BLG 317E
DATABASE SYSTEMS

WEEK 3

Slides credit:
S. Shivakumar, CS 145, Stanford University
Database System Concepts, Silberschatz, Korth, Sudarshan

ITU

Structured Query Language (SQL)

Structured Query Language

- **SQL is the standard language for querying and manipulating data in relational databases.**
- **SQL is a declarative language, it is not a general purpose programming language.**
- It was originally called as SEQUEL (Structured English Query Language).
- SQL commands only declare the required data manipulation operations, but not how to implement the operations.
- Standard SQL does not have If commands, or loop commands. But, most DBMSs have procedural language extensions, specific to vendor.
- Current SQL standard is SQL-2023 ANSI / ISO. (International Standards Organization).

SQL Command Categories

The followings are SQL command categories.

- Data Definition Language (DDL): CREATE, DROP, ALTER, etc. These commands are used to create and modify database schema and tables.
- Data Manipulation Language (DML): INSERT, UPDATE, DELETE. These commands are used to manipulate data stored in tables.
  - Data Query Language (DQL): SELECT. Used for querying (selecting / retrieving) a subset of data from a database.
- Data Control Language (DCL): GRANT, REVOKE, etc. Used for controlling access to data within a database, commonly used for granting user privileges.
- Transaction Control Language (TCL): COMMIT, ROLLBACK, etc. Used for managing groups of statements as a unit of work.

Database CRUD Operations

- CRUD is a commonly used acronym for database table operations.
- CRUD = Create , Read , Update, Delete
- The followings are the SQL equivalent commands of CRUD operations.

Operation	SQL command
Create	CREATE, INSERT
Read	SELECT
Update	UPDATE
Delete	DELETE

Modification of the Database (SQL DML)

- Deletion of tuples from a given relation.

```
DELETE FROM <tablename>  WHERE <condition>
```

Deletes records from a table, that are selected by given condition criteria. If no condition is specied, all tuples will be deleted.

- Insertion of new tuples into a given relation

```
INSERT INTO <tablename>  VALUES (<field values>)
```

Adds a new record to a table, with specified data.
Order of values must match order of columns.

- Updating of values in some tuples in a given relation

```
UPDATE <tablename>  SET <field name> = <new value>
   WHERE <condition>
```

Updates records in a table with new data, selected by given condition criteria. If no condition is specied, all tuples will be updated.

Deletion

- Delete all instructors

  delete from *instructor*

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= 'Finance';

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

  delete from *instructor*
  where *dept name* **in** (**select** *dept name*
  from *department*
  where *building* = 'Watson');

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

  delete from *instructor*
  where *salary* < (**select avg** (*salary*)
  from *instructor*);

  - Problem: as we delete tuples from deposit, the average salary changes
  - Solution used in SQL:
    1. First, compute **avg** (salary) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Commands for Deleting all records from Table

- The following two commands are equivalent.
- They delete all data records from an existing table.
- But they do not remove the table schema definition.

```
TRUNCATE TABLE <tablename>

DELETE FROM <tablename>
```

Insertion

- Add a new tuple to *course*

  insert into *course*
  values (‘BLG 317E', 'Database Systems', 'Comp. Eng.', 3);

- or equivalently

  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values (‘BLG 317E', 'Database Systems', 'Comp. Eng.', 3);

- Add a new tuple to *student* with *tot_creds* set to null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Updates

Change phone data content of an existing record.

The UPDATE command below changes the phone number of the EMPLOYEE whose ID number is 500.

```
update EMPLOYEE
set    phone = '2127001234'
where  emp_ID = 500;
```

Updates (Cont.)

- Give a 5% salary raise to all instructors

```
update instructor
set salary = salary * 1.05;
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
set salary = salary * 1.05
where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                 from instructor);
```

Updates (Cont.)

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;

update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the **case** statement (next slide)

Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
    set salary = case
                    when salary <= 100000 then salary * 1.05
                    else salary * 1.03
                 end
```

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Database Integrity

- Database Integrity refers to the validity and correctness of all data.
- Client application programs (C/C++, Java, Python, etc.) can implement some integrity constraints.
- They are applied to common database CRUD operations (Create, Read, Update, Delete).
  - IF-ELSE statements are used.
  - THROW-CATCH exception statements are used.
- In addition to the above methods, integrity constraints can be implemented at the database-level also.
- The DBMS can control automatically all constraints, which were previously declared by the CREATE TABLE command.

CREATE TABLE Command

Syntax1

```
CREATE TABLE table_name (
    column1  datatype  constraint1,
    column2  datatype  constraint2,
    column3  datatype  constraint3,
    ....
);
```

Syntax2

