COMP 4754

SQL

Fall 2024

Hafez Seliem

SQL Language

- By IBM in 1970
- Declarative!
  - Say **what** you want, not **how** to get it

- Two sublanguages:
  - DDL – Data Definition Language
    - Define and modify schema
  - DML – Data Manipulation Language
    - Queries can be written intuitively.

- RDBMS responsible for efficient evaluation.
  - Choose and run algorithms for declarative queries
    - Choice of algorithm must not affect query answer.

Example

- " BoatClub" database is to enable members of a boat club to reserve boats
  - Sailors —members of the boat club who reserve boats; and
  - Boats —boats in the club's inventory.

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Boats

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

Reserves

sid	bid	day
1	102	9/12/2015
2	102	9/13/2015

The SQL DDL

```
CREATE TABLE Sailors (
   sid    INTEGER,
   sname  CHAR(20),
   rating INTEGER,
   age    REAL,
   PRIMARY KEY (sid));

CREATE TABLE Boats (
   bid   INTEGER,
   bname CHAR(20),
   color CHAR(10),
   PRIMARY KEY (bid));

 CREATE TABLE Reserves (
   sid INTEGER,
   bid INTEGER,
   day DATE,
  PRIMARY KEY (sid, bid, day),
  FOREIGN KEY (sid) REFERENCES
   Sailors(sid),
  FOREIGN KEY (bid) REFERENCES
   Boats(bid));
```

<u>sid</u>	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

<u>bid</u>	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

<u>sid</u>	**<u>bid</u>**	**<u>day</u>**
1	102	9/12
2	102	9/13

MySQL data type categories

- Character
- Numeric
- Date and time
- Binary
- Large Object (LOB)
- Spatial
- JSON

The character types

Type	Bytes
CHAR(M)	Mx4
VARCHAR(M)	L+1

- **How the character types work (in case of default character set utf8mb4) ?**

Data type	Original value	Value stored Bytes
CHAR(2)	'CA'	8
CHAR(10)	'CA'	40
VARCHAR(10)	'CA'	3
VARCHAR(20)	'California'	11
VARCHAR(20)	'New York''New York'	9

The integer types

Type	Bytes
`BIGINT`	8
`INT`	4
`MEDIUMINT`	3
`SMALLINT`	2
`TINYINT`	1

The fixed-point type

- **DECIMAL(M, D)**
  - **M total number of digits**
  - **D number of digits in the fractional part.**
- **The floating-point types**

Type	Bytes
`DOUBLE`	8
`FLOAT`	4

- **How the fixed-point and floating-point types work ?**

Data type	Original value	Value stored	Bytes used
DECIMAL(9,2)	1.20	1.20	5
DECIMAL(9,2)	1234567.89	1234567.89	5
DECIMAL(9,2)	-1234567.89	-1234567.89	5
DECIMAL(18,9)	1234567.89	1234567.89	8
DOUBLE	1234567.89	1234567.89	8
FLOAT	1234567.89	1234570.00	4

The date and time types

Type	Bytes
DATE	3
TIME	3
DATETIME	8
TIMESTAMP	4
YEAR	1

- The **DATETIME** column does absolutely nothing concerning time zones. **TIMESTAMP** always converting to and from UTC.

```
CREATE TABLE timezone_test ( `timestamp` TIMESTAMP,
`datetime` DATETIME );
```

Example

```
SET SESSION time_zone = '+00:00’;
INSERT INTO timezone_test VALUES ('2029-02-14 08:47', '2029-02-14 08:47');
SELECT * FROM timezone_test;

Output:
-- | timestamp | datetime |
-- |---------------------|---------------------|
-- | 2029-02-14 08:47:00 | 2029-02-14 08:47:00 |
```

```
SET SESSION time_zone = '-05:00';
SELECT * FROM timezone_test;

Output:
-- | timestamp           | datetime            |
-- |---------------------|---------------------|
-- | 2029-02-14 03:47:00 | 2029-02-14 08:47:00 |
```

How MySQL interprets literal date/time values

Literal value	Stored Value
'2022-08-15'	2022-08-15
'2022-8-15'	2022-08-15
'22-8-15'	2022-08-15
'20220815'	2022-08-15
20220815	2022-08-15
'8-15-22'	(error)
'2022-02-31'	(error)

