SQL: Constraints, Schema modifications, Indexes

CS348

Instructor: Sujaya Maiyya

SQL features covered so far

- Query
  - SELECT-FROM-WHERE statements
  - Set and bag operations
  - Table expressions, subqueries
  - Aggregation and grouping
  - Ordering
  - Outerjoins (and NULL)
- Modification
  - INSERT/DELETE/UPDATE

☞Today: Constraints, schema changes, indexes

Constraints

- Restricts what data is allowed in a database
  - In addition to the simple structure and type restrictions imposed by the table definitions
- Why use constraints?
  - Protect data integrity (catch errors)
  - Tell the DBMS about the data (so it can optimize better)
- Declared as part of the schema and enforced by the DBMS

Types of SQL constraints

- NOT NULL
- Key
- Referential integrity (foreign key)
- Tuple- and attribute-based CHECK's

NOT NULL constraint examples

```
CREATE TABLE User
(uid INT NOT NULL,
 name VARCHAR(30) NOT NULL,
 twitterid VARCHAR(15) NOT NULL,
 age INT,
 pop FLOAT);
```

```
CREATE TABLE Group
(gid CHAR(10) NOT NULL,
 name VARCHAR(100) NOT NULL);
```

```
CREATE TABLE Member
(uid INT NOT NULL,
 gid CHAR(10) NOT NULL);
```

Key declaration examples

```
CREATE TABLE User
(uid INT NOT NULL PRIMARY KEY,
 name VARCHAR(30) NOT NULL,
 twitterid VARCHAR(15) NOT NULL UNIQUE,
 age INT,
 pop FLOAT);
```

At most one primary key per table

Any number of UNIQUE keys per table

```
CREATE TABLE Group
(gid CHAR(10) NOT NULL PRIMARY KEY,
 name VARCHAR(100) NOT NULL);
```

```
CREATE TABLE Member
(uid INT NOT NULL,
 gid CHAR(10) NOT NULL,
PRIMARY KEY(uid,gid));
```

This form is required for multi-attribute keys

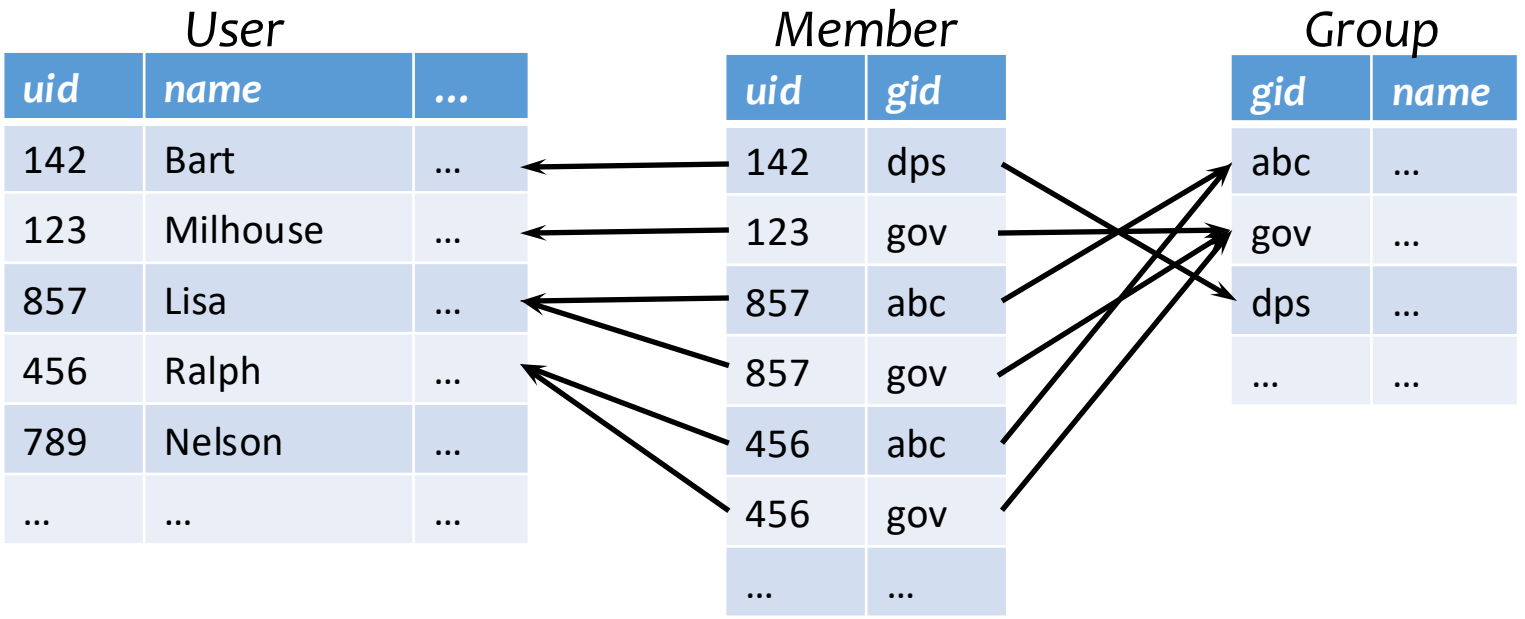
Incorrect!

```
CREATE TABLE Member
(uid INT NOT NULL PRIMARY KEY,
 gid CHAR(10) NOT NULL PRIMARY KEY,
```