```
CREATE TABLE table_name (
    column1  datatype,
    column2  datatype,
    column3  datatype,
    ....
    (constraint1),
    (constraint2),
    (constraint3),
    ....
```

Basic Data Types in SQL

Category	Data Type	Description
String	`CHAR(n)`	Fixed length character (string) (max n)
	`VARCHAR(n)`	Variable length character (max n)
	`TEXT`	Holds a string with a maximum length of 64KB bytes
Number	`INT, INTEGER`	Integer between ~ -2 billion to ~ +2 billion
	`SMALLINT`	Between -32768 to +32767
	`NUMERIC(size, d)` `DECIMAL(size, d)`	Fixed point number. size : Total number of digits d : Number of digits after decimal point)
	`REAL`	Floating point number
	`FLOAT(p)`	A floating point number. The p value determines whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, data type becomes FLOAT(). If p is from 25 to 53, data type becomes DOUBLE().

Basic Data Types in SQL

Category	Data Type	Description
Number	`DOUBLE(size, d)`	Floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.
	`BOOL, BOOLEAN`	Zero is considered as false, nonzero values are considered as true.
Date and Time	`DATE`	Year, Month, Day (“yyyy-mm-dd”)
	`TIME`	Hour, Minute, Second (“hh:mm:ss”)
	`DATETIME, TIMESTAMP`	Both (“yyyy-mm-dd hh:mm:ss”)
Large Data	`BLOB`	Binary Large Object. Binary files such as picture, audio, document, etc. can be stored in a table. Holds up to 64KB of data. Usage is not recommended.
	`CLOB`	Character Large Object

Integrity Constraints in CREATE TABLE command

- The following integrity constrains can be declared for each column of a table, when writing the CREATE TABLE command.
- By default, a column (field) value can be null.
- Declaring a field with NOT NULL constraint prevents null values.

  - **NOT NULL**
  - **DEFAULT <value>**
  - **UNIQUE**
  - **PRIMARY KEY**
  - **FOREIGN KEY**
  - **CHECK (condition)**
  - **AUTO_INCREMENT**

Automatic Control of Integrity Constraints

- When a user issues an INSERT, DELETE, or UPDATE command, the DBMS automatically checks all Integrity Constraints.
- If any of the declared constraints fails, then the issued command fails (INSERT/DELETE/UPDATE).
- The PRIMARY KEY declaration on an attribute (column) automatically ensures the NOT NULL and UNIQUE.
- Primary Key attributes cannot be NULL,
  but Foreign Key attributes can be NULL.

Example: Declaring Integrity Constraints

```
create table COURSE
        (course_id          varchar(10),
         course_title       varchar(30) NOT NULL,
         dept_id            varchar(5)  DEFAULT 'MAT',
         credits            smallint    CHECK (credits >= 0),
         PRIMARY KEY (course_id),
         FOREIGN KEY (dept_id)
           REFERENCES department (dept_id)
           ON DELETE SET NULL
           ON UPDATE CASCADE
        );
```

NOT NULL : The course_title can not be specified as null in an INSERT command. Otherwise DBMS returns error.

DEFAULT : When dept_id value is not specified in an INSERT command, then it will be automatically set to 'MAT' string by DBMS.

CHECK : When credits field value is negative in an INSERT command, DBMS returns error.

REFERENTIAL INTEGRITY CONSTRAINTS ON FOREIGN KEY:

ON DELETE SET NULL : When a department is deleted from the DEPARTMENT table, all corresponding foreign keys for that dept_id in the COURSE table will be automatically set to null by DBMS.

ON UPDATE CASCADE : When a dep_id is changed in the DEPARTMENT table, all corresponding foreign keys for that dept_id in the COURSE table will be automatically changed to new value by DBMS.

Examples: Constraints in INSERT commands

- **Example1:** Add a new record to COURSE table, with the 'MAT' value as department_id. All integrity constraints are successful.

```
insert into COURSE values
  ('MAT-101', ‘Calculus', 'MAT', 4);
```

- **Example2:** Command returns an error.
  Because the course_title field should not be null.

```
insert into COURSE values
   ('MAT-202', null, 'MAT', 4);
```

```
...
course_title      varchar(30) NOT NULL,
...
```

Example3: Command returns an error.
The CHECK (credits >= 0) constraint fails.

```
insert into COURSE values
  ('MAT-303', 'Statistical Methods', 'MAT', -7 );
```

```
...
credits smallint CHECK (credits >= 0)
...
```

Example4:

- Add a new record to COURSE table, without specifying the **department_id** value.
- According to the integration constraint, DBMS automatically assigns **'MAT'** to the **department_id** value in COURSE table.
- All integrity constraints are successful.

```
...
dept_id varchar(5) DEFAULT 'MAT'
...
```

```
insert into COURSE (course_id, course_title, credits)
values ('MAT-404', 'Lineer Cebir', 4 );
```

Example: Declaring a Primary Key with AUTO_INCREMENT

- The Employee Identification number field is declared as AUTO_INCREMENT (in MySQL).
- Default starting value is 1.
- Automatic increment will be performed sequentially by DBMS, every time an INSERT command is used.

