1. MySQL Workbench로 로그인
2. 이 때부터 기본으로 prod 데이터베이스를 기본으로 사용하게 됨 (MySQL workbench에서의 설정에 따라 다름)
3. 이제부터 다음 SQL들을 차례로 실행해보면서 강의에서 설명했던 내용과 비교해보기 바람

[SELECT](#_12ptkdg00nmw)

[GROUP BY](#_6tg9mlteexgo)

[INSERT/UPDATE/DELETE](#_kbzoi7cb8lfz)

[JOIN](#_sxbqttz5pidu)

[TRANSACTION](#_xei85pd5n4jw)

[VIEW](#_e44rtc8vu04j)

[Stored Procedure와 Trigger](#_qxvfdknaro23)

[Explain SQL과 Index](#_b8d1m4lwyyp7)

## 

| SHOW databases;  USE prod;  SHOW tables; |
| --- |

## SELECT

| SELECT \*  FROM prod.session;  SELECT id, user\_id, channel\_id  FROM prod.session;  SELECT \*  FROM prod.session  LIMIT 10;  -- 유일한 채널 ID를 알고 싶은 경우  SELECT DISTINCT channel\_id  FROM prod.session;  -- 채널 ID별 카운트를 하려면 GROUP BY/COUNT 함수!!  SELECT channel\_id, COUNT(1)  FROM prod.session  GROUP BY 1;  -- 테이블의 모든 레코드 수 카운트. COUNT(\*). 하나의 레코드  SELECT COUNT(1)  FROM prod.session;  -- channel 테이블의 모든 레코드들을 표시  SELECT \*  FROM prod.channel;  -- channel이 Facebook경우만 레코드수 카운트  SELECT COUNT(1)  FROM prod.session  WHERE channel\_id = 5; -- WHERE channel\_id in (5)  -- CASE WHEN 배우기  SELECT channel\_id,  CASE  WHEN channel\_id in (1, 5, 6) THEN 'Social-Media'  WHEN channel\_id in (2, 4) THEN 'Search-Engine'  ELSE 'Something-Else'  END channel\_type  FROM prod.session;  -- COUNT 제대로 이해하기  SELECT \* FROM prod.count\_test;  SELECT COUNT(1) FROM prod.count\_test;  SELECT COUNT(0) FROM prod.count\_test;  SELECT COUNT(NULL) FROM prod.count\_test;  SELECT COUNT(value) FROM prod.count\_test;  SELECT COUNT(DISTINCT value) FROM prod.count\_test;  -- WHERE 조건  SELECT COUNT(1)  FROM prod.session  WHERE channel\_id IN (4, 5);  SELECT COUNT(1)  FROM prod.channel  WHERE channel LIKE '%G%';  SELECT DISTINCT channel  FROM prod.channel  WHERE channel LIKE '%o%';  SELECT DISTINCT channel  FROM prod.channel  WHERE channel NOT LIKE '%o%';  -- String 함수들 써보기  SELECT  LENGTH(channel),  UPPER(channel),  LOWER(channel),  LEFT(channel, 4),  RPAD(channel, 15, '-')  FROM prod.channel;  -- ORDER BY and NULL  SELECT value  FROM prod.count\_test  ORDER BY value DESC;  SELECT value  FROM prod.count\_test  ORDER BY 1 DESC;  SELECT value  FROM prod.count\_test  ORDER BY value ASC;  -- DATE 관련 함수들 써보기  SELECT  created, CONVERT\_TZ(created, 'GMT', 'Asia/Seoul') seoul\_time,  YEAR(created) y, QUARTER(created) q, MONTH(created) m, MONTHNAME(created) mnn,  DATE(created) d, HOUR(created) h, MINUTE(created) m, SECOND(created) s  FROM session  LIMIT 10;  SELECT created,  DATEDIFF(now(), created) gap\_in\_days,  DATE\_ADD(created, INTERVAL 10 DAY) ten\_days\_after\_created  FROM session  LIMIT 10;  SELECT STR\_TO\_DATE('01,5,2013','%d,%m,%Y');  -- Type casting  SELECT cast('100.0' as float), convert('100.0', float); |
| --- |

