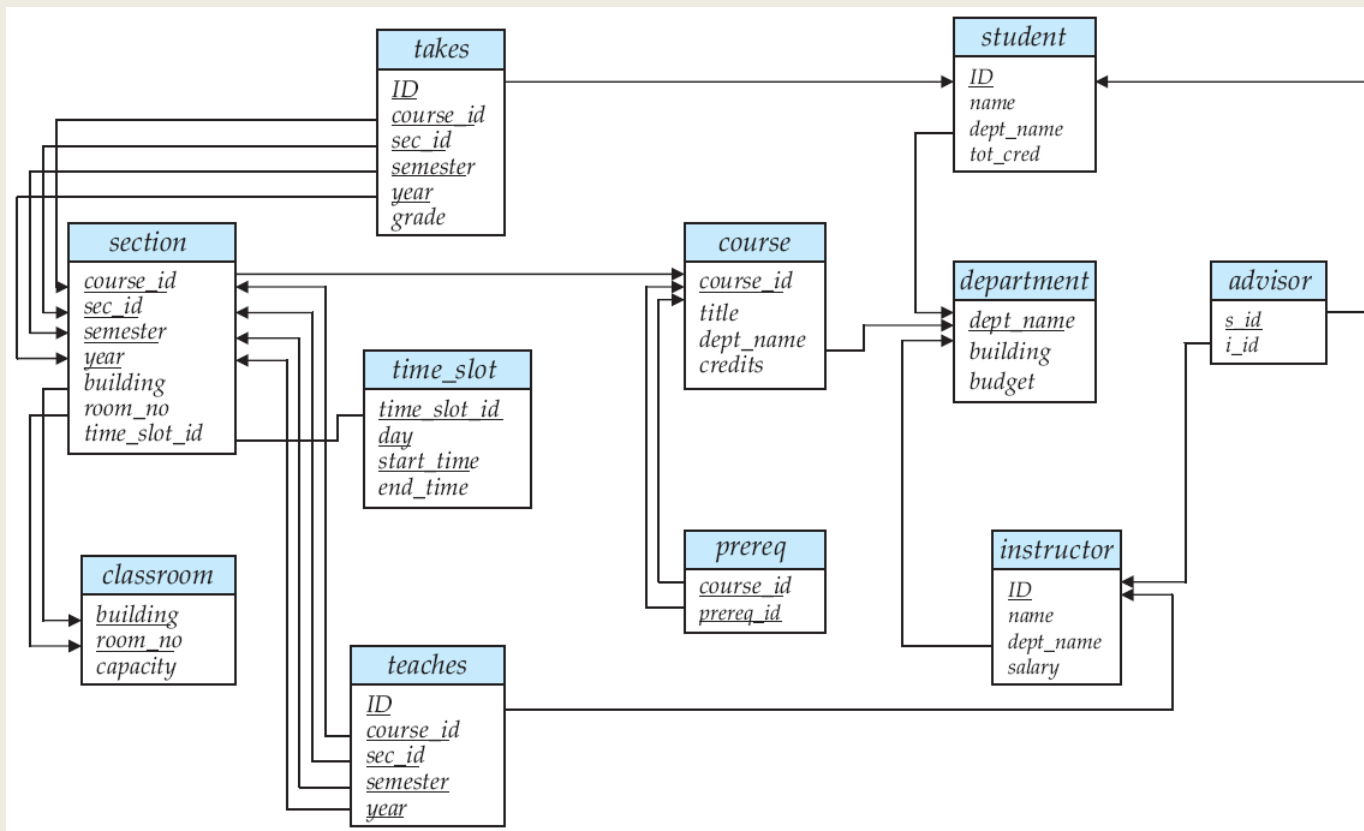# Databases – Intermediate SQL

Jaeyong Choi
Dept. of Software, Gachon University

Sample Database

❑ University database

University database *cont'd* ↩

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

Figure 2.1 The *instructor* 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

Figure 2.2 The *course* relation.

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
PhysicsI	Watson	70000

Figure 2.5 The *department* relation.

University database *cont'd*

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 2.6 The *section* relation.

course_id	prereq_id
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.

University database *cont'd* ↩

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

Figure 4.1 The *student* relation.

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	*null*

Figure 4.2 The *takes* relation.

Join Expressions

❑ For all instructors who have taught some course, find their names and the course ID of the courses they taught.

