UCC

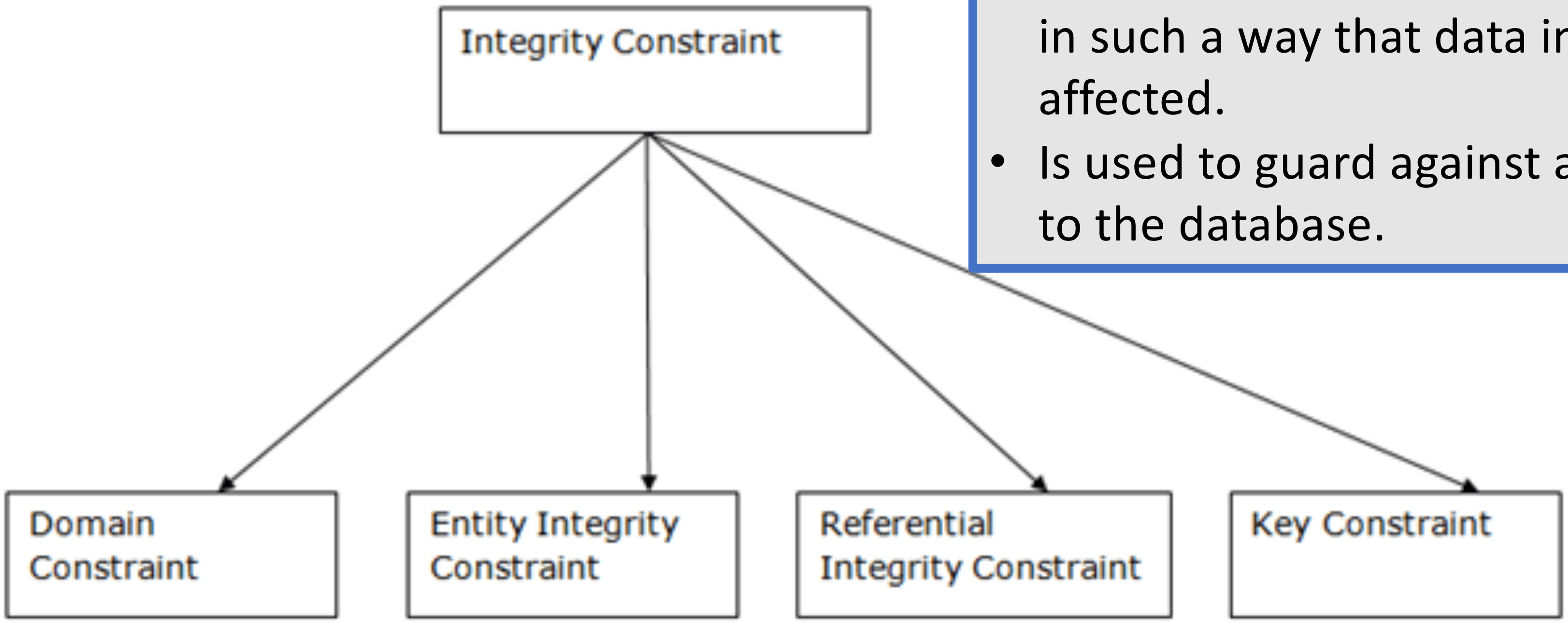
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Information Storage and Management I

Triggers

Dr. Alejandro Arbelaez

- Ensure that the data **insertion, updating, and other processes** have to be performed in such a way that data integrity is not affected.
- Is used to guard against accidental damage to the database.