Referential integrity example

- If a *uid* appears in *Member*, it must appear in *User*
  - *Member.uid* references *User.uid*
- If a *gid* appears in *Member*, it must appear in *Group*
  - *Member.gid* references *Group.gid*

☞That is, no "dangling pointers"



Referential integrity in SQL

- Referenced column(s) must be PRIMARY KEY
- Referencing column(s) form a FOREIGN KEY
- Example

Some system allow them to be non-PK but must be UNIQUE

```
CREATE TABLE Member
(uid INT NOT NULL REFERENCES User(uid),
 gid CHAR(10) NOT NULL,
PRIMARY KEY(uid,gid),
FOREIGN KEY (gid) REFERENCES Group(gid));
```

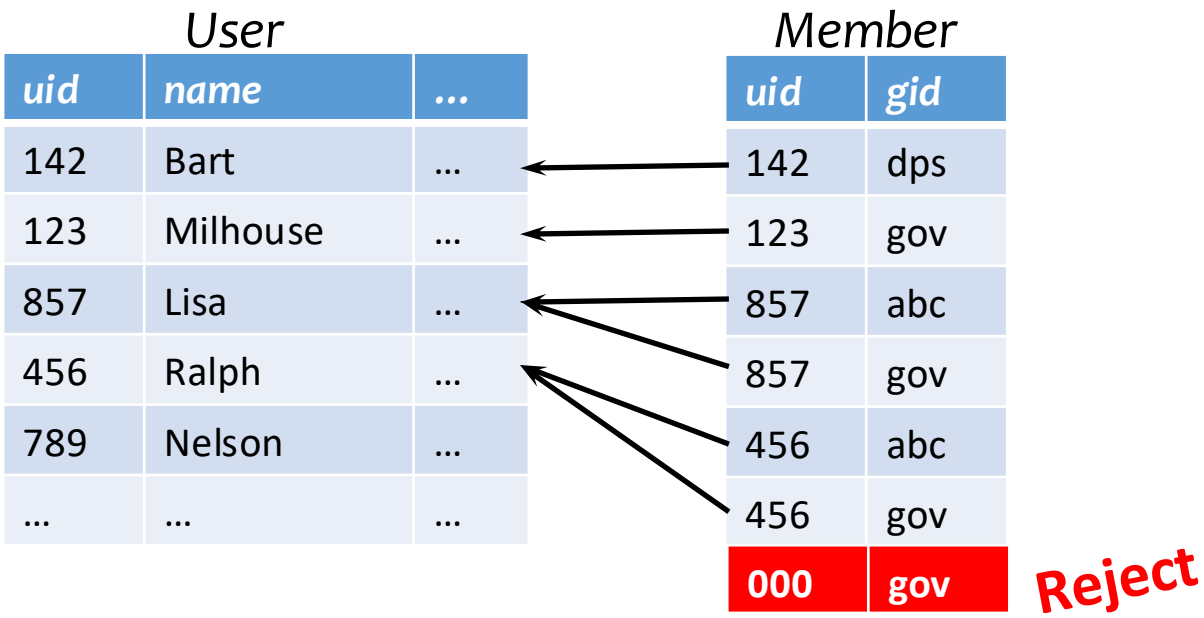
This form is required for multi-attribute foreign keys

```
CREATE TABLE MemberBenefits
(…..
FOREIGN KEY (uid,gid) REFERENCES Member(uid,gid));
```

