ECE30030/ITP30010 Database Systems

More SQL

Reading: Chapter 3

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Agenda

- Nested subqueries
- Set membership (SOME, ALL, EXISTS) unique.
- SQL DDL (Data Definition Language)

Running Examples

- Relations (tables): *instructor, teaches*

Instructor relation

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

teaches relation

ID	course_id	sec_id	semester	year
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
10101	CS-101	1	Fall	2017
45565	CS-101	1	Spring	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
10101	CS-315	1	Spring	2018
45565	CS-319	1	Spring	2018
83821	CS-319	2	Spring	2018
10101	CS-347	1	Fall	2017
98345	EE-181	1	Spring	2017
12121	FIN-201	1	Spring	2018
32343	HIS-351	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017

Running Examples

- Relations (tables): *course*, *takes*

course relation

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

takes relation

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	<null>

Running Examples

- Relations (tables): *student*

student relation

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Nested Subqueries

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

  SELECT $A_1, A_2, ..., A_n$
  FROM $r_1, r_2, ..., r_m$
  WHERE P

  as follows:

  B is attribute.

  - **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> to be defined later
  - **SELECT clause:**
    A_i can be replaced be a subquery that generates a single value (scalar subquery)

Subqueries in the FROM Clause

- Find the average instructors' salaries of those departments where the average salary is greater than $42,000
  - **SELECT** *D.dept_name*, *D.avg_salary*
    FROM (**SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
    FROM *instructor*
    GROUP BY *dept_name*) **AS** *D*
    WHERE *D.avg_salary* > 42000;

in this case where clause is the same as Having clause after the group by

dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Physics	91000.000000

WITH Clause

- The **WITH** clause provides a way of defining a temporary relation
  - The relation is available only to the query in which the **WITH** clause occurs

temporal table name   attribute name   value corresponding the attribute.

- Find all departments with the maximum budget
  - **WITH** *max_budget* (*value*) **AS** generate temporary relation
    (**SELECT MAX**(*budget*)
    FROM *department*)
    SELECT *department.dept_name*
    FROM *department, max_budget*
    WHERE *department.budget* = *max_budget.value*; and use it here.

Select dept_name,
from

dept_name
Finance

Scalar Subquery

- Scalar subquery is used where a single value is expected
  - Runtime error occurs if a subquery returns more than one result tuple
- List all departments along with the number of instructors in each department
  - **SELECT** *dept_name*,
    (**SELECT COUNT**(*)
    FROM *instructor*
    WHERE *department.dept_name* = *instructor.dept_name*)
    AS *num_instructors*
    FROM *department*;

dept_name	num_instructors
Biology	1
Comp. Sci.	3
Elec. Eng.	1
Finance	2
History	2
Music	1
Physics	2

Agenda

- Nested subqueries
- **Set membership (SOME, ALL, EXISTS)**
- SQL DDL (Data Definition Language)

Set Membership

- Find courses offered in Fall 2017 and in Spring 2018
  - **SELECT DISTINCT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Fall' **AND** *year*= 2017 **AND**
    course_id **IN** (**SELECT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Spring' **AND** *year*= 2018);

course_id
CS-101

Set Membership

- Find courses offered in Fall 2017 but not in Spring 2018
  - **SELECT DISTINCT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Fall' **AND** *year*= 2017 **AND**
    course_id ***NOT IN*** (**SELECT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Spring' **AND** *year*= 2018);

course_id
CS-347
PHY-101

Set Membership

- Name all instructors whose name is neither “Mozart” nor Einstein”
  - **SELECT DISTINCT** *name*
    FROM *instructor*
    WHERE *name* **NOT IN** ('Mozart', 'Einstein’);

we can manually list up the set member.

name
Srinivasan
Wu
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Set Membership

tuple level comparison.

- Find the total number of unique students who have taken course sections taught by the instructor with *ID* 10101
  - **SELECT COUNT**(**DISTINCT** *ID*)
    FROM *takes*
    WHERE (*course_id*, *sec_id*, *semester*, *year*) **IN**
    (**SELECT** *course_id*, *sec_id*, *semester*, *year*
    FROM *teaches*
    WHERE *teaches.ID*= 10101);

`COUNT(DISTINCT ID)`
6

- Note: *Above query can be written in a much simpler manner*
  The formulation above is simply to illustrate SQL features

How...?

