INTERNATIONAL UNIVERSITY OF JAPAN
Public Management and Policy Analysis Program
Graduate School of International Relations

ADC5030401 (2 Credits)
Introduction to Electronic Government
Winter 2019

This document explains Structured Query Language (SQL) to manipulate databases. Reserved words such as CREATE, TABLE, ALTER, and DROP are purposively uppercased in this document although SQL is not case-sensitive. Keep in mind that some commands used here are specific to MySql.

1. SQL Basics

This section discusses basics of structured query language such as operators and functions.

1.1 Data Definition and Data Manipulation Languages

SQL has data definition language (DDL), data manipulation language (DML), and data control language (DCL). DDL commands include CREATE, ALTER, DROP, and TRUNCATE. Notice that each SQL command ends with semi-colon (;).

- `CREATE DATABASE …;`
- `CREATE TABLE …;`
- `CREATE TABLE … AS SELECT … FROM …;`
- `ALTER DATABASE …;`
- `ALTER TABLE …;`
- `DROP DATABASE …;`
- `DROP TABLE …;`
- `DROP INDEX …;`
- `TRUNCATE TABLE …;`

CREATE creates a database or table; ALTER changes data structures; DROP remove a database, table, and other objects like index; and TRUNCATE delete all records in a table. CREATE TABLE may include some integrity constraints such as NOT NULL (null is not allowed), UNIQUE (a record should be unique), PRIMARY KEY (set a primary key), and DEFAULT (set a default value).

Common DML includes SELECT, INSERT, UPDATE, and DELETE. SELECT displays information from tables; INSERT adds records to a table; UPDATE replaces existing information in a table with new one; and DELETE removes a subset of records from a table.

- `SELECT … FROM …;`
- `SELECT … FROM … WHERE …;`
- `SELECT … FROM … HAVING …;`
- `SELECT … FROM … GROUP BY …;`
- `SELECT … FROM … ORDER BY …;`
- `SELECT … FROM … WHERE … GROUP BY … ORDER BY …;`
- `INSERT INTO … VALUES (…);`

- `UPDATE … SET …;`
- `DELETE FROM … WHERE …;`

The FROM clause specifies tables to be read; WHERE specifies conditions by which a subset of records are chosen; HAVING specifies conditions for aggregate functions; GROUP BY, often used in conjunction with aggregate functions, arranges the result by groups set by this clause; and ORDER BY sorts the result to be displayed. When you want to make some changes in data, pay special attention to WHERE and HAVING so that the right subset of records will be chosen.

DCL includes GRANT and REVOKE to control privileges given to database users.

1.2 Common Data Types

MySql supports various data types for numbers, string, and date and time. BIGINT and DOUBLE consume 8 bytes to store a value. In DECIMAL and DOUBLE, *m* is the total number of total digits used to represent a number and *d* is the number of digits below the decimal point. For instance, DECIMAL(10,2) stores a number 10 digits long including two digits below the decimal point (xxxxxxxx.xx).

Integer	Byte	Range	Real number	String	Date/Time	Others
`INT()`	4	-2.1E-9	`DECIMAL(m,d)`	`CHAR()`	`DATE`	`BINARY()`
`TINYINT ()`	1	-128 ~ 127	`FLOAT`	`VARCHAR()`	`DATETIME`	
`SMALLINT ()`	2	-32,768	`DOUBLE(m,d)`	`TEXT`	`YEAR`	
`MEDIUMINT()`	3	-8.4 millions				
`BIGINT()`	8	-9.2E-18				

If a string attribute has a fixed length (e.g., CA, IN, and IL), use CHAR(). When you need a binary attribute, use either TINYINT (-128~127) or BINARY (0~255).

An iron rule says, "Try to use a proper numeric type." TINYINT is better than INT for age (you can save 3 bytes per data item); BINARY is better than INT, CHAR, VARCHAR, and FLOAT for male (female).

1.3 Operators

SQL supports arithmetic, logical, relative, and other operators. In order for clarification, you need to use parentheses properly in order to avoid unintended mistake and confusion.

- Arithmetic operators: +, -, *, /, and ^ (power).
- Logical operators: AND, OR, and NOT.
- Relational operators: <, <=, >, >=, = (equal), and <> (not equal)
- Special operators: LIKE, BETWEEN, IS NULL, IN, and EXITS

Let us take some examples. The last example shows how parentheses are effectively used to clarify the condition.

```
… WHERE income >= 50000
… WHERE program = "IDP" OR program = "PMPP"
… WHERE (male = 0) AND (country = "Japan")
… WHERE (founded <= 1980) AND (sic = 7372 OR sic = 5371)
```

Special operators appear to be complicated but provide flexibility and efficiency instead.

```
… WHERE name LIKE 'P%';
… WHERE ssn LIKE '%13_';
… WHERE income BETWEEN 20 AND 100;
… WHERE ssn IS NULL;
… WHERE id IN(004, 007, 009);
… WHERE nationality IN('Korea', 'Indonesia', 'Sri Lanka');
… WHERE EXISTS (SELECT * FROM order WHERE order_quantity > 0);
```

The first example chooses all records whose attribute name begins with P (e.g., Park, Patrick, …); the second selects records whose ssn ends with13 plus any one character(e.g., …13A, …133, and …13$); the third includes records whose income ranges from 20 to 100; the fourth chooses any records whose ssn is not empty (missing value); the second from the bottom finds records whose id is 004, 007, or 009; and the last EXITS checks if the result of another query is not empty (no record meets the condition specified).

