In the Lecture Series Introduction to Database Systems

Advanced SQL

In the Lecture Series Introduction to Database Systems

Views

Presented by Stéphane Bressan

Creating a View

Views are named queries.

```
CREATE VIEW cs_student
AS (SELECT email, name, year, graduate
    FROM student
    WHERE department ='CS');
```

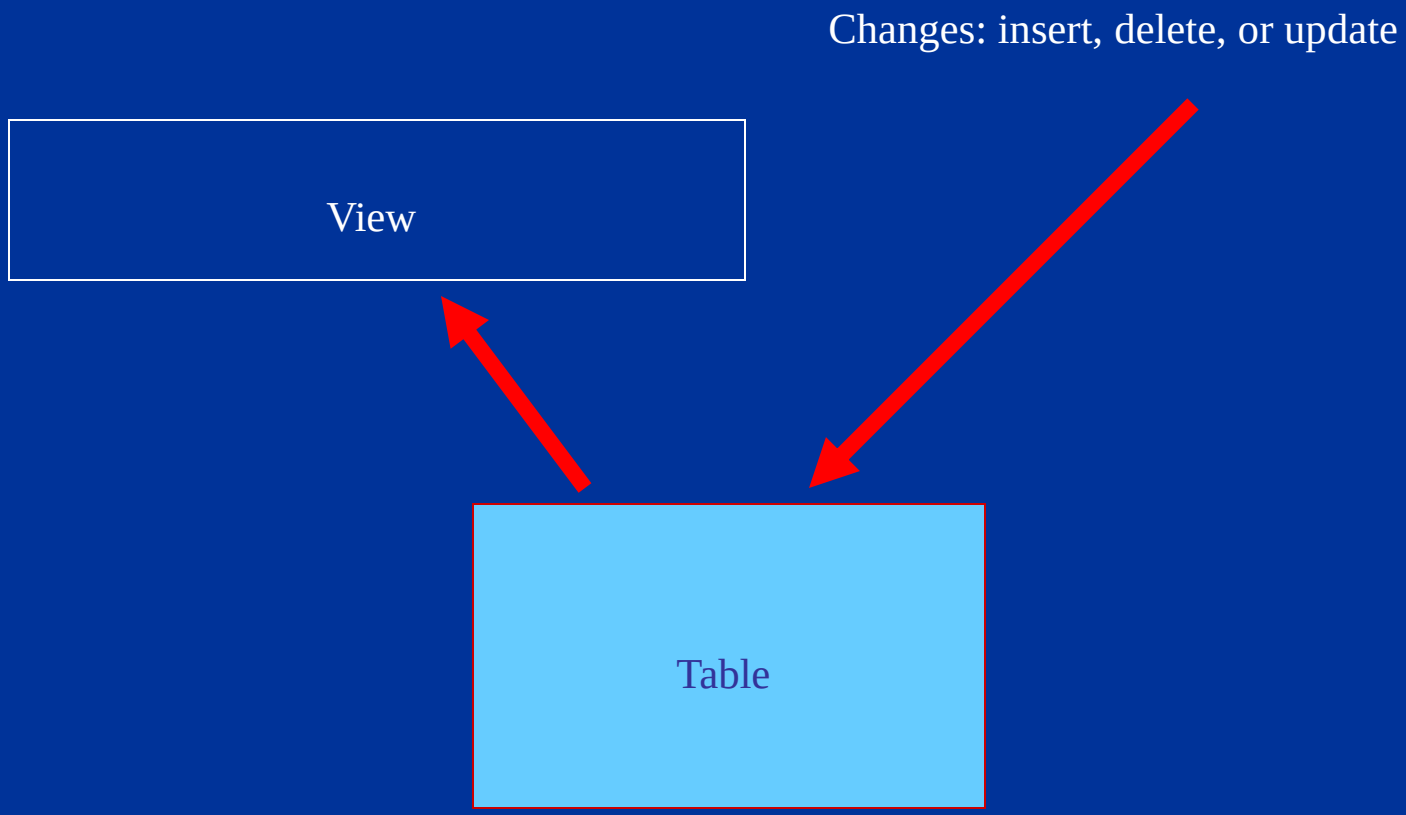
Querying a View

Views can be used as normal tables in queries.

