■ SQL Reserved Words

Reserved words are SQL keywords and other symbols that have special meanings when they are processed by the Relational Engine. Reserved words are not recommended for use as database, table, column, variable or other object names. If a reserved word is used as an object name, it must be enclosed in double-quotes to notify the Relational Engine that the word is not being used as a keyword in the given context.

※ Any command that you write in SQL is always gonna end with ; (1:22:44)

※ PRIMARY KEY : The one column that's gonna uniquely identify each row. (1:23:23)

CREAT TABLE : typed them in all capital letters. This is convention when writing SQL queries. The reason that people write these in cap because, then, it's easy to distinguish SQL than any other text.

| Basic Data Types | | |
|---|---|---|
| 1. INT<br>integer : 정수 | whole Numbers | ' : single quote<br>= : equal<br>/ : forward slash<br>: : colon<br>$ : dollar sign<br>> : greater than sign<br>~ : tilde<br>_ : underscore<br>- : hyphen |
| 2. DECIMAL(M, N)<br>DECIMAL(10, 4) : you want to store decimal number with 10 total digits and 4 of those digits coming after deciamal place(?) | Decimal Numbers - Exact Value<br>decimal : 십진법의 | |
| 3. VARCHAR(N) | String of text of length N | |
| 4. BLOB | Binary Large Object, stores large data<br>binary : 2진법의 | |
| 5. DATE | YYYY-MM-DD | |
| 6. TIMESTAMP | YYYY-MM-DD HH:MM:SS - used for recording | |

## Define your database skema

```
CREATE TABLE student (
        student_id INT PRIMARY KEY,
        name VARCHAR(20),
        major VARCHAR(20).
);
```

```
CREATE TABLE student (
        student_id INT,
        name VARCHAR(20),
        major VARCHAR(20).
        PRIMARY KEY(student_id)
);
```

create table statement

Expaination

give student_id a data type INT.

tell mysql that this is ganna be primary key.

I'm ganna make name VARCHAR. Remember, with VARCHAR data type, we have to tell mysql how many characters we want it to be able to store. How many characters do we really allocate to storing someone's name.

I'm defining each of attributes, each of columns on the table and I'm putting a comma.

That is how we could create a table.

and now we want to have this table store our data base.

1:27:07

```
CREATE TABLE student (
        student_id INT PRIMARY KEY,
        name VARCHAR(20),
        major VARCHAR(20).
) ;
```

```
Describe student ;
```

this statement is basically describe everything about this(student) table

## Delete and Modify the table

(Delete를 하기 위한 명령어)

```
DROP TABLE student ;
```

execute the Decribe query after Running DROP TABLE student command → ER_NO_SUCH_TABLE :

(Modify를 하기 위한 명령어)

(Click CREATE TABLE query and click RUN button, we get the table back)

```
ALTER TABLE student ADD gpa DECIMAL(3, 2) ;
```

add extra column onto the table

```
ALTER TALBE student DROP COLUMN gpa ;
```

drop a specific column

## Inserting Data

```
INSERT INTO student VALUES();
```

anytime typing out strings in sql, we're gonna make this ''(single quotation marks) and in here we can type out the strings.

```
CREATE TABLE student (
        student_id INT PRIMARY KEY,
        name VARCHAR(20),
        major VARCHAR(20).
);
```

| student_id | name | major |
|---|---|---|
| 1 | Jack | Biology |
| 2 | Kate | Sociology |
| 3 | Claire | English |
| 4 | Jack | Biology |
| 5 | Mike | Comp. Sci |

```
INSERT INTO student VALUES(1, 'Jack', 'Biology') ;
```

```
SELECT * FROM student ;
```

Describe student; 와 같은 결과가 도출되는듯

추가적인 데이터를 입력하기 위해서는 별도의 INSERT 쿼리를 작성해야하는 것은 아님.
INSERT INTO student VALUES(1, 'Jack', 'Biology'); → INSERT INTO student VALUES(2, 'Kate', 'Sociology');
로 내용을 바꾼 뒤 [실행]
What you can do is basically use the same format in order to keep inserting students into the student table. But I wanna show you another thing we can do

Let's say we have situation where we had a student who didn't have a major. Maybe they just had no majors or we didn't know what their major was. In this situation like that we can modify the statement little bit.

```
INSERT INTO student(student_id, name) VALUES(3, 'Claire')
```

You can specify what piece of information you wanna insert into the table by specifying them to the attributes part for student() and add the information in VALUE() section.
And it's important notice that you can't insert duplicate entry. Record with primary key already inside the table.

