CSc 484
Database Management Systems

Ken Gamradt

Spring 2024

Set Operations – Database Modifications

Set operations in SQL

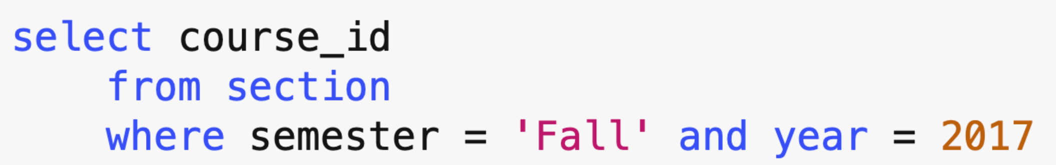
- **UNION**, **INTERSECT**, **EXCEPT** can be used to combine the results of two or more queries into a single result table
  - **UNION** of two tables A and B, is a table containing all rows that are either in A or B or both
  - **INTERSECT** of two tables A and B, is a table containing all rows that are common to A and B
  - **EXCEPT** of two tables A and B, is a table containing all rows that are in A but not in B

Set operations in SQL

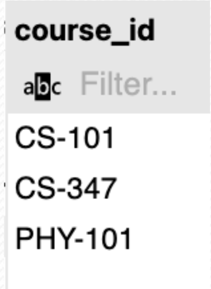
- Restrictions on using set operations
  - Two tables must contain the same number of columns
    - Their corresponding columns must have the same data types and length
  - It’s user’s responsibility to ensure that the data values in corresponding columns come from the same domain
    - E.g.: instructor’s name and building name

Set operations in SQL

- The list of all courses taught in Fall 2017

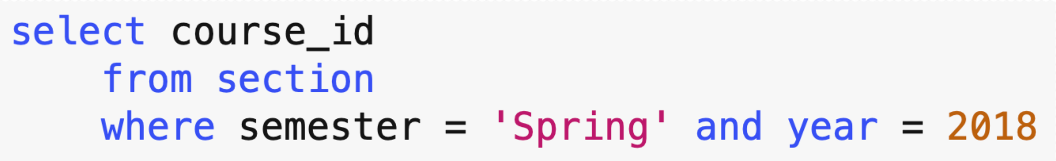


```
select course_id
    from section
    where semester = 'Fall' and year = 2017
```

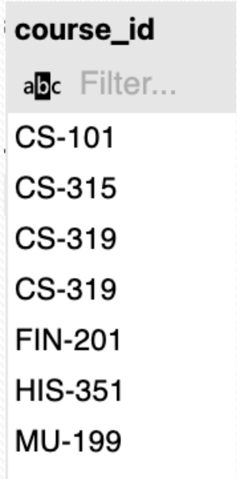


course_id
CS-101
CS-347
PHY-101

- The list of all courses taught in Spring 2018



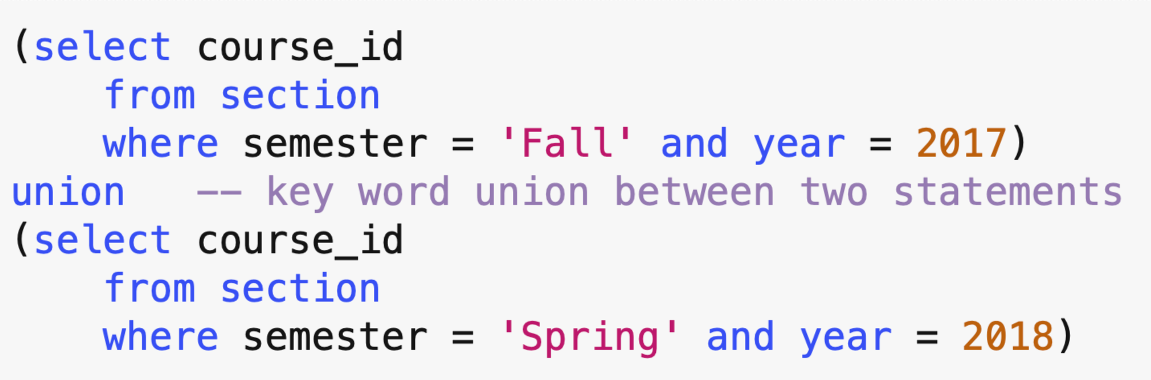
```
select course_id
    from section
    where semester = 'Spring' and year = 2018
```



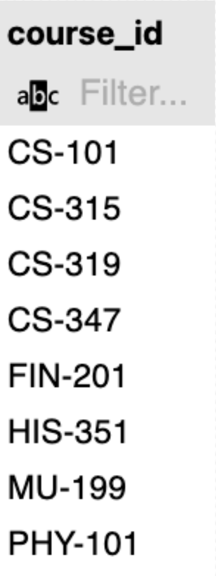
course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

Set operations in SQL – Union

- **UNION** of two tables A and B, is a table containing all rows that are either in A or B or both
- E.g., find all courses that taught either in Fall 2017 or in Spring 2018



