

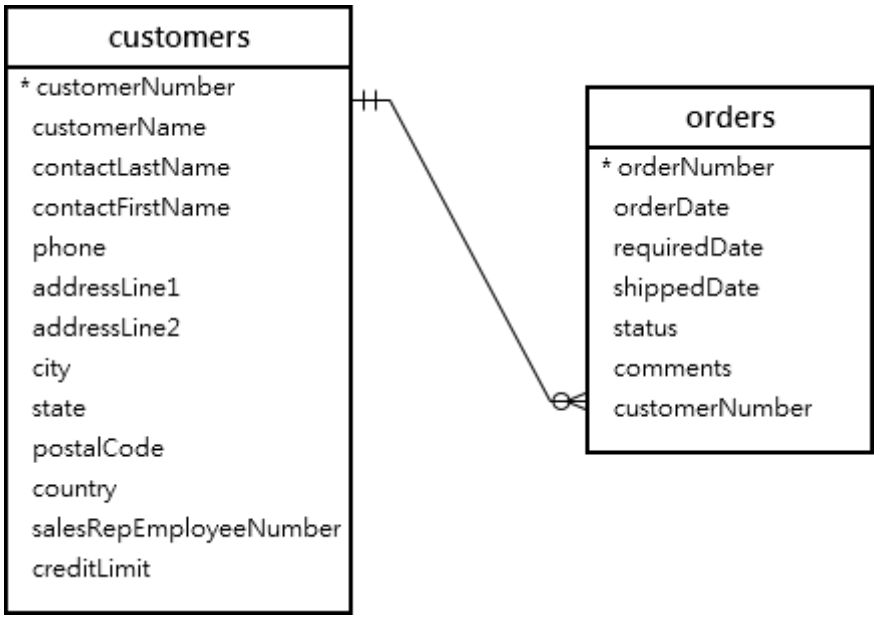
```
SELECT
    select_list
FROM
    a_table
WHERE
    [NOT] EXISTS(subquery);
```

하위 쿼리가 하나 이상의 행을 EXISTS반환 하면 연산자는 true를 반환하고 그렇지 않으면 false를 반환합니다.

또한 EXISTS연산자는 일치하는 행을 찾으면 즉시 추가 처리를 종료하므로 쿼리 성능을 향상시킬 수 있습니다.

NOT EXISTS는 서브 쿼리가 행을 반환하지 않으면 true 그렇지 않으면 false를 반환합니다.

SELECT EXISTS 예



다음 문은 EXISTS연산자를 사용하여 하나 이상의 주문이 있는 고객을 찾습니다.

```
SELECT
    customerNumber,
    customerName
FROM
    customers
WHERE
    EXISTS(
        SELECT
            1
        FROM
            orders
        WHERE
            orders.customernumber
              = customers.customernumber);
```

customerNumber	customerName
103	Atelier graphique
112	Signal Gift Stores
114	Australian Collectors, Co.
119	La Rochelle Gifts
121	Baane Mini Imports
124	Mini Gifts Distributors Ltd.
128	Blauer See Auto, Co.
129	Mini Wheels Co.
131	Land of Toys Inc.

다음 예에서는 NOT EXISTS연산자를 사용하여 주문이 없는 고객을 찾습니다.

```
SELECT
    customerNumber,
    customerName
FROM
```

```
    customers
WHERE
    NOT EXISTS(
       SELECT
            1
        FROM
            orders
        WHERE
            orders.customernumber = customers.customernumber
       );
```

	customerNumber	customerName
▶	125	Havel & Zbyszek Co
	168	American Souvenirs Inc
	169	Porto Imports Co.
	206	Asian Shopping Network, Co
	223	Natürlich Autos
	237	ANG Resellers
	247	Messner Shopping Network
	273	Franken Gifts, Co
	293	BG&E Collectables

UPDATE EXISTS 예

샌프란시스코 사무실에서 근무하는 직원의 내선 전화 번호를 업데이트 해야 한다고 가정합니다.

```
SELECT
    employeenumber,
    firstname,
    lastname,
    extension
FROM
    employees
```

```
WHERE
    EXISTS(
        SELECT
            1
        FROM
            offices
        WHERE
            city = 'San Francisco' AND
           offices.officeCode = employees.officeCode);
```

employeenumber	firstname	lastname	extension
1002	Diane	Murphy	x5800
1056	Mary	Patterson	x4611
1076	Jeff	Firrelli	x9273
1143	Anthony	Bow	x5428
1165	Leslie	Jennings	x3291
1166	Leslie	Thompson	x4065

이 예에서는 샌프란시스코 사무실에서 근무하는 직원의 내선 전화 번호에 1을 추가합니다.

```
UPDATE employees
SET
    extension = CONCAT(extension, '1')
WHERE
    EXISTS(
        SELECT
            1
        FROM
            offices
        WHERE
            city = 'San Francisco'
                AND offices.officeCode = employees.officeCode);
```

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	(26) 642-7555	ul. Filtrowa 68
168	American Souvenirs Inc	Franco	Keith	2035557845	149 Spinnaker Dr.
169	Porto Imports Co.	de Castro	Isabel	(1) 356-5555	Estrada da saúde n. 58
206	Asian Shopping Network, Co	Walker	Brydey	+612 9411 1555	Suntec Tower Three
223	Natürlich Autos	Kloss	Horst	0372-555188	Taucherstraße 10
237	ANG Resellers	Camino	Alejandra	(91) 745 6555	Gran Vía, 1
247	Messner Shopping Network	Messner	Renate	069-0555984	Magazinweg 7

Common Table Expressions

MySQL은 버전 8.0부터 공통 테이블 표현 식 또는 CTE 기능을 도입했습니다.

공통 테이블 식 또는 CTE 란?

공통 테이블 표현 식은 하나의 SQL 문 이를테면 SELECT, INSERT, UPDATE, 또는 DELETE 의 실행 범위 내에서만 존재하는 명명 된 임시 결과 집합을 의미합니다.

인라인 뷰(파생 테이블)와 유사하게 CTE는 객체로 저장되지 않으며 쿼리 실행 중에만 존재합니다.

파생 테이블과 달리 CTE는 자체 참조 혹은 동일한 쿼리에서 여러 번 재귀하여 참조 될 수 있습니다. 또한 CTE는 파생 테이블에 비해 더 나은 가독성과 성능을 제공합니다.

다음은 CTE의 기본 구문을 보여줍니다.

```
WITH cte_name (column_list) AS (
    query
)
SELECT * FROM cte_name;
```

column_list의 열 수는 의 열 query수와 같아야 합니다.