※ INSERT INTO student(student_id, name) VALUES(3, 'Claire', 'English')라고 앞에는 major를 선언 안한 상태에서 VALUE항목에 추가하면 어떻게 되지? (확인요망)

## Constraints

Namely, we can actually set up our database table in order to make it easier for us to insert element or control the type of information/lows(?) that we can insert into database table.

```
CREATE TABLE student (
        student_id INT,
        name VARCHAR(20) NOT NULL,
        major VARCHAR(20) UNIQUE,
        PRIMARY KEY(student_id),
);
```

| student_id | name | major |
|---|---|---|
| 1 | Jack | Biology |
| 2 | Kate | Sociology |
| 3 | Claire | English |
| 4 | Jack | Biology |
| 5 | Mike | Comp. Sci |

```
INSERT INTO student VALUES(1, 'Jack', 'Biology')
INSERT INTO student VALUES(2, 'Kate', 'Sociology')
INSERT INTO student VALUES(3, NULL, 'English')
```

※ NULL대신 입력을 안하는 방식으로 VALUES(3, , 'English')면 어떨까? (확인요망)

you have an error in your SQL syntax;
check the manual that corresponds to your MySQL server version for the right syntax to .....

```
INSERT INTO student VALUES(4, 'Jack', 'Biology')
```

ER_DUP_ENTRY : Duplicate entry 'Biology' for key 'major'

```
INSERT INTO student VALUES(5, 'Mike', 'Computer Science') ;
```

primary key is an attribute or column on the table that is both not NULL and UNIQUE.

Let's say we want to set a default value. Let's say somebody didn't answer in a major, we want to give them a default major.

```
major VARCHAR(20) DEFAULT 'undecided',
```

```
CREATE TABLE student (
        student_id INT AUTO_INCREMENT,
        name VARCHAR(20),
        major VARCHAR(20),
        PRIMARY KEY(student_id)
);

INSERT INTO student(name, major) VALUES('Jack', 'Biology') ;
```

## Update & Delete

주로 사용되는 SQL 구문

- 검색 : SELECT 필드명 FROM 테이블명 WHERE 조건
- 갱신 : UPDATE 테이블명 SET 필드명=내용 WHERE 조건
- 삭제 : DELETE 필드명 FROM 테이블명 WHERE 조건
- 추가 : INSERT INTO 테이블명(필드명1, 필드명2) VALUES (값1, 값2)

```
UPDATE student
SET major = 'bio'
WHERE major = 'biology' ;
```

```
UPDATE student
SET major = 'Computer Sci'
WHERE major = 'Computer Science' ;
```

```
UPDATE student
SET major = 'Economics'
WHERE student_id = 20092118 ;
```

```
UPDATE student
SET major = 'Biochemistry'
WHERE major = 'Bio' or major = 'Chemistry' ;
```

```
UPDATE student
SET name = 'MONU', major = 'Physics'
WHERE student_id = 20202277 ;
```

```
Delete student
WHERE student_id = 5 ;
```

```
Delete student
WHERE name = 'Tom' AND major = 'undecided' ;
```

## Basic Queries (Getting information from database)

query is essentially a block of SQL which is designed to ask the database management system for a particular piece of information.

this is actually huge topic and it's the topic that we're gonna be talking about for the most of rest of the cource.

```
SELECT name, major
FROM student
WHERE major <> 'Chemistry' AND student_id <= 5 ;
```

```
SELECT student.name, student.major
FROM student
ORDER BY major, student_id DESC
LIMIT 10 ;
you can also limit the amount of result you're getting
```

```
SELECT *
FROM student
WHERE name IN ('Kate', 'Mike', 'Claire') AND student_id <10 ;
```

as the database skemas get more complex, the queries that you need to select specific information are also gonna get more complex. if we have multiple different tables(maybe certain table has foreign keys), getting information can get more complex. as we go forward in the course, we're gonna design more complex database skema.

콤보상자 만들 듯이 테이블 생성시 필드에 입력마스크 혹은 유효성검사를 추가할 수 있나?

## Company database into

basically foreign key is to store primary key of different table.