```
create table EMPLOYEE
  (emp_ID int AUTO_INCREMENT,
   firstname varchar(10),
   lastname  varchar(10),
   PRIMARY KEY (emp_ID) );
```

Example: Declaring a Primary Key with AUTO_INCREMENT

- The ALTER command below changes the table.
- The initial value will start from 500 (overriding the default 1).

```
ALTER TABLE EMPLOYEE AUTO_INCREMENT=500;
```

- **PostgreSQL** DBMS does not have AUTO_INCREMENT data type.
- Instead, it has a similar data type called **SERIAL.**

```
create table EMPLOYEE (emp_ID SERIAL,
                       firstname varchar(10),
                       lastname  varchar(10),
                       PRIMARY KEY (emp_ID) );

ALTER SEQUENCE  EMPLOYEE_emp_id_seq  RESTART WITH 500;
```

Examples: Constraints in INSERT commands

Example1: For commands below, DBMS assigns 500, 501, and 502 for the Employee Identification numbers.

```
insert into EMPLOYEE (firstname, lastname)
  values ('Harvey','Hudson');

insert into EMPLOYEE (firstname, lastname)
   values ('Ralph', 'Davis');

insert into EMPLOYEE
   values (null, 'Mike', 'Williams');
```

Example2: The command fails.

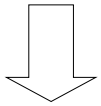
- SQL error message : “Column count does not match”.
- For auto-increment fields, the above INSERT syntax should be used.

```
insert into EMPLOYEE
  values ('James', 'Hopkins');
```

Checking the Table Content

- SELECT command below retrieves all records in EMPLOYEE table.
- Asterisk (*) symbol means select all columns in table.
- Employee numbers (500, 501, 502, etc) were automatically given by DBMS.

```
select *
from  EMPLOYEE;
```



Result-set of SELECT query

```
+--------+-----------+----------+
| emp_ID | firstname | lastname |
+--------+-----------+----------+
|    500 | Harvey    | Hudson   |
|    501 | Ralph     | Davis    |
|    502 | Mike      | Williams |
+--------+-----------+----------+
3 rows in set (0.00 sec)
```

ALTER TABLE command

EXAMPLE1: Add a new column to a table.

- The ALTER command below adds PHONE field as a new column to EMPLOYEE table.
- For all existing records in EMPLOYEE table, the new column is assigned null automatically.

```
alter table EMPLOYEE
    add phone char(10);
```

ALTER TABLE command

EXAMPLE2: Change data-type of a column.

ALTER command below modifies (changes) the data type of the PHONE column in EMPLOYEE table, from char(10) to varchar(15).

```
alter table EMPLOYEE
      modify column phone varchar(15);
```

ALTER TABLE command

EXAMPLE3: Remove a column from a table.

The ALTER command below drops (deletes) the PHONE column from EMPLOYEE table.

```
alter table EMPLOYEE
      drop column  phone;
```

ALTER TABLE command

EXAMPLE4: Add a new column to a table, and also declare its constraints.

- The expected format of email is declared by using the LIKE operator.
- The string length of email should be at least 5. The LENGTH() is a built-in SQL function.
- When an INSERT command is issued, DBMS controls all of the declared contraints, if all of them are True, then the command is successful.

```
alter table EMPLOYEE
        ADD email varchar(20)
        NOT NULL
        UNIQUE
        CHECK (email  LIKE  '%@%'  AND
               LENGTH(email)  >=  5) ;
```

Examples: Constraints in INSERT commands

Example1: Command is succesfull.

```
insert into EMPLOYEE (firstname, lastname, email)
                values ('Tom', 'Walter', 'twalter@com');

Query OK, 1 row affected (0.40 sec)
```

Example2: Command failed. The @ symbol is required.

```
insert into EMPLOYEE (firstname, lastname, email)
                values ('Tom', 'Walter', 'twalter');

ERROR 3819 (HY000):
Check constraint 'employee_chk_1' is violated.
```

Examples: Constraints in INSERT commands

Example3: Command failed. String length must be at least 5.

```
insert into EMPLOYEE (firstname, lastname, email)
                values ('Tom', 'Walter', 't@w');
```

```
ERROR 3819 (HY000): Check constraint
'employee_chk_1' is violated.
```

Example: CREATE TABLE command

```
create table SALES
(
  sale_id  int AUTO_INCREMENT,

  sale_date_and_time  datetime
         DEFAULT  CURRENT_TIMESTAMP(),

  product_name varchar(40),

  PRIMARY KEY (sale_id)
);
```

DATETIME data-type and CURRENT_TIMESTAMP() Function

The CREATE command below declares the following constraints:

- The sale_id field is primary key and auto-incremented.
- The sale_date_and_time field has a data-type of datetime.
- It will be assigned by DBMS by default, with with the current system date and time.
- The CURRENT_TIMESTAMP() is a built-in SQL function. (Parantheses are optional.)

Example: INSERT command

Adding new data records to SALES table:

- The following INSERT commands add new records to the SALES table.
- Only the name of a product is specified in the INSERT command.
- Other field values (the sale_id primary key field and the sale_date_and_time field) are assigned automatically by DBMS.

