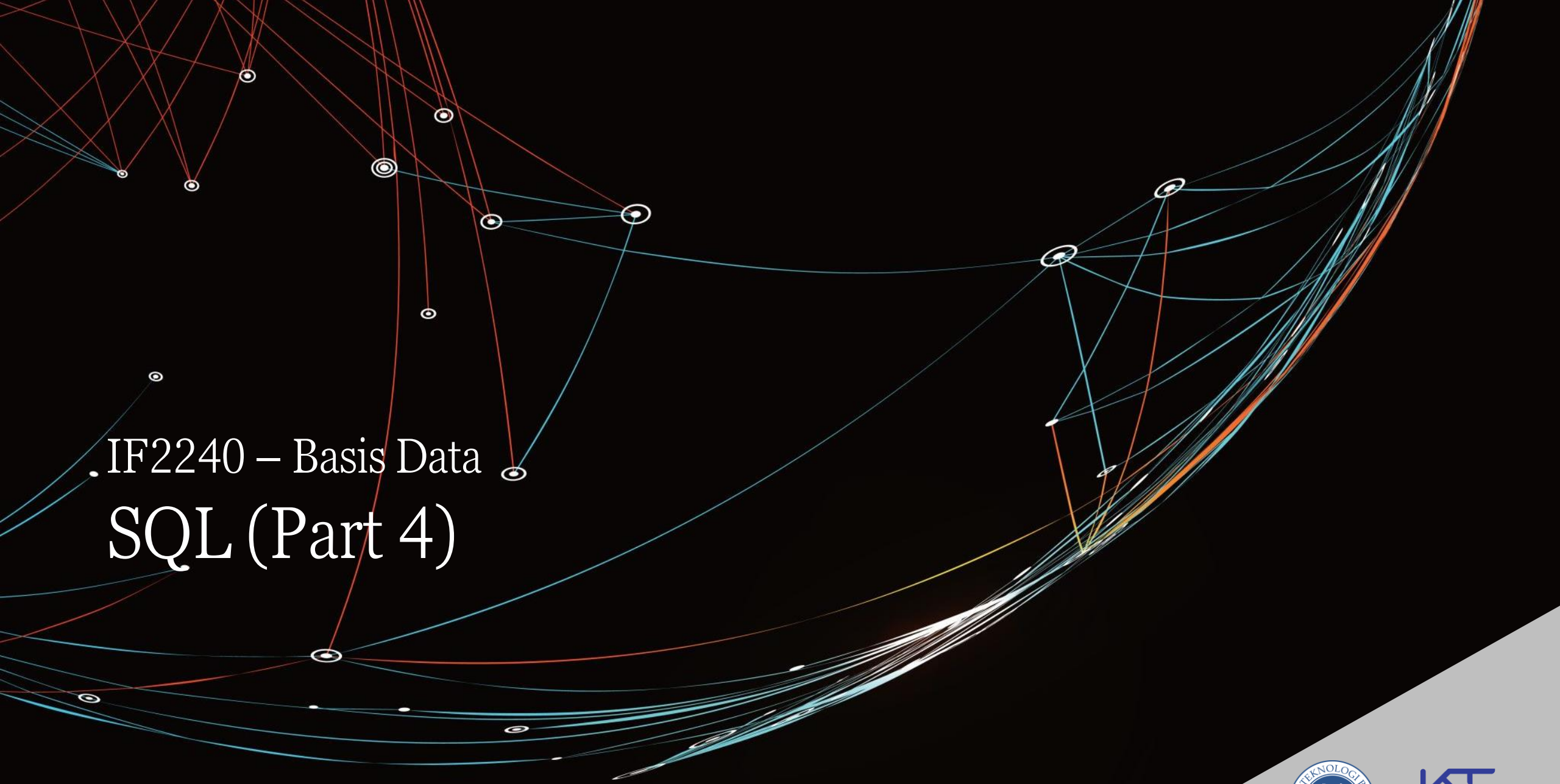
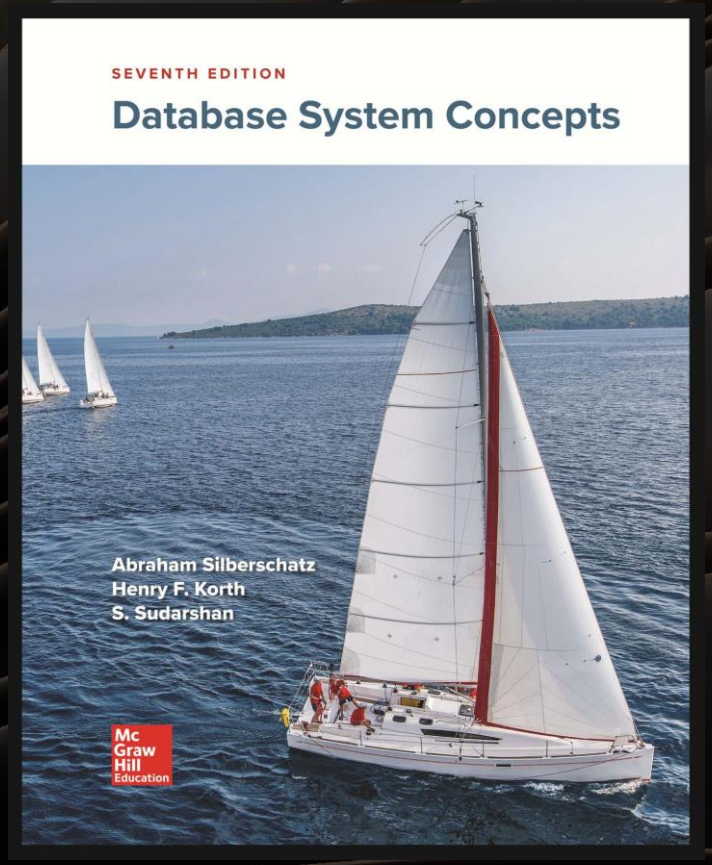