## GROUP BY

| -- 월별 세션수를 계산하는 SQL  SELECT  LEFT(created, 7) AS mon,  COUNT(1) AS session\_count  FROM prod.session  GROUP BY 1 -- GROUP BY mon, GROUP BY LEFT(created, 7)  ORDER BY 1;  -- 가장 많이 사용된 채널은 무엇인가?  SELECT  channel\_id,  COUNT(1) AS session\_count,  COUNT(DISTINCT user\_id) AS user\_count  FROM prod.session  GROUP BY 1 -- GROUP BY channel\_id  ORDER BY 2 DESC; -- ORDER BY session\_count DESC  -- 가장 많은 세션을 만들어낸 사용자 ID는 무엇인가?  SELECT  user\_id,  COUNT(1) AS count  FROM prod.session  GROUP BY 1 -- GROUP BY user\_id  ORDER BY 2 DESC -- ORDER BY count DESC  LIMIT 1;  -- 월별 유니크한 사용자 수  SELECT  LEFT(created, 7) AS mon,  COUNT(DISTINCT user\_id) AS user\_count  FROM prod.session  GROUP BY 1 -- GROUP BY mon, GROUP BY LEFT(created, 7)  ORDER BY 1;  -- 월별 채널별 유니크한 사용자 수  SELECT s.id, s.user\_id, s.created, s.channel\_id, c.channel  FROM session s  JOIN channel c ON c.id = s.channel\_id;  --  SELECT  LEFT(s.created, 7) AS mon,  c.channel,  COUNT(DISTINCT user\_id) AS mau  FROM session s  JOIN channel c ON c.id = s.channel\_id  GROUP BY 1, 2  ORDER BY 1 DESC, 2; |
| --- |

## INSERT/UPDATE/DELETE

| CREATE TABLE prod.vital (  user\_id int not null,  vital\_id int primary key,  date timestamp,  weight int not null  );  CREATE TABLE prod.alert (  alert\_id int primary key,  vital\_id int,  alert\_type varchar(32),  date timestamp,  user\_id int  );  -- INSERT  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(100, 1, '2020-01-01', 75);  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(100, 3, '2020-01-02', 78);  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(101, 2, '2020-01-01', 90);  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(101, 4, '2020-01-02', 95);  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(999, 5, '2020-01-02', -1);  INSERT INTO prod.vital(user\_id, vital\_id, date, weight) VALUES(999, 5, '2020-01-02', 10);  INSERT INTO prod.alert VALUES(1, 4, 'WeightIncrease', '2020-01-02', 101);  INSERT INTO prod.alert VALUES(2, NULL, 'MissingVital', '2020-01-04', 100);  INSERT INTO prod.alert VALUES(3, NULL, 'MissingVital', '2020-01-04', 101);  SELECT \* FROM prod.vital;  SELECT \* FROM prod.alert;  SELECT \* FROM prod.alert WHERE vital\_id = NULL;  SELECT \* FROM prod.alert WHERE vital\_id IS NULL;  SELECT \* FROM prod.alert WHERE vital\_id != NULL;  SELECT \* FROM prod.alert WHERE vital\_id IS NOT NULL;  -- DELETE  SELECT \* FROM prod.vital;  DELETE FROM prod.vital WHERE weight <= 0;  DELETE FROM prod.vital WHERE vital\_id = 5;  SELECT \* FROM prod.vital;  DELETE FROM prod.vital;  SELECT \* FROM prod.vital;  -- UPDATE  SELECT \* FROM prod.vital WHERE vital\_id = 4;  UPDATE prod.vital SET weight = 92 WHERE vital\_id = 4;  SELECT \* FROM prod.vital WHERE vital\_id = 4; |
| --- |

## JOIN

| -- INNER JOIN  SELECT \* FROM prod.vital v  JOIN prod.alert a ON v.vital\_id = a.vital\_id;  -- LEFT JOIN  SELECT \* FROM prod.vital v  LEFT JOIN prod.alert a ON v.vital\_id = a.vital\_id;  -- RIGHT JOIN  SELECT \* FROM prod.vital v  RIGHT JOIN prod.alert a ON v.vital\_id = a.vital\_id;  -- OUTER JOIN  SELECT \* FROM prod.vital v  LEFT JOIN prod.alert a ON v.vital\_id = a.vital\_id  UNION  SELECT \* FROM prod.vital v  RIGHT JOIN prod.alert a ON v.vital\_id = a.vital\_id;  -- CROSS JOIN  SELECT \* FROM prod.vital v CROSS JOIN prod.alert a;  -- SELF JOIN  SELECT \* FROM prod.vital v1  JOIN prod.vital v2 ON v1.vital\_id = v2.vital\_id; |
| --- |

## TRANSACTION

| DROP TABLE IF EXISTS test.**keeyong\_**name\_gender;  CREATE TABLE test.**keeyong\_**name\_gender (  name varchar(16) NOT NULL,  gender enum('Male','Female') default NULL  );  INSERT INTO test.**keeyong\_**name\_gender VALUES('Keeyong', 'Male');  INSERT INTO test.**keeyong\_**name\_gender VALUES('Jane', 'Female');  INSERT INTO test.**keeyong\_**name\_gender VALUES('Unknown');  INSERT INTO test.**keeyong\_**name\_gender VALUES('Keeyong2', 'Male2');  --  -- autocommit = True  --  SHOW VARIABLES LIKE 'AUTOCOMMIT';  -- SET autocommit=1;  SELECT \* FROM test.**keeyong\_**name\_gender;  BEGIN;  DELETE FROM test.**keeyong\_**name\_gender;  INSERT INTO test.**keeyong\_**name\_gender VALUES ('Kevin', 'Male');  ROLLBACK;  SELECT \* FROM test.**keeyong\_**name\_gender;  --  -- autocommit = False  --  SET autocommit=0;  SHOW VARIABLES LIKE 'AUTOCOMMIT';  SELECT \* FROM test.keeyong\_name\_gender;  -- BEGIN이 없음  DELETE FROM test.keeyong\_name\_gender;  INSERT INTO test.keeyong\_name\_gender VALUES ('Kevin', 'Male');  ROLLBACK;  SELECT \* FROM test.keeyong\_name\_gender; |
| --- |