Set Comparison – SOME

- Find names of instructors with salary greater than that of SOME (at least one) instructor in the Biology department
  - **SELECT DISTINCT** *T.name*
    FROM *instructor* **AS** *T*, *instructor* **AS** *S*
    WHERE *T.salary* > *S.salary* **AND** *S.dept name* = 'Biology';

- Same query using > **SOME** clause
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* > **SOME** (**SELECT** *salary*
    FROM *instructor*
    WHERE *dept_name* = 'Biology');

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

Interpretation of SOME

- F <comp> **SOME** *r* $\Leftrightarrow \exists t \in r$ such that (F <comp> *t*)
  Where <comp> can be: $<, \le, >, =, \neq$

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

(read: 5 < some tuple in the relation)

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

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

$$(5 \neq \textbf{SOME} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true (since } 0 \neq 5)$$

(= **SOME**) $\equiv$ **IN**
However, ($\neq$ **SOME**) $\not\equiv$ **NOT IN**

Set Comparison – ALL

- Find the names of ALL instructors whose salary is greater than the salary of ALL instructors in the Biology department
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* > **ALL** (**SELECT** *salary*
    FROM *instructor*
    WHERE *dept name* = 'Biology');

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

Interpretation of ALL

- F <comp> **ALL** $r \Leftrightarrow \forall t \in r$ (F <comp> t)

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

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

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

$$(5 \neq \textbf{ALL} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6\text{)}$$