## Creadting company database

```
CREATE TABLE employee (
        emp_id INT PRIMARY KEY,
        first_name VARCHAR(20),
        last_name VARCHAR(20),
        birth_day DATE,
        sex VARCHAR(1),
        salary INT,
        super_id INT,
        branch_id INT,
) ;
```

super_id와 branch_id는 foreign key인데 '아직은' super, branch 테이블이 존재하지 않으므로 foreign key로 지정할 수 없다.

```
CREATE TABLE branch (
        branch_id INT PRIMARY KEY,
        branch_name VARCHAR(20),
        mgr_id INT,
        mgr_start_date DATE,
        FOREIGN KEY(mgr_id) REFERENCES employee(emp_id) ON DELETE SET NULL
) ;
```

- mgr_id를 FOREIGN KEY로 지정하는데 employee TABLE의 emp_id를 참조하는 것?
  각각 입력되는 경우 일치하지 않을 수도 있지 않나? 한 쪽에서만 입력하고 다른 테이블에서 불러오는건가?
- For now, just know that whenever we create a foreign key we are gonna put ON DELETE SET NULL
- Next thing we need to do is we need to set these super_id and branch_id of employ TABLE as foreign key.

```
ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL ;
```

```
ALTER TABLE employee
ADD FOREIGN KEY(super_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL ;
```

```
CREATE TABLE client (
        client_id INT PRIMARY KEY,
        client_name VARCHAR(20),
        branch_id INT,
        FOREIGN KEY(branch_id) REFERENCES branch(brach_id) ON DELETE SET NULL
) ;
```

```
CREATE TABLE work_with (
        emp_id INT,
        client_id INT,
        total_sales INT,
        PRIMARY KEY(emp_id, client_id),
        FOREIGN KEY(emp_id) REFERENCES employee(emp_id) ON DELETE CASCADE,
        FOREIGN KEY(client_id) REFERENCES client(client_id) ON DELETE CASCADE,
) ;
```

work_with TABLE is pretty unique because it has a composite PRIMARY KEY. PRIMARY KEY has the emp_id and client_id. Actually what unique is that each component of the PRIMARY KEY is a foreign key.

```
CREATE TABLE branch_supplier (
        branch_id INT,
        supplier_name VARCHAR(20),
        supply_type VARCHAR(20),
        PRIMARY KEY(branch_id, supplier_name),
        FORIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE.
) ;
```

now we got all of these table created. all the tables for our database skema.
what we're gonna do now is, we're gonna insert information into those tables.
when we're inserting information into these table, because we have all these foreign key relationships, we have to do it in a specific way.

I notice that employee TABLE and branch TABLE have a foreign key that point to each other. the employ TABLE has branch_id which points to the branch (TABLE) or points to a specific branch. Branch TABLE has a column, mgr_id, which points to a specific employee.
So when we are inserting these elements,

여기까지가 데이터베이스 스키마 구성인 듯. 아래부터는 값의 입력.

```
-- Coorporate                    emp_id  first_name  last_name     bith_date     sex  salary  super_id branch_id
INSERT INTO employee VALUES(100, 'DAVID', 'Wallace', '1967-11-17', 'M', 25000, NULL, NULL ) ;

INSERT INTO branch VALUES(1, 'Corporate', 100, '2006-02-09') ;

UPDATE employee
SET branch_id = 1
WHERE emp_id = 100 ;

INSERT INTO employee VALUES(101, 'Jan', 'Levinson', '1961-05-11', 'F', 110000, 100, 1) ;
```

Branch를 기준으로 입력의 순서를 분류한 이유가 뭐지?
we have to do it that way because we have the circular relationship with foreign key between employ and brach table. ???

```
--- Scranton                     emp_id  first_name  last_name     bith_date     sex  salary  super_id branch_id
INSERT INTO employee VALUES(102, 'Michael', 'Scott', '1964-03-15', 'M', 75000, 100, NULL)

INSERT INTO branch VALUES(2, 'Scranton', 102, '1992-04-06') ;

UPDATE employee
SET branch_id = 2
WHERE emp_id = 102 ;
```

