# An Introduction to Database Systems

## chapter 4. An Introduction to SQL



spark@dblab.sogang.ac.kr

# 4.1 Introduction

- **SQL2 : "SQL/92" → SQL3(2000)**
  - **International Standard Database Language SQL(1992)**
- **Remarks**
  - **SQL was originally intended to be a data sublanguage. With the incorporation of the persistent stored Modules feature(PSM) (late 1996), SQL became computationally complete.**
    - **CALL, RETURN, SET, CASE, IF, LOOP, LEAVE, WHILE, REPEAT, as well as several related features such as variables and exception handlers**
  - **SQL was the term table in place of both of the terms relation and relvar.**
  - **SQL is an enormous language**
    - **Standard document > 600 pages, SQL3 > 1200 pages**
  - **SQL is very far from being the perfect relational language**



spark@dblab.sogang.ac.kr

# 4.2 Overview (CONT.)

## □ Definitional operations

**Fig.4.1 The suppliers and parts database (SQL definition)**

```
CREATE TABLE S
    ( S#                CHAR(5),
      SNAME             CHAR(20),
      STATUS            NUMERIC(5),
      CITY              CHAR(15),
     PRIMARY KEY        (S#) ) ;

CREATE TABLE P
    ( P#                CHAR(6),
      PNAME             CHAR(20),
      COLOR             CHAR(6),
       WEIGHT           NUMERIC(5, 1),
      CITY              CHAR(15),
    PRIMARY KEY         (P#) ) ;
```

```
CREATE TABLE SP
    ( S#                CHAR(5),
      P#                CHAR(6),
      QTY               NUMERIC(9),
    PRIMARY KEY         (S#, P#) ),
    FOREIGN KEY         (S#)  REFERENCES S,
    FOREIGN KEY         (P#)  REFERENCES P ;
```



spark@dblab.sogang.ac.kr

# 4.2 Overview (CONT.)

## □ CREATE TABLE statement

- SQL does not permit users to define their own types. Columns must be defined in terms of built-in (system-defined) types only :

| | | | | |
|---|---|---|---|---|
| CHARACTER | [VARYING] | (n) | INTEGER | DATA |
| BIT | [VARYING] | (n) | SMALLINT | TIME |
| NUMERIC | (p, g) | | FLOAT (p) | TIMESTAMP |
| DECIMAL | (p, g) | | | INTERVAL |

- lengths or precisions → integrity constraints



spark@dblab.sogang.ac.kr

# 4.2 Overview (CONT.)

## □ Manipulate operations

- SELECT, INSERT, UPDATE, DELETE

**Restrict :**

```
SELECT S#, P#, QTY
FROM   SP
WHERE  QTY <150 ;
```

**Result :**

| S# | P# | QTY |
|---|---|---|
| S1 | P5 | 100 |
| S1 | P6 | 100 |

**Project :**

```
SELECT S#, CITY
FROM   S;
```

**Result :**

| S# | CITY |
|---|---|
| S1 | London |
| S2 | Paris |
| S3 | Paris |
| S4 | London |
| S5 | Athens |



spark@dblab.sogang.ac.kr

# 4.2 Overview (CONT.)

- **Manipulate operations**
  - **SELECT, INSERT, UPDATE, DELETE**

**Join :**

```
SELECT S.S#, SNAME, STATUS, CITY, P#, QTY
FROM    S, SP
WHERE  S.S# = P.P# ;
```

**Result :**

| S# | SNAME | STATUS | CITY | P# | QTY |
|---|---|---|---|---|---|
| S1 | Smith | 20 | London | P1 | 300 |
| S1 | Smith | 20 | London | P2 | 200 |
| S1 | Smith | 20 | London | P3 | 400 |
| … | … | … | … | … | … |
| S4 | Clark | 20 | London | P5 | 400 |



spark@dblab.sogang.ac.kr

# 4.2 Overview (CONT.)

## □ Manipulate operations

- Qualified name (S.S#, SP.S#)
- A shorthand form

```
SELECT *                or      SELECT S.*
FROM S                          FROM S
```

- INSERT, UPDATE, DELETE – set level operations

```
INSERT
INTO      TEMP (P#, WEIGHT)
          SELECT P#, WEIGHT
          FROM   P
          WHERE COLOR='Red';
```

```
UPDATE    S
SET       STATUS = STATUS * 2
WHERE     CITY = 'Paris" ;
```

```
DELETE
FROM      SP
WHRE      P# = 'P2' ;
```

