SQL - Data Definition Language

SQL

Stands for Structured Query Language Supported by all major commercial database systems Standardized - many features over time Interactive via GUI or prompt, or embedded in programs Declarative, based on relational algebra

It is a

- Data Definition Language (DDL)
    - Define relational schemata
    - Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
    - Insert/delete/modify tuples in tables
    - Query one or more tables

Relations in SQL

Tables

- Kind of relation we deal with ordinarily
- Exists in the database and can be modified as well as queried Views
- Relations defined by computation
- Not stored, constructed when needed Temporary Tables
- Constructed by the SQL processor during query exectuion and data modification
- Not stored

Data Types in SQL

- CHAR(n)
    - Stores fixed-lenght string of up to n characters
    - Normally, shorter strings are padded by trailing blanks to make n characters
- VARCHAR(n)
    - Stores variable-lenght string of up to n characters
- INT (or INTEGER)
    - For whole numbers
- SHORTINT
    - Denotes whole numbers but the bits permitted may be less (depends on the implementation)
- FLOAT (or REAL)
    - For floating-point numbers

- DECIMAL (n, d)
  - Value that consists of n decimal digits, with the decimal point assumed to be d positions from the right
- BOOLEAN
  - Denotes an attribute whose value is logic. Possible values: TRUE, FALSE and UNKNOWN
- DATE
  - DATE '1948-05-14'
- TIME

  - TIME '15:00:02.5'
  - Two and a half seconds past three o'clock

DATE and TIME are essencially character strings of a special form We may coerce dates and times to string types and do the reverse if the string "makes sense" as a data or time Different implementations may provide different presentations

Storage Classes and Data Types in SQLite

- NULL
  - The value is a NULL value
- INTEGER
  - The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value
- REAL
  - The value is a floating point value, stored as an 8-byte IEEE floating point number
- TEXT
  - The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
- BLOB
  - The value is a binary large object of data, stored exactly as it was input SQLite does not have a separate Boolean storage class. Boolean values are stored as integers. SQLite does not have a storage class for storing dates and/or times:
- TEXT as IS8601 strings ('YYYY-MM-DD HH:MM:SS.SSS')
- REAL as Julian day numbers, the number of days since noon in Greenwich on November 24, 4714 B.C.
- INTEGER as Unix Time, the number of seconds since 1970-01-01 00:00:00 UTC. SQLite has built-in date and time functions to convert between formats

Type Affinities in SQLite

To maximize compatibility between SQLite and other database engines, SQLite supports the concept of "type affinity". The type affinity of a column is the recommended type for data stored in that column. Each column has one of the following type affinities:

- TEXT
- NUMERIC
- INTEGER
- REAL
- BLOB

Determination of Type Affinities in SQLite

If the declared type contains

- INT: INTEGER affinity
- CHAR, CLOB, TEXT: TEXT affinity
- BLOB or no type specified: BLOB affinity
- REAL, FLOAT, DOUBLE: REAL affinity
- Otherwise: NUMERIC affinity

Rules should be assessed by the above order

- Text affinity
    - Storage Classes: NULL, TEXT or BLOB
    - Numerical data is converted into TEXT when inserted into a column with text affinity
- Numeric affinity
    - All storage classes
    - Text data is converted into INTEGER or REAL if conversion in lossless and to TEXT otherwise
- Integer affinity
    - Similar to the numeric affinity
- Real affinity
    - Similar to numeric affinity but forcing integer values into floating point representation
- BLOB affinity
    - No attempt to coerce data from one storage class into another

Simple Table Declarations

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

Modifying Relation Schemas

- To remove an entire table and all its tuples
    - **DROP TABLE** <table_name>;

- To modify the schema of an existing relation:
    - ALTER TABLE <table_name> ADD <colum_name> <data_type>;
    - ALTER TABLE <table_name> DROP <colum_name>;

Default Values

CREATE TABLE <table_name> (<column_name> <data_type> DEFAULT <default_value>, ... <column_name> <data_type>);

The default default value is NULL

Constraints

Part of the SQL standard but system vary considerable. Impose restrictions on allowable data, beyond those imposed by structure and types.

Classification of constraints:

- Non-null constraints
- Key constraints
- Attribute-based and tuple-base constraints
- Referencial integrity (foreign key)
- General assertions

Declaring and enforcing constraints

- Declarations
    - With original schema
        - Checked after bulk loading
    - Or later
        - Checked at the time it's declared on the current state 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
    - Deffered 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>);

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.

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 (<colum_name>, <colum_name>));

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. It 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

Multiple Column Unique Key

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

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

Tuple-based check constraint

Constrain relationships between different values in each tuple

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

Checked whenver we insert or update a tuple

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

Referential Integrity

Integrity of references, no "dangling pointers" Referential integrity from R.A to S.B - each value in column A of table R must appear in column B of table S.

- A is called the foreign key
- B is usually required to be the primary key for table S or at least unique
- Multi-attribute foreign keys are allowed Potentially violating modifications
- Insert into R

- Delete from S
- Update R.A
- Update S.B

Referential Integrity Enforcement (R.A to S.B)

- Delete from S
    - Restrict (default)
        - Generate an error, modification disallowed
    - Set Null
        - Replace R.A by NULL
    - Cascade
        - Delete tuples having a referencing value
- Update S.B
    - Restrict (default)
        - Generate an error, modification disallowed
    - Set Null
        - Replace R.A by NULL
    - Cascade
        - Do the same update to R.A

Foreign Key Declaration

CREATE TABLE <table_A> (<column_A> <data_type> PRIMARY KEY, <column_B> <data_type>, ... <column_C> <data_type>);

CREATE TABLE <table_B> (<column_X> <data_type> PRIMARY KEY, <column_Y> <data_type>, ... <column_Z> <data_type> REFERENCES <table_A> (<column_A>));

Foreign Key to Primary Key

If the referenced column is the primary key of the other table, we can omit the name of the column.

CREATE TABLE <table_A> (<column_A> <data_type> PRIMARY KEY, <column_B> <data_type>, ... <column_C> <data_type>);

CREATE TABLE <table_B> (<column_X> <data_type> PRIMARY KEY, <column_Y> <data_type>, ... <column_Z> <data_type> REFERENCES <table_A>);

Multiple Column Foreign Key Declaration

CREATE TABLE <table_A> (<column_A> <data_type>, <column_B> <data_type>, ... <column_C> <data_type>, PRIMARY KEY (<column_A>, <column_B>));

CREATE TABLE <table_B> (<column_X> <data_type> PRIMARY KEY, <column_Y>

<data_type>, ... <column_Z> <data_type>, FOREIGN KEY (<column_X>, <column_Y>) REFERENCES <table_A> (<column_A>, <column_B>));

On Delete and On Update Actions

Define actions that take place when deleting or modifying parent key values.

Use the ON DELETE and ON UPDATE clauses with one of three possible values:

- RESTRICT - prohibit operation on a parent key when there are child keys mapped to it
- SET NULL - child key columns are set to NULL
- CASCADE - propagates the operation on the parent key to each dependent child key

Enabling Foreing Key Support in SQLite

Foreign keys are disabled by default. Must be enabled for each database connection. PRAGMA foreign_keys = ON;

Constraint Naming

Naming constraints is optional but is a good practice. It makes it easier to identify the constraints when errors occur and to refer to them.

Assertions

Constraints on entire relation or entire database. Are in the SQL standard but are not supported by any database system. CREATE ASSERTION <assertion_name> CHECK ();