

Database System

北京交通大学软件学院

王方石 教授

[E-mail: fshwang@bjtu.edu.cn](mailto:fshwang@bjtu.edu.cn)

Chapter 3-Contents

3.1 Introduction to SQL

3.2 Data Definition Statements

3.3 Data Query Statements

3.4 Data Modification Statements

3.5 Views

3.6 Programmatic SQL

3.4 Data Modification Statements

3.4.1 INSERT statement

(1) INSERT INTO ... VALUES

**INSERT INTO TableName [(columnList)]
VALUES (dataValueList)**

- Insert a new tuple into the table.
- *columnList* is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been **declared as NULL** when table was created, unless **DEFAULT was specified** when creating column.

INSERT

**INSERT INTO TableName [(columnList)]
VALUES (*dataValueList*)**

***dataValueList* must match *columnList* as follows:**

- ◆ The **number of items** in each list must be same;
- ◆ **data type** of each item in *dataValueList* must be **compatible** with data type of corresponding column;
- ◆ It must be direct correspondence in **position** of items in two lists.

Example 3.41 INSERT ... VALUES

**Insert a new row into Staff table
supplying data for all columns.**

**Staff (staffNo, fName, IName, position,
sex, DOB, salary, branchNo)**

INSERT INTO Staff

**VALUES ('SG16', 'Alan', 'Brown', 'Assistant',
'M', Date'1957-05-25', 8300, 'B003');**

Example 3.41 INSERT using Defaults

Insert a new row into Staff table supplying data for all mandatory columns.

Staff (staffNo, fName, IName, position,
sex, DOB, salary, branchNo)

```
INSERT INTO Staff (staffNo, fName, IName,  
                  position, salary, branchNo)  
VALUES ( 'SG44', 'Anne', 'Jones',  
        'Assistant', 8100, 'B003');
```

- Or

```
INSERT INTO Staff  
VALUES ('SG44', 'Anne', 'Jones', 'Assistant',  
      NULL, NULL, 8100, 'B003');
```

(2) INSERT ... SELECT

- The second form of INSERT allows multiple rows to be copied **from the query result** of one or more tables **to another**:

```
INSERT INTO TableName [ (columnList) ]  
SELECT ...
```

Example 3.42 INSERT ... SELECT

Assume there is a table **StaffPropCount** that contains names of staff and the number of properties they manage:

StaffPropCount (staffNo, fName, lName, propCnt)

Populate **StaffPropCount** using **Staff** and **Property** tables.

Example 3.42 INSERT ... SELECT

```
INSERT INTO StaffPropCount
( SELECT s.staffNo, fName, IName, COUNT(*)
  FROM Staff s, Property p
  WHERE s.staffNo = p.staffNo
  GROUP BY s.staffNo, fName, IName )
```

UNION

```
( SELECT staffNo, fName, IName, 0
  FROM Staff
  WHERE staffNo NOT IN
    ( SELECT DISTINCT staffNo
      FROM Property )
);
```

Example 3.42 INSERT ... SELECT

staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

If the second part of UNION is omitted, excludes those staff who currently do not manage any properties.

Example 3.43 Find the students who did not pass an exam. Insert the student numbers, names, course numbers and grade into an existing table named NoPass(sno ,sn ,cno,g).

```
insert into NoPass  
select s.sno , sn , cno, g  
from s , sc  
where g <60 and s.Sno =sc.Sno;
```

3.4.2 UPDATE statement

UPDATE TableName

SET columnName1 = dataValue1

[, columnName2 = dataValue2...]

[WHERE searchCondition]

- ◆ *TableName* can be the name of a base table or an updatable view.
- ◆ SET clause specifies the names of **one or more columns** that are to be updated.

UPDATE

◆ **WHERE clause** is optional:

- if **omitted**, named columns are updated for all rows in table;
- if **specified**, only those rows that satisfy *searchCondition* are updated.

◆ New *dataValue(s)* must be **compatible** with data type for corresponding column.

Example 3.44/45 UPDATE All Rows

(1) Give all staff a 3% pay increase.

UPDATE Staff

SET salary = salary*1.03;

(2) Give all Managers a 5% pay increase.