1.4 Functions

SQL has mathematical, string, and aggregate functions. Aggregate functions produce aggregate information such as the number of records that meet the condition (`COUNT`).

Aggregate Functions	Math Functions	String Functions
`COUNT()`	`ABS()`	`LENGTH()`
`MIN()`	`SIN(); COS()`	`TRIM(); LTRIM(); RTRIM()`
`MAX()`	`CEIL(); FLOOR()`	`LEFT(); RIGHT()`
`SUM()`	`EXP(); LN()`	`CHAR()`
`AVG()`	`MOD();`	`LOCATE(pattern, string);`
`FIRST()`	`POWER()`	`LOWER(); UPPER()`
`LAST()`	`ROUND()`	`REPLACE(string,from,to)`
	`SQRT()`	`SUBSTR(string,start,length)`

For instance, COUNT() returns the number of records that meet the condition specified and AVG() (Not AVERAGE) produces average of an attribute or expression. LENGTH() reports the number of characters used in a string attribute. LOCATE('pattern', string) returns the first starting position of a pattern in a string; `LOCATE(',', name)` returns the position where the first comma appears in an attribute name. SUBSTR(string, start, length) returns a part of a string from the starting point. The following command returns the last name from an attribute `name`.

```
mysql> SELECT SUBSTR(name, 1, LOCATE(',', name)-1 ) FROM student;
```

1.5 Naming Tables and Database

Please follow the naming convention strictly (no space, short name, lowercased, …). A database name and a table name are separated by a dot. If you choose a database, you may omit the database name.

```
mysql> USE kucc625;

mysql> SELECT * FROM firm;
```

However, you need to specify database, in particular, when joining tables. The following expression means a table `firm` in a database `kucc625`.

```
mysql> SELECT * FROM kucc625.firm;
```

2. Start MySql

Let us start MySql at the Linux prompt using your account (replace "kucc625" with your nickname). First, change the working directory to `www`.

```
$ cd ~/www

$ mysql -u kucc625 -p
Enter password:
```

The above command says, "Start MySql using a database user `kucc625` and its password will be provided." When you hit Enter, MySql server asks you to type in your password. If authentication is successful, you will get the MySql prompt `mysql>`.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 5.7.21-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

In order to import .csv using the LOAD command, you need to start MySql with the `--local-infile` option as follows.

```
$ mysql -u kucc625 -p --local-infile
```

3. Create a Database

First, check current databases available in your database account. Run SHOW DATABSES to list the databases available for you. Do not omit semi-colon (;) at the end of a MySql command.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
```

You may get a different list of databases. A database with your nickname was created already. The following command created your database (replace `kucc625` with your nickname).

```
mysql> CREATE DATABASE kucc625 CHARACTER SET UTF8 COLLATE UTF8_GENERAL_CI;
```

The clause from CHARACTER SET sets the character set used in the database. UTF8 is the most common UNICODE character set for multiple languages. Alternatively, you can create a database and then change the character set and collate of a database using ALTER DATABASE.

```
mysql> CREATE DATABASE kucc625;
```

```
mysql> ALTER DATABASE kucc625 CHARACTER SET = utf8;

mysql> ALTER DATABASE kucc625 COLLATE = utf8_unicode_ci;
```

Now, check there are two databases.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kucc625            |
+--------------------+
```

If you want to remove a database, run DROP DATABASE followed by the database name.

```
mysql> DROP DATABASE kucc625;
```

4. Create a Table

Open your database using the USE command as if you changed a directory.

```
mysql> USE kucc625;
```

Before creating a table, run SHOW TABLES to check current tables available in your database. There is no table in the database `kucc625`.

```
mysql> SHOW TABLES;
```

4.1 Create a Table

Let us create a table `firm` using CREATE TABLE. This command is followed by a table name and a list of attribute definitions in parenthesis.

```
mysql> CREATE TABLE firm (firm VARCHAR(15) NOT NULL, firm_no TINYINT(2) NOT NULL,
ticker CHAR(4) NOT NULL, sic SMALLINT(4), gics BIGINT(8), founded YEAR, hq
VARCHAR(17), product VARCHAR(17), revenue DECIMAL(10.0) DEFAULT 0, employee
DECIMAL(10.0) DEFAULT 0, PRIMARY KEY (firm_no) );
```

PRIMARY KEY in the last line defines an attribute `firm_no` as primary key. The entire command above appears to be quite complicated and confusing. MySql allows you to type in a command in multiple lines; MySql consider them as a continued command until encountering a semi-colon.

```
mysql> CREATE TABLE firm (
    -> firm VARCHAR(15) NOT NULL,
    -> firm_no TINYINT(2) NOT NULL,
    -> ticker CHAR(4) NOT NULL,
    -> sic SMALLINT(4),
    -> gics BIGINT(8),
    -> founded YEAR,
    -> hq VARCHAR(17),
    -> product VARCHAR(17),
    -> revenue DECIMAL(10.0) DEFAULT 0,
    -> employee DECIMAL(10.0) DEFAULT 0,
    -> PRIMARY KEY (firm_no) );
```

This CREATE TABLE has several integrity constraints such as NOT NULL, AUTO_INCREMENT, and DEFAULT. An attribute `firm` and `firm_no` must have a certain value (NOT NULL) and the default value of `revenue` is 0 (DEFAULT 0). Finally, PRIMARY KEY sets `firm_no` as a primary key.

Let us create another tables `balance`.

```
mysql> CREATE TABLE balance (firm VARCHAR (15) NOT NULL, firm_no TINYINT(2) NOT
NULL, year YEAR, item VARCHAR(35), item_no SMALLINT(4), value DECIMAL(10.0) DEFAULT
0 );
```

Now, we have two tables.