Enforcing referential integrity

Example: *Member.uid* references *User.uid*

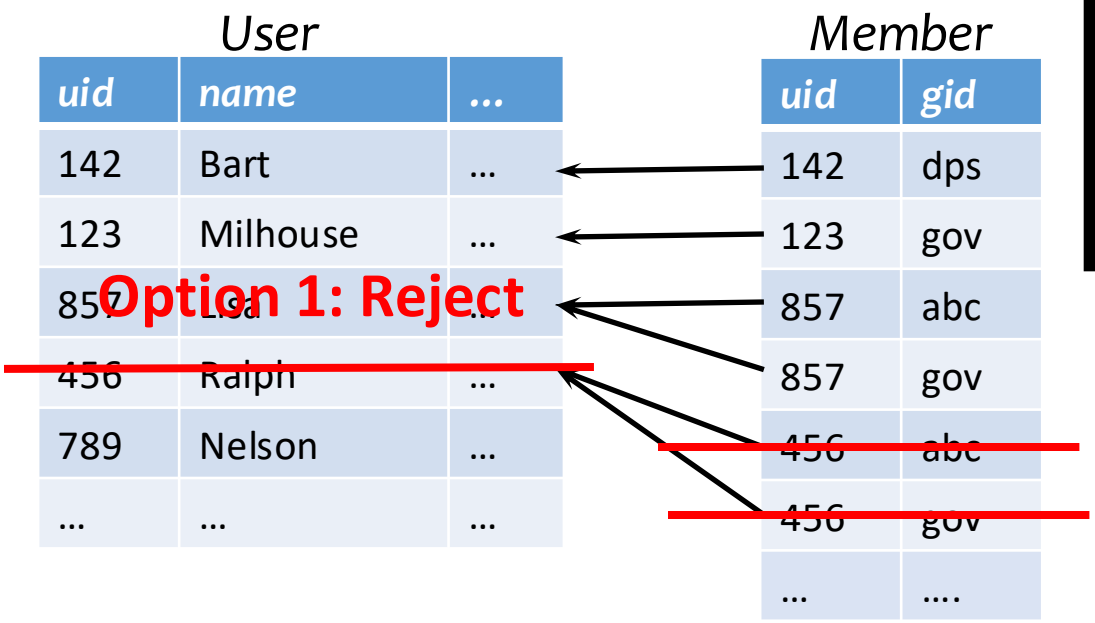
- Insert or update a *Member* row so it refers to a non-existent *uid*
  - Reject



Enforcing 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)