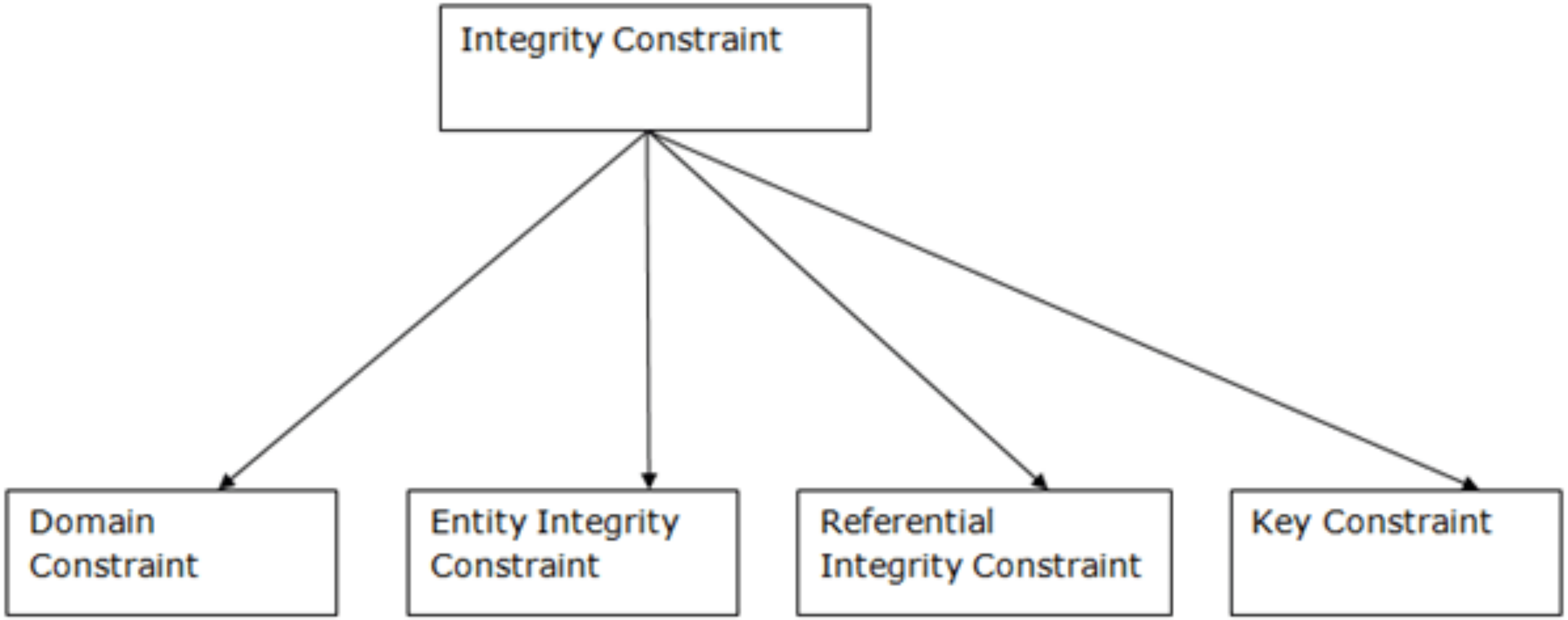
Integrity Constraint

Domain Constraint

Entity Integrity Constraint

Referential Integrity Constraint

Key Constraint



Integrity Constraint

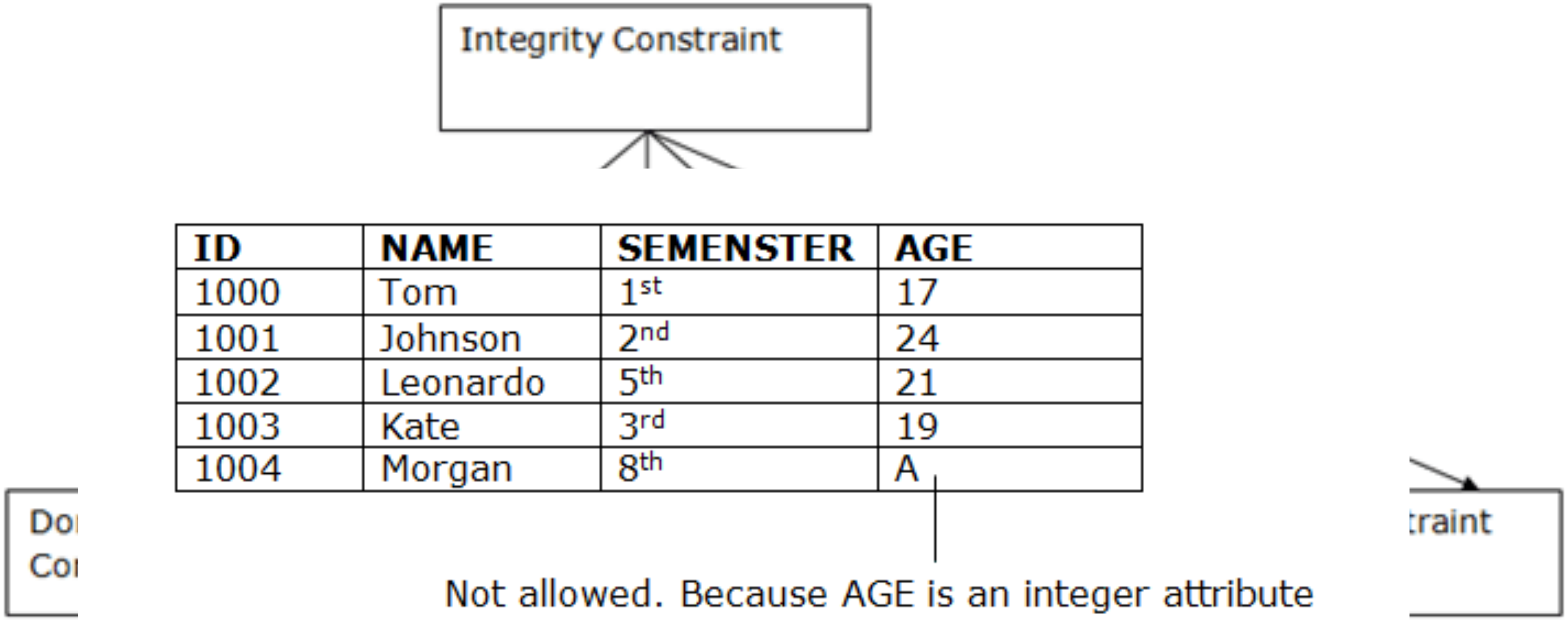
Domain Constraint

Entity Integrity Constraint

Referential Integrity Constraint

Key Constraint

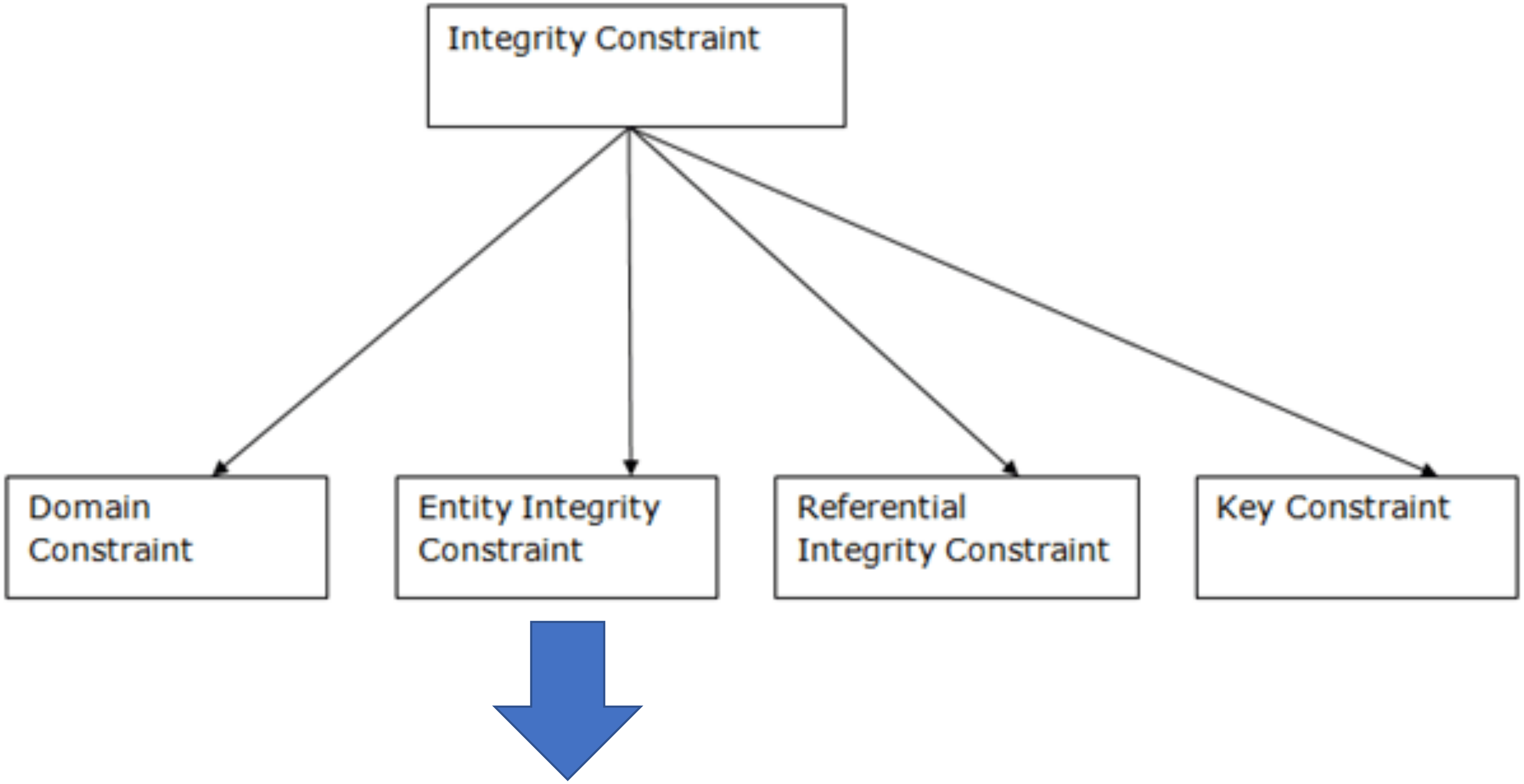
- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc.
- The value of the attribute must be available in the corresponding domain.


Integrity Constraint
Do
Co
traint

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc.
- The value of the attribute must be available in the corresponding domain.



Integrity Constraint

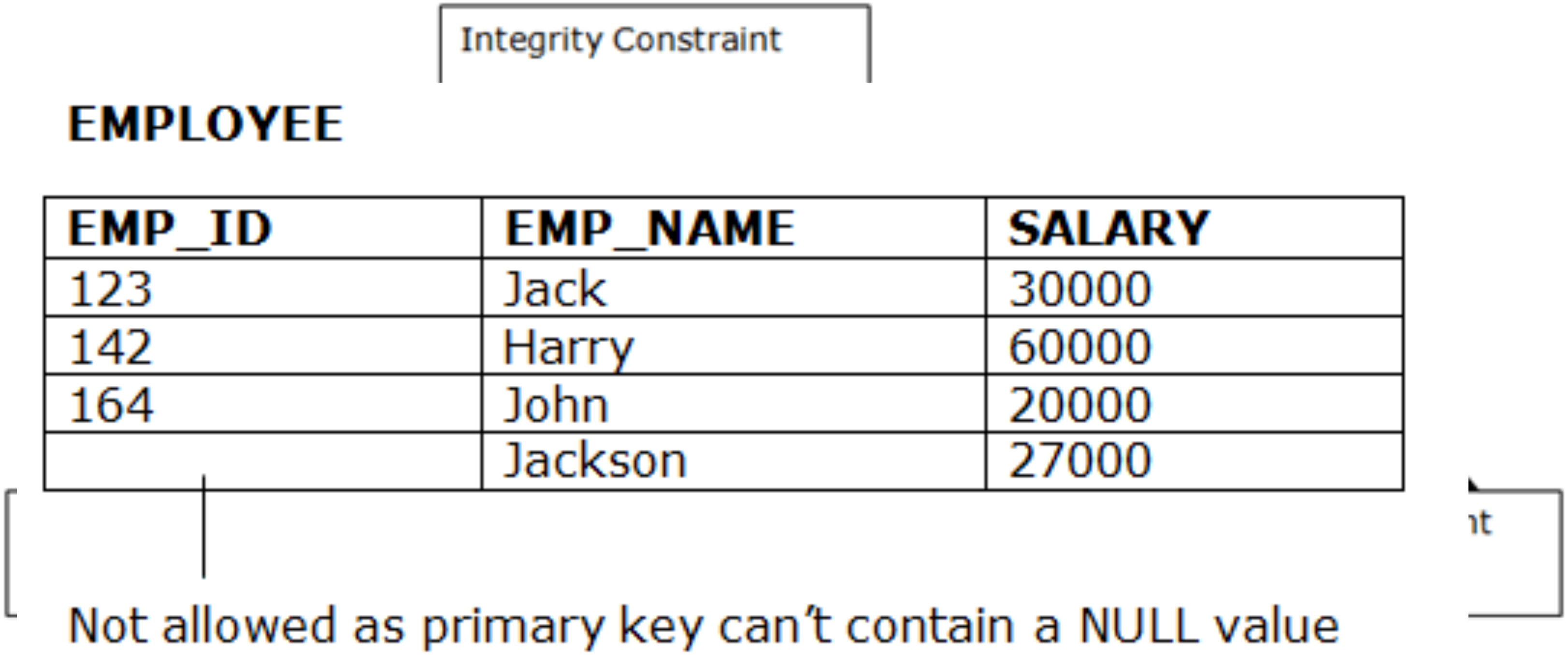
Domain Constraint

Entity Integrity Constraint

Referential Integrity Constraint

Key Constraint

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.