## VIEW

| SELECT s.id, s.user\_id, s.created, s.channel\_id, c.channel  FROM session s  JOIN channel c ON c.id = s.channel\_id;  CREATE OR REPLACE VIEW test.keeyong\_session\_details AS  SELECT s.id, s.user\_id, s.created, s.channel\_id, c.channel  FROM session s  JOIN channel c ON c.id = s.channel\_id;  SELECT \* FROM test.keeyong\_session\_details; |
| --- |

## Stored Procedure, Trigger

| -- Stored Procedure  ---- # 1  DELIMITER //  CREATE PROCEDURE return\_session\_details()  BEGIN  SELECT \*  FROM session\_details;  END //  DELIMITER ;  CALL return\_session\_details();  ----- # 2  DROP procedure IF EXISTS return\_session\_details;  DELIMITER //  CREATE PROCEDURE return\_session\_details(IN channelName varchar(64))  BEGIN  SELECT \*  FROM session\_details  WHERE channel = channelName;  END //  DELIMITER ;  CALL return\_session\_details('Facebook');  ----- # 3  DROP procedure IF EXISTS return\_session\_count;  DELIMITER //  CREATE PROCEDURE return\_session\_count(IN channelName varchar(64), INOUT totalRecord int)  BEGIN  SELECT COUNT(1) INTO totalRecord FROM session\_details  WHERE channel = channelName;  END //  DELIMITER ;  SET @facebook\_count = 0  CALL return\_session\_count('Facebook', @facebook\_count);  SELECT @facebook\_count;  -- Stored Function  DELIMITER $$  CREATE FUNCTION test.Channel\_Type(  channel varchar(32)  )  RETURNS VARCHAR(20)  DETERMINISTIC  BEGIN  DECLARE channel\_type VARCHAR(20);  IF channel in ('Facebook', 'Instagram', 'Tiktok') THEN  SET channel\_type = 'Social Network';  ELSEIF channel in ('Google', 'Naver') THEN  SET channel\_type = 'Search Engine';  ELSE  SET channel\_type = channel;  END IF;  -- return the customer level  RETURN (channel\_type);  END$$  SELECT channel, test.Channel\_Type(channel)  FROM prod.channel;  -- Trigger  DESCRIBE test.keeyong\_name\_gender;  DROP TABLE IF EXISTS test.keeyong\_name\_gender\_audit;  CREATE TABLE test.keeyong\_name\_gender\_audit (  name varchar(16),  gender enum('Male', 'Female'),  modified timestamp  );  SELECT \* FROM test.keeyong\_name\_gender;  DROP TRIGGER IF EXISTS test.before\_update\_keeyong\_name\_gender;  CREATE TRIGGER test.before\_update\_keeyong\_name\_gender  BEFORE UPDATE ON test.keeyong\_name\_gender  FOR EACH ROW  INSERT INTO test.keeyong\_name\_gender\_audit  SET name = OLD.name,  gender = OLD.gender,  modified = NOW();  UPDATE test.keeyong\_name\_gender  SET name = 'Keeyong'  WHERE name = 'Keeyong2';  SELECT \* FROM test.keeyong\_name\_gender\_audit;  USE test;  SHOW TRIGGERS; |
| --- |

## Explain SQL과 Index 최적화

| -- EXPLAIN  EXPLAIN SELECT  LEFT(s.created, 7) AS mon,  c.channel,  COUNT(DISTINCT user\_id) AS mau  FROM session s  JOIN channel c ON c.id = s.channel\_id  GROUP BY 1, 2  ORDER BY 1 DESC, 2;  -- Index. 이걸 실습을 하고 싶다면 test 밑에 테이블을 따로 만들어보세요  DROP TABLE IF EXISTS prod.session\_with\_index;  CREATE TABLE prod.session\_with\_index (  id int NOT NULL auto\_increment,  user\_id int not NULL,  created timestamp not NULL default CURRENT\_TIMESTAMP,  channel\_id int not NULL,  PRIMARY KEY(id),  FOREIGN KEY(channel\_id) references channel(id),  INDEX user\_id(user\_id)  );  -- INSERT INTO prod.session\_with\_index SELECT \* FROM prod.session;  SHOW INDEX FROM prod.session;  SHOW INDEX FROM prod.session\_with\_index;  SHOW INDEX FROM prod.channel;  SELECT user\_id, COUNT(1) FROM prod.session GROUP BY 1;  SELECT user\_id, COUNT(1) FROM prod.session\_with\_index GROUP BY 1; |
| --- |