UPDATE Staff

SET salary = salary*1.05

WHERE position = 'Manager';

Example 3.46 UPDATE Selected Rows

(3) Give all staff who work in London a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03
WHERE branchNo in
      ( SELECT branchNo
        FROM Branch
        WHERE city = 'London');
```

```
UPDATE Staff , Branch
SET salary = salary*1.03
WHERE Staff.branchNo =Branch.branchNo;
```

Wrong !

Example 3.47 UPDATE Multiple Columns

Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

UPDATE Staff

SET position = 'Manager',

salary = 18000

WHERE staffNo = 'SG14';

Example 3.48 UPDATE in SQL Server

Set the grade to null, which is less than 60 for course 'Data Structure'.

```
UPDATE sc
SET grade = NULL
FROM sc, course
WHERE sc.cno = course.cno
      AND grade <60
      AND course.cname = 'Data Structure'
```

Update sc

set grade = null

Where grade <60 and cno in

(select cno from course where cname='Data Structure')

Example 3.49 Cancel the grade of student *s1*'s course *c1*.

UPDATE SC

SET grade=null

WHERE sno = 's1' and cno='c1';

注意 '=' 的用法，**set**后不能用 '**is null**', **where**后不能用 '**=null**'

Example 3.50 Increase the grades of course *C2* which is not null by 10%.

UPDATE SC

SET grade=grade*1.1

WHERE cno='c2'

OK !

3.4.3 DELETE Statement

DELETE FROM TableName
[WHERE searchCondition]

- ◆ *TableName* can be the name of a base table or an updatable view.
- ◆ *searchCondition* is optional; if omitted, all rows are deleted from table.
- ◆ If *searchCondition* is specified, only those rows that satisfy condition are deleted.
- ◆ This does **not delete table definition**.
- ◆ If we want to delete a column value in all rows, could write **UPDATE s SET sex=null**

Example 3.51/52 DELETE Specific Rows

- (1) Delete all viewings that relate to property PG4.**

```
DELETE FROM Viewing  
WHERE propertyNo = 'PG4';
```

- (2) Delete all records from the Viewing table.**

```
DELETE FROM Viewing;
```

Example 3.53 DELETE Specific Rows

Delete all course records of all female students.

```
DELETE FROM sc  
WHERE sno IN  
    ( SELECT sno  
      FROM s  
      WHERE sex='F'  
    );
```

```
DELETE FROM sc, s  
WHERE s.sno =sc.sno and sex= 'F'
```

Wrong !

Example 3.54 Remove all tuples in both table sc and s which are involved with student 's8'.

delete from sc where sno='s8';

delete from s where sno='s8';

3.5 Views

3.5.1 Definition of View

It is a **Dynamic result** of one or more relational operations operating on base relations to produce another relation.

- ◆ **Virtual relation:** Its data does not necessarily actually exist in the database, only its **definition** is kept in DD.
- ◆ It is produced **upon request**, at time of request.

SQL - CREATE VIEW

CREATE VIEW ViewName

[(newColumnName [...])]

AS subselect

[WITH CHECK OPTION]

- ◆ Can assign a name to each column in view.
- ◆ If a list of column names is specified, it must have **the same number** of items as the number of columns produced by *subselect*.
- ◆ If omitted, each column takes the name of corresponding column in *subselect*.
- ◆ The table name must be specified if there is any **ambiguity** in a column name.

SQL - CREATE VIEW

- ◆ The *subselect* is known as the defining query (定义查询) .
- ◆ **WITH CHECK OPTION** ensures that if a row fails to satisfy **WHERE clause of defining query**, it is not added to underlying base table.
- ◆ Creating a view needs **SELECT privilege** on all tables referenced in subselect. Otherwise, there will be a data security problem.

Example 3.55- Create Horizontal View

Create a view so that the manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS SELECT *
FROM Staff
WHERE branchNo = 'B003';
```

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example 3.56- Create Vertical View

Create a view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
```

```
AS SELECT staffNo, fName, lName, position, sex
```

```
FROM Staff
```

```
WHERE branchNo = 'B003';
```

Table 6.4 Data for view Staff3.

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Example 3.57- Grouped and Joined Views

Create a view of staff who manage properties for rent, including the **branch number** they work at, **staff number**, and **the number of properties** they manage.