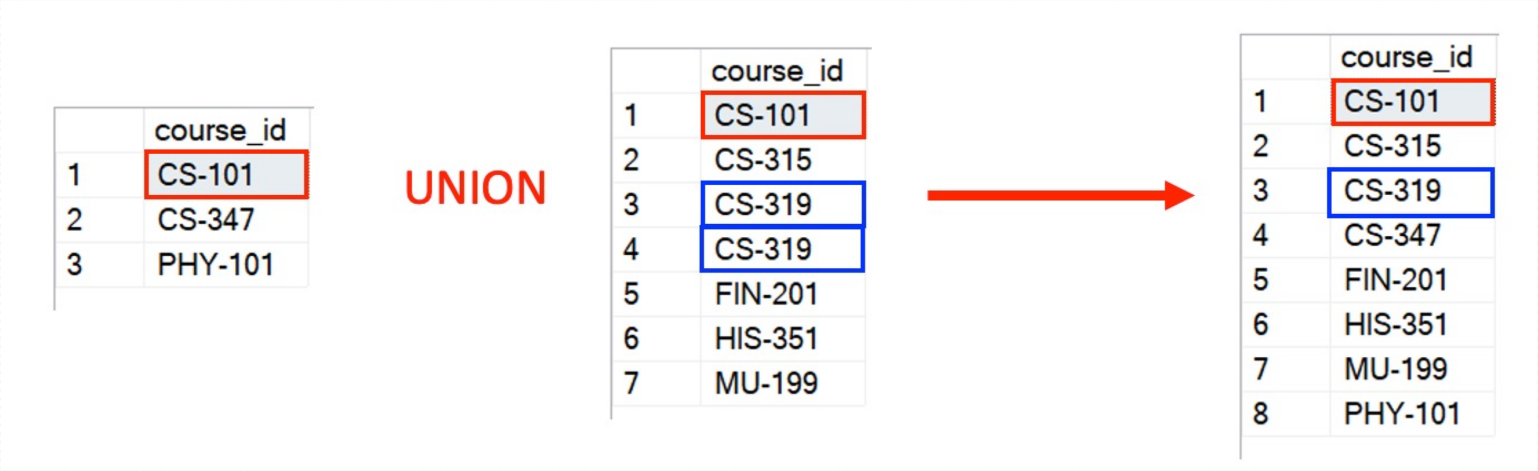
```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
union   -- key word union between two statements
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```



course_id
abc Filter...
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set operations in SQL – Union

- The **UNION** operation automatically eliminates duplicates



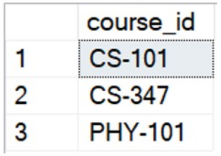
	course_id
1	CS-101
2	CS-347
3	PHY-101

	course_id
1	CS-101
2	CS-315
3	CS-319
4	CS-319
5	FIN-201
6	HIS-351
7	MU-199

	course_id
1	CS-101
2	CS-315
3	CS-319
4	CS-347
5	FIN-201
6	HIS-351
7	MU-199
8	PHY-101

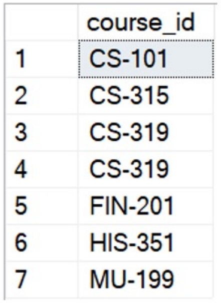
Set operations in SQL – Union All

- Use **UNION ALL** instead of **UNION** to retain all duplicates

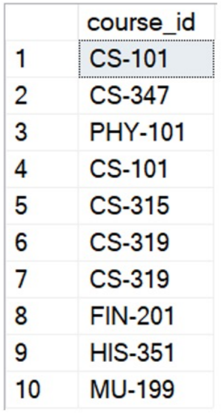


	course_id
1	CS-101
2	CS-347
3	PHY-101

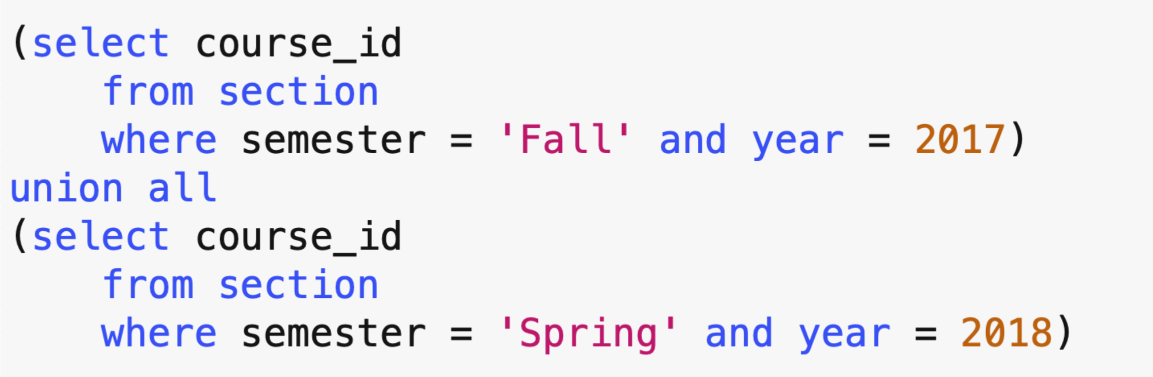
UNION ALL



	course_id
1	CS-101
2	CS-315
3	CS-319
4	CS-319
5	FIN-201
6	HIS-351
7	MU-199



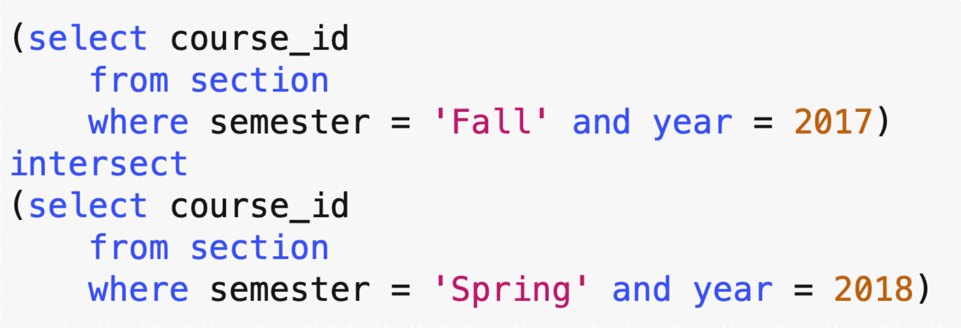
	course_id
1	CS-101
2	CS-347
3	PHY-101
4	CS-101
5	CS-315
6	CS-319
7	CS-319
8	FIN-201
9	HIS-351
10	MU-199



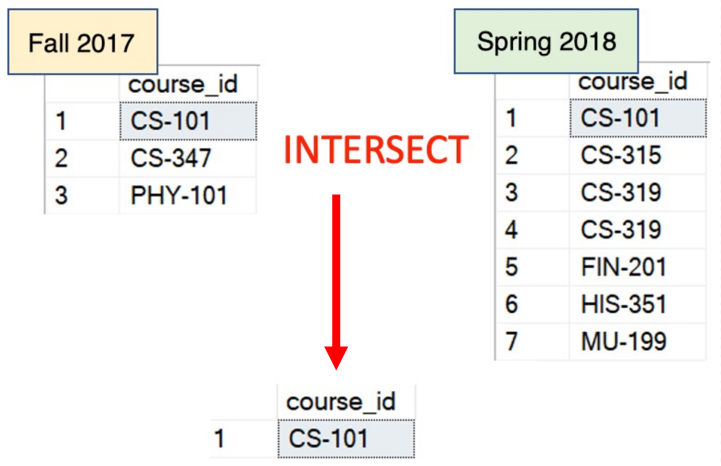
```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
union all
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```

