# Intermediate SQL

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

# Transactions

- A **transaction** consists of a sequence of query and/or update statements and is a “unit” of work
- The SQL standard specifies that a transaction begins implicitly when an SQL statement is executed
- The transaction must end with one of the following statements:
  - **Commit work**: The updates performed by the transaction become permanent in the database
  - **Rollback work:** All the updates performed by the SQL statements in the transaction are undone
- Atomic transaction
  - Either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions

# 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 Constraints

- **not null**

  – Declare *name* and *budget* to be **not null**

  *name* **varchar**(20) **not null**

  *budget* **numeric**(12, 2) **not null**

# Unique Constraints

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

# The *Check* Clause

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation

- Example: 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),
*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***

# Referential Integrity

- Foreign *keys can be* specified as part of the SQL **create table** statement

  **foreign key** (*dept_name*) **references** *department*

- By default, a foreign key references the primary key attributes of the referenced table

- SQL allows a list of attributes of the referenced relation to be specified explicitly

  **foreign key** (*dept_name*) **references** *department* (*dept_name*)

# Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation

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

- An alternative, in case of delete or update is to cascade

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

- Instead of cascade we can use:
  - **set null**
  - **set default**

# Integrity Constraint Violation During Transactions

- Consider:

**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 without causing constraint violation?

- Insert father and mother of a person before inserting person
- OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
- OR defer constraint checking

# Complex *Check* Conditions

- The predicate in the *check* clause can be an arbitrary predicate that can include a subquery

  **check** (*time_slot_id* **in** (**select** *time_slot_id* **from** *time_slot*))

- The check condition states that the time_slot_id in each tuple in the *section* relation is actually the identifier of a time slot in the *time_slot* relation
  - The condition has to be checked not only when a tuple is inserted or modified in *section*, but also when the relation *time_slot* changes

# Assertions

- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy

- The following constraints, can be expressed using assertions:
  - For each tuple in the *student* relation, the value of the attribute *tot_cred* must equal the sum of credits of courses that the student has completed successfully
  - An instructor cannot teach in two different classrooms in a semester in the same time slot

- An assertion in SQL takes the form:

  **create assertion** <assertion-name> **check** (<predicate>);

# Next Lecture

**Intermediate SQL**

# Thank you for your attention...

Any question?

**Contact:**
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
**Phone:** +91-9477678768
**E-mail:** shrutilipi@nitk.edu.in