```
CREATE VIEW StaffPropCnt (branchNo,  
staffNo, cnt)
```

```
AS SELECT s.branchNo, s.staffNo, COUNT(*) as cnt  
FROM Staff s, Property p  
WHERE s.staffNo = p.staffNo  
GROUP BY s.branchNo, s.staffNo;
```

Example 3.57- Grouped and Joined Views

Result table for Example 3.56

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

SQL - DROP VIEW

DROP VIEW ViewName [RESTRICT | CASCADE]

- Causes the **definition** of view to be deleted from the DD of the database.
- For example:

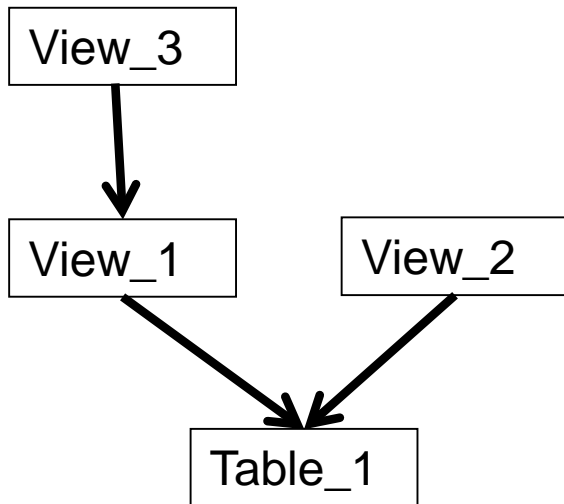
DROP VIEW Manager3Staff;

SQL - DROP VIEW

- With **CASCADE**, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With **RESTRICT (default)**, if any other objects depend for their existence on continued existence of view being dropped, command is rejected.

SQL - DROP VIEW

- **RESTRICT (default)**



Drop view View_3 **succeed**

Drop view View_2 **succeed**

Drop view View_1 **fail**

Drop view View_1 **cascade** **succeed**

3.5.2 View Resolution (视图分解)

- From the point of users' view, searching in views and searching in tables are the same.
- In fact, they are not the same because of **view resolution**.
- With **view resolution**, any operations on view are automatically translated into operations on relations from which it is derived.

3.5.2 View Resolution

Count the number of properties managed by each member at branch B003.

View Definition:

```
CREATE VIEW StaffPropCnt  
    (branchNo, staffNo, cnt)  
AS SELECT s.branchNo, s.staffNo, COUNT(*)  
    FROM Staff s, Property p  
    WHERE s.staffNo = p.staffNo  
    GROUP BY s.branchNo, s.staffNo;
```

View Query:

```
SELECT staffNo, cnt  
FROM StaffPropCnt  
WHERE branchNo = 'B003'  
ORDER BY staffNo;
```

(1) Process of View Resolution

- (a) View **column names** in SELECT list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

- (b) **View names** in FROM are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, Property p
```

(1) Process of View Resolution

- (c) WHERE from user query is combined with WHERE of defining query using AND:

**WHERE s.staffNo = p.staffNo
AND branchNo = 'B003'**

- (d) GROUP BY and HAVING clauses copied from defining query:

GROUP BY s.branchNo, s.staffNo

- (e) ORDER BY copied from query with view column name translated into defining query column name

ORDER BY s.staffNo

(1) Process of View Resolution

(f) Final merged query is now executed to produce the result:

```
SELECT s.staffNo, COUNT(*)  
FROM staff s, Property p  
WHERE s.staffNo = p.staffNo AND  
       branchNo = 'B003'  
GROUP BY s.branchNo, s.staffNo  
ORDER BY s.staffNo;
```

(2) Restrictions on Views

SQL imposes several restrictions on creation and use of views.

(a) If a column in a view is based on an **aggregate function**, then

- ◆ The column may appear **only in SELECT and ORDER BY clauses** of queries that access the view.
- ◆ The column may **not** be used **in WHERE** nor be **an argument to an aggregate function** in any query based on view.

