Integrity Constraints and Authorization
What Can SQL Do?

• Define databases
  – What kinds of data? (e.g., names are character strings)
  – How to store them? (e.g., in alphabetical order)
  – Quality control (e.g., age cannot be negative)

• Retrieve data from databases
  – Which attributes are needed?
  – Which tuples should be retrieved?

• Access control (e.g., only instructors can assign final grades)

• Interface to other language / development tools (e.g., C/C++, Java, 4GL, etc.)
Outline

• User-define data types
• More on Integrity Constraints
  – NOT NULL and primary key were introduced when we discussed data storage
• Authorization in SQL
Data Types in Applications

• An application may come with various data types
  – Example: US dollars are numeric (12, 2), while Japanese Yens are integer, user-name must be at least 5 character long
  – Capturing application constraints

• Can we create data types in databases to reflect the application constraints?
User-Defined Types

• **create type** construct in SQL Server 2005 creates user-defined type
  
  ```sql
  create type Dollars from numeric (12,2) not null
  ```
  drop type Dollars

• **create domain** construct in SQL-92 creates user-defined domain types
  
  ```sql
  create domain person_name char(20) not null
  ```

• Domains can have constraints, such as not null, specified on them
Domains as Integrity Constraints

- New domains can be created from existing data types (not directly supported by SQL Server 2005)
  
  - create domain Dollars numeric(12, 2)
  - create domain Pounds numeric(12,2)

- Cannot assign or compare a value of type Dollars to a value of type Pounds
  
  - However, we can convert type (should also multiply by the dollar-to-pound conversion-rate)
    
    (cast r.A * rate as Pounds)
Large-Object Types

• Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  – **BLOB**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  – **CLOB**: character large object -- object is a large collection of character data
  – What is the correspondence in SQL Server 2005?

• When a query returns a large object, a pointer is returned rather than the large object itself
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 no less than 0
  - The salary of a bank employee must be at least $7.25 an hour
  - A customer must have a (non-null) phone number
- In what situations authorized changes to the database may result in a loss of data consistency?
Constraints on a Single Relation

- not null
- primary key
- unique
- check (P), where P is a predicate
The Uniqueness Constraint

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
  - Key: no two tuples have the same value on the key attributes

- Generally, null value is allowed in candidate key attributes
  - Primary key does not allow null value
The Check Clause

- check (P), where P is a predicate
- Example: Declare `branch_name` as the primary key for `branch` and ensure that the values of `assets` are non-negative

```sql
create table branch
  (branch_name char(15),
   branch_city char(30),
   assets integer,
   primary key (branch_name),
   check (assets >= 0))
```
The Check Clause

• The check clause in SQL-92 permits domains to be restricted
  – Use check clause to ensure that an hourly_wage domain allows only values greater than a specified value
    create domain hourly_wage numeric(5,2)
    constraint value_test check(value >= 7.25)
  – The domain has a constraint ensuring that hourly_wage is at least 7.25
  – The clause constraint value_test is optional
    • Useful to indicate which constraint an update violated
Referential Integrity

- Ensure 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 “Perryridge” is a branch name appearing in one of the tuples in the account relation, then there exists a tuple in the branch relation for branch “Perryridge”
More about Keys

• A unique clause lists attributes that comprise a candidate key
  – A candidate key can be appointed as the primary key

• The primary key clause lists attributes that comprise the primary key
  – A table can have only one primary key

• The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key
  – By default, a foreign key references the primary key attributes of the referenced table
Referential Integrity – Example

create table customer
  (customer_name  char(20),
   customer_street  char(30),
   customer_city    char(30),
   primary key (customer_name ))

create table branch
  (branch_name    char(15),
   branch_city     char(30),
   assets          numeric(12,2),
   primary key (branch_name ))
Referential Integrity – Example (2)

create table account
    (account_number char(10),
     branch_name char(15),
     balance integer,
     primary key (account_number),
     foreign key (branch_name) references branch )

create table depositor
    (customer_name char(20),
     account_number char(10),
     primary key (customer_name, account_number),
     foreign key (account_number) references account,
     foreign key (customer_name) references customer )
Assertions

• A predicate expressing a condition that we wish the database always to satisfy

  create assertion <assertion-name> check <predicate>
  – Not supported in SQL Server 2005

• When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion

  – This testing may introduce a significant amount of overhead; hence assertions should be used with great care
Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance of $1000.00

```sql
create assertion balance_constraint check
  (not exists (  
    select * from loan  
    where not exists (  
      select * from borrower, depositor, account  
      where loan.loan_number = borrower.loan_number  
      and borrower.customer_name = depositor.customer_name  
      and depositor.account_number = account.account_number  
      and account.balance >= 1000))
```
Tips: Round-about FOR-ALL

- SQL does not have a “for all X, P(X)” construct
- Asserting “for all X, P(X)” is achieved in a round-about fashion using not exists X such that not P(X)
Assertion Example (2)

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch

```sql
create assertion sum_constraint check
(not exists
  (select * from branch
   where (select sum(amount) from loan
         where loan.branch_name = branch.branch_name )
   >= (select sum (amount ) from account
        where loan.branch_name =
          branch.branch_name )))
```
Authorization

- Read - allow reading, but not modification of data
- Insert - allow insertion of new data, but not modification of existing data
- Update - allow modification, but not deletion of data
- Delete - allow deletion of data
Authorization on Schema Access

- Index - allow creation and deletion of indices
- Resources - allow creation of new relations
- Alteration - allow addition or deletion of attributes in a relation
- Drop - allow deletion of relations
Authorization in SQL

- The **grant** statement is used to confer authorization:
  
  ```sql
  grant <privilege list>
  on <relation name or view name> to <user list>
  ```

- **<user list>** is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (to be discussed later)
Authorization Rules

• Granting a privilege on a view does not imply granting any privileges on the underlying relations

• The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator)
Privileges in SQL

- **select**: allow read access to relation, or the ability to query using the view
  - Example: grant users $U_1$, $U_2$, and $U_3$ *select* authorization on the *branch* relation:
    
    ```
    grant select on branch to U_1, U_2, U_3
    ```

- **insert**: the ability to insert tuples

- **update**: the ability to update using the SQL update statement

- **delete**: the ability to delete tuples

- **all privileges**: used as a short form for all the allowable privileges
Revoking Authorization in SQL

• The `revoke` statement is used to revoke authorization

  `revoke <privilege list>`

  `on <relation name or view name> from <user list>`

  `revoke select on branch from U_1, U_2, U_3`
Summary

• Using user-defined types to capture application constraints
• Integrity constraints
• Authorization in SQL
To-do List

• Using the SQL Server online help to understand the constraint statement in SQL Server 2005

• SQL Server 2005 provides a rich set of authorization mechanisms. Please check the SQL Server online help to understand the grant/revoke statements in Transact-SQL