jinyPHP 2권 - 012. 조건

**12. 조건**

====================

테이블에는 다수의 데이터 레코드 열(rows)를 포함하고 있습니다. 우리가 복잡한 데이터베이스 시스템을 사용하는 이유는 수많은 데이터 중에서 자신이 원하는 정보만을 추출하여 데이터 처리하기 위해서 입니다.

테이블에서 원하는 데이터만을 추출하기 위해서는 데이터를 검색하고 필터일 하는 조건이 필요합니다. 또한 다양한 조건을 처리 하기 위해서 연산자를 같이 사용을 하기도 합니다.

**11.1 where**

====================

where 조건 키워드는 데이터를 검색(Select), 수정(Update), 삭제(Delete) 문에서 공통적으로 사용이 가능한 조건 지정 키워드 입니다.

where 키워드는 SELECT, UPDATE, DELETE 쿼리 다음에 붙여서 같이 사용을 할 수 있습니다.

**쿼리문법)**

|  |
| --- |
| SELECT , UPDATE, DELETE + WHERE 조건; |

**11.1.1 쿼리 실습**

====================

데이터를 조회하는 SELECT 쿼리 명령문에 조건을 추가하여 데이터를 검색해 보도록 하겠습니다.

select \* from members **where City = 'seoul'**;

회원(members) 테이블에서 도시가 seoul인 회원만을 출력해 보도록 하겠습니다. 기존 select 쿼리 뒤에 조건을 연결하여 지정을 하면 됩니다. 이때 = 기호는 정확인 문자열이 일치하는 조건을 뜻합니다.

|  |
| --- |
| mysql> **select \* from members where City = 'seoul';**  +----+----------+-----------+-------------+-------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+-------------+-------+---------+  | 1 | hojin | lee | shinchon | seoul | korea |  | 3 | james | kim | youngdun-po | seoul | korea |  | 4 | eric | jung | seocho-gu | seoul | korea |  +----+----------+-----------+-------------+-------+---------+  3 rows in set (0.00 sec) |

출력결과를 확인해 봅니다. 도시가 seoul 인 회원만 추출해서 화면에 출력합니다.

**11.2 연산자**

====================

앞전에서 간단히 데이터 처리를 위한 조건을 추가해 보았습니다. 도시가 seoul인 회원을 찾기 위해서 = 연산 기호를 사용하였습니다. WHERE 은 컬럼과 값을 다양한 연산자를 통하여 조건을 설정할 수 있습니다.

MYSQL에서는 9가지의 다양한 연산자를 지원합니다.

**11.2.1 Equal**

====================

컬럼의 동일한 값을 비교할 때는 = 기호를 사용합니다. = 기호는 컬럼의 데이터 값이 우측에 지정한 값과 정확히 일치하였을 경우에 조건이 성립이 됩니다. = 기호는 데이터 처리 조건으로 가장 많이 이용하는 기호 입니다.

**쿼리~ where Id = 7;**

Id 값이 정확히 7 숫자인 데이터를 검색하여 결과를 출력합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id =7;**  +----+-------+  | Id | code |  +----+-------+  | 7 | O\_007 |  +----+-------+  1 row in set (0.00 sec) |

PHP 코드를 통하여 검색조건 처리를 실험해 보도록 합니다.

**PHP 예제)**

mysql.class.php 파일에 메서드 예제를 추가합니다.

|  |
| --- |
| public function selectRowss($queryString)  {  if ($result = mysqli\_query($this->dbcon, $queryString)) {  $this->msgEcho("쿼리성공] ".$queryString);  $rowss = "";  $row\_cnt = mysqli\_num\_rows($result);  for ($i=0; $i<$row\_cnt; $i++) {  $rowss[$i] = mysqli\_fetch\_object($result);  }  $result->free();  return $rowss;  } else {  $this->msgEcho("Error] ".$queryString);  }  } |

**예제파일) where-01.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "SELECT FirstName, LastName FROM members where **Id='3'**";  if($rowss = $db->selectRowss($queryString)){  echo "tables fields = ". count($rowss) . "<br>";    for($i=0;$i<count($rowss);$i++){  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

출력화면)

mysql connected!

쿼리성공] SELECT FirstName, LastName FROM members where Id='3'

tables fields = 1

0=stdClass Object ( [FirstName] => kim [LastName] => james )

**11.2.2 NOT Equal**

====================

“같다”를 의미하는 = 기호의 반대 입니다. 컬럼의 값이 일치하지 않을 때 비교하는 연산자는 <> 를 사용합니다.