(column_list)를 생략하면 column_listCTE 는 CTE를 정의하는 쿼리의 열 목록을 사용합니다.


```
WITH customers_in_usa AS (
    SELECT
        customerName, state
    FROM
        customers
    WHERE
        country = 'USA'
)
SELECT
    customerName
 FROM
    customers_in_usa
 WHERE
    state = 'CA'
 ORDER BY customerName;
```

customerName
Boards & Toys Co.
Collectable Mini Designs Co.
Corporate Gift Ideas Co.
Men 'R' US Retailers, Ltd.
Mini Gifts Distributors Ltd.
Mini Wheels Co.
Signal Collectibles Ltd.
Technics Stores Inc.
The Sharp Gifts Warehouse
Toys4GrownUps.com
West Coast Collectables Co.

```
WITH topsales2003 AS (
    SELECT
        salesRepEmployeeNumber employeeNumber,
        SUM(quantityOrdered * priceEach) sales
    FROM
        orders
            INNER JOIN
        orderdetails USING (orderNumber)
            INNER JOIN
        customers USING (customerNumber)
    WHERE
        YEAR(shippedDate) = 2003
            AND status = 'Shipped'
    GROUP BY salesRepEmployeeNumber
    ORDER BY sales DESC
    LIMIT 5
)
SELECT
    employeeNumber,
    firstName,
```

```
    lastName,
    sales
FROM
    employees
        JOIN
    topsales2003 USING (employeeNumber);
```

	employeeNumber	firstName	lastName	sales
▶	1165	Leslie	Jennings	413219.85
	1370	Gerard	Hernandez	295246.44
	1401	Pamela	Castillo	289982.88
	1621	Mami	Nishi	267249.40
	1501	Larry	Bott	261536.95

이 예에서 CTE는 2003 년에 상위 5 명의 영업 담당자를 반환합니다.

그 후 topsales2003CTE를 참조하여 이름과 성을 포함한 영업 담당자에 대한 추가 정보를 얻었습니다.

Salesrep, customer_salesrep 2개의 CTE가 적용된 구문을 보겠습니다.

```
WITH salesrep AS (
    SELECT
        employeeNumber,
        CONCAT(firstName, ' ', lastName) AS salesrepName
    FROM
        employees
    WHERE
        jobTitle = 'Sales Rep'
),
customer_salesrep AS (
    SELECT
```

```
        customerName, salesrepName
    FROM
        customers
            INNER JOIN
        salesrep ON employeeNumber = salesrepEmployeeNumber
)
SELECT
    *
FROM
    customer_salesrep
ORDER BY customerName;
```

customerName	salesrepName
Alpha Cognac	Gerard Hernandez
American Souvenirs Inc	Foon Yue Tseng
Amica Models & Co.	Pamela Castillo
Anna's Decorations, Ltd	Andy Fixter
Atelier graphique	Gerard Hernandez
Australian Collectables, Ltd	Andy Fixter
Australian Collectors, Co.	Andy Fixter
Australian Gift Network, Co	Andy Fixter
Auto Associés & Cie.	Gerard Hernandez
Auto Canal+ Petit	Loui Bondur
Auto-Moto Classics Inc.	Steve Patterson
AV Stores, Co.	Larry Bott

첫 번째 CTE (salesrep)는 직책이 영업 담당자인 직원을 가져옵니다. 두 번째 CTE (customer_salesrep)는 INNER JOIN절의 첫 번째 CTE를 참조하여 영업 담당자와 각 영업 담당자가 담당하는 고객을 가져옵니다.

두 번째 CTE를 얻은 후 ORDER BY절이 있는 간단한 SELECT문을 사용하여 해당 CTE의 데이터를 쿼리 합니다.

이렇게 공통 테이블 표현 식(CTE)을 사용하여 복잡한 쿼리를 단순화 할 수 있습니다.

UNION 연산자

UNION 연산자로 두 개 이상의 쿼리 결과 집합을 단일 결과 집합으로 결합 할 수 있습니다. 다음은 UNION 연산자의 구문 입니다.

```
SELECT column_list
UNION [DISTINCT ¦ ALL]
SELECT column_list
UNION [DISTINCT ¦ ALL]
SELECT column_list
```

UNION 연산자를 규칙은 다음과 같습니다.

첫째, 모든 SELECT문에 표시되는 열의 수와 순서는 동일해야 합니다.

둘째, 열의 데이터 유형이 동일하거나 호환 가능해야 합니다.

기본적으로 UNION 연산자는 명시적으로 지정하지 않은 경우에도 중복 행을 제거합니다.

```
DROP TABLE IF EXISTS t1;
DROP TABLE IF EXISTS t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY
);

CREATE TABLE t2 (
    id INT PRIMARY KEY
);

INSERT INTO t1 VALUES (1),(2),(3);
INSERT INTO t2 VALUES (2),(3),(4);
```

다음 문은 t1및 t2테이블 에서 반환된 결과 집합을 결합합니다.

```
SELECT id

FROM t1

UNION

SELECT id

FROM t2;

+----+

¦ id ¦

+----+

¦  1 ¦

¦  2 ¦

¦  3 ¦

¦  4 ¦

+----+

4 rows in set (0.00 sec)
```

UNION ALL을 사용하는 경우 중복 행이 결과에 남아 있습니다. 때문에 UNION ALL은 중복된 결과도 무방하거나, 각각의 집합에 값들이 겹치지 않았음을 확신 할 수 있을 때 사용해야 합니다.

```
SELECT id

FROM t1

UNION ALL

SELECT id

FROM t2;

+----+

¦ id ¦

+----+

¦  1 ¦

¦  2 ¦

¦  3 ¦

¦  2 ¦
```

```
¦  3 ¦
¦  4 ¦
+----+
```

	fullname
▶	Adrian Huxley
	Akiko Shimamura
	Alejandra Camino
	Alexander Feuer
	Alexander Semenov
	Allen Nelson
	Andy Fixter
	Ann Brown
	Anna O'Hara
	Annette Roulet

그 외 타 DBMS 의 intersect, minus 는 MySQL 은 지원하지 않습니다.

INSERT 문

MySQL INSERT예

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    priority TINYINT NOT NULL DEFAULT 3,
    description TEXT,
    PRIMARY KEY (task_id)
);
```

```
INSERT INTO tasks(title,priority)
VALUES('Learn MySQL INSERT Statement',1);
```

```
SELECT * FROM tasks;
```

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL

```
INSERT INTO tasks(title,priority)
VALUES('Understanding DEFAULT keyword in INSERT statement',DEFAULT);
```

테이블에 날짜 삽입

열에 리터럴 날짜 값을 삽입하려면 다음 형식을 사용합니다.

'YYYY-MM-DD'

YYYY 4 자리 연도를 나타냅니다 (예 : 2018).

MM 두 자리 월을 나타냅니다 (예 : 01, 02 및 12).

DD 두 자리 일 (예 : 01, 02, 30)을 나타냅니다.

```
INSERT INTO tasks(title, start_date, due_date)
VALUES('Insert date into table','2018-01-09','2018-09-15');
```

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL

```
INSERT INTO tasks(title,start_date,due_date)
VALUES('Use current date for the task',CURRENT_DATE(),CURRENT_DATE())
```

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL
4	Use current date for the task	2018-09-02	2018-09-02	3	NULL

INSERT - 여러 행 삽입 예

```
INSERT INTO tasks(title, priority)
VALUES
        ('My first task', 1),
        ('It is the second task',2),
        ('This is the third task of the week',3);
```

task_id	title	start_date	due_date	priority	description
1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL
2	Understanding DEFAULT keyword in INSERT statement	NULL	NULL	3	NULL
3	Insert date into table	2018-01-09	2018-09-15	3	NULL
4	Use current date for the task	2018-09-02	2018-09-02	3	NULL
5	My first task	NULL	NULL	1	NULL
6	It is the second task	NULL	NULL	2	NULL
7	This is the third task of the week	NULL	NULL	3	NULL

INSERT를 사용하여 테이블에 여러 행을 추가 할 때 처리 중에 오류가 발생하면 MySQL은 문을 종료하고 오류를 반환합니다. 그러나 INSERT IGNORE 명령문을 사용하면 오류 행은 무시되고 유효한 데이터가 있는 행은 테이블에 삽입됩니다.

