

► Agenda

- Constraints
- Modification
- View
- Index

SQL: Part (III)

Spring, 2024

2

► SQL integrity constraints

Restrictions enforced on a database to maintain its integrity.

- Declared as part of the schema
- Enforced by the DBMS

Benefits of constraints

- Protect data integrity
- Enhance query efficiency

► Not NULL constraint

Types of SQL constraints

- NOT NULL
- Keys
- Referential integrity
- CHECK
- Assertion

```
CREATE TABLE Artists(  
  ID varchar(8),  
  Artist varchar(20) NOT NULL,  
  Year numeric(4,0) NOT NULL,  
  City varchar(20),  
  PRIMARY KEY (ID));
```

- Not NULL constraint prohibits the insertion of a null value for the attribute.
- SQL prohibits NULL values in the primary key of a relation.

3

4

► UNIQUE constraint

- A table can have any number of **UNIQUE constraints** of the following form.

UNIQUE (A₁, A₂, ..., A_k)

It states that the attributes A₁, A₂, ..., A_k form a **superkey**.

- At most one **PRIMARY KEY** declaration of the following form per table.
PRIMARY KEY (A₁, A₂, ..., A_k)

► Referential integrity

- Referenced attributes must be a **PRIMARY KEY**.
- Referencing attributes form a **FOREIGN KEY**.
- No **dangling pointers** from the attributes of a foreign key.

Example

```
CREATE TABLE ArtistAlbum
  Artist_ID varchar(8),
  Album_ID varchar(8),
  PRIMARY KEY (Artist_ID, Album_ID),
  FOREIGN KEY (Artist_ID) REFERENCES Artists,
  FOREIGN KEY (Album_ID) REFERENCES Albums
)
```

- Referencing relation: ArtistAlbum
- Referencing attributes: Artist_ID, Album_ID
- Referenced relations: Artist, Album

5

6

► Enforcing referential integrity

- **Reject**: reject any update that violates the referential integrity (**default option**).
- **SET NULL**: set all references to NULL.
- **CASCADE**: ripple changes to all referring rows.

Example

```
CREATE TABLE advisor (
  s_id varchar(5),
  i_id varchar(5),
  PRIMARY KEY (s_id),
  FOREIGN KEY (i_id) REFERENCES instructor (ID) ON DELETE SET NULL,
  FOREIGN KEY (s_id) REFERENCES instructor (ID) ON DELETE CASCADE);
```

Note. A foreign key is **nullable**.

7

8

► CHECK

A CHECK clause

CHECK (P)

specifies a predicate P that must be satisfied by **every tuple** in a relation.

Example

```
CREATE TABLE instructor(
  ID varchar(5),
  name varchar(20) NOT NULL,
  dept_name varchar(20),
  salary numeric(8,2) CHECK(salary > 29000),
  PRIMARY KEY(ID),
  FOREIGN KEY (dept name) REFERENCES department);
```

► Assertion

- An assertion defines a condition that we wish the database always to satisfy.
- To create an assertion in SQL, use

```
CREATE ASSERTION assertion_name CHECK (assertion_condition);
```

Example

```
-- The capacity of each course is at most 60.  
CREATE ASSERTION asse_max_capacity  
CHECK (60 >= ALL(SELECT count(*) FROM takes  
                GROUP BY course_id, year, semester));
```

9

► Update with SQL

- Insertion of new tuples into a given relation
- Deletion of tuples from a given relation.
- Updating of values in some tuples in a given relation

10

► Insert

- Insert one tuple

```
INSERT INTO instructor VALUES('10211', 'Turing', 'Comp. Sci.', 95000);  
INSERT INTO instructor(ID, name) VALUES('10222', 'Root');
```

- Insert multiple tuples

```
INSERT INTO instructor VALUES  
( '10211', 'Turing', 'Comp. Sci.', 95000),  
( '10222', 'Root', NULL, NULL);
```

- Insert tuples based on the result of a query

```
INSERT INTO instructor  
SELECT ID, name, dept_name, 18000  
FROM student  
WHERE dept_name= 'Music' AND total_cred >144;
```

11

► Delete

- Delete all instructors from the Finance department.
`DELETE FROM instructor WHERE dept_name= 'Finance';`
- Clean up the instructor table.
`DELETE FROM instructor;`
- Delete all instructors whose salary is below the avg. salary of instructors.
`DELETE FROM instructor
WHERE salary < (SELECT avg(salary) FROM instructor);`

– What really happens if a `DELETE` operation also affects the average salary ?

