SQL (I)
DDL and DML data modification sentences

Databases 2018-2019

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

.

DB Query Languages

- **SQL** is the standard language for querying relational DBs (regarding queries it is mainly a superset of relational algebra).
  - Developed by IBM in the mid 70's.
  - Other query languages are: **DATALOG**, based on logic programming, and **XQUERY**, functional language for querying XML documents.
- **1979:** Oracle presented the first commercial implementation of SQL.
- **1986:** ANSI adopted SQL as standard language for relational DBMS. ISO did the same next year.
- **1992:** SQL92, the most popular standard version of the language.
- New versions of the standard were approved the following years to include language enhancements: **1999**, **2003**, **2006**, **2008**, **2011**, **2016.**

Introduction to SQL

- SQL is actually composed of several languages:
  - **Data definition language** (**DDL**): table, index creation, table structure modification, etc.
  - **Data Manipulation Language** (**DML**): `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
  - **Data Control Language** (**DCL**): User access control.
  - **Transaction Control Language** (**TCL**): `COMMIT`, `ROLLBACK`.
- Execution modes:
  - **Direct:** SQL sentences are provided to a client directly connected to an SQL server.
  - **Embedded:** SQL code is part of the source code of another host language (C, Java).
    - ⋆ A **precompiler** is used to translate embedded SQL sentences to library function calls to connect and perform queries to a DBMS.
  - **Dynamic:** SQL sentences are generated during the execution of the host program and are submitted to the DBMS as a string.

DDL – Data Definition Language

- SQL uses different **terms** to refer to the elements of a DB:
  - **Tables** correspond to the **relation schemas** and **relation instances** of the Relational Model.
  - **Columns** correspond to the **attributes** of the Relational Model.
  - **Rows** correspond to the **tuples** of the Relational Model.
- **DDL** allows us to specify the structure of the DB:
  - Definition of the **table structure** and **column** domains (types).
  - Specification of **integrity constraints:** primary keys, foreign keys, uniqueness, null values, and other constraints.
  - Furthermore, **indices** can be created on tables to speed up some queries (although this is usually done for a low-level tuning by DB administrators.)
  - Other DB objects can also be created using DDL, e.g. **sequences.**

DDL – Column domains – Basic data types

- String data types:
  - **`CHAR(n)`:** fixed length character string, right space padded to length n. Up to 2000 characters, 1 by default.
  - **`VARCHAR2(n)`:** variable length character string, up to n (4000 characters maximum).
- Numeric types:
  - **`NUMBER`:** range $-10^{125}..10^{125}$ with 38 digits of precision.
  - **`NUMBER(p,s)`:** decimal numbers, p is the **total** number of digits and s is the number of digits to the right of the decimal point.
  - **`INTEGER`:** 32-bit integer values.
  - Examples:

Assigned Value	numeric type	stored value
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	Not valid, overflows the data type
7,456,123.89	NUMBER(7,-2)	7456100

DDL – Column domains – Basic data types

- **`DATE:`** Date and time in a single column: year, month, day, hour, minute, and second (and even milliseconds.)
- From January, 1st, 4712 BC to December, 31st, 9999 AC.
- Date particular format given by the parameter `NLS_DATE_FORMAT`.
- Dates are internally stored as the number of days since some point in time. **Arithmetic operations** are allowed:

```
'1-JAN-2001' + 10                 =  '11-JAN-2001'
'27-FEB-2000' + 2                 =  '29-FEB-2000'
'10-MAY-2000' - '1-MAY-2000'      =  9
```

- Oracle has many other data types: `http://docs.oracle.com`
  - **`DECIMAL, SHORTDECIMAL, SHORTINTEGER, LONGINTEGER, NCHAR, NVARCHAR2, TIMESTAMP, BLOB, CLOB, BFILE,`** etc.

DDL – Table creation

- Table creation is done using the sentence **CREATE TABLE:**

```
CREATE TABLE TableName
 (column_1 type_1 [properties],
  column_2 type_2 [properties],
  ......
  column_n type_n [properties],
  integrity_constraint_1,
  ......
  integrity_constraint_k );
```

- Column properties:
  - **DEFAULT *value***
    Default value for this column when inserting a new row in the table.
  - **NOT NULL**
    This column does not allow null values (null values allowed by default.)
  - single-column **constraints** (see next slide.)
  - ... and other properties.