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_kucc625 |
+-------------------+
| balance           |
| firm              |
+-------------------+
```

4.2 Viewing a Table Structure

Now, take a look at the data structure of the tables `firm` and `balance`. Use either DESCRIBE or EXPLAIN followed by a table name.

```
mysql> DESCRIBE firm;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| firm     | varchar(15)   | NO   |     | NULL    |       |
| firm_no  | tinyint(2)    | NO   | PRI | NULL    |       |
| ticker   | char(4)       | NO   |     | NULL    |       |
| sic      | smallint(4)   | YES  |     | NULL    |       |
| gics     | bigint(8)     | YES  |     | NULL    |       |
| founded  | year(4)       | YES  |     | NULL    |       |
| hq       | varchar(17)   | YES  |     | NULL    |       |
| product  | varchar(17)   | YES  |     | NULL    |       |
| revenue  | decimal(10,0) | YES  |     | 0       |       |
| employee | decimal(10,0) | YES  |     | 0       |       |
+----------+---------------+------+-----+---------+-------+
```

Try to run EXPLAIN with both database and table names separated by a period.

```
mysql> EXPLAIN kucc625.balance;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| firm    | varchar(15)   | NO   |     | NULL    |       |
| firm_no | tinyint(2)    | NO   |     | NULL    |       |
| year    | year(4)       | YES  |     | NULL    |       |
| item    | varchar(35)   | YES  |     | NULL    |       |
| item_no | smallint(4)   | YES  |     | NULL    |       |
| value   | decimal(10,0) | YES  |     | 0       |       |
+---------+---------------+------+-----+---------+-------+
```

4.3 Copy a Table

The following CREATE TABLE… AS… copies a table from the query (getting all pieces of information from a table `firm`) and then pastes into a new table `firm2`.

```
mysql> CREATE TABLE firm2 AS SELECT * FROM firm;
```

4.4 Change a Table Name

You can change a table name using RENAME TABLE… TO…. You may specific database and table names separated by a dot.

```
mysql> RENAME TABLE firm2 TO kucc625.firm3;
```

4.5 Remove a Table

DROP TABLE removes a table. First, check the list of tables available now in the database before executing the command.

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_kucc625 |
+-------------------+
| balance           |
| firm              |
| firm3             |
+-------------------+

mysql> DROP TABLE firm3;
```

5. Change the Data Structure of a Table

ALTER TABLE changes the data structure of a table. This command has many clauses such as MODIFY, CHANGE, ADD, ALTER, RENAME, and DROP. Let us copy `firm` and create `test` for practice.

```
mysql> CREATE TABLE test AS SELECT * FROM firm;
```

5.1 Modify Existing Attributes

You can change an attribute using the MODIFY clause, which modifies the type of an existing attribute. Let us modify data types of an attribute `employee`.

```
mysql> ALTER TABLE test MODIFY employee INT(5);

mysql> ALTER TABLE test ALTER employee SET DEFAULT 0;

mysql> ALTER TABLE test ALTER employee DROP DEFAULT;
```

First ALTER with MODIFY changes the current data type of `employee` from DECIMAL (10,0) to INT (6). Second and third ALTER with ALTER sets and resets the default value of `employee`. Try to change data types of other variables.

```
mysql> ALTER TABLE test MODIFY gics CHAR(8) NOT NULL;
```

5.2 Change an Existing Attribute into New One (Rename Attributes)

Now, try to change data type using CHANGE. This clause requires current and new attribute names. The following command renames the current `revenue` to `sales` and change variable type from DECIMAL(10,0) to INT(10) with NOT NULL and DEFAULT 0.

```
mysql> ALTER TABLE test CHANGE revenue sales INT(10) NOT NULL DEFAULT 0;
```

Now, let us check the latest data structure of the example table `test`.

```
mysql> DESCRIBE test;
```

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| firm     | varchar(15)  | NO   |     | NULL    |       |
| firm_no  | tinyint(2)   | NO   |     | NULL    |       |
| ticker   | char(4)      | NO   |     | NULL    |       |
| sic      | smallint(4)  | YES  |     | NULL    |       |
| gics     | char(8)      | NO   |     | NULL    |       |
| founded  | year(4)      | YES  |     | NULL    |       |
| hq       | varchar(17)  | YES  |     | NULL    |       |
| product  | varchar(17)  | YES  |     | NULL    |       |
| sales    | int(10)      | YES  |     | NULL    |       |
| employee | int(5)       | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

5.3 Add New Attributes

The ADD clause adds attributes and other properties to a table. Let us first add a primary key to `test`.

```
mysql> ALTER TABLE test ADD PRIMARY KEY (firm);
```

If you want to remove the primary key, run ALTER with the DROP PRIMARY KEY clause.

```
mysql> ALTER TABLE test DROP PRIMARY KEY;
```

Now, add attributes to the current table. You may provide more than one ADD clause in an ALTER TABLE command.

```
mysql> ALTER TABLE test ADD profit DECIMAL(10.2) DEFAULT 0;

mysql> ALTER TABLE test
    -> ADD foreigner BINARY(1) NOT NULL,
    -> ADD note TEXT AFTER employee;
```

The AFTER clause in the last command above specifies the location to which a new attribute is added. Let us check the latest data structure and make sure the changes you have made so far.