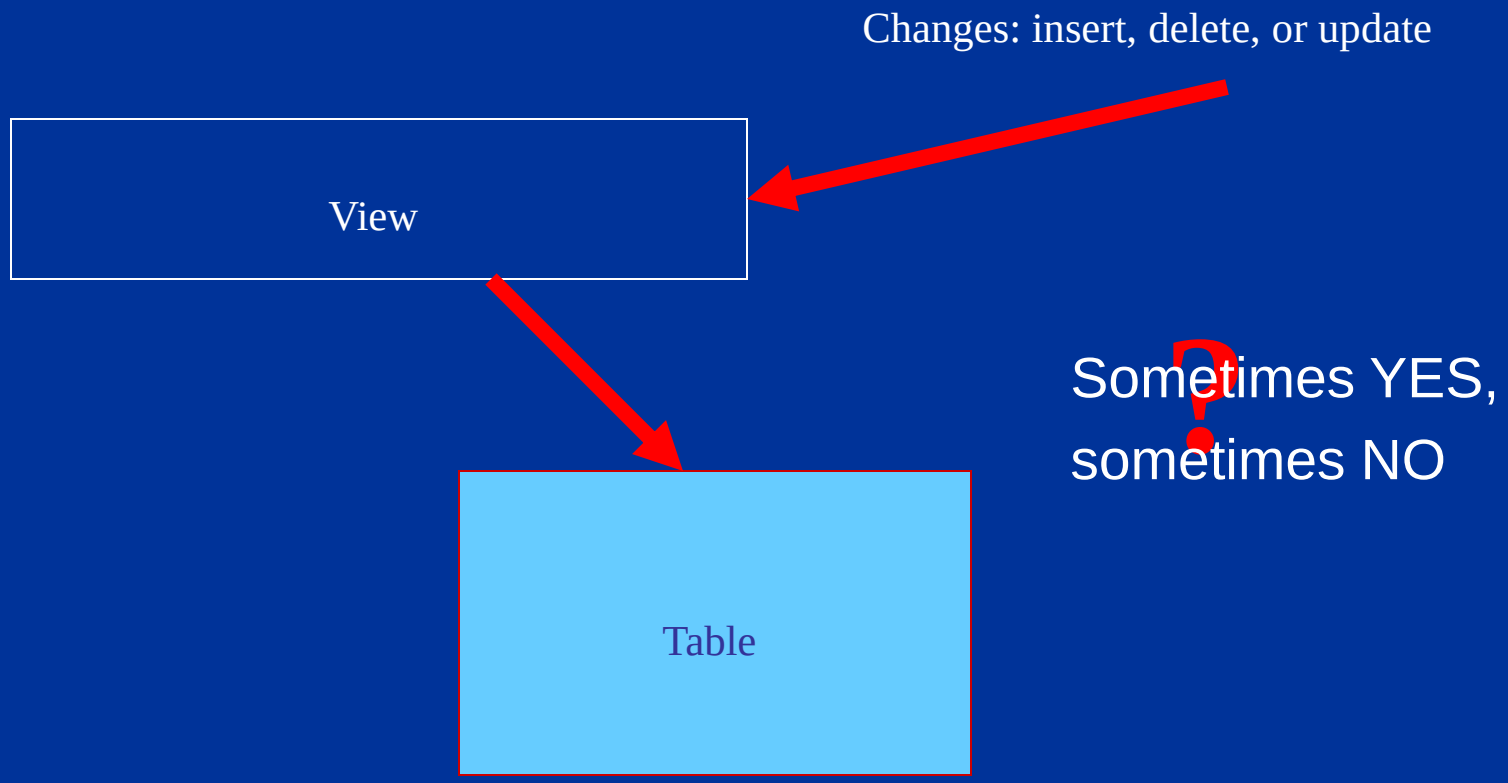
```
SELECT *
FROM cs_student s, copy c
WHERE s.email = c.owner;
```

Views

Changes in the base tables are propagated



Can we update the views?