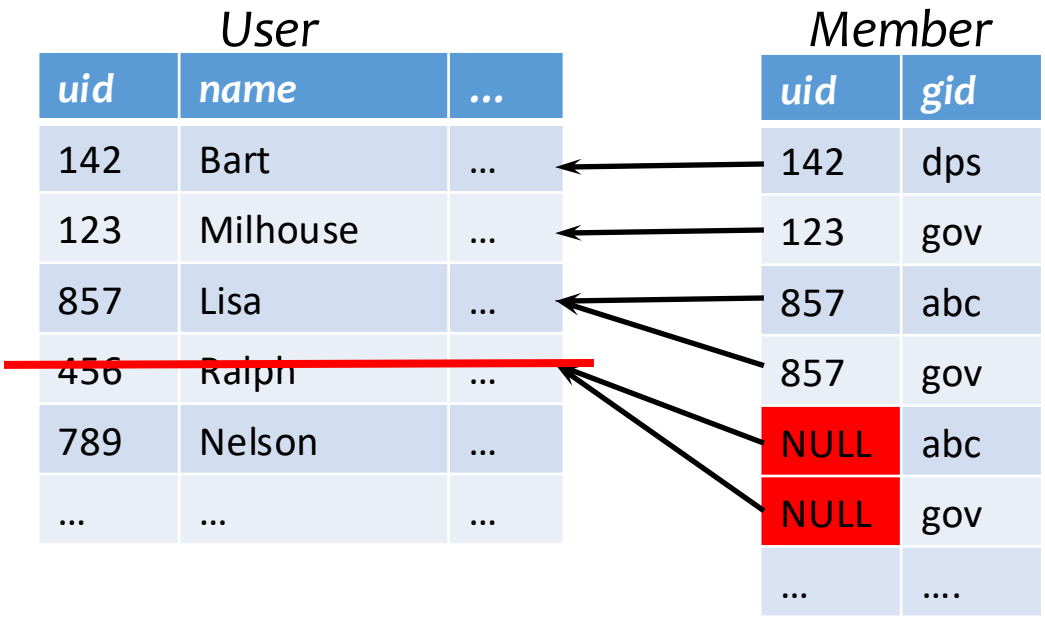

```
CREATE TABLE Member
(uid INT NOT NULL
REFERENCES User(uid)
ON DELETE CASCADE,
……);
```

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

Enforcing 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)




```
CREATE TABLE Member
(uid INT NOT NULL
REFERENCES User(uid)
ON DELETE SET NULL,
……);
```

Option 3: Set NULL
(set all references to NULL)

Tuple- and attribute-based CHECK's

- Associated with a single table
- Only checked when a tuple/attribute is inserted/updated
  - Reject if condition evaluates to FALSE
  - TRUE and UNKNOWN are fine
- Examples:

```
CREATE TABLE User(...
 age INTEGER CHECK(age > 0),
 ...);
```

```
CREATE TABLE Member
(uid INTEGER NOT NULL,
 CHECK(uid IN (SELECT uid FROM User)),
 ...);
```

Checked when new tuples are added to Member but not when User is modified

Naming constraints

- It is possible to name constraints

```
CREATE TABLE User(...
 age INT, CONSTRAINT minAge CHECK(age > 0),
 ...);
```

Exercises

Consider this db instance:

Member

uid	*gid*
857	dps
123	gov
857	abc
857	gov
456	abc
456	gov

MemberBenefits

uid	*gid*	*discount*
857	dps	10
123	gov	25
857	abc	5

- *MemberBenefits* table references the *Member* table
- (*uid*,*gid*) forms the primary key of *MemberBenefits* table
- Assume discount is of type INT (and *uid* is INT and *gid* is string with a max of 30 characters)

- Write a DDL to create the *MemberBenefits* table

Exercises

Consider this db instance:

Member

uid	*gid*
857	dps
123	gov
857	abc
857	gov
456	abc
456	gov

MemberBenefits

uid	*gid*	*discount*
857	dps	10
123	gov	25
857	abc	5

- *MemberBenefits* table references the *Member* table
- (*uid*,*gid*) forms the primary key of *MemberBenefits* table
- Assume discount is of type INT (and *uid* is INT and *gid* is string with a max of 30 characters)

```
CREATE TABLE MemberBenefits
(uid INT,
gid VARCHAR(30),
discount INT,
PRIMARY KEY (uid,gid),
FOREIGN KEY (uid,gid) REFERENCES Member(uid,gid));
```