5.4 Remove an Attribute

ALTER with the DROP clause removes an attribute from a table. Do not be confused by DROP TABLE that removes a table from a database! Be careful whenever you use the DROP command and DROP clause in the ALTER command.

```
mysql> ALTER TABLE test DROP note;
```

5.5 Rename a Table

Like RENAME TABLE, ALTER TABLE… RENAME changes a table name.

```
mysql> ALTER TABLE test RENAME test2;

mysql> DROP TABLE test2;
```

6. Import an External File (CSV)

Let us read CSV (Comma Separated Values) files into a MySql table.

6.1 File Preparation

You need to prepare data files that have the same data structures of tables defined in MySql. Now, let us exit MySql console to get back to Linux prompt. Change the working directory to your www directory.

```
mysql> EXIT;

kucc625@joosin:~$ cd ~/www
```

Now, download the archived CSV file (firm.zip) of firm and balance from the course Web page. Then upload firm.zip to your www directory. Extract firm.zip using the 7z utility with the *e* option and check if all CSV files are there.

```
~/www$ 7z e firm.zip

~/www$ ls -al *.csv
-rw-r--r-- 1 kucc625 kucc625 9877 Apr  7 22:57 balance.csv
-rw-r--r-- 1 kucc625 kucc625 1014 Apr  7 22:50 firm.csv
```

6.2 Read CSV Files

Start MySql with your database account. Don't forget to add `--local-infile`.[1]

```
~/www$ mysql -u kucc625 –p --local-infile
```

At the MySql prompt, run the following LOAD DATA DATA INFILE command to read `firm.csv` and then import into the table `firm`.[2]

```
mysql> LOAD DATA LOCAL INFILE '/home/kucc625/www/firm.csv' INTO TABLE firm
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
    -> (firm, firm_no, ticker, sic, gics, founded, hq, product, revenue, employee);
Query OK, 13 rows affected, 30 warnings (0.03 sec)
Records: 13  Deleted: 0  Skipped: 0  Warnings: 30
```

MySql responds by saying that 13 records were imported into the table `firm`. Repeat this process for `balance` as follows.

```
mysql> LOAD DATA LOCAL INFILE '/home/kucc625/www/balance.csv' INTO TABLE balance
FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n' (firm, firm_no,
year, item, item_no, value);
```

7. Display Information from Tables

SELECT retrieves information from a table or linked tables. You can obtain information out of databases using this command. It has many clauses FROM, WHERE, ORDER BY, GROUP BY, and HAVING.

7.1 SELECT Basics

SELECT is followed by a list of attributes to be displayed and then the FROM clause that specifies table(s) from which attributes are retrieved.

---

[1] If you skip this option, you will get "`ERROR 1148 (42000): The used command is not allowed with this MySQL version`"

[2] Alternatively, you may use the `sqlimport` utility from the UNIX prompt.

```
mysql> SELECT firm, founded, hq, revenue, employee FROM firm;
+-----------------+---------+------------------+---------+----------+
| firm            | founded | hq               | revenue | employee |
+-----------------+---------+------------------+---------+----------+
| adobe           |    1982 | San Jose, CA     |       4 |       11 |
| apple           |    1976 | Cupertino, CA    |     171 |       80 |
| dell            |    1984 | Austin, TX       |      57 |      109 |
| facebook        |    2004 | Menlo Park, CA   |       8 |        6 |
| google          |    1998 | Mountain View, CA |      60 |       48 |
| hewlett-packard |    1939 | Palo Alto, CA    |     112 |      318 |
| ibm             |    1911 | Armonk, NY       |     100 |      434 |
| intel           |    1968 | Santa Clara, CA  |      53 |      107 |
| microsoft       |    1975 | Redmond, WA      |      78 |      101 |
| oracle          |    1977 | Redwood City, CA |      37 |      123 |
| symantec        |    1982 | Mountain View, CA |       7 |       20 |
| verizon         |    1983 | New York         |     121 |      177 |
+-----------------+---------+------------------+---------+----------+
```

You may use * to display all attributes in a table. It is convenient to list attributes but will give you overwhelming output if the table has large numbers of attributes.

```
mysql> SELECT * FROM balance WHERE value >= 60000;
+-----------+---------+------+--------------------+---------+--------+
| firm      | firm_no | year | item               | item_no | value  |
+-----------+---------+------+--------------------+---------+--------+
| microsoft |       9 | 2004 | Cash & Equivalents |       1 |  60592 |
| microsoft |       9 | 2012 | Cash & Equivalents |       1 |  63040 |
| microsoft |       9 | 2013 | Cash & Equivalents |       1 |  77022 |
| apple     |       2 | 2011 | Common Equity      |      36 |  76615 |
| apple     |       2 | 2012 | Common Equity      |      36 | 118210 |
| apple     |       2 | 2013 | Common Equity      |      36 | 123549 |
| google    |       5 | 2012 | Common Equity      |      36 |  71715 |
| google    |       5 | 2013 | Common Equity      |      36 |  87309 |
| microsoft |       9 | 2003 | Common Equity      |      36 |  61020 |
| microsoft |       9 | 2004 | Common Equity      |      36 |  74825 |
| microsoft |       9 | 2012 | Common Equity      |      36 |  66363 |
| microsoft |       9 | 2013 | Common Equity      |      36 |  78944 |
+-----------+---------+------+--------------------+---------+--------+
```

7.2 Choose a Subset of Records

Oftentimes you need to get a subset of records of a table. The WHERE clause specifies a set of conditions and chooses a subset of the table that meet the condition.

```
mysql> SELECT firm, revenue, employee FROM firm WHERE revenue > 50;

mysql> SELECT firm, founded FROM firm WHERE (revenue > 50) AND (employee > 50);
```