**쿼리~ where Id <> 7’;**

쿼리에서 Id 값이 7이 아닌 모든 내용을 찾아서 결과를 출력합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id <> 7;**  +----+-------+  | Id | code |  +----+-------+  | 1 | O\_001 |  | 2 | O\_002 |  | 3 | O\_003 |  | 4 | O\_004 |  | 5 | O\_005 |  | 6 | O\_006 |  | 8 | O\_008 |  | 9 | O\_009 |  | 10 | O\_010 |  +----+-------+  9 rows in set (0.00 sec) |

출력결과를 확인해 보면 Id가 7인 경우만 빼고 모두 출력이 되었습니다.

PHP 코드를 통하여 결과를 실험해 보도록 합니다.

**예제파일) where-02.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "SELECT FirstName, LastName FROM members where **Id<>'3'**";  if ($rowss = $db->selectRowss($queryString)) {  echo "tables fields = ". count($rowss) . "<br>";    for ($i=0;$i<count($rowss);$i++) {  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

**출력화면)**

mysql connected!

쿼리성공] SELECT FirstName, LastName FROM members where Id<>'3'

tables fields = 5

0=stdClass Object ( [FirstName] => lee [LastName] => hojin )

1=stdClass Object ( [FirstName] => [LastName] => jiny )

2=stdClass Object ( [FirstName] => jiny [LastName] => 1234 )

3=stdClass Object ( [FirstName] => jiny [LastName] => 1234 )

4=stdClass Object ( [FirstName] => jiny [LastName] => 123400 )

**11.2.3 크기 비교**

====================

where 조건 설정시 크기를 비교하는 부등호 기호를 같이 사용할 수 있습니다.

> 는 왼쪽의 값이 오른쪽 값 보다 큰 값인지를 비교하는 연산자 입니다.

>= 는 왼쪽의 값이 오른쪽 값 보다 큰 값, 또는 같은 값 인지를 비교하는 연산자 입니다.

**쿼리~ where Id > 7;**

**쿼리~ where Id >= 7;**

Id 값이 7보다 큰 값만 출력해 봅니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id >7;**  +----+-------+  | Id | code |  +----+-------+  | 8 | O\_008 |  | 9 | O\_009 |  | 10 | O\_010 |  +----+-------+  3 rows in set (0.00 sec) |

7보다 큰 값이기 때문에 7은 같이 출력이 되지 안았습니다. 7을 포함하려고 하면 >= 등호기호도 같이 작성을 합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id >=7;**  +----+-------+  | Id | code |  +----+-------+  | 7 | O\_007 |  | 8 | O\_008 |  | 9 | O\_009 |  | 10 | O\_010 |  +----+-------+  4 rows in set (0.00 sec) |

< 는 왼쪽의 값이 오른쪽 보다 큰 값인지를 비교하는 연산자 입니다.

<= 는 왼쪽의 값이 오른쪽 보다 큰 값, 또는 작은값 인지를 비교하는 연산자 입니다.

**쿼리~ where Id < 5;**

**쿼리~ where Id <= 5;**

Id값이 5보다 작은 값만 출력을 합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id <5;**  +----+-------+  | Id | code |  +----+-------+  | 1 | O\_001 |  | 2 | O\_002 |  | 3 | O\_003 |  | 4 | O\_004 |  +----+-------+  4 rows in set (0.00 sec) |

5보다 작은 값이기 때문에 5를 출력하지는 안았습니다. 5를 포함할려고 하면 <= 처럼 등호 기호를 같이 사용을 합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id <=5;**  +----+-------+  | Id | code |  +----+-------+  | 1 | O\_001 |  | 2 | O\_002 |  | 3 | O\_003 |  | 4 | O\_004 |  | 5 | O\_005 |  +----+-------+  5 rows in set (0.00 sec) |

PHP 코드를 통하여 크기비교를 실습해 보도록 합니다.

**예제파일) where-03.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "SELECT Id, FirstName, LastName FROM members where Id**>=**3";  if ($rowss = $db->selectRowss($queryString)) {  echo "tables fields = ". count($rowss) . "<br>";    for ($i=0;$i<count($rowss);$i++) {  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

**출력화면)**

mysql connected!

쿼리성공] SELECT Id, FirstName, LastName FROM members where Id>=3

tables fields = 4

0=stdClass Object ( [Id] => 3 [FirstName] => kim [LastName] => james )

1=stdClass Object ( [Id] => 5 [FirstName] => jiny [LastName] => 1234 )

2=stdClass Object ( [Id] => 6 [FirstName] => jiny [LastName] => 1234 )

3=stdClass Object ( [Id] => 7 [FirstName] => jiny [LastName] => 123400 )

**11.2.4 BETWEEN ~ AND ~**

====================

연산기호를 사용한 조건은 어떠한 방향성을 설정하여 검색하는 방법입니다. 이와 달리 BETWEEN 연산자는 입력된 값의 사이의 범위를 지정할 수 조건입니다. 범위로 사용할 수 있는 값은 숫자, 텍스트 또는 날짜 데이터 타입 입니다.

