# SQL: Triggers, Views, Indexes

## SQL

- Basic SQL (queries, modifications, and constraints)

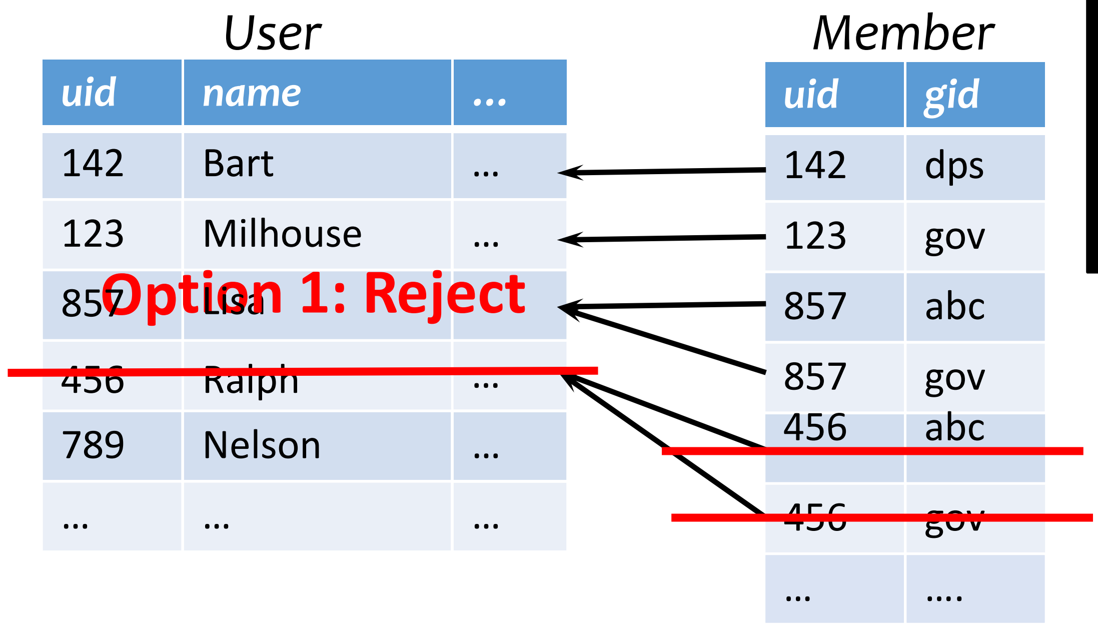
- Intermediate SQL
  - Triggers
  - Views
  - Indexes

- Advanced SQL
  - Programming
  - Recursive queries (Optional)

# Still remember "referential integrity"?

Example: *Member.uid references User.uid*

- Delete or update a *User* row whose *uid* is referenced by some *Member* row
  - Multiple Options (in SQL)



*User*

| *uid* | *name* | *…* |
|---|---|---|
| 142 | Bart | … |
| 123 | Milhouse | … |
| 857 | Lisa | … |
| ~~456~~ | ~~Ralph~~ | ~~…~~ |
| 789 | Nelson | … |
| … | … | … |

*Member*

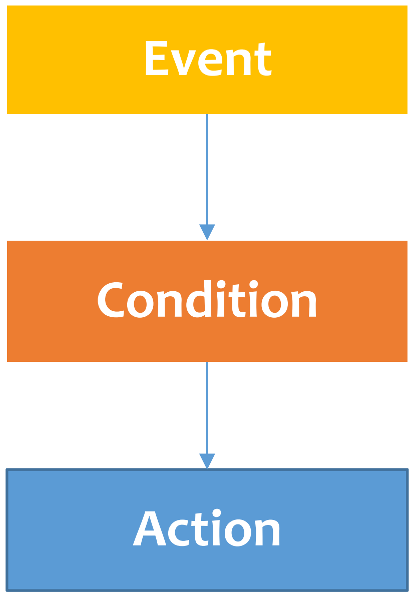
| *uid* | *gid* |
|---|---|
| 142 | dps |
| 123 | gov |
| 857 | abc |
| 857 | gov |
| ~~456~~ | ~~abc~~ |
| | |
| ~~456~~ | ~~gov~~ |
| … | …. |

**Option 1: Reject**

```
CREATE TABLE Member
(uid DECIMAL(3,0) NOT NULL
REFERENCES User(uid)
ON DELETE CASCADE,
…..);
```

**Option 2: Cascade**
(ripple changes to all referring rows)

# Can we generalize it?



Delete/update a User row

Whether its uid is referenced by some Member row

Yes: reject/delete cascade/null

Referential constraints

Some user's popularity is updated

Whether the user is a member of "S group" and *pop* drops below 0.5

Yes: kick that user out of S group

Data Monitoring

# Triggers

- A trigger is an event-condition-action (ECA) rule
  - When event occurs, test condition; if condition is satisfied, execute action

```
CREATE TRIGGER PickySGroup
AFTER UPDATE OF pop ON User
REFERENCING NEW ROW AS newUser
FOR EACH ROW
        WHEN (newUser.pop < 0.5)
                AND (newUser.uid IN (SELECT uid
                                     FROM Member
                                     WHERE gid = 'sgroup'))
                DELETE FROM Member
                WHERE uid = newUser.uid AND gid = 'sgroup';
```

