

Database System Concepts, 7th Edition

Chapter 3: Introduction to SQL

Silberschatz, Korth and Sudarshan

February 16, 2025

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit <https://db-book.com/>.

Plan

Overview of The SQL Query Language

History

- ▶ IBM Sequel language developed as part of System R project at IBM San Jose Research Laboratory.
- ▶ Renamed Structured Query Language (SQL).
- ▶ ANSI and ISO standard SQL:
 - ▶ SQL-86
 - ▶ SQL-89
 - ▶ SQL-92
 - ▶ SQL:1999¹
 - ▶ SQL:2003
- ▶ Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
 - ▶ Not all examples here may work on your particular system.

¹language name became Y2K compliant!.

SQL Parts

- ▶ DML – provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- ▶ Integrity – the DDL includes commands for specifying integrity constraints.
- ▶ View definition – the DDL includes commands for defining views.

SQL Parts

- ▶ Transaction control – includes commands for specifying the beginning and ending of transactions.
- ▶ Embedded SQL and dynamic SQL – define how SQL statements can be embedded within general-purpose programming languages.
- ▶ Authorization – includes commands for specifying access rights to relations and views.

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- ▶ The schema for each relation.
- ▶ The type of values associated with each attribute.
- ▶ The integrity constraints.
- ▶ The set of indices to be maintained for each relation.
- ▶ Security and authorization information for each relation.
- ▶ The physical storage structure of each relation on disk.

Domain Types in SQL

- ▶ `char(n)`. Fixed length character string, with user-specified length `n`.
- ▶ `varchar(n)`. Variable length character string, with user-specified maximum length `n`.
- ▶ `int`. Integer (a finite subset of integers that is machine-dependent).
- ▶ `smallint`. Small integer (a machine-dependent subset of integer domain type).

Domain Types in SQL

- ▶ `numeric(p, d)`. Fixed point number, with user-specified precision of `p` digits, with `d` digits to the right of decimal point. (ex., `numeric(3,1)`, allows 44.5 to be stored exactly, but not 444.5 or 0.32).
- ▶ `real`, `double precision`. Floating point and double-precision floating point numbers, with machine-dependent precision.
- ▶ `float(n)`. Floating point number, with user-specified precision of at least `n` digits.
- ▶ More are covered in Chapter 4.

Create Table Construct

- ▶ An SQL relation is defined using the `create table` command:

```
CREATE TABLE r (  
     $A_1D_1, A_2D_2, \dots, A_nD_n,$   
    (integrity_constraints1),  
    (integrity_constraints2),  
     $\vdots$   
    (integrity_constraintsk)  
);
```

- ▶ *r* is the name of the relation.
- ▶ each A_i is an attribute name in the schema of relation *r*.
- ▶ D_i is the data type of values in the domain of attribute A_i .

Create Table Construct

Example:

```
1 CREATE TABLE instructor (  
2     ID char(5),  
3     name varchar(20),  
4     dept_name varchar(20),  
5     salary numeric(8,2)  
6 );
```

Integrity Constraints in Create Table

- ▶ Types of integrity constraints:
 - ▶ `primary key(A_{j_1}, \dots, A_{j_m})`
 - ▶ `foreing key(A_{k_1}, \dots, A_{k_n})`
 - ▶ `not null`
- ▶ SQL prevents any update to the database that violates an integrity constraint.

Integrity Constraints in Create Table

Example:

```
1  CREATE TABLE instructor (  
2      ID          char(5),  
3      name        varchar(20),  
4      dept_name   varchar(20),  
5      salary      numeric(8,2),  
6      primary key (ID),  
7      foreign key (dept_name) references department  
8  );
```

Few more Relation Definitions

Example:

```
1  CREATE TABLE student (  
2      ID          varchar(5),  
3      name        varchar(20) not null,  
4      dept_name   varchar(20),  
5      tot_cred    numeric(3,0),  
6      primary key (ID),  
7      foreign key (dept_name) references department  
8  );
```

Few more Relation Definitions

Example:

```
1 CREATE TABLE takes (  
2     ID          varchar(5),  
3     course_id   varchar(8),  
4     sec_id      varchar(8),  
5     semester    varchar(6),  
6     year        numeric(4,0),  
7     grade       varchar(2),  
8     primary key (ID, course_id, sec_id, semester, year),  
9     foreign key (ID) references student,  
10    foreign key (course_id, sec_id, semester, year)  
11        references section  
12 );
```