Set operations in SQL – Intersect

- **INTERSECT** of two tables A and B, is a table containing all rows that are common to A and B
- E.g., list all the courses that are taught in both Fall 2017 and Spring 2018



```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
intersect
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```



Fall 2017

	course_id
1	CS-101
2	CS-347
3	PHY-101

INTERSECT

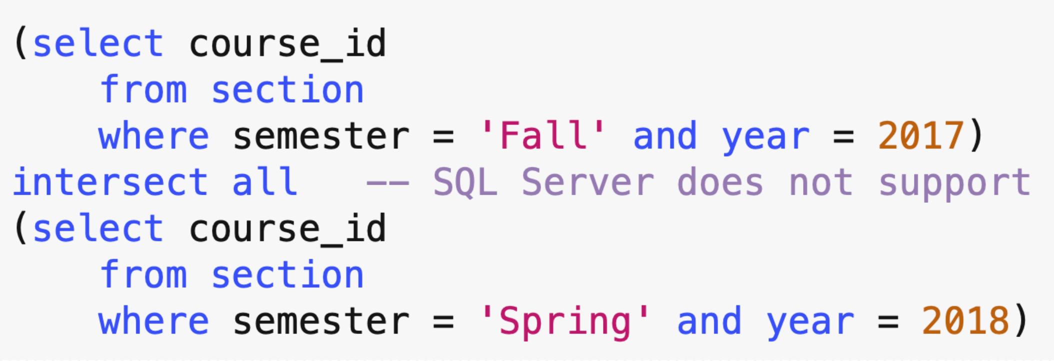
Spring 2018

	course_id
1	CS-101
2	CS-315
3	CS-319
4	CS-319
5	FIN-201
6	HIS-351
7	MU-199

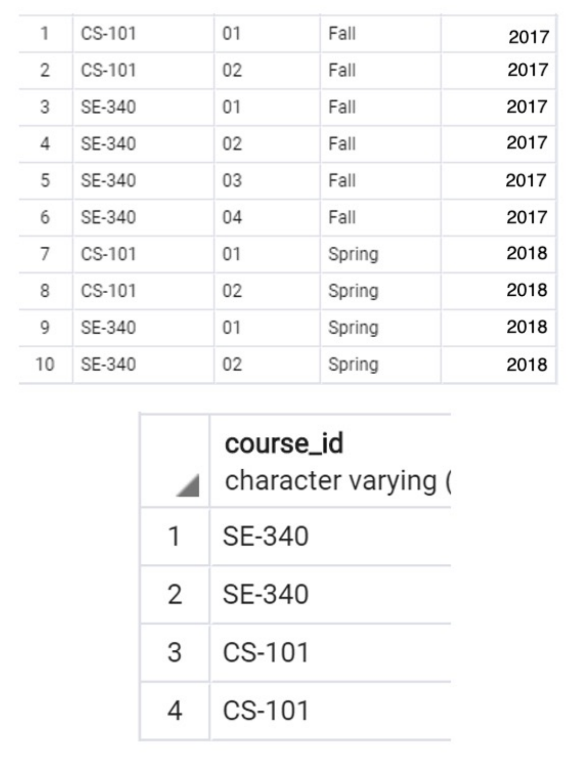
	course_id
1	CS-101

Set operations in SQL – Intersect All

- The **INTERSECT** operation automatically eliminates duplicates
- Use **INTERSECT ALL** instead to keep all duplicates



```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
intersect all   -- SQL Server does not support
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```



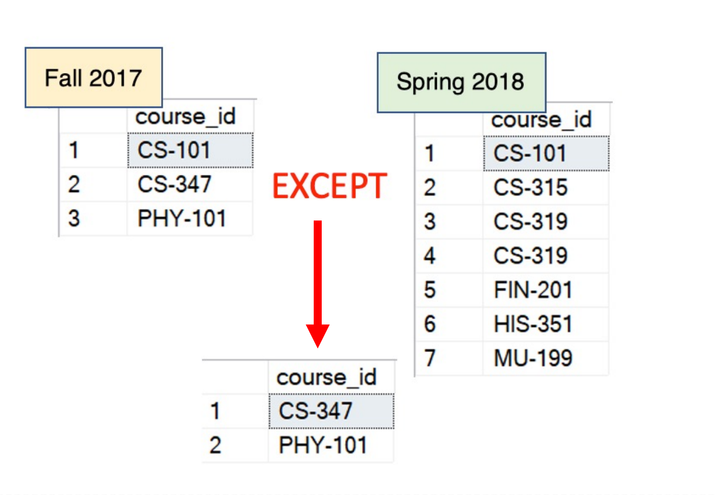
1	CS-101	01	Fall	2017
2	CS-101	02	Fall	2017
3	SE-340	01	Fall	2017
4	SE-340	02	Fall	2017
5	SE-340	03	Fall	2017
6	SE-340	04	Fall	2017
7	CS-101	01	Spring	2018
8	CS-101	02	Spring	2018
9	SE-340	01	Spring	2018
10	SE-340	02	Spring	2018