```
insert into
  SALES (product_name) values
    ('Notebook'),
    ('Pencil'),
    ('Eraser'),
    ('Coloring Pens');
```

Example: SELECT command

SELECT query below retrieves all records in SALES table.

```
select * from SALES;
```

Result-set

sale_id	sale_date_and_time	product_name
1	2023-09-15 10:47:30	Notebook
2	2023-09-15 10:47:31	Pen
3	2023-09-15 10:47:32	Eraser
4	2023-09-15 10:47:33	Coloring Pens

The sale_id and sale_date_and_time fields are assigned automatically by DBMS.

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Foreign Key Constraints (Referential Integrity Constraints)

- Before deleting a record from a main table, it is always necessary to check whether there exists a reference to the record in question in secondary tables.
- This checking can be done automatically by DBMS, if a foreign key was defined.
- **Referential Integrity constraints** can be declared for **Foreign Keys**, when writing the CREATE TABLE command.

Main Table

Primary Key	Other Columns

Secondary Table

Foreign Key	Other Columns

Foreign Key Constraints

- The following is the general syntax of CREATE TABLE command, for defining foreign keys and their referential integrity constraints.

Syntax

```
CREATE TABLE table_name1 (

    column_definitions,

    ....

        FOREIGN KEY (foreign_key_id)

            REFERENCES table_name2 (primary_key_id)

            ON DELETE <constraint_option>

            ON UPDATE <constraint_option>

);
```

Foreign Key Constraints

- ON DELETE means whenever a record in **main table** with the primary key is deleted.
- ON UPDATE means whenever the primary key of a record in **main table** is updated.

Constraint Option	Action Description (ON DELETE, ON UPDATE)
`RESTRICT, NO ACTION`	Don't allow delete / update on records in main table.
`CASCADE`	Reflect the change to affected records in secondary table.
`SET NULL`	Assign null value to foreign key in secondary table.
`SET DEFAULT`	Assign default value to foreign key in secondary table.

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Example: Two Related Tables

- The following two tables are related over a common key field.
- The Primary key (category_id) of the CATEGORY table is also used as a foreign key in the PRODUCT table.

CATEGORY table

category_id	category_name
1	Electronics
2	Furniture

PRODUCT table

product_id	product_name	category_id
10	Desk	2
20	Chair	2
30	TV	1
40	Refrigirator	1

Example: Create category table

```
CREATE TABLE category (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(15)
);
```

```
INSERT INTO category VALUES
(1, 'Electronics'),
(2, 'Furniture');
```

Foreign Key with (ON DELETE RESTRICT) constraint (default)

- The “ON DELETE RESTRICT” is a defult constraint.
- It works even if not written.
- When user tries to delete a record from CATEGORY table, DBMS first checks for existing corresponding foreign keys in PRODUCT table. If there are any matching records in PRODUCT table, then delete operation a record from CATEGORY table fails.

```
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(20),
    category_id INT,
    CONSTRAINT FOREIGN KEY (category_id)
               REFERENCES category(category_id)
               ON DELETE RESTRICT    <--Default
);
```

Example: Populate Product Table

```
INSERT INTO product VALUES
(10, 'Desk', 2),
(20, 'Chair', 2),
(30, 'TV', 1),
(40, 'Refrigirator', 1);
```

Example: INSERT command error (Foreign Key reference without Primary Key)

- When calling an INSERT command for a secondary table with a foreign key, the corresponding primary key must already exist in the main table.
- The following command fails, because there is no corresponding primary key (category_id = 3) in the CATEGORY table, for the foreign key specified (category_id = 3)

```
INSERT INTO product VALUES
  (50, 'Bicycle', 3);
```

Result

ERROR : Cannot add or update a child row: a foreign key constraint fails (product, CONSTRAINT FOREIGN KEY (category_id) REFERENCES category (category_id))

Example: DELETE command error (Primary Key referenced by Foreign Keys)

- The command below fails, because there are some records in PRODUCT table whose foreign keys are (category_id = 1).
- Firsltly, user should delete all the records with (category_id = 1) from PRODUCT table.
- Then, user can delete the record with (category_id = 1) from CATEGORY table.