IF2240 – Basis Data

SQL (Part 4)

KNOWLEDGE & SOFTWARE ENGINEERING

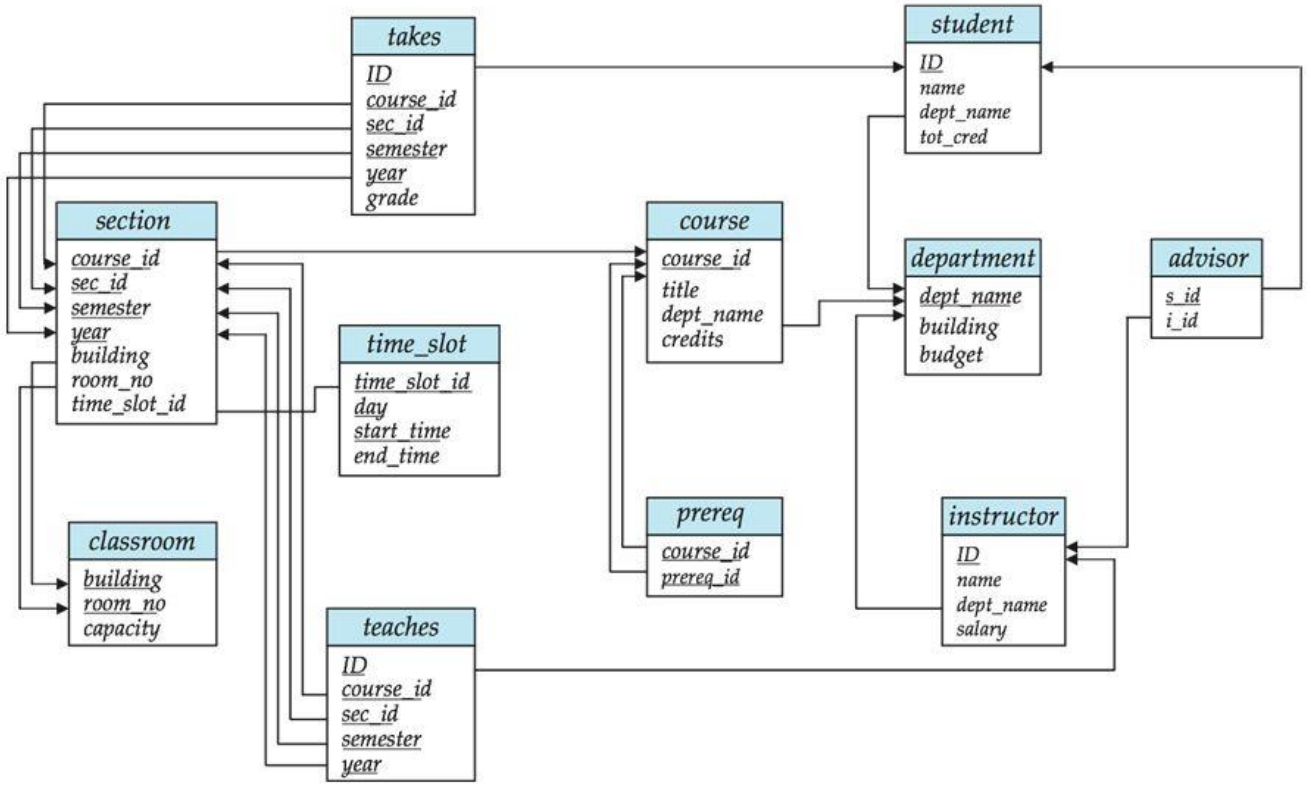


Sumber

Silberschatz, Korth, Sudarshan: "Database System Concepts", 7th Edition

- Chapter 3 : Introduction to SQL
- Chapter 4 : Intermediate SQL

Schema Diagram



Database System Concepts - 6th Edition 2 ©Silberschatz, Korth and Sudarshan

Views

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

Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
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 which has the form

create view *v* **as** < query expression >

where <query expression> is any legal SQL expression. The view name is represented by *v*.

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

- Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

View Definition and Use

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;
```

Views Defined Using Other Views

One view may be used in the expression defining another view

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

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

Views Defined Using Other Views

```
create view physics_fall_2017 as
    select course.course_id, sec_id, building, room_number
    from course, section
    where course.course_id = section.course_id
                and course.dept_name = 'Physics'
                and section.semester = 'Fall'
                and section.year = '2017';


create view physics_fall_2017_watson as
     select course_id, room_number
     from physics_fall_2017
     where building= 'Watson';
```

KNOWLEDGE & SOFTWARE ENGINEERING

Materialized Views

Certain database systems allow view relations to be physically stored.

- Physical copy created when the view is defined.
- Such views are called **Materialized view**:

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.

Modification of the Database

Deletion of tuples from a given relation.

Insertion of new tuples into a given relation

Updating of values in some tuples in a given relation

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 *instructor*, 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)

Insertion

Add a new tuple to *course*

```
insert into course
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

or equivalently

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

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

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

Insertion (Cont.)

Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

```
insert into instructor
        select ID, name, dept_name, 18000
        from   student
        where  dept_name = 'Music' and total_cred > 144;
```

The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

```
insert into table1 select * from table1
```

would cause problem

Updates

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
```

Updates with Scalar Subqueries

Recompute and update tot_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id  and
                      S.ID= takes.ID and
                      takes.grade <> 'F' and
                      takes.grade is not null);