BETWEEN 연산자는 범위를 설정하기 때문에 시작값과 끝값이 필요로 합니다.

**쿼리문법)**

|  |
| --- |
| 쿼리~ WHERE 컬럼명 BETWEEN 시작값 AND 끝값; |

BETWEEN 연산자는 WHERE 컬럼 >= 값1 and 컬럼 <= 값2 로 바꾸어 사용할 수 있습니다. 같은 의미 입니다. 즉, >= 와 <= 사이를 말합니다.

**select \* from orders where Id between 3 and 8;**

Id가 3 와 8 사이의 값을 출력합니다.

|  |
| --- |
| mysql> select \* from orders WHERE Id between 3 and 8;  +----+-------+  | Id | code |  +----+-------+  | 3 | O\_003 |  | 4 | O\_004 |  | 5 | O\_005 |  | 6 | O\_006 |  | 7 | O\_007 |  | 8 | O\_008 |  +----+-------+  6 rows in set (0.00 sec) |

3부터 8까지를 출력합니다. BETWEEN은 =를 포함하기 때문에 3과 8도 같이 출력이 됩니다.

BETWEEN 조건은 해당 범위의 사이의 값의 항목들을 출력합니다. 이와 반대로 해당 범위의 여집합을 선택하여 출력할 수도 있습니다. NOT BETWEEN 은 범위가 아닌 밖을 지정하여 출력합니다.

**select \* from orders where Id not between 3 and 8;**

Id의 범위가 3보다 크고 8보다 작은 영역이 아닌 그 외의 집합을 찾아 출력합니다.

|  |
| --- |
| mysql> **select \* from orders WHERE Id not between 3 and 8;**  +----+-------+  | Id | code |  +----+-------+  | 1 | O\_001 |  | 2 | O\_002 |  | 9 | O\_009 |  | 10 | O\_010 |  +----+-------+  4 rows in set (0.00 sec) |

해당 범위만 제외되고 전체가 출력됩니다.

PHP 코드를 통하여 BETWEEN 조건을 실험해 봅니다.

**예제파일) where-04.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "select \* from orders WHERE **Id between 3 and 8**";  if ($rowss = $db->selectRowss($queryString)) {  echo "tables fields = ". count($rowss) . "<br>";    for ($i=0;$i<count($rowss);$i++) {  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

**출력화면)**

mysql connected!

쿼리성공] select \* from orders WHERE Id between 3 and 8

tables fields = 6

0=stdClass Object ( [Id] => 3 [code] => O\_003 [email] => eric@jinyphp.com )

1=stdClass Object ( [Id] => 4 [code] => O\_004 [email] => )

2=stdClass Object ( [Id] => 5 [code] => O\_005 [email] => )

3=stdClass Object ( [Id] => 6 [code] => O\_006 [email] => )

4=stdClass Object ( [Id] => 7 [code] => O\_007 [email] => )

5=stdClass Object ( [Id] => 8 [code] => O\_008 [email] => )

BETWEEN의 연산범위로 텍스트 값을 사용할 수도 있습니다.

SELECT \* FROM members

WHERE username **BETWEEN** '장승빈' **AND** '이호진'

SELECT \* FROM members

WHERE username **NOT BETWEEN** '장승빈' **AND** '이호진'

BETWEEN의 연산범위로 날짜 값을 사용할 수도 있습니다.

SELECT \* FROM Orders WHERE regDate **BETWEEN** #07/04/1996# **AND** #07/09/1996#;

**11.2.5 IN**

====================

방향과 범위는 해당 조건에 맞는 그룹의 모두를 의미합니다. 그룹의 모두가 아니라 특정한 조건을 가지는 몇몇 다수를 선택하고자 할 때는 사용을 할 수 없습니다. 또는 = 기호로 모두 나열하는 것은 불편할 것입니다.

IN 연산자는 다수의 선택 값을 지정하여 결과를 출력할 수 있습니다.

**쿼리문법)**

|  |
| --- |
| ~~~ WHERE 컬럼명 IN (값1, 값2, ….); |

IN 연산자는 컬럼명=값1 or 컬럼명=값2 or 컬럼명=값3 or …. 처럼 or 연산자로 묶여져 있는 것과 같습니다.