Integrity Constraint

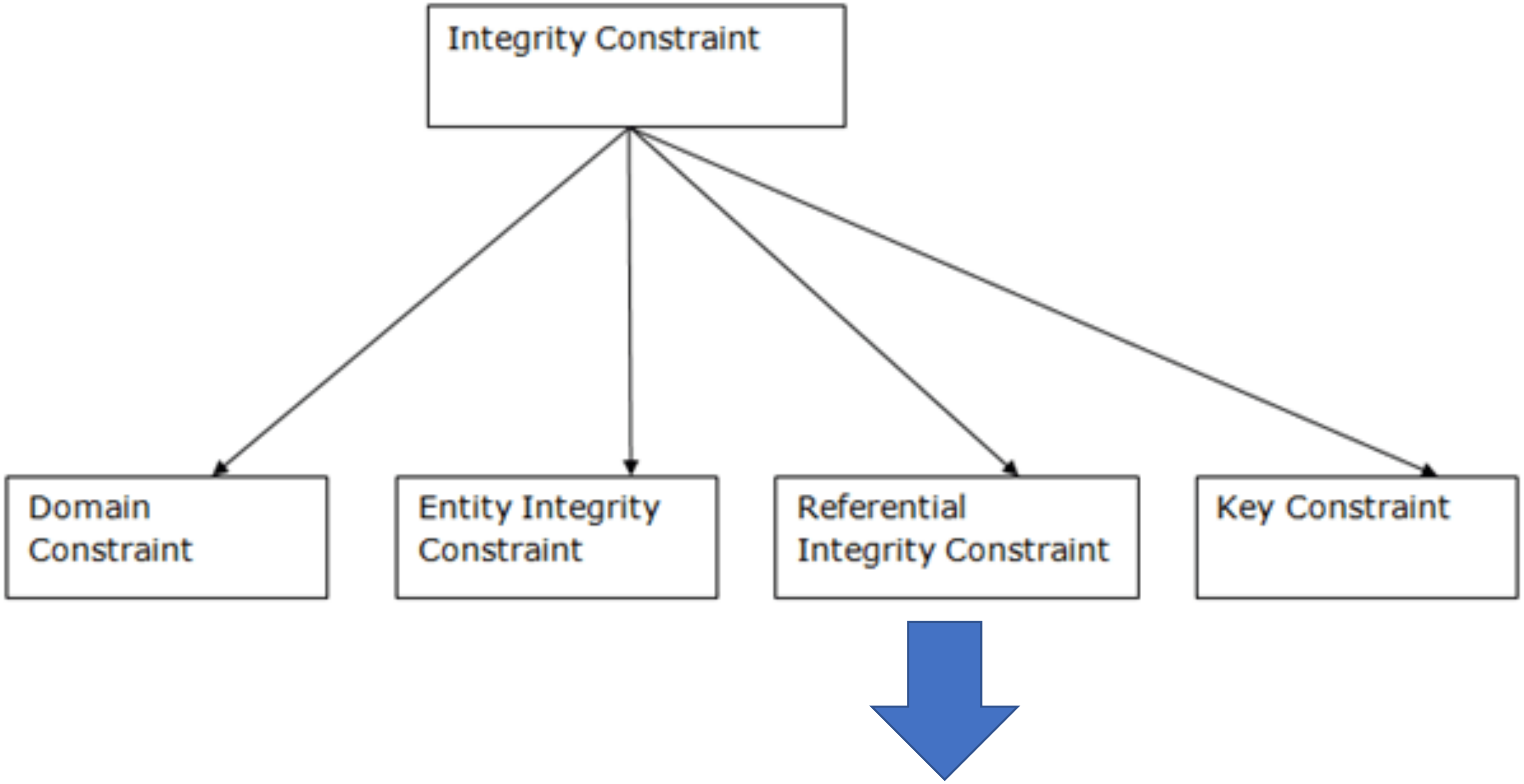
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

nt

Not allowed as primary key can't contain a NULL value

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.



Integrity Constraint

Domain Constraint

Entity Integrity Constraint

Referential Integrity Constraint

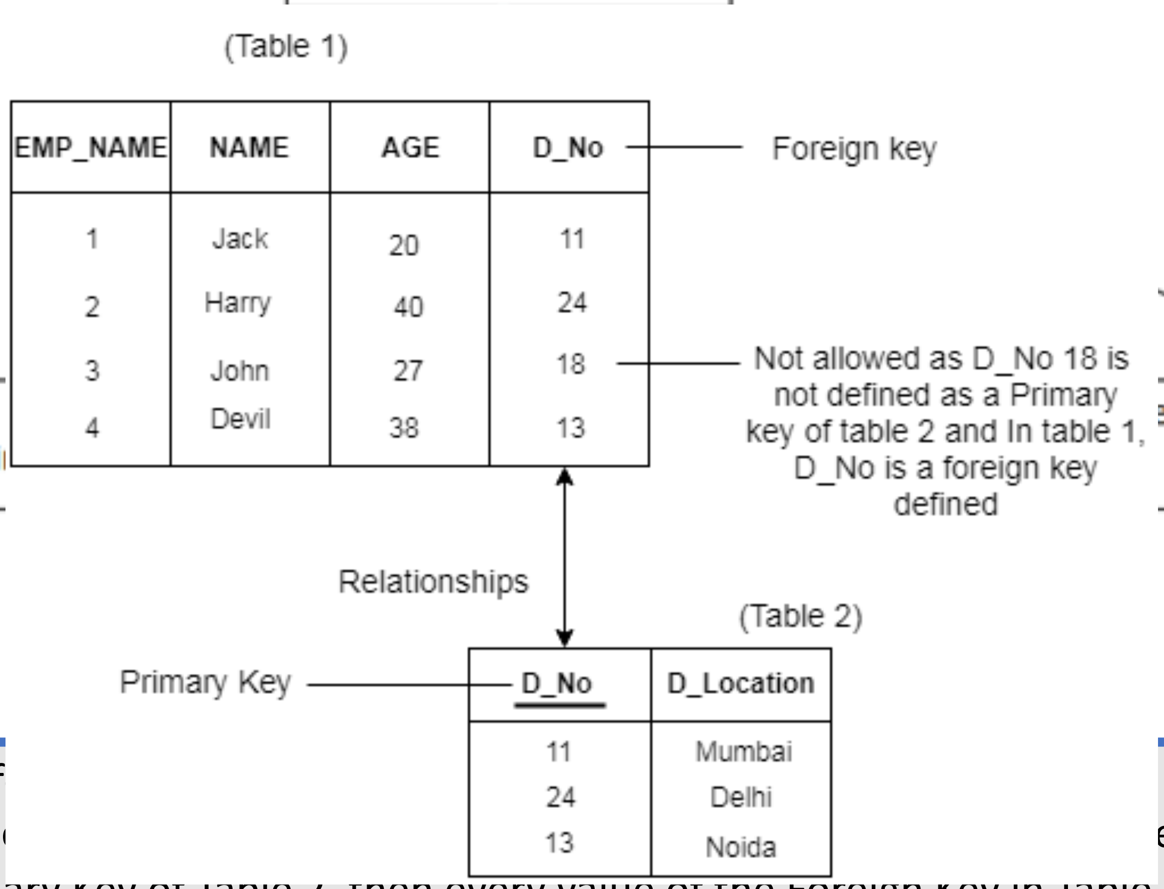
Key Constraint

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Integrity Constraint

Domain Constrai

ey Constraint



(Table 1)