  ▢ **select** name, course_id **from** instructor, teaches **where** instructor.ID = teaches.ID;

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

instructor

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009

teaches

Join Expressions

instructor

select name, course_id **from** instructor, teaches **where** instructor.ID = teaches.ID;

teaches

join

Join Expressions

❑ General syntax
  ◻ **select** * **from** *table1* **join** *table2* **on** *search_condition*
 ◻ **select** * **from** *table1* **join** *table2* **using** (*join_column_list*)
❑ Examples
  ◻ **select** * **from** table1, table2;
  ◻ **select** * **from** table1 **INNER JOIN** table2 **ON** table1.id = table2.id;
  ◻ **select** * **from** table1 **LEFT JOIN** table2 **ON** table1.id = table2.id;
  ◻ **select** * **from** table1 **LEFT JOIN** table2 **USING** (id);
  ◻ **select** * **from** table1 **JOIN** table2 **ON** table1.id = table2.id **JOIN** table3 **ON** table2.id = table3.id;

Join Expressions

❑ **(INNER) JOIN**: Returns records that have matching values in both tables

❑ **LEFT JOIN**: Returns all records from the left table, and the matched records from the right table

❑ **RIGHT JOIN**: Returns all records from the right table, and the matched records from the left table

❑ **FULL JOIN**: Returns all records when there is a match in either left or right table
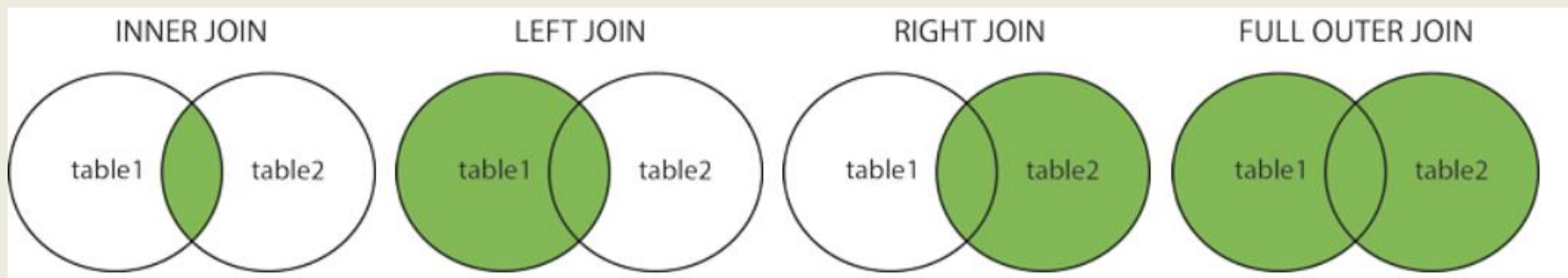
Join Expressions

❑ Join conditions
  ◻ The **on** condition allows a general predicate over the relations being joined
    ◻ **select** * **from** student **join** takes **on** student.ID= takes.ID; ⓘ
  ◻ Equivalent to:
    ◻ **select** * **from** student, takes **where** student.ID= takes.ID; ⓘ
  ◻ Almost the same as: .
    ◻ **select** * **from** student **natural join** takes;   ⓘ
    ◻ The result has the ID attribute listed only once

Natural join

❑ Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column

▫ **select** * **from** instructor **natural join** teaches;

natural join : id   1
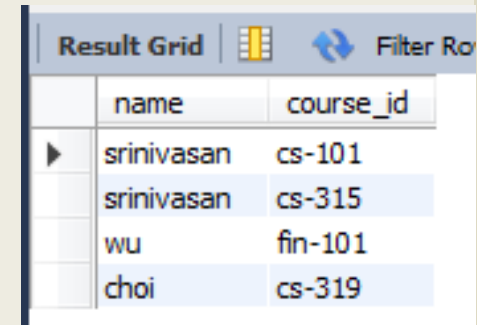join : ID column   2

instructor

teaches

Natural join

❑ List the names of instructors along with the course ID of the courses that they taught.

- **select** name, course_id
  from instructor, teaches
  where instructor.ID = teaches.ID;

- **select** name, course_id
  from instructor **natural join** teaches;

- **select** name, course_id **from** instructor **join** teaches **on** instructor.ID = teaches.ID;

- **select** name, course_id **from** instructor **join** teaches **using**(ID);

name	course_id
srinivasan	cs-101
srinivasan	cs-315
wu	fin-101
choi	cs-319

Views

❑ Views

◘ In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database)

◻ Consider a person who needs to know an instructor's ID, name, and department, but not the salary:

◻ **select** ID, name, dept_name **from** instructor;

◘ A *view* provides a mechanism to hide certain data from the view of certain users

◻ Any relation that is *not* of the conceptual model but is made visible to a user as a "virtual relation" is called a *view*

View definition

- ❏ A view is defined using the **create view** statement
  - ▫ **create view** *v* **as** *<query expression>*
  - ▫ The view name is represented by *v*; *<query expression>* is any legal SQL expression
- ❏ Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
- ❏ View definition is not the same as creating a new relation by evaluating the query expression
  - ▫ A view definition causes the saving of an SQL statement

View examples

❑ A view of instructors without their salary

  ◻ **create view** faculty **as select** ID, name, dept_name **from** instructor;

❑ Find all instructors in the Biology department

  ◻ **select** name **from** faculty **where** dept_name = 'Biology';

❑ Create a view of department salary totals

  ◻ **create view** departments_total_salary (dept_name, total_salary) **as select** dept_name, sum (salary) **from** instructor **group by** dept_name;

create view dept_sal_tot as select * from instructor gorup group by dept_name;
select * from dept_sal_tot;

            1   .

column
sum(salary)
total_salary

Views defined using other views

❑ One view may be used in the expression defining another view

                     v1   **v2**   .