(2) Restrictions on Views

- For example, following query would fail:

```
SELECT COUNT(cnt)  
FROM StaffPropCnt;
```

- Similarly, following query would also **fail**:

```
SELECT *  
FROM StaffPropCnt  
WHERE cnt > 2;
```

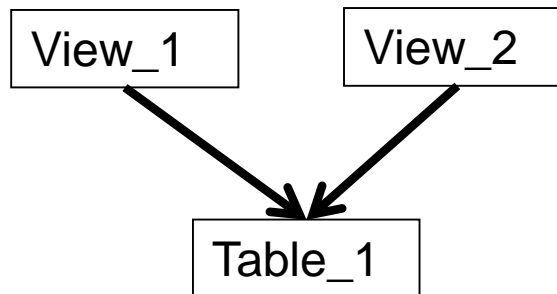
(2) Restrictions on Views

- (b) A grouped view may never be joined with a base table or a view.**
- For example, *StaffPropCnt* view is a grouped view, so any attempt to join this view with another table or view fails.**

(3) View Updatability

(视图的可更新性)

- All updates to a base table should be **reflected in all views** that are based on the base table.
- Similarly, we may expect that if the view is updated then **the base table(s) will reflect the change.**



(3) View Updatability

- However, consider again view **StaffPropCnt**.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt  
VALUES ('B003', 'SG5', 2);
```

- We have to insert 2 records into **Property** showing which properties SG5 manages. However, we do not know which properties they are; i.e. **do not know primary keys!**

(3) View Updatability

- If we change the definition of the view and replace *count* with actual *property numbers*:

```
CREATE VIEW StaffPropList (branchNo,  
                           staffNo, propertyNo)  
AS SELECT s.branchNo, s.staffNo,  
          p.propertyNo  
FROM Staff s, Property p  
WHERE s.staffNo = p.staffNo;
```

(3) View Updatability

Property (propertyNo, Street, city, **postcode**, ownerNo, Type, rent, room, **staffNo**)

- Now try to insert the record:

```
INSERT INTO StaffPropList  
VALUES ('B003', 'SG5', 'PG19');
```

- Still problem, because in **Property** all columns except **postcode/staffNo** are **not allowed nulls**.
- However, we have no way of giving the remaining **non-null columns** values.

(3) View Updatability

- ◆ ISO specifies the views that must be updatable in system that conforms to standard.
- ◆ A view is **updatable** if and only if:
 - (1) **DISTINCT** is not specified.
 - (2) Every element in **SELECT** list of defining query is a **column name** (not a constant, an expression or an aggregate function) and no column appears more than once.

(3) View Updatability

- (3) **FROM** clause specifies **only one table**, excluding any views based on a join, union, intersection or difference.
- (4) **No nested SELECT** referencing outer table (i.e. Non-correlation subquery).
- (5) **No GROUP BY or HAVING clause.**

Also, every row added through view must **not violate integrity constraints** of base table.

nested SELECT referencing outer table

```
INSERT INTO STUDENT(Sno, SNAME, SEX)
  SELECT Sno, SNAME, SEX
  FROM S
 WHERE 80<=all ( SELECT g FROM SC
                  WHERE S.Sno=SC.Sno);
```

Correlation subquery

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo =
  (SELECT branchNo
   FROM Branch
   WHERE street = '163 Main St');
```

Non-correlation subquery

Updatable View

(可更新视图)

For the view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

为了使视图可更新，D B M S 必须具有以下能力：
对于任何一个行或列都能追溯到其源表中相应的
行或列。

(4) WITH CHECK OPTION

- ◆ The **rows exist** in a view because they satisfy WHERE condition of defining query.
- ◆ If a row changes and no longer satisfies condition, it **disappears** from the view.
- ◆ **New rows** will **appear** within the view when **insert/update** on the base table cause them to satisfy WHERE condition.
- ◆ The rows that **enter or leave** a view are called *migrating rows*.
- ◆ WITH CHECK OPTION **prohibits** a row migrating **out of the view**.

Example 3.58 - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff  
AS      SELECT *  
        FROM Staff  
        WHERE branchNo = 'B003'  
        WITH CHECK OPTION;
```

- **In the above view**, we **cannot change** the branch number of row **B003 to B002** as this would cause row to migrate from view.
- Also we **cannot insert** a row into view with a branch number that does **not equal B003**.

WITH CHECK OPTION

WITH CHECK OPTION has no affect on Source Table.