($\neq$ **ALL**) $\equiv$ **NOT IN**

However, (= **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

- Yet another way of specifying the query "Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester"
  - ```
    SELECT course_id
    FROM teaches AS S
    WHERE semester = 'Fall' AND year = 2017 AND
          EXISTS (SELECT *
                  FROM teaches AS T
                  WHERE semester = 'Spring' AND year= 2018
                        AND S.course_id = T.course_id);
    ```

course_id
CS-101
    ```

Use of NOT EXISTS

- Find all students who have taken all courses offered in the Music department
  - **SELECT DISTINCT** *S.ID*, *S.name*
    FROM *student* **AS** *S*
    WHERE NOT EXISTS (**SELECT** *course_id*
    FROM *course*
    WHERE *dept_name* = 'Music'
    AND *course_id* **NOT IN**
    (**SELECT** *T.course_id*
    FROM *takes* **AS** *T*
    WHERE *S.ID* = *T.ID*));

ID	name
55739	Sanchez

Use of NOT EXISTS

- Note: Renaming (**AS**) is optional in certain contexts
  - **SELECT DISTINCT** *ID*, *name*
    FROM *student*
    WHERE NOT EXISTS (**SELECT** *course_id*
    FROM *course*
    WHERE *dept_name* = 'Music'
    AND *course_id* **NOT IN**
    (**SELECT** *course_id*
    FROM *takes*
    WHERE *student*.*ID* = *takes*.*ID*));
- Exception: the following query results in an empty relation
  - **SELECT DISTINCT** *name*
    FROM *instructor*
    WHERE *salary* > *salary* **AND** *dept_name* = 'Biology';

Use of NOT EXISTS

- Some systems support the **EXCEPT** clause (MySQL does not)
- Find all students who have taken all courses offered in the Music department
  - **SELECT DISTINCT** *S.ID*, *S.name*
    FROM *student* **AS** *S*
    WHERE NOT EXISTS ((**SELECT** *course_id*
    FROM *course*
    WHERE *dept_name* = 'Music')
    EXCEPT
    (**SELECT** *T.course_id*
    FROM *takes* **AS** *T*
    WHERE *S.ID* = *T.ID*));

Test for Absence of Duplicate Tuples

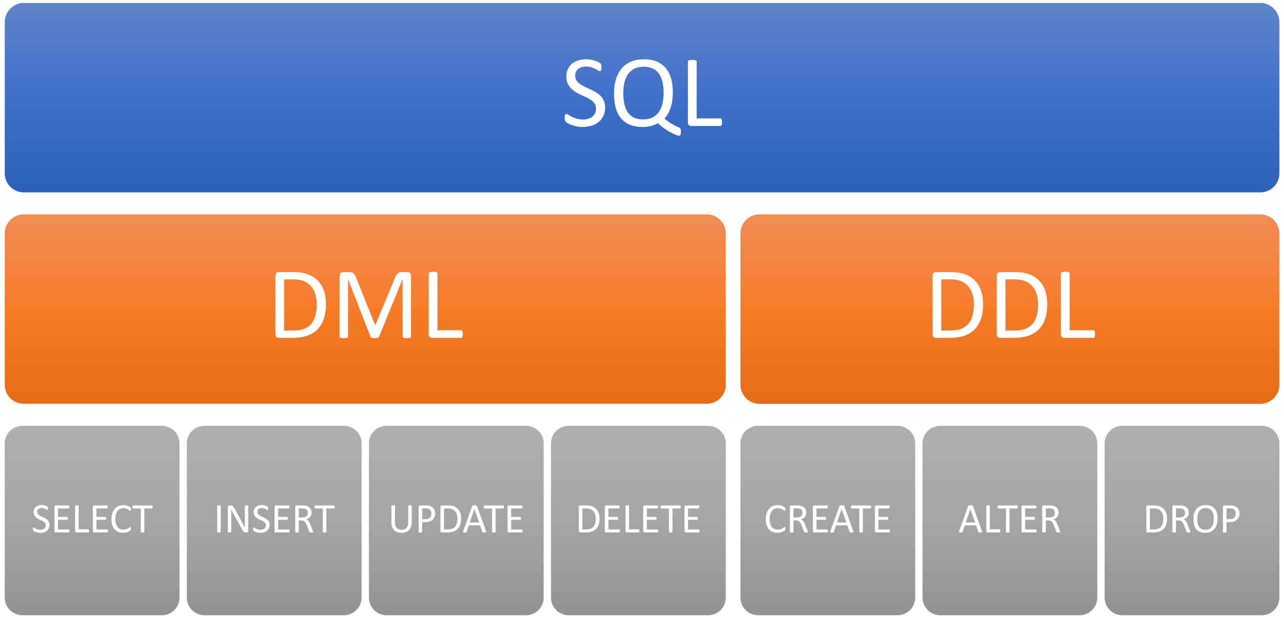
- The **UNIQUE** construct tests whether a subquery has any duplicate tuples in its result
  - **UNIQUE** evaluates to "true" if a given subquery contains no duplicates
  - MySQL does not support the UNIQUE test (UNIQUE in MySQL is a constraint specifier)

- Find all courses that were offered at most once in 2017
  - **SELECT** *T.course_id*
    FROM *course* **AS** *T*
    WHERE UNIQUE (**SELECT** *R.course_id*
    FROM *teaches* **AS** *R*
    WHERE *T.course_id*= *R.course_id* **AND** *R.year* = 2017);

Agenda

- Nested subqueries
- Set membership (SOME, ALL, EXISTS)
- **SQL DDL (Data Definition Language)**

SQL Commands



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

- SQL Data Types
  - **CHAR(*n*)**: Fixed length character string, with user-specified length *n*
    - Maximum length *n* = [0, 255]
  - **VARCHAR(*n*)**: Variable length character strings, with user-specified maximum length *n*
    - Maximum length *n* = [0, 65,535]

    - *If the length is always the same, use a CHAR-type attribute;*
      if you are storing wildly variable length strings, use a VARCHAR-type attribute

  - **TEXT**: for strings longer than the range of VARCHAR
    - TINYTEXT 0 – 255 bytes
    - TEXT 0 – 65,535 bytes
    - MEDIUMTEXT 0 – 16,777,215 bytes
    - LONGTEXT 0 – 4,294,967,295 bytes

Domain Types in SQL

- Difference between CHAR and VARCHAR

Value	CHAR(4)	Storage	VARCHAR(4)	Storage
‘’	‘ ‘	4 bytes	‘’	1 bytes
‘ab’	‘ab ‘	4 bytes	‘ab’	3 bytes
‘abcd’	‘abcd’	4 bytes	‘abcd’	5 bytes
‘abcdefg’	‘abcd’	4 bytes	‘abcd’	5 bytes

Domain Types in SQL

- “\”%ab%\””

Domain Types in SQL

- SQL Data Types
  - **INT, INTEGER**: Integer (a finite subset of the integers that is machine-dependent)
  - **SMALLINT**: Small integer (a machine-dependent subset of the integer domain type)
  - **BIGINT**: Small integer (a machine-dependent subset of the integer domain type)

  - **TINYINT** and **MEDIUMINT** are also available

Domain Types in SQL

- Different R-DBMSs support different combinations of those integer types

	Bytes	MySQL	MS SQL	PostgresSQL	DB2
TINYINT	1	✓	✓		
SMALLINT	2	✓	✓	✓	✓
MEDIUMINT	3	✓			
INT/INTEGER	4	✓	✓	✓	✓
BIGINT	8	✓	✓	✓	✓

  - *C.f.*, Oracle only has a NUMBER datatype

Domain Types in SQL

- SQL Data Types
  - **NUMERIC(*p*,*d*)**: Fixed point number (exact value) with user-specified precision of *p* digits, with *d* digits to the right of decimal point
    - *E.g.*, **NUMERIC**(3,1) allows 44.5 to be stores exactly, but not 444.5 or 0.32)
    - In MySQL, **DECIMAL** is NUMERIC
  - **FLOAT**: Floating point number (approximate) with single-precision
  - **REAL, DOUBLE**: Floating point number (approximate) with double-precision

Domain Types in SQL

- DECIMAL vs INT/FLOAT/DOUBLE
  - FLOAT and DOUBLE are faster than DECIMAL
  - DECIMAL values are exact
    - Example

floats: FLOAT	decimals: DECIMAL(3,2)
1.1	1.10
1.1	1.10
1.1	1.10

- SELECT SUM(…) → DECIMAL values are precise

SUM(floats)	SUM(decimals)
3.3000000715255737	3.30

Domain Types in SQL

- SQL Data Types
  - **DATE**: ‘YYYY-MM-DD’
    - Rage: 1000-01-01 to 9999-12-31
    - *E.g.*, ‘2020-03-01’ for March 1, 2020
  - **TIME**: ‘HH:MM:SS’
    - Range: -838:59:59 to 838:59:59
    - *E.g.*, ’14:30:03.5’ for 3.5 seconds after 2:30pm
  - **DATETIME**: ‘YYYY-MM-DD HH:MM:SS’
    - Range: 1000-01-01 00:00:00 to 9999-12-31 23:59:59
  - **YEAR**: ‘YYYY’
    - Range: 1901 to 2155, or 0000 (illegal year values are converted to 0000)

Domain Types in SQL

- SQL Data Types
  - **TIMESTAMP(*n*)**: Unix time (time since Jan 1, 1970)
    - Range: 1970-01-01 00:00:01 UTC to 2038-01-19 03:14:07 UTC
    - Typically used for logging (keeping records of all the system events)

    - Depending on size *n*, the display pattern changes

	Format
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

Domain Types in SQL

- SQL Data Types
  - **BINARY(*n*)**: binary byte data type, with user-specified length *n*
    - Contains a byte strings (rather than a character string)
    - Maximum length *n* = [0, 255]
  - **VARBINARY(*n*)**: binary byte data type, with user-specified maximum length *n*
    - Maximum length *n* = [0, 65,535]
  - **BLOB**: Binary Large OBject data type
    - TINYBLOB 0 – 255 bytes
    - BLOB 0 – 65,535 bytes (65 KB)
    - MEDIUMBLOB 0 – 16,777,215 bytes (16 MB)
    - LONGBLOB 0 – 4,294,967,295 bytes (4 GB)

CREATE TABLE Construct

- A new relation is defined using the **CREATE TABLE** command:

  CREATE TABLE *r*
  ($A_1\ D_1,\ A_2\ D_2,\ \ldots,\ A_n\ D_n$,
  (*integrity-constraint*$_1$),
  …,
  (*integrity-constraint*$_k$))

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

- Example:

```
CREATE TABLE instructor(
    ID          CHAR(5),
    name        VARCHAR(20),
    dept_name   VARCHAR(20),
    salary      NUMERIC(8,2))
```

Integrity Constraints in CREATE TABLE

- SQL prevents any update to the database that violates an integrity constraint
  - Integrity constraints allow us to specify what data makes sense for us
- Types of integrity constraints
  - Primary key: **PRIMARY KEY** (A_1, ..., A_n)
  - Foreign key: **FOREIGN KEY** (A_m, ..., A_n) **REFERENCES** *r*
  - Unique key: **UNIQUE**
  - Not null: **NOT NULL**
- Example:

```
CREATE TABLE instructor(
    ID          CHAR(5),
    name        VARCHAR(20) NOT NULL,
    dept_name   VARCHAR(20)
    salary      NUMERIC(8, 2),
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name) REFERENCES department);
```

Declaring Keys

- An attribute or list of attributes may be declared as PRIMARY KEY or UNIQUE
  - Meaning: no two tuples of the relation may agree in all the attribute(s) on the list
    - That is, the attribute(s) do(es) not allow duplicates in values
    - PRIMARY KEY/UNIQUE can be used as an identifier for each row
  - Comparison: PRIMARY KEY vs UNIQUE

PRIMARY KEY	UNIQUE
Used to serve as a unique identifier for each row in a relation	Uniquely determines a row which is not primary key
Cannot accept NULL	Can accept NULL values (some DBMSs accept only one NULL value)
A relation can have only one primary key	A relation can have more than one unique attributes
Clustered index	Non-clustered index

Declaring Keys

- **CREATE TABLE** *student* (