Event

Transition variable

Condition

Action

# Trigger option 1 – possible events

- Possible events include:
  - INSERT ON *table*; DELETE ON *table*; UPDATE [OF *column*] ON *table*

```
CREATE TRIGGER PickySGroup
AFTER UPDATE OF pop ON User
REFERENCING NEW ROW AS newUser
FOR EACH ROW
        WHEN (newUser.pop < 0.5)
                AND (newUser.uid IN (SELECT uid
                                     FROM Member
                                     WHERE gid = 'sgroup'))
                DELETE FROM Member
                WHERE uid = newUser.uid AND gid = 'sgroup';
```

Event

Condition

Action

# Trigger option 2 – timing

- Timing—action can be executed:
  - AFTER or BEFORE the triggering event
  - INSTEAD OF the triggering event on views (more later)

```
CREATE TRIGGER NoFountainOfYouth                 Event
BEFORE UPDATE OF age ON User
REFERENCING OLD ROW AS o, NEW ROW AS n
FOR EACH ROW
        WHEN (n.age < o.age)                Condition
                SET n.age = o.age;            Action
```

# Trigger option 3 – granularity

- Granularity—trigger can be activated:
  - FOR EACH ROW modified

```
CREATE TRIGGER PickySGroup
AFTER UPDATE OF pop ON User                          Event
REFERENCING NEW ROW AS newUser
FOR EACH ROW
      WHEN (newUser.pop < 0.5)                       Condition
            AND (newUser.uid IN (SELECT uid
                                 FROM Member
                                 WHERE gid = ‘sgroup'))
            DELETE FROM Member                       Action
            WHERE uid = newUser.uid AND gid = ‘sgroup';
```

# Trigger option 3 – granularity

- Granularity—trigger can be activated:
  - FOR EACH ROW modified
  - FOR EACH STATEMENT that performs modification

```
CREATE TRIGGER PickySGroup2
AFTER UPDATE OF pop ON User
REFERENCING NEW TABLE AS newUsers
FOR EACH STATEMENT
        DELETE FROM Member
                WHERE gid = 'sgroup'
                AND uid IN (SELECT uid
                                FROM newUsers
                                WHERE pop < 0.5);
```

Event

Transition table: contains all the affected rows

Condition & Action

# Trigger option 3 – granularity

- Granularity—trigger can be activated:
    - FOR EACH ROW modified
    - FOR EACH STATEMENT that performs modification

```
CREATE TRIGGER PickySGroup2
AFTER UPDATE OF pop ON User
REFERENCING NEW TABLE AS newUsers
FOR EACH STATEMENT
          DELETE FROM Member
                    WHERE gid = 'sgroup'
                    AND uid IN (SELECT uid
                                        FROM newUsers
                                        WHERE pop < 0.5);
```

Transition table: contains all the affected rows

Only can be used with **AFTER** triggers

# Transition variables/tables

- OLD ROW: the modified row before the triggering event
- NEW ROW: the modified row after the triggering event
- OLD TABLE: a hypothetical read-only table containing all rows to be modified before the triggering event
- NEW TABLE: a hypothetical table containing all modified rows after the triggering event

| Event | Row | Statement |
|---|---|---|
| Delete | old r; old t | old t |
| Insert | new r; new t | new t |
| Update | old/new r; old/new t | old/new t |

AFTER Trigger

| Event | Row | Statement |
|---|---|---|
| Update | old/new r | - |
| Insert | new r | - |
| Delete | old r | - |

BEFORE Trigger

# SQL features covered so far

- Basic SQL

- Intermediate SQL
  - Triggers
  - Views

## Views

- A view is like a "virtual" table
  - Defined by a query, which describes how to compute the view contents on the fly
  - Stored by DBMS instead of view contents
  - Can be used in queries just like a regular table

```
CREATE VIEW SGroup AS
     SELECT * FROM User
     WHERE uid IN (SELECT uid
                   FROM Member
                   WHERE gid = 'sgroup');
```

Base tables

```
     SELECT AVG(pop) FROM SGroup;
```

```
     SELECT MIN(pop) FROM SGroup;
```

```
     SELECT … FROM SGroup;
```

```
SELECT AVG(pop)
FROM (SELECT * FROM User
        WHERE uid IN
        (SELECT uid FROM Member
         WHERE gid = 'sgroup'));
```

```
DROP VIEW SGroup;
```

## Why use views?

- To hide complexity from users
- To hide data from users
- Logical data independence
- To provide a uniform interface for different implementations or sources