**select \* from orders where Id in (1,3,9);**

Id 값이 1과 3과 9인 경우만 검색하여 출력합니다.

|  |
| --- |
| mysql> select \* from orders WHERE Id in (1,3,9);  +----+-------+  | Id | code |  +----+-------+  | 1 | O\_001 |  | 3 | O\_003 |  | 9 | O\_009 |  +----+-------+  3 rows in set (0.00 sec) |

선택하고자 하는 값은 IN 연산자 소괄호 ()안에 직접 입력하면 됩니다. 값이 여러 개일 경우 콤마(,)를 통하여 구분을 합니다. 또한 ()안에 또다른 SELECT 조회문을 입력하여 결과를 매칭 할 수도 있습니다.

**쿼리문법)**

|  |
| --- |
| 쿼리~ WHERE 컬럼명 IN (SELECT 쿼리문); |

먼저 IN 안에 있는 SELELCT 조건이 먼저 수행이 됩니다. 해당 조건에 맞는 값들이 IN 연산자 안에 값이 선택값으로 들어갑니다. 두번째로 메인 SELECT 가 실행되면서 IN 연산자와 일치된 목록만 출력됩니다.

**SELECT \* FROM members**

**WHERE email IN ( SELECT email FROM orders where regdate >= ’2017-05-26’ );**

위의 쿼리를 실행하게 되면 오늘 주문내역이 있는 회원 정보만 출력합니다.

PHP 코드를 통하여 IN 연산자를 실험해 보겠습니다.

**예제파일) where-05.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "select \* from orders WHERE Id in (1,3,9)";  if ($rowss = $db->selectRowss($queryString)) {  echo "tables fields = ". count($rowss) . "<br>";    for($i=0;$i<count($rowss);$i++){  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

출력화면)

mysql connected!

쿼리성공] select \* from orders WHERE Id in (1,3,9)

tables fields = 3

0=stdClass Object ( [Id] => 1 [code] => O\_001 [email] => hojin@jinyphp.com )

1=stdClass Object ( [Id] => 3 [code] => O\_003 [email] => eric@jinyphp.com )

2=stdClass Object ( [Id] => 9 [code] => O\_009 [email] => )

**11.2.6 ANY 와 ALL 연산자**

====================

앞에서 여러 개의 데이터를 매칭하여 처리를 하는 IN 연산자를 학습하였습니다. IN 연산자는 입력된 조건 값에 대해서 같은(=) 값을 가지는 것들만 처리를 합니다. ANY 연산자는 IN 연산자와 유사하지만 >, >=, <, <= 와 같이 양도 같이 비교처리를 할 수 있습니다

**쿼리문법)**

|  |
| --- |
| 쿼리~ WHERE 컬럼명 >= ANY(SELECT 쿼리문); |

위의 쿼리문법의 예를 보면 ANY 안에 있는 SELECT 쿼리문의 결과보다 크거나 같은 데이터 조건을 출력합니다.

**select \* from members where emoney > any (select prices from products);**

예제는 현재 판매하고 있는 상품의 가격을 ANY의 인자로 전달합니다. 그리고 회원이 적립금과 비교를 합니다. ANY 안에 있는 값중에 하나라도 만족하면 됩니다. 즉 제품의 모든 가격과 회원수의 적립금을 N:N 으로 검사를 하게 됩니다. 즉, 위의 쿼리 결과는 가장 금액이 작은 상품, 최소한 상품 1개 이상을 살수 있는 적립금을 가진 회원을 출력합니다.

반면에 ALL 쿼리는 서브 쿼리의 조건 모두가 일치하는지 확인을 합니다.

**쿼리문법)**

|  |
| --- |
| 쿼리~ WHERE 컬럼명 >= ALL(SELECT 쿼리문); |

ALL안에 있는 SELECT의 모든 조건을 만족하는 결과를 처리합니다.

**select \* from members where emoney > all (select prices from products)**

앞전에 ANY는 최저 가격의 상품 보다 이상의 금액을 가지는 회원 목록이었다면, ALL은 모든 상품의 가격을 다 만족을 해야 합니다. 즉, 최고 가격의 상품 이상의 회원 적립금을 가지는 회원을 출력합니다.

위의 쿼리를 달리 작성을 한다면

**select \* from members where emoney > (select max(prices) from products)**

와 같을 수 있습니다. ANY와 ALL 연산자는 WHERE절과 HAVING절 모두 사용이가능합니다.

**11.3 패턴검색**

====================

등호, 부등호와 같은 연산자는 크기와 양을 비교합니다. 크기를 비교하는 것은 숫자나 날짜등의 값일 경우 쉽게 적용할 수 있습니다. 하지만 문자 데이터의 경우에는 다른 방법의 검색을 적용하여야 합니다.