```
CREATE TABLE subscribers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(50) NOT NULL UNIQUE
);

INSERT INTO subscribers(email)
VALUES('john.doe@gmail.com');

INSERT INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

Error Code: 1062. Duplicate entry 'john.doe@gmail.com' for key 'email'

이메일 john.doe@gmail.com은 UNIQUE제약 조건을 위반합니다 .

그러나 INSERT IGNORE대신 문을 사용하는 경우는 오류 행은 무시되고 유효한 데이터가 있는 행은 테이블에 삽입됩니다.

```
INSERT IGNORE INTO subscribers(email)
VALUES('john.doe@gmail.com'),
      ('jane.smith@ibm.com');
```

DELETE 문

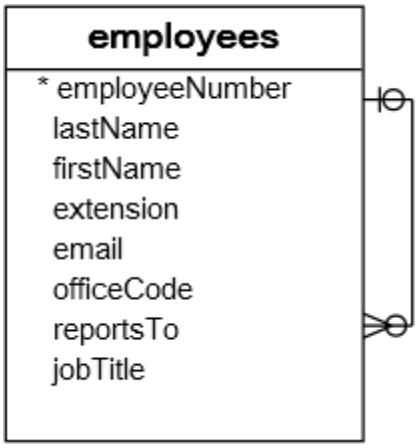
```
DELETE FROM table_name
WHERE condition;
```

조건에 맞는 행을 삭제합니다.

이 WHERE절은 선택 사항입니다. WHERE절을 생략하면 DELETE 명령문은 테이블의 모든 행을 삭제합니다.

테이블에서 데이터를 삭제하는 것 외에도 DELETE문은 삭제 된 행의 수를 반환합니다.

mysql> desc classicModels.employees;.



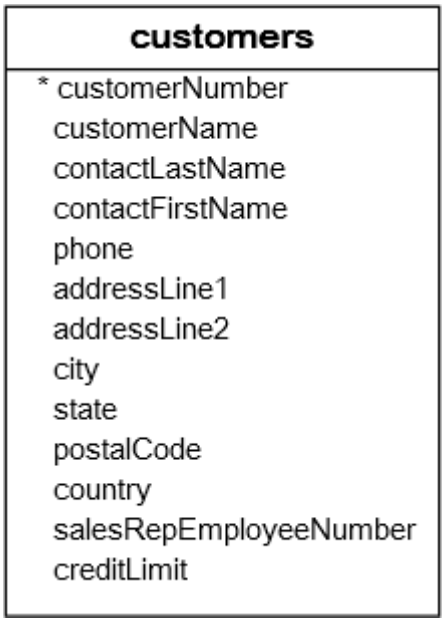

```
DELETE FROM employees
WHERE officeCode = 4;
```

```
DELETE FROM employees;
```

employees테이블의 모든 행이 삭제되었습니다.

삭제할 행 수를 제한하려면 LIMIT 절을 사용합니다.

LIMIT절 을 사용할 때는 항상 ORDER BY절을 사용해야 합니다.



고객 이름을 알파벳순으로 정렬하고 처음 10 명의 고객을 삭제합니다.

```
DELETE FROM customers
ORDER BY customerName
LIMIT 10;
```

고객을 선택 France하고 신용 한도 별로 오름차순 정렬 한 다음 처음 5 명의 고객을 삭제합니다.

```
DELETE FROM customers
WHERE country = 'France'
ORDER BY creditLimit
LIMIT 5;
```

ON DELETE CASCADE

Buildings 테이블 생성.

```
CREATE TABLE buildings (
    building_no INT PRIMARY KEY AUTO_INCREMENT,
    building_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL
);
```

rooms테이블 생성

```
CREATE TABLE rooms (
    room_no INT PRIMARY KEY AUTO_INCREMENT,
    room_name VARCHAR(255) NOT NULL,
    building_no INT NOT NULL,
    FOREIGN KEY (building_no)
        REFERENCES buildings (building_no)
        ON DELETE CASCADE
);
```

```
INSERT INTO buildings(building_name,address)
VALUES('ACME Headquaters','3950 North 1st Street CA 95134'),
      ('ACME Sales','5000 North 1st Street CA 95134');
```

```
SELECT * FROM buildings;
```

building_no	building_name	address
1	ACME Headquaters	3950 North 1st Street CA 95134
2	ACME Sales	5000 North 1st Street CA 95134

```
INSERT INTO rooms(room_name,building_no)
VALUES('Amazon',1),
```

```
('War Room',1),
('Office of CEO',1),
('Marketing',2),
('Showroom',2);
```

```
SELECT * FROM rooms;
```

room_no	room_name	building_no
1	Amazon	1
2	War Room	1
3	Office of CEO	1
4	Marketing	2
5	Showroom	2

```
DELETE FROM buildings
WHERE building_no = 2;
```

```
SELECT * FROM rooms;
```

room_no	room_name	building_no
1	Amazon	1
2	War Room	1
3	Office of CEO	1

building_no2 를 참조하는 모든 행이 자동으로 삭제되었습니다.

MySQL ON DELETE CASCADE작업의 영향을 받는 테이블을 찾는 예

데이터베이스에서 삭제 규칙이 있는 buildings 테이블과 연결된 테이블을 검색

```
USE information_schema;

SELECT
    table_name
```

```
FROM
    referential_constraints
WHERE
        referenced_table_name = 'buildings'
         AND delete_rule = 'CASCADE'
```

table_name
rooms

트랜잭션

MySQL 트랜잭션을 사용하면 데이터베이스에 부분 작업의 결과가 포함되지 않도록 MySQL 작업 집합을 실행할 수 있습니다. 일련의 작업에서 그 중 하나가 실패하면 롤백이 발생하여 데이터베이스를 원래 상태로 복원됩니다. 오류가 발생하지 않으면 전체 문 집합이 데이터베이스에 커밋됩니다.

MySQL 트랜잭션 문

트랜잭션을 시작하려면 START TRANSACTION 문을 사용합니다. BEGIN 또는 BEGIN WORK의 별칭입니다.

현재 트랜잭션을 커밋하고 영구적으로 변경하려면 COMMIT문을 사용합니다.

현재 트랜잭션을 롤백하고 변경 사항을 취소하려면 ROLLBACK문을 사용합니다.

현재 트랜잭션에 대해 자동 커밋 모드를 비활성화하거나 활성화하려면 SET autocommit문을 사용합니다 .

기본적으로 MySQL은 변경 사항을 데이터베이스에 영구적으로 자동 커밋합니다. MySQL이 변경 사항을 자동으로 커밋하지 않도록 하려면 다음 문을 사용합니다.

```
SET autocommit = 0;
```

또는

```
SET autocommit = OFF
```

다음 문을 사용하여 자동 커밋 모드를 명시 적으로 활성화합니다.

```
SET autocommit = 1;
```

또는

```
SET autocommit = ON;
```

Savepoint 를 사용하여 저장 지점을 만들 수 있습니다.