DDL – Table creation – Constraints

- **Constraints** are conditions that **must be fulfilled** by one or several columns in a table.
  - Single-column constraints can be located **next to the column definition.**
  - Multiple-column constraints **must be located at the end of the table definition.**
  - Constraints are assigned a name: it can optionally be a meaningful **name.**
- General syntax for constraints:

```
[CONSTRAINT name] constraint
```

(Square brackets are not part of the language: they stand for **optional syntax.**)

DDL – Table creation – Constraint types

- Constraint types (column names are only required for multi-column constraints):
  - **`PRIMARY KEY [(`**$C_1, \ldots, C_j$**`)]`**
    The table does not allow rows with duplicate values for $C_1, \ldots, C_j$ and they cannot contain null values.
  - **`[FOREIGN KEY (`**$C_1, \ldots, C_j$**`)] REFERENCES`** ***`table`*** **`[(`**$B_1, \ldots, B_j$**`)]`**
    Values for $C_1, \ldots, C_j$ for any row must either be **null** or they must correspond to an existing row in `table`'s primary key (or `unique` constraint).
  - **`UNIQUE [(`**$C_1, \ldots, C_j$**`)]`**
    The table does not allow rows with duplicate values for $C_1, \ldots, C_j$.
  - **`CHECK(`*`condition`*`)`**
    Boolean expression `condition` must hold for all rows in the table.

DDL – Table creation – Constraint types

- **Examples:**

```
CREATE TABLE branch
 (branch_name VARCHAR2(15) PRIMARY KEY,
  city CHAR(20) NOT NULL UNIQUE,
  assets NUMBER(12,2) DEFAULT 0,
 );

CREATE TABLE client
 (client_id VARCHAR2(9) NOT NULL,
  client_name CHAR(35) NOT NULL,
  address CHAR(50) NOT NULL,
  CONSTRAINT cl_PK PRIMARY KEY (clientId)
 );
```

DDL – Table creation – Constraint types

- **More examples:**

```
CREATE TABLE account
 (accnt_nr CHAR (20) PRIMARY KEY,
  branch_name VARCHAR2(15) REFERENCES branch,
  balance NUMBER(12,2) DEFAULT 100,
  CHECK (balance >= 100)
 );

Create table account_holder
 (client_id VARCHAR2(9) REFERENCES client,
  account_nr CHAR(20) NOT NULL,
  PRIMARY KEY (client_id, account_nr),
  FOREIGN KEY (account_nr) REFERENCES cuenta
 );
```

DDL – Table creation – Constraint enforcement

- When rows are inserted to or updated in a table **foreign key constraints must be enforced:**
  - When a row is modified and **the value of the columns of a primary/unique key change**, but it is referenced by a foreign key.
  - When a **row is deleted** and it contains a value for a primary/unique key referenced by a foreign key.
- As a general rule, when a foreign key constraint is violated, the action **is rejected and an error is raised.**
- This default behaviour can be changed for a given foreign key by means of specific clauses:
  - **`ON DELETE CASCADE:`** if a row in referenced table is deleted, all rows referencing it are deleted as well (this behaviour is not recommended!).
  - **`ON DELETE SET NULL | SET DEFAULT:`** Columns referencing deleted rows are set to **`NULL`** or their default values.
  - **`ON UPDATE CASCADE | SET NULL | SET DEFAULT.`**

DDL – Table creation – Constraint enforcement

- **Examples:**

```
CREATE TABLE account
 (account_nr CHAR (20) PRIMARY KEY,
  branch_name VARCHAR2(15)
    REFERENCES branch ON DELETE SET NULL,
  balance NUMBER(12,2) DEFAULT 100,
  CHECK (balance >= 100)
 );

CREATE TABLE account_holder
 (client_id VARCHAR2(9) REFERENCES client ON DELETE CASCADE,
  account_nr CHAR(20) NOT NULL,
  PRIMARY KEY (client_id, account_nr),
  FOREIGN KEY (account_nr)
    REFERENCES account ON DELETE CASCADE
 );
```

DDL – Changing structure of existing tables

We can change the structure of an existing table containing data (to some extent):

- Add a collumn to a table:

  ```
  ALTER TABLE table ADD column domain [properties];
  ```

- Remove a column from a table:

  ```
  ALTER TABLE table DROP COLUMN column;
  ```

  A column participating in a constraint cannot be dropped, unless the following is used:

  ```
  ALTER TABLE tabl3 DROP column CASCADE CONSTRAINTS;
  ```

- Change a column in a table (to some extent, e.g., for enlarging the domain):

  ```
  ALTER TABLE table MODIFY (column domain [properties]);
  ```

- Rename columns in a table:

  ```
  ALTER TABLE tabl3 RENAME COLUMN column TO newName;
  ```

DDL – Changing structure of existing tables

- Add constraints to a table:

```
ALTER TABLE table ADD CONSTRAINT name Type (columns);
```

- Remove constraints from a table:

```
ALTER TABLE table DROP PRIMARY KEY;
ALTER TABLE table DROP UNIQUE(columns);
ALTER TABLE table DROP CONSTRAINT name [CASCADE];
```

  Option **CASCADE** makes dependant constraints are removed as well.

