



Active Database

Tim Pengajar MBD Teknik Informatika

Outline

- Active Database
- View
- Sequence
- Trigger
- Function
- Procedure
- Cursor

Passive Database

- Sistem manajemen basis data tradisional (DBMS) bersifat pasif dalam arti bahwa **perintah dijalankan oleh basis data** (misalnya query, perbarui, hapus) sebagaimana dan ketika diminta oleh pengguna atau program aplikasi.
- Basis data ini memiliki prinsip pembaruan pasif. Update pada DBMS diatur oleh *client*.
- Namun, beberapa kasus pemodelan basis data tidak dapat dimodelkan secara efektif oleh pola ini.

Problem pada *Passive DB*

Inventory control

- Memesan Kembali item ketika jumlah stok dalam gudang berkurang hingga di bawah threshold.

Travel waiting list

- Memesan tiket sesegera mungkin setelah yang tepat tersedia.

Stock market

- Beli/jual stok Ketika harganya di atas /bawah threshold.

Active Database

- *Active database* mendukung aplikasi sebelumnya dengan cara **memindahkan perilaku reaktif** dari aplikasi **ke dalam DBMS**.
- *Active DB* dapat **memonitor dan bereaksi terhadap keadaan tertentu** yang relevan terhadap aplikasi.
- Sistem database aktif harus mendukung:
 - Model pengetahuan (contoh: deskripsi mekanisme), dan
 - Model eksekusi (contoh: strategi runtime) untuk mendukung perilaku reaktif.
- *Active rules*: aturan-aturan yang secara otomatis dipicu oleh *event* dalam basis data.

VIEW



View

- **View** merupakan table virtual yang memuat result-set / **hasil query**.
- Sebuah view berisi baris dan kolom seperti pada **table terstruktur**.
- Kolom-kolom pada view bisa berasal dari satu atau lebih tabel.
- View dapat digunakan sebagai *keamanan / security*
 - Sebuah informasi dapat diambil dari query view, bukan dari tabel asli.

Sintaks View

Create

```
CREATE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

Update

```
CREATE OR REPLACE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

Drop

```
DROP VIEW view_name
```

Display

```
SELECT * FROM view_name
```


Contoh View

Create

```
CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName,UnitPrice  
FROM Products  
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

Update

```
CREATE OR REPLACE VIEW [Products Above Average Price]  
AS  
SELECT ProductID, ProductName,UnitPrice  
FROM Products  
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

Drop

```
DROP VIEW [Products Above Average Price]
```

Display

```
SELECT * FROM [Products Above Average Price]
```

Manfaat View

- Dapat merepresentasikan subset data di DB sehingga membatasi pengetahuan pihak luar terhadap DB
- Dapat menggabungkan dan menyederhanakan beberapa tabel yang kompleks
- Dapat bertindak sebagai aggregate table
- Menyembunyikan kompleksitas data
- Menggunakan space yang kecil, bukan meng-copy semua data yang direpresentasikan

Contoh

```
CREATE OR REPLACE VIEW BoatRented_recap
AS
SELECT B.BID, B.BNAME, B.COLOR, COUNT(R.SID) AS RENTED
FROM BOATS B LEFT JOIN RESERVES R
ON B.BID = R.BID
GROUP BY B.BID,B.BNAME, B.COLOR
ORDER BY B.BID ASC;

INSERT INTO BOATS (BID, BNAME, COLOR)
VALUES (105, 'Aquamarine', 'blue');

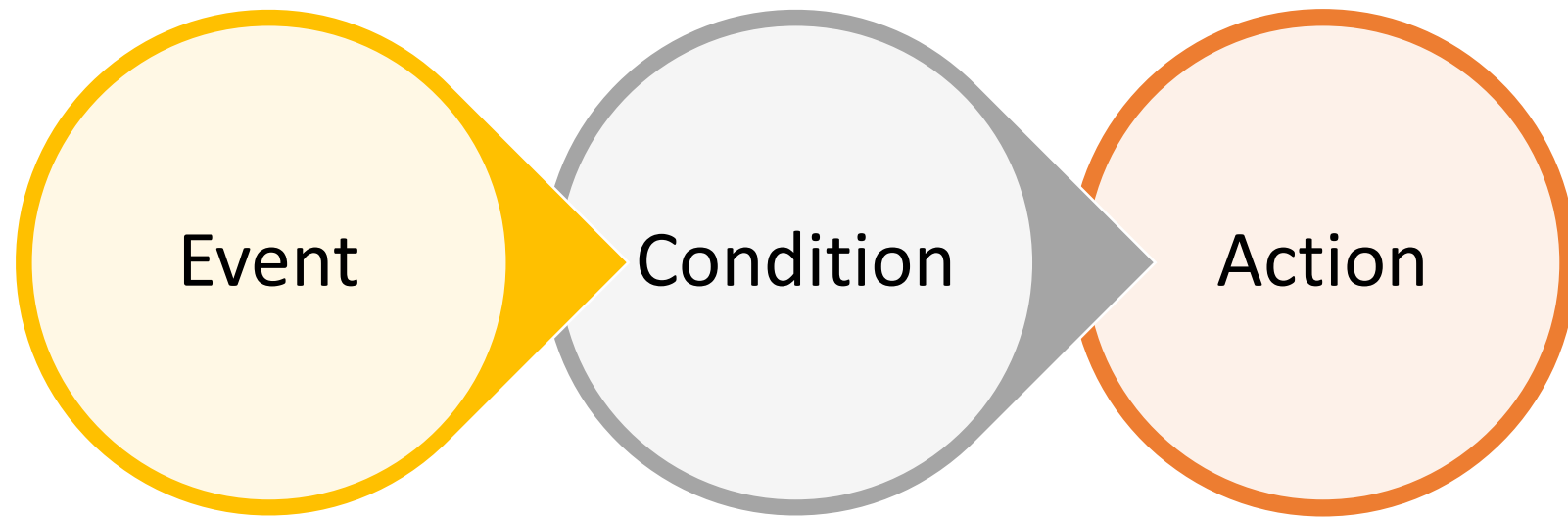
SELECT * FROM BoatRented_recap
```