```
CREATE VIEW view_80
AS SELECT *
FROM sc
WHERE grade > 80
WITH CHECK OPTION ;
```

```
SELECT * FROM VIEW_80
```

```
UPDATE view_80
SET grade=78
WHERE sno='08000025' AND cno='801'
```

Results		Messages	
	sno	cno	grade
1	08000025	801	87
2	08300012	801	91
3	08300012	803	94
4	08300012	804	81
5	08300012	805	89
6	08300015	801	99

WITH CHECK OPTION

```
UPDATE view_80  
SET grade=78  
WHERE sno='08000025' AND cno='801'
```

Messages

Msg 550, Level 16, State 1, Line 1

The attempted insert or update failed because the target view either specifies
WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION

The statement has been terminated.

```
UPDATE sc
```

```
SET grade=78
```

```
WHERE sno='08000025'
```

```
AND cno='801'
```

```
SELECT * FROM VIEW_80
```

Results

Messages

	sno	cno	grade
1	08300012	801	91
2	08300012	803	94
3	08300012	804	81
4	08300012	805	89
5	08300015	801	99
6	08300015	805	92

3.6 Programmatic SQL

SQL in Real Programs

- ◆ We have seen only how SQL is used at the generic query interface --- an environment where we sit at a terminal and ask queries of a database, *interactive SQL*.
- ◆ Reality is almost always different, SQL do not have *Control flow command*, such as IF... THEN...ELSE, GO TO or DO WHILE statements. So SQL are embedded in a conventional *host language*, *Programmatic SQL*.

Methods of Programmatic SQL

- ◆ **PSM** (*persistent stored modules*). A collection of stored functions or procedures, some of which are host-language code and some of which are SQL statements. PSM is stored in the database itself, as part of the schema. (e.g., T-SQL, PL/SQL---**Stored procedure and trigger**)
- ◆ **Embedded SQL**. SQL statements are embedded in a *host language* (e.g., C). SQL statements *need to be precompiled*. (需预编译**SQL**, **Static & Dynamic Embedded SQL**)
- ◆ **API** (*Application programming interface*). It is the connection tools that are used to allow a conventional language to access a database. It is a standard set of functions that can provide the same functionality as embedded statements and *remove the need for any precompilation*. (无需预编译**SQL**, e.g.**CLI, ODBC/JDBC, ADO**).

3.6 Programmatic SQL

3.6.1 PSM (*persistent stored modules*).

- ◆ Stored Procedure
- ◆ Constraints and Trigger

3.6.2 Embedded SQL

- ◆ Static Embedded SQL
- ◆ Dynamic Embedded SQL

3.6.3 API (*Application programming interface*).

- ◆ CLI
- ◆ ODBC
- ◆ JDBC

3.6.1 PSM

- ◆ PSM or SQL/PSM (*persistent stored modules*) is part of the latest revision to the SQL standard, called **SQL:2003**. It allows us to write **stored procedures** and **user-defined functions** as database schema elements.
- ◆ PSM = a **mixture** of conventional statements (**if, while, etc.**) and **SQL**.
- ◆ Let us do things we cannot do in SQL alone.
- ◆ Each commercial DBMS offers its own extension of PSM.
- ◆ PSM: (1) **Stored procedures** (2) **Triggers**

1. Stored procedures

- ◆ SP is the program that is **compiled ahead** and stored in DBMS. It can complete the specified operations on DB.
- ◆ SP is the **mixed set** of SQL statements and control flow statements. **It is compiled at the first time of being executed.**
- ◆ SP can be called actively by an application program. Its executing speed is high.

1. Stored procedures

```
CREATE PROCEDURE <name> (  
    <parameter list> )  
    <optional local declarations>  
    <body>;
```

Example: Create a SP

Transfer an amount of money from Account 1 to Account 2

```
CREATE PROCEDURE transfer
```

```
(@inAccount INT, @outAccount INT,  
 @amount FLOAT)
```

```
/*define a SP named transfer, parameter:  
intoAccount、outAccount、transferAmount*/
```