And more still

Example:

```
1  CREATE TABLE course (  
2      course_id  varchar(8),  
3      title      varchar(50),  
4      dept_name  varchar(20),  
5      credits    numeric(2,0),  
6      primary key (course_id),  
7      foreign key (dept_name) references department  
8  );
```


Updates to Tables

► Insert:

```
INSERT INTO instructor  
VALUES ('10211', 'Smith', 'Biology', 66000);
```

► Delete:

```
DELETE FROM studentes;
```

► Drop a table:

```
DROP TABLE r;
```

Updates to Tables

- ▶ Alter:
 - ▶ alter table r add A D :
 - ▶ where A is the name of the attribute to be added to relation r and D is the domain of A .
 - ▶ all existing tuples in the relation are assigned *null* as the value for the new attribute.
 - ▶ alter table r drop A :
 - ▶ where A is the name of an attribute of relation r .
 - ▶ dropping of attributes not supported by many databases.

Basic Query Structure

- ▶ A typical SQL query has the form:

SELECT

A_1, A_2, \dots, A_n

FROM

r_1, r_2, \dots, r_m

WHERE

P

- ▶ A_i represents an attribute.
 - ▶ r_i represents a relation.
 - ▶ P is a predicate.
- ▶ The result of an SQL is a relation.

The SELECT Clause

- ▶ The **SELECT** clause lists the attributes desired in the result of a query.
 - ▶ It corresponds to the projection (Π) operation of the relational algebra.

Example:

“Find the names of all instructors.”

```
SELECT name FROM instructor;
```

The SELECT Clause

Note

SQL names are case sensitive (i.e., you may use upper- or lower-case letters.)

- ▶ E.g., Name \equiv NAME \equiv name.
- ▶ Some people use upper case wherever we use bold font.

The SELECT Clause (Cont.)

- ▶ SQL allows duplicates in relations as well as in query results.
- ▶ To force the elimination of duplicates, insert the keyword **DISTINCT** after the **SELECT** clause.

Example:

“Find the department names of all instructors, and remove duplicates.”

```
SELECT DISTINCT dept_name FROM instructor;
```

- ▶ The keyword **ALL** specifies that duplicates should not be removed.

```
SELECT ALL dept_name FROM instructor;
```

The SELECT Clause (Cont.)

- ▶ An asterisk (*) in the SELECT clause denotes “all attributes”.

```
SELECT * FROM instructor;
```

- ▶ An attribute can be a literal with FROM clause.

```
SELECT 'A' FROM instructor;
```

- ▶ Result is a table with one column and N rows (number of tuples in the *instructor* table), each row with value “A”.

The SELECT Clause (Cont.)

- ▶ An attribute can be a literal with no FROM clause.

```
SELECT '42';
```

- ▶ Results is a table with one column and a single row with value “42”.
- ▶ Can give the column a name using:

```
SELECT '42' AS foo;
```


The SELECT Clause (Cont.)

- ▶ The SELECT clause can contain arithmetic expression involving the operation, +, −, *, and /, and operating on constants or attributes of tuples.
 - ▶ The query:

```
SELECT ID, name, salary/12 FROM instructor;
```

would return a relation that is the same as the *instructor* relation, except that the value of the attribute **salary** is divided by 12.

- ▶ Can rename ‘‘salary/12’’ using the AS clause

```
SELECT ID, name, salary/12 AS monthly_salary  
FROM instructor;
```

The WHERE Clause

- ▶ The **WHERE** clause specifies conditions that the result must satisfy
 - ▶ Corresponds to the selection (σ) predicate of the relational algebra.

Example:

“Find all instructors in Comp. Sci. department.”

```
▶ SELECT name  
FROM instructor  
WHERE dept_name = 'Comp. Sci.';
```

The WHERE Clause

- ▶ SQL allows the use of the logical connectives **and**, **or**, and **not**.
- ▶ The operands of the logical connectives can be expressions involving the comparison operators $<$, $<=$, $>$, $>=$, $=$, and $<>$.
- ▶ Comparisons can be applied to results of arithmetic expressions.