물론 등호(=)를 통하여 동일한 문자를 검색할 수도 있습니다. 하지만 등호는 정확이 일치하는 데이터만 검색이 되기 때문에 제한적입니다. SQL에서는 문자열 안의 일부 내용을 포함하고 있는지 검색할 수 있는 패턴 검색을 지원합니다. 패턴 검색은 다른 말로 퍼지 검색 이라고도 합니다. SQL에서 대표적인 퍼지 검색 키워드는 LIKE 입니다.

**11.3.1 패턴이란**

====================

SQL은 LIKE 연산자를 지원 합니다. LIKE는 영여 표현으로 “~와 같다, 비슷하다”의 의미로 정확하지는 않지만 유사하다는 것입니다. 즉, LIKE 패턴은 문자열 안에 지정한 문자의 내용이 포함하고 있는 지를 검사하는 것입니다.

**11.3.2 와일드기호**

====================

LIKE 연산자는 패턴의 유사성을 지정하기 위해서 와일드카드와 같은 기호를 같이 사용합니다. LIKE 에서 사용하는 대표적인 와일드카드 기호는 2가지가 있습니다.

* % : 다수의 문자를 포함하는 문자열을 가리킵니다.
* \_ : 하나의 문자를 가리킵니다.

이 2개의 와일드카드 기호는 LIKE 연산자와 같이 사용을 합니다. LIKE 연산 시 패턴 앞 또는 뒤에 같이 사용하여 패턴규약을 정할 수 있습니다.

**11.3.3 페센트 %**

====================

페센트 기호(%)는 여러개의 문자열을 매칭을 처리하는 와일드 기호 입니다. 다음과 같이 쿼리 명링이 있습니다.

**SELECT \* FROM members WHERE Country LIKE 'k%';**

위의 쿼리는 회원 목록을 출력합니다. 출력 조건은 Country LIKE ‘k%’ 입니다. 회원의 국가 컬럼에 퍼지 패턴 검색을 적용하였습니다. k%는 첫글자는 k로 시작해야 되어야 한다고 강제 지정을 하였습니다. %는 이후에 글자가 몇개가 와도 상관이 없다는 것입니다. 즉, k로 시작되는 문자열을 검색하라는 의미 입니다.

% 기호는 문자 앞/뒤에 양쪽으로 적을 수도 있습니다. 양쪽에 % 기호가 있다는 것은 중간의 글자가 포함되는 내용을 출력하라는 의미입니다.

**SELECT \* FROM members WHERE City LIKE '%es%';**

위의 예는 도시이름 중에서 es 가 들어있는 모든 회원정보를 출력합니다.

**11.3.4 밑줄 \_**

====================

밑줄(\_) 기호는 한글자를 의미합니다. %는 문자열을 의미하지만 \_은 단일 한 개의 문자를 의미합니다. 만일 \_기호를 두번 사용할때는 두글자를 의미합니다.

**SELECT \* FROM members WHERE City LIKE '\_\_san';**

위의 예는 앞에 두글자는 어떤 글자가 와도 됩니다. 하지만 이후에는 “san” 문자열로 끝아야 합니다. 즉, busan, ulsan 등의 도시들이 출력 될 것입니다.

밑줄(\_)와일드 기호는 문자열 중간에도 사용을 할 수 있습니다.

**SELECT \* FROM members WHERE City LIKE 's\_o\_l';**

**11.3.5 지정문자**

====================

한글자를 의미하는 밑줄(\_) 와일드 기호는 A~Z 또는 어떤 문자가 와도 허용이 됩니다. 전체 글자가 아니라 지정한 문자만의 패턴을 지정할 수도 있습니다. 대괄호 [ ] 안에 해당 글자만 적어 주시면 됩니다.

**SELECT \* FROM members WHERE City LIKE '[abcd]%';**

위의 예는 대괄호로 [abcd]로 정의되어 있습니다. 이는 어떤 문자가 와도 되는데 반드시 a, b, c, d 글자로 시작되어야 한다는 것입니다.

반대로 지정 글자 이외의 단어들도 선언할 수 있습니다. 예외문자는 [안에 부정 !를 삽입하여 문자를 적어줍니다.

**SELECT \* FROM members WHERE City LIKE '[!efg]%';**

위의 예제는 e, f, g 글자가 제외된 글자로 시작된 도시들만 선택됩니다. 또는 NOT LIKE 명령을 사용할 수도 있습니다.

**SELECT \* FROM members WHERE City NOT LIKE '[bsp]%';**

**11.3.6 %, \_ 조합**

====================

퍼지 검색시 사용되는 퍼센트(%) 와 밑줄(\_) 기호는 서로 조합하여 사용이 가능합니다.

예를 들어,

LIKE 'a%' , LIKE '%a' , LIKE '%or%' , LIKE '\_r%' , LIKE 'a\_%\_%' , LIKE 'a%o'

와 같이 썩어서 사용이 가능합니다.