Rollback to, 또는 rollback 을 사용하여 저장지점으로 rollback 할 수 있습니다.

commit 으로 트랜잭션을 확정 지을 수 있습니다.

데이터베이스 생성

데이터로 다른 작업을 수행하기 전에 데이터베이스를 만들어야 합니다. 데이터베이스는 데이터 컨테이너입니다. 연락처, 공급 업체, 고객 또는 생각할 수 있는 모든 종류의 데이터를 저장합니다.

MySQL에서 데이터베이스는 테이블, 뷰, 트리거 등 데이터를 저장하고 조작하는 데 사용되는 개체 모음입니다.

MySQL에서 데이터베이스를 만들려면 CREATE DATABASE 다음 문 을 사용합니다.

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

CREATE DATABASE 명령문 뒤에는 만들려는 데이터베이스 이름이 있습니다. 데이터베이스 이름은 가능한 한 의미 있고 설명적인 이름이어야 합니다.

IF NOT EXISTS 문장의 선택적 절입니다. 이 IF NOT EXISTS절은 데이터베이스 서버에 이미 존재하는 새 데이터베이스를 만드는 오류를 방지합니다

```
CREATE DATABASE classicmodels;
```

이 SHOW DATABASES명령문은 MySQL 데이터베이스 서버의 모든 데이터베이스를 나열합니다. SHOW DATABASES문을 사용하여 만든 데이터베이스를 확인하거나 새 데이터베이스를 만들기 전에 데이터베이스 서버의 모든 데이터베이스를 볼 수 있습니다. 예를 들면 다음과 같습니다.

```
SHOW DATABASES;
```

특정 데이터베이스로 작업하기 전에 USE문을 사용하여 작업 할 데이터베이스를 MySQL에 알려야 합니다.

```
USE database_name;
```

데이터베이스 제거

```
DROP DATABASE [IF EXISTS] database_name;
```

```
CREATE DATABASE IF NOT EXISTS tempdb;

SHOW DATABASES;

DROP DATABASE IF EXISTS temp_database;
```

CREATE TABLE

CREATE TABLE문을 사용하면 데이터베이스에 새 테이블을 만들 수 있습니다.

다음은 CREATE TABLE 명령문 의 기본 구문을 보여줍니다 .

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

생성시 테이블에 대한 스토리지 엔진을 선택적으로 지정할 수 있습니다 ENGINE. InnoDB 및 MyISAM과 같은 모든 스토리지 엔진을 사용할 수 있습니다. 스토리지 엔진을 명시 적으로 선언하지 않으면 MySQL은 기본적으로 InnoDB를 사용합니다.

InnoDB는 MySQL 버전 5.5 이후 기본 스토리지 엔진이 되었습니다. InnoDB 스토리지 엔진은 ACID 트랜잭션, 참조 무결성 및 크래시 복구와 같은 관계형 데이터베이스 관리 시스템의 많은 이점을 제공합니다. 이전 버전에서 MySQL은 MyISAM을 기본 스토리지 엔진으로 사용했습니다.

열 정의 구문은 다음과 같습니다.

```
column_name data_type(length)
          [NOT NULL] [DEFAULT value] [AUTO_INCREMENT] column_constraint;
```

AUTO_INCREMENT는 새로운 행 추가마다 자동으로 컬럼의 값이 하나씩 증가되는 것을 나타냅니다. 각 테이블에는 최대 하나의 AUTO_INCREMENT 열이 있습니다.

열 목록 뒤에 UNIQUE, CHECK, PRIMARY KEY 및 FOREIGN KEY 와 같은 테이블 제약 조건을 정의 할 수 있습니다.

1) MySQL CREATE TABLE 예

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)  ENGINE=INNODB;


DESCRIBE tasks;
```

2) CREATE TABLE 기본 키가 있는 예

각 작업에 체크리스트 또는 할 일 목록이 있다고 가정합니다. 작업 체크리스트를 저장하기 위해 checklists다음과 같은 이름의 새 테이블을 생성 할 수 있습니다.

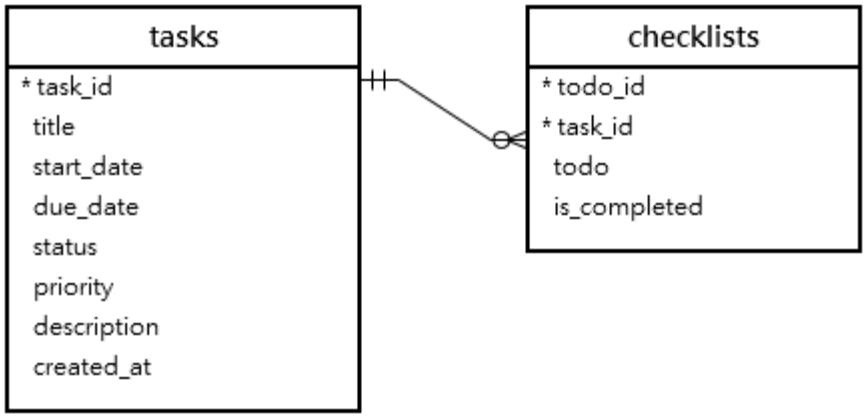
```
CREATE TABLE IF NOT EXISTS checklists (
    todo_id INT AUTO_INCREMENT,
    task_id INT,
    todo VARCHAR(255) NOT NULL,
    is_completed BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (todo_id , task_id),
    FOREIGN KEY (task_id)
        REFERENCES tasks (task_id)
        ON UPDATE RESTRICT ON DELETE CASCADE
);
```

테이블 checklists에는 두 개의 열로 구성된 기본 키가 있습니다. 따라서 테이블 레벨 제약 조건을 사용하여 기본 키를 정의 했습니다.

```
PRIMARY KEY (todo_id, task_id)
```

또한 테이블 task_id의 열을 참조하는 외래 키 열이며 외래 키 제약 조건을 사용하여 이 관계를 설정했습니다.