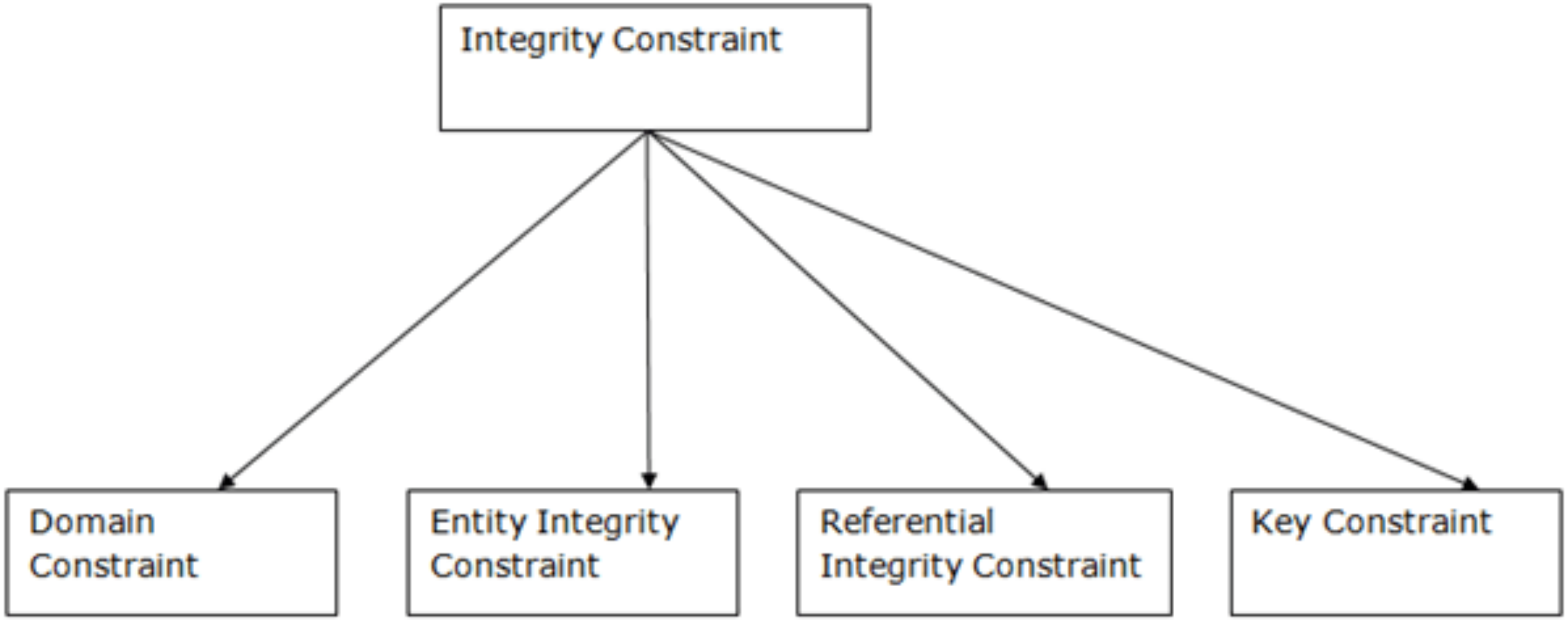
EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
Primary Key

(Table 2)

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

- A ref
- In th ers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



Integrity Constraint

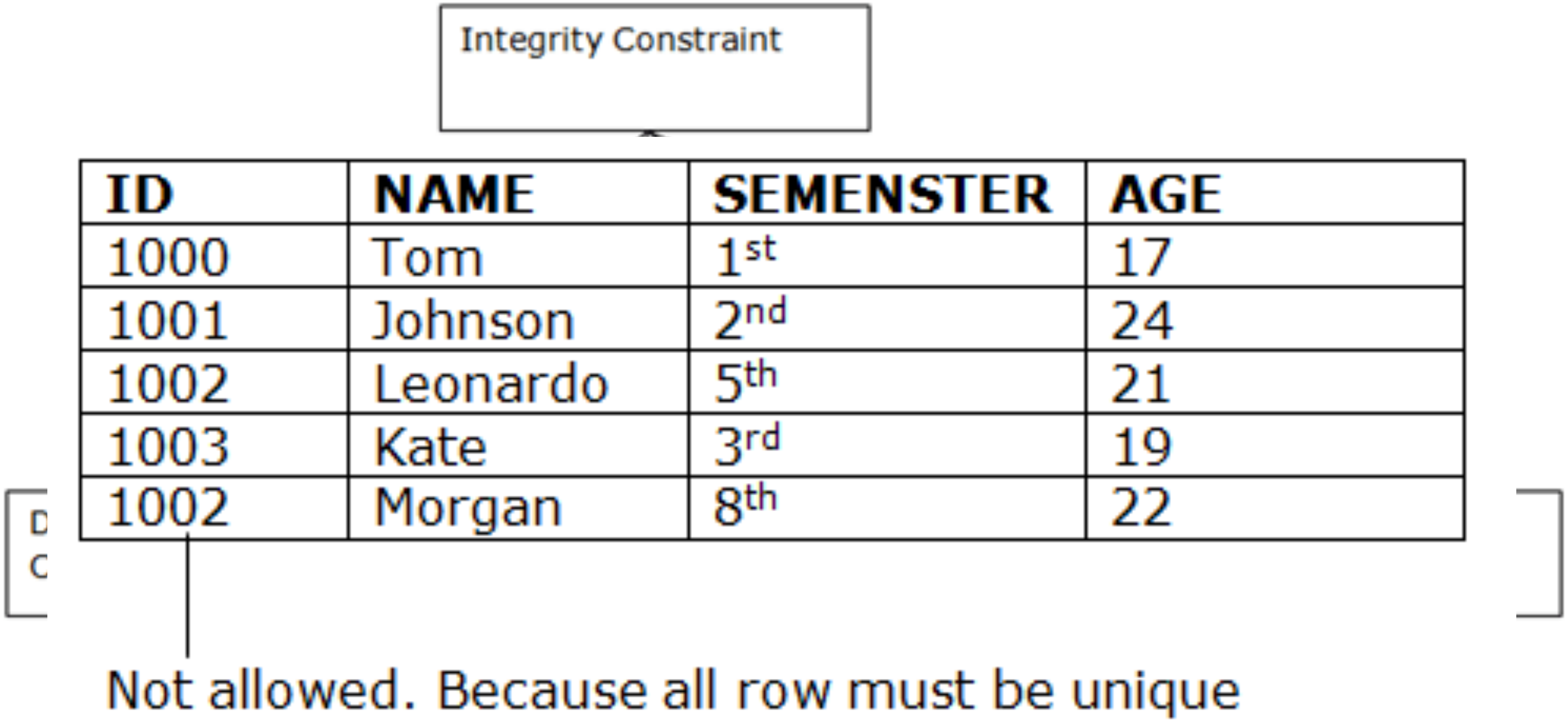
Domain Constraint

Entity Integrity Constraint

Referential Integrity Constraint

Key Constraint

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



Integrity Constraint

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

An Example via CHECK Clause

```
CREATE TABLE  Students (
        sid  INT,
        sname  VARCHAR(10),
        rating  INT,
        age  INT,
        PRIMARY KEY  (sid),
        CONSTRAINT checkRating
        CHECK  (rating >= 1 AND rating <= 10 )
)
```

```
  INSERT INTO Students VALUES(1, 'Jones', 9, 19);
  INSERT INTO Students VALUES(2, 'Smith', 7, 19);
X INSERT INTO Students VALUES(2, 'Peter', 19, 19);
```

ASSERTION Example
Constraints Over Multiple Relations

```
CREATE ASSERTION smallSchool
CHECK  (
        (SELECT COUNT (S.sid) FROM Stu S) +
        (SELECT COUNT (P.pid) FROM Prof P) < 500
)
```

ASSERTION Example
The KEY Constraint

```
CREATE ASSERTION Key
CHECK  (
        (SELECT COUNT (DISTINCT sid) FROM Stu) =
        (SELECT COUNT (*) FROM Stu)
);
```

- Note: ASSERTION is in standard SQL but not implemented
- Unfortunately, MySQL does not support ASSERTIONS, but we can uses triggers to implement this functionality

Triggers

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs
- A trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated

When **event** occurs, check **condition**; if true do **action**

Advantages

- To move application logic and business rules into database.
- Allows more functionality for DBAs to establish vital constraints/rules of applications.
- Rules managed in some central “place”.
- Rules automatically enforced by DBMS, no matter which applications later come on-line.