**11.4 NULL 검색**

====================

NULL은 프로그램, 데이터베이스 에서도 공백과는 또다른 의미 입니다. NULL은 아무것도 없다 입니다. 테이블에 데이터를 입력할 때 값을 지정하기 않거나 없을 경우에는 해당 컬럼을 NULL로 자동 설정이 됩니다.

물론, 제약사항 속성을 통하여 NULL이 지정이 되지 않게 할 수도 있습니다. 또는, 초기값을 지정하여 설정을 할 수도 있습니다.

**11.4.1 NULL 확인**

====================

NULL로 설정된 컬럼을 조건을 통하여 검색을 할 수 있습니다.

**쿼리문법)**

|  |
| --- |
| 쿼리~ WHERE 컬럼명 IS NULL; |

컬럼의 상태가 NULL 인지를 확인하기 위해서는 IS NULL 연산을 통하여 확인 할 수 있습니다.

**select \* from members where address IS NULL;**

위의 쿼리 예제는 주소가 NULL로 설정된 데이터만 검색하여 출력을 합니다.

|  |
| --- |
| mysql> **select \* from members where address IS NULL;**  +----+----------+-----------+---------+------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+---------+------+---------+  | 2 | jiny | NULL | NULL | NULL | korea |  +----+----------+-----------+---------+------+---------+  1 row in set (0.00 sec) |

결과를 확인하면 주소 컬럼이 NULL로 설정된 것을 확인할 수 있습니다.

**11.4.2 NOT NULL 확인**

====================

반대로 NULL이 아닌 값을 검색은 IS NOT NULL 로 사용을 합니다.

**쿼리문법)**

|  |
| --- |
| ~~ WHERE 컬럼명 IS NOT NULL; |

**select \* from members where address IS NOT NULL;**

위의 쿼리 예제는 주소부분이 NULL이 아닌 회원 목록만을 출력합니다.

|  |
| --- |
| mysql> select \* from members where address IS NOT NULL;  +----+----------+-----------+-------------+-------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+-------------+-------+---------+  | 1 | hojin | lee | shinchon | seoul | korea |  | 3 | james | kim | youngdun-po | seoul | korea |  | 4 | eric | jung | seocho-gu | seoul | korea |  +----+----------+-----------+-------------+-------+---------+  3 rows in set (0.00 sec) |

주소를 가지고 있는 회원만 출력을 하였습니다.

**11.4.3 PHP 실습**

====================

PHP 코드를 통하여 컬럼의 NULL을 확인해 봅니다.

**예제파일) sql-.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "select \* from members WHERE address IS NOT NULL";  if ($rowss = $db->selectRowss($queryString)) {  echo "tables fields = ". count($rowss) . "<br>";    for ($i=0;$i<count($rowss);$i++) {  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

출력화면)

mysql connected!

쿼리성공] select \* from members WHERE address IS NOT NULL

tables fields = 2

0=stdClass Object ( [Id] => 1 [LastName] => hojin [FirstName] => lee [Address] => shinchon [City] => seoul [Country] => Korea [manager] => infohojin [email] => hojin@jinyphp.com )

1=stdClass Object ( [Id] => 3 [LastName] => james [FirstName] => kim [Address] => youngdun-po [City] => seoul [Country] => Korea [manager] => info [email] => james@jinyphp.com )

**11.5 조건결합**

====================

WHERE 조건은 논리 연산자를 같이 사용할 수 있습니다. 논리 연산자를 이용한다는 것은 여러 개의 조건을 논리적으로 결합하여 처리를 할 수 있다는 것입니다.

논리 연산자 결합은 좌측에서 우측으로 실행이 됩니다. 또한 연산자 우선 순위를 적용 받기 때문에 필요에 따라서는 우선 처리를 위한 괄호() 를 사용해야 되는 경우도 발생이 됩니다.

**11.5.1 AND**

====================

AND 논리 연산자는 두개의 조건을 결합할 수 있습니다. AND 연산자로 연결된 조건은 모두 참 인경우만 출력되는 연산입니다.

**쿼리~ where Id>=3 and City = 'seoul';**

두개의 연산자가 결합이 되었습니다. Id값은 3보다 큰 상태와 City가 seoul 이어야 합니다. 동시 조건을 만족을 해야 합니다.

|  |
| --- |
| mysql> **select \* from members where Id>=3 and City = 'seoul';**  +----+----------+-----------+-------------+-------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+-------------+-------+---------+  | 3 | james | kim | youngdun-po | seoul | korea |  | 4 | eric | jung | seocho-gu | seoul | korea |  +----+----------+-----------+-------------+-------+---------+  2 rows in set (0.00 sec) |

위의 예제를 보면 두개의 조건이 참이 결과를 반환 합니다.

