ICS 321 Data Storage & Retrieval
Constraints, Triggers, Views & Indexes

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
PK and FK Constraints

CREATE TABLE Studio (  
  name CHAR(30) NOT NULL PRIMARY KEY,  
  address VARCHAR(255),  
  presC# INT REFERENCES MovieExec(cert#) )

CREATE TABLE Studio (  
  name CHAR(30) NOT NULL,  
  address VARCHAR(255),  
  presC# INT,  
  PRIMARY KEY(name),  
  FOREIGN KEY(presC#) REFERENCES MovieExec(cert#) )

Cert# must be declared with PRIMARY KEY or UNIQUE constraint
Maintaining Referential Integrity

CREATE TABLE Studio (  
  name CHAR(30) NOT NULL PRIMARY KEY,  
  address VARCHAR(255),  
  presC# INT REFERENCES MovieExec(cert#) )

• INSERT INTO studio VALUES (…)
• UPDATE studio SET presC#=? ...
• DELETE FROM MovieExec WHERE ...
• UPDATE MovieExec SET cert#=? ...

If new presC# value does not exist in MovieExec, reject!
If deleted cert# values are used in studio, reject!
If old cert# values are used in studio, reject!
Other Options for Referential Integrity

```sql
CREATE TABLE Studio (  
    name CHAR(30) NOT NULL PRIMARY KEY,  
    address VARCHAR(255),  
    presC# INT REFERENCES MovieExec(cert#)  
      ON DELETE SET NULL  
      ON UPDATE CASCADE )
```

- **CASCADE**: changes to referenced attributes are mimicked at FK.
- **SET NULL**: changes to referenced attributes makes affected FK null
- **DEFERABLE**: checking can wait till end of transaction
  - **INITIALLY DEFERRED**: defer checking
  - **INITIALLY IMMEDIATE**: check immediately
Check Constraints

• Attribute, tuple-based, multi-table
• Syntax: `CHECK conditional-expression`

```sql
CREATE TABLE Studio (  
    name CHAR(30) NOT NULL PRIMARY KEY,  
    address VARCHAR(255),  
    presC# INT REFERENCES MovieExec(cert#) CHECK ( presC# >=100000 ) )
```

```sql
CREATE TABLE MovieStar (  
    name CHAR(30) NOT NULL PRIMARY KEY,  
    address VARCHAR(255),  
    gender CHAR(1), birthdate DATE,  
    CHECK ( gender = ‘F’ OR name NOT LIKE ‘Ms.%’ ) )
```
Naming Constraints

CREATE TABLE Studio (
  name CHAR(30) CONSTRAINT nameiskey PRIMARY KEY,
  address VARCHAR(255),
  presC# INT REFERENCES MovieExec(cert#)
CONSTRAINT sixdigit CHECK ( presC# >=100000 )
)

ALTER TABLE Studio DROP CONSTRAINT nameiskey;

ALTER TABLE Studio ADD CONSTRAINT nameiskey
   PRIMARY KEY(name);

• Constraints can be named, so that you can refer to them in alter table statements
Constraints over Multiple Tables

• Example: number of boats + number of sailors < 100

```sql
CREATE TABLE Sailors ( sid INTEGER, sname CHAR(10),
rating INTEGER, age REAL, PRIMARY KEY (sid),
CHECK ( (SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

• When is the constraint enforced?
• What happens if the sailors table is empty?
• Think of a case when the constraint is violated but the system never catches it.
CREATE ASSERTION

• Allows constraints that are not associated with any table.
• Evaluated whenever tables in the condition are updated

CREATE ASSERTION smallClub
CHECK ( (SELECT COUNT (S.sid) FROM Sailors S) + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
Triggers

• Trigger: procedure that starts automatically if specified changes occur to the DBMS

• Three parts:
  – Event (activates the trigger)
  – Condition (tests whether the triggers should run)
  – Action (what happens if the trigger runs)
Example of a Trigger

CREATE TRIGGER youngSailorUpdate
  AFTER INSERT ON SAILORS
  REFERENCING NEW TABLE NewSailors
  FOR EACH STATEMENT
  INSERT
    INTO YoungSailors(sid, name, age, rating)
    SELECT sid, name, age, rating
    FROM NewSailors N
    WHERE N.age <= 18

- Why is “NewSailors” needed?
- What is the difference between a constraint and a trigger?
Another Example of a Trigger

- Create a trigger that will cause an error when an update occurs that would result in a salary increase greater than ten percent of the current salary.

```
CREATE TRIGGER RAISE_LIMIT
AFTER UPDATE OF SALARY ON EMPLOYEE
REFERENCING NEW AS N OLD AS O
FOR EACH ROW
WHEN (N.SALARY > 1.1 * O.SALARY)
SIGNAL SQLSTATE '75000'
SET MESSAGE_TEXT='Salary increase>10%'
```
Views

CREATE VIEW YoungActiveStudents (name, grade) AS
    SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age < 21

• A *view* is just a relation, but we store a *definition*, rather than a set of tuples.
• Views can be dropped using the **DROP VIEW** command.
• What if table that the view is dependent on is dropped?
  • **DROP TABLE** command has options to let the user specify this.
Querying Views

CREATE VIEW YoungActiveStudents (name, grade) AS
  SELECT S.name, E.grade
  FROM Students S, Enrolled E
  WHERE S.sid = E.sid and S.age < 21

SELECT name
FROM YoungActiveStudents
WHERE grade = 'A'

Conceptually, you can think of rewriting using a subquery

SELECT name
FROM (SELECT S.name, E.grade
      FROM Students S, Enrolled E
      WHERE S.sid = E.sid and S.age < 21)
WHERE grade = 'A'

Query views as with any table
Updateable Views

- In general views are not updateable. Why?
- A view on R is updateable when
  - WHERE : must not involve R in a subquery
  - FROM : only one occurrence of R and no joins.
  - SELECT : include enough attributes to fill out other attributes in R

CREATE VIEW ParamountMovies AS
  SELECT title, year
  FROM movies
  WHERE studioName=‘Paramount’

INSERT INTO ParamountMovies VALUES (‘Star Trek’, 1979)

INSERT INTO Movies ( title, year ) VALUES (‘Star Trek’, 1979)

SELECT * FROM ParamountMovies
Indexes in SQL

```sql
SELECT *
FROM Movies
WHERE studioName='Disney' AND year=1990
```

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Genre</th>
<th>studioName</th>
<th>producerC#</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10,000 rows

200 movies are made in 1990

An index on attribute A is a data structure that makes it efficient to find those tuples that have a fixed value for attribute A.
Creating Indexes

- **Clustered Index**: an index on an attribute that the tuples are sorted in.
- If a primary key is specified in the CREATE TABLE statement, an (unclustered) index is automatically created for the PK.
- To create a clustered PK index:
  - Create table without PK constraint
  - Create index on PK with cluster option
  - Alter table to add PK constraint
- To get rid of unused indexes: `DROP INDEX myIdx;`

```sql
CREATE INDEX myIdx ON mytable(col1, col3)
CREATE UNIQUE INDEX myUniqIdx ON mytable(col2, col5)
CREATE INDEX myIdx ON mytable(col1, col3) CLUSTER
```
Materialized Views

- Views can be “materialized” for efficiency
- Updating the materialized view (materialized query table in DB2) : incremental or batch

Queries on base relation may be able to exploit materialized views!