```
FOREIGN KEY (task_id)
    REFERENCES tasks (task_id)
    ON UPDATE RESTRICT
    ON DELETE CASCADE
```



Alter Table

#테이블에 컬럼을 추가

ALTER TABLE table_name

ADD

new_column_name column_definition

[FIRST | AFTER column_name]

FIRST | AFTER column_name 테이블에서 새 열의 위치를 기존 열 (ATER column_name) 뒤에 또는 첫 번째 열 FIRST) 로 열을 추가 할 수 있습니다 . 이 절을 생략하면 테이블의 열 목록 끝에 열이 추가됩니다.

테이블의 열을 수정합니다.

ALTER TABLE table_name

MODIFY column_name column_definition

[FIRST | AFTER column_name];

컬럼을 삭제 합니다.

ALTER TABLE table_name

DROP COLUMN column_name;

테이블 이름을 바꿉니다.

ALTER TABLE table_name

RENAME TO new_table_name;

DROP TABLE

DROP TABLE 기본 구문입니다 .

DROP TABLE [IF EXISTS] table_name [, table_name] ...

[RESTRICT | CASCADE]

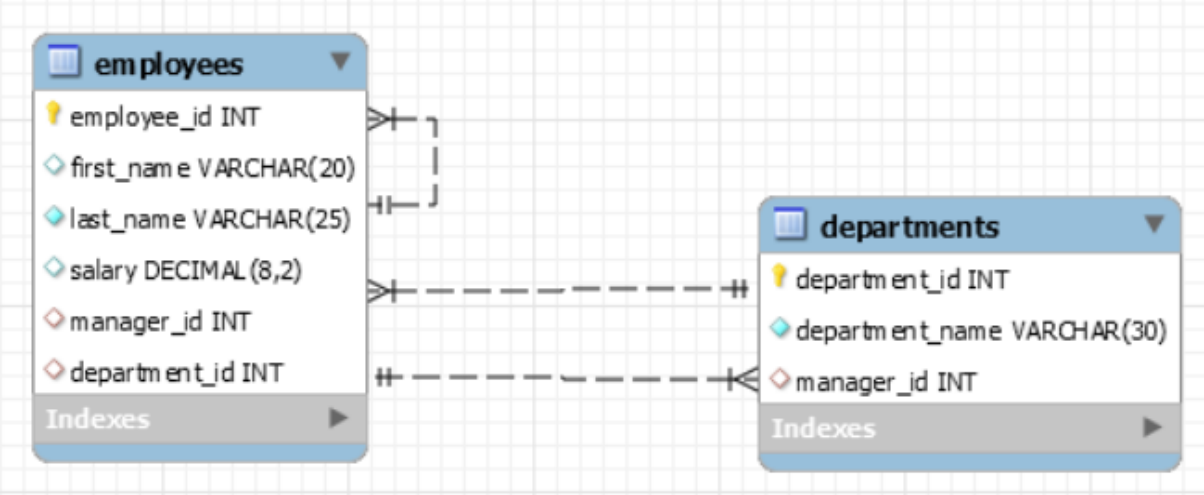
이 DROP TABLE명령문은 데이터베이스에서 테이블과 데이터를 영구적으로 제거합니다. MySQL에서는 단일 DROP TABLE문을 사용하여 여러 테이블을 제거 할 수 있습니다. 각 테이블은 쉼표 (,)로 구분됩니다.

이 IF EXISTS옵션은 테이블이 있는 경우에만 조건부로 테이블을 삭제합니다

이 DROP TABLE문은 테이블 만 삭제합니다. 테이블과 관련된 특정 사용자 권한은 제거하지 않습니다. 따라서 삭제 된 테이블과 동일한 이름으로 테이블을 생성하면 MySQL은 기존 권한을 새 테이블에 적용하여 보안 위험을 초래할 수 있습니다.

RESTRICT및 CASCADE 옵션은 MySQL의 미래 버전의 예약된 키워드입니다.

mydatabase라는 스키마를 만들고 아래와 같이 테이블을 구성하세요.


employees
employee_id INT
first_name VARCHAR(20)
last_name VARCHAR(25)
salary DECIMAL(8,2)
manager_id INT
department_id INT
Indexes
departments
department_id INT
department_name VARCHAR(30)
manager_id INT
Indexes

데이터 타입

컬럼의 데이터 타입을 선정하는 작업은 물리 모델링에서 빼놓을 수 없는 중요한 작업이다. 컬럼의 데이터 타입과 길이를 선정할 때 가장 주의해야 할 사항은 다음과 같다.

- 저장되는 값의 성격에 맞는 최적의 타입을 선정
- 가변 길이 컬럼은 최적의 길이를 지정
- 조인 조건으로 사용되는 컬럼은 똑같은 데이터 타입을 선정

컬럼의 데이터 타입을 선정할 때 실제 저장되는 값의 특성을 고려하지 않고 가능한 최대 길이 값을 기준으로 컬럼의 길이를 선택하는 것이 일반적이다. 하지만 무분별하게 컬럼의 길이가 크게 선정되면 디스크의 공간은 물론 메모리나 CPU 의 자원도 함께 낭비된다. 또한 그로 인해 SQL 의 성능이 저하되는 것은 당연한 결과일 것이다.

또한 컬럼의 타입이 잘못 선정되거나 길이가 너무 부족하다면 서비스 도중에 스키마 변경이 필요할 수 도 있다. 그런데 스키마 변경 작업은 서비스 중지나 읽기 전용 모드로의 전환 작업이 필요해질 수도 있다. 데이터 타입의 길이는 너무 넉넉하게 선택해도 문제가 되고, 부족하게 선택해도 문제가 된다. 항상 실제로 저장되는 값의 성격을 정확히 분석하고 최적의 타입과 길이를 선정하는 것이 중요하다.

문자열(CHAR 와 VARCHAR)

문자열 컬럼을 사용할 때는 우선 CHAR 타입과 VARCHAR 타입 중 어떤 타입을 사용할지 결정해야 한다. 그래서 CHAR 와 VARCHAR 타입의 차이가 무엇이고 어떤 타입을 사용하는 것이 좋은지에 관한 질문도 많은 편이다. 처음 데이터베이스를 사용할 때는 둘 중에서 뭘 선택해야 할지 고민하다가 결국 VARCHAR 만 쭉 사용하는 사람들도 있다. 하지만 지금까지 모든 DBMS 에서 CHAR 나 VARCHAR 타입을 구분해서 제공하는 것을 보면 그만큼의 장단점을 가지고 있음을 짐작할 수 있을 것이다. 우선 제 장 공간과 비교 방식의 관점에서 CHAR 와 VARCHAR 를 한번 비교해보고, MySQL 내부적으로 어떤 차이가 있는지도 한번 살펴보자.