How MySQL interprets literal date/time values

-

Literal value	Stored Value
'7:32'	7:32:00
'19:32:11'	19:32:11
'193211'	19:32:11
193211	19:32:11
'19:61:11'	(error)

-

DATETIME or TIMESTAMP	Stored Value
'2022-08-15 19:32:11'	2022-08-15 19:32:11
'2022-08-15'	2022-08-15 00:00:00

How an ENUM('Yes','No','Maybe') column works

Value	Value stored	Value displayed
'Yes'	1	'Yes'
'No'	2	'No'
'Maybe'	3	'Maybe'
'Possibly'	(error)	
''	(error)	

How a SET ('Pepporoni','Mushrooms','Olives') column works ?

•Value	Valued stored (binary)	Value displayed
'Pepperoni'	1 (00000001)	'Pepperoni'
'Mushrooms'	2 (00000010)	'Mushrooms'
'Olives'	4 (00000100)	'Olives'
'Olives,Pepperoni'	5 (00000101)	'Pepperoni,Olives'
'Olives,Olives,Mushrooms'	6 (00000110)	'Mushrooms,Olives'
'Pepperoni,Sausage'	**(error)**	

The binary data types

Type	Bytes
`BINARY`(M)	M
VARCHAR(M)	L+1

The large object types

Type	Bytes
LONGBLOB	L+4
MEDIUMBLOB	L+3
BLOB	L+2
TINYBLOB	L+1
LONGTEXT	L+4
MEDIUMTEXT	L+3
TEXT	L+2
TINYTEXT	L+1

- **Terms to know about large objects**
  - BLOB (binary large object) types
  - CLOB (character large object) types

Constraints

- Recall that the schema defines the legal instances of the relations.

- Data types are a way to limit the kind of data that can be stored in a table, but they are often insufficient.

  - e.g., prices must be positive values
  - uniqueness, referential integrity, etc.

- Can specify constraints on individual columns or on tables.

Integrity Constraints

- IC conditions that every legal instance of a relation must satisfy.
  - Inserts/deletes/updates that violate ICs are disallowed.
  - Can ensure application semantics (e.g., sid is a key),

  …or prevent inconsistencies (e.g., sname has to be a string, age must be < 200)
- **Types of IC's**: Domain constraints, primary key constraints, foreign key constraints, general constraints.
  - **Domain constraints**: Field values must be of right type. Always enforced.

Primary Keys

- A set of fields is a superkey if:
  - No two distinct tuples can have same values in all these fields
- A set of fields is a key for a relation if it is minimal:
  - It is a superkey
  - No subset of the fields is a superkey
- what if >1 key for a relation?
  - One of the keys is chosen to be the primary key. Other keys are called candidate keys.
- For example:
  - sid is a key for Students.
  - What about name?
  - The set {sid, gpa} is a superkey.

Primary and Candidate Keys

- Possibly many candidate keys (specified using `UNIQUE`), one of which is chosen as the primary key.
  - Keys must be used carefully!

```
CREATE TABLE Enrolled1
  (sid   CHAR(20),
   cid   CHAR(20),
   grade CHAR(2),
   PRIMARY KEY
(sid,cid))
```

Not good either!

```
CREATE TABLE Enrolled2
   (sid   CHAR(20),
    cid   CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid,
grade))
```

"For a given student and course, there is a single grade."

Foreign Keys, Referential Integrity

- Foreign key: a “logical pointer”
  - Set of fields in a tuple in one relation
    that `refer’ to a tuple in another relation.
  - Reference to *primary* key of the other relation.

Foreign Keys in SQL

- For example, only students listed in the Students relation should be allowed to enroll for courses.
  - sid is a foreign key referring to Students:

```
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students(sid));
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B
~~11111~~	~~English102~~	~~A~~

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- *sid* in Enrolled: foreign key referencing Students.
- **Scenarios**:
  - Insert Enrolled tuple with non-existent student id?
  - Delete a Students tuple?
    - Also delete Enrolled tuples that refer to it? (CASCADE)
    - Disallow if referred to? (NO ACTION)
    - Set sid in referring Enrolled tups to a default value? (SET DEFAULT)
    - Set sid in referring Enrolled tuples to null, denoting `unknown’ or `inapplicable’. (SET NULL)

- Similar issues arise if primary key of Students tuple is updated.

Foreign keys actions