```
DELETE FROM category
WHERE category_id = 1;
```

Result

ERROR : Cannot delete or update a parent row: a foreign key constraint fails (**product**, CONSTRAINT FOREIGN KEY (category_id) REFERENCES category(category_id))

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Foreign Key with (ON DELETE SET NULL) constraint

- In the example below, the foreign key (category_id) in the PRODUCT table is defined with “ON DELETE SET NULL” constraint.
- If a record with from CATEGORY table is deleted, then DBMS will automatically set NULL all corresponding foreign keys (category_id) in the PRODUCT table.

```
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(20),
    category_id INT,
    CONSTRAINT FOREIGN KEY (category_id)
               REFERENCES category(category_id)
               ON DELETE SET NULL
);
```

Data contents of tables before the delete operation on CATEGORY table:

Query

```
SELECT * FROM category;
```

Result-set of Query

category_id	category_name
1	Electronics
2	Furniture

Query

```
SELECT * FROM product;
```

Result-set of Query

product_id	product_name	category_id
10	Desk	2
20	Chair	2
30	TV	1
40	Refrigirator	1

Data contents of tables after the delete operation on CATEGORY table:

Delete Operation

```
DELETE FROM category
WHERE category_id = 1;
```

- The "Electronics" category (category_id = 1) is deleted from CATEGORY table.
- Also all foreign keys (with category_id = 1) in PRODUCT table are automatically changed to NULL, because of the defined constraint.

Query

```
SELECT * FROM category;
```

Result

category_id	category_name
2	Furniture

Query

```
SELECT * FROM product;
```

Result

product_id	product_name	category_id
10	Desk	2
20	Chair	2
30	TV	NULL
40	Refrigirator	NULL

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Foreign Key with (ON DELETE CASCADE) constraint

- Here the foreign key (category_id) of PRODUCT table is defined with “ON DELETE CASCADE” constraint.
- If a record with from CATEGORY table is deleted, then DBMS will also automatically delete all corresponding records (whose foreign keys are matching the primary key) from PRODUCT table.

```
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(20),
    category_id INT,
    CONSTRAINT FOREIGN KEY (category_id)
               REFERENCES category(category_id)
               ON DELETE CASCADE
);
```

Data contents of tables after the delete operation on CATEGORY table:

Delete Operation

```
DELETE FROM category
WHERE category_id = 1;
```

Query

```
SELECT * FROM category;
```

Result

category_id	category_name
2	Furniture

Query

```
SELECT * FROM product;
```

Result

product_id	product_name	category_id
10	Desk	2
20	Chair	2

- "Electronics" category is deleted from CATEGORY table.
- Also product records whose category "Electronics" are automatically deleted from PRODUCT table.

Topics

- Integrity Constraints
- Foreign Key Constraints
  - ON DELETE RESTRICT
  - ON DELETE NULL
  - ON DELETE CASCADE
- Example Database
- SQL SELECT Command

Example: UNIVERSITY Database

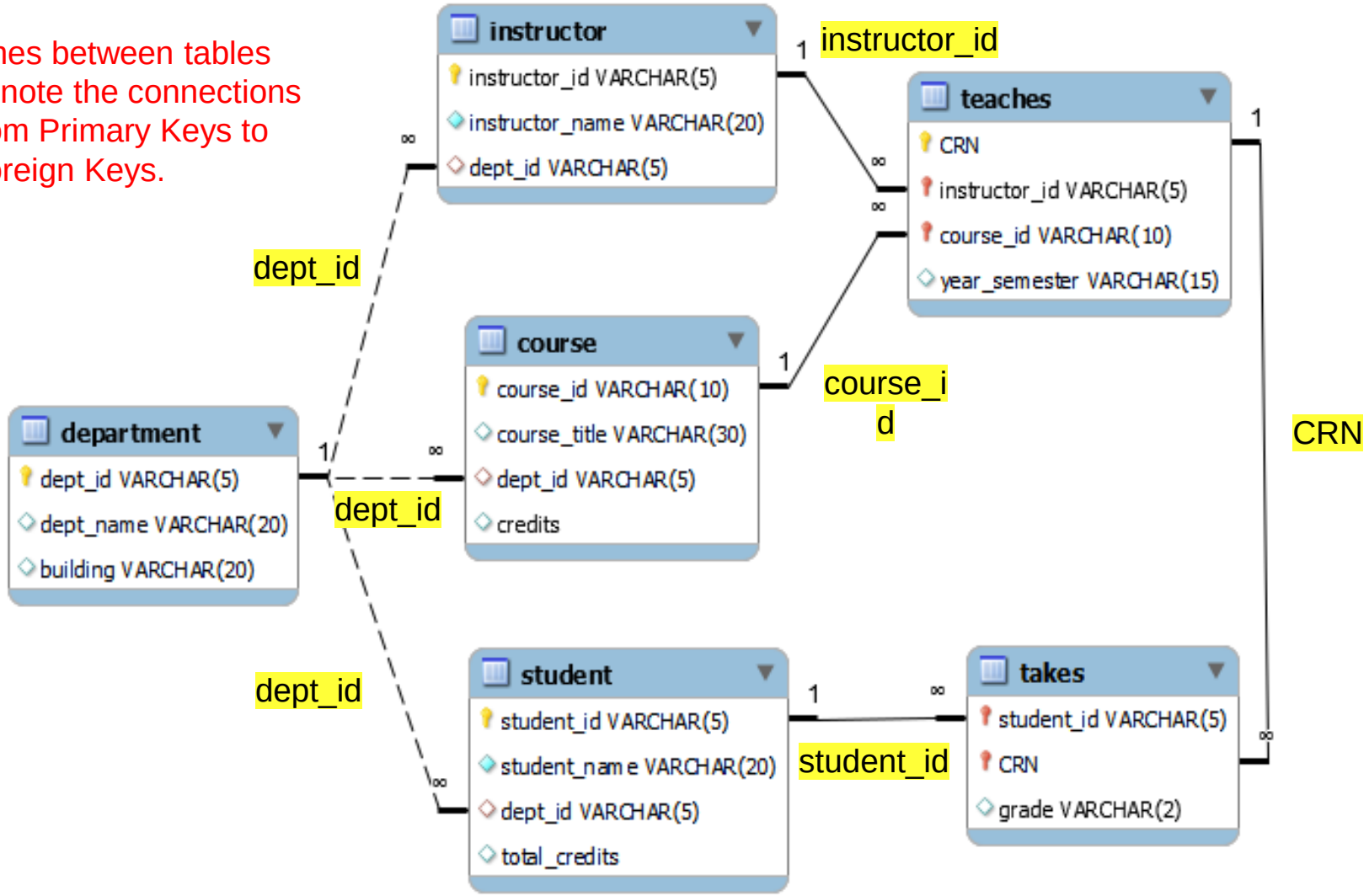
- Main tables : DEPARTMENT, COURSE, INSTRUCTOR, STUDENT
- Secondary tables : TEACHES, TAKES
- Primary Keys are underscored.

DB Schema