Example:

“Find all instructors in Comp. Sci. with salary $>$ 80000.”

```
SELECT name  
FROM instructor  
WHERE dept_name = 'Comp. Sci.' AND salary > 80000;
```

The FROM Clause

- ▶ The FROM clause lists the relations involved in the query.
 - ▶ Corresponds to the Cartesian product (\times) operation of the relational algebra.

Example:

“Find the Cartesian product of *instructor* \times *teaches*.”

```
SELECT *  
FROM instructor, teaches;
```

The FROM Clause

- ▶ generates every possible instructor – teaches pair, with all attributes from both relations.
- ▶ for common attributes (e.g., `ID`), the attributes in the resulting table are renamed using the relation name (e.g., `instructor.ID`)
- ▶ Cartesian product is not very useful directly, but it is useful combined with the `WHERE` clause condition (σ operation in relational algebra).

Examples

“Find the names of all instructors who have taught some course and the course_id.”

Examples

“Find the names of all instructors who have taught some course and the course_id.”

```
1      SELECT
2          name, course_id
3      FROM
4          instructor, teaches
5      WHERE
6          instructor.ID = teaches.ID;
```

Examples

“Find the names of all instructors in the Art department who have taught some course and the course_id.”

Examples

“Find the names of all instructors in the Art department who have taught some course and the course_id.”

```
1  SELECT
2      name, course_id
3  FROM
4      instructor, teaches
5  WHERE
6      instructor.ID = teaches.ID AND
7      instructor.dept_name = 'Art';
```

The Rename Operation

- ▶ The SQL allows renaming relations and attributes using the **AS** clause:

old_name AS new_name

“Find the names of all instructors who have a higher salary than some instructor in Comp. Sci. deparment.”

```
1  SELECT DISTINCT
2      T.name
3  FROM
4      instructor AS T, instructor AS S
5  WHERE
6      T.salary > S.salary AND
7      S.dept_name = 'Comp. Sci.';
```

The Rename Operation

- ▶ keyword **AS** is optional and may be omitted.

instructor **AS** \equiv *instructor* *T*

Self Join Example

- ▶ Relation *employee – supervisor*:

person	supervisor
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- ▶ Exercises:
 - ▶ “Find the supervisor of Bob”
 - ▶ “Find the supervisor of the supervisor of Bob”
 - ▶ “Could you find ALL the supervisor (direct and indirect) of Bob?”

String Operations

- ▶ SQL includes a string-matching operator for comparisons on character strings. The operator **LIKE** uses patterns that are described using two special characters:
 - ▶ percent (%) – The % character matches any substring.
 - ▶ underscore(_) – The _ character any character.

Example:

“Find the names of all instructors whose name includes the substring ‘dar’.”

String Operations

```
1  SELECT
2      name
3  FROM
4      instructor
5  WHERE
6      name LIKE '%dar%';
```

- ▶ Match the string '100%'

```
LIKE '100\%';
```

in that above we use backslash (\) as the escape character.

String Operations (Cont.)

- ▶ Patterns are case sensitive.
- ▶ Pattern matching examples:
 - ▶ `'Intro%'` matches any string beginning with “Intro”.
 - ▶ `'%Comp%'` matches any string containing “Comp” as a substring.
 - ▶ `'___'` matches any string of exactly three characters.
 - ▶ `'___%'` matches any string of at least three characters.
- ▶ SQL supports a variety of string operations such as:
 - ▶ concatenation (using “||”).
 - ▶ converting from upper to lower case (and viceversa).
 - ▶ finding string length, extracting substrings, etc.

Ordering the Display of Tuples

Example:

“List in alphabetic order the names of all instructors.”

```
1      SELECT DISTINCT
2          name
3      FROM
4          instructor
5      ORDER BY
6          name;
```


Ordering the Display of Tuples

- ▶ We may specify **DESC** for descending order of **ASC** for ascending order, for each attribute; **ASC** is the default.

```
ORDER BY name DESC
```

- ▶ Can sort in multiple attributes.

```
ORDER BY dept_name, name
```

Where Clause Predicates

- ▶ SQL includes a **BETWEEN** comparison operator.

Example:

“Find the names of all instructors with salary between \$90000 and \$100000 (that is, \geq \$90000 and \leq \$100000).”

```
1      SELECT
2          name
3      FROM
4          instructor
5      WHERE
6          salary BETWEEN 90000 AND 100000;
```

Where Clause Predicates (Cont.)

