Databases

Cap. 5. SQL DDL. Constraints. Active queries

Textbook: Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

2020 UPT

Conf.Dr. Dan Pescaru

Introduction

1. SQL – Structured Query Language
2. Beginning: IBM SystemR project - SEQUEL (1970) (Structured English Query Language)
3. Oracle V2 (1979)
4. Current standard for RDBMS
5. Declarative language
6. Commercial DBMS - SQL dialects +procedural extensions (Oracle PL/SQL, Ms T-SQL …)

SQL standard

1. ANSI/ISO standard
   - SQL'1986 (SQL-87) – First formalized by ANSI
   - SQL'1992 (SQL2) – major revision (ISO 9075), most used nowadays in general purpose relational DBs
   - SQL'1999 (SQL3) – recursive queries, object-oriented types
   - SQL'2003 (SQL/XML) – introduces some XML-related features
   - SQL'2011 – adds temporal data
   - SQL'2016 – adds pattern matching support
   - SQL'2019 – multidimensional arrays (under development)

SQL Parts

1. Data Definition Language (DDL)
   - Commands for defining a database and its objects (tables, indexes, constraints)
2. Data Manipulation Language (DML)
   - Commands for changing data and querying a database
3. Data Control Language (DCL)
   - Commands that control a database, including administering privileges and committing data

Why to use SQL?

1. DDL
   - Create/delete databases, tables, and views
   - Create/delete de indexes
   - Create/delete stored procedures
2. DML
   - Insert/delete/update records
   - Querying data
3. DCL
   - Add/delete users, roles and privileges
   - Include TCL (Transactions Control Language)

SQL DDL data types

1. Dependent on specific RDBMS
2. Essential for space requirement tailoring
3. In most cases includes also a specific range
4. Includes associated operators
5. Issue: it should be mapped on client programming language data types

ANSI SQL data types

1. Character strings
   - CHARACTER(n) or CHAR(n): fixed-width n-character string
   - VARCHAR(n): variable-width string with a maximum size of n characters
2. Numbers
   - INTEGER, SMALLINT and BIGINT
   - FLOAT, REAL and DOUBLE PRECISION
   - NUMERIC(precision, scale) or DECIMAL(precision, scale)
3. Date and time
   - DATE: for date values (e.g. 2014-01-03)
   - TIME: for time values (e.g. 13:50:12)
   - TIMESTAMP: a DATE and a TIME put together in one variable (e.g. 2014-01-03 13:50:12)

Oracle SQL data types

1. Character strings
   - CHAR(size), VARCHAR2(size) , NCHAR(size), NVARCHAR2(size) – max 2000 chars (with conversion to local coding e.g. ASCII)
   - RAW, LONG RAW – for binary strings (32KB, 2GB – no conversion)
2. Numbers
   - BINARY_INTEGER (4B), BINARY_FLOAT (4B), BINARY_DOUBLE (8B) - supports the values infinity and NaN
   - NUMERIC(precision[, scale]), precision<=38
3. Date and time
   - DATE: between Jan 1, 4712 BC and Dec 31, 9999 AD
   - TIMESTAMP: includes fractional seconds precision - a number between 0 and 9 (default is 6)
4. Large objects (LOB)
   - BLOB, CLOB, NCLOB: for binary/char/Unicode up to 128TB

MySQL data types

1. Character strings
   - CHAR(size) max 255 chars, VARCHAR2(size) max 255 chars
   - TEXT – 64K chars, LONGTEXT – 4G chars
   - ENUM(x,y,z, …), SET(x,y,z, …) list or set of user identifiers
2. Numbers
   - TINYINT (1B), SMALLINT (2B), INT(size) (4B), BIGINT(size) (8B), FLOAT(size,d) (4B), DOUBLE(size,d) (8B)
   - DECIMAL(precision, scale), fixed decimal point
3. Date and time
   - DATE: supported range is from '1000-01-01' to '9999-12-31'
   - DATETIME: from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
   - TIME: from '-838:59:59' to '838:59:59', YEAR: from 1901 to 2155
   - TIMESTAMP: from '1970-01-01 00:00:01' to '2038-01-09 03:14:07'
4. Large objects (LOB)
   - BLOB, LONGBLOB: for 64KB / 4GB