TRIGGER

Trigger

- Trigger
 - Sebuah konsep, yaitu Teknik untk menentukan tipe tertentu dari *active rules* dalam database.
- Database yang memiliki sejumlah trigger yang saling terhubung dinamakan dengan *active database*.
- Trigger seperti **procedure** yang secara **otomatis dipanggil oleh DBMS** sebagai respon terhadap perubahan tertentu pada database.
- Trigger sama seperti Daemon yang memonitor sebuah database, dan tereksekusi ketika database mengalami perubahan yang cocok dengan spesifikasi *event*.
- Trigger menggunakan ECA model

ECA Rule



ECA Rule

Event

- Menjelaskan suatu kejadian yang mungkin dapat ditanggapi oleh aturan

Condition

- Memeriksa konteks dimana *event* tersebut terjadi

Action

- Menjelaskan tugas yang akan dilaksanakan oleh aturan tersebut jika peristiwa yang relevan telah terjadi dan kondisinya dievaluasi menjadi benar

Event

- **Event** adalah sebuah peristiwa yang terjadi pada satu waktu tertentu.
- Bisa berupa:

- insert, update, access

structure operation

- the message display is sent to an object of type widget

behavior invocation

- abort, commit, begin-transaction

transaction

- an attempt to access some data without appropriate authorization

exception

- the first day of every month

clock

- the temperature reading goes above 30 degrees

External

Condition

- *Condition* (kondisi) tersebut menunjukkan apakah tindakan aturan harus dijalankan.
- Dalam aturan ECA, *condition* umumnya opsional.
- Setelah *event trigger* terjadi, *condition* dapat dievaluasi. Jika *condition* dinilai benar, *action rules* akan dijalankan.
- Jika tidak ada *condition* yang ditentukan, *action* akan dijalankan setelah *event* terjadi.

Pengaturan Waktu Trigger

- *Action* dapat diaplikasikan **before** atau **after event** trigger dijalankan.
- **BEFORE** Trigger
 - Eksekusi *action* dari trigger sebelum *statement trigger*.
 - Menghilangkan proses-proses yang tidak penting dari *statement trigger*.
- **AFTER** Trigger
 - Trigger ini digunakan jika kita menginginkan *statement trigger* terselesaikan sebelum mengeksekusi *action trigger*.

Tipe Trigger

- Sebuah statement SQL dapat mengubah beberapa baris.
 - Terapkan action **for each row** yang terimbas oleh SQL statement.
 - Terapkan action **once** untuk per SQL statement.

Row Triggers

Statement Triggers

Tipe Trigger

Row Triggers

- Row trigger diaktifkan **setiap kali tabel dipengaruhi** oleh statement trigger.
- Jika statement trigger **tidak memengaruhi baris manapun**, maka pemicu baris **tidak** akan dijalankan sama sekali.