테이블 생성시 FOREIGN KEY로 선언한 값은 입력에 제한이 발생하는 듯. FOREIGN키의 REFERENCES가 이미 생성된 TABLE에 존재하는(and 일치하는) 경우에만 입력이 가능한가?
일단은 ON DELETE NULL 조건과 FOREIGN키를 NULL로 처리하고 나중에 UPDATE하는 이유의 상관성은?
BRANCH에 VALUE들을 다 넣어놓고 입력하면 employee입력과 branch입력을 번갈아가며 할 필요가 없어지는거 아닌가?
↳ 테이블 값을 입력시 FOREIGN KEY가 없는 테이블 입력을 먼저 완료하는 것이 유리할 듯. 다만 현재 주어진 상황은 FOREIGN KEY의 두 테이블이 서로 REFERENCE의 대상이기 때문에 한 쪽 테이블을 먼저 완성시키는 형태의 입력방식이 작동하지 않는 듯

```
INSERT INTO employee VALUES(103, 'Angela', 'Martin', '1971-06-25', 'F', 63000, 102, 2) ;
INSERT INTO employee VALUES(104, 'Kelly', 'Kappor', '1980-02-05', 'F', 55000, 102, 2) ;
INSERT INTO employee VALUES(105, 'Stanley', 'Hudson', '1958-02-19', 'M', 69000, 102, 2) ;

--- Standford
INSERT INTO employee VALUES(106, 'Josh', 'Porter', '1969-09-05', 'M', 78000, 100, NULL) ;

INSERT INTO branch VALUES(3, 'Stanford', 106, '1998-02-13') ;

UPDATE employee
SET branch_id = 3
WHERE emp_id = 106 ;

INSERT INTO employee VALUES(107, 'Andy', 'Bernard', '1973-07-22', 'M', 65000, 106, 3) ;
INSERT INTO employee VALUES(108, 'Jim', 'Halpert', '1978-10-01', 'M', 71000, 106, 3) ;
```

…… Hopefully that shows how you might have to insert information into a more complex database skema. When we're just inserting into the student TABLE, it's really easy. But when we have foreign keys linking all over the place, it can get a little bit complicated.

now we can insert normally.

```
--- BRANCH SUPPLIER
INSERT INTO branch_supplier VALUES(2, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(2, 'Uni-ball', 'Writing Utensils');
INSERT INTO branch_supplier VALUES(3, 'Patriot Paper', 'Paper');
INSERT INTO branch_supplier VALUES(2, 'J.T. Forms & Labels', 'Custom Forms');
INSERT INTO branch_supplier VALUES(3, 'Uni-ball', 'Writing Utensils');
INSERT INTO branch_supplier VALUES(3, 'Hammer Mill', 'Paper');
INSERT INTO branch_supplier VALUES(3, 'Stamford Lables', 'Custom Forms');

--- CLIENT
INSERT INTO client VALUES(400, 'Dunmore Highschool', 2);
INSERT INTO client VALUES(401, 'Lackawana Country', 2);
INSERT INTO client VALUES(402, 'FedEx', 3);
INSERT INTO client VALUES(403, 'John Daly Law, LLC', 3);
INSERT INTO client VALUES(404, 'Scranton Whitepages', 2);
INSERT INTO client VALUES(405, 'Times Newspaper', 3);
INSERT INTO client VALUES(406, 'FedEx', 2);

-- WORKS_WITH
INSERT INTO works_with VALUES(105, 400, 55000);
INSERT INTO works_with VALUES(102, 401, 267000);
INSERT INTO works_with VALUES(108, 402, 22500);
INSERT INTO works_with VALUES(107, 403, 5000);
INSERT INTO works_with VALUES(108, 403, 12000);
INSERT INTO works_with VALUES(105, 404, 33000);
INSERT INTO works_with VALUES(107, 405, 26000);
INSERT INTO works_with VALUES(102, 406, 15000);
INSERT INTO works_with VALUES(105, 406, 130000);
```

## More Basic Queries

In this tutorial I'm gonna show you select statement which allow us query the company database.

```
--- Find all employees / Find all clients / Find the first name and last name of all employees / Fund the forename and surnames of all employees
        SELECT first_name AS forename, last_name AS surname
        FROM employee ;
--- Find all employees ordered by salary / Find all empoyees ordered by sex then name
--- Find the first 5 employees in the table
--- Find out all the different genders / Find out all the different branch ids
        SELECT DISTINCT sex
        FROM employees ;
```

This is little bit more about how we can use select queries and it kind of guve us how we might query data from the company database skema that we set up.