```
DEPARTMENT (dept_id,  dept_name, building)
COURSE     (course_id, course_title, dept_id, credits)
INSTRUCTOR (instructor_id, instructor_name, dept_id)
STUDENT    (student_id        , student_name, dept_id,
total_credits)
TEACHES    (CRN, instructor_id, course_id, year_semester)
TAKES      (student_id, CRN, grade)
```

Schema Diagram of UNIVERSITY Database

Lines between tables denote the connections from Primary Keys to Foreign Keys.



CREATE / DROP DATABASE Commands

- **CREATE DATABASE <databasename>**
  Creates an empty database on DBMS server computer.

- **USE <databasename>**
  Tells the DBMS the name of database to be used.
  (If not specified, DBMS uses the master database as default.)

- **DROP DATABASE <databasename>**
  Deletes an existing database schema definition and all of its data.
  (All table definitions and their data are deleted.)

SQL Script File for Table Create commands

university_schema.sql File

Part 1

```
create database UNIVERSITY;

use university;

create table DEPARTMENT
 (dept_id    varchar(5),
  dept_name  varchar(20),
  building   varchar(20),
  primary key (dept_id)
 );

create table COURSE
 (course_id     varchar(10),
  course_title  varchar(30),
  dept_id       varchar(5),
  credits       smallint CHECK (credits >= 0),
  primary key (course_id),
  foreign key (dept_id) references department (dept_id)
     on delete set null
 );
```

SQL Script File for Table Create commands

Part 2

```
create table INSTRUCTOR
 (instructor_id     varchar(5),
  instructor_name   varchar(20) not null,
  dept_id           varchar(5),
  primary key (instructor_id),
  foreign key (dept_id) references department (dept_id)
    on delete set null
 );

create table TEACHES
 (CRN int UNIQUE,     -- CRN : Course Registration Number
  instructor_id    varchar(5),
  course_id        varchar(10),
  year_semester    varchar(15),
  primary key (CRN, instructor_id, course_id),
  foreign key (course_id) references course (course_id)
     on delete cascade,
  foreign key (instructor_id) references instructor (instructor_id)
     on delete cascade
 );
```

SQL Script File for Table Create commands

Part 3