ASSERTION Example
Constraints Over Multiple Relations

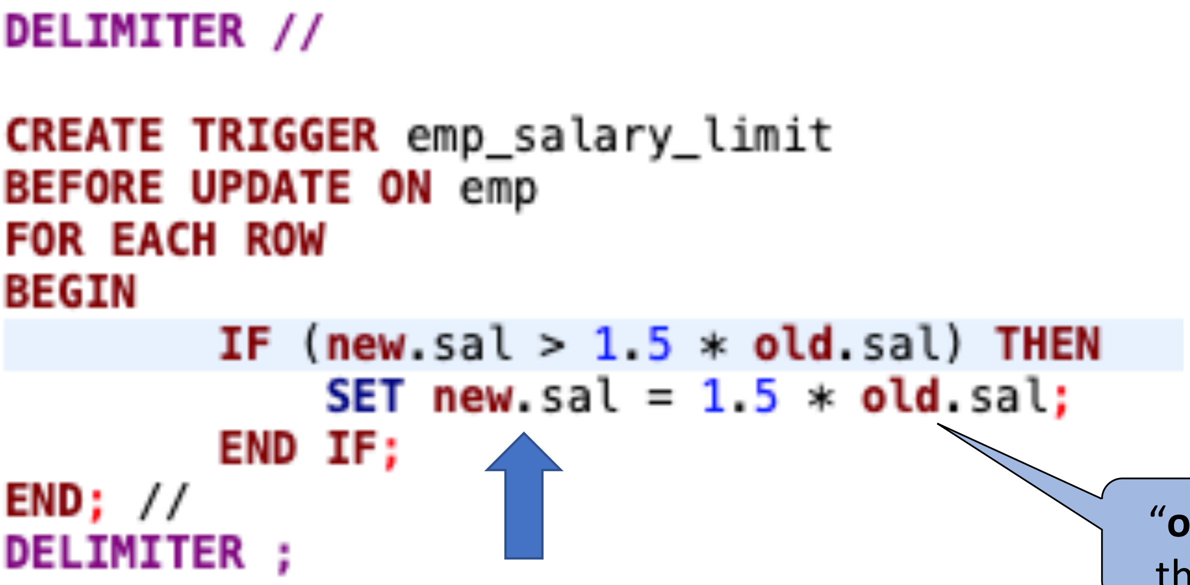
- Consider a very small school: the count of students and professors should be less than 500
- The following is a poor integrity test as it is associated with one relation (the Students table could be empty and thus the integrity rule is never checked!
- Disassociate from the Students table

```
CREATE TABLE  Students (
            sid  INTEGER,
            sname  VARCHAR(10),
            rating  INT,
            age  INT,
            PRIMARY KEY  (sid),
            CHECK  (
            (SELECT COUNT (S.sid) FROM Students S) +
            (SELECT COUNT (P.pid) FROM Profesor P) < 500 )
)
```

The Event-Condition-Action Model

- Actions may apply before or after the triggering event is executed
- An SQL statement may change several rows
  - Apply action once per SQL statement
  - Apply action for each row changed by SQL statement

Limit all salary increases to 50%



```
DELIMITER //

CREATE TRIGGER emp_salary_limit
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
        IF (new.sal > 1.5 * old.sal) THEN
            SET new.sal = 1.5 * old.sal;
        END IF;
END; //
DELIMITER ;
```

“**new**” refers to the new tuple.

“**old**” refers to the old tuple.

OLD and NEW

You can refer to columns in the subject table (the table associated with the trigger) by using the aliases OLD and NEW.

OLD.col_name refers to a column of an existing row before it is updated or deleted.

NEW.col_name refers to the column of a new row to be inserted or an existing row after it is updated.

Event-Condition-Action (ECA)

- Event occurs in databases
  - e.g. addition of a new row, deletion of a row
- Conditions are checked
  - e.g. Is batch complete? Has student passed?
- Actions are executed if conditions are satisfied
  - e.g. send batch to supplier, congratulate student

Extending Information Processing Capabilities of DBMS using Triggers

- Processing of database content, performed by the DBMS engine itself, not by the application client
  - execution of the trigger (Event)
- Initiated by certain specified condition, depending on the type of the trigger
  - firing of the trigger (Condition)
- All data actions performed by the trigger execute within the same transaction in which the trigger fires, but in a separate session (Action)
  - Triggers are checked for different privileges as necessary for the processed data
  - Cannot contain transaction control statements (COMMIT, SAVEPOINT, ROLLBACK not allowed)

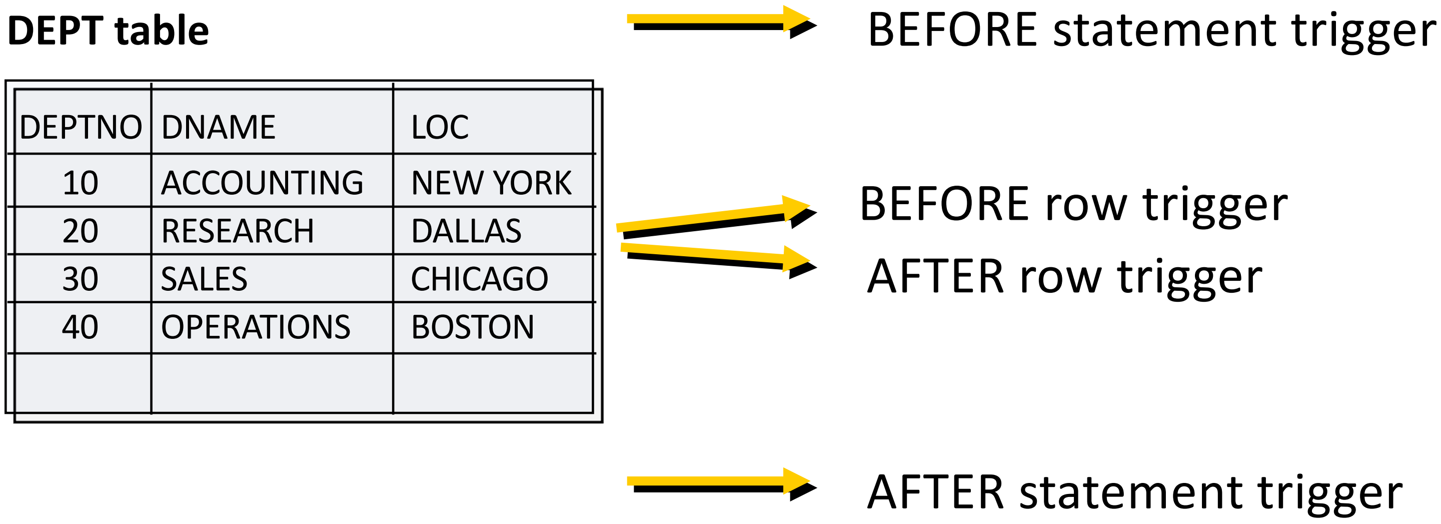
Database Triggers in SQL

- Not specified in SQL-92, but standardized in SQL3 (SQL1999)
- Available in most enterprise DBMSs (Oracle, IBM DB2, MS SQL server) and some public domain DBMSs (Postgres)
  - but not present in smaller desktop (Oracle Lite) and public domain DBMS (MySQL)
- Some vendor DBMS permit native extensions to SQL for specifying the triggers
  - e.g. PL/SQL in Oracle, Transact SQL in MS SQL Server
- Some DBMS also allow use of general purpose programming language instead of SQL
  - e.g. C/C++ in Poet, Java in Oracle, C#/VB in SQL Server
- Some DBMS extend the triggers beyond tables
  - for example also to views as in Oracle