```
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students(sid)
        ON DELETE NO ACTION  );
```

vs

```
FOREIGN KEY (sid) REFERENCES Students(sid)
        ON DELETE CASCADE);
```

vs

```
FOREIGN KEY (sid) REFERENCES Students(sid)
        ON DELETE SET NULL);
```

SQL: Modification Commands

- Deletion:

```
DELETE FROM  <relation>
[WHERE  <predicate>]
```

account(bname, acct_no, balance)

- Example:

```
1.   DELETE FROM account  where
           balance > 100
     -- deletes all tuples with balance > 100


2.  DELETE FROM account
     WHERE bname IN (SELECT bname
                     FROM   branch
                     WHERE bcity = 'Bkln' )
   -- deletes all accounts from Brooklyn branch
```

account(bname, acct_no, balance)

SQL: Modification Commands

Insertion:

INSERT INTO <relation> values (.., .., ...)

or

INSERT INTO <relation>(att1, .., attn) values(..., ..., ...)

or

INSERT INTO <relation> <query expression>

account(bname, acct_no, balance)

SQL: Modification Commands

Insertion:

Examples:

```
INSERT INTO account VALUES ( ‘Perry’ , A-768, 1200)

     or
INSERT INTO account( bname, acct_no, balance)
        VALUES ( ‘Perry’ , A-768, 1200)


INSERT INTO account
      SELECT    bname, lno, 200
      FROM      loan
      WHERE     bname =  ‘Kenmore’
```

gives free $200 savings account for each loan holder at Kenmore

SQL: Modification Commands

Update:

```
UPDATE <relation>
    SET  <attribute> = <expression>
        WHERE  <predicate>
```

SQL: Modification Commands

- Example:

```
UPDATE  account
         SET       balance = balance * 1.06
         WHERE   balance > 10000

         UPDATE account
         SET       balance = balance * 1.05
         WHERE   balance <= 10000
```

Alternative:

```
UPDATE account
              SET    balance =
                         (CASE
                WHEN balance <= 10000 THEN balance*1.05
                           ELSE  balance*1.06
                          END)
```

SQL DML 1: Basic Single-Table Queries

- ```
  SELECT [DISTINCT] <column expression list>
    FROM <single table>
  [WHERE <predicate>]
  [GROUP BY <column list>
   [HAVING <predicate>]
  [ORDER BY <column list>] ;
  ```
  ```

Basic Single-Table Queries

- ```
  SELECT [DISTINCT] <column expression list>
    FROM <single table>
  [WHERE <predicate>]
  [GROUP BY <column list>
   [HAVING <predicate>]
  [ORDER BY <column list>] ;
  ```

- Simplest version is straightforward
  - Produce all tuples in the table that satisfy the predicate
  - Output the expressions in the SELECT list
  - Expression can be a column reference, or an arithmetic expression over column refs
  ```

Basic Single-Table Queries

- Example
  - SELECT S.name, S.gpa
    FROM students S
    WHERE S.dept = 'CS’

- Simplest version is straightforward
  - Produce all tuples in the table that satisfy the predicate
  - Output the expressions in the SELECT list
  - Expression can be a column reference, or an arithmetic expression over column refs

Basic Single-Table Queries

- Example:
  - ```
    SELECT DISTINCT S.name, S.gpa
      FROM students S
     WHERE S.dept = 'CS’;
    ```

- DISTINCT flag specifies removal of duplicates before output
    ```

ORDER BY

- ```
  SELECT DISTINCT S.name, S.gpa
    FROM students S
    WHERE S.dept = 'CS’
    ORDER BY S.gpa DESC, S.name ASC;
  ```

- Ascending order by default, but can be overridden
  - DESC flag for descending, ASC for ascending
  ```

Aggregates

- ```
  SELECT AVG(S.gpa)
    FROM students S
   WHERE S.dept = 'CS'
  ```

- Before producing output, compute a summary (a.k.a. an aggregate) of some arithmetic expression
- Produces 1 row of output
  - with one column in this case
- Other aggregates: SUM, COUNT, MAX, MIN
- Note: can use DISTINCT inside the agg function
  - SELECT COUNT(DISTINCT S.name) FROM Students S
  - vs. SELECT DISTINCT COUNT (S.name) FROM Students S;
  ```