	course_id character varying (
1	SE-340
2	SE-340
3	CS-101
4	CS-101

Set operations in SQL – Except

- **EXCEPT** of two tables A and B, is a table containing all rows that are in A but not in B
- E.g., find all courses taught in Fall 2017 but not in Spring 2018

```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
except
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```



Fall 2017

	course_id
1	CS-101
2	CS-347
3	PHY-101

EXCEPT

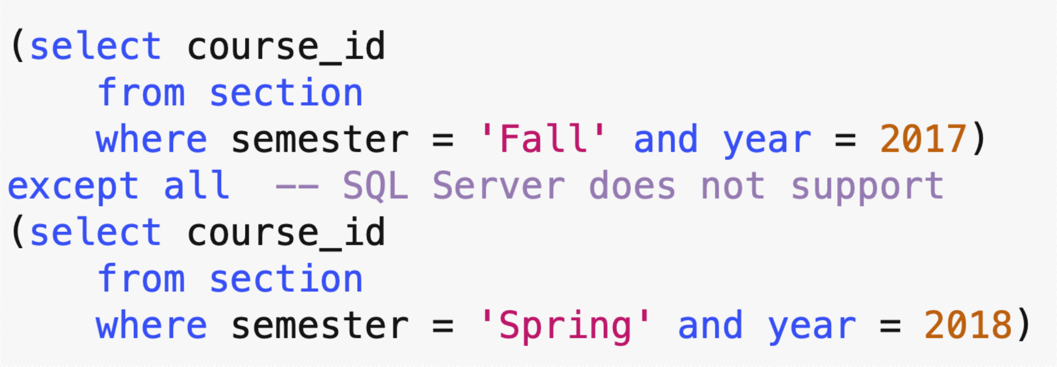
Spring 2018

	course_id
1	CS-101
2	CS-315
3	CS-319
4	CS-319
5	FIN-201
6	HIS-351
7	MU-199

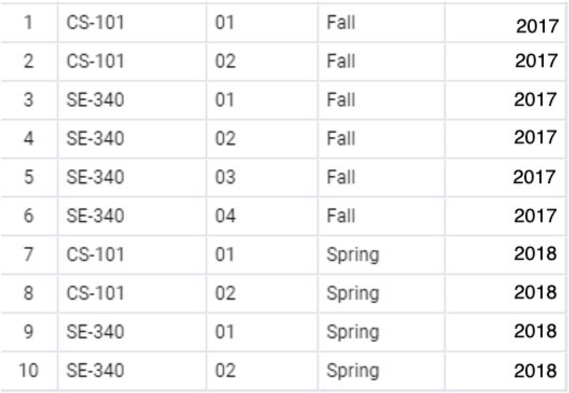
	course_id
1	CS-347
2	PHY-101

Set operations in SQL – Except All

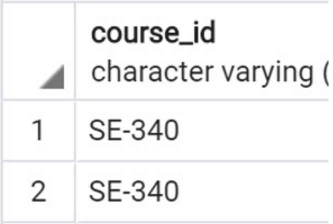
- **EXCEPT** operation automatically eliminates duplicates
- Use **EXCEPT ALL** instead to retain duplicates



```
(select course_id
    from section
    where semester = 'Fall' and year = 2017)
except all  -- SQL Server does not support
(select course_id
    from section
    where semester = 'Spring' and year = 2018)
```



1	CS-101	01	Fall	2017
2	CS-101	02	Fall	2017
3	SE-340	01	Fall	2017
4	SE-340	02	Fall	2017
5	SE-340	03	Fall	2017
6	SE-340	04	Fall	2017
7	CS-101	01	Spring	2018
8	CS-101	02	Spring	2018
9	SE-340	01	Spring	2018
10	SE-340	02	Spring	2018



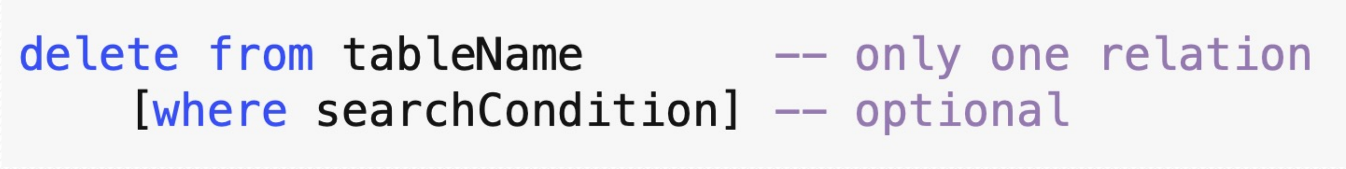
	course_id character varying (
1	SE-340
2	SE-340

Modification of the database

- **Deletion**
- **Insertion**
- **Updates**

Deletion

- The **DELETE** statement allows rows to be deleted from a named table
- The format of **DELETE**:



```
delete from tableName          -- only one relation
    [where searchCondition] -- optional
```

- **WHERE** clause can be as complex as a **SELECT** command’s **WHERE** clause

Deletion

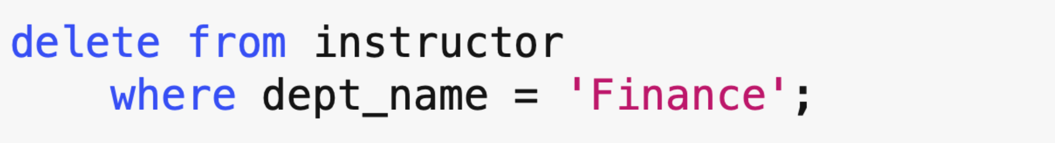
- E.g., delete all tuples from the instructor relation

delete from *instructor;*

- All tuples are deleted
- The instructor relation still exists
  - Empty

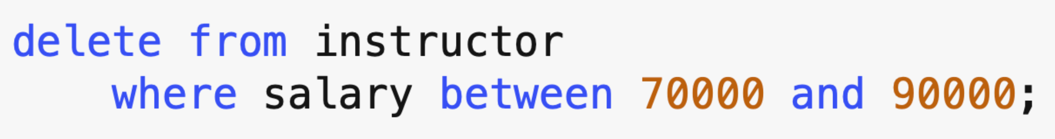
Deletion

- E.g., delete all instructor's information from Finance department



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

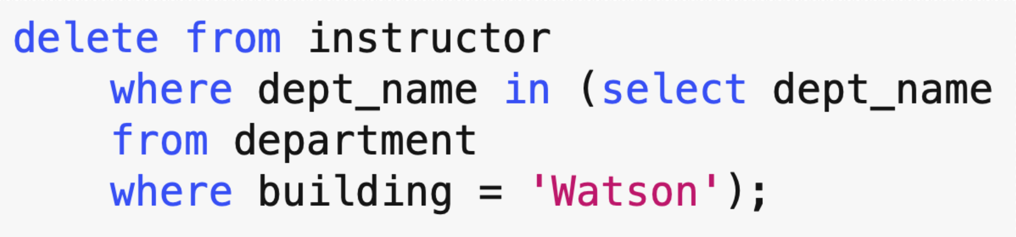
- E.g., delete all instructors with a salary between $70,000 and $90,000