## Functions

--- Find the number of employees

```
SELECT COUNT(emp_id)
FROM employee ;
```

use a special sql function. when we run this we get 9. because there are 9 employees inside of the table

CREATE TABLE 명령에서는 각 필드의 성격을 부여한 뒤 ‘,’기호로 다음 줄(필드)과의 구분을 실시했었고,
INSERT INTO를 사용하여 데이터를 입력한 구문에서는 한 줄당 ‘;’ 기호로 쿼리를 완료시켰었는데,
SELECT - FROM - (WHERE) 구문은 하나의 구문이 여러 줄로 입력되고 있는 것이므로 줄이 전환될 때 별도의 기호가 삽입되지 않는다.

--- Find the number of female employees born after 1970

```
SELECT COUNT(emp_id)
FROM employee
WHERE sex = 'F' AND birth_date > '1971-01-01' ;
```

--- Find the average of all employee's salary

```
SELECT AVG(salary)
FROM employee ;
```

--- Find the sum of all employee's salary

```
SELECT SUM(salary)
FROM employee ;
```

--- Find out how many males and females there are

| | | |
|---|---|---|
| SELECT COUNT(sex)<br>FROM employee ; | SELECT COUNT(sex), sex<br>FROM employee ; | SELECT COUNT(sex), sex<br>FROM employee<br>GROUP BY sex ; |
| COUNT(sex)<br>9 | COUNT(sex) sex<br>9 M | COUNT(sex) sex<br>3 F<br>6 M |

I'm telling mysql to group the information by sex. It's counting how many employees in the sex column and print COUNT(sex) data along side whether they're male or female. that is what we call it aggregation.

--- Find the total sales of each salesman

```
SELECT SUM(total_sales), emp_id
FROM works_with
GOUP BY emp_id ;
```

--- Find the total expenditure of each client

```
SELECT SUM(total_sales), client_id
FROM works_with
GROUP BY client_id ;
```

we can use aggregation in order to organize the data that we get from using this function(?). I can add up the total_sales of each client and I can group them by client_id

엑셀에서 고급필터, 조건부서식 등등의 기능을 이용할 때는 구문(eg : SELECT - FROM - WHERE - GROUP BY)을 모두 입력할 필요없이 기능 사용에 필요한 항목들이 이미 분류되어 있어 항목이름명(여기서의 emp_id, client_id, works_with 등등) 정도만 알맞게 입력하면 되었음. SQL에서는 구문 전체가 완전하게 입력되어야 원하는 동작을 작동시킬 수 있는 차이가 있음.

## Wildcards

% = any # characters, _ = one character

```
--- Find any client's who are an LLC
SELECT *
FROM client
WHERE client_name LIKE '%LLC' ;
```

I can use a couple of different special characters. basically what I can do here is defining a pattern. If the *client_name* matches the pattern that I define here then, this condition(or this statement) will be true and end up returning that client.
Inside the quotation marks we can use 2 special characters. '%' sign stands for any number of characters and '_' which stands for a one character.

```
--- Find any branch suppliers who are in the label business
SELECT *
FROM branch_supplier
WHERE supplier_name LIKE '% Label %' ;

--- Find any employee born in October
SELECT *
FROM employee
WHERE birth_date LIKE '____-10%' ;

--- Find any clients who are schools
```

## Union

A special sql operator which we can use to combine the results of multiple SELECT statement into one. I might have 2-3 different SELECT statement that I',m using and if I wanted to combine all of them into same result and get a big list of table/ a big table back from database.

```
--- Find a list of employee and branch names
SELECT first_name AS Names
FROM employee
UNION
SELECT branch_name
FROM branch ;
```

now we have one single sql query which is going to ask the relational database management system to return not only the employee's first_name but also the branch_name's in a single column.

when we're using UNIONS you can do unions on a bunch of different things. There are a couple of rules. The first rule is that you have to have the same number of columns that you are getting in each SELECT statement. So in this SELECT statement I'm grabbing one column and in the second SELECT statement I'm grabbing one column. If I was to come up here(In the first SELECT statement) and also try to grab last_name. we're gonna get an error. because up here(In the first SELECT statement) we have 2 columns and down here(in the second SELECT statement) we have only one. So that's the first rule. You have to have the same number of columns.
They also have to have similar data type. first_name and branch_name the both are strings. Both of them are similar data type so we were able to return them in the same result. But if you have 2 thing what are very different data types, it might not necessarily work out as well.

```
--- Find a list of all clients & branch supplier's names

--- Find a list of all money spent or earned by the company
```

UNION basically combines ---- two SELECT statements. that's essentially all it does. but it ____ certain rules like you have to have same number of column in both statements and they have to be similar data type.