Oracle – creating a new database

1. Complex process

   http://docs.oracle.com/cd/B28359_01/server.111/b28310/create003.htm#ADMIN11073

2. Several steps

   - Step 1: Specify an instance identifier (SID)
   - Step 2: Set the environment variables
   - Step 3: Choose a database administrator authentication method
   - Step 4: Create the initialization parameter File
   - Step 5: (on Windows) Create an instance
   - Step 6: Connect to the instance
   - Step 7: Create a Server Parameter File
   - Step 8: Start the instance
   - Step 9: Issue the CREATE DATABASE statement
   - Step 10: Create additional Tablespaces

Oracle – CREATE DATABASE

1. Using Oracle Managed Files:

```
CREATE DATABASE my_new_db
    USER SYS IDENTIFIED BY sys_password
    USER SYSTEM IDENTIFIED BY system_password
    EXTENT MANAGEMENT LOCAL
    DEFAULT TEMPORARY TABLESPACE temp
    UNDO TABLESPACE undotbs1
    DEFAULT TABLESPACE users;
```

MySQL – CREATE DATABASE

1. Need CREATE privilege (e.g. root):

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [ [DEFAULT] CHARACTER SET=charset_name |
     [DEFAULT] COLLATE=collation_name ]
```

- IF NOT EXISTS – avoid errors if DB already exists
- CHARACTER SET – set of allowed symbols (e.g. latin1, utf8, cp1250…)
  - for all supported CS run SHOW CHARACTER SET
- COLLATE – specify how characters are compared. E.g:
  - utf8_general_ci sorts by stripping away all accents and sorting as if it were ASCII
  - utf8_unicode_ci uses the Unicode sort order, so it sorts correctly in more languages

Oracle – CREATE SCHEMA

1. Tricky: the CREATE SCHEMA statement is used only to create objects (ie: tables, views, etc) in a schema using a single SQL statement, but does not actually create the schema itself
2. Steps to create and populate a new schema
   - STEP 1 - Create a new user (CREATE USER)
   - STEP 2 - Assign system privileges to new user (GRANT)
   - STEP 3 - Create objects in the schema (CREATE TABLE, CREATE VIEW or CREATE SCHEMA)
   - STEP 4 - Grant object privileges (GRANT)
   - STEP 5 - Create synonyms for objects (CREATE PUBLIC SYNONYM)

Oracle – CREATE USER

1. The CREATE USER statement creates a database account that allows you to log into the Oracle database

   CREATE USER user_name
   IDENTIFIED BY user_passwd
   [DEFAULT TABLESPACE tbs_perm_01]
   [QUOTA 20M ON tbs_perm_01]
   [PASSWORD EXPIRE];

   – PASSWORD EXPIRE the password must be reset before the user can log into the Oracle database

2. Basic of granting privileges

   GRANT privileges ON object TO user;

   – *privileges:* all, select, insert, update, delete …

Oracle – CREATE SCHEMA example

Step1:

```
CREATE USER smith IDENTIFIED BY smithpass
   DEFAULT TABLESPACE tbs_perm_01
   TEMPORARY TABLESPACE tbs_temp_01
   QUOTA 20M ON tbs_perm_01;
```

Step2:

```
GRANT create session TO smith;
GRANT create table TO smith;
GRANT create view TO smith;
GRANT create any trigger TO smith;
GRANT create any procedure TO smith;
GRANT create synonym TO smith;
```

Step3

```
CREATE SCHEMA AUTHORIZATION smith
      CREATE TABLE products (
      p_id NUMBER(7) not null,
      p_name VARCHAR2(32) not null,
      p_category VARCHAR2(32),
      CONSTRAINT pkc1 PRIMARY KEY(p_id));
```

Step4:

```
GRANT
      SELECT, INSERT, UPDATE, DELETE ON
      products TO smith;
```

Step5:

```
CREATE PUBLIC SYNONYM
      products
      FOR smith_schema.products;
```

Result: now can use