```
delete from instructor
    where salary between 70000 and 90000;
```

Deletion

- E.g., delete all instructors whose dept located in the Watson building

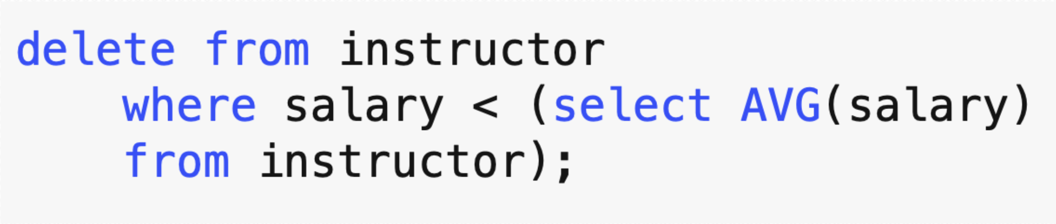


```
delete from instructor
    where dept_name in (select dept_name
    from department
    where building = 'Watson');
```

- First, find all departments located in Watson
- Then, delete tuples pertaining to those departments

Deletion

- E.g., delete all the records of all instructors with salary below the average at the university



```
delete from instructor
    where salary < (select AVG(salary)
    from instructor);
```

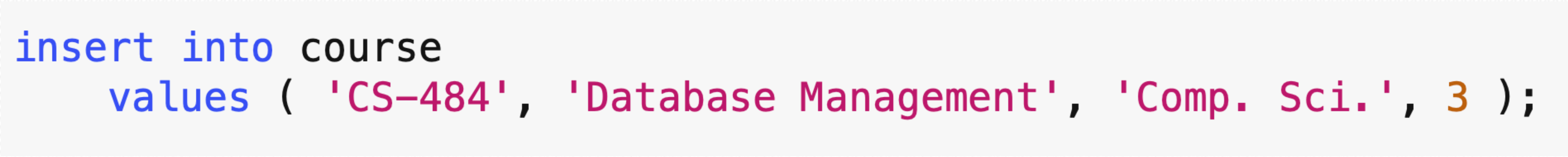
- First, test each tuple in the relation
- Then, delete all those tuples passing the test
  - instructors with a salary lower than average

Insertion

- To insert data into a relation
  - Specify a tuple to be inserted
    - **INSERT … VALUES**
  - Write a query whose result is a set of tuples to be inserted
    - **INSERT … SELECT**

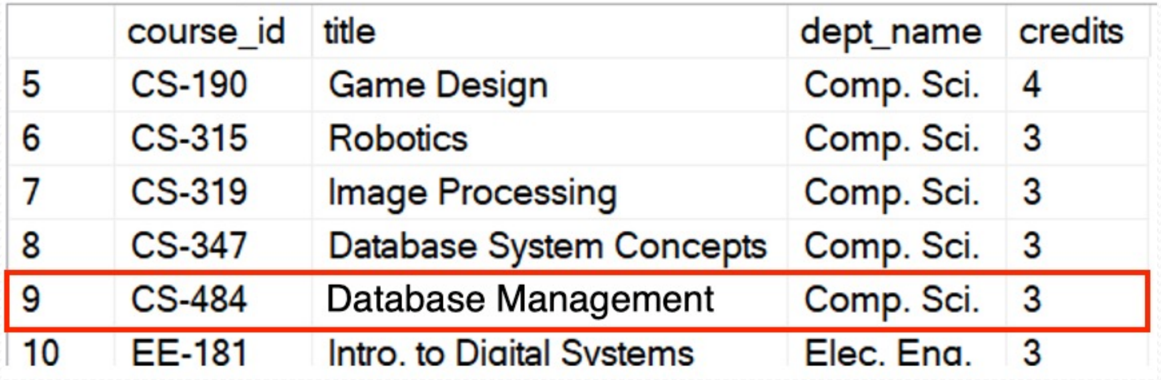
Insertion – specify tuple to be inserted

- E.g., insert a course CS-484 in the Computer Science department