저장 공간

우선 CHAR 와 VARCHAR 의 가장 큰 공통점은 문자열을 저장할 수 있는 데이터 타입이라는 점이고, 가장 큰 차이는 고정 길이인지 가변 길이인지 여부다.

- 고정 길이는 실제 입력되는 컬럼 값의 길이에 따라 사용하는 저장 공간의 크기가 변하지 않는다. CHAR 타입은 이미 저장 공간의 크기가 고정적이다. 실제 저장된 값의 유효 크기가 얼마인지 별도로 저장할 필요가 없으므로 추가로 공간이 필요하지 않다.

- 이 가변 길이는 최대로 저장할 수 있는 값의 길이는 제한돼 있지만, 그 이하 크기의 값이 저장되면 그만큼 저장공간이 줄어든다. 하지만 VARCHAR 타입은 저장된 값의 유효 크기가 얼마인지를 별도로 저장해 둬야 하므로 1~2 바이트의 저장 공간이 추가로 더 필요하다.

하나의 글자를 저장하기 위해 CHAR(1)과 VARCHAR(1) 타입을 사용할 때 실제 사용되는 저장 공간의 크기를 한번 살펴보자. 우선 두 문자열 타입 모두 한 글자를 저장할 때 사용하는 문자집합에 따라 실제 저장 공간은 1~3 바이트까지 사용하게 된다. 여기서 하나의 글자가 CHAR 타입에 저장될 때는 추가적인 공간이 더 필요하지 않지만 VARCHAR 타입에 저장할 때는 문자열의 길이를 관리하기 위한 ¦ 1~2 바이트의 공간을 추가적으로 더 사용한다. VARCHAR 타입의 길이가 255 바이트 이하이면 1 바이트만 사용하고 타입의 길이가 256 바이트 이상으로 설정되면 2 바이트를 길이를 저장하는 데 사용한다. VARCHAR 타입의 최대 길이는 2 바이트로 표현할 수 있는 이상은 사용할 수 없다. 즉 VARCHAR 타 입의 최대 길이는 65,536 이상으로 설정할 수 없다.

MySQL 에서는 하나의 레코드에서 TEXT 와 BLOB 타입을 제외한 컬럼의 전체 크기가 65KB 를 초과할 수 없다. 만약 테이블에 VARCHAR 타입의 컬럼 하나만 있다면 이 VARCHAR 타입은 최대 64KB 크기의 데이터를 저장할 수 있다. 하지만 이미 다른 컬럼에서 40KB 의 크기를 사용하고 있다면 VARCHAR 타입은 24KB 만 사용 할 수 있다. 이때 만약 24KB 를 초과하는 크기의 VARCHAR 타입을 생성하려고 하면 에러가 발생하거나 자동으로 VARCHAR 타입이 TEXT 타입으로 대체된다. 그래서 컬럼을 새로 추가할 때는 VARCHAR 타입이 TEXT 타입으로 자동적으로 변환되지 않았는지 확인해 보는 것이 좋다.

문자열 타입의 저장 공간을 언급할 때는 1 문자와 1 바이트를 구분해서 사용하고 있다. 1 문자는 실제 저장되는 값의 문자 법에 따라 1~3 바이트까지 공간을 사용할 수 있기 때문이다. 위의 VARCHAR 타입의 컬럼 하나만 가지는 테이블의 예에서 VARCHAR 타입은 최대 64KB 크기의 데이터를 저장할 수 있다고 했는데, 이 수치는 바이트 수를 의미하므로 실제 65536 글자까지 저장할 수 있다는 것은 아니다. 실제 저장되는 문자가 아시아권의 언어라면 저장 가능한 글자 수는

반으로 줄고, UTF-8 문자를 저장한다면 실제 저장 가능한 글자 수는 1/3 로 줄어들 것이다.

문자열 값의 길이가 항상 일정하다면 CHAR 를 사용하고 가변적이라면 VARCHAR 를 사용하는 것이 일반적이다. 왜 길이가 고정적일 때 CHAR 를 사용하면 좋을까? VARCHAR 타입을 선택해도 기껏 디스크에서 1 바이트만 더 사용할 뿐인데, 이렇게 고민해가면서 시간을 투자할 가치가 있는 것일까?

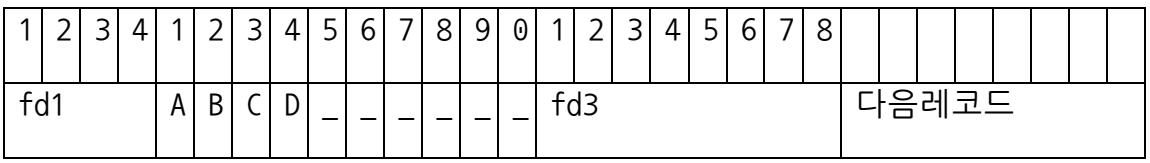
실제 문자열 값의 길이가 정적이냐 가변적이냐 만으로 CHAR 와 VARCHAR 타입을 결정하는 것은 적절하지 않다. CHAR 타입과 VARCHAR 타입을 결정할 때 중요한 판단 기준은 다음과 같다.

- 저장되는 문자열의 길이가 대개 비슷한가?
- 컬럼의 값이 자주 변경되는가?

CHAR 와 VARCHAR 타입의 선택 기준은 값의 길이도 중요하지만, 해당 컬럼의 값이 얼마나 자주 변경되는지가 타입 선택의 기준이 되어야 한다. 컬럼의 값이 얼마나 자주 변경되는지가 왜 중요한지 그림으로 한번 살펴보자. 우선 다음과 같이 테스트용 테이블이 있고, 그 테이블에 레코드 1 건이 저장된다고 가정해보자.