- SQL does not include a direct analog of the relational assignment operation
  - INSERT… SELECT…



spark@dblab.sogang.ac.kr

# 4.3 The Catalog (CONT.)

- SQL standard : Information schema
- SQL catalog consists of the descriptors for an individual database
- SQL schema consists of the descriptors for that portion of that database that belongs to some individual user
- Database → any number of catalogs → each divided up into an number of schemas

INFORMATION_SCHEMA



spark@dblab.sogang.ac.kr

# 4.3 The Catalog (CONT.)

- **Information Schema**
  - A set of SQL tables whose contents echo all of the definitions from all of the other schemas in the catalog
  - A set of views of a hypothetical “Definition Schema”
- **Implementation is required**
  a. to support some kind of Definition Schema and
  b. to support views of that Definition Schema
- **Rationale**
  1. Existing products do support something akin to the Definition Schema. However, those Definition Schemas vary widely from one to another
  2. an (not the) Information Schema



spark@dblab.sogang.ac.kr

# 4.3 The Catalog (CONT.)

- **List some of the more important Information Schema Views**

| | |
|---|---|
| SCHEMATA | REFERENTIAL_CONSTRAINTS |
| DOMAINS | CHECK_CONSTRAINTS |
| TABLES | KEY_COLUMN_USAGE |
| VIEWS | ASSERTIONS |
| COLUMNS | VIEW_TABLE_USAGE |
| TABLE_PRIVILEGES | VIEW_COLUMN_USAGE |
| COLUMN_PRIVILEGES | CONSTRAINT_TABLE_USAGE |
| USAGE_PRIVILEGES | CONSTRAINT_COLUMN_USAGE |
| DOMAIN_CONSTRAINTS | CONSTRAINT_COLUMN_USAGE |
| TABLE_CONSTRAINTS | CONSTRAINT_DOMAIN_USAGE |



spark@dblab.sogang.ac.kr

# 4.4 Views (CONT.)

- an SQL view

```
CREATE VIEW     GOOD_SUPPLIER
  AS      SELECT  S#, STATUS, CITY
          FROM    S
          WHERE   STATUS > 15 ;
```

- An SQL query against this view

```
SELECT  S#, STATUS
FROM    GOOD_SUPPLIER
WHERE   CITY = 'London' ;
```



spark@dblab.sogang.ac.kr

# 4.4 Views (CONT.)

- **Substituting the view definition to the view name**

```
SELECT  GOOD_SUPPLIER.S#, GOOD_SUPPLIER.STATUS
FROM    ( SELECT  S#, STATUS, CITY
          FROM    S
          WHERE   STATUS > 15 ) AS GOOD_SUPPLIER
WHERE   GOOD_SUPPLIER.CITY = 'London' ;
```

- **Simplify it**

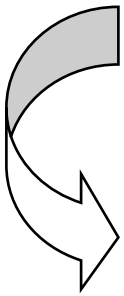
```
SELECT  S#, STATUS
FROM    S
WHERE   STATUS > 15
AND     CITY = 'London' ;
```



spark@dblab.sogang.ac.kr

# 4.4 Views (CONT.)

## □ DELETE operation



```
DELETE
FROM    GOOD_SUPPLIER
WHERE   CITY = 'London' ;

DELETE
FROM    S
WHERE   STATUS > 15
AND     CITY = 'London' ;
```



spark@dblab.sogang.ac.kr

# 4.5 Transactions

- **SQL's analog of COMMIT and ROLLBACK**
  - **COMMIT [WORK], ROLLBACK[WORK]**

- **BEGIN TRANSACTION (X)**
  - **A transaction is begun implictly whenever the program executes a transaction-initiating statement**



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **dual-mode principle**
  - any SQL statement that can be used interactively can also be used in an application program
- **Embedded SQL statement**
  - are prefixed EXEC SQL
  - are terminated special terminator
  - can include references to host variables
- **executable statement**
  - can appear wherever an executable host statement can appear
  - but, not declarative statement (DECLARE CURSOR, BEGIN , END, DECLARE SECTION, etc.. )



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **INTO clause**
  - data type compatibility
- **all host variable will be referenced in SQL**
  - must be defined within an embedded SQL declare section
- **SQLSTATE**
  - every SQL program must include a host variable, SQLSTATE
  - 00000 :executed successfully
  - 02000 : executed but no data was found to satisfy the request
- **WHENEVER**
  - EXEC SQL WHENEVER condition action terminator
  - condition : SQLERROR(error occurred), NOT FOUND(0200)
  - action : CONTINUE, GO TO