```
insert into course
    values ( 'CS-484', 'Database Management', 'Comp. Sci.', 3 );
```

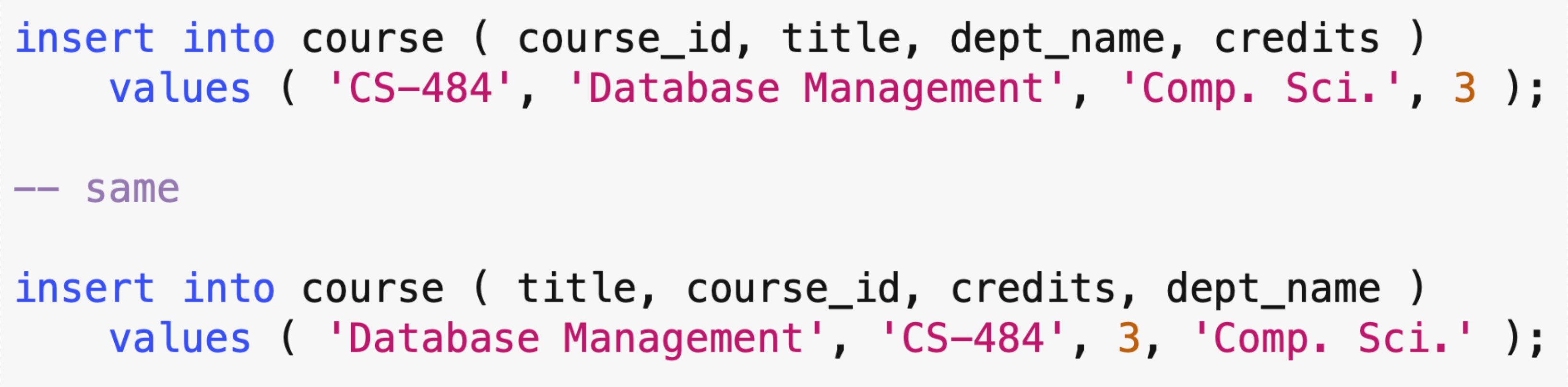
- The values are specified in the order in which the corresponding attributes are listed in the relation schema



	course_id	title	dept_name	credits
5	CS-190	Game Design	Comp. Sci.	4
6	CS-315	Robotics	Comp. Sci.	3
7	CS-319	Image Processing	Comp. Sci.	3
8	CS-347	Database System Concepts	Comp. Sci.	3
9	CS-484	Database Management	Comp. Sci.	3
10	EE-181	Intro. to Digital Systems	Elec. Eng.	3

Insertion – specify tuple to be inserted

- SQL allows the attributes to be specified as part of the **INSERT** statement



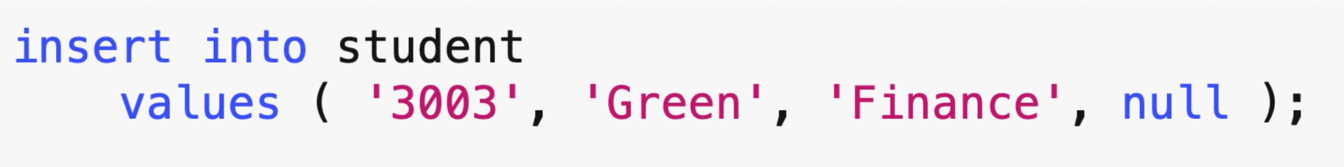
```
insert into course ( course_id, title, dept_name, credits )
    values ( 'CS-484', 'Database Management', 'Comp. Sci.', 3 );

-- same

insert into course ( title, course_id, credits, dept_name )
    values ( 'Database Management', 'CS-484', 3, 'Comp. Sci.' );
```

Insertion – specify tuple to be inserted

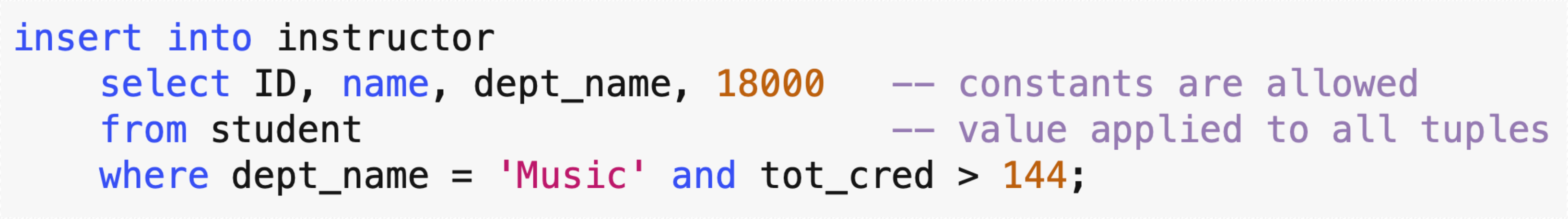
- If values are given on only some attributes, the remaining attributes are assigned a null value



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

Insert – Select

- Insert tuples based on the result of a query
- E.g., for those students in music dept who earned more than 144 credits, make them to be instructors in the music dept with a salary of $18,000

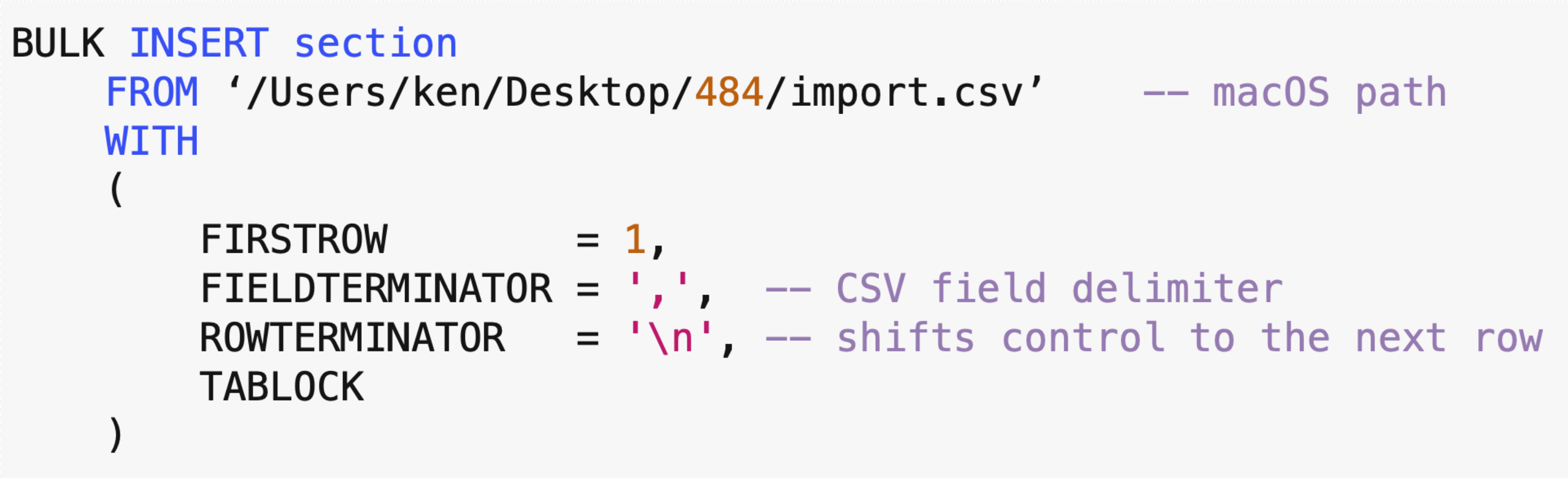