## Joins

--- Find all branches and the names of their managers

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

```
SELECT employee.emp_id, employee.first_name, branch.branch_name
FROM employee
JION branch
ON employee.employ_id = branch.mgr_id
```

when we get this table back, we're gonna be getting the emp_id, the employee's first_name, the branch_name
Essencially we combined a row from the branch TABLE with rows from employee TABLE into one single table. But we only combine them when employ_id is equal to the branch's mgr_id.

여기서는 JOIN이 두 테이블의 관계를 형성하는 것이 아닌 SELECT 구문내의 조건설정(WHERE 대신에)을 위해 사용되는 듯. 만약 branch TABLE mgr_id가 employee TABLE의 employ_id에 항상 일치해야만 하도록 설정하는 방법은 없나? 위에 방법은 그냥 ★★★WHERE employee.employ_id = branch.mgr_id로 기술했을 때는 작동하지 않는건가?

whenever we use LEFT JOIN, all of the rows in the TABLE of FROM statement are gonna get included in the results + only the rows in the branch table that match (employee.emp_id = branch.mgr_id) are gonna get included because the branch table is the right table.

RIGHT JOIN is gonna do the opposite. instead of including all the rows from the employee table, no matter what, now it's gonna include all of the rows from the branch table.

FULL OUTER JOIN. we can't do it in mysql. it basically LEFT JOIN and RIGHT JOIN combined.

## Nested query

nested query is basically a query where we're gonna be using muiltiple SELECT statements in order to get a specific piece of information. we are gonna need to inform the result of another SELECT statement(?).
A lot of information you're gonna want to get is gonna invole nested query.

--- Find names of all employees who have sold over 30,000 to a single client

the first thing I would do if we're trying to figure out how to write this query is look at the information that we have. we have works_with TABLE. we have total sales in works_with Table but what we don't have is employee's first_name and last_name. what we do have though is the id of employee. we can use the employee's id in order to get their first_name and last name. in this case we have a part of the data here in the works_with TABLE and we have the other part of data in the employee TABLE.

```
SELECT works_with.emp_id
FROM works_with
WHERE works_with.total_sales > 30000 ;
```

Write a query which is going to get me all of the employee id who have sold more than 30,000 to a single client.

```
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (
        SELECT works_with.emp_id
        FROM works_with
        WHERE works_with.total_sales > 30000
) ;
```

IN keyword is going to give us the result of specified values inside of ()

WHERE 바로 옆 employee.emp_id 는 () 안의 SELECT에서 works_with.emp_id와 관계가 있어야 하는 요소인가?
we're basically checking to see if emp_id is in 'IN( )' result.

--- Find all clients who are habdled by the branch that Michael Scott manages. Assume you know Michael's ID.

| 풀이 | 나의 풀이 |
| --- | --- |
| SELECT client.client_name<br>FROM client<br>WHERE client.branch_id = (<br>SELECT branch.branch_id<br>FROM branch<br>WHERE branch.mgr_id = 102<br>LIMIT 1<br>) ; | SELECT Client_client_name<br>FROM Client<br>WHERE Client.mgr_id IN (<br>SELECT branch.mgr_id<br>FROM branch<br>WHERE branch.mgr_id = 102<br>) ; |

## On Delete

in this tutorial, I'm gonna talk to you guys about deleting entries in the database when they have foreign keys associate to them.
Imagine I came over here in my employ table and I deleted one of the employees. We are to delete Michael Scott from the database. What's gonna happen to mgr_id in branch TABLE? the mgr_id is supposed to be linking us to an actual row of employ TABLE. but if we delete Michael Scott, 102, that deosn't mean anything.
This is what I'm gonna talk to you about today, which is different thing we can do to handle this situation. 1. ON DELETE SET NULL, If we delete one of eamployees, the mgr_id associate to that employee is gonna get set to NULL. 2. ON DELETE CASCADE, If we delete the employee whose id is stored in mgr_id column, then the entire row is gonna be deleted.