**11.5.2 OR**

====================

OR 논리 연산자는 두개의 조건을 모두 선택할 수 있습니다. OR 연산자는 조건 중에서 하나만 이라도 참 이면 모두 참인 경우 입니다.

**쿼리~ where Id=3 or Id=4;**

Id가 3 이거나 id가 4이면 됩니다. 두 개중에 하나라도 만족을 하면 모두 출력이 됩니다.

|  |
| --- |
| mysql> **select \* from members where Id=3 or Id=4;**  +----+----------+-----------+-------------+-------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+-------------+-------+---------+  | 3 | james | kim | youngdun-po | seoul | korea |  | 4 | eric | jung | seocho-gu | seoul | korea |  +----+----------+-----------+-------------+-------+---------+  2 rows in set (0.00 sec) |

두개의 조건이 모두 만족하는 결과 2개를 출력합니다.

**11.5.3 AND, OR**

====================

논리연산자 AND 와 OR 는 서로 복합적으로 결합하여 사용할 수 있습니다. 두개의 연산자를 복합하여 사용을 할 때에는 연산자 우선순위에 따라서 조건필터가 다르게 해석될 수도 있습니다.

AND 와 OR 을 같이 사용할 때는 소괄호 ( )를 사용하여 연산자 우선순위를 확인해야 합니다.

**쿼리~ where (Id=3 or Id=4) and FirstName = ‘kim’;**

우선순위에 따라서 괄호 부분이 먼저 해석이 됩니다. Id 값이 3이거나 4 가 되면 괄호 부분은 참이 됩니다. 그리고 FirstName 부분이 참인지를 검사합니다.

|  |
| --- |
| mysql> select \* from members where (Id=3 or Id=4) and FirstName = 'kim';  +----+----------+-----------+-------------+-------+---------+  | Id | LastName | FirstName | Address | City | Country |  +----+----------+-----------+-------------+-------+---------+  | 3 | james | kim | youngdun-po | seoul | korea |  +----+----------+-----------+-------------+-------+---------+  1 row in set (0.00 sec) |

다수의 조건이 만족한 결과를 출력합니다.

**11.6 조건 분기**

====================

조건분기는 if else 처럼 SQL에서도 컬럼의 데이터 값을 비교하여 처리를 할 수 있습니다. 컬럼필드 선택부분에 CASE WHEN ~ 문장을 넣어 주시면 됩니다.

**쿼리문법)**

|  |
| --- |
| CASE  WHEN 조건1 THEN 값1  WHEN 조건2 THEN 값2  WHEN 조건3 THEN 값3  ELSE 값4  END AS 별칭 |

**11.6.1 쿼리실습**

====================

다음은 Id 값에 따라서 서수 번호를 NO 라는 이름으로 처리하는 예제 쿼리 입니다.

**select FirstNAme,LastName,**

**case**

**when Id>= 5 then '5th'**

**when Id>= 4 then '4th'**

**when Id>= 3 then '3th'**

**when Id>= 2 then '2th'**

**else '1th'**

**end as No**

**from members;**

출력은 FirstNAme,LastName,No 형태로 3개의 컬럼이 출력이 됩니다.

|  |
| --- |
| mysql> select FirstNAme,LastName,  -> case  -> when Id>= 5 then '5th'  -> when Id>= 4 then '4th'  -> when Id>= 3 then '3th'  -> when Id>= 2 then '2th'  -> else '1th'  -> end as No  -> from members;  +-----------+----------+-----+  | FirstNAme | LastName | No |  +-----------+----------+-----+  | lee | hojin | 1th |  | NULL | jiny | 2th |  | kim | james | 3th |  | jung | eric | 4th |  +-----------+----------+-----+  4 rows in set (0.00 sec) |

No 컬럼은 별칭을 선언되어 처리를 합니다. 별칭 기능은 나중에 다시 학습을 합니다.

**11.6.2 PHP 실습**

====================

PHP 코드를 통하여 조건분기 쿼리를 실행해 봅니다.

**예제파일) where-09.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  $queryString = "select FirstNAme,LastName,  case  when Id>= 5 then '5th'  when Id>= 4 then '4th'  when Id>= 3 then '3th'  when Id>= 2 then '2th'  else '1th'  end **as No**  from members;";    if($rowss = $db->selectRowss($queryString)){  echo "tables fields = ". count($rowss) . "<br>";    for($i=0;$i<count($rowss);$i++){  echo $i."=";  print\_r($rowss[$i]);  echo "<br>";  }  }  ?> |

출력화면)

mysql connected!