```
AS DECLARE /*define variables*/
```

```
    @balance Float; /*存放余额的变量*/
```

Example: Create a SP

```
BEGIN    /*SP main body*/
```

```
/*Check outAccount */
```

```
SELECT Total INTO @balance
```

```
FROM Accout
```

```
WHERE accountnum=@outAccount ;
```

```
IF @balance IS NULL THEN    /*If outAccount does not  
BEGIN
```

```
    ROLLBACK;
```

```
    exist or have no fund */
```

```
    RETURN;
```

```
    /*回滚事务*/
```

```
END;
```

```
IF @balance < @amount THEN    /*If balance is not enough*/  
BEGIN
```

```
    ROLLBACK;
```

```
    /*回滚事务*/
```

```
    RETURN;
```

```
END ;
```

Example: Create a SP

/* Check inAccount */

IF @inAccount IS NULL /* If inAccount does not exist */

THEN

BEGIN

ROLLBACK;

RETURN;

END;

/*回滚事务*/

Example: Create a SP

```
/*transfer operation */
```

```
/* change the balance of outAccount, reduced by  
transferAmount */
```

```
UPDATE Account
```

```
SET total=total-@amount
```

```
WHERE accountnum=@outAccount;
```

```
/* change the balance of inAccount , increased  
by transferAmount */
```

```
UPDATE Account
```

```
SET total=total + @amount
```

```
WHERE accountnum=@inAccount;
```

```
COMMIT;
```

```
/*commit the transaction */
```

```
END;
```

Call a SP

Format:

CALL/PERFORM PROCEDURE **SP-Name**
([para1, para2,...]);

EXAMPLE:

Transfer \$100 from Account **0103815868** to Account **0103813828**

CALL PROCEDURE transfer (**0103813828**, **0103815868**, **100**);
In **Out**

EXEC transfer 0103813828, 0103815868, 100; //SQL Server

Drop a SP

DROP PROCEDURE SP-Name;

EXAMPLE: **DROP PROCEDURE** transfer;

Example: Use SQL Server to modify the name of a student in table s from oldName to newName.

```
CREATE PROCEDURE changeName (@oldName CHAR(10), @newName  
CHAR(10))
```

```
AS
```

```
BEGIN tran
```

```
    IF NOT EXISTS( SELECT *  
                    FROM s  
                    WHERE sname=@oldName)
```

```
    BEGIN
```

```
        ROLLBACK;
```

```
        RETURN;
```

```
    END;
```

```
    UPDATE s
```

```
    SET sname=@newname
```

```
    WHERE sname=@oldName;
```

```
COMMIT tran;
```


2. Constraints and Triggers

(1) Constraints (约束)

A *constraint* is a relationship among data elements that the DBMS is required to enforce.

Two Kinds of Constraints

- ◆ **Attribute-based** constraints. (列级约束)
 - It constrains the values of a particular attribute.
- ◆ **Tuple-based** constraints. (表级约束)
 - It constrains the relationship among components.

Attribute-Based Checks

- ◆ Constraints on the value of **a particular attribute**.
- ◆ Add **CHECK(<condition>)** to the declaration for the attribute.
- ◆ **Timing of Checks**: Attribute-based checks are performed only when a value for that attribute is **inserted or updated**.

Example: **S(SNO,SN,age,sex)**

CREATE TABLE S

(SNO CHAR(4) NOT NULL,

SN CHAR(8) NOT NULL,

AGE SMALLINT check (age>17) ,

SEX CHAR(1),

PRIMARY KEY (SNO)

);

Tuple-Based Checks

- ◆ **CHECK (<condition>)** may be added as a relation-schema element.
- ◆ The condition may refer to any attribute of **the relation**.
- ◆ Checked on **insert or update** only.

Example: **S(SNO,SN,age,sex)**

CREATE TABLE S

**(SNO CHAR(4) NOT NULL,
SN CHAR(8) NOT NULL,
AGE SMALLINT,
SEX CHAR(1),**

PRIMARY KEY (SNO) ,

**check((sex='f' and age<21) or
(sex='m' and age<23))**

);

Motivation for Trigger

- ◆ Attribute- and tuple-based checks are checked at known times, but are **not powerful**.
- ◆ **Triggers** let the user decide when to check for any condition, **not only on insert or update**.

(2) Triggers

- ◆ *Triggers* are code stored in the database and **invoked** (*triggered*) **by events** that occur in the database.
- ◆ Triggers are **only executed** when a specified condition occurs, e.g., insertion of a tuple.
- ◆ It is **Easier to implement** than complex constraints.

Event-Condition-Action Rules

- ◆ Another name for “trigger” is *ECA rule*, or *event-condition-action* rule.
- ◆ **Event**: typically a type of database modification, e.g., “insert on Student.”
- ◆ **Condition**: Any SQL boolean-valued expression, e.g., $\text{age} < 18$.
- ◆ **Action**: Any SQL statements.

Trigger in SQL SERVER

- ◆ **Trigger** is a **special kind** of **stored procedure** in which parameters are not allowed.
- ◆ Triggers can not be called directly by Users, only can **be automatically triggered** and executed by system when given data items are modified.
- ◆ Triggers are used to **guarantee the data integrity**, i.e., the match between PK and FK, also used to implement the complex business rules.

Trigger in SQL SERVER

- ◆ **Triggers** can not be **created** on **temporary tables**, but temporary tables can be used in triggers.
- ◆ There are two special temporary tables in SQL SERVER, i.e., **inserted** and **deleted**, which have the same structures with the trigger table or view.
- ◆ Users can refer the above two temporary tables, but can not modify the data in them.

- ◆ When doing *inserting* operation, the inserted row(s) is (are) also copied to the temporary table *inserted* .
- ◆ When doing *deleting* operation, the deleted row(s) is (are) also copied to the temporary table *deleted* .
- ◆ When doing *updating* operation, first delete the old row(s) which is (are) also copied to the temporary table *deleted* , then insert the new row(s) which is (are) also copied to the temporary table *inserted* .

Syntax for CREATE Triggers statement