Employee

| emp_id | first_name | last_name | birth_day | sex | salary | super_id | branch_id |
|---|---|---|---|---|---|---|---|
| 100 | David | Wallace | 1967-11-17 | M | 25000 | NULL | 1 |
| 101 | Jan | Levinson | 1961-05-11 | F | 110000 | 100 | 1 |
| 102 | Michael | Scott | 1964-03-15 | M | 75000 | 100 | 2 |
| 103 | Angela | Martin | 1971-06-25 | F | 63000 | 102 | 2 |
| 104 | Kelly | Kapoor | 1980-02-05 | F | 55000 | 102 | 2 |
| 105 | Stanley | Hudson | 1958-02-19 | M | 69000 | 102 | 2 |
| 106 | Josh | Porter | 1969-09-05 | M | 78000 | 100 | 3 |
| 107 | Andy | Bernard | 1973-07-22 | M | 65000 | 106 | 3 |
| 108 | Jim | Halpert | 1978-10-01 | M | 71000 | 106 | 3 |

Branch

| branch_id | branch_name | mgr_id | mgr_start_date |
|---|---|---|---|
| 1 | Corporate | 100 | 2006-02-09 |
| 2 | Scranton | 102 | 1992-04-06 |
| 3 | Stanford | 106 | 1998-02-13 |

FOREIGN KEY(mgr_id) REFERENCES employee(emp_id) ON DELETE SET NULL ;
: if the emp_id in employee TABEL gets deleted, I wanna set the mgr_id equal to NULL.
DELETE FROM employee WHERE emp_id = 102 ;
employ 테이블에서 emp_id 102를 삭제했는데 emp_id를 foreign key로 설정했던 branch 테이블의 mgr_id의 값이 NULL로 변경됨.
employ 테이블의 super_id가 102이었던 엔트리는 NULL값으로 변경됨.(super_id도 employee.emp_id를 참조하는 foreign key로 설정했었음)

FOREIGN KEY(branch_id) REFERENCE branch(brach_id) ON DELETE CASCADE
: if the branch_id stored as foreign key in branch_supplier TABLE,
then we're just gonna delete the entire row in the database
DELETE FROM branch WHERE branch_id = 2 ;
★★★employ테이블이나 branch 테이블에서 branch_id 가 2였던 row들은 어떻게 변하지?

Branch_Supplier

| branch_id | supplier_name | supply_type |
|---|---|---|
| 2 | Hammer Mill | Paper |
| 2 | Uni-ball | Writing Utensils |
| 3 | Patriot Paper | Paper |
| 2 | J.T. Forms & Labels | Custom Forms |
| 3 | Uni-ball | Writing Utensils |
| 3 | Hammer Mill | Paper |
| 3 | Stamford Lables | Custom Forms |

in the branch table we used ON DELETE SET NULL, because the mgr_id on the branch table is just a foreign key not actually primary key. the mgr_id is not absolutely essential for the branch table. However, in the branch_supplier table, you notice that the branch_id is also a part of primary key. the branch_id on the branch_supplier table is absolutely crucial for this database. If the branch_id disappear we can't set that branch_id to NULL because a primary key can't have a NULL value. we just have to delete the entire thing. that's why we use ON DELETE CASCADE as opposit to(?) ON DELETE SET NULL

## Triggers

Trigger is basically a block of sql code which will define a certain action that should happen when a certain operation gets performed on a database. I can write a trigger which will tell mysql to do something when like a entry was added into a particular table on the database or when something is deleted from a database table.

```
CREATE TABLE trigger test(
        message VARCHAR(100)
) ;
```

Up to this point in this course we've been using this popsql. But when we are writing triggers, we have to define the triggers here in command line. that's because we have to change the sql delimiter. In order to do that we have to do that **inside of the terminal**.

if you are on the osx in your using terminal(?), you can just type in mysql -u root -p it will shoot prompt you for your password and you can log in.
mysql Command Line Client

```
DELIMITER $$
CREATE
        TRIGGER my_trigger1 BEFORE INSERT
        ON EMPLOYEE
        FOR EACH ROW BEGIN
                INSERT INTO trigger_test VALUES('added new employee');
        END$$
DELIMITER ;
```