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- example

```
EXEC SQL BEGIN DECLARE SECTION;
      DCL SQLSTATE    CHAR(5);
      DCL P#          CHAR(6);
      DCL WEIGHT      FIXED DECIMAL(3);
EXEC SQL END DECLARE SECTION ;

P# = 'P2';                                        /* for example        */
EXEC SQL     SELECT P.WEIGHT
                    INTO      :WEIGHT
                    FROM P
                    WHERE P.P# = :P#;
IF SQLSTATE = '00000'
THEN         ..... ;                  /* WEIGHT = retrieved value */
ELSE         .... ;                   /* some exception occurred */
```



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **operations not involving cursor**

**Singleton SELECT :** Get status and city for the supplier whose supplier number is given by the host variable GIVENS#

```
EXEC SQL   SELECT  STATUS, CITY
           INTO    :RANK, :CITY
           FROM    S
           S# = :GIVENS#;
```

**INSERT :** Insert a new part into table P

**(part no., name, and weight given by host variable P#, PNAME,PWT ; color and city unknown)**

```
EXEC SQL INSERT
           INTO P ( P#, PNAME, WEIGHT)
           VALUSE (:P#, :PNAME, :PWT)
```



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

**UPDATE :** increase the status of all London suppliers by the amount given by the host variable RAISE

```
EXEC SQL UPDATE S
                SET STATUS = STATUS + :RAISE
                WHERE CITY = 'London';
```

**If no rows satisfy the WHERE, SQLSTATE will be set to 02000**

**DELETE :** Delete all shipments for suppliers whose city is given by the host variable CITY

```
EXEC SQL   DELETE
           FROM   SP
           WHERE :CITY = ( SELECT CITY
                           FROM S
                           WHERE S.S# = SP.S# ) ;
```



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- operations Involving Cursors
- declare

```
EXEC SQL   DECLARE cursor CURSOR
           FOR <table-expression>
           [ ORDER BY order-item-commalist ]
```

- executable statements to operate on CURSOR
  - OPEN, FETCH, CLOSE



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

## □ Multi-row retrieval example

```
EXEC SQL DECLARE X CURSOR FOR /* define the cursor */
                SELECT    S.S#, S.SNAME, S.STATUS
                FROM      S
                WHERE     S.CITY = :Y
                ORDER BY S# ASC ;

EXEC SQL OPEN X ;                              /* execute the query */
                DO  for all S rows accessible via X;
                          EXEC SQL FETCH X INTO :S#, :SNAME, :STATUS;
                          .....                          /* fetch next supplier */
                END;
EXEC SQL        CLOSE X;                       /* deactivate cursor X */
```



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **dynamic SQL**
  - **a set of embedded SQL facilities that are provided specifically to allow the construction of generalized, online, and possibly interactive applications.**
- **online application**
  1. **Accept a command from the terminal**
  2. **Analyze that command**
  3. **Execute appropriate SQL statements on the database**
  4. **Return a message and/or results to the terminal**



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **dynamic statements**
  - **PREPARE and EXECUTE**
  - **example**

```
DCL SQLSOURCE CHAR VARYING (65000);
SQLSOURCE = 'DELETE FROM SP WHERE QTY <300';
EXEC SQL PREPARE SLQPREPPED FROM :SQLSOURCE;
EXEC SQL EXECUTE SQLPREPPED;
```



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **SQLSOURCE**
  - identifies a PL/I  a varying length character string variable
- **SQLPREPPED**
  - identifies an SQL variable, not a PL/I variable, that will be used to hold the compiled form of the SQL statement whose source form is given in SQLSOURCE
- **assignment " SQLSOURCE = ..."**
  - assigns to SQLSOURCE the source form of an SQL DELETE statement
- **PREPARE**
  - "prepares"(i.e., compiles) it to produce an executable version, which it stores in SQLPREPPED
- **EXECUTE**
  - executes that SQLPREPPED version and thus causes the actual DELETE to occur



spark@dblab.sogang.ac.kr

# 4.6 Embedded SQL (CONT.)

- **SQL Call-level interface (SQL/CLI, CLI)**
  - **based on MS ODBC**
  - **CLI permits an application written in one of the usual host languages to issue database requests by invoking certain vendor-provided CLI routines**
  - **address the same general problem as dynamic SQL does.**
- **SQL/CLI represent a better approach than dynamic SQL does**
  1. **Dynamic SQL is a source code standard → require the srvices of some kind of SQL complier (PREPARE, EXECUTE).**

  **CLI merely standardize the details of certain routine invocations. (no special complier) (shrink-wrapped object code form)**

  2. **Those applications can be DBMS-independent.**



spark@dblab.sogang.ac.kr

# 4.7 SQL in not Perfect

- There is no product on the market today that supports every detail of the relational model



spark@dblab.sogang.ac.kr