- ▶ Tuple comparison:

```
1  SELECT
2      name, course_id
3  FROM
4      instructor, teaches
5  WHERE
6      (instructor.ID, dept_name) = (teaches.ID = 'Biology');
```

Set Operations

“Find courses that ran in Fall 2017 or in Spring 2018.”

```
1 (SELECT course_id FROM section WHERE sem = 'Fall' AND year = 2017)
2 UNION
3 (SELECT course_id FROM section WHERE sem = 'Spring' AND year = 2018)
```

“Find courses that ran in Fall 2017 and in Spring 2018.”

```
1 (SELECT course_id FROM section WHERE sem = 'Fall' AND year = 2017)
2 INTERSECT
3 (SELECT course_id FROM section WHERE sem = 'Spring' AND year = 2018)
```

Set Operations (Cont.)

“Find courses that ran in Fall 2017 but not in Spring 2018.”

```
1 (SELECT course_id FROM section WHERE sem = 'Fall' AND year = 2017)
2 EXCEPT
3 (SELECT course_id FROM section WHERE sem = 'Spring' AND year = 2018)
```

- ▶ Set operations UNION, INTERSECT, and EXCEPT automatically eliminate duplicates.
- ▶ To retain all duplicate use UNION ALL, INTERSECT ALL, and EXCEPT ALL.

Null Values

- ▶ It is possible for tuples to have a null value, denoted by `null`, for some of their attributes.
- ▶ `null` signifies an unknown value or that a value does not exist.
- ▶ The results of any arithmetic expression involving `null` is `null`.

Example

`5 + null` returns `null`.

Null Values (Cont.)

- ▶ The predicate `IS NULL` can be used to check for null values.

Example

“Find all instructors whose salary is null.”

```
SELECT name FROM instructor WHERE salary IS NULL
```

- ▶ The predicate `IS NOT NULL` succeeds if the value on which it is applied is not null.

Null Values (Cont.)

- ▶ SQL treats as **unknown** the result of any comparison involving a null value (other than predicates `IS NULL` and `IS NOT NULL`).
 - ▶ Example: `5 < null` or `null <> null` or `null = null`

Null Values (Cont.)

- ▶ The predicate in a WHERE clause can involve Boolean operations (AND, OR, NOT); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
 - ▶ AND:
 - $(\text{true AND unknown}) = \text{unknown}$
 - $(\text{false AND unknown}) = \text{false}$
 - $(\text{unknown AND unknown}) = \text{unknown}$
 - ▶ OR:
 - $(\text{unknown OR true}) = \text{true}$
 - $(\text{unknown OR false}) = \text{unknown}$
 - $(\text{unknown OR unknown}) = \text{unknown}$
- ▶ Result of WHERE clause predicate is treated as **false** if it evaluates to **unknown**.

Aggregate Functions

- ▶ These functions operate on the multiset of values of a column of a relation, and return a value:
 - ▶ **AVG**: average value.
 - ▶ **MIN**: minimum value.
 - ▶ **MAX**: maximum value.
 - ▶ **SUM**: sum of values.
 - ▶ **COUNT**: number of values.

Aggregate Functions Examples

“Find the average salary of instructors in the Computer Science department.”

```
1      SELECT
2          AVG(salary)
3      FROM
4          instructor
5      WHERE
6          dept_name = 'Comp. Sci.';
```

Aggregate Functions Examples

“Find the total number of instructors who teach a course in the Spring 2018 semester.”

```
1      SELECT
2          COUNT(DISTINCT ID)
3      FROM
4          teaches
5      WHERE
6          semester = 'Spring' AND year = 2018
```

Aggregate Functions Examples

“Find the number of tuples in the course relation.”

```
1      SELECT
2          COUNT(*)
3      FROM
4          course;
```

Aggregate Functions – Group By

“Find the average salary of instructors in each department.”

```
1  SELECT dept_name, AVG(salary) AS avg_salary
2  FROM instructor
3  GROUP BY dept_name;
```

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

- ▶ Attributes in **SELECT** clause outside of aggregate functions must appear in **GROUP BY** list.

```
1      /* erroneous query */
2      SELECT
3          dept_name, ID, AVG(salary)
4      FROM
5          instructor
6      GROUP BY
7          dept_name;
```

Aggregate Functions – Having Clause

“Find the names and average salaries of all departments whose average salary is greater than 42000”.

```
1      SELECT dept_name, AVG(salary) as avg_salary
2      FROM instructor
3      GROUP BY dept_name
4      HAVING AVG(salary) > 42000;
```

Note:

Predicates in the **HAVING** clause are applied after the formation of groups whereas predicates in the **WHERE** clause are applied before forming groups.