Updating a View

Under certain circumstances views can be updated.

```
INSERT INTO cs_student VALUES('GOH WEE SIONG',
   'gohws1989@gmail.com', '2008-01-01', NULL);
```

"cannot insert NULL into student"

But not our `cs_student` view because the faculty and department cannot be NULL (notice that the department at least should be understood)

An Updatable View

```
CREATE VIEW student1
AS (SELECT email, name, year, faculty, department
    FROM student);

INSERT INTO student1 VALUES('GOH WEE SIONG',
  'gohws1989@gmail.com', '2008-01-01', 'School of
  Computing', 'CS');
```

`graduate` is NULL in the table `student`.

A Surely not Updatable View

```
CREATE VIEW total_student (total)
AS (SELECT COUNT(*) FROM student);

INSERT INTO total_student VALUES(100);
```

"data manipulation operation not legal on this view"

Updating Views with Triggers

Views can be updated by triggers using INSTEAD OF.

Views

- Logical Data Independence is achieved by means of views
- Views can be pre-compiled
- However views may fool the optimizer

In the Lecture Series Introduction to Database Systems

PL/SQL

Presented by Stéphane Bressan

Stored Procedures

- Similar to procedures in programming languages
- Stored inside the database
- Can access database tables
- May contain SQL statements
- Additionally they support:
  - LOOP
  - IF-THEN-ELSE statement

Creating a Procedure

A procedure has a name and a body.

```
CREATE OR REPLACE PROCEDURE test IS
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END test;
```

Cursors

Cursors are used to iterate over query results.

```
CREATE OR REPLACE PROCEDURE test2 IS
    e student.email%TYPE;
    CURSOR c IS SELECT email FROM student;
BEGIN
OPEN c;
LOOP
FETCH c INTO e;
EXIT WHEN c%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Email: '|| e);
END LOOP;
CLOSE c;
END;
```

Control Structures

Cursors are used to iterate over query results.

```
CREATE OR REPLACE PROCEDURE test3 IS
    e student.email%TYPE;
    d student.department%TYPE;
    CURSOR c IS SELECT email, department FROM student;
BEGIN
OPEN c;
LOOP
FETCH c INTO e, d;
EXIT WHEN c%NOTFOUND;
IF e='CS' THEN DBMS_OUTPUT.PUT_LINE('CS Email: '|| e);
ELSE DBMS_OUTPUT.PUT_LINE('Email: '|| e);
END IF;
END LOOP;
CLOSE c;
END;
```

Control Structures

Print the top ten results.

```
CREATE OR REPLACE PROCEDURE test4 IS
    e student.email%TYPE;
     i NUMBER := 1;
CURSOR c IS SELECT email FROM student;
BEGIN
OPEN c;
LOOP
FETCH c INTO e;
i := i+1;
EXIT WHEN (i>10 OR c%NOTFOUND);
DBMS_OUTPUT.PUT_LINE('Email '||i|| ': ' || e);
END LOOP;
CLOSE c;
END;
```

In the Lecture Series Introduction to Database Systems

Triggers

Presented by Stéphane Bressan

Trigger

- Database Triggers refer to an active mechanism that allows the programming of reaction to database events.
  - Insertion, deletion, update
- A database trigger takes the general form:
  - ``When something happens and if some condition is met do something''

Trigger (Example with Oracle)

```
CREATE OR REPLACE TRIGGER say_something
BEFORE DELETE OR INSERT OR UPDATE ON student
FOR EACH ROW WHEN (new.department = 'CS' OR
  old.department= 'CS')
BEGIN
dbms_output.put_line('Something happened to
  us');
END;
```

Trigger (Applications)