```
ID          VARCHAR(5),
name        VARCHAR(20) NOT NULL,
dept_name   VARCHAR(20),
tot_cred    NUMERIC(3,0),
PRIMARY KEY (ID),
FOREIGN KEY (dept_name) REFERENCES department);
```

More Examples

- **CREATE TABLE** *student* (
    ID **VARCHAR**(5) **PRIMARY KEY,**
    name **VARCHAR**(20) **NOT NULL,**
    dept_name **VARCHAR**(20),
    tot_cred **NUMERIC**(3,0),
    FOREIGN KEY *(dept_name*) **REFERENCES** *department*);

More Examples

- **CREATE TABLE** *takes* (

```
    ID          VARCHAR(5),
    course_id   VARCHAR(8),
    sec_id      VARCHAR(8),
    semester    VARCHAR(6),
    year        NUMERIC(4,0),
    grade       VARCHAR(2),
    PRIMARY KEY (ID, course_id, sec_id, semester, year),
    FOREIGN KEY (ID) REFERENCES  student,
    FOREIGN KEY (course_id, sec_id, semester, year)
                REFERENCES section);
```

More Examples

- **CREATE TABLE** *course* (

  course_id **VARCHAR**(8),
  title **VARCHAR**(50),
  dept_name **VARCHAR**(20),
  credits **NUMERIC**(2,0),
  PRIMARY KEY *(course_id),*
  FOREIGN KEY (*dept_name*) **REFERENCES** *department*);

More Examples

- **CREATE TABLE** *course* (

  course_id **VARCHAR**(8),
  title **VARCHAR**(50),
  dept_name **VARCHAR**(20) **DEFAULT** 'Comp. Sci',
  credits **NUMERIC**(2,0),
  PRIMARY KEY *(course_id),*
  FOREIGN KEY *(dept_name)* **REFERENCES** *department*);

More Examples

- **CREATE TABLE** *neighbors*(
  name **CHAR**(30) **PRIMARY KEY**,
  addr **CHAR**(50) **DEFAULT** ‘123 Sesame St.’,
  phone **CHAR**(16));

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES (‘Elmo’);

name	**addr**	**phone**
‘Elmo’	‘123 Sesame St.’	NULL

More Examples

- **CREATE TABLE** *neighbors*(
  name **CHAR**(30) **PRIMARY KEY**,
  addr **CHAR**(50) **DEFAULT** '123 Sesame St.',
  phone **CHAR**(16) **NOT NULL**);

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES ('Elmo');

      ➔ If phone were NOT NULL, this insertion would have been rejected

Table Updates (Updating Tuples)

- INSERT
  - **INSERT INTO** *instructor* **VALUES** (‘10211’, ‘Smith’, ‘Biology’, 66000)

- DELETE
  - **DELETE FROM** *student*
    - Remove all tuples from the *student* relation

Table Updates (Updating Table Schemas)

- DROP TABLE
  - **DROP TABLE** *r*
    - Remove relation *r*

- ALTER
  - **ALTER TABLE** *r* **ADD** *A D*
    - *A* is the name of the new attribute to add to relation *r*; *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*
    - *A* is the name of an attribute in *r*
    - Dropping of attributes not supported by many databases (MySQL does)

Table Updates (Updating Table Schemas)

- Examples
  - **DROP TABLE** time_slot_backup;
  - **ALTER TABLE** time_slot_backup **ADD** remark VARCHAR(20);
  - **ALTER TABLE** time_slot_backup **DROP** remark;

EOF

- Coming next:
  - Designing a database

SQL CASE Examples

- **SELECT** OrderID, Quantity,
  CASE
     WHEN Quantity > 30 **THEN** 'The quantity is greater than 30'
     WHEN Quantity = 30 **THEN** 'The quantity is 30'
     ELSE 'The quantity is under 30'
  END AS QuantityText
  FROM OrderDetails;

- **SELECT** CustomerName, City, Country
  FROM Customers
  ORDER BY
  (**CASE**
     WHEN City **IS NULL THEN** Country
     ELSE City
  END);