```
CREATE TRIGGER [owner.] trigger-name  
ON [owner.] table-name
```

```
{FOR { INSERT, UPDATE, DELETE}
```

//case 1, perform the actions provided that
the event occurs. (row-level trigger, 行级触发器)

```
AS      <SQL statement>
```

```
FOR {INSERT, UPDATE}
```

```
AS      IF UPDATE (<column-name>)
```

```
[{AND|OR} UPDATE (< column-name >).....]
```

```
<SQL statement>}
```

//case 2, perform the actions only when the
specified column is modified.

(column-level trigger, 列级触发器)

Insert into **s values('s6','ss6',19)**

inserted (sno,sname,age,sex).

Delete from **sc where sno='s1'**

Deleted (sno,cno,grade) .

inserted

sno	sname	age	sex
s6	ss6	19	

deleted

sno	cno	grade
s1	c2	89
s1	c4	96
s1	c5	84

Two Types of Triggers in SQL SERVER

1. **AFTER** (or **For** can only be created on tables.)

(1) Triggers are only awakened when certain events, such as ***Insert, Update or Delete***, occur.

(2) The trigger tests a condition, i.e., Checking on the constraint.

◆ If the condition **does not hold**, then nothing else associated with the trigger happens in response to this event.

◆ If the condition **is satisfied**, the actions of **AFTER** triggers is performed by DBMS.

Two Types of Triggers in SQL SERVER

2. **INSTEAD OF** (can be created both on tables and views)

- ◆ If a view is created with “**with check option**”, a trigger can not be created on it.
- ◆ When a trigger is defined on a view, we can use **Instead of** in place of **AFTER**.
- ◆ If we do so, then when an event awakens the trigger, the action of the trigger is **done instead of the event itself**.

SQL SERVER中两种类型的触发器

1. **AFTER 类型**(或 **For** 只能建于基本表上)

- ◆ 当某一事件，如**Insert, Update** 或 **Delete**发生时，触发器才会被触发。
- ◆ 触发器测试条件，即检查约束性。若条件不成立，则针对此事件，不会引发相关的触发器。
- ◆ 若条件满足，则**DBMS**执行**AFTER** 触发器的动作。

2. **INSTEAD OF类型** (既可建于基本表上，也可建于视图上)

- ◆ 若视图在创建时使用了“**with check option**”选项，则不能在其上创建触发器。
- ◆ 当在一个视图上定义了一个触发器，可以用**Instead of**代替**AFTER**触发器。
- ◆ 若如此，当一个事件唤醒触发器时，则执行该触发器的动作，而不执行该事件本身。

Notes

- ◆ Each event of *insert, update or delete* on one table or view can have **only one** **INSTEAD OF** trigger.
- ◆ Each event of *insert, update or delete* on one table can have **many** **AFTER** triggers.
- ◆ **Executing process of AFTER trigger**
If an operation of insert, update or delete violate the data integrity, then the actions of **AFTER** triggers won't be executed because the checking on constraint is done before the actions of the trigger.

特别说明

- ◆ 针对一个基本表或视图上的 *insert, update* 或 *delete* 的每个事件，只能定义一个 **INSTEAD OF** 类型的触发器。
- ◆ 针对一个基本表上的 *insert, update* 或 *delete* 的每个事件，可以定义多个 **AFTER** 类型的触发器。
- ◆ **AFTER** 触发器的执行过程
 - 在执行触发器动作之前，先检查完整性约束；
 - 因此，若 *insert, update* 或 *delete* 操作违反了数据完整性，则 **AFTER** 触发器的动作不会被执行。

Preliminary Example: A Trigger

Instead of using a foreign-key constraint and rejecting insertions into `sc(sno, cno, grade)` with `unknown sno`, a trigger can add that `sno` to Table `s`, with other columns as NULL values in Table `s`.

S

sno	sname	age	sex
s1	ss1	18	f
s2	ss2	19	m

SC

sno	cno	grade
s1	c1	90
s3	c2	87

Being rejected !

Trigger Definition in SQL Server

