SQL: Part 2
DDL+

Lecture 4

Outline

1. Transactions: **BEGIN**, **COMMIT/ROLLBACK**

2. Schemas: **CREATE/DROP/ALTER**, **USE**

3. Authorization: **GRANT/REVOKE**

Caution

Note that the specific syntax/functionality of all commands in this lecture are highly dependent upon the chosen DBMS (and possibly even the version)

These slides should be taken as an overview of common options; for actual implementation you should reference DBMS documentation

Transactions

- Review: ACID

- In most DBMSs, each individual query, by default, is a transaction

- To group multiple operations:
  - Start: **BEGIN**
  - End: **COMMIT** (default) or **ROLLBACK**

Schema Specification

SQL is used to create/edit/delete a …

– Database

– Table

– Column

– Data type/domain

– Primary/foreign/unique key(s)

– Other (more later)

- Index, view
- Trigger, assertion
- User, role, privilege

Schema description is stored in the *catalog* (sometimes represented/accessible as tables)

Database

```
CREATE { DATABASE | SCHEMA }
[IF NOT EXISTS] database_name;
```

```
DROP { DATABASE | SCHEMA }
[IF EXISTS] database_name;
```

After, common to need a **USE** `database_name`
or similar statement to indicate active
database context (in multi-database DBMSs)

SQL: Part 2 (DDL+)

Table

```
CREATE TABLE [IF NOT EXISTS] table_name (
    column_name1 TYPE [OPTIONS],
    column_name2 TYPE [OPTIONS],
    {constraint},
    …
);
```

High-level notes
- If an option applies to a single column, it can go with the column; else separate entry, or sometimes separate command
- Separate elements may/not have name (for later manipulation)
- Change: `ALTER TABLE table_name ADD/ALTER/DROP …;`

SQL: Part 2 (DDL+)

Table: Common Data Types

- **BIT**
- **INT** (capacity, length, signed)
- **REAL**/**DOUBLE**/**FLOAT** (size, digits)
- **DATE**/**TIME**/**DATETIME**/**TIMESTAMP**
- **CHAR** (length)
- **VARCHAR** (length)
- **TEXT**/**CLOB**
- **BINARY**/**BLOB**

Table: Custom Data Types

- **CREATE DOMAIN**
  - Name, base type, constraint(s) via CHECK

- **CREATE TYPE**

Table: Common Column Options

- `[NOT] NULL`

- `DEFAULT <value>`

- `UNIQUE`

- `PRIMARY KEY`

- `CHECK <expr>`

- `AUTOINCREMENT`
  - DBMS-specific

SQL: Part 2 (DDL+)

Table: Keys

Separate line required if multi-column.
Optional: **CONSTRAINT** `constraint_name`

PRIMARY KEY `(c_name1, c_name2, …)`

FOREIGN KEY
`(l_c_name1, l_c_name2, …)`
REFERENCES `table_name(f_c_name1, …)`
`[`**ON** `<`**DELETE**`/`**UPDATE**`> <`**CASCADE**`/`**SET NULL**`>]`

SQL: Part 2 (DDL+)

Index (1)

- Supplementary data structure used to make some operations faster

- Defined on a sequence of field(s) of a single table
  - May optionally enforce uniqueness

- More detail in physical tuning
  - When to use, types, tradeoffs

Index (2)

```
CREATE [UNIQUE] INDEX index_name
ON table_name (c_name1, …)
[OPTIONS];
```

Notes

- Ordering of columns is VERY important
- Options often refer to the type of index being used (e.g. btree, hash, spatial – VERY important)

View

A "virtual" table defined via a SELECT query over *base* table(s) and/or other views

```
CREATE VIEW view_name
AS SELECT …;
```

Common uses

- Convenience/code re-use: if multiple queries rely upon a common data transformation
- Security: users only see the data they "need" to see (e.g. calculation/join/aggregation over base data)
- Performance: a view may optionally be **materialized** (sometimes **indexed**), meaning the DBMS actually stores its contents on disk – can reduce query time via caching complex operations/aggregations (more in physical tuning)

SQL: Part 2 (DDL+)

Assertion

Declarative constraint that is outside the scope of *implicit/explicit* constraints

Typically cross-table
  – Else **CHECK**

CREATE ASSERTION assertion_name
CHECK (multi-table expr);

Trigger

Part of an *active database* – specifies actions that automatically occur as a result of database events

Typically composed of three components
  1. Database update **event(s)**
  2. Before/after the event(s) occur, the **condition** that determines if the rule action applies
  3. The **action** to be taken, typically a set of SQL statements

```
CREATE TRIGGER trigger_name
<BEFORE/AFTER> <INSERT/UPDATE/DELETE>
ON table_name FOR EACH ROW
{body};
```

Stored Procedure/Function

- Some DBMSs support the ability to store code modules within the database, for access via SQL or library API
    - Reduces duplication
    - Decreases latency
    - More complex constraints than SQL

- **SQL/PSM** (SQL/Persistent Stored Modules) is a standard for such modules, but each DBMS varies widely
    - `CREATE FUNCTION/PROCEDURE …`

SQL: Part 2 (DDL+)

Discretionary Access Control

- Create/remove users
  - **CREATE USER** …
  - **DROP USER** …

- Grant/revoke privilege(s)
  GRANT/REVOKE <privilege list>
  ON <database/table>
  TO/FROM user

- **WITH GRANT OPTION** supports propagation of grant privilege

SQL: Part 2 (DDL+)

Summary

- You have now been exposed to a selection of SQL DDL components
  - **BEGIN**, **COMMIT/ROLLBACK**
  - **CREATE/DROP/ALTER**, **USE**
  - **GRANT/REVOKE**

- These commands are very DBMS-specific and are used to create/modify/remove…
  - Schema elements (e.g. table, column, data types)
  - Physical implementation (e.g. indexes, views)
  - Constraints (e.g. keys, assertions)
  - Access (e.g. users, privileges)

SQL: Part 2 (DDL+)