Types of SQL Triggers

- How many times should the trigger body execute when the triggering event takes place?
  - Per statement: the trigger body executes once for the triggering event. This is the default.
  - For each row: the trigger body executes once for each row affected by the triggering event.
- When the trigger can be fired
  - Relative to the execution of an SQL DML statement (before or after or instead of it)
  - Exactly in a situation depending on specific system resources (e.g. signal from the system clock, expiring timer, exhausting memory)

Firing Sequence of Database Triggers on a Single Row



DEPT table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

BEFORE statement trigger

BEFORE row trigger

AFTER row trigger

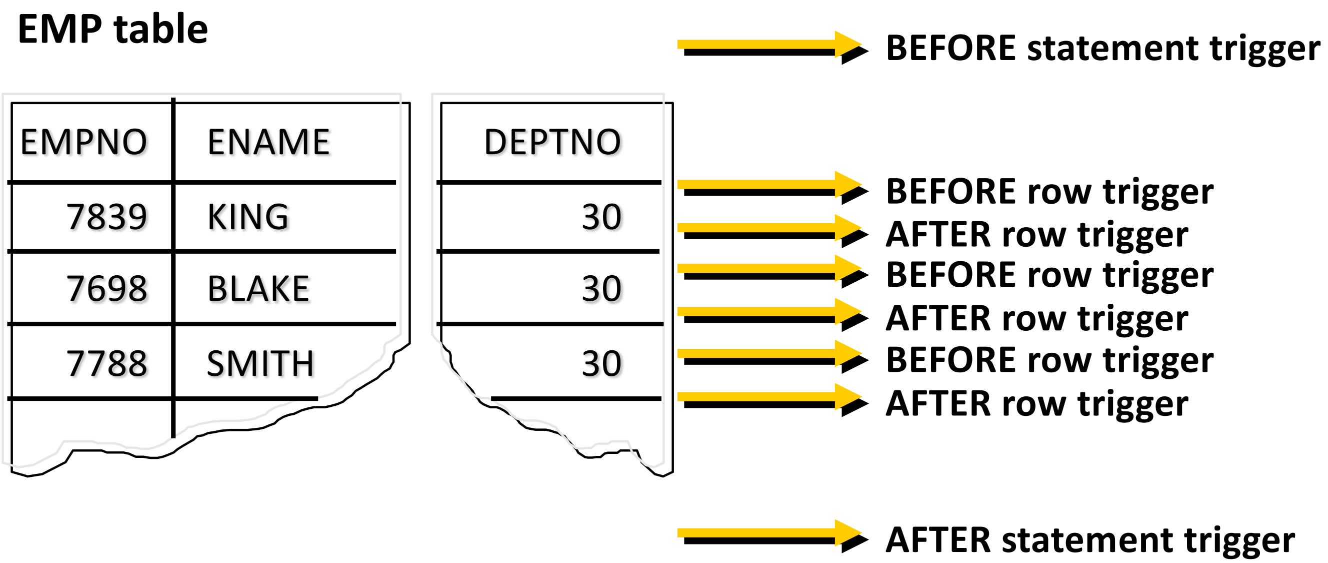
AFTER statement trigger

The Company Database

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

1. Limit all salary increases to 50%.
2. Enforce policy that salaries may never decrease.
3. Maintain `TotalSalary` in `DEPARTMENT` relation as employees and their salaries change.
4. Inform a supervisor whenever a supervisee's salary becomes larger than the supervisor's.
5. All new hires for a given job code get the same starting salary, which is available in the `STARTING_PAY` table.

Firing Sequence of Database Triggers on Multiple Rows



EMP table

EMPNO	ENAME	DEPTNO
7839	KING	30
7698	BLAKE	30
7788	SMITH	30

BEFORE statement trigger

BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger

AFTER statement trigger

Statement and Row Triggers

Example 1: Monitoring Statement Events

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

Execute only once even if multiple rows affected

Example 2: Monitoring Row Events

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```

Execute for each row of the table affected by the event

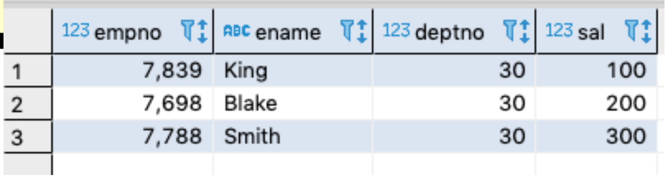
Statement and Row Triggers

Example 1: Monitoring Statement Events

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

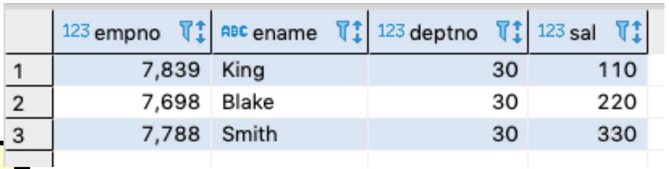
Execute only once even if multiple rows affected

Before Update



	empno	ename	deptno	sal
1	7,839	King	30	100
2	7,698	Blake	30	200
3	7,788	Smith	30	300

After Update



	empno	ename	deptno	sal
1	7,839	King	30	110
2	7,698	Blake	30	220
3	7,788	Smith	30	330

Example 2: Monitoring Row Events

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```

Execute for each row of the table affected by the event

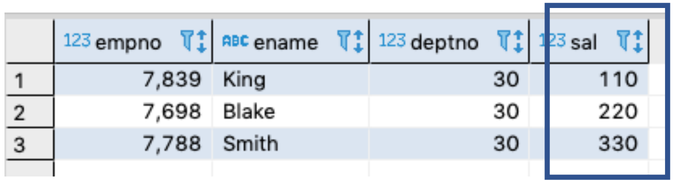
Statement and Row Triggers

Example 1: Monitoring Statement Events

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

Execute only once even if multiple rows affected

Update



	123 empno	ABC ename	123 deptno	123 sal
1	7,839	King	30	110
2	7,698	Blake	30	220
3	7,788	Smith	30	330

Not allowing more than 50%

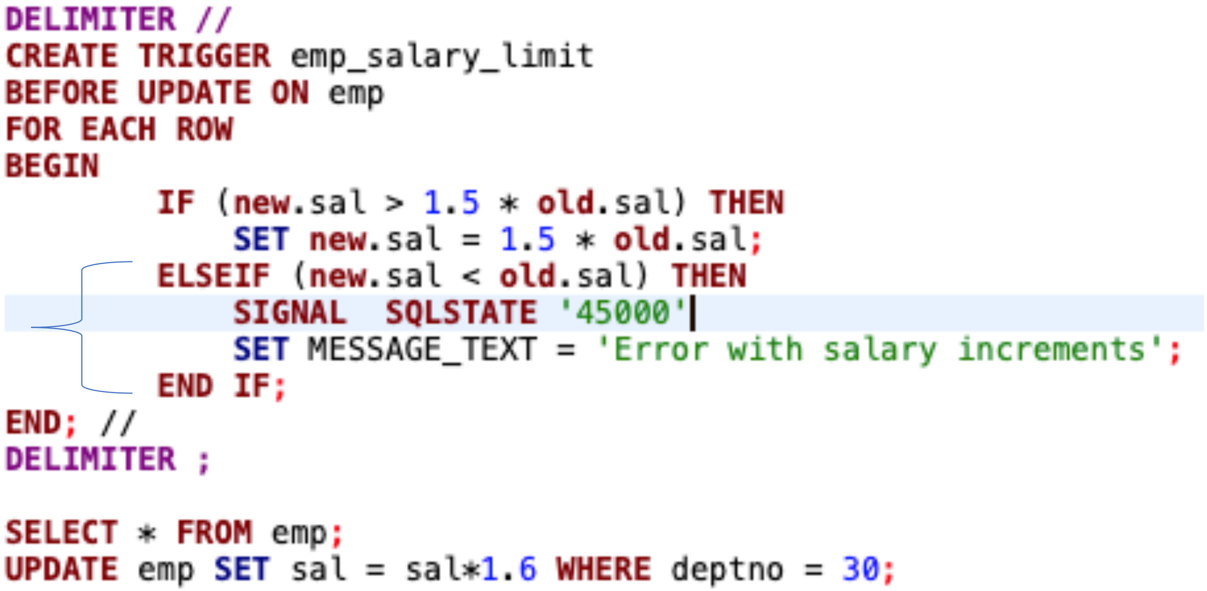
Example 2: Monitoring Row Events

```
SQL> UPDATE emp
   2   SET sal = sal * 2
   3 WHERE deptno = 30;
```