```
insert into instructor
    select ID, name, dept_name, 18000   -- constants are allowed
    from student                        -- value applied to all tuples
    where dept_name = 'Music' and tot_cred > 144;
```

- The system evaluates the **SELECT** statement fully before it performs any insertions
- The resulting relation must be consistent with the instructor schema

Insertion

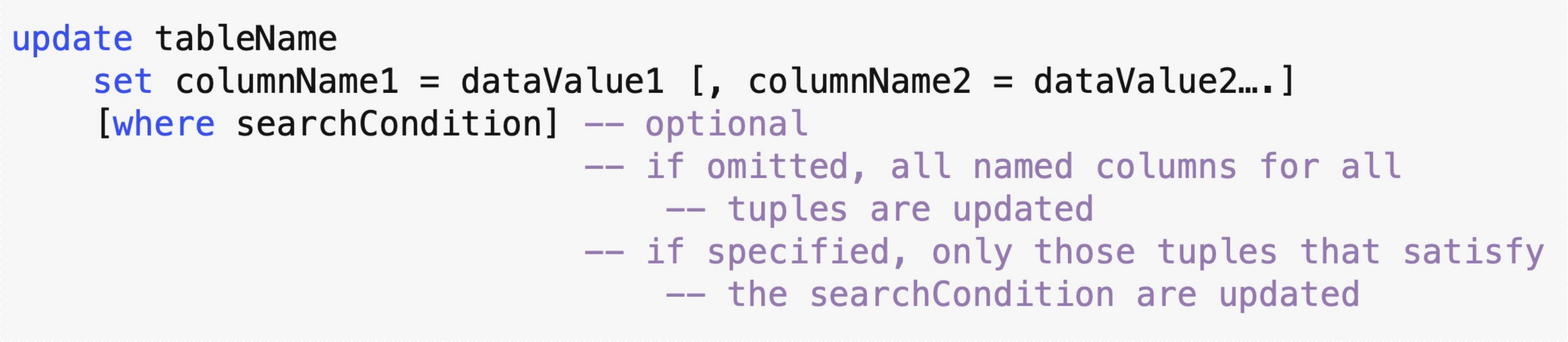
- Most RDBMS have special “bulk loader” utilities to insert a large set of tuples into a relation
- E.g., read a CSV file into a relation in SQL Server



```
BULK INSERT section
    FROM ‘/Users/ken/Desktop/484/import.csv’    -- macOS path
    WITH
    (
        FIRSTROW        = 1,
        FIELDTERMINATOR = ',',  -- CSV field delimiter
        ROWTERMINATOR   = '\n', -- shifts control to the next row
        TABLOCK
    )
```

Update

- The **UPDATE** statement allows the contents of existing rows in a named table to be changed.



```
update tableName
    set columnName1 = dataValue1 [, columnName2 = dataValue2….]
    [where searchCondition] -- optional
                            -- if omitted, all named columns for all
                                -- tuples are updated
                            -- if specified, only those tuples that satisfy
                                -- the searchCondition are updated
```

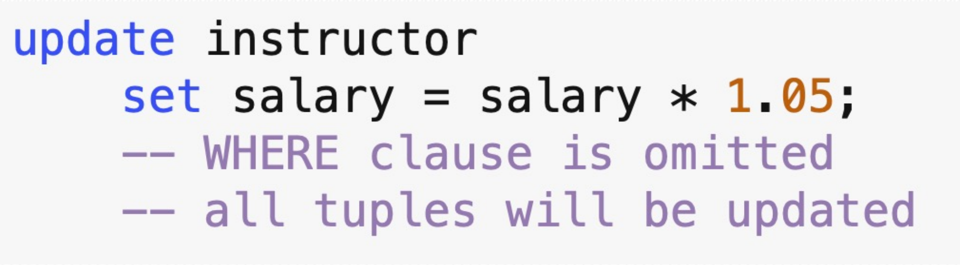
- Specify the names of one or more columns to be updated

Update

- **WHERE** clause
  - In general, the **WHERE** clause of the **UPDATE** statement may contain any legal construct legal found in the **WHERE** clause of the **SELECT** statement

Update

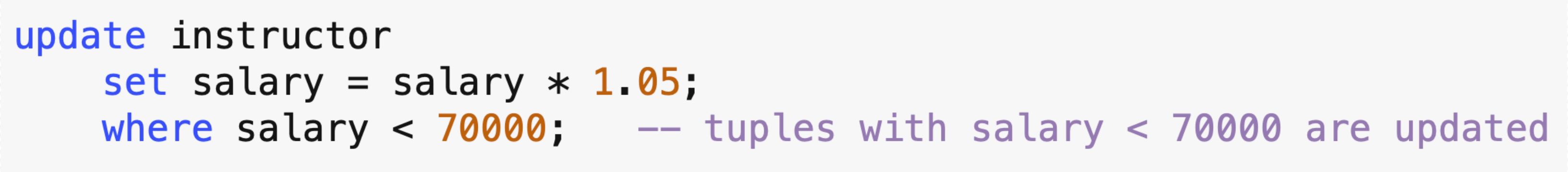
- E.g., increase all instructor's salary by 5 percent



```
update instructor
    set salary = salary * 1.05;
    -- WHERE clause is omitted
    -- all tuples will be updated
```

Update

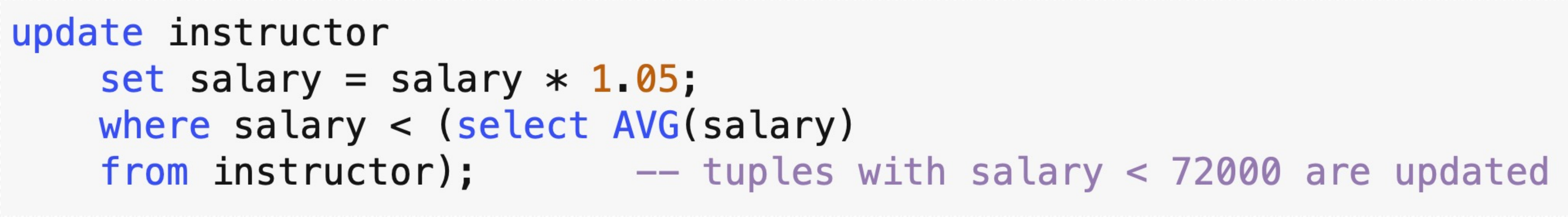
- E.g., apply a 5% salary increase to those instructors whose salaries are less than $70,000