```
CREATE TABLE tb_test (

fd1 INT,

fd2 CHAR(10),

fd3 DATETIME

);

INSERT INTO tb_test (fd1, fd2, fd3)

VALUES (1, 'ABCD', '2011-06-27 11:02:11');
```

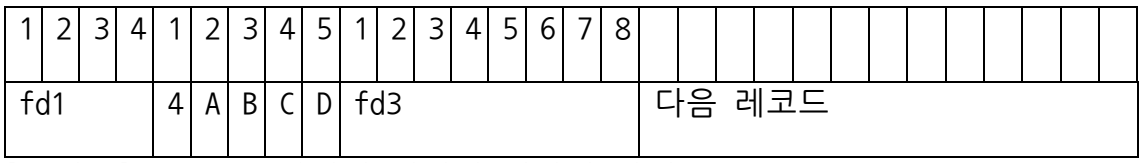
tb_test 테이블에 레코드 1 건을 저장하면 내부적으로 디스크에는 그림 1 과 같이 저장될 것이다.



[그림 1] CHAR 타입의 컬럼이 저장된 상태

fdl 컬럼은 INTEGER 타입이므로 고정 길이로 4 바이트를 사용하며, fd3 또한 DATETIME 이므로 고정 길이로 8 바이트를 사용한다. 지금 여기서 관심사는 fdl 과 fd3 컬럼이 아니라 그 사이에 위치한 fd2 컬럼이다. fd2 컬럼이 사용하고 있는 공간을 눈여겨보자. 그림 1 에서 fd2 컬럼은 정확히 10 바이트를 사용하면서 앞쪽의 4 바이트만 유효한 값으로 채워졌고 나머지는 공백 문자로 채워져 있다(그림 1 에서 공백 문자(Space character)는 이해를 돕기 위해 "_" 문자로 대체해서 표기했다).

그러면 이번에는 tb test 테이블의 fd2 컬럼만 CHAR(10) 대신 VARCHAR(10)으로 변경해서 똑 같은 데이터를 저장했을 때는 디스크에 어떻게 저장되는지 그림 2 로 살펴보자.



[그림 2] VARCHAR 타입의 컬럼이 저장된 상태

fid1 컬럼과 fd3 컬럼 사이에서 fd2 컬럼은 5 바이트의 공간을 차지하고 있는데, 첫 번째 바이트에는 저장된 컬럼 값의 유효한 바이트 수인 숫자 4(문자 '4'가 아님)가 저장되고 두 번째 바이트부터 다섯 번째 바이트까지 실제 컬럼 값이 저장된다.

그림 1 이나 그림 2 는 이미 대략 예측하고 있는 사실일 것이다. 하지만 중요한 것은 레코드 한 건이 저장된 상태가 아니라 fd2 컬럼의 값이 변경될 때 어떤 현상이 발생하느냐다. fd2 컬럼의 값을 "ABCDE"로 UPDATE 했다고 가정해 보자.

- CHAR(10) 타입을 사용하는 그림 15-1 에서는 fd2 컬럼을 위해 공간이 10 바이트가 준비돼 있으므로 그냥 변경되는 컬럼의 값을 업데이트만 하면 된다.

- VARCHAR(10) 타입을 사용하는 그림 15~2 에서는 fd2 컬럼에 4 바이트밖에 저장할 수 없는 구조로 만들어져 있다. 그래서 "ABCDE"와 같이 길이가 더 큰 값으로 변경될 때는 레코드 자체를 다른 공간으로 옮기거나(Row migration) 컬럼 값의 나머지 부분을 다른 공간에 저장(Row chaining)해야 한다.

물론 주민등록번호처럼 항상 값의 길이가 고정적일 때는 당연히 CHAR 타입을 사용해야 한다. 또한 값이 2~3 바이트씩 차이가 나더라도 자주 변경될 수 있는 부서 번호나 게시물의 상태 값 등은 CHAR 타입을 사용하는 것이 좋다. 자주 변경돼도 레코드가 물리적으로 다른 위치로 이동시키거나 분리하지 않아도 되기

때문이다. 레코드의 이동이나 분리는 CHAR 타입으로 인해 발생하는 2~3 바이트 공간 낭비 보다 더 큰 공간이나 자원을 낭비하게 만든다.

문자열 데이터 타입을 사용할 사용할 때 또 하나 주의해야 할 사항이 있다. CHAR 나 VARCHAR 키워드 뒤에 인자로 전달하는 숫자 값의 의미를 알아야 한다는 점이다. 다른 DBMS 에 익숙한 사용자에게는 상당히 혼란스러울 수 있는데, MySQL 에서 CHAR 나 VARCHAR 뒤에 지정하는 숫자는 그 컬럼의 바이트 크기가 아니라 문자의 수를 의미한다. 즉 CHAR(10) 또는 VARCHAR(10)으로 컬럼을 정의하면, 이 컬럼은 10 바이트를 저장할 수 있는 공간이 아니라 10 글자(문자)를 저장할 수 있는 공간을 의미한다. 그래서 CHAR(10) 타입을 사용하더라도 이 컬럼이 실제적으로 디스크나 메모리에서 사용하는 공간은 각각 달라진다.

- 일반적으로 영어를 포함한 서구권 언어는 각 문자가 1 바이트를 사용하므로 10 바이트를 사용한다.
- 한국이나 일본어와 같은 아시아권 언어는 각 문자가 최대 2 바이트를 사용하므로 20 바이트를 사용한다.
- UTF-8 과 같은 유니코드는 최대 3 바이트까지 사용하므로 30 바이트까지 사용할 수 있다.

사실 UTF-8 은 저장하는 문자에 따라 최소 1 바이트부터 최대 4 바이트까지 사용할 수 있다. 주로 4 바이트를 사용하는 문자는 확장 문자로 구분하기도 한다. 하지만 MySQL 5.1 버전까지는 UTF-8 의 모든 문자를 저장할 수 있는 것이 아니라 3 바이트를 사용하는 UTF-8 문자만 저장할 수 있었다. 그 밖에 4 바이트까지 사용하는 UTF-8 문자를 MySQL 5.1 이하의 버전에서 저장하려고 하면 " 알 수 없는 문자 " 라는 에러가 출력된다. 대표적으로 애플의 스마트 폰 문자가 있어서 자주 이런 문제의 원인이 되기도 한다. 4 바이트를 사용하는 UTF-8 문자는 MySQL 5.5 이상의 버전에서부터 사용할 수 있으므로 필요하다면 MySQL 5.1 보다는 MySQL 5.5 이후 버전을 사용하자.

TEXT 데이터

TEXT는 1 바이트에서 4 GB까지 길이가 긴 텍스트 문자열을 저장하는데 유용하지만 느립니다.

MySQL은 네 가지 제공 TEXT 유형: TINYTEXT, TEXT, MEDIUMTEXT,와 LONGTEXT.

다음은 TEXT문자를 저장하는 데 1 바이트를 사용하는 문자 집합을 사용한다는 가정하에 각 유형의 크기를 보여줍니다.

TINYTEXT - 255 바이트 (255 자)

TEXT - 64KB (65,535 자)

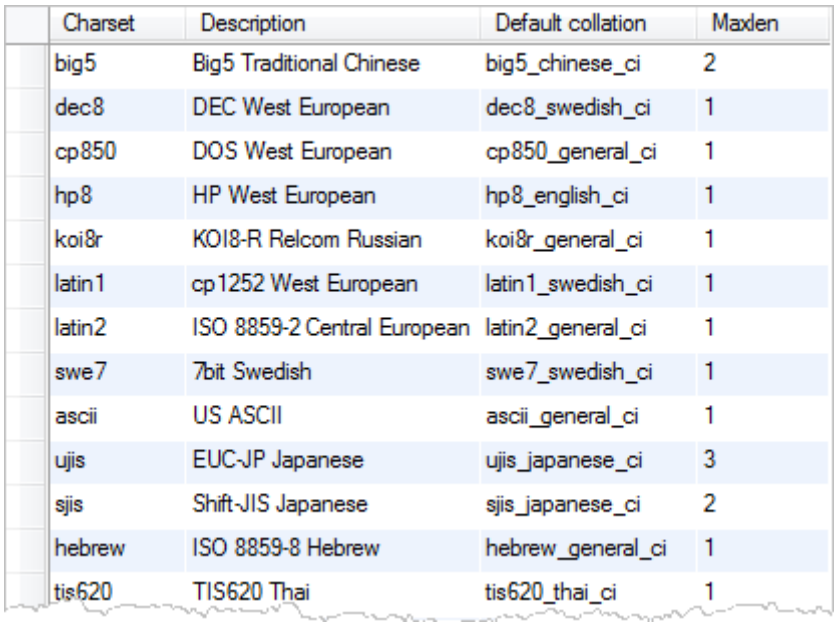
MEDIUMTEXT - 16MB (16,777,215 자)

LONGTEXT - 4GB (4,294,967,295 자)

MySQL CharacterSet

데이터베이스 서버에서 사용 가능한 모든 문자 집합을 가져옵니다.

```
SHOW CHARACTER SET;
```



Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1

MySQL의 기본 문자 집합은 latin1. 여러 언어의 문자를 단일 열에 저장하려는 경우 유니코드 문자 집합을 사용할 수 있습니다.

Maxlen열의 값은 문자 집합의 문자가 보유하는 바이트 수를 지정합니다.

MySQL은 LENGTH 문자열 길이를 바이트 단위 CHAR_LENGTH로 가져 오는 함수와 문자열 길이를 문자 단위로 가져오는 함수를 제공합니다. 문자열에 멀티 바이트 문자가 포함 된 경우 LENGTH 함수의 결과는 CHAR_LENGTH()함수의 결과보다 큽니다.

```
SET @str = CONVERT('안녕🙏🙏' USING utf8mb4);

SELECT LENGTH(@str), CHAR_LENGTH(@str);
```

이 CONVERT함수는 문자열을 특정 문자 집합으로 변환합니다.

MySQL은 서로 다른 문자 집합간에 문자열을 변환 할 수 있는 두 가지 함수를 제공합니다.

CONVERT함수 의 구문은 다음과 같습니다.

