Advanced SQL
Domain Types in SQL

- **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 $n$ digits to the right of decimal point.
- **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_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) not null, 
  dept_name varchar(20),
  salary numeric(8,2))
```

• **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);
• **insert into** *instructor* **values** (‘10211’, null, ’Biology’, 66000);
Integrity Constraints in Create Table

• **not null**
• **primary key** \((A_1, \ldots, A_n)\)
• **foreign key** \((A_m, \ldots, A_n)\) references \(r\)

Example: Declare \textit{branch\_name} as the primary key for \textit{branch}

```
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)
```

**primary key** declaration on an attribute automatically ensures **not null**
And a Few More Relation Definitions

- **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`);

- **create table** `takes` (  
  `ID` varchar(5) primary key,  
  `course_id` varchar(8),  
  `sec_id` varchar(8),  
  `semester` varchar(6),  
  `year` numeric(4,0),  
  `grade` varchar(2),  
  **foreign key** (`ID`) references `student`,  
  **foreign key** (`course_id`, `sec_id`, `semester`, `year`) references `section`);
• create table course (  
course_id varchar(8) primary key,  
title varchar(50),  
dept_name varchar(20),  
credits numeric(2,0),  
foreign key (dept_name) references department) ;
Drop and Alter Table Constructs

- **drop table**
- **alter table**
  - `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 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.
Modification of the Database – 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');`
Example Query

• 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 deposit, 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)
Modification of the Database – Insertion

• Add a new tuple to course
  
  \textbf{insert into} course  
  \textbf{values} (\textquote{CS-437}, \textquote{Database Systems}, \textquote{Comp. Sci.}, 4);

• or equivalently

  \textbf{insert into} course (\textit{course\_id}, \textit{title}, \textit{dept\_name}, \textit{credits})  
  \textbf{values} (\textquote{CS-437}, \textquote{Database Systems}, \textquote{Comp. Sci.}, 4);

• Add a new tuple to student with \textit{tot\_creds} set to null

  \textbf{insert into} student  
  \textbf{values} (\textquote{3003}, \textquote{Green}, \textquote{Finance}, \textquote{null});
Modification of the Database – Insertion

• Add all instructors to the *student* relation with tot_creds set to 0
  
  ```
  insert into student
  select ID, name, dept_name, 0
  from instructor
  ```

• 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 problems)
Modification of the Database – Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - Write two `update` statements:
    ```sql
    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

```sql
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
  
  ```sql
  update student S
  set tot_cred = ( select sum(credits) 
                  from takes natural join course 
                  where 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:
  
  ```sql
  case
    when sum(credits) is not null then sum(credits)
    else 0
  end
  ```
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 instructor's name and department, but not the salary. This person should see a relation described, in SQL, by

    ```sql
    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**.
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.
Example Views

- A view of instructors without their salary
  ```sql
  create view faculty as
  select ID, name, dept_name
  from instructor
  ```

- Find all instructors in the Biology department
  ```sql
  select name
  from faculty
  where dept_name = 'Biology'
  ```

- Create a view of department salary totals
  ```sql
  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

- create view physics_fall_2009 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 = '2009';

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

• Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_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 = '2009')
where building= 'Watson';
```
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.
View Expansion

• A way to define the meaning of views defined in terms of other views.
• Let view $v_1$ be defined by an expression $e_1$ that may itself contain uses of view relations.
• View expansion of an expression 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.
Update of a View

• Add a new tuple to \textit{faculty} view which we defined earlier

\textbf{insert into faculty values} ('30765', 'Green', 'Music');

This insertion must be represented by the insertion of the tuple

(‘30765’, ’Green’, ’Music’, null)

into the \textit{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');
  • which department, if multiple departments in Taylor?
  • what if no department is in Taylor?

• 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`  
  ```sql
  select *
  from instructor
  where dept_name= 'History';
  ```
- **Insert** (’25566’, ’Brown’, ’Biology’, 100000) into `history_instructors`
Transactions

• Unit of work
• Atomic transaction
  – either fully executed or rolled back as if it never occurred
• Isolation from concurrent transactions
• Transactions begin implicitly
  – Ended by commit work or rollback work
• But default on most databases: each SQL statement commits automatically
  – Can turn off auto commit for a session (e.g. using API)
  – In SQL:1999, can use: begin atomic .... end
Integrity Constraints

• Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  – A checking account must have a balance greater than $10,000.00.
  – A salary of a bank employee must be at least $4.00 an hour.
  – A customer must have a (non-null) phone number.
Constraints on a Single Relation

- not null
- primary key
- unique
- check \((P)\), where \(P\) is a predicate
Not Null and Unique Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**
    
    ```
    name varchar(20) not null
    budget numeric(12,2) not null
    ```

- **unique** \( (A_1, A_2, \ldots, A_m) \)
  - The unique specification states that the attributes \( A_1, A_2, \ldots, A_m \) form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).
The check clause

- **check** \((P)\)
  where \(P\) is a predicate

Example: ensure that semester is one of fall, winter, spring or summer:

```sql
create table section (
    course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time_slot_id varchar (4),
    primary key (course_id, sec_id, semester, year),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```
Referential Integrity

- 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.
  - Example: If “Biology” is a department name appearing in one of the tuples in the instructor relation, then there exists a tuple in the department relation for “Biology”.

- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. 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.
Cascading Actions in Referential Integrity

• `create table course (  
  course_id  char(5) primary key,  
  title      varchar(20),  
  dept_name  varchar(20) references department
 )`

• `create table course (  
  ...  
  dept_name  varchar(20),  
  foreign key (dept_name) references department  
    on delete cascade  
    on update cascade,  
  ...  
 )`

• alternative actions to cascade: `set null, set default`
Integrity Constraint Violation During Transactions

• E.g.,

```sql
create table person (  
    ID char(10),  
    name char(40),  
    mother char(10),  
    father char(10),  
    primary key ID,  
    foreign key father references person,  
    foreign key mother references person)
```

• How to insert a tuple?

• What if `mother` or `father` is declared not null?
  
  – `constraint father_ref foreign key father references person,  
    constraint mother_ref foreign key mother references person)`
  
  – `set constraints father_ref, mother_ref deferred`
Complex Check Clauses

- **check** \((\text{time\_slot\_id \text{ in} (select \text{time\_slot\_id from time\_slot)})\)\)
  - why not use a foreign key here?

- Every section has at least one instructor teaching the section.
  - how to write this?

- Unfortunately: subquery in check clause not supported by pretty much any database
  - Alternative: triggers (later)

- **create assertion** \(<\text{assertion\_name}> \text{ check } <\text{predicate}>;\)
  - Also not supported by anyone
Built-in Data Types in SQL

- **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