- ◻ A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1

- ◻ A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2 **v2** **v1**   .

- ◻ A view relation v is said to be *recursive* if it depends on itself   .

Views defined using other views

❑ Create a view with class IDs, section ID, and building information taught in the physics department in the fall of 2009.

    ▢ **create view** physics_fall_2009 **as**
       select course.course_id, sec_id, building   → Attribute to view
       from course, section
       where course.course_id = section.course_id
              and course.dept_name = 'Physics'
              and section.semester = 'Fall'   → Condition
              and section.year = '2009';

    ▢ **select** * from **physics_fall_2009**;

Views defined using other views

- Create a view with information on the courses taught in the watson building among the courses taught in the physics department in the fall of 2009.

  - **create view** physics_fall_2009_watson **as**
    select course_id, room_number
    from physics_fall_2009
    where building= 'Watson';

```
create view CS_Spering_2009_IT as select course.course_id, sec_id, building from course, section
where course.course_id = section.course_id and    course.dept_name = 'Comp. Sci' and
section.semester = 'Spring'  and section.year = '2009' and building = 'IT';
```

```
create view CS_spring_2009_IT2 as select * from CS_Spring_2009 where building = 'IT';
```

View expansion

❑ Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations

❑ View expansion of an expression e_1 repeats the following replacement step:

   ▪ **repeat**
      Find any view relation v_i in e_1;
      Replace the view relation v_i by the expression defining v_i;
      until no more view relations are present in e_1;

   ▪ As long as the view definitions are not recursive, this loop will terminate

View expansion

❑ Create a view with information on the courses taught in the watson building among the courses taught in the physics department in the fall of 2009.

▫ **create view** physics_fall_2009_watson **as**   v1
 select course_id, room_number
 from (**select** course.course_id, building, room_number   e1
   from course, section
   where course.course_id = section.course_id
       and course.dept_name = 'Physics'
       and section.semester = 'Fall'
       and section.year = '2009') as physics_fall_2009
 where building= 'Watson';

Update of a view

❑ A view of instructors without their salary

   ❑ **create view** faculty **as select** ID, name, dept_name
      from instructor;

❑ Add a new tuple to faculty view which we defined earlier

   ❑ **insert into** faculty **values** ('30765', 'Green', 'Music');

❑ This insertion must be represented by the insertion of the following tuple into the *instructor* relation

   ❑ ('30765', 'Green', 'Music', null)

Update of a view

❑ Some updates cannot be translated uniquely

  ◻ Consider a view:

    ◻ **create view** instructor_info **as**
      select ID, name, building
      from instructor, department ⓘ
      where instructor.dept_name= department.dept_name;

  ◻ When trying:

    ◻ **insert into** instructor_info **values** ('69987', 'White', 'Taylor');

    ◻ which department, if multiple departments in Taylor?

    ◻ what if no department is in Taylor?

Update of a view

❑ Most SQL implementations allow updates only on simple views
- ◻ The **from** clause has only one database relation
- ◻ The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification
- ◻ Any attribute not listed in the **select** clause can be set to null
- ◻ The query does not have a **group by** or **having** clause

❑ And some not at all
- ◻ **create view** history_instructors **as select** * **from** instructor **where** dept_name= 'History';
- ◻ What happens if we insert ('25566', 'Brown', 'Biology', 100000) into history_instructors?

Materialized views

❏ *Materializing* a view: create a *physical* table containing all the tuples in the result of the query defining the view

- It is a concept to improve the speed of Query by saving the results of frequently used views to disk.