12

► Update

- Give a 5% salary raise to all instructors.
`UPDATE instructor SET salary = salary * 1.05;`
- Give a 5% salary raise to those instructors who earn less than 70,000.
`UPDATE instructor SET salary = salary*1.05
WHERE salary < 70000;`
- Give a 5% salary raise to instructors whose salary is less than average
`UPDATE instructor
SET salary = salary*1.05
WHERE salary < (SELECT AVG (salary) FROM instructor);`

13

► CASE statement

```
CASE  
  WHEN condition1 THEN result1  
  WHEN condition2 THEN result2  
  ...  
  WHEN conditionN THEN resultN  
  ELSE resultx  
END
```

Example

```
SELECT ID, name,  
CASE  
  WHEN grade >= 90 THEN 'A'  
  WHEN grade >= 80 AND grade < 90 THEN 'B'  
  ELSE 'C'  
END AS letter_grade  
FROM exam_grade;
```

15

► Update (cont'd)

- Increase salaries of instructors with salary over 100,000 by 3%, and all others by a 5%.
`UPDATE instructor SET salary = salary*1.03 WHERE salary > 100000;
UPDATE instructor SET salary = salary*1.05 WHERE salary <= 100000;`
– What happens if we change the update `order`?
- We can rewrite the above query with a `CASE` statement.
`UPDATE instructor
SET salary = CASE
 WHEN salary <= 100000 THEN salary * 1.05
 ELSE salary*1.03
END;`

14

► View

A `view` is like a “virtual” table. To `create a view`, use

```
CREATE VIEW view_name AS query_expr;
```

Example

```
CREATE VIEW faculty AS  
SELECT ID, name, dept_name -- the subquery that defines the view  
FROM instructor;
```

- DBMS only stores the `view definition query` instead of the view contents.
- Views can be used in queries just like regular tables.
- A view is visible to all queries once created. This differs from the `WITH` statements.
- To drop a view, use
`DROP VIEW view_name;`

16

► Using views in queries

- Create a view for computing the average salary for each department.

```
CREATE VIEW dept_avg_salary(dept_name, avg_salary) AS
SELECT dept_name, AVG(salary)
FROM instructor -- instructor is a based table
GROUP BY dept_name;
```
- Use the defined view to find the maximum average salary among all departments.

```
SELECT MAX(avg_salary) FROM dept_avg_salary;
-- replace the reference to the view by its definition
SELECT MAX(avg_salary)
FROM (SELECT dept_name, AVG(salary) AS avg_salary
      FROM instructor
      GROUP BY dept_name);
```

17

► Indexes

An **index** is an auxiliary data structure that helps to find tuples more efficiently.

- B⁺-tree index
- Hash table

Example

```
SELECT * FROM student WHERE name = 'Levy';
```

- No index on **student.name**: linearly scan the entire table **student**.
- With index: go directly to the tuples with **name = 'Levy'**.

More indexes later in this course.

19

► Materialize views

Physically store the actual table defined by a view if it is used frequently enough.

```
CREATE MATERIALIZED VIEW view_name as query_expr;
```

- Physical copy created when the view is defined.
- Such views are called **materialized view**.

Maintenance of materialized views

- As the base tables are updated, the corresponding materialized requires maintenance.
- It is possible to update a materialized view **incrementally**.

18

► Indexes (cont'd)

- Create indexes

```
CREATE INDEX index_name ON table_name(attribute_1, ..., attribute_n);
```

- Drop indexes

```
DROP INDEX index_name;
```

- Indexes take space and need to be maintained when data is updated.

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

20