쿼리성공] select FirstNAme,LastName, case when Id>= 5 then '5th' when Id>= 4 then '4th' when Id>= 3 then '3th' when Id>= 2 then '2th' else '1th' end as No from members;

tables fields = 6

0=stdClass Object ( [FirstNAme] => lee [LastName] => hojin [No] => 1th )

1=stdClass Object ( [FirstNAme] => hojin [LastName] => lee [No] => 2th )

2=stdClass Object ( [FirstNAme] => kim [LastName] => james [No] => 3th )

3=stdClass Object ( [FirstNAme] => jiny [LastName] => 1234 [No] => 5th )

4=stdClass Object ( [FirstNAme] => jiny [LastName] => 1234 [No] => 5th )

5=stdClass Object ( [FirstNAme] => jiny [LastName] => 123400 [No] => 5th )

**05.7 서브쿼리**

====================

앞에 IN, ANY, ALL 등의 연사자를 설명하면서 서브 쿼리를 미리 사용해 보았습니다. 서브 쿼리란

SELECT 로 조회한 데이터를 가지고 다시 메인 SELECT 의 연산의 기초 값이 되는 것입니다. 즉 SELECT를 연결하여 이중조회를 하는 것을 말합니다.

서브쿼리는 MYSQL 4.1 이상 버전에서만 지원합니다.

**쿼리문법)**

|  |
| --- |
| SELECT \* from 테이블 where 컬럼 연산자 (서브쿼리); |

**05.7.1 쿼리 실습**

====================

서브 쿼리를 작성할 때는 괄호를 이용하여 감싸 줍니다.

**select \* from members where manager IN (select manager from members where City='seoul');**

select manager from members where City='seoul' 쿼리가 괄호 안에 있기 때문에 먼저 실행이 됩니다. 결과 값은 IN 연산자의 대입값으로 처리가 되고, 이에 따라서 메인 쿼리 select \* from members where manager IN () 가 실행이 됩니다.

|  |
| --- |
| mysql> **select \* from members where manager IN (select manager from members where City='seoul');**  +----+----------+-----------+-------------+-------+---------+-----------+-------------------+  | Id | LastName | FirstName | Address | City | Country | manager | email |  +----+----------+-----------+-------------+-------+---------+-----------+-------------------+  | 1 | hojin | lee | shinchon | seoul | korea | infohojin | hojin@jinyphp.com |  | 2 | jiny | NULL | NULL | NULL | korea | infohojin | jiny@jinyphp.com |  | 3 | james | kim | youngdun-po | seoul | korea | info | james@jinyphp.com |  | 4 | eric | jung | seocho-gu | busan | korea | info | eric@jinyphp.com |  +----+----------+-----------+-------------+-------+---------+-----------+-------------------+  4 rows in set (0.00 sec) |

값을 포함하고 있는지 검사할때는 IN 연산자를 사용합니다. 크기를 비교할 때는 >, >=, <, <= 를 사용할 수 있습니다.

**05.8 EXISTS**

====================

서브쿼리를 이용하여 이중 검색을 할 때 첫번째 쿼리의 조건에 대한 결과 값이 없을 수도 있습니다. 존재하지 않는 결과 값을 이용하여 메인 쿼리를 실행하게 되면 오류가 발생됩니다.

이런경우 첫번에 실행되는 서브쿼리의 결과 값이 있는지를 확인할 수 있는 키워드를 제공합니다. EXISTS는 서브쿼리의 존재 여부를 판단하여 메인 쿼리를 실행하게 됩니다.

**쿼리문법)**

|  |
| --- |
| SELECT 컬럼명, 컬럼명... FROM 테이블명 WHERE EXISTS  (SELECT column\_name FROM table\_name WHERE condition); |

EXISTS 는 해당 쿼리의 결과값을 TRUE / FALSE 로 반환합니다. EXISTS 의 반대로 NOT EXISTS 를 사용할 수 있습니다. NOT EXISTS 는 서브쿼리 결과가 없다는 의미 입니다.

**05.8.1 쿼리실습**

====================

예로 들면 특정한 고객의 주문이 있는 경우에, 회원정보를 출력하는 쿼리를 작성을 할 때 매우 유용합니다.

다음 SQL 문은 TRUE를 반환하고 제품 가격이 20 미만인 공급 업체를 나열합니다.

**SELECT SupplierName FROM Suppliers**

**WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);**

위의 SQL 문은 TRUE를 반환하고 제품 가격이 20 미만인 공급 업체를 나열합니다.

**SELECT SupplierName FROM Suppliers**

**WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price = 22);**

EXISTS 서브쿼리의 결과 값이 존재를 하면 본 select 쿼리를 실행을 합니다.