```
update instructor
    set salary = salary * 1.05;
    where salary < 70000;   -- tuples with salary < 70000 are updated
```

Update

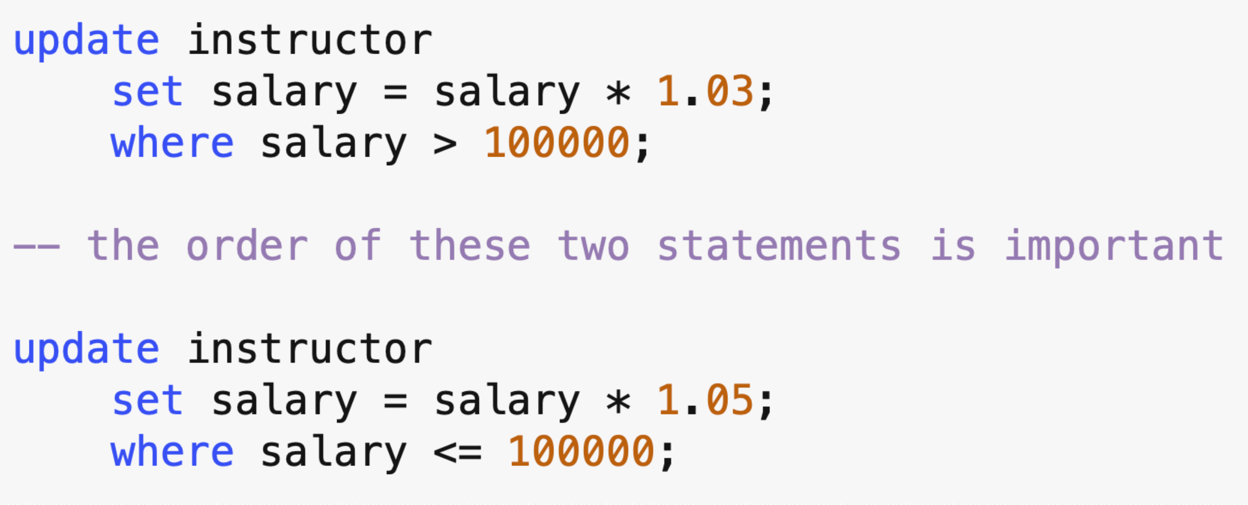
- E.g., apply a 5% salary increase to those instructors whose salaries are less than the average instructor salary



```
update instructor
    set salary = salary * 1.05;
    where salary < (select AVG(salary)
    from instructor);        -- tuples with salary < 72000 are updated
```

Update

- E.g., apply a salary increase to all instructors
  - 3% for salaries greater than $100,000
  - 5% for all others



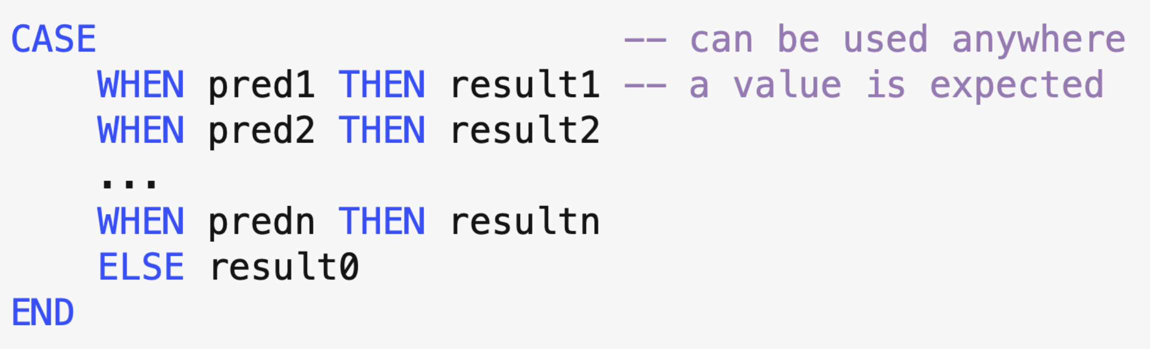
```sql
update instructor
    set salary = salary * 1.03;
    where salary > 100000;

-- the order of these two statements is important

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

Update

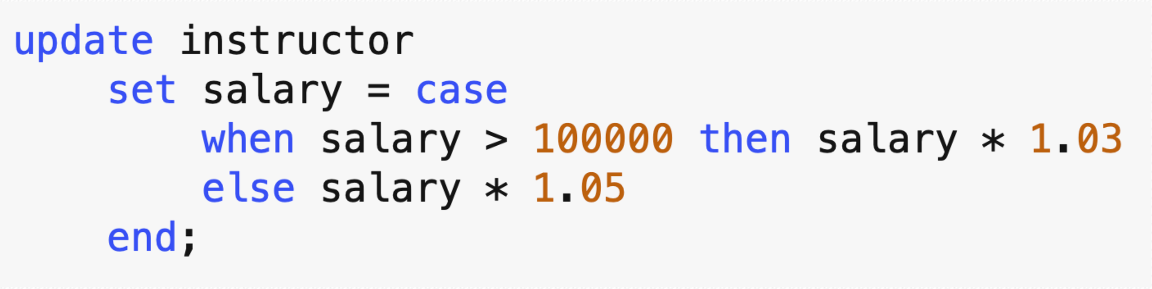
- SQL provides a **CASE** construct that we can use to perform both updates with a single **UPDATE** statement, avoiding the problem with the order of updates



```
CASE                            -- can be used anywhere
    WHEN pred1 THEN result1 -- a value is expected
    WHEN pred2 THEN result2
    ...
    WHEN predn THEN resultn
    ELSE result0
END
```

Update

- E.g., apply a salary increase to all instructors
  - 3% for salaries greater than $100,000
  - 5% for all others



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

Acknowledgements

- WIKIPEDIA
  - https://en.wikipedia.org/wiki/SQL