```

Sets *tot_creds* to null for students who have not taken any course

Instead of **sum**(*credits*), use:

```
case
 when sum(credits) is not null then sum(credits)
 else 0
end
```

Update of a View

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 into the *instructor* relation

- Must have a value for salary.

Two approaches

- Reject the insert
- Insert the tuple

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

into the *instructor* relation

Some Updates Cannot be Translated Uniquely

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

insert into instructor_info

              values ('69987', 'White', 'Taylor');
```

Issues

- Which department, if multiple departments in Taylor?
- What if no department is in Taylor?

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?*

View Updates in SQL

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.

Data Definition Language

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

1. The schema for each relation.
2. The type of values associated with each attribute.
3. The Integrity constraints
4. The set of indices to be maintained for each relation.
5. Security and authorization information for each relation.
6. The physical storage structure of each relation on disk.

Domain Types in SQL (1)

char(n). Fixed length character string, with user-specified length *n*.

varchar(n). Variable length character strings, with user-specified maximum length *n*.

int. Integer (a finite subset of the integers that is machine-dependent).

smallint. Small integer (a machine-dependent subset of the integer domain type).

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 stores 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.

Domain Types in SQL (2)

date: Dates, containing a (4 digit) year, month and date

- Example: **date** '2005-7-27'

time: Time of day, in hours, minutes and seconds.

- Example: **time** '09:00:30' **time** '09:00:30.75'

timestamp: date plus time of day

- Example: **timestamp** '2005-7-27 09:00:30.75'

interval: period of time

- Example: 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

Create Table Construct

An SQL relation is defined using the **create table** command:

create table *r*

(A_1 D_1, A_2 D_2, ..., 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*
- 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

Types of integrity constraints

- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*
- **not null**

SQL prevents any update to the database that violates an integrity constraint.

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);
```

And a Few More Relation Definitions

```
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);

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);
```

And more still

```
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);
```

Updates to tables

Drop Table

- drop table *r*

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 exiting 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.

View Updates in SQL

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.