Bazy Danych

Tomasz Przechlewski

luty 2020

SQLite3

SQL: Declarative, case-insensitive language

DDL (Definition): `CREATE TABLE` – defines name of the table and name + type of each column

```
CREATE TABLE table_name (
   column_name columnType columnCconstraint,
```

Strona główna

Strona tytułowa

◀◀ ▶▶

◀ ▶

Strona 2 z 5

Powrót

Full Screen

Zamknij

Koniec

```
[...,]
table_constraints, [...,]   )
```

ColumnType: `TEXT`, `NUMERIC`, `INTEGER`, `REAL`, `BLOB`

ColumnConstraint:

`NOT NULL` – Ensures that a column cannot have NULL value.

`DEFAULT (v)` – Provides a default value for a column when none is specified.

`UNIQUE` – Ensures that all values in a column are different.

`PRIMARY KEY` – Uniquely identifies each row/record in a database table.

`CHECK Constraint` – Ensures that all values in a column satisfies certain conditions. `SALARY REAL CHECK(SALARY > 0)`

TableConstraint

`UNIQUE (c1, c2...)`

`PRIMARY KEY(c1, c2...)`

`CHECK (SALARY > 0)`

Note on autoincrement

By default, every row in SQLite has a special column, usually called the `rowid`, that uniquely identifies that row within the table. However if the phrase `WITHOUT ROWID` is added to the end of a `CREATE TABLE` statement, then the special `rowid` column is omitted. There are sometimes space and performance advantages to omitting the rowid.

Whenever you create a table without specifying the `WITHOUT ROWID` option, you get an implicit auto-increment column called rowid.

```
CREATE TABLE people (
  first_name TEXT NOT NULL,  last_name TEXT NOT NULL );
  INSERT INTO people (first_name, last_name) VALUES ('John', 'Doe');
  SELECT rowid, first_name, last_name FROM  people;
```

DML (Manipulation): `INSERT`, `UPDATE`, `DELETE`

```
INSERT INTO table (column1,column2 ,..)
   VALUES( value1,    value2 ,...);
```

DQL (Query): `SELECT`

```
 SELECT [DISTINCT] selectHeading
```

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 4 z 5
Powrót
Full Screen
Zamknij
Koniec

```
FROM table, table
WHERE filterExpression
GROUP BY groupingExpression
    HAVING filterExpression
ORDER BY orderingEexpression
LIMIT count
```

selectHeading – defines the result set columns and (if applicable) grouping aggregates

HAVING filterExpression – filters specific rows out of the grouping table. Requires GROUP BY

`DISTINCT` – eliminates duplicates

Note on aliases:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;
```

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Strona główna

Strona tytułowa

◀◀ ▶▶

◀ ▶

Strona 5 z 5

Powrót

Full Screen

Zamknij

Koniec

```
JOINS
  SELECT Title, Name FROM albums
  INNER JOIN artists ON artists.ArtistId = albums.ArtistId;

  with aliases (l, r)

  SELECT l.Title, r.Name FROM albums l
  INNER JOIN artists r ON r.ArtistId = l.ArtistId
```

A subquery is a SELECT statement nested in another statement. See the following statement.

```
SELECT column_1
FROM table_1
WHERE column_1 = (
   SELECT column_1
   FROM table_2
);
```

You must use a pair of parentheses to enclose a subquery. Note that you can nest a subquery inside another subquery with a certain depth.

Typically, a subquery returns a single row as an atomic value, though it may return multiple rows for comparing values with the IN operator.

You can use a subquery in the SELECT, FROM, WHERE, and JOIN clauses.

CLI extensions

```
.databases
.read FILE.sql
.q
.tables
```

Formatting output:

```
.header on
.mode (list|csv|insert)
.separator "x"
```

Writing to a file

```
.mode list
.separator ';'
.output PLIK albo .once (tylko nast. polecenie)
```

Strona główna

Strona tytułowa

◀◀ ▶▶

◀ ▶

Strona 7 z 5

Powrót

Full Screen

Zamknij

Koniec

CSV Import

```
.mode csv
.separator ';'
.import PLIK tabela
```