DELIMITER를 $$로 바꿔도 CREATE 명령 내에서는 여전히 ';'가 구분문자로 작동하는 건가?

```
INSERT INTO employee
VALUES(109, 'Oscar', 'Martinez', '1968-02-19', 'M', 69000, 106, 3) ;
```

로 새로운 데이터를 입력하면 triger_test TABLE에 added new employee 라는 글이 삽입된다.

```
DELIMITER $$
CREATE
        TRIGGER my_trigger2 BEFORE INSERT
        ON employee
        FOR EACH ROW BEGIN
                INSERT INTO trigger_test VALUES(NEW.first_name) ;
        END $$
DELIMITER ;
```

employee테이블에 NEW 데이터가 입력되면 first_name이 my_trigger TABLE에 삽입된다.

```
INSERT INTO employee
VALUES(110, 'Kevin', Malone', '1978-02-19', 'M', 69000, 106, 3) ;
```

```
DELIMITER $$
CREAT
        TRIGGER my_trigger3 BEFORE(AFTER) INSERT(/UPDATE/DELETE)
        ON employee
        FOR EACH ROW BEGIN
                IF NEW.sex = 'M' THEN
                        INSERT INTO trigger_test VALUES('added male employee') ;
                ELSE IF NEW.SEX = 'F' THEN
                        INSERT INTO trigger_test VALUES('added female') ;
                ELSE
                        INSERT INTO trigger_test VALUES('added other employee') ;
                END IF ;
        END $$
DELIMITER ;

INSET INTO employee
VALUES(111, 'Pam', Beasly', '1988-02-19', 'F', 69000, 106, 3) ;
```

## ER Diagrams Intro

ER = Entity Relationship
How ER Diagram put together different symbols and ER Diagram what they represent

you can use the ER-Diagram to map out the different relationship, and the different entities and the different attributes for those entities.

Entity : An object we want to model & store information about
Attributes : Specific pieces of information about an entity
Primary Key : An attribute(s) that uniquely identify an entry in the database table
Composite Attribute : An attribute that can be broken up into sub-attributes
Multi-valued Attribute : An attribute that can have more than one value (eg : Stundents might be involved in many different clubs, Students can have more than one club that they belong to)
Derived Attribute : An attribute that can be derived from the other attributes
(eg : Maybe school is going to say that anybody with the gpa of 3.5 above will have honors, age attribute from employee entity, which is derived from birh_date attribute)
Multiple Entity : You can define more than one entity in the diagram
Relationships : Defines a relationship between two entities.
Total participation : All members must participate in the relationship.


Not all students need to take a class. Only some of the students need to take a class.(partial participation)
All of the classes need to be taken by at least a single student. All classes need to have students who are taking them.

Relationship Attribute : An attribute about the relationship
Relational Cardinality : the number of instances of an entity from a relation that can be associated with the relation(N:M, 1:N, 1:1)
Weak Entity : An entity that cannot be uniquely identified by its attributes alone. an entity that is going to rely on or depend on another entity

<both entities have total participation in the works for relationship>
All branches must have employees working for them and All employee must work for a branch.
<every branch is gonna particiapte in that 'Manages' relationship, Not all employees need to be managers of a branch>
we have full participation. All branches must have someone managing them. In fact, by large majority, most employees will not be the manager of a branch

<cardinality relationship>
brach can have any number of employees working for it and a employee can work for one branch

## Converting ER Diagrams into Skemas

Step1 : Mapping of Regular Entity Types(which is just Table)

Step2 : Mapping of Weak Entity Types

the primary key of the new relation should be the partial key of the weak entity plus primary key of it's own.

Branch Supplier

| branch_id | supplier_name | supply_type |
|---|---|---|

For this table, we included supplier_name and branch_id, both of them come together, to make a composite key(compound key).

Step3 : Mapping of Binary 1:1 Relationship Type

Include one side of the relationship as a foreign key in the other Favor total participation

Branch

| branch_id | branch_name | mgr_id | mgr_start_date |
|---|---|---|---|

Step4 : Mapping of Binary 1:N Relationship Type

Include the 1 side's primary key as a foreign key on the N side relation(table)

Client

| client_id | client_name | branch_id |
|---|---|---|

Step5 : Mapping of Binary M:N Realationship Types

Create a new relation(table) whose primary key is a combination of both entities' primary keys.

Work_On

| emp_id | client_id | total_sales |
|---|---|---|