```
SELECT  *
      FROM products;

instead

SELECT  *
      FROM smith_schema.products;
```

MySQL – CREATE USER

1. The CREATE USER statement creates new MySQL accounts. Could be create with GRANT also

   CREATE USER user_name

   IDENTIFIED BY [PASSWORD] passwd;

   – *user_name:* syntax for account names is *user_name@host_name*. An account name consisting only of a username is equivalent to *user_name@%.* Host name could specify a domain, e.g. *user_name@%.upt.ro*

2. Grant privileges

   GRANT priv_type [(columns)], ... ON priv_obj

   TO user [IDENTIFIED BY [PASSWORD] 'passwd']

   [WITH {GRANT OPTION | MAX_QUERIES_PER_HOUR n}]

   – *priv_type* – all, create, drop, select, insert, delete, update etc.

   – *priv_obj* - *, db_name.*, db_name.tbl_name

   – GRANT OPTION – user can delegate its rights (using GRANT)

SQL – CREATE TABLE

1. It is used to create a table in a database
2. Syntax

```
CREATE TABLE table_name (
    column_name1 data_type(size) constraints,
    column_name2 data_type(size) constraints,
    ....  );
```

3. Constraints
   - NOT NULL - the column cannot store NULL value
   - UNIQUE - each row must have a unique column value
   - PRIMARY KEY - combination of NOT NULL and UNIQUE
   - FOREIGN KEY - ensure the referential integrity
   - CHECK - a specific condition for column values
   - DEFAULT - specifies a default value form column

Oracle – CREATE TABLE

```
CREATE TABLE table_name (
        column1 data_type(size) [null | not null],
        column2 data_type(size) [null | not null],
        ....
        CONSTRAINT c_name PRIMARY KEY (column1, ... column_n)
        CONSTRAINT fk_column
                FOREIGN KEY (column1, ... column_n)
                REFERENCES parent_table (column1, ... column_n)
                [ON DELETE CASCADE | ON DELETE SET NULL]
        CONSTRAINT c_name UNIQUE (column1, … column_n)
        CONSTRAINT c_name CHECK (column condition)
    );
```

- Constraints
  - a primary key can not contain more than 32 columns
  - some of the fields that are part of the unique constraint can contain null values as long as the combination is unique

Oracle – CREATE INDEX

1. By default, Oracle uses B-tree indexes. It supports also bitmap, hash [partitioned] and cluster indexes
2. Syntax

   CREATE [UNIQUE] INDEX index_name
   ON table_name (column1, ... column_n)
   [COMPUTE STATISTICS];

3. Parameters
   - UNIQUE - indicates that the combination of values in the indexed columns must be unique
   - COMPUTE STATISTICS - force Oracle to collect statistics during the creation of the index. The statistics are then used by the optimizer to choose an execution plan when SQL statements are evaluated
   - Expressions could be used instead columns (e.g. UPPER(column_i)
4. Alternative: domain index – user defined and managed indexes

MySQL – CREATE TABLE

1. Syntax

   CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
   [(column1 data_type [constraint],...)]
   [table_options]
   [partition_options]
   [select_statement]

2. Parameters
   - *data_type* supports AUTO_INCREMENT
   - TEMPORARY – it is visible only to the current connection, and is dropped automatically when the connection is closed
   - IF NOT EXISTS – suppress error from if the table exists
   - *table_options* – engine = engine_name (e.g. MyISAM –, for non-transactional tables, MEMORY – in memory storage, InnoDB – default, and BerkeleyDB for transactional tables, ref. integrity)
   - *partition_options* - by range, hash, list, composite
   - *select_statement* – create one table from another

MySQL – CREATE INDEX

1. Syntax

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    ON tbl_name (column1 [ASC | DESC], ...)
    [ USING { BTREE | HASH } ]
```

2. Parameters
   - UNIQUE – indicates that the combination of values in the indexed columns must be unique
   - FULLTEXT – for CHAR, VARCHAR, and TEXT columns. Supports boolean search (e.g. "+word" as for Google search), natural language search and query expressions
   - SPATIAL – enable the generation, storage, and analysis of geographic features (e.g. OpenGIS geometry model)

SQL – delete DB objects

1. Delete database
   MySQL: DROP DATABASE [IF EXISTS] db_name
   Oracle: DROP DATABASE db_name [INCLUDING BACKUPS]
   [NOPROMPT] (via RECOVERY MANAGER)
2. Delete table
   MySQL: DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name
   Oracle: DROP TABLE tbl_name [CASCADE CONSTRAINTS]
   [PURGE]
3. Delete index
   MySQL: DROP INDEX index_name ON tbl_name
   Oracle: DROP INDEX index_name [FORCE]
   - FORCE applies only to domain indexes. This clause drops the domain index even if the index is marked IN PROGRESS