```
create table STUDENT
 (student_id      varchar(5),
  student_name    varchar(20) not null,
  dept_id         varchar(5),
  total_credits   smallint CHECK (total_credits >= 0),
  primary key (student_id),
  foreign key (dept_id) references department (dept_id)
     on delete set null
 );

create table TAKES
 (student_id   varchar(5),
  CRN          int,
  grade        varchar(2),
  primary key (student_id, CRN),
  foreign key (CRN) references teaches (CRN)
     on delete cascade,
  foreign key (student_id) references student (student_id)
     on delete cascade
 );
```

Tables with Data

DEPARTMENT

dept_id	dept_name	building
BIO	Biology	Watson Bld.
CS	Computer Science	Taylor Bld.
EE	Electronics Eng.	Taylor Bld.
FIN	Finance	Painter Bld.
HIS	History	Painter Bld.
MU	Music	Packard Bld.
PHY	Physics	Watson Bld.
SPR	Sports	NULL

COURSE

course_id	course_title	dept_id	credits
BIO-101	Intro. to Biology	BIO	4
BIO-301	Molecular Biology	BIO	4
BIO-401	Biology Lab	BIO	3
CS-101	Intro. to Algorithms	CS	4
CS-190	Web Design	CS	4
CS-315	Robotics	CS	3
CS-319	Image Processing	CS	3
CS-347	Database System Concepts	CS	3
EE-184	Intro. to Digital Systems	EE	3
EE-208	Circuit Analysis	EE	5
EE-306	Semiconductors	EE	2
FIN-201	Accounting Methods	FIN	3
HIS-351	World History	HIS	3
MU-199	Music Production	MU	3
PHY-101	Optics Principles	PHY	4
SPR-202	Swimming	SPR	1

INSTRUCTOR

instructor_id	instructor_name	dept_id
10101	Murphy	CS
10203	Harrison	SPR
12121	Marsh	FIN
15151	Jones	MU
22222	Smith	PHY
32343	Castillo	HIS
33456	Collins	PHY
45565	Hernandez	CS
58583	Patterson	HIS
76543	Brown	FIN
76766	Thompson	BIO
83821	Fixter	CS
98345	Smith	EE

TEACHES

CRN	instructor_id	course_id	year_semester
1	10101	CS-101	2022-Spring
2	10101	CS-315	2022-Spring
3	10101	CS-347	2022-Spring
4	12121	FIN-201	2022-Spring
5	15151	MU-199	2022-Spring
6	22222	PHY-101	2022-Spring
7	32343	HIS-351	2023-Fall
8	45565	CS-101	2023-Fall
9	45565	CS-319	2023-Fall
10	76766	BIO-101	2023-Fall
11	76766	BIO-301	2023-Fall