Execute for each row of the table affected by the event

Salaries never decrease

Define error
use '45000', which means “unhandled user-defined exception.”



```sql
DELIMITER //
CREATE TRIGGER emp_salary_limit
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
        IF (new.sal > 1.5 * old.sal) THEN
            SET new.sal = 1.5 * old.sal;
        ELSEIF (new.sal < old.sal) THEN
            SIGNAL  SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error with salary increments';
        END IF;
END; //
DELIMITER ;

SELECT * FROM emp;
UPDATE emp SET sal = sal*1.6 WHERE deptno = 30;
```

Syntax for creating triggers in SQL

- Trigger name - unique within one database schema
- Timing - depends on the order of controlled events (before or after or instead of)
- Triggering event - event which fires the trigger (E)
- Filtering condition - checked when the triggering event occurs (C)
- Target - table (or view) against which the trigger is fired; they should be both created within the same schema
- Trigger Parameters - parameters used to denote the record columns; preceded by colon
  - new, old for new and old versions of the values respectively
- Trigger action - SQL statements, executed when the trigger fires; surrounded by Begin ... End (A)

Syntax for Creating Statement Triggers

The trigger body consists of SQL statements will be executed **only once** according to the prescribed timing, when the event1 (event2, event3) occurs against the monitoring table in question

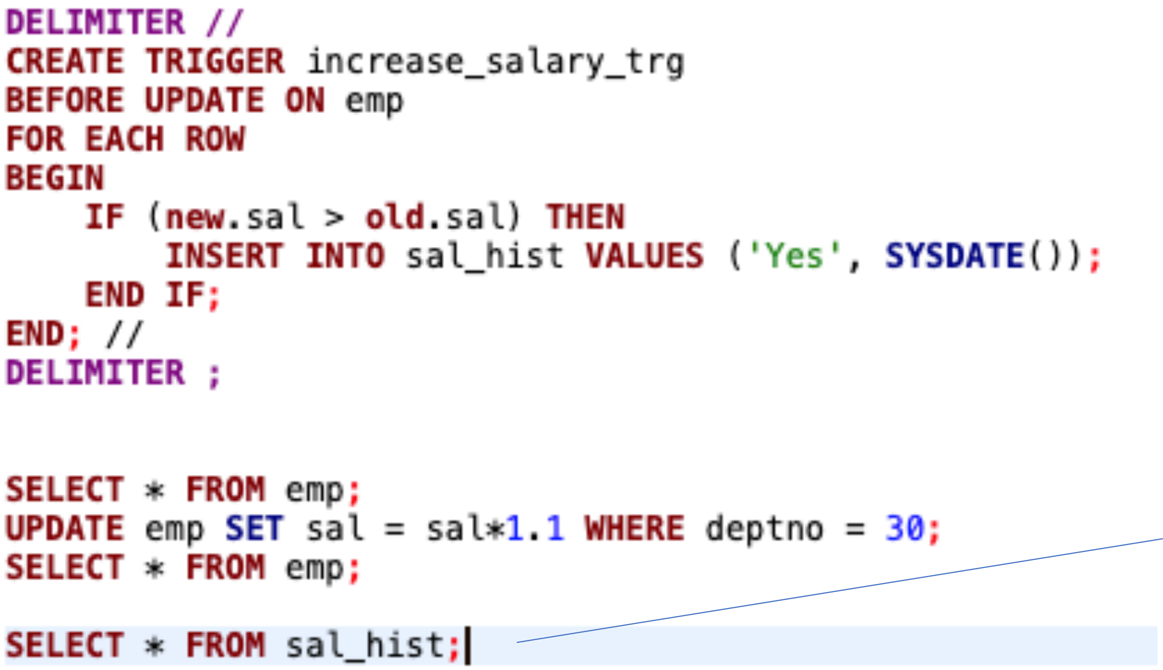
```
CREATE [OR REPLACE] TRIGGER trigger_name
    timing event1 [OR event2 OR event3]
    ON table_name
BEGIN
  SQL statements;
END;
```

Registering Operations

```
CREATE TRIGGER increase_salary_trg
       BEFORE UPDATE ON emp
       FOR EACH ROW
BEGIN
  IF (new.sal > old.sal) THEN
     INSERT INTO sal_hist(increased, changedOn)
       VALUES ('YES', SYSDATE());
  END IF;
END;
```

Trigger name:	`increase_salary_trg`
Timing:	`BEFORE` executing the statement
Triggering event:	UPDATE of `sal` column
Target:	`emp` table
Trigger action:	INSERT values INTO `sal_hist table`

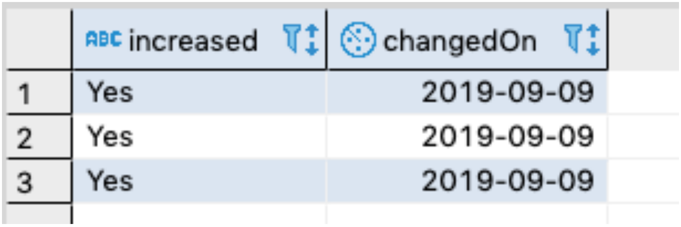
Registering Operations



```
DELIMITER //
CREATE TRIGGER increase_salary_trg
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
    IF (new.sal > old.sal) THEN
        INSERT INTO sal_hist VALUES ('Yes', SYSDATE());
    END IF;
END; //
DELIMITER ;


SELECT * FROM emp;
UPDATE emp SET sal = sal*1.1 WHERE deptno = 30;
SELECT * FROM emp;

SELECT * FROM sal_hist;
```



	increased	changedOn
1	Yes	2019-09-09
2	Yes	2019-09-09
3	Yes	2019-09-09

Syntax for Creating Row Triggers

The trigger body consisting of SQL statements will be executed once for each row affected by event1 (event2, event3) in the table named table_name subject to the additional condition

```
CREATE [OR REPLACE] TRIGGER trigger_name
       timing event1 [OR event2 OR event3]
       ON table_name
       FOR EACH ROW
       [IF condition]
BEGIN
  SQL statements;
END
```

Calculating Derived Columns

```
SQL>CREATE OR REPLACE TRIGGER derive_commission_trg
  2 BEFORE UPDATE OF sal ON emp
  3 FOR EACH ROW
  4 IF (new.job = 'SALESMAN')
  5 BEGIN
  6   new.comm := old.comm * (new.sal/old.sal);
  7 END;
  8 /
```

Trigger name:	derive_commission_trg
Timing:	BEFORE executing the statement
Triggering event:	UPDATE of sal column
Filtering condition:	job = ‘SALESMAN’
Target:	emp table
Trigger parameters:	old, new
Trigger action:	calculate the new commission to be updated

Calculating Derived Columns

```
SQL>CREATE OR REPLACE TRIGGER derive_commission_trg
  2 BEFORE UPDATE OF sal ON emp
  3 FOR EACH ROW
  4 IF( new.job = 'SALESMAN')
  5 BEGIN
  6   new.comm := old.comm * (new.sal/old.sal);
  7 END;
  8 /
```

Note: no (colon :) before new in WHEN

Trigger name:	derive_commission_trg
Timing:	BEFORE executing the statement
Triggering event:	UPDATE of sal column
Filtering condition:	job = ‘SALESMAN’
Target:	emp table
Trigger parameters:	old, new
Trigger action:	calculate the new commission to be updated

Controlling Triggers using SQL

- Disable or Re-enable a database

```
ALTER TRIGGER trigger_name  DISABLE | ENABLE
```