SQL – DML active queries

1. Allow database data modification:

- Adding new data using INSERT
- Updating/correcting information using UPDATE
- Discarding unwanted data using DELETE

Example database (harbor)

- **Sailors table**

sid	name	rank	age
22	John	7	45
31	Horace	1	33
58	Andrei	8	54
71	John	9	55

- **Boats Table**

bid	name	color
101	Cleo	Blue
102	Gazelle	Red
103	Poseidon	Green

- **Reserves Table**

rid	bid	date
22	101	10/03/2014
31	102	18/10/2014
71	103	22/10/2014

SQL - INSERT

1. Add new records:

```
INSERT INTO table(field_list)
        VALUES (values_list);
INSERT INTO table
        VALUES (1:1_value_list);
INSERT INTO table
        SET field1=exp1, ... ;
```

Note: - unknown position after insertion – no guaranty to be at the end of the table

- missing attribute = NULL (error in case of *not null* field constraint was specified)

2. EX: INSERT INTO Sailors VALUES (58, "Mihai", 8, 54);

ORACLE - INSERT

1. Add multiple records:

```
INSERT ALL
  INTO Sailors (sid, name,rank,age)
        VALUES (101, 'Katy',4,32)
  INTO Sailors (sid, name,rank,age)
        VALUES (105, 'Viktor',2,27)
  INTO Sailors (sid, name,rank,age)
        VALUES (117, 'Miky',7,42);
```

2. Copy data from one table to another

```
INSERT INTO Sailors (sid, name,rank,age)
        SELECT id, n, r, a FROM Other_Sailors
```

MySQL - INSERT

```
INSERT
    [LOW_PRIORITY | DELAYED | HIGH_PRIORITY][IGNORE]
    [INTO] tbl_name {VALUES}
                    ({expr | DEFAULT}, ...),(...)
    [ ON DUPLICATE KEY UPDATE
      column_i=expr [, column_j=expr] ... ]
```

- Parameters:
  - LOW_PRIORITY execution is delayed until no other clients are reading from the table
  - DELAYED write in a buffer and return
  - HIGH_PRIORITY block all concurrent inserts
  - IGNORE ignore errors (when inserting multiple rows)
  - ON DUPLICATE KEY UPDATE cause update of the old row

SQL - UPDATE

1. Updating/correcting data:

```
UPDATE table
    SET field1=exp1, ...
    WHERE condition;
```

2. Ex:

```
UPDATE Sailors
    SET rank = rank+1
    WHERE rank<9 AND age>40;
```

3. Note: for undo `SET rank = rank – 1;`

Some expressions are not reversible!

MySQL - UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE]
    table_reference
   SET col_name1={expr1|DEFAULT} [, ...]
   [WHERE where_condition][ORDER BY ...]
   [LIMIT row_count]
```

- Parameters:
  - LIMIT places a limit on the number of rows that can be updated
  - ORDER BY the rows are updated in the order specified
  - E.g.

```
UPDATE Sailors SET sid = sid + 1; // error: duplicate PK
UPDATE Sailors SET sid = sid + 1 ORDER BY sid DESC;  //ok
```

SQL - DELETE

1. Deleting records:

   DELETE FROM table

   WHERE condition;

2. E.g.:

   DELETE FROM Sailors

   WHERE age>65;

   !!! DELETE FROM Sailors;

3. Note: no undo. Just by using transactions and ROLLBACK!

Oracle - DELETE

```
DELETE [FROM] <table>
   WHERE <condition>
   RETURNING <r_expr> INTO <var_items>;
```

- Undo through ROLLBACK (before a COMMIT)
- Parameters
  - *r_expr* - a set of expressions based on the affected row (e.g. sum(column))
  - *var_items* - a valid set of PL/SQL variables in which to load the values returned by the expressions
  - E.g. DELETE Sailors WHERE rank=8
    RETURNING avg(age) INTO :avgAge;

MySQL - DELETE

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name
    [WHERE where_condition] [ORDER BY ...]
    [LIMIT row_count];
```

- Parameters
  - The QUICK modifier affects whether index nodes are merged for delete operations. It is most useful when index values for deleted rows are replaced by similar index values from rows inserted later (empty entries are reused)
  - ORDER BY and LIMIT could be used together
  - E.g. DELETE FROM my_log WHERE user = 'joe'
    ORDER BY entry_time LIMIT 1; // just the oldest one