12	83821	CS-190	2023-Fall
13	83821	CS-319	2023-Fall
14	98345	EE-184	2023-Fall

STUDENT

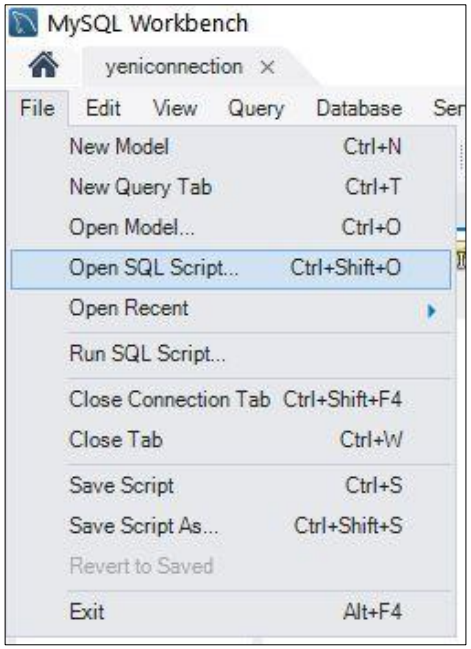
student_id	student_name	dept_id	total_credits
12345	Patel	CS	32
12860	Clark	CS	102
19991	Brown	HIS	80
23121	Chavez	FIN	110
44553	Young	PHY	56
45678	Walker	PHY	46
54321	Williams	CS	54
55739	Sanchez	MU	38
70557	Snow	PHY	0
76543	Brown	CS	58
76653	Hudson	EE	60
98765	Blake	EE	98
98988	Bradley	BIO	120

TAKES

student_id	CRN	grade
12345	3	C
12345	5	A
12345	6	A
12345	9	A
12860	3	A
12860	9	A-
19991	12	B
23121	11	C+
44553	14	B-
45678	3	C
45678	7	B
54321	4	A-
54321	5	B+
55739	13	A-
76543	4	A
76543	7	A
76653	10	C
98765	4	C-
98765	6	B
98988	1	A
98988	2	B

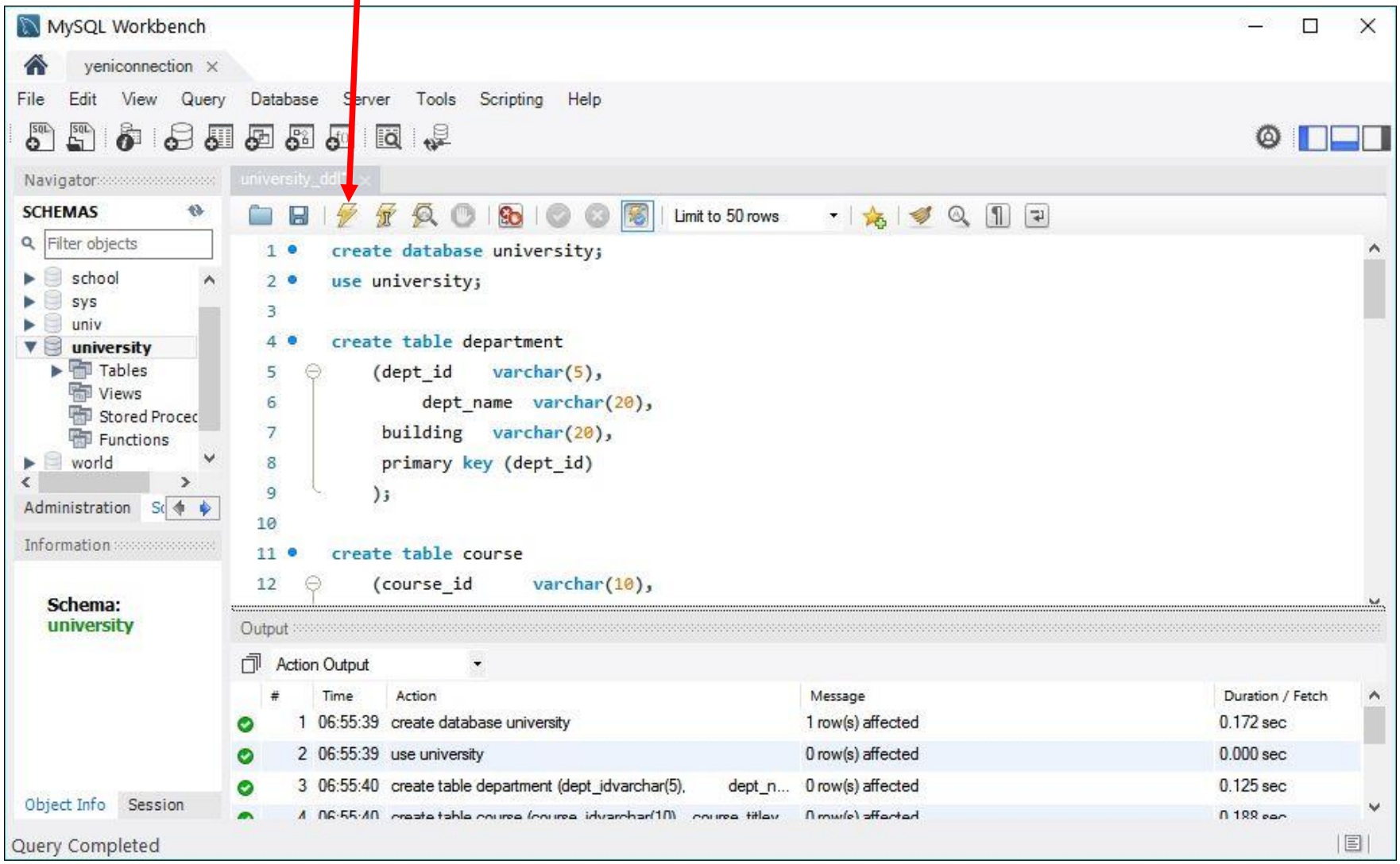
Using MySQL Workbench Program to execute SQL script files

- There are two separate SQL script files.
- First, execute the (university_schema.sql) file. (Contains CREATE statements.)
- Then, execute the (university_data.sql) file. (Contains INSERT statements.)
- From the File menu, click “Open SQL Script”.
- Open the SQL script file named university_schema.sql to query editor window.



Executing the (university_schema.sql) Script File

Click the Execute button



SQL DDL SUMMARY

CREATE / DROP TABLE Commands

- **CREATE TABLE** <tablename> (<field1> <datatype1>,
  <field2> <datatype2>, …)
  Creates an empty table, with specified fields (columns) and data types.

- **DROP TABLE** <tablename>
  Deletes an existing table schema definition and all of its data.

ALTER TABLE Commands

The followings are most used ALTER TABLE sub-commands : ADD, DROP, RENAME, MODIFY

- **`ALTER TABLE <tablename> ADD <newfield> <datatype>`**
  Adds a new column to an existing table.

- **`ALTER TABLE <tablename> DROP COLUMN <field>`**
  Removes a column from an existing table.

- **`ALTER TABLE <tablename> RENAME <newtablename>`**
  Renames an existing table.

- **`ALTER TABLE <tablename> MODIFY <field> <newdatatype>`**
  Changes the data type of an existing column.