- Implement more application logic into the database (good idea!)
- Implement integrity constraint checking (bad idea but no sometimes choice)
- Implement integrity constraint checking and propagation (not the best of theoretically possible ways idea but no sometimes choice)

Statement-level Trigger (Syntax)

```
CREATE [OR REPLACE] TRIGGER <name>
[BEFORE | AFTER]
[DELETE | INSERT | UPDATE [OF <column>[,
  column]*]
ON <table>
[WHEN (<condition>)]
<PL/SQL block>
```

Statement-level Trigger (Semantics)

- A statement trigger is fired once for the triggering statement, regardless of the number of rows affected and even if no row is affected.
- For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

Statement-level Trigger (Example)

```
CREATE OR REPLACE TRIGGER say_something
BEFORE DELETE OR INSERT OR UPDATE ON student
BEGIN
dbms_output.put_line('Something happened to
  us');
END;
```

Row-level Trigger (Syntax)

```
CREATE [OR REPLACE] TRIGGER <name>
[BEFORE | AFTER]
[DELETE | INSERT | UPDATE [OF <column>[,
  column]*]
ON <table>
FOR EACH ROW
[REFERENCING OLD|NEW AS <name>]
[WHEN (<condition>)]
<PL/SQL block>
```

Row-level Trigger (Semantics)

- A row trigger is fired each time the table is for each row affected by the triggering statement.
- If the triggering statement affects no rows then nothing happens.
- For example, if a UPDATE statement updates several rows from a table, a row-level UPDATE trigger is fired for each affected row.

Row-level Trigger (Example)

```
CREATE OR REPLACE TRIGGER say_something
BEFORE DELETE OR INSERT OR UPDATE ON student
FOR EACH ROW
WHEN (new.department = 'CS' OR
  old.department= 'CS')
BEGIN
dbms_output.put_line('Something happened to
  us');
END;
```

Old/New in SQL Standard

- OLD and NEW reference the afftected rows only not the rows in the new and old table
  - OLD ROW AS <name>
  - NEW ROW AS <name>
  - OLD TABLE AS <name>
  - NEW TABLE AS <name>

CASCADE DELETE

```
CREATE OR REPLACE TRIGGER
  cascade_delete_student_to_copy_and_loan
BEFORE DELETE
ON student
FOR EACH ROW
BEGIN
DELETE FROM loan WHERE
  loan.owner=:old.email;
DELETE FROM copy WHERE
  copy.owner=:old.email;
END
```

Why not so good?

CASCADE DELETE

```
CREATE OR REPLACE TRIGGER
  cascade_delete_student_to_copy
BEFORE DELETE
ON student
FOR EACH ROW
BEGIN
DELETE FROM copy WHERE
  copy.owner=:old.email;
END
```

CASCADE DELETE

```
CREATE OR REPLACE TRIGGER
  cascade_delete_copy_to_loan
BEFORE DELETE
ON copy
FOR EACH ROW
BEGIN
DELETE FROM loan WHERE copy.owner=:old.owner
AND copy.book=:old.book
AND copy.copy=:old.copy;
END
```

CASCADE DELETE in SQL Standard

```
CREATE OR REPLACE TRIGGER
  cascade_delete_student_to_copy
BEFORE DELETE
ON student
REFERENCING OLD TABLE AS OT
BEGIN
DELETE FROM copy WHERE copy.owner in (SELECT
  email FROM OT);
END
```

Other Triggers

- Oracle allows other types of triggers:
  - Schema triggers, monitoring modification to the schema (ALTER, CREATE, DROP, GRANT etc.)
  - Database triggers, monitoring system events (login, logoff, shutdown, etc.)
  - INSTEAD OF triggers for views (to define the update of a view)

Schema Trigger (Example)

```
CREATE OR REPLACE TRIGGER drop_student
BEFORE DROP ON my.SCHEMA
BEGIN
RAISE_APPLICATION_ERROR (
   • num => -20000, msg => 'Cannot!');
END;
```

In the Lecture Series Introduction to Database Systems

Indexes

Presented by Stéphane Bressan

Indexes in Oracle

- An index is a data structure built and used for fast access.
- Oracle creates B-tree indexes.