- Disable or Re-anble all triggers for a table

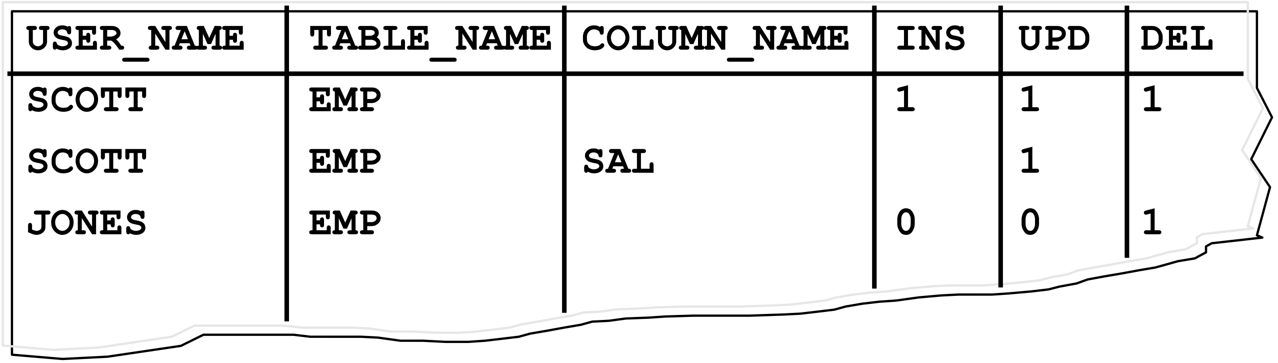
```
ALTER TABLE table_name   DISABLE | ENABLE  ALL TRIGGERS
```

```
DROP TRIGGER trigger_name
```

Using Database Triggers for Information Processing

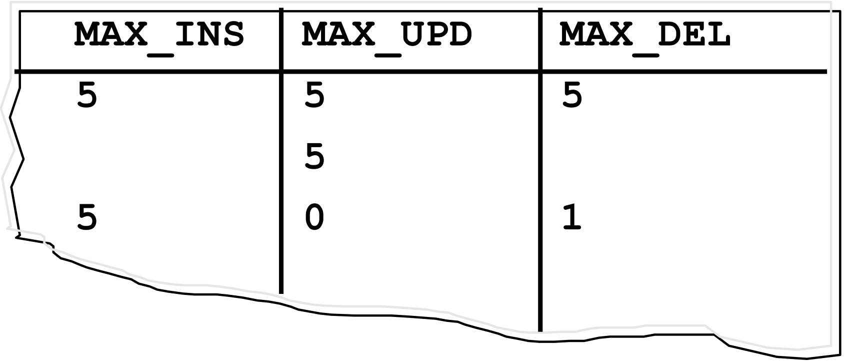
- Auditing Table Operations
  - Each time a table is accessed auditing information is recorded against it
- Tracking Record Value Changes
  - Each time a record value is changed the previous value is recorded
- Protecting Database Referential Integrity: if foreign key points to changing records
  - Referential integrity must be maintained
- Maintenance of Semantic Integrity
  - e.g., when the factory of items in the trolley should correspond to the current session selection
- Storing Derived Data
  - e.g., the number of items in the trolley should correspond to the current session selection
- Security Access Control
  - e.g., changing user privileges when accessing sensitive infromation

Auditing Table Operations



USER_NAME	TABLE_NAME	COLUMN_NAME	INS	UPD	DEL
SCOTT	EMP		1	1	1
SCOTT	EMP	SAL		1	
JONES	EMP		0	0	1

… continuation



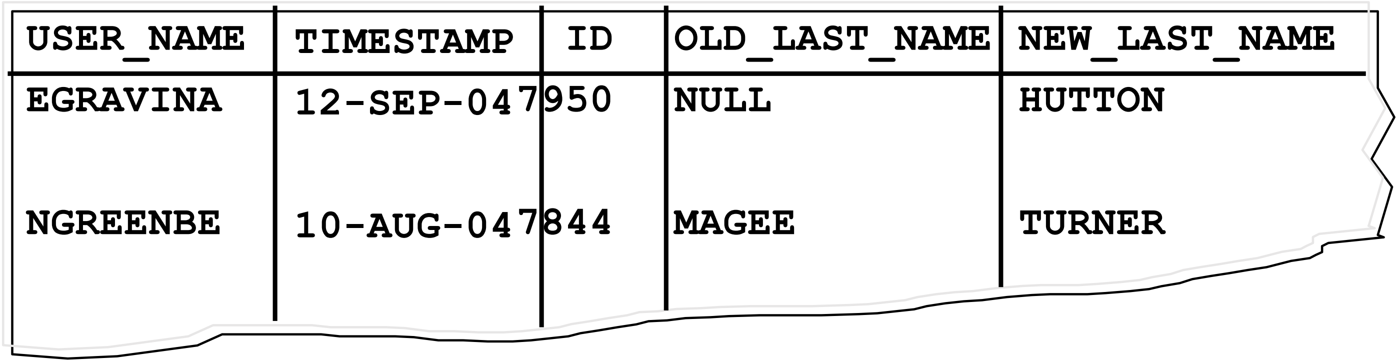
MAX_INS	MAX_UPD	MAX_DEL
5	5	5
	5	
5	0	1

Example: Counting Statement Execution

```
SQL>CREATE OR REPLACE TRIGGER audit_emp
  2 AFTER DELETE ON emp
  3 FOR EACH ROW
  4 BEGIN
  5      UPDATE audit_table SET del = del + 1
  6      WHERE user_name = USER
  7      AND table_name = 'EMP’;
  7 END;
  8 /
```

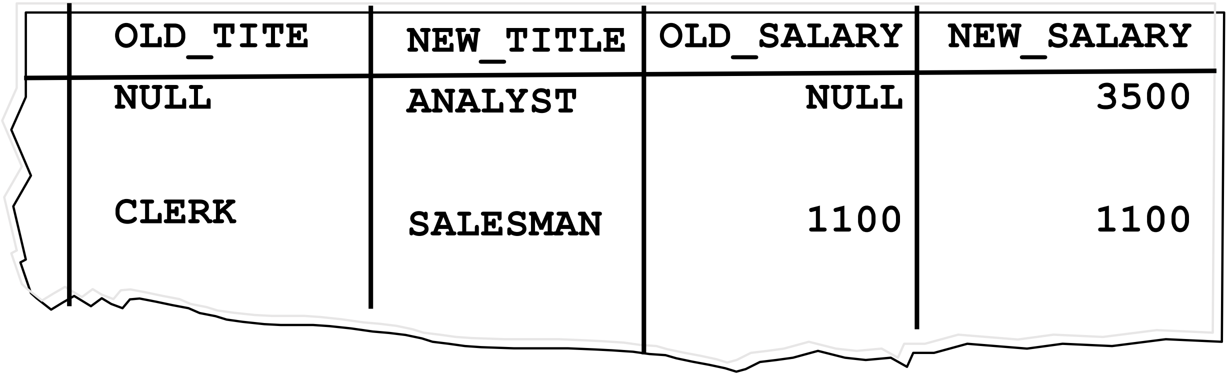
Whenever an employee record is deleted from the database, the counter in an audit table registering the number of deleted rows for the current user in system variable USER is incremented.

Example: Tracing Record Value Changes



USER_NAME	TIMESTAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME
EGRAVINA	12-SEP-04	7950	NULL	HUTTON
NGREENBE	10-AUG-04	7844	MAGEE	TURNER

… continuation



	OLD_TITE	NEW_TITLE	OLD_SALARY	NEW_SALARY
	NULL	ANALYST	NULL	3500
	CLERK	SALESMAN	1100	1100

Example: Protecting Referential Integrity

```
SQL>CREATE OR REPLACE TRIGGER cascade_updates
  2 AFTER UPDATE OF deptno ON dept
  3 FOR EACH ROW
  4 BEGIN
  5   UPDATE emp
  6   SET    emp.deptno = new.deptno
  7   WHERE  emp.deptno = old.deptno;
  8 END
  9 /
```

Whenever the department number changes, all employee records for this department will automatically be changed as well, so that the employees will continue to work for the same department.

Rules for Good Practice

- Rule 1: Do not change data in the primary key, foreign key, or unique key columns of any table
- Rule 2: Do not update records in the same table you read during the same transaction
- Rule 3: Do not aggregate over the same table you are updating
- Rule 4: Do not read data from a table which is updated during the same transaction
- Rule 5: Do not use SQL DCL (Data Control Language) statements in triggers

QUESTIONS