```
CREATE TRIGGER sc_Trig ON sc
```

Trigger table

```
FOR INSERT
```

```
AS if ((select sno from inserted)
```

Has the same structure with sc

```
NOT IN (SELECT sno FROM s))
```

条件为真,
执行此语
句

```
INSERT INTO s(sno)  
select sno from inserted;
```

Has the same structure with sc

S

sno	sname	age	sex
s1	ss1	18	f
s2	ss2	19	m
s3	null	null	null

SC

sno	cno	grade
s1	c1	90
s3	c2	87

Trigger that guarantees deleting
in cascading way

```
CREATE TRIGGER dele_trig ON s  
FOR DELETE  
AS DELETE sc FROM deleted, sc  
where sc.sno=deleted.sno
```

* There must be **sc** after **DELETE**.

• **sc** after **FROM** is not essential.

• **DELETE**后必须有**sc**, **FROM**后有无**sc**均可

Column-level Trigger in SQL Server

Example : Create a trigger to stop the attempt to lower the grade of a student.

```
CREATE TRIGGER No_lower_Grade ON sc
FOR UPDATE
AS IF UPDATE(grade)
    IF ((SELECT grade FROM deleted) >
        (SELECT grade FROM inserted))
        ROLLBACK TRANSACTION ;
```

or

```
CREATE TRIGGER No_lower_Grade ON sc  
FOR UPDATE
```

```
AS IF UPDATE(grade)
```

```
IF((SELECT grade FROM deleted) >  
   (SELECT grade FROM inserted))
```

```
UPDATE sc
```

```
SET grade= (SELECT grade FROM deleted)  
WHERE sno= (SELECT sno FROM inserted)  
and cno= (SELECT cno FROM inserted);
```

Yes!

deleted ?

SC

sno	cno	grade
s1	c1	87

deleted

sno	cno	grade
s1	c1	90

inserted

sno	cno	grade
s1	c1	87

Triggers on Views

- ◆ Generally, it is impossible to modify a virtual view, because it doesn't exist.
- ◆ But an **INSTEAD OF** trigger let us interpret view modifications in a way that makes sense.

Instead of

```
CREATE VIEW B003_Staff
```

```
As select StaffNo, Iname  
      from Staff where branchno='B003';
```

Example: INSERT a new row into view B003_Staff

We cannot insert into a virtual view.

So we need to interpret a View Insertion (view resolution)

Example: Instead of Trigger

// In SQL SERVER

```
CREATE TRIGGER B003_insert ON B003_Staff  
INSTEAD OF INSERT
```

As INSERT INTO Staff (staffNo, lname, branchNo)

SELECT sno, sname, 'B003'

from inserted;

Inserted (StaffNo, lname)

Trigger view

End of Lecture 10