When handling string values, enclose them with quotes. You may use operators and functions in the condition.

```
mysql> SELECT firm, revenue, employee FROM firm WHERE firm = 'google';
+--------+---------+----------+
| firm   | revenue | employee |
+--------+---------+----------+
| google |      60 |       48 |
+--------+---------+----------+

mysql> SELECT firm, revenue, employee FROM firm WHERE firm LIKE 'a%';
+-------+---------+----------+
| firm  | revenue | employee |
+-------+---------+----------+
| adobe |       4 |       11 |
```

```
| apple |     171 |       80 |
+-------+---------+----------+

mysql> SELECT firm, revenue, employee FROM firm WHERE LENGTH(TRIM(firm)) >= 7;
+-----------------+---------+----------+
| firm            | revenue | employee |
+-----------------+---------+----------+
| facebook        |       8 |        6 |
| hewlett-packard |     112 |      318 |
| microsoft       |      78 |      101 |
| symantec        |       7 |       20 |
| verizon         |     121 |      177 |
+-----------------+---------+----------+
```

The last condition indicates the length of the string variable `firm` (LENGTH) after removing heading and ending spaces (TRIM).

7.3 Sort the Result

You may sort the result in the ascending (ASC), default, or descending order (DESC).

```
mysql> SELECT firm, revenue FROM firm WHERE hq LIKE '%CA%' ORDER BY revenue DESC;
+-----------------+---------+
| firm            | revenue |
+-----------------+---------+
| apple           |     171 |
| hewlett-packard |     112 |
| google          |      60 |
| intel           |      53 |
| oracle          |      37 |
| facebook        |       8 |
| symantec        |       7 |
| adobe           |       4 |
+-----------------+---------+
```

You may sort on more than one attribute. The following SELECT sorts on `product` in the ascending order and then on `revenue` in the descending order.

```
mysql> SELECT firm, product, revenue FROM firm ORDER BY product, revenue DESC;
+-----------------+--------------------+---------+
| firm            | product            | revenue |
+-----------------+--------------------+---------+
| facebook        | clouding           |       8 |
| apple           | computer           |     171 |
| hewlett-packard | computer           |     112 |
| ibm             | computer           |     100 |
…
| oracle          | software           |      37 |
| symantec        | software           |       7 |
| adobe           | software           |       4 |
| verizon         | telecommunication  |     121 |
| google          | Web                |      60 |
+-----------------+--------------------+---------+
```

7.4 Get Aggregate Statistics

You may get aggregate information such as average and maximum of revenue using aggregate functions as attributes. Notice that AVG returns the average of an attribute.

```
mysql> SELECT SUM(revenue), AVG(revenue), MAX(employee), MIN(employee) FROM firm;
+--------------+--------------+---------------+---------------+
| SUM(revenue) | AVG(revenue) | MAX(employee) | MIN(employee) |
+--------------+--------------+---------------+---------------+
```

```
|          808 |      67.3333 |           434 |             6 |
+--------------+--------------+---------------+---------------+
```

You may provide an alias using AS to give a meaning name for the aggregate information.

```
mysql> SELECT SUM(revenue) AS rev_sum, AVG(revenue) AS rev_mean,
    -> MAX(employee) AS emp_max, MIN(employee) AS emp_min FROM firm;
+---------+----------+---------+---------+
| rev_sum | rev_mean | emp_max | emp_min |
+---------+----------+---------+---------+
|     808 |  67.3333 |     434 |       6 |
+---------+----------+---------+---------+
```

You may provide an expression in the aggregate function to get more meaningful information.

```
mysql> SELECT AVG(revenue/employee) AS per_rev FROM firm;
+------------+
| per_rev    |
+------------+
| 0.73267122 |
+------------+
```

7.5 Categorize the Result

Aggregate information will be more meaningful, if it is provided category by category. The GROUP BY clause categorizes groups to be displayed. COUNT reports the number of records in each group.

```
mysql> SELECT product, count(firm), AVG(revenue) AS per_rev FROM firm
    -> GROUP BY product;
+-------------------+-------------+----------+
| product           | count(firm) | per_rev  |
+-------------------+-------------+----------+
| clouding          |           1 |   8.0000 |
| computer          |           3 | 127.6667 |
| hardware          |           3 |  36.6667 |
| product           |           1 |   0.0000 |
| software          |           4 |  31.5000 |
| telecommunication |           1 | 121.0000 |
| Web               |           1 |  60.0000 |
+-------------------+-------------+----------+
```

7.6 Limit Aggregate Statistics

You may limit listing to a subset of aggregate statistics using the HAVING clause (WHERE is not allowed in this usage). The following command list industries whose average revenue is greater than USD 100 billions.

```
mysql> SELECT product, count(firm), AVG(revenue) AS per_rev FROM firm
    -> GROUP BY product HAVING AVG(revenue) > 100;
+-------------------+-------------+----------+
| product           | count(firm) | per_rev  |
+-------------------+-------------+----------+
| computer          |           3 | 127.6667 |
| telecommunication |           1 | 121.0000 |
+-------------------+-------------+----------+
```

8. Add Records to a Table

INSERT INTO appends records to a table. The values of attributes are provided in the VALUES clause. The list of attributes should match the corresponding list of values.

```
mysql> INSERT INTO firm (firm, firm_no, ticker, founded, hq, revenue, employee)
    -> VALUES ("lg", 13, "LG", 1958, "Seoul, Korea", 45.2, 86.7);
```