Nested Subqueries

- ▶ SQL provides a mechanism for the nesting of subqueries. A **subquery** is **SELECT-FROM-WHERE** expression that is nested within another query.
- ▶ The nesting can be done in the following SQL query:

SELECT

A_1, A_2, \dots, A_n

FROM

r_1, r_2, \dots, r_m

WHERE

P

as follows:

Nested Subqueries

SELECT

A_1, A_2, \dots, A_n

FROM

r_1, r_2, \dots, r_m

WHERE

P

as follows:

- ▶ *From clause*: r_i can be replaced by any valid subquery.
- ▶ *Where clause*: P can be replaced with an expression of the form:

$$B < operation > (subquery)$$

B is an attribute and $< operation >$ will be defined later.

- ▶ *Select clause*: A_i can be replaced by a subquery that generates a single value.

Set Membership

Find courses offered in Fall 2017 and in Spring 2018.

```
1  SELECT DISTINCT
2      course_id
3  FROM
4      section
5  WHERE
6      semester = 'Fall' AND year = 2017 AND
7      course_id IN (SELECT
8                      course_id
9                      FROM
10                     section
11                     WHERE
12                     semester = 'Spring' AND year = 2018
13                     );
```

Set Membership (Cont.)

Find courses offered in Fall 2017 but not in Spring 2018.

```
1  SELECT DISTINCT
2      course_id
3  FROM
4      section
5  WHERE
6      semester = 'Fall' AND year = 2017 AND
7      course_id NOT IN (SELECT
8                          course_id
9                          FROM
10                             section
11                             WHERE
12                                 semester = 'Spring' AND year = 2018
13                             );
```

Set Membership (Cont.)

Name all instructors whose name is neither “Mozart” nor “Einstein”.

```
1      SELECT DISTINCT
2          name
3      FROM
4          instructor
5      WHERE
6          name NOT IN ('Mozart', 'Einstein');
```

Set Membership (Cont.)

Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101.

```
1      SELECT
2          COUNT(DISTINCT ID)
3      FROM
4          takes
5      WHERE
6          (course_id, sec_id, semester, year) IN (
7              SELECT
8                  course_id, sec_id, semester, year
9              FROM
10                  teaches
11              WHERE
12                  teaches.ID = 10101
13          );
```

Note

Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

Set Comparison – “some” Clause

Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
1  SELECT DISTINCT
2      T.name
3  FROM
4      instructor AS T, instructor AS S
5  WHERE
6      T.salary > S.salary AND
7      S.dept_name = 'Biology';
```

Set Comparison – “some” Clause (Cont.)

Same query using > SOME clause.

```
1      SELECT DISTINCT
2          name
3      FROM
4          instructor
5      WHERE
6          salary > SOME (
7              SELECT
8                  salary
9              FROM
10                 instructor
11             WHERE
12                 dept_name = 'Biology'
13         );
```


Definition of “some” Clause

$F < comp > \text{SOME } r \Leftrightarrow \exists t \in r \text{ such that } (F < comp > t)$ where
 $< comp >$ can be: $<, \leq, >, \geq, =, \neq$.

$$\left(5 < \text{SOME} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array} \right) = true.$$

read: $5 <$ some tuple in the relation.

$$\left(5 < \text{SOME} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array} \right) = false.$$

$$\left(5 = \text{SOME} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array} \right) = true.$$

$$\left(5 \neq \text{SOME} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array} \right) = true.$$

Note $(= \text{SOME}) \equiv \text{IN}$. However, $(\neq \text{SOME}) \not\equiv \text{NOT IN}$.

Set Comparison – “all” Clause

Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
1      SELECT
2          name
3      FROM
4          instructor
5      WHERE
6          salary > ALL (
7              SELECT
8                  salary
9              FROM
10                 instructor
11             WHERE
12                 dept_name = 'Biology'
13         );
```

Definition of “all” Clause

$$F < comp > \text{ALL } r \Leftrightarrow \exists t \in r(F < comp > t).$$

$$\left(5 < \text{ALL} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array} \right) = false.$$

$$\left(5 < \text{ALL} \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array} \right) = true.$$

$$\left(5 = \text{ALL} \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array} \right) = false.$$

$$\left(5 \neq \text{ALL} \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array} \right) = true.$$

Note ($\neq \text{ALL}$) \equiv NOT IN. However, ($= \text{ALL}$) $\not\equiv$ IN.