No Index

```
CREATE TABLE book (
title VARCHAR(255) NOT NULL,
format CHAR(9),
pages INT,
language VARCHAR(32),
authors VARCHAR(255),
publisher VARCHAR(64),
year DATE,
ISBN10 CHAR(10),
ISBN13 CHAR(14)
);
```

Query Plan

```
SELECT *
FROM book
WHERE ISBN13 = '978-1449389673'
```

SELECT STATEMENT

TABLE ACCESS BOOK FULL

σ Filter Predicates

ISBN13='978-1449389673‘

The system performs a full table scan to find a row

Primary Key

```
CREATE TABLE book (
title VARCHAR(255) NOT NULL,
format CHAR(9),
pages INT,
language VARCHAR(32),
authors VARCHAR(255),
publisher VARCHAR(64),
year DATE,
ISBN10 CHAR(10),
ISBN13 CHAR(14) PRIMARY KEY
);
```

Primary Key

```
CREATE UNIQUE INDEX "STEPH"."SYS_C007097" ON
   "STEPH"."BOOK" ("ISBN13")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1
   MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL
   DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
```

The system has created an index and can use it to find a row given the ISBN13

Query Plan

```
SELECT *
FROM book
WHERE ISBN13 = '978-1449389673'
```

SELECT STATEMENT

TABLE ACCESS BOOK BY INDEX ROWID

INDEX SYS_C007097 UNIQUE SCAN

σ Access Predicates

ISBN13='978-1449389673'

UNIQUE

```
CREATE TABLE book (
title VARCHAR(255) NOT NULL,
format CHAR(9),
pages INT,
language VARCHAR(32),
authors VARCHAR(255),
publisher VARCHAR(64),
year DATE,
ISBN10 CHAR(10) UNIQUE NOT NULL,
ISBN13 CHAR(14) PRIMARY KEY
);
```

Query Plan

```
SELECT *
FROM book
WHERE ISBN10 = '1449389678'
```

SELECT STATEMENT

TABLE ACCESS BOOK BY INDEX ROWID

INDEX SYS_C007098 UNIQUE SCAN

σ Access Predicates

ISBN13= '1449389678'

The system creates an index and uses it to find a row given the ISBN10

Index on Attribute

We can create an index on frequently accessed attributes

```
CREATE INDEX student_name ON student(name);
```

The system has created an index and can use it to find a row given the name

No Index on Attribute

```
SELECT
s.email
FROM student s
WHERE s.name LIKE 'G%';
```

SELECT STATEMENT

TABLE ACCESS STUDENT FULL

σ Filter Predicates S.NAME LIKE 'G%'

!

Index on Attribute

```
SELECT
s.email
FROM student s
WHERE s.name LIKE 'G%';
```

SELECT STATEMENT

TABLE ACCESS STUDENT BY INDEX ROWID

INDEX SUDENT_NAME RANGE SCAN

σ Access Predicates S.NAME LIKE 'G%'

σ Filter Predicates S.NAME LIKE 'G%'

!

Index on Foreign Key

We can create an index on a foreign key

```
CREATE INDEX copy_owner ON copy(owner);
```

The system has created an index and can use it to find a row given the owner

No Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s, copy c
WHERE
s.email=c.owner
GROUP BY s.email;
```

SELECT STATEMENT

HASH GROUP BY

TABLE ACCESS COPY FULL

There is no need to access student!

Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s, copy c
WHERE
s.email=c.owner
GROUP BY s.email;
```

SELECT STATEMENT

HASH GROUP BY

INDEX COPY_OWNER FAST FULL SCAN

No Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s RIGHT OUTER JOIN  copy c ON s.email=c.owner
GROUP BY s.email;
```

SELECT STATEMENT
- HASH GROUP BY
  - ▷◁ NESTED LOOPS
    - TABLE ACCESS COPY FULL
    - INDEX SYS_C007126 UNIQUE SCAN
      - σ Access Predicates S.EMAIL(+)=C.OWNER

Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s RIGHT OUTER JOIN copy c ON s.email=c.owner
GROUP BY s.email;
```