When you omit a list of attributes, you must provide all values of attributes. Make sure that the values are arranged in the same order as attributes of the table.

```
mysql> INSERT INTO firm VALUES ("lg", 13, "LG", 3571, 45202008, 1958,
    -> "Seoul, Korea", "Appliance", 45.2, 86.7);
```

You may provide more than one record at a time. Each set of values is separated by a comma.

```
mysql> INSERT INTO firm VALUES ( … ), ( … ), … ( … );
```

You may also insert records from other tables using INSERT INTO … SELECT … FROM. The SELECT … FROM … in this usage is called a *nested query* (subquery or inner query). Make sure that you select right records and that each attribute listed in both tables has the same data type. For instance, `firm.revenue` and `company.sale` must share the same data type (DECIMAL)

```
mysql> INSERT INTO firm (firm, revenue, employee) SELECT company.name, company.sale,
    -> company.labor FROM company WHERE company.country='Korea';
```

9. Update Existing Records in a Table

UPDATE … SET replaces current information in a table with new one. Since UPDATE involves changes in databases, you must make sure that the right records are selected.

```
mysql> UPDATE firm SET firm = "LG Electronics" WHERE ticker="LG";
```

You may update more than one attribute at a time.

```
mysql> UPDATE firm SET sic=3571, gics=45202008 WHERE firm = "LG Electronics";
```

10. Delete Records from a Table

DELETE FROM removes records from a table. Check the condition carefully and then make sure that you choose the right records of a table.

```
mysql> SELECT firm, product, revenue, employee FROM firm WHERE product = "software";

mysql> DELETE FROM firm WHERE product = "software";

mysql> SELECT founded, revenue FROM firm WHERE (founded < 1990) AND (employee < 50);

mysql> DELETE FROM firm WHERE (founded < 1990) AND (employee < 50);
```

TRUNCATE TABLE truncates (delete all records in) the table. But be cautious whenever using DELETE and TRUNCATE.

```
mysql> TRUNCATE TABLE test;
```

11. Join Tables (Relations)

Relation (joining tables) characterizes the relational databases. Joining requires key attributes that are shared by related tables. The degrees of relationships include unary, binary, ternary relationships. The connectivity can be one-to-one, one-to-many, and many-to-many.

Join has various types including cross join, inner join, and outer join. An outer join can be left join, right join, or full join.

11.1 Cross Join to Get Cartesian Product

A *cross join* returns the Cartesian product of two tables. That is, a cross join considers all possible combinations of every element of each set. If the number of records that meet the condition is 10 in table A and 5 in B, 50 (= 10 X 5) records will be displayed. This cross join is less frequently used in practice because the result tends to be messy.

In order for a cross join, you need to list table names separated by a comma or CROSS JOIN in the FROM clause. For example,

```
mysql> SELECT … FROM table1, table2 … WHERE …

mysql> SELECT … FROM table1 CROSS JOIN table2 … WHERE …
```

The following cross join produces 3146 combinations (=13 X 242) of student and instructor tables. Yes, the result should be messy. Notice that there are 13 companies in table `firm` and 242 balance items in `balance`.

```
mysql> SELECT firm.firm, balance.year, balance.value FROM firm, balance;
+-----------------+------+--------+
| firm            | year | value  |
+-----------------+------+--------+
| adobe           | 2003 |   1097 |
| apple           | 2003 |   1097 |
| dell            | 2003 |   1097 |
| facebook        | 2003 |   1097 |
| google          | 2003 |   1097 |
| hewlett-packard | 2003 |   1097 |
| ibm             | 2003 |   1097 |
| intel           | 2003 |   1097 |
| microsoft       | 2003 |   1097 |
| oracle          | 2003 |   1097 |
| symantec        | 2003 |   1097 |
| verizon         | 2003 |   1097 |
| lg              | 2003 |   1097 |
| adobe           | 2004 |   1313 |
| apple           | 2004 |   1313 |
| dell            | 2004 |   1313 |
…
```

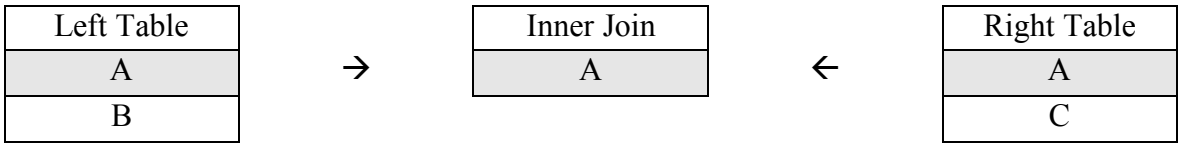
11.2 Inner Join

An *inner join* returns only matched records from joining tables. This join is most commonly used in practice. The old inner join is nothing but cross join with a WHERE clause to match attributes. For example, the following inner join produces 6 combinations of three female Sri Lankan students (Pathmini, Sureka, and Uttara) in table `student` and only two female instructors (Kimura and Kumagai) in `instructor`.

```
mysql> SELECT firm.firm, balance.year, balance.value FROM firm, balance WHERE
    -> firm.firm = balance.firm;
+-----------------+------+--------+
| firm            | year | value  |
+-----------------+------+--------+
| adobe           | 2003 |   1097 |
| adobe           | 2004 |   1313 |
| adobe           | 2005 |   1701 |
```

```
| adobe           | 2006 |   2281 |
| adobe           | 2007 |   1994 |
| adobe           | 2008 |   2019 |
| adobe           | 2009 |   1904 |
| adobe           | 2010 |   2468 |
| adobe           | 2011 |   2912 |
| adobe           | 2012 |   3538 |
| adobe           | 2013 |   3174 |
| apple           | 2003 |   4566 |
| apple           | 2004 |   5464 |
| apple           | 2005 |   8261 |
| apple           | 2006 |  10110 |
| apple           | 2007 |  15386 |
| apple           | 2008 |  24490 |
| apple           | 2009 |  23464 |
| apple           | 2010 |  25620 |
| apple           | 2011 |  25952 |
| apple           | 2012 |  29129 |
| apple           | 2013 |  40546 |
| dell            | 2003 |   4638 |
| dell            | 2004 |   5405 |
…
```

The following figure illustrates how inner join works. A indicates matched records in both tables, while B and C are records in left and right tables, respectively, that are not matched. In the example above, LG is not listed because the table `balance` does not contain any record about LG.