Test for Empty Relations

- ▶ The **EXISTS** construct returns the value **true** if the argument subquery is nonempty.
- ▶ **EXISTS**

$$r \Leftrightarrow r \neq \emptyset$$

- ▶ **NOT EXISTS**

$$r \Leftrightarrow r = \emptyset$$

Use of “exists” Clause

- ▶ Yet another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”.

```
1      SELECT
2          course_id
3      FROM
4          section AS S
5      WHERE
6          semester = 'Fall' AND year = 2017 AND
7          EXISTS (
8              SELECT *
9              FROM section AS T
10             WHERE semester = 'Spring' AND
11                   year= 2018 AND
12                   S.course_id = T.course_id
13         );
```

Use of “exists” Clause

```
1      SELECT
2          course_id
3      FROM
4          section AS S
5      WHERE
6          semester = 'Fall' AND year = 2017 AND
7          EXISTS (
8              SELECT *
9              FROM section AS T
10             WHERE semester = 'Spring' AND
11                   year= 2018 AND
12                   S.course_id = T.course_id
13          );
```

- ▶ **Correlation name:** variable S in the outer query.
- ▶ **Correlated subquery:** the inner query,

Use of “not exists” Clause

- Find all students who have taken all courses offered in the Biology department.

```
1      SELECT DISTINCT
2          S.ID, S.name
3      FROM
4          student AS S
5      WHERE
6          NOT EXISTS (
7              ( SELECT course_id
8                  FROM course
9                  WHERE dept_name = 'Biology' )
10         EXCEPT
11         ( SELECT T.course_id
12             FROM takes AS T
13             WHERE S.ID = T.ID )
14     );
```

Use of “not exists” Clause

```
1      SELECT DISTINCT
2          S.ID, S.name
3      FROM
4          student AS S
5      WHERE
6          NOT EXISTS (
7              ( SELECT course_id
8                  FROM course
9                  WHERE dept_name = 'Biology' )
10             EXCEPT
11             ( SELECT T.course_id
12                 FROM takes AS T
13                 WHERE S.ID = T.ID )
14         );
```

- ▶ First nested query lists all courses offered in Biology.
- ▶ Second nested query lists all courses a particular student took.
- ▶ **Note:** That $X - Y = \emptyset \Leftrightarrow X \subseteq Y$.
- ▶ **Note:** Cannot write this query using = ALL and its variants.

Test for Absence of Duplicate Tuples

- ▶ The **UNIQUE** construct tests whether a subquery has any duplicate tuples in its result.
- ▶ The **UNIQUE** construct evaluates to **'true'** if a given subquery contains no duplicates.

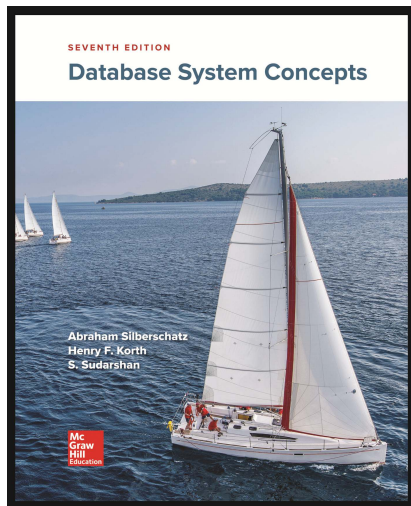
Test for Absence of Duplicate Tuples (Cont.)

- Find all courses that were offered at most once in 2017.

```
1      SELECT
2          T.course_id
3      FROM
4          course AS T
5      WHERE
6          UNIQUE (
7              SELECT
8                  R.course_id
9              FROM
10                 section AS R
11             WHERE
12                 T.course_id = R.course_id AND
13                 R.year = 2017
14         );
```

1

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit <https://db-book.com/>.