SELECT STATEMENT
- HASH GROUP BY
  - ▷◁ NESTED LOOPS
    - INDEX COPY_OWNER FAST FULL SCAN
    - INDEX SYS_C007126 UNIQUE SCAN
      - σ Access Predicates S.EMAIL(+)=C.OWNER

No Index on Foreign Key

```
SELECT s.email, s.name, COUNT(*)
FROM student s, copy c
WHERE
s.email=c.owner
GROUP BY s.email, s.name;
```

SELECT STATEMENT

HASH GROUP BY

▷◁ HASH JOIN

σ Access Predicates S.EMAIL=C.OWNER

TABLE ACCESS STUDENT FULL

TABLE ACCESS COPY FULL

Index on Foreign Key

```
SELECT s.email, s.name COUNT(*)
FROM student s, copy c
WHERE
s.email=c.owner
GROUP BY s.email, s.name;
```

SELECT STATEMENT

HASH GROUP BY

▷◁ HASH JOIN

σ Access Predicates S.EMAIL=C.OWNER

TABLE ACCESS STUDENT FULL

INDEX COPY_OWNER FAST FULL SCAN

Composite Index

```
CREATE INDEX loan_owner_ISBN13_copy ON loan(owner, ISBN13,
   copy);
```

The system has created a B-tree index and can use it to find a row given the owner, ISBN13 and copy (the book)

Queries that access all three columns, only the owner column or only the owner and ISBN13 columns can use this index.

No Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s, loan l
WHERE s.email=l.owner
GROUP BY s.email;
```

SELECT STATEMENT

HASH GROUP BY

▷◁ NESTED LOOPS

TABLE ACCESS LOAN FULL

INDEX SYS_C007126 UNIQUE SCAN

σ Access Predicates S.EMAIL=L.OWNER

Index on Foreign Key

```
SELECT s.email, COUNT(*)
FROM student s, loan l
WHERE s.email=l.owner
GROUP BY s.email;
```

SELECT STATEMENT

HASH GROUP BY

▷◁ NESTED LOOPS

INDEX LOAN_OWNER_BOOK_COPY FAST FULL SCAN

INDEX SYS_C007126 UNIQUE SCAN

σ Access Predicates S.EMAIL=L.OWNER

Credits

Clipart and media are licensed from Microsoft Office Online Clipart and Media

- **KHACHHANG** (MAKH, HOTEN, DCHI, SODT, NGSINH, DOANHSO, NGDK)
  NHANVIEN (MANV,HOTEN, NGVL, SODT)
  SANPHAM (MASP,TENSP, DVT, NUOCSX, GIA)
  HOADON (SOHD, NGHD, MAKH, MANV, TRIGIA)
  CTHD (SOHD,MASP,SL)

Ràng buộc: Ngày mua hàng (NGHD) của một khách hàng thành viên sẽ lớn hơn hoặc bằng ngày khách hàng đó đăng ký thành viên (NGDK).

```
CREATE TRIGGER CHECK_NGAYNV --Tên Trigger
ON HOADON
FOR UPDATE,INSERT
AS
   IF UPDATE(NGHD) --Kiểm tra việc cập nhật trên cột
   BEGIN
   DECLARE @NGHD SMALLDATETIME, @NGVL SMALLDATETIME
   SET @NGHD=(SELECT NGHD FROM INSERTED)
   SET @NGVL=(SELECT NGVL FROM NHANVIEN A,INSERTED B WHERE A.MANV=B.MANV)
   IF(@NGHD<@NGVL)
      BEGIN
      PRINT 'NGHD PHAI LON HON NGVL'
      ROLLBACK TRAN -- Câu lệnh quay lui khi thực hiện biến cố không thành công
      END
      END
```