Since inner join produces combinations of matched records, it will be less meaningful, if not useless, when matching attributes are not unique.

The following inner join (JOIN … ON …) produces the same result. Switching the order of firm and balance in the syntax does not make any difference.

```
mysql> SELECT firm.firm, balance.year, balance.value FROM firm JOIN balance ON
    -> firm.firm = balance.firm;
```

If the attribute to be compared has the same name in both tables, you may use the USING clause instead of ON.[3]

```
mysql> SELECT firm.firm, balance.year, balance.value FROM firm JOIN balance
    -> USING(firm);
```

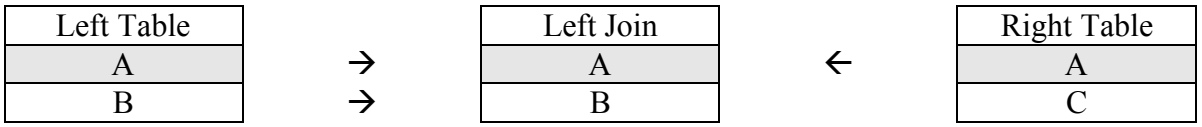
You may join more than two tables in a relational database.

```
mysql> SELECT … FROM table1, table2, table3, …
    -> WHERE (table1.key = … ) AND (table2.key = … ) AND … ;
```

11.3 Outer Join: Left Join

[3] Also you may explicitly use the NATURAL JOIN clause that returns only the records with matching values in the matching attributes. Accordingly, the attribute should have the same name and compatible type. This inner join determines common attributes and return only the records with common values in the common attributes.

Outer join is either left join or right join. A *left join* returns matched records as well as all unmatched records from the left table. All records in the left table appear regardless of the result of matching (See the following figure). Right attributes of unmatched records are left null (filled with the missing value). This join is frequently used in practice.



The syntax for left join is,

```
mysql> SELECT … FROM left_table LEFT JOIN right_table ON …;
```

The following left join displays company name, revenue, year, and value of balance item. from tables `firm` and `balance`. The common attribute is `firm_no` that is a primary key only in the table `firm`; `balance.firm_no` is a *foreign key*.

```
mysql> SELECT firm.firm, firm.revenue, balance.year, balance.value FROM firm
    -> LEFT JOIN balance ON firm.firm_no = balance.firm_no;
+-----------------+---------+------+--------+
| firm            | revenue | year | value  |
+-----------------+---------+------+--------+
| adobe           |       4 | 2003 |   1097 |
| adobe           |       4 | 2004 |   1313 |
| adobe           |       4 | 2005 |   1701 |
| adobe           |       4 | 2006 |   2281 |
| adobe           |       4 | 2007 |   1994 |
| adobe           |       4 | 2008 |   2019 |
| adobe           |       4 | 2009 |   1904 |
| adobe           |       4 | 2010 |   2468 |
| adobe           |       4 | 2011 |   2912 |
| adobe           |       4 | 2012 |   3538 |
| adobe           |       4 | 2013 |   3174 |
| apple           |     171 | 2003 |   4566 |
| apple           |     171 | 2004 |   5464 |
…
| verizon         |     121 | 2003 |  33466 |
| verizon         |     121 | 2004 |  37560 |
| verizon         |     121 | 2005 |  39680 |
| verizon         |     121 | 2006 |  48535 |
| verizon         |     121 | 2007 |  50581 |
| verizon         |     121 | 2008 |  41706 |
| verizon         |     121 | 2009 |  41606 |
| verizon         |     121 | 2010 |  38569 |
| verizon         |     121 | 2011 |  35970 |
| verizon         |     121 | 2012 |  33157 |
| verizon         |     121 | 2013 |  38836 |
| symantec        |       7 | NULL |   NULL |
| lg              |      45 | NULL |   NULL |
+-----------------+---------+------+--------+
```

Unlike the inner join, the left outer join include all records in table A no matter whether they are matched to table B. Therefore, the result of left join includes Symantec and LG, whose values in `balance` are NULL (missing).

You may add condition to choose a subset of records using the WHERE clause.

```
mysql> SELECT firm.firm, firm.revenue, balance.year, balance.value FROM firm
    -> LEFT JOIN balance ON firm.firm_no = balance.firm_no
```

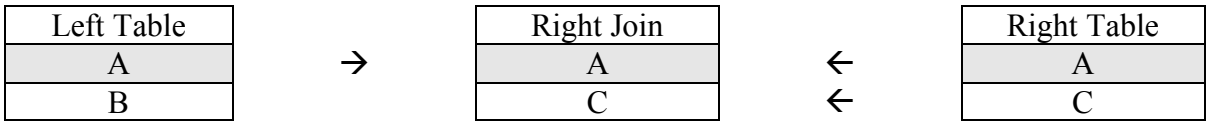
```
    -> WHERE balance.value >=70000;
+-----------+---------+------+--------+
| firm      | revenue | year | value  |
+-----------+---------+------+--------+
| microsoft |      78 | 2013 |  77022 |
| apple     |     171 | 2011 |  76615 |
| apple     |     171 | 2012 | 118210 |
| apple     |     171 | 2013 | 123549 |
| google    |      60 | 2012 |  71715 |
| google    |      60 | 2013 |  87309 |
| microsoft |      78 | 2004 |  74825 |
| microsoft |      78 | 2013 |  78944 |
+-----------+---------+------+--------+
```