Statement Triggers

- Statement trigger dipicu sekali atas nama statement trigger
- Terlepas dari jumlah baris dalam tabel yang dipengaruhi oleh statement trigger (**bahkan jika tidak ada baris yang terpengaruh**)

Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER} {INSERT [OR] | UPDATE [OR] | DELETE}
[OF column name] ON table_name
[FOR EACH ROW]
[WHEN condition]
DECLARE
Declaration statements
BEGIN
Executable statements
EXCEPTION
Exception-handling statements
END trigger_name;
```

ECA pada Trigger

```
CREATE TRIGGER init_count BEFORE INSERT ON Students
```

event

```
DECLARE
```

```
    count INTEGER;
```

```
BEGIN
```

```
    count:=0;
```

action

```
END
```

```
CREATE TRIGGER incr_count AFTER INSERT ON Students
```

```
WHEN (new.age < 18)
```

condition

```
FOR EACH ROW
```

```
BEGIN
```

```
    count:=count + 1;
```

```
END
```

Contoh: Row-Trigger

- Trigger ini digunakan Ketika *action* berdampak pada perubahan record secara individu.
- Menggunakan klausa: FOR EACH ROW
- Dapat menggunakan klausa WHEN

```
CREATE TRIGGER incr_count AFTER INSERT ON Students  
WHEN (new.age < 18)  
FOR EACH ROW  
BEGIN  
        count:=count + 1;  
END
```

Contoh: Statement-Trigger

- Digunakan bila sebuah trigger dieksekusi untuk tiap statement INSERT, UPDATE, atau DELETE (tanpa menghiraukan jumlah record yg terdampak)
- Menggunakan klausa: **FOR EVERY STATEMENT** (tidak perlu ditulis, karena secara default sudah ada)
- Tidak memiliki klausa WHEN.

```
CREATE TRIGGER init_count BEFORE INSERT ON Students
DECLARE
    count INTEGER;
FOR EACH STATEMENT
BEGIN
    count:=0;
END
```


Mutating Table:

Sebuah permasalahan pada Trigger

- Mutating table:
 - tabel yang saat ini sedang dimodifikasi melalui sebuah INSERT, DELETE, ataupun UPDATE.
- Tabel yang sedang *mutating* tidak dapat di-query, karena data yang akan didapatkan tidak konsisten.
- Statement-triggers tidak memiliki problem ini selama trigger tersebut tidak dieksekusi dari DELETE CASCADE.

Contoh Mutating Table

Original
EMP Table

EMP Table	
ENAME	SAL
SMITH	1000
JONES	1000
WARD	1000

SQL Statement That
Fires an **AFTER**
Row Trigger


UPDATE emp
SET sal=sal *1.1;

Mutating
EMP Table

EMP Table	
ENAME	SAL
SMITH	1100
JONES	1000
WARD	1000

AFTER Row
Trigger Fired,
Contains:

SELECT sal
FROM emp
WHERE...


Not allowed because EMP
table is a mutating table

Consider the following trigger:

```
CREATE OR REPLACE TRIGGER emp_count
```

```
AFTER DELETE ON EMP
```

```
FOR EACH ROW
```

```
DECLARE
```

```
    n INTEGER;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO n FROM emp;
```

```
    DBMS_OUTPUT.PUT_LINE(' There are now ' || n ||  
        ' employees.');
```

```
END;
```

If the SQL statement

```
DELETE FROM emp WHERE empno = 7499;
```

is issued, the **following error is returned**:

ORA-04091: table SCOTT.EMP is mutating,
trigger/function may not see it

- Solusi:
- If you delete the line "FOR EACH ROW" from the trigger above, the trigger becomes a statement trigger, the table is not mutating when the trigger fires, and the trigger does output the correct data.

Aplikasi Trigger



Modifikasi data audit

Event logging

Penerapan business rules secara ketat

Menghitung nilai kolom secara otomatis

Penerapan otorisasi keamanan yang kompleks

Memelihara replika table

Menjaga *referential integrity*

Contoh Trigger

Modifikasi Data Audit

```
create table Orders ( Order_no integer,  
status varchar2(10) )
```

```
insert into Orders (Order_no, status) values  
(1, 'process')
```

```
insert into Orders (Order_no, status) values  
(2, 'deliver')
```

```
create table log ( change_date date,  
Order_num integer, old_status varchar2(10),  
new_status varchar2(10) )
```

```
Update Orders SET status='deliver'  
WHERE Order_no = 1
```

```
CREATE OR REPLACE TRIGGER  
log_change_Order_status  
AFTER UPDATE ON Orders  
FOR EACH ROW  
BEGIN  
  
INSERT INTO log (change_date,  
Order_num, old_status, new_status)  
values (SYSDATE, :old.Order_no,  
:old.status, :new.status);  
  
END;
```

Menghitung nilai kolom secara otomatis

- Tabel **Book** memiliki 3 kolom: *columns id, price, dan discount*.
- Jika harga buku lebih dari 200 maka akan mendapat diskon 20%.
- Ketika data buku baru dimasukkan ke dalam Book, kolom discount harus terupdate sesuai dengan harga buku.
- Bagaimana cara memfasilitasi proses ini bila terjadi penambahan dan perubahan record pada table Book?