❏ If relations used in the query are updated, the materialized view result becomes out of date

- Need to maintain the view, by updating the view whenever the underlying relations are updated

❏ Syntax (oracle, Ms sql but not mysql)

- CREATE MATERIALIZED VIEW MV_MY_VIEW
  REFRESH FAST START WITH SYSDATE
  NEXT SYSDATE + 1
  AS SELECT * FROM <table_name>;

Transactions

- ❑ Transactions
  - ◘ Consists of a sequence of query and /or update statements
  - ◘ *Atomic* transaction
    - ◘ Either fully executed or rolled back as if it never occurred
  - ◘ Isolation from concurrent transactions
  - ◘ One of the following SQL statement must end the transaction
    - ◘ Ended by *commit* or *rollback*
  - ◘ *Auto-commit*: each SQL statement commits automatically
    - ◘ Can turn off for a session
  - ◘ More on chapter 14

Integrity Constraints

❑ Integrity constraints

  ◘ Guards against accidental damage to the database

  ◘ Ensures authorized changes to the database do not result in a loss of data consistency

❑ Examples:

  ◘ A checking account must have a balance ≥ $10,000

  ◘ A salary of a bank employee must be ≥ $4.00 an hour

  ◘ A customer must have a (non-null) phone number

❑ Integrity constraints on a single relation

    ◻ **not null**

    ◻ **primary key**

    ◻ **unique**

    ◻ **check**(P), where P is a predicate

❏ *Not null*

- ▣ Declare name and budget attributes to be not null
  - ▣ **create table** …
    name **varchar**(20) **not null**,
    budget **numeric**(12,2) **not null** …

❏ *Unique*

- ▣ **unique**(A_1, A_2, … , A_m)
- ▣ States the attributes A_1, A_2, … , A_m form a candidate key
- ▣ Candidate keys are permitted to be null (in contrast to the primary key)

❑ *Check* clause

◘ E.g., ensure that semester is one of fall, winter, spring, or summer

◘ **create table** section (
    course_id **varchar** (8),
    sec_id **varchar** (8),
    semester **varchar** (6),
    year **numeric** (4,0),
    …
    primary key (course_id, sec_id, semester, year),
    check (semester **in** ('Fall', 'Winter', 'Spring', 'Summer')));

❑ Referential integrity (참조 무결성)

◻ Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation

◻ E.g., if "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there must exist a tuple in the *department* relation for "Biology"

◻ *A* is said to be a *foreign key* of *R*, if for any values of *A* appearing in *R* these values also appear in *S*

◻ Let *A* be a set of attributes. Let *R* and *S* be two relations that contain attributes *A*, and *A* is the *primary key* of *S*

R
A
B

S
A
C
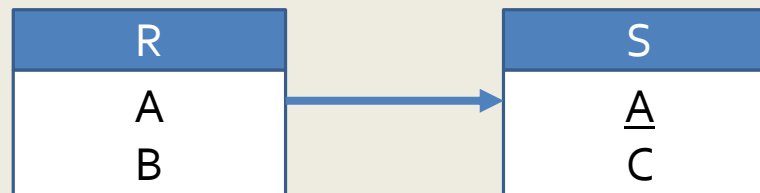
❑ Cascading actions in referential integrity

　◻ **create table** course (🛈

　　　…
　　　dept_name **varchar** (20),
　　　foreign key (dept_name) **references** department
　　　　　on delete cascade
　　　　　on update cascade, …
　　　);

　◻ Alternative actions: **set null**, **set default**

Exercises

❑ Create the table and use the "on update cascade", "on delete cascade", "on delete set null"

- ⬛ Change the software department's code to 501 and make sure all the changed values are reflected in other related databases.
- ⬛ Delete the chemistry department code and change the department code of chemistry students to null.

Stud_id	Dept_code	Name
20201234	101	홍길동
20191211	201	김가천
20184213	102	최지우
20171235	101	배용준
20209822	202	최민수
20192385	102	이민호
20195431	101	송민국

Dept_code	Dept_name
101	소프트웨어학과
102	전자과
201	화학과
202	경영학과

Built-in data types in SQL

- **date**: dates, containing (4 digit) year, month, and date
  - E.g., date '2005-7-27'
- **time**: time of day, in hours, minutes, and seconds
  - E.g., time '09:00:30', time '09:00:30.75'
- **timestamp**: date plus time of day
  - E.g., timestamp '2005-7-27 09:00:30.75'
- **interval**: period of time
  - E.g., interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

Built-in data types in SQL

❑ Examples
- ◘ CREATE TABLE people (
  id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  birth_date **DATE** NOT NULL,
  curr_time **timestamp**) ;
- ◘ INSERT INTO people VALUES('10', 'John','Doe','1990-09-01`, now());

❑ Functions related to date
- ◘ SELECT NOW();
- ◘ SELECT DATE(NOW());
- ◘ SELECT CURDATE();
- ◘ SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y') today;

Index creation

❏ Index
  ◘ Indices are data structures used to speed up access to records with the specified values for index attributes

❏ Examples
  ◘ **create table** student (
      ID **varchar** (5) **primary key**,
      name **varchar** (20) **not null**,
      dept_name **varchar** (20),
      tot_cred **numeric** (3,0) **default** 0);
      create index studentID_index on student(ID)
  ◘ **select** * **from** student **where** ID = '12345';
      ◘ Can be executed by using the index to find the required record, without looking at all records of student
  ◘ More on indices in Chapter 11