The following example shows how to produce aggregate information using GROUP BY and HAVING clauses together.

```
mysql> SELECT firm.product, balance.year, COUNT(firm.firm), AVG(firm.revenue),
    -> SUM(balance.value) from balance RIGHT JOIN firm
    -> ON firm.firm_no = balance.firm_no
    -> GROUP BY firm.product, balance.year
    -> HAVING AVG(balance.value) >= 20000;
+------------------+------+------------------+------------------+--------------------+
| product          | year | count(firm.firm) | avg(firm.revenue) | sum(balance.value) |
+------------------+------+------------------+------------------+--------------------+
| computer         | 2006 |                6 |          127.6667 |             121444 |
| computer         | 2008 |                6 |          127.6667 |             139324 |
| computer         | 2009 |                6 |          127.6667 |             135327 |
| computer         | 2010 |                6 |          127.6667 |             161404 |
| computer         | 2011 |                6 |          127.6667 |             183932 |
| computer         | 2012 |                6 |          127.6667 |             213542 |
| computer         | 2013 |                6 |          127.6667 |             233516 |
| hardware         | 2010 |                4 |           55.0000 |              88111 |
| hardware         | 2011 |                4 |           55.0000 |              82904 |
| hardware         | 2012 |                4 |           55.0000 |              93100 |
| hardware         | 2013 |                4 |           55.0000 |             101800 |
| software         | 2003 |                6 |           39.6667 |             125105 |
| software         | 2004 |                6 |           39.6667 |             154735 |
| software         | 2010 |                6 |           39.6667 |             139890 |
| software         | 2011 |                6 |           39.6667 |             187174 |
| software         | 2012 |                6 |           39.6667 |             213970 |
| software         | 2013 |                6 |           39.6667 |             242729 |
| telecommunication | 2004 |                2 |          121.0000 |              42107 |
| telecommunication | 2005 |                2 |          121.0000 |              42954 |
| telecommunication | 2006 |                2 |          121.0000 |              54188 |
| telecommunication | 2007 |                2 |          121.0000 |              53978 |
| telecommunication | 2008 |                2 |          121.0000 |              51997 |
| telecommunication | 2009 |                2 |          121.0000 |              44105 |
| telecommunication | 2010 |                2 |          121.0000 |              45782 |
| telecommunication | 2011 |                2 |          121.0000 |              49924 |
| telecommunication | 2013 |                2 |          121.0000 |              92965 |
| Web              | 2008 |                2 |           60.0000 |              44085 |
| Web              | 2009 |                2 |           60.0000 |              60489 |
| Web              | 2010 |                2 |           60.0000 |              81216 |
| Web              | 2011 |                2 |           60.0000 |             102771 |
| Web              | 2012 |                2 |           60.0000 |             119803 |
| Web              | 2013 |                2 |           60.0000 |             146026 |
+------------------+------+------------------+------------------+--------------------+
```

11.4 Outer Join: Right Join

A *right join* returns matched records and also includes all unmatched records from the right table. All records chosen in the second (right) table appear regardless of the result of matching. Therefore, a right join is the opposite to a left join.



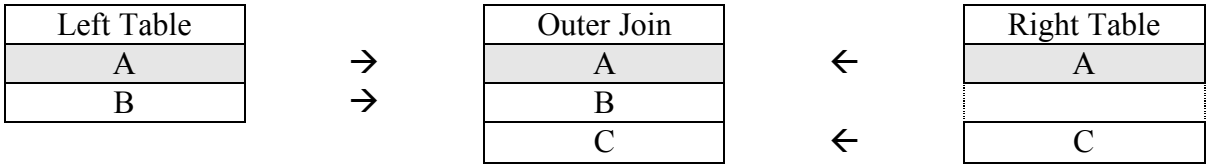
The left join in 11.3 is the same as the right join below.

```
mysql> SELECT firm.firm, firm.revenue, balance.year, balance.value FROM balance
    -> RIGHT JOIN firm ON firm.firm_no = balance.firm_no;
```

The choice between left join and right join, although the left join is recommended, is a matter of your taste.

11.5 Outer Join: Full Join

A *full join* returns all matched and unmatched records from both tables (See the following figure). Attributes of unmatched records are left null in both tables.



However, MySql does not appear to support this type of outer join although there is an indirect way of conducting the similar operation using LEFT JOIN, UNION, and RIGHT JOIN.

```
mysql> SELECT firm.firm, firm.revenue, balance.year, balance.value FROM firm
    -> LEFT JOIN balance ON firm.firm_no = balance.firm_no UNION
    -> SELECT firm.firm, firm.revenue, balance.year, balance.value FROM firm
    -> RIGHT JOIN balance ON firm.firm_no = balance.firm_no;
```

References

Coronel, Carlos, Steven Morris, and Peter Rob. 2013. *Database principles: Fundamentals of design, implementation, and management*. 10th ed. Cengage Learning EMEA. ISBN 978-1133311973.

SAS. 2011. *SAS 9.3 SQL procedure: User's guide*. Cary, NC: SAS Institute.