```
CONVERT(expression USING character_set_name)
```

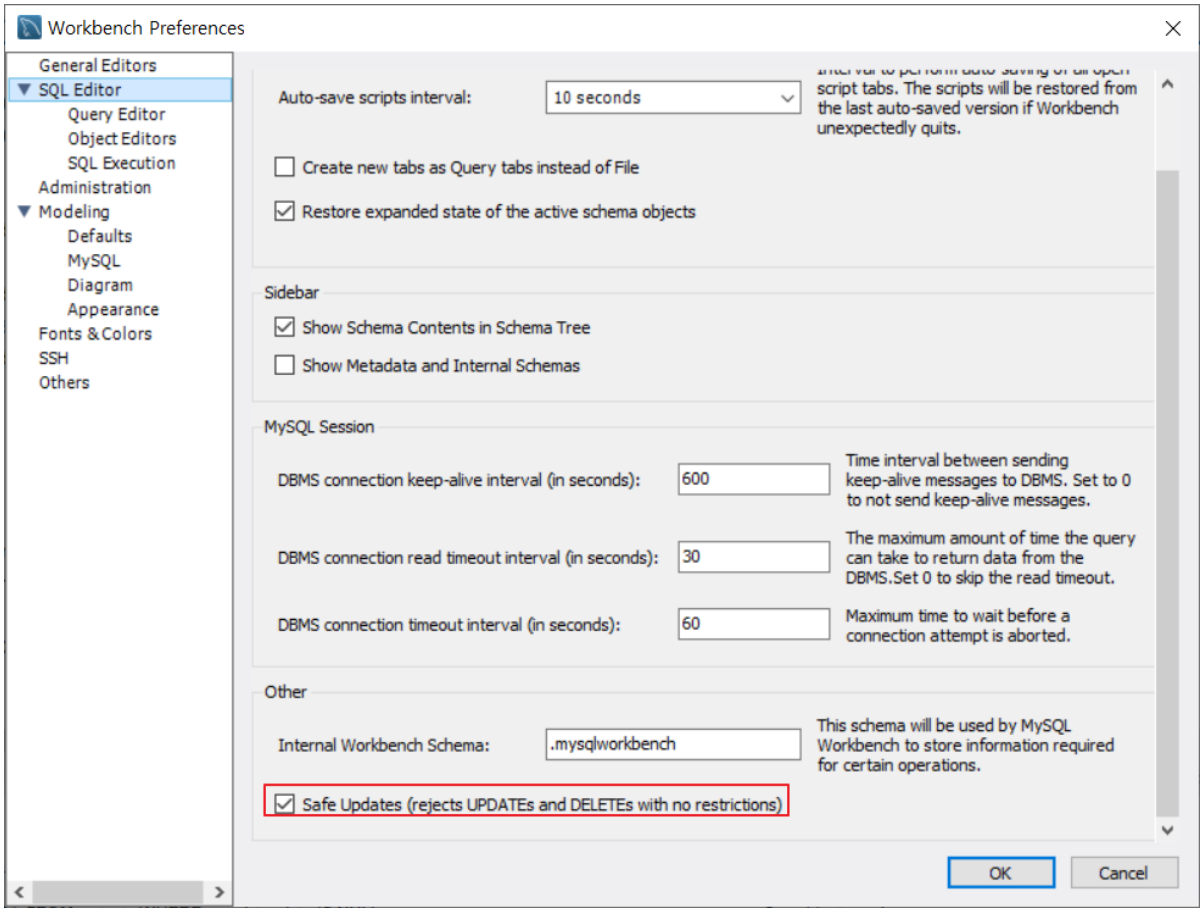
CAST기능은 CONVERT기능과 유사하며 문자열을 다른 문자 세트로 변환합니다.

CAST(string AS character_type CHARACTER SET character_set_name)

Appendix A. 주요 에러 대처 방안

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences

대처 1.



Edit > Preparences… > SQL Editor > Safe Updats (rejects UPDATEs and DELETEs with no restrictions)의 체크 해제

대처 2.

SET SQL_SAFE_UPDATES = 0;

다시 안전 수정모드로 바꿀 때는 SET SQL_SAFE_UPDATES = 1;

Appendix B. 권장 MySQL 서적

- 데이터 분석을 위한 SQL 레시피 : http://www.yes24.com/24/goods/59411396
- Real MySQL : http://www.yes24.com/24/goods/6960931
- DBA를 위한 MySQL 운영 기술 : http://www.yes24.com/24/goods/19242110
- Real MariaDB : http://www.yes24.com/24/Goods/12653486
- MariaDB 실전 활용 노하우 : http://www.yes24.com/24/Goods/12982561
- 이것이 MySQL이다 : http://www.yes24.com/Product/Goods/90118480
- Oracle, PostgreSQL, MySQL Core Architecture : http://www.yes24.com/24/goods/35095380
- Oracle, PostgreSQL, MySQL Core Architecture 2 : http://www.yes24.com/24/goods/46643302

Appendix C. Oracle DBMS와 MySQL DBMS의 차이.

ISSUE	Oracle11g DBMS	MySQL 8.0 DBMS
NUMERIC Data Type	정수 및 고정소수점 타입 NUMBER	정수 INT … 고정소수점 DECIMAL
CHARACTER TYPE	VARCHAR2 최대 4000 BYTE. 그 이상은 CLOB으로 4G이상 지원.	VARCHAR 레코드당 최대 64KB. TEXT 타입 최대 64KB. LONGTEXT 사용시 최대 4G지원.
SEQUENCE	OBJECT 단위	COLUMN 속성
CONSTRAINT REFERENCE	참조 키에 컬럼 명이 일치할 경우 생략가능	참조 키에 컬럼 명을 반드시 명시적 기술
Disable Integrity Constraint	ALTER TABLE 테이블명 DISABLE CONSTRAINT 참조제약명;	SET FOREIGN_KEY_CHECKS = 0;
Enable Integrity Constraint	ALTER TABLE 테이블명 ENABLE CONSTRAINT 참조제약명;	SET FOREIGN_KEY_CHECKS = 1;
DATE FORMAT	TO_DATE('02/18/2021 00:00:00', 'MM/DD/YYYY HH24:MI:SS')	STR_TO_DATE('02/18/2021 00:00:00', '%m/%d/%Y %H:%i:%s')
TABLE ALIAS	AS를 붙일 수 없음.	AS를 붙일 수 있음. COLUMN ALIAS와 유사.
INLINE VIEW	TABLE ALIAS에 생략 가능	참조하지 않아도 TABLE ALIAS 생략불가능.
Recyclebin	10g 이후로 존재	없음.
JOIN	ORACLE JOIN, ANSI JOIN 2가지 지원. ORACLE JOIN은 FULL JOIN 지원하지 않음.	ANSI JOIN 지원. JOIN을 콤마로 사용 가능. FULL JOIN 지원하지 않음. NATURAL JOIN, JOIN USING 잘 지원 됨.
GROUP BY	ALIAS 불가능	ALIAS 가능. SELECT 절의 ALIAS도 참조 가능.
SQL Parsing 순서	SELECT, GROUP BY, Having은 불명확한 부분이 있음.	from – where – select – group by – having – order by - limit 순서가 명확함
LIKE 대소문자구분	구분	구분하지 않음
대소문자 구분	성능 이슈 됨.	*성능 이슈를 매뉴얼에서 찾을 수 없음.(테스트 필요)
DELETE	DELETE [FROM] tablename	DELETE FROM tablename FROM 생략시 에러
TOP N 쿼리	12C 부터 offset과 fetch 사용. 11g 이하 버전은 중첩된 inline view와 rownum을 이용하여 직접 작성.	LIMIT [offset,] row_count PostgreSQL LIMIT 과의 호환성을 위해 OFFSET 대체 절을 제공
ONE QUERY	8.1.6 이후 ROLLUP, CUBE, GROUPING SET등 지원	ROLLUP(), GROUPING() 지원. 8.1버전에서 CUBE() 지원 예정.