SQL – Data Definition Language

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Agenda

~~SQL Introduction~~

~~Defining a Relation Schema in SQL~~

Constraints

(Integrity) Constraints

Part of the SQL standard but systems vary considerable

Impose restrictions on allowable data, beyond those imposed by structure and types

Examples

- $0.0 < GPA \leq 4.0$
- enrolment < 50,0000
- Decision: ‘Y’, ‘N’ or NULL
  $Major =' CS' \Rightarrow decision = NULL$
  $sizeHS < 200 \Rightarrow not\ admitted\ in\ colleges\ with\ enroll > 30{,}000$

Why use constraints?

Data-entry errors (inserts)

Correctness criteria (updates)

Enforce consistency

Tell system about data – store, query processing

Classification of constraints

Non-null constraints

Key constraints

Attribute-based and tuple-based constraints

Referential integrity (foreign key)

General assertions

Declaring and enforcing constraints

Declaration

- With original schema
  - Checked after bulk loading
- Or later
  - Checked at the time it's declared on the current sate of the DB

Enforcement

- Check after every modification
  - Check only the dangerous ones
  - If there is a constraint on one table, there is no need to check updates on another tables
- Deferred constraint checking
  - Instead of checking after every modification, checking is done after every transaction

Non-null constraint

Defines that a column does not have NULL values

```
CREATE TABLE <table_name> (
<column_name> <data_type> NOT NULL,
...
<column_name> <data_type>
);
```

Example

```
CREATE TABLE MovieStar (
        id          INTEGER,
        name        CHAR(30) NOT NULL,
        address     VARCHAR(255),
        gender      CHAR(1) DEFAULT '?',
        birthdate   DATE
);
```

Primary key (PK) constraint

We can define one, and only one, primary key for a table

```
CREATE TABLE <table_name> (
<column_name> <data_type> PRIMARY KEY,
...
<column_name> <data_type>
);
```

A PK cannot be NULL and cannot have repeated values

In SQLite, INTEGER PK will auto increment if a NULL value is inserted in the PK column

Example

```
CREATE TABLE MovieStar (
        id          INTEGER PRIMARY KEY,
        name        CHAR(30),
        address     VARCHAR(255),
        gender      CHAR(1) DEFAULT '?',
        birthdate   DATE
);
```

Multiple Column PK

If a primary key is composed by more than one column

```
CREATE TABLE <table_name> (
<column_name> <data_type>,
...
<column_name> <data_type>,
PRIMARY KEY (<column_name>, <column_name>)
);
```

Example

```
CREATE TABLE MovieStar (
        name        CHAR(30),
        address     VARCHAR(255),
        gender      CHAR(1) DEFAULT '?',
        birthdate   DATE,
        PRIMARY KEY (name, birthdate)
);
```

ROWID in SQLite

If a table is created without specifying the WITHOUT ROWID option, SQLite adds an implicit column called rowid that stores 64-bit signed integer

The rowid is a key that uniquely identifies the row in its table

- Can be accessed using ROWID, _ROWID_, or OID (except if ordinary columns use these names)

On an INSERT, if the ROWID is not explicitly given a value, then it will be filled automatically with an unused integer greater than 0, usually one more than the largest ROWID currently in use.

SQLite PK and ROWID

If a table has a one-column PK defined as INTEGER, this PK column becomes an alias for the rowid column

Tables with rowid are stored as a B-Tree using rowid as the key

- Retrieving and sorting by rowid are very fast
- Faster than using any other PK or indexed value

SQLite Autoincrement

Imposes extra CPU, memory, disk space, and disk I/O overhead

If an AUTOINCREMENT keyword appears after INTEGER PRIMARY KEY, that changes the ROWID assignment algorithm to prevent the reuse of ROWIDs

The purpose of AUTOINCREMENT is to prevent the reuse of ROWIDs from previously deleted rows

Should be avoided if not strictly needed

Unique key constraint

We can define multiple unique keys for a table

```
CREATE TABLE <table_name> (
<column_name> <data_type> UNIQUE,
...
<column_name> <data_type>
);
```

Unique Keys allow NULL values

The SQL standard and most DBMS do allow repeated NULL values in UNIQUE keys

Example

```
CREATE TABLE MovieStar (
        id          INTEGER PRIMARY KEY,
        name        CHAR(30),
        address     VARCHAR(255) UNIQUE,
        gender      CHAR(1) DEFAULT '?',
        birthdate   DATE,
        phone       CHAR(16) UNIQUE
);
```

Address cannot have repeated values

Phone cannot have repeated values

Multiple Column Unique Key

If a unique key is composed by more than one column

```
CREATE TABLE <table_name> (
<column_name> <data_type>,
...
<column_name> <data_type>,
UNIQUE (<column_name>, <column_name>)
);
```

Example

```
CREATE TABLE MovieStar (
        id          INTEGER PRIMARY KEY,
        name        CHAR(30),
        address     VARCHAR(255),
        gender      CHAR(1) DEFAULT '?',
        birthdate   DATE,
        UNIQUE (name, birthdate),
        UNIQUE (name, address)
);
```

Attribute-based check constraint

Constrain the value of a particular attribute

```
CREATE TABLE <table_name> (
<column_name> <data_type> CHECK <check_expression>,
...
<column_name> <data_type>
);
```

Checked whenever we insert or update a tuple

Example

```
CREATE TABLE Student (
        sID         INTEGER PRIMARY KEY,
        sName       TEXT,
        GPA         REAL CHECK (GPA<=4.0 and GPA>0.0),
        sizeHS      INTEGER CHECK (sizeHS < 5000),
);
```

GPAs must be less than or equal to 4.0 and greater than zero

The size of the high school must be less than five thousand

Tuple-based check constraint

Constrain relationships between different values in each tuple

```
CREATE TABLE <table_name> (
<column_name> <data_type>,
...
<column_name> <data_type>,
CHECK (<check_expression>)
);
```

Checked whenever we insert or update a tuple

Example

```
CREATE TABLE Apply (
        sID         INTEGER,
        cName       TEXT,
        major       TEXT,
        decision    TEXT,
        PRIMARY KEY (sID, cName, major),
        CHECK (decision='N' or cName <>'Stanford' or major <>'CS')
);
```

Either the decision is null or the college name is not Stanford or the major is not CS

There are no people who have applied to Stanford and been admitted to CS at Stanford

Example

```
CREATE TABLE person (
        id          INTEGER PRIMARY KEY,
        name        TEXT,
        salary      TEXT,
        taxes       TEXT,
        CHECK(taxes<salary)
);
```

Taxes have to be lower than salary

Subqueries and check constraints

```
CREATE TABLE Student (sID INTEGER PRIMARY KEY, sName
TEXT, GPA REAL, sizeHS INTEGER);

CREATE TABLE Apply (
        sID          INTEGER,
        cName        TEXT,
        major        TEXT,
        decision     TEXT,
        PRIMARY KEY (SID, cName, major),
        CHECK (sID in (select sID from Student))
);
```

Syntactically valid in the SQL standard but no SQL systems supports subqueries and check constraints.

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 2.3 – Defining a Relation Schema in SQL
- Section 2.5 – Constraints on Relations
- Section 7.1 – Keys and Foreign Keys
- Section 7.2 – Constraints on Attributes and Tuples
- Section 7.3 – Modification if Constraints
- Section 7.4 - Assertions