Exercises

Consider this db instance:

Member

uid	*gid*
857	dps
123	gov
857	abc
857	gov
456	abc
456	gov

- Assume all foreign key references are set to ON DELETE SET NULL
- (Assume the db allows this, just for this exercise)

- What happens when user 857 is deleted from the *User* table? (Recall *Member* table references *uid* of *User* table)

Exercises

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Assume the User table requires *pop* column values to be between 0 and 1. Complete the following DDL statement.

```
CREATE TABLE User
(uid INT PRIMARY KEY,
 name VARCHAR(30),
 age INT,
 pop FLOAT ???);
```

Exercises

User (*uid* int, *name* string, *age* int, *pop* float)
Group (*gid* string, *name* string)
Member (*uid* int, *gid* string)

- Assume the User table requires *pop* column values to be between 0 and 1. Complete the following DDL statement.

```
CREATE TABLE User
(uid INT PRIMARY KEY,
 name VARCHAR(30),
 age INT,
 pop FLOAT CHECK(pop IS NULL OR (pop >= 0 AND pop < 1));
```

Schema modification

- How to add constraints once the schema is defined??
- Add or Modify attributes/domains
- Add or Remove constraints

Add or Modify attributes/domains

- *Alter table table_name Add column column_name*
- *Alter table table_name Rename column old_name to new_name*
- *Alter table table_name Drop column column_name*

Domain change:

- Alter table *table_name* Alter column *column_name datatype*

Error if column already has conflicting data!

Add or Remove constraints

- *Alter table table_name Add constraint constraint_name constraint_condition*

```
ALTER TABLE Member
ADD CONSTRAINT fk_user FOREIGN KEY(uid)
REFERENCES  User(uid)
```

- *Alter table table_name Drop constraint constraint_name*

```
ALTER TABLE Member
DROP CONSTRAINT fk_user
```

SQL features covered so far

Basic & Intermediate SQL

- Query
- Modification
- Constraints
- Indexes

Motivating examples of using indexes

```
SELECT * FROM User WHERE name = 'Bart';
```

- Can we go "directly" to rows with *name*='Bart' instead of scanning the entire table?

  → index on *User.name*

```
SELECT * FROM User, Member
WHERE User.uid = Member.uid AND Member.gid = 'popgroup';
```

- Can we find relevant *Member* rows "directly"?

  → index on *Member.gid*

- For each relevant *Member* row, can we "directly" look up *User* rows with matching *Member.uid*

  → index on *User.uid*

Indexes

- An index is an auxiliary persistent data structure that helps with efficient searches
  - Search tree (e.g., B⁺-tree), lookup table (e.g., hash table), etc.

  ☞ More on indexes later in this course!

- CREATE [UNIQUE] INDEX $indexname$ ON $tablename(columnname_1,\ldots,columnname_n);$
  - With UNIQUE, the DBMS will also enforce that $\{columnname_1, \ldots, columnname_n\}$ is a key of $tablename$
- DROP INDEX $indexname;$

- Typically, the DBMS will automatically create indexes for PRIMARY KEY and UNIQUE constraint declarations

Indexes

- An index on $R.A$ can speed up accesses of the form
  - $R.A = value$
  - $R.A > value$ (sometimes; depending on the index type)

- An index on $(R.A_1, \ldots, R.A_n)$ can speed up
  - $R.A_1 = value_1 \wedge \cdots \wedge R.A_n = value_n$
  - $(R.A_1, \ldots, R.A_n) > (value_1, \ldots, value_n)$ (again depends)

Questions (lecture 12):

- ☞ Ordering of index columns is important—is an index on $(R.A, R.B)$ equivalent to one on $(R.B, R.A)$?
- ☞ How about an index on $R.A$ plus another on $R.B$?
- ☞ More indexes = better performance?

SQL features covered so far

Basic & Intermediate SQL

- Query
- Modification
- Constraints
- Indexes

☞Next: Views, Triggers