```
CREATE OR REPLACE TRIGGER book_discount_rule
BEFORE INSERT OR UPDATE ON BOOK
FOR EACH ROW
BEGIN
    if :new.price > 200
    then
        :new.discount := 20;
    else
        :new.discount := 0;
    end if;
END;
```

Penerapan business rules

```
EMPLOYEE(Name, SSN, Salary, DNO, SupervisorSSN, JobCode)
DEPARTMENT(DNO, TotalSalary, ManagerSSN)
STARTING_PAY(JobCode, StartPay)
```

Membatasi semua kenaikan gaji maksimal 50% dari gaji semula.

```
CREATE TRIGGER emp_salary_limit
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
WHEN (new.Salary > 1.5 * old.Salary)
SET new.Salary = 1.5 * old.Salary;
```


Menjaga referential integrity

DEPT		
deptno	deptname	deptloc
11	HR	L-101
12	Sales	B-102
13	Finance	L-203

EMP		
emp_id	emp_name	deptno
345	Habsy	11
346	Syafa	13
347	Zaky	12

```
CREATE OR REPLACE TRIGGER
cascade_updates
AFTER UPDATE OF deptno ON DEPT
FOR EACH ROW
BEGIN
UPDATE EMP
SET EMP.deptno = :new.deptno
WHERE EMP.deptno = :old.deptno
END;
```

SEQUENCE

Sequence

- **SEQUENCE** digunakan untuk membuat **AUTO_NUMBER** pada tabel
- Syntax

```
CREATE SEQUENCE [ IF NOT EXISTS ] sequence_name
    [ AS { SMALLINT | INT | BIGINT } ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ]
    [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ]
    [ CACHE cache ]
    [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

Merujuk Sequence pada Trigger

```
1 CREATE SEQUENCE simple_employees_seq
  AS integer
  INCREMENT BY 1
  MINVALUE 1
  MAXVALUE 9999
  START WITH 1;
```

```
2 create table SIMPLE_EMPLOYEES
( empno integer primary key,
  name varchar(50) not null,
  job varchar(50) );
```

```
3 CREATE OR REPLACE FUNCTION add_new_emp()
  RETURNS TRIGGER
  AS $add_new_emp$
BEGIN
  IF NEW.empno IS NULL
  THEN NEW.empno := NEXTVAL('simple_employees_seq');
  END IF;
  RETURN NEW;
END;
$add_new_emp$ LANGUAGE plpgsql;
```

```
4 CREATE TRIGGER add_new_emp BEFORE INSERT ON simple_employees
  FOR EACH ROW EXECUTE PROCEDURE add_new_emp();
```

```
5 INSERT INTO simple_employees (name, job) VALUES ('Jacob', 'Programmer');

SELECT * FROM simple_employees;
```

SEQUENCE

Hasil

	IDPERIKSA	NID	IDPASIEN	TGLPERIKSA	DIAGNOSA	BIAYA	CATATAN	URUTAN
1	PR01	D003	PS00001	05-11-2014	Demam Berdarah Dengue	150000	Credit Payment	1
2	PR04	D004	PS00004	10-11-2014	Infeksi Mata	50000	PembayaranTunai	4
3	PR05	D005	PS00005	29-11-2014	Influenza	30000	PembayaranTunai	5
4	PR07	D001	PS00001	30-12-2014	Radang Gusi	35000	PembayaranTunai	7
5	PR02	D006	PS00002	08-11-2014	Sakit Gigi	85000	PembayaranTunai	2
6	PR03	D003	PS00003	10-11-2014	Maag Kronis	250000	Credit Payment	3
7	PR06	D005	PS00002	30-12-2014	Influenza	30000	PembayaranTunai	6