- Disable constraints:

```
ALTER TABLE table DISABLE CONSTRAINT name [CASCADE];
```

- Enable constraints:

```
ALTER TABLE table ENABLE CONSTRAINT name;
```

DDL – Changing structure of existing tables

- **Examples:**

```
ALTER TABLE account ADD comission NUMBER(4,2);

ALTER TABLE account ADD opening_date DATE;

ALTER TABLE account DROP COLUMN branch_name;

ALTER TABLE account MODIFY comission DEFAULT 1.5;

ALTER TABLE client MODIFY client_name NULL;

ALTER TABLE branch ADD CONSTRAINT cd_UK UNIQUE(city);
```

DDL – Other operations on tables

- Show table description.

  ```
  DESCRIBE table;
  ```

- Remove a table.

  ```
  DROP TABLE table [CASCADE CONSTRAINTS];
  ```

  Option **CASCADE** removes dependant constraints.

- Rename a table.

  ```
  RENAME TABLE table TO newName;
  ```

- Delete table contents.

  ```
  TRUNCATE TABLE table;
  ```

DDL – Sequences

- **Sequences** are used to automatically generate different numbers every time is required.
- The main idea of this DB object is to be able to **atomically** generate new different values from possibly concurrent sessions.
- Sequences are independent from the tables that use them.
- Sequence creation:

```
CREATE SEQUENCE seq_name [INCREMENT BY m]
  [START WITH n] [MAXVALUE p|NOMAXVALUE]
  [MINVALUE q|NOMINVALUE] [CYCLE|NOCYCLE];
```

DDL – Sequences

- There are two functions (methods) to apply to sequences:
  - **NEXTVAL** increments the sequence's current value and returns it.
  - **CURRVAL** just returns the sequence's current value.
- They can be used anywhere a numeric expression is expected (except `DEFAULT` clauses).
- Sequences' parameters can be changed after using them, but the change only affects values generated after the changed.
- **Example:**

```
CREATE SEQUENCE sq_client_id INCREMENT BY 1
  START WITH 10 MAXVALUE 200000;
SELECT sq_client_id.CURRVAL FROM DUAL;
INSERT INTO client (client_id,name)
  VALUES (sq_client_id.NEXTVAL,'John Doe');
```

DDL – Indices

- **Indices** are used by DBMS to speed up query and sorting operations on those columns referenced by the index (they are implemented internally as a B-tree or a bitmap).
  - Nevertheless, they slowdown data modification operations (`UPDATE`, `INSERT`).
- Most indices are automatically created, as a result of **`PRIMARY KEY`** and **`UNIQUE`** **constraints**.
- We can create additional indices for those columns that will be frequently used for querying data or sorting results.

```
CREATE [UNIQUE] INDEX indexName
  ON tableName(col1,...,colk);
```

- For instance, it may be convenient to create indices for foreign keys, if many queries on these column combinations are expected.
- Anyway, index creation is a DB administrator task.

DML – Data Manipulation Language

- The **Data Manipulation Language (DML)** consists of four main sentences:
- **`INSERT`** for inserting rows in a table.
- **`DELETE`** for deleting existing rows from a table. tabla.
- **`UPDATE`** for changing the contents of existing rows in a table.
- **`SELECT`** for making queries to tables.

DML – **INSERT**.

- Sentence **INSERT** adds one or several new rows to a table.
- The basic **syntax for inserting a single row** is:

```
INSERT INTO tableName [(col1,...,colk)]
  VALUES (val1,...,valk);
```

- The order in the list of column names *col1,...,colk* must match the **order of the list of values *val1,...,valk***.
  If list of column names is omitted, the list of values must match the order of the column definition when creating the table.
- **Example:**

```
INSERT INTO client VALUES (37,'John Doe','5th Main St.');
```

- **Literals** must be enclosed using **single quotations.**
- **Alternate Syntax:** We can use a `SELECT` query to generate the rows to insert (More on this later):

```
INSERT INTO client SELECT * FROM mailingClients WHERE ...;
```

DML – **DELETE**.

- Sentence **DELETE** erases existing rows that satisfy a Boolean condition from a table.
- **Syntax:**

```
DELETE FROM tableName WHERE condition;
```

- `Condition` in the **WHERE** clause is evaluated **for each row.** If it is true, the row is erased.
- The `WHERE` clause is **optional,** but if it is omitted it is evaluated as true for every row: all rows in the table are erased.
- The **WHERE** clause may contain complex expressions, and even nested subqueries.
  - We will study it in detail when will see the **SELECT** sentence.
- **Example:**

```
DELETE FROM client WHERE client_id = 37;
```

DML – **UPDATE**.

- Sentence **UPDATE** changes the values of the existing rows in a table that satisfy a Boolean condition.
- **Syntax:**

```
UPDATE tableName SET col1=expr1,..., colk = exprk
  WHERE condition;
```

- `Condition` in the **WHERE** clause is evaluated **for each row.** If it is true, the row is changed according to the assignments in the **SET** clause.
- If the **WHERE** clause is omitted, all rows in the table are changed.
- The **WHERE** clause may contain complex expressions, and even nested subqueries.
- **Examples:**

```
UPDATE employee SET salary=salary*1.1
                WHERE deptId='IS';
UPDATE account SET balance=balance*1.03, commission=0.25
               WHERE branch_name='UCM';
```