	IDPERIKSA	NID	IDPASIEN	TGLPERIKSA	DIAGNOSA	BIAYA	CATATAN	URUTAN
1	PR01	D003	PS00001	05-11-2014	Demam Berdarah Dengue	150000	Credit Payment	1
2	PR04	D004	PS00004	10-11-2014	Infeksi Mata	50000	PembayaranTunai	4
3	PR05	D005	PS00005	29-11-2014	Influenza	30000	PembayaranTunai	5
4	PR07	D001	PS00001	30-12-2014	Radang Gusi	35000	PembayaranTunai	7
5	PR02	D006	PS00002	08-11-2014	Sakit Gigi	85000	PembayaranTunai	2
6	PR03	D003	PS00003	10-11-2014	Maag Kronis	250000	Credit Payment	3
7	PR06	D005	PS00002	30-12-2014	Influenza	30000	PembayaranTunai	6
8	PR08	D005	PS00001	01-01-2015	Influenza	30000	PembayaranTunai	8

INSERT INTO PEMERIKSAAN VALUES
(`'PR08'`,`'D005'`,`'PS00001'`,`'01-01-2015'`,
`'Influenza'`,30000,`'PembayaranTunai'`,
`pemeriksaan_seq.NEXTVAL`);

PROCEDURE

Procedure

- **Procedure** adalah sebuah modul yang melakukan sebuah **action** atau lebih.
- Procedure tidak mengembalikan sebuah nilai.
- Sebuah procedure bisa memiliki nol s.d. banyak parameter.
- Tersusun dari dua bagian utama:
 - **Header**: berisi nama procedure dan daftar parameter
 - **Body**: semua yang tertulis setelah kata kunci IS atau AS
- **REPLACE** bersifat optional
 - Jika REPLACE tidak digunakan, saat ingin mengubah kode dalam procedure maka procedure harus di-drop terlebih dahulu kemudian di-create ulang.

Syntax

```
CREATE OR REPLACE PROCEDURE name [(parameter[, parameter, ...])]  
AS  
    [local declarations]  
BEGIN  
    executable statements  
[EXCEPTION  
    exception handlers]  
END [name];
```

Syntax untuk mengeksekusi:

```
EXECUTE Procedure_name
```


Contoh (Procedure)

```
CREATE OR REPLACE PROCEDURE Discount
AS
```

```
    CURSOR c_group_discount
    IS
        SELECT distinct s.course_no, c.description
        FROM section s, enrollment e, course c
        WHERE s.section_id = e.section_id
              AND c.course_no = s.course_no
        GROUP BY s.course_no, c.description,
              e.section_id, s.section_id
        HAVING COUNT(*) >=8;
```

Cursor digunakan dalam *looping*.

```
BEGIN
FOR r_group_discount IN c_group_discount
    LOOP
        UPDATE course
            SET cost = cost * .95
        WHERE course_no = r_group_discount.course_no;
        DBMS_OUTPUT.PUT_LINE
            ('A 5% discount has been given to'||
            r_group_discount.course_no||' '||
            r_group_discount.description
            );
    END LOOP;
END;
```

EXECUTE Discount

FUNCTION

Function

- Function mirip dengan procedure, namun function mengembalikan sebuah nilai.
- Function dapat menerima nol, satu atau lebih parameter namun harus **memiliki klausa return** pada bagian executable dari function
- Tipe data dari return value harus dideklarasikan di header function
- Tidak bersifat stand-alone executable: harus digunakan pada suatu konteks,
- Output dari sebuah function harus:
 - Dimasukkan ke dalam variabel, atau
 - Digunakan pada pernyataan SELECT

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name  
    (parameter list)  
    RETURN datatype  
  
IS  
  
BEGIN  
    <body>  
    RETURN (return_value);  
  
END;
```

Contoh (Function)

```
CREATE OR REPLACE FUNCTION show_description (i_course_no number)
RETURN varchar2
AS
    v_description varchar2(50);
BEGIN
    SELECT description
    INTO v_description
    FROM course
    WHERE course_no = i_course_no;
    RETURN v_description;
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        RETURN('The Course is not in the database');
    WHEN OTHERS
    THEN
        RETURN('Error in running show_description');
END;
```

Cara memanggil Function

Anonymous block

```
SET SERVEROUTPUT ON
DECLARE
    v_description VARCHAR2(50);
BEGIN
    v_description := show_description(&sv_cnumber);
    DBMS_OUTPUT.PUT_LINE(v_description);
END;
```

SQL statement

```
SELECT course_no, show_description(course_no)
FROM course;
```

Practice

```
Select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

Latihan

1. Buatlah ***sequence*** untuk mengisi ID secara otomatis ketika terjadi penambahan data baru
2. Buatlah ***view*** untuk menampilkan customers dengan gaji dan usia tertentu.
3. Buatlah ***trigger*** yang digunakan untuk menampilkan data perbedaan gaji customers jika dilakukan perubahan pada basis data